LC 101

Unit 3 - AJAX

March 13, 2017

JSON

- The JavaScript Object Notation (JSON) is a text-based format for storing and exchanging data
  - Originally based on the JavaScript format for objects and arrays
  - Now used as a general data format across languages and systems
- JSON values are strings whose syntax is similar to JavaScripts object and array syntax
  - Keys must be in double-quotes
  - Values can only be objects, arrays, strings, numbers, true, false, or null
    - No functions or variables
    - No comments
  - Newlines and whitespace (outside of quoted strings) is unimportant

```
'[{ "name":"Alice", "age":31, "city":"New York" }, { "name":"Bob", "age":42, "city":"Saint Louis" }]'
```

AJAX

- AJAX: **A**synchronous **J**avaScript **a**nd **X**ML
- Uses the **`XMLHttpRequest`** JavaScript object built into web browsers to send requests to and receive data from servers
- Requests can be initiated by JavaScript
- **Does not require a page reload**
  - We can receive the response via JavaScript and do whatever we want with it, such as modify the existing page DOM
- Though XML is in the name, XML is not required
  - These days it is more common to use **JSON** as the data format

Request

- From the web server’s point of view, the request format is exactly the same as for a normal HTTP request
  - Uses the same URL format
  - Uses the same methods (GET, POST, etc.)
  - Includes headers
  - Can include cookies
- But since we are building the request (and not the browser), we can send data in the request in any format we want
  - We could use the same “key1=value1&key2=value2” encoding that browsers use for forms
  - We could instead use something like JSON
  - Of course, the back-end code on the server needs to be able to understand our request

Response

- The response send back from the server will also correspond to the HTTP specification
  - Will include response headers
  - Can include cookies
- But the contents of the response do not have to be HTML
  - Instead, it will often send back data in some format (generally JSON) we can use
    - Though it could be an HTML *fragment* that we could insert into the page
  - Of course, the back-end code needs to be written to do this

Asynchronous

- Normal calls made by the web browser when clicking a link are *synchronous*
  - After the request is made, the browser will pause and wait for the response
- The request made by the **`XMLHttpRequest`** object is *asynchronous*
  - The function will return immediately and **not wait** for the response
    - This is necessary to avoid the webpage freezing while waiting for the response
  - We need to include a *callback* function when we use the **`XMLHttpRequest`** object
    - This function will be called when the response is received from the server
  - In the meantime, we can continue doing other things

jQuery ajax

- Though we could directly use the **`XMLHttpRequest`** object, jQuery makes the job much easier
- jQuery's **`ajax`** method is the most generic
  - Can make any type of request with any type of data
- jQuery also has more specific helper methods
  - **`getJSON`** specifically performs a GET request for JSON data

jQuery getJSON

- To use the jQuery **`getJSON`** method we need to provide a URL and a callback function
- It will
  - Make a GET request to the URL
  - Call **`JSON.parse`** on the response body
    - So if the body is not JSON then this will cause an error
  - Call our callback function, passing the resulting object

```
$.getJSON(someURL, function(data) {
    // do something with the data object
});
```

Error handling

- The callback function passed to the **`getJSON`** method will only be called on a successful response
- We can add another callback function to be called on errors using the **`fail`** method
  - Note that the jQuery method changed from **`error`** to **`fail`** in version 3

```
$.getJSON(someURL, function(data) {
    // do something with the data object
}).fail(function() {
    // oops, handle error
});
```

Cross Domain Requests

- Web browsers enforce the same-origin policy
  - For security reasons, not allowed to make AJAX requests to other domains
- Generally a good thing
  - Keeps malicious scripts from accessing sensitive data on other servers
- For most web applications this is not a problem
  - Will only make AJAX requests to the same server that served the initial page
- But is problematic when creating AJAX APIs for other sites to use
  - Need to use a workaround like CORS or JSONP

CORS

- CORS: Cross-Origin Resource Sharing
- Created to provide a way to allow access to normally restricted resources
- Must be enabled on the server
  - Servers can restrict access by domain and/or require authentication
- Supported by modern web browsers
  - But, again, older versions of IE are problematic
- Generally transparent to the AJAX client
  - So we don’t need to do anything special
    - Unless we need to authenticate or include cookies (which aren’t normally included in cross-origin calls)

JSONP

- JSONP: JSON with padding
- Alternative to CORS that does not require special browser support
  - So it works with older browsers
- Still requires server support as the response from the server must be properly formatted
- Restricted to GET requests
- Does not actually use AJAX at all
  - Instead, JavaScript is used to insert a script tag into the DOM where the `src` attribute is the URL of the request
  - The server responds with the JSON data wrapped in a function call

JSONP

- Injected script tag would look like

```
<script src="http://otherDomain/path?callback=handleResponse"></script>
```

- Response would look like

```
handleResponse(jsonData);
```

- Would then cause the **`handleResponse`** function (defined somewhere in our code) to be called
- jQuery's **`getJSON`** will handle all of this for us if we include "**`callback=...`**" in the URL

JSONP Error Handling

- Though getJSON will automatically handle JSONP if we include "`callback=...`" in the URL, it has problems
  - Won't work if the server expects something other than "`callback`" for the callback key
  - Will not call the `fail` callback if an error occurs
- The `ajax` method gives us more flexibility
  - We can specify something other than "`callback`" for the callback key
  - We can specify a `timeout`, which will cause a call to the `fail` callback if that much time passes without a response
    - This is the only way we can get a callback on errors with JSONP

jQuery ajax JSONP Example

```
var wikiUrl = 'http://en.wikipedia.org/w/api.php?action=opensearch&search=' +
  cityStr + '&format=json';

$.ajax({
    url: wikiUrl,
    dataType: 'jsonp',
    jsonp: 'callback',  // will add a 'callback=?' parameter to the url
    timeout: 5000  // timeout in milliseconds
}).done(function(data) {
    // do something
}).fail(function(jqXHR, textStatus, errorThrown) {
    // will be called with textStatus === 'timeout' if a timeout occurs
});
```