Rich Harris
Graphics editor, @nytimes investigations team. Open sourceror
Apr 6, 2017 · 4 min read

Webpack and Rollup: the same but different

This week, Facebook merged a monster pull request into React that replaced its existing build process with one based on Rollup, prompting several people to ask 'why did you choose Rollup over webpack'?

Which is a completely reasonable question. Webpack is one of the modern JavaScript community's greatest success stories, with millions of downloads every month powering tens of thousands of websites and applications. It has a large ecosystem, dozens of contributors, and—unusually for a community open source project—meaningful financial support.

By comparison, Rollup is a minnow. But React isn't alone—Vue, Ember, Preact, D3, Three.js, Moment, and dozens of other well-known libraries also use Rollup. So what's going on? Why can't we have just one JavaScript module bundler that everyone agrees on?

A tale of two bundlers

webpack was started in 2012 by Tobias Koppers to solve a hard problem that existing tools didn't address: building complex single-page applications (SPAs). Two features in particular changed everything:

1. **Code-splitting** makes it possible to break your app apart into manageable chunks that can be loaded on-demand, meaning your users get an interactive site much faster than if they had to wait for the whole application to download and parse. You *can* do this manually, but, well… good luck.
2. **Static assets** such as images and CSS can be imported into your app and treated as just another node in the dependency graph. No more carefully placing your files in the right folders and hacked-together scripts for adding hashes to file URLs—webpack can take care of it for you.

Rollup was created for a different reason: to build flat distributables of JavaScript libraries as efficiently as possible, taking advantage of the ingenious design of ES2015 modules. Other module bundlers—webpack included—work by wrapping each module in a function, putting them in a bundle with a browser-friendly implementation of `require`, and evaluating them one-by-one. That's great if you need things like on-demand loading, but otherwise it's a bit of a waste, and it gets worse if you have lots of modules.

ES2015 modules enable a different approach, which Rollup uses. All your code is put in the same place and evaluates in one go, resulting in leaner, simpler code that starts up faster. You can see it for yourself with the Rollup REPL.

But there's a trade-off: code-splitting is a much hairier problem, and at the time of writing Rollup doesn't support it. Similarly, Rollup doesn't do hot module replacement (HMR). And perhaps the biggest pain point for people coming to Rollup—while it handles most CommonJS files (via a plugin), some things just don't translate to ES2015, whereas webpack handles everything you throw at it with aplomb.

So which should I use?

By now, hopefully it's clear why both tools coexist and support each other—they serve different purposes. The tl;dr is this:

> *Use webpack for apps, and Rollup for libraries*

That's not a hard and fast rule—lots of sites and apps are built with Rollup, and lots of libraries are built with webpack. But it's a good rule of thumb.

If you need code-splitting, or you have lots of static assets, or you're building something with lots of CommonJS dependencies, Webpack is a better choice. If your codebase is ES2015 modules and you're making something to be used by other people, you probably want Rollup.

Package authors: use pkg.module!

For a long time, using JavaScript libraries was a bit of a crapshoot, because you and the library author effectively had to agree on a module system. If you were using Browserify but she preferred AMD, you would have to duct tape things together before you could actually build anything. The Universal Module Definition (UMD) format *sort of* fixed that, but because it wasn't enforced anywhere you never knew quite what you were going to get.

ES2015 changes all that, because `import` and `export` are part of the language. In the future, there'll be no ambiguity, and things will work a lot more seamlessly. Unfortunately, because browsers (mostly) and Node don't yet support `import` and `export`, we still need to ship UMD files (or CommonJS, if you're building something Node-only).

By adding a `"module": "dist/my-library.es.js"` entry to your library's package.json file (aka `pkg.module`), it's possible to serve UMD and ES2015 at the same time, right now. **That's important because Webpack and Rollup can both use `pkg.module` to generate the most efficient code possible**—in some cases, they can even both tree-shake unused parts of your library away.

Learn more about `pkg.module` on the Rollup wiki.

Hopefully this article makes the relationship between the two projects a bit clearer. If you still have questions, find us on Twitter at rich_harris/rollupjs and thelarkinn. Happy bundling!

Our thanks to Rich Harris for writing this article. We believe that collaboration in open source is incredibly vital to ensure we push technology and the web forward together.

No time to help contribute? Want to give back in other ways? Become a Backer or Sponsor to webpack by donating to our open collective. Open Collective not only helps support the Core Team, but also supports contributors who have spent significant time improving our organization on their free time! ❤

JavaScript Webpack Rollup Es2015 React