Matt Lim

Follow 46 Followers About

The three differences between require and import in Node.js

Matt Lim Aug 18, 2020 · 2 min read

These differences apply to the `import` statement, not the `import` expression (see this page for more info on the latter, which can be used to import modules dynamically).

This is part one of a four part series about JavaScript modules.

3. Using ES modules with CommonJS modules in the browser

4. Using ES modules with CommonJS modules with webpack

Check out full code examples here: https://github.com/arcticmatt/javascript_modules/tree/master/import_vs_require.

. . .

1. **When using `import ... from ...`, the module path must be a string literal. When using `require`, the module path can be dynamic.**

For example, this works:

```
const name = "module2";
const obj = require(`./${name}`);
```

But this will result in the error `SyntaxError: Unexpected template string` when run with `node`.

```
const name = "module2";
import { func } from `./${name}`;
```

Why is this? See the next point.

. . .

2. Order of execution differs. `require` will be run inline, after the code above it has executed. `import` runs before the rest of the script.

Assuming `module2.js` has `console.log("require module2");` at the top, then if we run this code:

```
console.log("require module1");

const obj = require("./module2");
console.log(`module2 = ${obj.module2}`);
```

it results in the following:

```
require module1
require module2
module2 = require module2
```

With ES modules, on the other hand…

```
  import module2 from "./module2.js";
  console.log(`module2 = ${module2}`);
```

Running this results in the following:

```
require module2
require module1
module2 = require module2
```

ES modules: A cartoon deep-dive goes into this subject in much more depth.

. . .

3. You can leave out a `.js` extension when importing a local module with `require`, but cannot do the same when using `import`.

This is true by default in the browser and Node.js. For example, `require("./module2")` works, but the equivalent using `import` must be written as `import module2 from "./module2.js"`. If you omit the extension in Node.js, you will get an error like: `Error [ERR_MODULE_NOT_FOUND]: Cannot find module …`

In Node.js, you can use the `--experimental-specifier-resolution=node` option to circumvent this behavior, i.e. this will

Furthermore, webpack has an option that changes this behavior. Specifically, if `resolve.enforceExtension` is `true`, then extensions are required. This option is set to `false` by default, which explains why in many frameworks (like Next.js, which uses webpack behind the scenes) you can use `import` without specifying file extensions.

JavaScript Modules Commonjs Import Nodejs