Dmitri Pavlutin
I help developers understand Frontend technologies

How to Timeout a fetch() Request

Updated December 3, 2020

fetch

11 Comments

When developing an application that uses the network, the first rule to remember is *don't rely on the network*.

The network is unreliable because an HTTP request or response can fail for many reasons:

- The user is offline
- DNS lookup failed
- The server doesn't respond
- The server responds but with an error
- and more.

Users are OK to wait up to 8 seconds for simple requests to complete. That's why you need to set a timeout on the network requests and inform the user after 8 seconds about the network problems.

I'm going to show you how to use `setTimeout()`, the abort controller, and `fetch()` API to make requests with a configurable timeout time (interesting demos included!).

1. Default fetch() timeout

By default a `fetch()` request timeouts at the time indicated by the browser. In Chrome a network request timeouts at 300 seconds, while in Firefox at 90 seconds.

```
async function loadGames() {
  const response = await fetch('/games');
  // fetch() timeouts at 300 seconds in Chrome
  const games = await response.json();
  return games;
}
```

300 seconds and even 90 seconds are way more than a user would expect a simple network request to complete.

In the demo the `/games` URL was configured to respond in 301 seconds. Click *Load games* button to start the request, and it will timeout at 300 seconds (in Chrome).

2. Timeout a fetch() request

`fetch()` API by itself doesn't allow canceling programmatically a request. To stop a request at the desired time you need additionally an abort controller.

The following `fetchWithTimeout()` is an improved version of `fetch()` that creates requests with a configurable timeout:

```
async function fetchWithTimeout(resource, options = {}) {
  const { timeout = 8000 } = options;
```

```
  const controller = new AbortController();
  const id = setTimeout(() => controller.abort(), timeout);

  const response = await fetch(resource, {
    ...options,
    signal: controller.signal
  });
  clearTimeout(id);

  return response;
}
```

First, `const { timeout = 8000 } = options` extracts the timeout param in milliseconds from the `options` object (defaults to 8 seconds).

`const controller = new AbortController()` creates an instance of the abort controller. This controller lets you stop `fetch()` requests at will. Note that for each request a new abort controlled must be created, in other words, controllers aren't reusable.

`const id = setTimeout(() => controller.abort(), timeout)` starts a timing function. After `timeout` time, if the timining function wasn't cleared, `controller.abort()` is going to abort (or cancel) the fetch request.

Next line `await fetch(resource, { ...option, signal: controller.signal })` starts properly the fetch request. Note the special `controller.signal` value assigned to `signal` property: it connectes `fetch()` with the abort controller.

Finally, `clearTimeout(id)` clears the abort timing function if the request completes faster than `timeout` time.

Now here's how to use `fetchWithTimeout()`:

```
async function loadGames() {
  try {
    const response = await fetchWithTimeout('/games', {
      timeout: 6000
    });
    const games = await response.json();
    return games;
  } catch (error) {
    // Timeouts if the request takes
    // longer than 6 seconds
```

```
    console.log(error.name === 'AbortError');
  }
}
```

`fetchWithTimeout()` (instead of simple `fetch()`) starts a request that cancels at `timeout` time — 6 seconds.

If the request to `/games` hasn't finished in 6 seconds, then the request is canceled and a timeout error is thrown.

You can use the expression `error.name === 'AbortError'` inside the `catch` block to determine if there was a request timeout.

Open the demo and click *Load games* button. The request to `/games` timeouts because it takes longer than 6 seconds.

3. Summary

By default a `fetch()` request timeouts at the time setup by the browser. In Chrome, for example, this setting equals 300 seconds. That's way longer than a user would expect for a simple network request to complete.

A good approach when making network requests is to configure a request timeout of about 8 - 10 seconds.

As shown in the post, using `setTimeout()` and abort controller you can create `fetch()` requests configured to timeout when you'd like to.

Check the browser support of the abort controller because as of 2020 it is an experimental technology. There's also a polyfill for it.

Please note that without the use of an abort controller there's no way you can stop a `fetch()` request. Don't use solutions like this.

What other good practices regarding network requests do you know?

Like the post? Please share!

Suggest Improvement

Quality posts into your inbox

I regularly publish posts containing:

- Important JavaScript concepts explained in simple words
- Overview of new JavaScript features
- How to use TypeScript and typing
- Software design and good coding practices

Subscribe to my newsletter to get them right into your inbox.

Enter your email Subscribe

Join 5519 other subscribers.

About Dmitri Pavlutin

Tech writer and coach. My daily routine consists of (but not limited to) drinking coffee, coding, writing, coaching, overcoming boredom 😉.

Recommended reading:

An Interesting Explanation of async/await in JavaScript

javascript async await

How to Use Fetch with async/await

fetch async await

© 2022 Dmitri Pavlutin

Licensed under CC BY 4.0