LC 101

Unit 3 - More AJAX

April 3, 2017

XMLHttpRequest

- So far we've used jQuery to make asynchronous HTTP requests.
- Under the hood, jQuery uses the **`XMLHttpRequest`** API that is built into every modern browser
  - Hopefully you will never need to use the **`XMLHttpRequest`** API directly, but it is instructional to look at how it works

History

- A precursor to the XMLHttpRequest API was first created by Microsoft as part of their ActiveX framework in the late 1990s
- The Mozilla project (creators of the Firefox browser) was actually the first to make the API accessible via JavaScript in the early 2000s
- Despite the name, data formats other than XML can be used
  - Protocols other than HTTP can be used also
  - It is now widely considered a design flaw to have tied two disparate concepts (making a request and parsing a particular data format) together into one thing
    - But luckily the XML part is optional

Making a Request

- The `open` method on the `XMLHttpRequest` object initiates a request
  - The first argument is the request method (GET, POST, etc.)
  - The second argument is the URL
  - The third argument indicates asynchronous (`true`) or synchronous (`false`) request
  - The fourth and fifth arguments are meant to be used for basic authentication, but support seems spotty
- Although technically the third argument can be set to false to initiate a synchronous request, this should not be done outside of workers
  - This would cause the page to freeze until the request is complete
  - Support for this is being removed from the web standards
    - https://xhr.spec.whatwg.org/#the-open()-method
- After `open`, the `send` method is used to send the body of the request
  - Should call `send` with `null` for requests with empty bodies

Making a Request

- Because the request is asynchronous (when the third parameter to **`open`** is **`true`**), the **`open`** and **`send`** methods will return right away
  - To get the response, we need to add a handler for the **`load`** event on the request object

```
var req = new XMLHttpRequest();
req.addEventListener('load', function() {
  // do something with the response
});
req.open('GET', url, true);
req.send(null);
// response not available here
```

Handling the Response

- Once the response has been received, the **`status`** property of the request object will hold the numeric HTTP status code of the response
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- And the **`responseText`** property will hold the body of the response as a string
  - If the response was actually in XML format then the **`responseXML`** property would also hold the parsed XML document object

Handling the Response

```
var req = new XMLHttpRequest();
req.addEventListener('load', function() {
  if (req.status == 200) {  // 200 is the OK status
    // handle OK response
    console.log(req.responseText);
  } else {
    // handle other status
  }
});
req.open('GET', url, true);
req.send(null);
// response not available here
```

Handling Errors

- If the request completes and we get a response then the **`load`** event will fire, even if the response indicates an error
  - We can check the **`status`** code in our event handler as shown on the previous slide
- However, if the request never even completes then **`load`** will not be fired
  - We can add a handler for the **`error`** event, which will be called if the request fails entirely

```
req.addEventListener('error', function() {
  // handle request failure
});
```

Encapsulating the Request Object

- It is useful to have access to the request object in our event handlers
  - This requires that the request object be in the scope of the event handlers
  - Should avoid contaminating the global scope by *encapsulating* the request and handlers in a surrounding function
- There is also a lot of boilerplate code that is needed every time we want to make a request
  - We can avoid repeating this code by creating a function to do it
    - And we can handle responses to different requests by passing in *callback* functions

Reusable getUrl Function

```
function getUrl(url, onLoad, onError) {
  var req = new XMLHttpRequest();
  req.addEventListener('load', function() {
    onLoad(req);
  });
  req.addEventListener('error', function() {
    onError(req);
  });
  req.open('GET', url, true);
  req.send(null);
}
```

```
function onGetFooResponse(req) {
  // handle response
}

function onGetFooError(req) {
  // handle error
}

getUrl('/get/foo',
       onGetFooResponse,
       onGetFooError);
```

Exception Handling

- Wrapping the **`open`** and **`send`** calls, or the calls to **`addEventListener`**, in a **`try`** block will not catch exceptions caused by the asynchronous response

```
try {
  var req = new XMLHttpRequest();
  req.addEventListener('load', function() {
    onLoad(req);  // need to add try...catch block around this
  });
  req.addEventListener('error', function() {
    onError(req);  // need to add try..catch block around this
  });
  req.open('GET', url, true);
  req.send(null);
} catch (err) {
  // will NOT catch errors thrown by onLoad
  // or onError as those calls are not actually
  // executed here
}
```

Errors that occur in **`onLoad`** or **`onError`** will **not** be caught by this **`try...catch`** block.

Instead the error will propagate up to the **calling** scope.