COIN-078B
Internet Programming with XML

Processing XML Documents with JavaScript in Mozilla

Module Objectives

- Learn to load and parse XML documents with JavaScript in Mozilla
- Handle events while loading XML documents
- Transform XML documents with XSLT stylsheets dynamically
- Update XML components using XML DOM API
- Use XMLHttpRequest object

Loading and Parsing XML Documents

There are 2 ways to do it in Firefox or Mozilla browsers:

- DOMParser to parse XML from strings into DOM trees: This is good if you have small XML content that you can pass in `DOMParser.parseFromString()`.

- XMLHttpRequest to parse XML from files into DOM trees. It's widely used in AJAX. Its `responseXML()` method returns a parsed document object of which you can do DOM or SAX processing. Although DOMParser does have a method named `parseFromStream()`, it's actually easier to XMLHttpRequest which works for remote (not limited to HTTP) and local files. **Note** that there is cross-domain restriction imposed by the browsers, ie. you can only use XMLHttpRequest to fetch files or contents within the same host website. See topic on AJAX to learn how to deal with this restriction.

DOMParser

DOMParser can be used to parse strings and streams of XML text. It can't be used to parse HTML "tag soup". DOMParser is available to unprivileged scripts. While it's available to web pages, it's not part of any standard and not supported in other browsers.

Example

```
var parser = new DOMParser();
var doc = parser.parseFromString("<book>...</book>","text/xml");
```

The XML source string passed into `parseFromString()` is parsed into a DOM `document` object.

XMLHttpRequest

XMLHttpRequest is a JavaScript object that was created by Microsoft and adopted by Mozilla. You can use it to easily retrieve data via HTTP. Despite its name, it allows you to retrieve any type of data, not just XML, and supports connections other than HTTP (including file and ftp).

Basic Usage

Using `XMLHttpRequest` is very simple. You create an instance of the object, open a URL, and send the request. The HTTP status code of the result, as well as the result document are available in the request object afterwards.

Example

```
req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', false);
req.send(null);
if(req.status == 200)
  dump(req.responseText);
```

Note that this example works synchronously, so it will block the user interface if you call this from your javascript. You should not use this in practice.

Asynchronous Usage

Most often `XMLHttpRequest` is used to load asynchronously such as in DHTML technique so-called AJAX (Asynchronous JavaScript and XML). In asynchronous usage, you get a callback when the data has been received, which lets the browser continue to work as normal while your request is happening.

Example

```
req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', true);
req.onreadystatechange = function()
{
if (req.readyState == 4)
{
     if(req.status == 200)
      dump(req.responseText);
     else
      dump("Error loading page\n");
  }
};
req.send(null);
```

Other Properties and Methods

In addition to the properties and methods shown above, there are other useful properties and methods on the request object.

responseXML and overrideMimeType()

If you load an XML document, the `responseXML` property will contain the

document as an XMLDocument object that you can manipulate using DOM methods. For loaded non-XML documents (whose Content-Type header does not indicate XML content), the `responseXML` property is `null`. If you know in advance that the document is valid XML, you can remedy and override a broken Content-Type header, by invoking `overrideMimeType('text/xml')` (perhaps tucking on `'; charset=iso-8859-1'` or similar, as needed) before the `send()` call.

```
var req = new XMLHttpRequest();
req.open('GET', 'http://www.example.com/really-xml.txt', false);
req.overrideMimeType('text/xml');
req.send(null);
alert(req.responseXML);
```

setRequestHeader()

This method can be used to set an HTTP header on the request before you send it.

```
req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', true);
req.setRequestHeader("X-Foo", "Bar");
req.send(null);
```

getResponseHeader()

This method can be used to get an HTTP header from the server response.

```
req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', true);
req.send(null);
dump("Content-Type: " + req.getResponseHeader("Content-Type") + "\n");
```

XSL Transformations

To start, you need to create an XSLTProcessor object.

```
var processor = new XSLTProcessor();
```

Next you must import a stylesheet with the `importStylesheet()` function. It has a single parameter, which is the DOM Node of the XSLT stylesheet to import - note that the import is live, meaning that if you alter the stylesheet DOM after importing it, this will be reflected in the processing. It is however recommended to use stylesheet parameters instead of modifying the DOM. This is usually easier and can give better performance.

```
processor.importStylesheet(xsltNode);
```

Then you can use the `transformToDocument()` or `transformToFragment()` methods to transform a document using the specified XSLT stylesheet.

```
var outputDoc = processor.transformToDocument(sourceNode);
```

`transformToDocument()` takes one argument, the source node to transform,

and returns a new DOM `Document` with the results of the transformation

The resultant object is an `HTMLDocument` if the output method of the stylesheet is `html`, an `XMLDocument` for `xml`, and for output method `text` an `XMLDocument` with a single root element `<transformiix:result>` with the text as a child.

You can also use `transformToFragment()` which will return a DOM `DocumentFragment` node. This is handy because appending a fragment to another node transparently appends all the children of that fragment, and the fragment itself is not merged. Fragments are therefore useful for moving nodes around and storing them without the overhead of a full document object.

`transformToFragment` takes two arguments: the source document to be transformed (as above) and the a `Document` object that will own the fragment (all fragments must be owned by a document).

```
var ownerDocument = document.implementation.createDocument("", "test", nu
var newDocument = processor.transformToFragment(domToBeTransformed, owner
```

You can control parameters for the stylesheet using the `setParameter`, `getParameter`, and `removeParameter` methods. These all take a namespace URI and a local name as the first two parameters, with `setParameter` taking a third - the value of the parameter to be set.

The `XSLTProcessor` object also implements a `reset()` method, which can be used to remove all stylesheets and parameters then put the processor back into its initial state. This method is implemented in Mozilla 1.3 and later.

Resources:

- XML in Mozilla
- Mozilla DOM
- DOMParser
- XSLTProcessor
- XMLHttpRequest
- "Using the XMLHttpRequest Object" (jibbering.com)

Hands-on Assignment:

- **Assignment 3** due **Week 6**