

**The future of  
modules**



CJ Silverio  
CTO, @ceejbot

you use a **lot** of modular javascript  
billions & billions of packages

**if you use code from npm  
the future of the language  
matters to you**

a module is a scope plus  
an API that lets you reuse it

**files are modules**  
**groups of files can be**

give that group of files a  
package.json and it's a **package**

**module syntax:**  
**this is what I'm sharing**  
**this is what I'm re-using**

**CommonJS** is one module syntax  
**ECMAScript Modules** is another

**you've seen the syntax a million times  
but here's a fast refresher**

```
const chalk = require('chalk'); // cjs
import chalk from 'chalk'; // esm

console.log(chalk.bold('hello world!'));
```

```
const { yellow, bold } = require('chalk'); // cjs
import { yellow, bold } from 'chalk'; // esm

console.log(`${bold('hello')} ${yellow('world!')}`);
```

```
const mycolorapi = { red, green, blue, bold, ...};  
  
module.exports = mycolorapi; // cjs  
export default mycolorapi; // esm
```

**why is this worth a talk?  
it isn't because of syntax**

it's because the transition  
has not been smooth

TopGearGifs.tumblr



**node needs both  
and it still doesn't do ESM natively**

**ESM** is the future  
of modular JavaScript

**in order to understand the future  
we must first understand the past**

in 2009 JS had no module spec  
but CommonJS had one

**there was a lot of server-side js going on  
and people needed a way to share code**

**node adopted the CommonJS syntax  
and wrote a loader**

**node's resolution algorithm**  
turns a **human-friendly module name**  
into a **path to code that can be loaded**

```
const chalk = require('chalk');
// loads './node_modules/chalk/index.js'
const other = require('./chalk');
// loads './chalk.js' if it finds it
```

**node's take on modules is useful  
you're using the heck out of it today**

**browser-side js was still concatenating files**



**TC39** started working on the problem in 2009

**import & export  
arrived in ES2015**

**esm design choices differ from cjs!  
its intended uses are different!**

**inherently asynchronous not synchronous**

(mostly) not dynamic:  
can't import from a **variable** source

**designed to support static analysis**  
**the import() variation is, however, dynamic**

see Lin Clark's cartoon deep-dive  
for an exploration of the details

**static analysis is important  
when your target is the browser**

**node's nested deps are a great solution  
server-side, where disk space is cheap**

**static analysis & tree shaking help  
minimize download burden in FE**

A black and white photograph of a man with dark hair and a mustache, wearing a dark flight suit and goggles. He is looking out of a cockpit window, which shows the interior of a plane and some structural elements. The background is bright and overexposed.

**SPEEEED!!**

**so front-end focused developers want ESM**

in between 2009 and now  
something very **interesting** happened

node's user base changed

**throwbacks like me still write server-side js  
most of you use node for frontend tooling**

**you didn't want to wait for new js features  
you wanted them now**

**transpilation took off**  
**babel was the new hotness**

**babel transpiled the ESM syntax into CJS  
and the future looked like it was now**

**but babel was CJS behind the scenes  
which meant node's loader was doing the work**

**problem: there is no esm loader**

**no loader? no loader.  
well, html's loader.**

```
<script type="module" src="entry.js"></script>

import {chalk} from 'https://example.com/js/chalk/index.js';
import {foo} from '../js/foo.js';
```

**file paths are super-predictable  
and super-annoying to write by hand**

**node's loader is part of its magic  
and there's no magic in spec-compliant ESM loading**

**babel set some expectations  
about how esm in node should behave**

**there's no escape from build steps in sight**

**meanwhile, back in Gotham City...**

**node's been using CJS for 9 years**

npm has **725K** public packages

**you will recall, front-end dev is now what  
people are using node to do**

**node should use JS standards  
when those standards exist**

**therefore node should use ESM.  
but what does this mean?**

**ESM in node looked like  
it was not going good places**

**very depressing talk by Myles Borins  
about how hard the integration was**

**the systems have to co-exist  
but they have little in common**

how do CJS & ESM **interoperate?**

can you `require()` an esm file?  
can you `import` a cjs file?

**how do you know which is which without parsing?**

**node has no type="module" marker  
for its entry points**

**so it proposed .mjs as a file suffix for  
javascript with certain features**



TopGear on tumblr

I'm the Stig table-flipping.

**"Well," I said. "What happens if  
we choose different constraints?"**

**SOUNDS LIKE A CHALLENGE,**

**top**

**HOW HARD CAN IT BE.**

**well, not very hard.**  
**Chris Dickinson & I implemented a**  
**usable take on ESM + CJS in node.**

**no mjs, parse entry point to figure out  
require module consumer know its api in advance**

reception was mixed



BrendanEich

@BrendanEich

Follow



This is good:



npm's proposal for supporting ES modules in node

npm's proposal for supporting ES modules in node

gist.github.com

4:35 PM - 12 Jan 2018

137 Retweets 283 Likes



HONEYCAM



**technical problems are easy**

**people problems are hard**

**what is node for?**  
**how do we use node?**  
**where is node going?**

**fundamental differences in answers  
lead to fundamental differences in designs**

**talking  
thinking  
typing**

**the node modules problem needed more talking**



great news!

**node has a modules working group  
doing the hard work of talking**

it started with use cases

TODO TODO some examples of use cases

**this brings us to the present state  
not quite there yet, but soon!**

**you can use ESM now if you want**

**if you are using babel in your build  
you have no reason not to**

**those of us in server-side nodeland  
had no other options until recently**

<https://github.com/standard-things/esm>

npm init esm -y

*record scratch noise*  
npm install -g npm@latest

**npm@6 has some goodies**

like templated init  
even better than npm init -y

**npm audit & npm audit fix**  
**report and automatically fix**  
**known vulnerabilities in your dep tree**

**npm, Inc, is a company that  
sells goods and services**

**you can pay us to run a very nice  
single-tenant registry for you**

**now back to modules...**

to use ESM in node **right now**

npm init esm -y



**transpilation on the fly by  
monkey-patching the module object**

```
require = require('esm')(module);
const Widget = require('./main.js').default;

// over in main.js
import fs from 'fs';
import cjs from './cjs.js';
const esm = require('./esm.js');
export default class Widget { ... }
```

**Q: Was that transparent interop?  
A: I believe it was, Bob.**

see talk repo for code examples:  
[github.com/ceejbot/future-of-modules](https://github.com/ceejbot/future-of-modules)

is node going to end up there?  
I hope so!

**so what is the future of modules?**

**learn the **syntax** now if you haven't yet**

this trend will continue:  
the future of node is **front-end**

**the future of node is you**

**future-you will assume ESM not CJS**

you'll install trillions of packages from **npm**  
but they'll all be **ESM** not CJS

**CommonJS modules won't go away or stop working  
you'll slowly use more & more ESM**

**no need to migrate until you have a reason**

**the future of js modules will be just fine**

**AND ON THAT**

**BOMBSHELL...**



you

**[github/ceejbot/future-of-modules](https://github.com/ceejbot/future-of-modules)**