Solomon Ayoola

Axios vs. Fetch?

Solomon Ayoola

Jan 29, 2020 • 10 Min read • 35,887 Views

Jan 29, 2020 • 10 Min read • 35,887 Views

Web Development React

Introduction

99

Introduction

One of the fundamental tasks of a frontend application is to communicate with servers through the HTTP protocol. JavaScript can send network requests to the server and load new information whenever needed without reloading the page.

The term for that kind of operation is Asynchronous JavaScript And XML (AJAX), which uses the XMLHttpRequest object to communicate with the servers. It can send and receive information in various formats, including JSON, XML, HTML, and text files.

In this guide, we will be looking at how to make a network request to get information from the server asynchronously with the fetch() API and Axios and learn how they can be used to perform different operations.

Fetch

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global `fetch()` method that provides an easy, logical way to fetch resources asynchronously across the network.

The `fetch()` method takes one mandatory argument—the path to the resource you want to fetch—and returns a Promise that resolves with an object of the built-in Response class as soon as the server responds with headers.

The code below demonstrates a very basic fetch request in which we are fetching a JSON file across the network.

javascript

```
fetch('examples/example.json')
  .then((response) => {
    // Do stuff with the response
```

```
  })
  .catch((error) => {
    console.log('Looks like there was a problem: \n', error);
  });
```

The second argument that the `fetch()` method takes is the request object.

JSON

```
{
  method: 'POST', // *GET, POST, PUT, DELETE, etc.
  mode: 'cors', // no-cors, *cors, same-origin
  cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-
  credentials: 'same-origin', // include, *same-origin, omit
  headers: {
    'Content-Type': 'application/json'
  },
  redirect: 'follow', // manual, *follow, error
  referrerPolicy: 'no-referrer', // no-referrer, *client
  body: JSON.stringify(data) // body data type must match "Content-Type"
}
```

Once a `Response` is retrieved, the returned object contains the following properties:

- `response.body`: A simple getter exposing a ReadableStream of the body contents
- `response.bodyUsed`: Stores a Boolean that declares whether the body has been used in a response yet

- `response.headers`: The headers object associated with the response
- `response.ok`: A Boolean indicating whether the response was successful or not
- `response.redirected`: Indicates whether or not the response is the result of a redirect
- `response.status`: The status code of the response
- `response.statusText`: The status message corresponding to the status code
- `response.type`: The type of the response
- `response.url`: The URL of the response

There are a number of methods available to define the body content in various formats:

- `response.json()`: Parse the response as JSON
- `response.text()`: Read the response and return as text
- `response.formData()`: Return the response as FormData object
- `response.blob()`: Return the response as Blob
- `response.arrayBuffer()`: Return the response as ArrayBuffer

Axios

Axios is a Javascript library used to make http requests from `node.js` or `XMLHttpRequests` from the browser, and it supports the Promise API that is native to JS ES6.

Some core features of Axios, according to the documentation, are:

- It can be used intercept http requests and responses.
- It automatically transform request and response data.
- It enables client-side protection against XSRF.
- It has built-in support for download progress.
- It has the ability to cancel requests.

Axios does not come as a native JavaScript API, so we will have to manually import into our project. To get started, we will have to include the following commands:

- Using cdn

```html
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

- Using npm

