ES Modules are terrible, actually

> This post was originally published at https://gist.github.com/joepie91/bca2fda868c1e8b2c2caf76af7dfcad3, which was in turn adapted from an earlier Twitter thread.

It's incredible how many collective developer hours have been wasted on pushing through the turd that is ES Modules (often mistakenly called "ES6 Modules"). Causing a big ecosystem divide and massive tooling support issues, for... well, no reason, really. There are no actual advantages to it. At all.

It looks shiny and new and some libraries use it in their documentation without any explanation, so people assume that it's the new thing that must be used. And then I end up having to explain to them why, unlike CommonJS, it doesn't actually work everywhere yet, and may never do so. For example, you can't import ESM modules from a CommonJS file! (Update: I've released a module that works around this issue.)

And then there's Rollup, which apparently requires ESM to be used, at least to get things like treeshaking. Which then makes people believe that treeshaking is not possible with CommonJS modules. Well, it is - Rollup just chose not to support it.

And then there's Babel, which tried to transpile `import`/`export` to `require`/`module.exports`, sidestepping the ongoing effort of standardizing the module semantics for ESM, causing broken imports and `require("foo").default` nonsense and spec design issues all over the place.

And then people go "but you can use ESM in browsers without a build step!", apparently not realizing that that is an utterly useless feature because loading a full dependency tree over the network would be unreasonably and unavoidably slow - you'd need as many roundtrips as there are levels of depth in your dependency tree - and so you need some kind of build step anyway, eliminating this entire supposed benefit.

And then people go "well you can statically analyze it better!", apparently not realizing that ESM doesn't actually change any of the JS semantics other than the `import`/`export` syntax, and that the `import`/`export` statements are equally analyzable as top-level `require`/`module.exports`.

"*But in CommonJS you can use those elsewhere too, and that breaks static analyzers!*", I hear you say. Well, yes, absolutely. But that is inherent in dynamic imports, which by the way, ESM also

supports with its dynamic `import()` syntax. So it doesn't solve that either! Any static analyzer still needs to deal with the case of dynamic imports *somehow* - it's just rearranging deck chairs on the Titanic.

And *then*, people go "but now we at least have a standard module system!", apparently not realizing that CommonJS was *literally that*, the result of an attempt to standardize the various competing module systems in JS. Which, against all odds, *actually succeeded*!

... and then promptly got destroyed by ESM, which reintroduced a split and all sorts of incompatibility in the ecosystem, rather than just importing some updated variant of CommonJS into the language specification, which would have sidestepped almost all of these issues.

And while the initial CommonJS standardization effort succeeded due to none of the competing module systems being in particularly widespread use yet, CommonJS is so ubiquitous in Javascript-land nowadays that it will never fully go away. Which means that runtimes will forever have to keep supporting two module systems, and **developers will forever be paying the cost of the interoperability issues between them**.

But it's the future!

Is it really? The vast majority of people who believe they're currently using ESM, aren't even actually doing so - they're feeding their entire codebase through Babel, which deftly converts all of those snazzy `import` and `export` statements back into CommonJS syntax. Which works. So what's the point of the new module system again, if it all works with CommonJS anyway?

And it gets worse; `import` and `export` are designed as special-cased statements. Aside from the obvious problem of needing to learn a special syntax (which doesn't *quite* work like object destructuring) instead of reusing core language concepts, this is also a downgrade from CommonJS' `require`, which is a *first-class expression* due to just being a function call.

That might sound irrelevant on the face of it, but it has very real consequences. For example, the following pattern is simply **not possible** with ESM:

```
const someInitializedModule = require("module-name")(someOptions);
```

Or how about this one? Also no longer possible:

```
const app = express();
// ...
app.use("/users", require("./routers/users"));
```

Having language features available as a first-class expression is one of the most desirable properties in language design; yet for some completely unclear reason, ESM proponents decided to *remove* that property. There's just no way anymore to directly combine an `import` statement with

some other JS syntax, whether or not the module path is statically specified.

The only way around this is with `await import`, which would break the supposed static analyzer benefits, only work in async contexts, and even then require weird hacks with parentheses to make it work correctly.

It also means that you now need to make a choice: do you want to be able to use ESM-only dependencies, or do you want to have access to patterns like the above that help you keep your codebase maintainable? ESM or maintainability, your choice!

So, congratulations, ESM proponents. You've destroyed a successful userland specification, wasted many (hundreds of?) thousands of hours of collective developer time, many hours of my own personal unpaid time trying to support people with the fallout, and created ecosystem fragmentation that will never go away, in exchange for... fuck all.

This is a disaster, and the only remaining way I see to fix it is to stop trying to make ESM happen, and deprecate it in favour of some variant of CommonJS modules being absorbed into the spec. It's not too late *yet*; but at some point it will be.

Revision #1
Created 2024-12-11 01:14:30 UTC by joepie91
Updated 2024-12-11 18:44:19 UTC by joepie91