DO NOT EDIT - COPIED TO GITHUB

Node.js Modules Team Meeting 2020-05-06

Links

* **Recording**:
* **GitHub Issue**: $GITHUB_ISSUE$
* **Minutes Google Doc**: $MINUTES_DOC$

Present

* Modules team: @nodejs/modules
* Jordan Harband (@ljharb)
* Geoffrey Booth (@GeoffreyBooth)
* Rob Palmer (@robpalme)
* Michael Zasso (@targos)
* Bradley Farias (@bmeck)
* Corey Farrell (@coreyfarrell)
* Wesley Wigham (@weswigham)
* Jan Krems (@jankrems)

Agenda

Announcements

*Extracted from **modules-agenda** labelled issues and pull requests from the **nodejs org** prior to the meeting.

nodejs/node

* cli: --dev flag for development exports resolution condition
[#33171](https://github.com/nodejs/node/pull/33171)

GB: It's incredibly useful for bundlers. Standard pattern checking process.env. Two things happen: determining the graph + dead code elimination. Benefit of graph level is that side-effects of the whole graph get triggers. Most dependencies are not pure. So bundlers struggle to remove dev code. You can "await import()" and import dev dependencies - top-level await enables this. Two problems: 1. we don't have to-level await. 2. no bundlers support this yet. Doing this at the graph level has lots of benefits. Just have two builds you switch between. If you want to point to a development build. webpack (Tobias) are interested in this - not sure

about Rollup. Do we put this in Node to set a precedent for bundlers? If it goes in Node, how should it be enabled? Right now it's a flag per-run. It triggers a "development" condition in exports. We're struggling with reflecting this information in a runtime accessible variable. Hard to get consensus in Node. Motivation: trying to be adoptable universally. We don't have a pattern for universal environment variables apart from globals - but adding a global is not an option. No one will allow us unless we put it in process. We could add it later. Happy to land this?
WW: Node's conditional exports is not the first runtime to have this. React Native has it. It is terrible from a tooling perspective. So I can only recommend conditions as a last resort. I recommend we recommend people only use conditional exports as a last resort.
GB: But you want widespread bundler support for conditional exports?
WW: Yes.
BF: .... having tooling needed to parse JS control flow graphs is simpler than any alternative.
JH: Feels weird to put this feature in. Currently the conditional exports switch between module systems. Switching between platforms. But dev/production is like a matrix applied to both. So it doesn't seem right to conflate them in the same object. Seems overly prescriptive. It's unfortunate React behaves differently in production and development. Would prefer a world where they don't need multiples. Would like to remove the need to pivot based on the environment.
WW: On React, the next version to swap dev vs prod, they emit entirely different things. So no need for runtime switching.
GB: This is highly specific to React. Let's not think about this as one solution. Runtime and export switching are different. Some publishers can highly optimize their build. The common catch is that they don't realize they need to set the environment and they can get two copies. If you're a team like React, maximising optimization with Closure compiler etc is going to get you best performance - conditional exports is tailored to this.
JK: I don't see the dev build as something that allows more optimized prod builds. It doesn't preclude you maintaining your current package with no debug features. It's very non-standard to add debug features. This would give a clear way to do so. No risk of getting blamed. Can packages become more helpful because of this feature?
BF: Let's consider alternatives to see if this is a good direction. Currently we have a kind of standard way to do this using process.env.NODE_ENV. Some bundlers support it. But it's a pain to parse. Even if you use it you end up with a similar problem of not wanting two builds. You have code that's only executed in one mode. So with React Devtools you expose things you would never want used in production. These aren't merely for production builds. We have things we want when we are the app runner. But we also run other people's code. Running with a dev flag seems like a boon to me. You kind of have it with if/else but may not work with tooling. No clear alternative apart from saying people shouldn't want this. It would be helpful if people suggest alternatives.
WW: Yes, we are just moving the branch from explicit in code to implicitly in a loader. There is a problem that makes this unsafe: exports only affect external references, but not to internal references. So unless you have independent trees (dev and prod folders) you're liable to call back across entrypoints. Self-referential would solve this.

BF: Disagree with implicit vs explicit. It's always explicit.
GB: From Tobias, seems like he will implement this in webpack. The fact that proposing this triggered interest suggests strong demand. We don't control whether it exists. We can control where it exists.
no conclusion - let's return to it in a month

* \[v12.x\] Backport unflag --experimental-modules
[#33055](https://github.com/nodejs/node/pull/33055)

