16 AUGUST 2022

How to Handle Multiple Named Exports in One JavaScript File

Modern, interesting, and interactive websites and web applications generally need a lot of JavaScript. It's easy to run into huge, tangled codebases, with one file (usually named `site.js` or `scripts.js`) comprising thousands of lines of code. An absolute maintenance nightmare.

Once upon a time, this made some sense simply for the performance gains that could be achieved by combining multiple scripts into a single file, although if you did this manually it could genuinely become untenable extremely quickly.

Thankfully, we can avoid this problem altogether with modern, modular JavaScript development. ES6 allows us to break up our spaghetti-junction JavaScript file into smaller files, and then `import` or `export` the classes (which can contain functions or data) into and out of each of the files. This helps keep your codebase clean and easy to maintain and allows you and your team to decide upon and develop your own system of classification and storage of different functional types within your codebase.

All of this is great, but what if you want to export multiple classes from one single JavaScript file? Well, no biggie - we can handle that by using Named Exports.

One-by-One

If you're working with multiple named exports out of a single JavaScript file, you can handle that by exporting each class as you create it. For example (some code from a project I worked on many years ago - please excuse the old-school syntax):

```
export class Band {
  function getMembers(bandName) {
```

```
  function getGenre(bandName) {
    return bandName.genre;
  }

  function getAlbum(albumName, bandName) {
    if(albumName == 'latest') {
      return bandName.album[0];
    } else {
      return bandName.albumName;
    }
  }
}

export class Person {
  function getBand(personName) {
    // fetch bandName where `members` contains personName
  }
  function getInstrument(personName) {
    return personName.instruments.main;
  }
}
```

This is handy if you prefer to work straight down as you write, but can cause some issues in terms of readability. I would recommend using the next method, which allows you to export each class in one single statement at the bottom of the file.

All at Once

This is a much nicer way to export your classes, in my opinion. We'll set up the exact same classes and functions but add an export statement at the end instead of exporting each one individually:

```
class Band {
  function getMembers(bandName) {
    return bandName.members;
  }

  function getGenre(bandName) {
    return bandName.genre;
  }

  function getAlbum(albumName, bandName) {
    if(albumName == 'latest') {
      return bandName.album[0];
    } else {
  }
}

class Person {
  function getBand(personName) {
    // fetch bandName where `members` contains personName
  }
  function getInstrument(personName) {
    return personName.instruments.main;
  }
}

export { Band, Person };
```

I prefer this way as it keeps your class declarations uncluttered, and doesn't add yet another thing to bear in mind while you're developing, allowing you to maintain some flow.

On top of that, the single export statement sitting predictably at the bottom of your files will make it easier to gather, at a glance, what you're exporting from each file. This should help to make your maintenance quicker and less stressful, preventing you from having to trawl through files and taking the time to look for the export statements hidden throughout.

The Default Export

You can export one default class per file with the ES6 modular syntax. A lot of codebases out in the wild don't use default exports, preferring to rely on more explicit export and import statements as it makes your development clearer (as well as allowing you to use the autocomplete functionality in your IDE, if your IDE of choice has that feature).

Here's how you can export a default class with the one-by-one method (check out the default keyword in the first export statement:

```
// One-by-one
export default class Band {
  function getMembers(bandName) {
    return bandName.members;
  }

  function getGenre(bandName) {
    return bandName.genre;
  }

  function getAlbum(albumName, bandName) {
    if(albumName == 'latest') {
      return bandName.album[0];
    }
  }
}

export class Person {
  function getBand(personName) {
    // fetch bandName where `members` contains personName
  }

  function getInstrument(personName) {
    return personName.instruments.main;
  }
}
```

and here’s how you’d handle it with the all-in-one statement:

```
export { Band as default, Person };
```

The Wrap-Up

Modular JavaScript development is the way forward. It’s easy, keeps your codebases clear and straightforward to maintain, and can seriously cut down on the fatigue and stress you might get opening up and working inside a monolithic site.js file, comprising thousands of lines of code to power everything on your site.

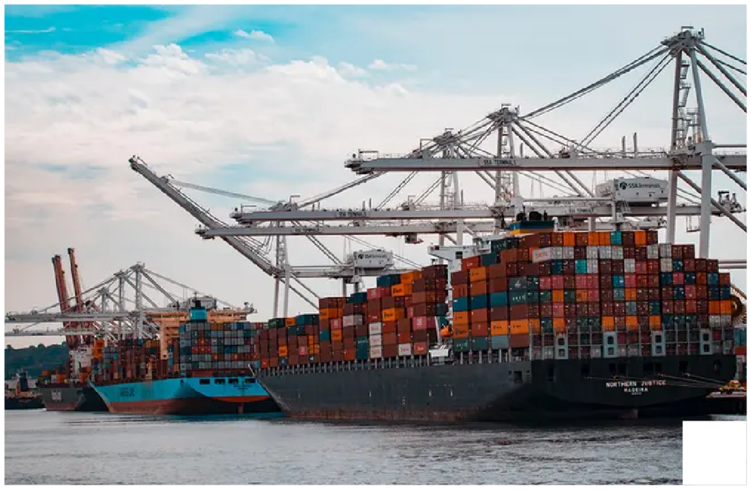
If you’re looking for a how-to guide on this functionality from the other side, check out How to Import all Named Exports from a JavaScript file.

1,897

CATEGORIES: Front-End Development ES6 JavaScript

More articles

View all



28 JULY 2022

How to Import All Named Exports From a JavaScript File

01 JANUARY 2024

Staying Current: Automating Copyright Year Updates

07 APRIL 2021

What is Front-End Development?

"John Kavanagh is an exceptional programmer who consistently delivered high-quality work at Selfridges. He possesses exceptional front-end development skills and has a remarkable ability to tackle complex problems with creative and effective solutions.

What sets John apart is his strong work ethic and dedication to delivering results. He is a reliable and proactive team player, willing to go the extra mile to ensure project success. I was particularly impressed with his ability to resolve issues and make the explanations simple and succinct. He communicates effectively, collaborates seamlessly with team members, and approaches challenges with a positive and solution-oriented mindset."

Gene Da Rocha

GENERATIVE AI ENGINEER