

Web Front-End Development

Week 6: JSON

The content of many web sites is not static data that is part of the HTML page but stored separately in a variety of formats. We're going to look at two of the most common data formats, XML and JSON, and how to work with them.

XML

XML stands for eXtensible Markup Language

XML is a markup language designed to structure data

You'll notice it looks a lot like HTML, both are Derived from SGML (Standard Generalized Markup Language)

HTML is a markup language that describes the structure of a web page.

XML was designed to describe data and to focus on what data is.

XML provides a facility to define tags and the structural relationships between them.

XML tags are not predefined -- you must define your own tags.

XML is generally stricter than HTML and doesn't handle missing tags or incorrect nesting.

It's up to the application using XML data to process the document.

Example:

<http://creative.colorado.edu/~apierce/samples/data/harrypotter.xml>

XML has many of the same rules as HTML

- XML documents must have a root element
- Attribute values must be quoted
- All elements must have a closing tag
- Tags are case sensitive
- Elements must be properly nested
- XML must be well formed
 - Errors in XML documents will stop your XML program
- XML elements identify the nature of the content they surround.
- XML elements have relationships
 - Each element has one parent
 - Each element can have multiple children

You can create XML files using a text editor or an XML editor.

AJAX

Most web pages are synchronous so when a file is requested the user has to wait for the whole page to reload.(slide)

Since IE5 in 1998 we've had the ability to make asynchronous network calls.

Asynchronous means that the client(browser) can request new pieces of information from the server at any time. It doesn't have to wait for a page to reload. A new request can be triggered by an event like clicking on a button, hovering over an image, or whatever.(slide)

Asynchronous requests

- page sends HTTP requests while the user continues to interact with the page

- allows a web page's content to refresh without having to reload the entire page
- makes web apps more responsive and interactive

The XMLHttpRequest object provides the ability to do asynchronous data retrieval

- HTTP requests from the browser and pulls down content in small chunks.
- JavaScript class that handles the HTTP requests and responses to and from web servers without reloading the user's entire web page
- History
 - Microsoft first implemented the XMLHttpRequest object in IE5 for Windows as an ActiveX object. Starting in IE7 they adopted the XMLHttpRequest model.
 - All other browsers implemented it as well, but not as an ActiveX object.

The two most common request methods are

- GET: used to request data from a source
- POST: used to submit data to a source

Asynchronous JavaScript and XMLHttpRequest needed an acronym, so it was abbreviated to AJAX -- Asynchronous JavaScript and XML.

JavaScript

- Coordinates all browser activity
- controls the HTTP requests
- gets the results of the HTTP requests and presents the results in the user interface

XML

- Data interchange and manipulation
- Standards-based format for data exchange between client and server
- Although the 'X' in ajax stands for XML, the data transferred can be in any format such as plain text, HTML, XML, or JSON.

So AJAX is really just a term for a technology that lets you build pages that update without requiring a page reload. JavaScript does most of the heavy lifting with AJAX, and it uses the XMLHttpRequest object to handle the communication between the client and the server.

Google Maps was the first widely known Ajax app

- retrieves the next grid of map panels without needing to refresh the page
- no waiting for the user

Gmail also uses ajax to show email messages without updating the whole page.

<https://repl.it/@aileenjp/AJAX-JavaScript>

Example

- To run these examples, the files must be on the server. If you want to do it locally you can use Firefox as it has built in support for ajax calls or setup a local server.
- All our examples will use the GET request method because we are retrieving data

- XMLHttpRequest by default must be from the same origin (a combination of URI, hostname, and port number) due to security. Cross-origin permissions must be specified to be allowed.

XMLHttpRequest Object

<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

XMLHttpRequest is implemented slightly differently in Firefox, Safari, and Opera than it was in Internet Explorer's original implementation, so you'll have to try instantiating the object in a number of different ways if you're not targeting a specific browser.

- Internet Explorer versions 6 and earlier use a special class called ActiveXObject that's built into Microsoft's scripting engine.
- All other browsers create XMLHttpRequest objects using a class called XMLHttpRequest

Processing the Request

You can connect a JavaScript function to the onreadystatechange property with an anonymous function.