MB: We have discussed unflagging on semver minor on 12. It got delayed a bit. I think we're waiting on consensus on tiny bits. 1. Are we cool with backporting removal of the flag? tsc needs to agree. 2. Do we backport with or without the warning? I have spoken to tsc. I think we should backport removal of the flag, but I think we should keep the "experimental modules" warning until 14 goes LTS in late October. Reason: removing the warning on 14 - that's current not yet LTS - people using it know the warnings. I don't think that same signal of support and adoption should be given to LTS.
JH: I was taken aback by the PR. Clarification: anyone using require() will see no warnings?
MB: Yes.
JH: So only import() and static import would trigger the warning. I would prefer no warning at all because it hinders adoption.
MB: That's the point - we don't want to give too strong of a signal. There are rough edges. I think tsc would pushback on removal of the warning.
CF: My only concern is that some packages might be using import() to detect support for ESM. So unflagging with a warning might trigger warnings based on this detection.
MB: Those packages wouldn't work today?
CF: Correct. Today it fails or doesn't. This proposal makes it pass but side-effects is a warning.
JH: So they are try-catching eval'd code?
CF: Yes. I may be doing this in NYC.
GBB: We already had this with 13/14.
JH: That's kinda what I was concerned about. The point of the warning is to give actionable feedback. In top-level code that's fine. In code where you try to use a package, and you get it that's fine. But transitive dependencies triggering this trains you to ignore warnings. Removing the flag is a much stronger signal than removing the warning.
MB: Removing flag makes it easier to adopt. Removing the warning makes it easy to adopt en-masse.
CF: I double-checked NYC. I have a separate load-esm/cjs which does import() - only loaded if someone attempts loading mjs. So no warning would be produced unless someone was actually using ESM. I have seen the suggestions "why don't you use import() and fallback on require() if that fails?"
LH: I have a package to do feature detection of exports in Node. It doesn't tell you about ESM.
MB: I looked at self-referential, you can try-catch it. I think this is the right way to sniff it.

Richard Lau brought up: anyone not opting into ESM but have dependencies that will act differently on unflagging. So there's a question about whether flag removal is semver major.
BF: Did we remove the warning for conditional exports in 12?
MB: We removed them. I believe "exports" work. But conditional exports and self-referential are still behind a flag.
BF: My feeling is that without conditional exports, I don't view it as breaking.
LH: Exports field does not work at all in Node 12. In spirit, people should write "main" and "exports" having the same API. It's possible to break. I hope no one wrote a package like this.
BF: I have not heard of anyone doing that so far.
LH: I wrote es-get-iterator. It does terrible things in old browsers to get an iterator interface. I used conditional exports to guide Node versions. I could have used fancy syntax because I knew which version of Node it would be run in.
GB: This issue is something we had a long session with James Snell on. We discussed if it would be semver major. He thought it would be fine to land as a minor. There are >500 packages using "exports" now. I have a list. We could test them against unflagged Node 12.
MB: The only packages that would have behavioural changes are those where the author has opted in. CJS reached via exports would continue to have the same interface. A benefit is that broken packages should be able to be fixed in a patch release. My gut (famous last words) is that it's not going to be a significant undertaking to fix this. These shouldn't be deep in module graphs. Adopters are likely to have a faster cadence at dealing with this. Anyone want to block landing this in 12 with the warning intact? Caveat: aiming to remove warning in October unless ecosystem outcry or technical issues.
silence => consensus!

nodejs/modules

* Rediscussion for out-of-order CJS exec?
[#509](https://github.com/nodejs/modules/issues/509)

MB: TLDR: today importing CJS is supported but not with named exports. Is the problem of named exports from CJS something we want to solve? Reducing delta between faux modules and node modules. Solutions: stack analysis, double execution. First, is this a problem space we want to dig into. We seemed to have consensus on this. I don't think dynamic modules has legs. Do we want to revisit this problem space?
GBB: Yes. Users are complaining. It's a big pain point. We need to show effort in saying why we can't do this. We need to try the best solution we have thought of. Then we have evidence.
BF: We're seeing people have a problem transitioning their modules. I want to allow easier transition for some. I don't think we can reach 100% coverage. I am very against re-ordered evaluation - weird side effects and no tools do this today.

JK: I slightly disagree with "we need to write code" because I think we understand how well the approaches will work. The objections have rarely been "hey this wouldn't work", it's more that people disagree to the degree with which they will work - "we don't like heuristics". Can't be resolved by writing code - it becomes less convincing. I strongly agree we should decide on how much we care about the level of support. No level will match today's behaviour in CJS/faux-modules. module.exports = require("other") will always say there is one export if you analyze one file. Is it really a big deal to add a wrapper mjs? So we know the trade-offs. We just have to make a decision.
MB: I started a PR https://github.com/nodejs/node/pull/33256 when a named import fails it checks to see if the module is CJS, and if so gives a better warning including copyable code to replace your named import. The RegExps need more testing. Another approach is improving docs and error - what broke, why it broke. I lean towards Jan. I want to improve the experience. But the clear cut "you have a default" is easy to teach, easy to understand, does not require static/dynamic understanding. It's cleaner.
JK: Parsing is viable, but not strong enough for me to implement it.
GB: I lean towards wanting to implement this. I thought we would get away with just default. But tooling will still permit named exports. So friction will continue. So it's worthwhile to mitigate this pain.Maybe we can only do it at the boundary of ESM->CJS - overhead could be <1ms per module. I would like to ask if we have consensus to do it.
MB: Let's kick this back to the issue to discuss consensus - no time now on this call.
GB: If I put in dev time I would like to know if there would be objections.
JK: People on the issue tracker made strong objections.
GBB: Anyone interested in working on this?
JK: My concern is exports. By definition we have a collision. If we want to be compatible with faux modules - they will treat it as a default.
MB: I suggest closing the current issue - it's too long and off-topic. Create a new issue on named exports as an RFC for what you plan to implement so we can respect your time.
GB: Let's return to this in the next meeting. Node 12 support is time critical.
MB: We need to decide if this is needed before unflagging in Node 12.
Conclusion: Guy will write an RFC as a new issue

* Chartering the Modules team [#412](https://github.com/nodejs/modules/issues/412)