COURSEWARE

XMLHttpRequests

Contents

Overview

XMLHttpRequests (XHRs) can be used to interact with external APIs.

XHR objects are used to interact with servers. We can retrieve data from a URL without having to do a full page refresh. In essence, it enables a Web page to update just part of a page without disrupting what the user is doing.

Tutorial

Despite its name, XHR can be used to retrieve any type of data, not just XML. This built-in library allows us to create, manipulate, and respond to API calls in a variety of ways.

With the XMLHttpRequest Object we can:

- Update a Web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

The following is the syntax for creating an XHR object:

```
const req = new XMLHttpRequest();
```

Sending a request To a Server

To send a request to a server, we must use the `open()` and `send()` methods of the XHR object:

```
req.open("GET", "http://mylocation:8080/api");
req.send();
```

- `req.open(method,url)` specifies the type of request
  - `method` : Tells the us if we are looking to `GET` or `POST` data
  - `url` : This is the server (file) location (where we can get the data from / where we can post the data to)

- `req.send()` Sends the request to the server(used for `GET`)
  - `req.send(string)` Sends the request to the server (used for `POST`)

`GET` or `POST`?

`method` is how information is passed to the server, we have two different approaches.

`GET` is simpler and faster than `POST`, and can be used in most cases.

Always try to use `POST` requests when:

- A file or database needs updating on the server
- Sending a large amount of data to the server (`POST` has no limitations)
- Sending user input (which can contain sensitive information such as passwords, or unknown characters); `POST` is more secure and robust than `GET`.

Sending Information to a server

`GET`

If you want to send information with a `GET` method, add the information to the URL as query parameters:

For example:

```
req.open("GET", "http://mylocation:8080/api?
forename=Chris&middlename=P&surname=Bacon");
req.send();
```

This will send the following information to the server:

- `forename` = Chris
- `middlename` = P
- `surname` = Bacon

It is generally not recommended to include query parameters for a `GET` request, so use this sparingly.

`POST`

To `POST` data, such as the content of an HTML form, add a HTTP Header with `setRequestHeader()`.

This is far safer than sending information directly through the URL as query parameters, as the values are hidden in the request headers instead.

Specify the data you want to send in the `send(data)` method:

```
req.open();
req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
req.send("forename=Chris&middlename=P&surname=Bacon"); // {/*Example request
body*/}
```

Always set headers **after** `open()` but **before** `send()`.

This will behave exactly as the previous code, but the values are not sent as query parameters inside the URL.

Handling Responses

Requests are asynchronous - to handle responses from the server to these requests, we can use *callbacks*.

XHR provides several properties and event methods that can be used to handle various state changes during a request:

Properties

- The `readyState` property holds the status of the XMLHttpRequest

- The `status` property and the `statusText` property holds the status of the XHR object

`readyState`

`readyState` holds the status of the XMLHttpRequest.

- 0: Request is not initialised
- 1: Server connection established
- 2: Request received
- 3: Processing request
- 4: Request finished; response is ready

`status`

The most common HTTP `status` codes for request responses include:

- 200: OK
- 403: Forbidden
- 404: Not found

A complete list of these is available on the Mozilla HTTP Status Messages page.

When `readyState` is 4 and `status` is 200, the response is ready.

Events

Common events include:

- `onload` - This is called when the response has been received. Called once.
- `onreadystatechange` - This is called whenever the `readyState` property of the request changes. Can be called multiple times.

Let's have a look at an example:

```
req.onreadystatechange = () => {
  if (req.status === 200 && req.readyState === 4) {
    console.log(req.responseText);
  } else {
    console.log("handle error");
  }
};
```

Response Headers

Retrieving response headers is as straightforward as setting headers to a request:

```
req.getResponseHeader(header); // E.g. 'Content-Type'
```

The Final Look

Pulling everythign together, a typical set of requests might look like the following:

```
const req = new XMLHttpRequest();
req.onreadystatechange = () => {
  // Example handle logic
  if (req.status === 200 && req.readyState == 4) {
    if (req.getResponseHeader("Content-Type") === "application/json") {
      console.log("oh look its some JSON: " + req.responseText);
    } else {
      console.log(
        "Looks like its not JSON but lets see what it is... " + req.responseText
      );
    }
  } else {
    console.log("Oh no... handle error");
  }
};
req.open("GET", "http://my.api");
req.setRequestHeader("example-header", "some-value");
req.send();
```

Object Methods

Method	Description	Parameters
`new XMLHttpRequest()`	Creates a new XMLHttpRequest object	
`abort()`	Cancels the current request	
`getAllResponseHeaders()`	Returns header information	
`getResponseHeader()`	Returns specific header information	
`open(method,url)`	Specifies/Initialises the request	- *`method`*: the request type `GET` or `POST`, - *`url`*: the file location
`send()`	Sends the request to the server.	Used for `GET` requests
`send(string)`	Sends the request to the server.	Used for `POST` requests
`setRequestHeader()`	Adds a label/value pair to the header to be sent	*(N.B. Must be called after `open()` but before `send()`*
`onreadystatechange()`	Defines a function to be called when the `readyState` property changes	
`onload()`	Fired when transaction completes successfully	

Object Properties

Property	Description	Parameters
`readyState`	Holds the status of the XMLHTTPRequest	0: Request Not initialised --> 4: Request finished and response ready

Property	Description	Parameters
`responseText`	Returns the response of the data as a string	
`responseXML`	Returns the response of the data as XML data	
`status`	Returns the status-number of a request	Mozilla HTTP Status Messages page
`statusText`	Returns the status-text (e.g. "OK" or "Page not Found")	

Read the XMLHttpRequest Documentation for more information.

Exercises

Using this link as your API, create the following:

1. `GET` request for 'List User'
2. `GET` request for 'Single User'
3. `POST` request for 'Create'
4. `POST` request for 'Register - Successful'
5. `POST` request for 'Login - Successful'

You must expect the correct response in return and use the methods taught in this module to create the XMLHttpRequest objects.

▶ Solution