<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/onreadystatechange>

- When there's a change in that property, the function will be called
- The code inside handles the download

status <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

readyState <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState>

responseXML <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseXML>

JSON

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>

JSON stands for JavaScript Object Notation.

Although XML is still used in many applications, JSON has become a more popular way to structure data.

JSON uses name/value pairs to create objects of data. These objects use the same syntax as JavaScript objects. This similarity makes it really easy to parse, or convert, JSON objects to JavaScript objects using JavaScript, although it is language independent.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Like XML, JSON files can be retrieved using the XMLHttpRequest object.

JSON is considered simpler, less verbose, and faster than XML.

JSON wasn't supported in older browsers.

The JSON object has a parse() method that parses a JSON string and returns a JavaScript object.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Example:

<http://creative.colorado.edu/~apierce/samples/data/harrypotter.json>

- JSON.parse creates a JavaScript object from the JSON string

- Can't use `getElementsByTagName()` because this is not a HTML or XML file, it's a JavaScript Object
- Instead of nodes you use the key from the JavaScript object

XML vs. JSON

XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

JSON

- Fetch a JSON string
- `JSON.Parse` to parse the JSON string

jQuery

<http://api.jquery.com/category/ajax/>

Working with the `XMLHttpRequest` object was verbose and you had to deal with browser differences. jQuery makes using ajax even easier and handles the cross-browser issues for you.

<http://api.jquery.com/category/ajax/low-level-interface/> `jQuery.ajax()`

<https://repl.it/@aileenjp/AJAX-jQuery>

Example:

- `dataType` can be text, xml, json, etc
- the ajax request retrieves the data and it's automatically sent to the success function
- `getElementsByTagName()` returns a collection of elements with the tag name. We can use `firstChild` because each item only has one name, and `nodeValue` for its value

Because this is such a common function, jquery has shorthand methods that make this even easier. <http://api.jquery.com/category/ajax/shorthand-methods/>

`jQuery.get()` is a shorthand for the `ajax()` function to load data from the server using a GET request.

`jQuerygetJSON()` is a shorthand for the `ajax()` function to load JSON data from the server using a GET request.

Example:

- change to use the shorthand `get()` method, everything else is the same
- use `getJSON()` for json data. The data returned is not xml/text but a parsed JavaScript object
- use dot notation to access the values for a given key of the JavaScript Object

Fetch API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

The Fetch API was added to make fetching JSON data easier without the need for a library like jQuery. It has good support now in modern browsers, was not supported in IE or older browsers. <https://caniuse.com/#search=fetch>

<https://repl.it/@aileenjp/JSON-Fetch>

The simplest use of `fetch()` takes one argument — the path to the resource you want to fetch — and returns a promise containing the HTTP response. To extract the JSON you just call `.json` on the response. Then you can parse the JSON data and write it to the page as we've been doing.

The Fetch API is creating a Request object which represents a resource request. The Request object has more capabilities including sending options, credentials, headers, and even uploading JSON.

The Response object returned by a `fetch()` call contains all the information about the request and the response of the network request.

Response objects have other properties and methods <https://developer.mozilla.org/en-US/docs/Web/API/Response>

Promises https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

The Promise object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a *promise* to supply the value at some point in the future.

(This automatically handles what we had to code for the XMLHttpRequest object to wait for the status to change to figure out when the asynchronous call was complete).

A Promise is in one of these states:

- *pending*: initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation completed successfully.
 - Fulfilled with a value
- *rejected*: meaning that the operation failed.
 - Rejected with a reason(error)

When a promise is either fulfilled or rejected, the associated handlers queued up by a promise's `then` method are called. Promises `.then()` method is passed the Response object with the json data. We then chain another `.then()` method that takes the json data so we can then do something with the data.

Catching errors

Since `fetch()` returns a promise, we can use the `catch` method of the promise to intercept any error occurring during the execution of the request, and the processing done in the `then` callbacks.

Note that the Promise returned from `fetch()` **won't reject on HTTP error status** even if the response is an HTTP 404 or 500. Instead, it will resolve normally (with `ok` status set to `false`), and it will only reject on network failure or if anything prevented the request from completing.

Most APIs that transfer data provide it in JSON format which we'll look at Thursday.