```node
npm install axios
```

- Using bower

```bash
bower install axios
```

And make a request as follows:

javascript

```
axios.get('examples/example.json')
  .then((response) => {
              // handle success
    1         console.log(response);
    2       })
    3       .catch((error) => {
    4         // handle error
    5         console.log(error);
    6       })
    7
    8
    9
```

Axios also provides more functions to make other network requests as well, matching the HTTP verbs that you wish to execute, such as:

- `axios.request(config)`
- `axios.get(url[, config])`
- `axios.delete(url[, config])`
- `axios.head(url[, config])`
- `axios.options(url[, config])`
- `axios.post(url[, data[, config]])`
- `axios.put(url[, data[, config]])`
- `axios.patch(url[, data[, config]])`

You can check out the comprehensive request config in the official documentation.

The response from a request contains the following information:

- `response.data`: The response provided by the server
- `response.status`: The HTTP status code from the server response
- `response.statusText`: HTTP status message from the server response
- `response.headers`: The headers that the server responded with
- `response.config`: The config that was provided to `axios` for the request
- `response.request`: The request that generated this response

We're now familiar with the definitions and usage of `axios` and `fetch()`. Let's focus our attention on how to perform some real world operations with both Axios and Fetch.

JSON data

There is a two-step process when handling JSON data with `fetch()`. First, we have to make the actual request, and then we call the `.json()` method on the response.

javascript

```
fetch("examples/example.json") // first step
  .then(response => response.json()) // second step
  .then(data => {
    console.log(data)
  })
  .catch(error => console.error(error))
```

Axios automatically transforms JSON data once the request is resolved, so we do not have to do much here.

javascript

```
axios
  .get("examples/example.json")
  .then(response => {
    console.log(response.data)
  })
  .catch(error => {
    console.log(error)
  })
```

Error Handling

Handling errors with Axios is pretty straightforward because bad responses (like 404 or 500) will reject the promise.

javascript

```
axios
  .get("examples/example.json")
  .then(response => {
    console.log("response", response)
  })
  .catch(error => {
    if (error.response) {
      // The request was made and the server responded with a status code
```

```
      // that falls out of the range of 2xx
      console.log(error.response.data)
    } else if (error.request) {
      // The request was made but no response was received
      // `error.request` is an instance of XMLHttpRequest in the browser
      // http.ClientRequest in node.js
      console.log(error.request)
    } else {
      // Something happened in setting up the request that triggered an E
      console.log("Error", error.message)
    }
    console.log(error.config)
  })
```

Evaluating the success of responses is particularly important when using `fetch()` because bad responses (like 404 or 500) still resolve. The only time a fetch promise will reject is if the request was unable to complete. The code would look something like this:

javascript

```
fetch("examples/example.json")
  .then(response => {
    if (!response.ok) {
      throw Error(response.statusText)
    }
    return response.json()
  })
  .then(data => {
    console.log(data)
  })
  .catch(error => console.error(error))
```

Intercepting Requests and Responses

One of the key features of Axios is its ability to intercept HTTP requests. Interceptors can be really helpful when we want to examine or change HTTP requests from our application to the server, such as retrieving a token from local storage and including it in all requests.

javascript

```
// Add a request interceptor
axios.interceptors.request.use(
  function(config) {
    // Do something before request is sent
    return config
  },
  function(error) {
    // Do something with request error
    return Promise.reject(error)
  }
)

// Add a response interceptor
axios.interceptors.response.use(
  function(response) {
    // Do something with response data
    return response
  },
```

```
    function(error) {
      // Do something with response error
      return Promise.reject(error)
    }
  )

  // sent a GET request
  axios.get("examples/example.json").then(response => {
    console.log(response.data)
  })
```

By default, `fetch()` does not provide a way to intercept HTTP requests. We can overwrite the global `fetch()` method and define our own interceptor, like this:

javascript

```
fetch = (originalFetch => {
  return (...arguments) => {
    const result = originalFetch.apply(this, arguments)
    return result.then(console.log("Request was sent"))
  }
})(fetch)

fetch("examples/example.json")
  .then(response => response.json())
  .then(data => {
    console.log(data)
  })
```

Conclusion

In this guide, we have looked at Fetch and Axios and checked out some real-world operations. While Axios is widely supported among the majority of browsers and can also be used in the nodejs environment, Fetch, on the other hand, isn't widely supported among old browsers. If you need to support older browsers, a polyfill is available.

Here are some useful links to learn more about various use-cases of both Axios and Fetch:

- Intro to Fetch
- You may not need Axios
- Axios Beginner Guide
- How to make HTTP requests like a pro with Axios