Type-Stripping 2025-02-20

Adhoc Work Group Session held post-TC39 in Seattle

Mark does not know the history of Type Annotations, but believes it got stalled.

MM: Maybe the attempt was to not privilege TypeScript to allow other grammars, enabling neutrality.

MM: I didn't see a way to solve that problem.

MM: I have seen ever-growing adoption of erasure-driven types. Seeing three compatible versions. It's the subset of TS that can be meaningfully converted ONLY by erasure. Meaning replacing tokens with whitespace, modulo the edge case of needing semi-colon injection sometimes.

MM: Would like feedback from Facebook if we were deprioritizing ... something... the reason for us to standardize it in TC39 is because JS and TS have to coordinate to avoid collisions. There's only one grammar. Could lead to disaster. Syntax is not modular. So TC39 helps coordinate this.

MM: Separately: TS grammatically is not purely a superset of JS. The is <> ambiguity in expressions. So you need the erasure-only TS to be marked as TS instead of being JS. Would love to just standardize one language, but it would require addressing this superset problem.

CM: Is there a doc that describes the subset of TS that matches this?

ACE: I created one of these implementations. And I treat it as the canonical one. It's written in JS. Attempts to be simple, not fast. Name: ts-blank-space. If you search for it you will get a lot of Taylor Swift.

mark laughs loudly

ACE: On npm searching for ts-blank-space does well.

MM: There's constructor parameters and namespaces.

JGT: My understanding is these are discouraged.

BLY: There is a new TS option to ban these from your projects.

JGT: It's clear TS considers them to be a mistake. By standardizing the erasure, are we limiting the design space of TS? Right now, whatever tools do the erasure have to catchup with TS evolution.

ACE: Defining the erasable grammar in JS would massively limit TS evolution. Or token soup can be used to permit arbitrary syntax within balanced brackets. But this also allows anything inside there.

MM: We only need to define the grammar rules on erasure. The result would be that any cover grammar acceptable to TS that widens erasure rules is something we would be jointly happy with. Anything not covered by a cover grammar would be subject to negotiation. No one is here from TS. My guess is they would prefer the consequences of this becoming a standard part of JS.

RPR: Daniel from TS has presented on upcoming TS grammar proposals.

SYG: Current V8 position. We are not against standardizing the syntax. But we are against accepting it at runtime. Because it's work for no gain. If runtime erasure were mandatory in the proposal, we would be against it.

MM: Why not just standardize the erasure rules, but not force any runtime change? Seems very natural that the erasure would be done by tools. I had not considered that it would be done in the engine.

SYG: Currently we have one language. As Dan said. So anything defined in the spec is expected to go into the engine.

MM: So integration into the TC39 process requires distinguising the two languages.

RPR: There is an issue on the Type Annotations proposal suggesting use of :: prefix on type arguments, e.g. func::<arg>(args)

MM: I'd be happy defining the two separate levels of the language, to avoid engines bearing burden.

ACE: I have concerns with JS Sugar, but not for Type Annotations because it's only erasure. It makes it trivial to see how it works transparently. You have a tool that strips. No need for sourcemaps. As opposed to Sugar that has runtime semantics. Right now it's easy to put a transform before V8. A web server could handle this prior to serving.

SYG: This seems like a change in motivation.

RPR: I am a champion, and only speak for myself. There were two motivations. DX and language coordination. I personally think that the coordination goal stands alone and is highly valuable.

MM: It's good for this to be coordinated by one committee. By seeing both levels as blessed by TC39, it would help grow adoption of erasure-only TS.

SYG: I thought the ecosystem already had a preference for erasable only TS?

RPR: The ecosystem is confused. Language-aware folk, NodeConfEU attendees, prefer erasable TS. But most developers are not aware of the distinction between full TS and erasable TS.

MH: What about Node? They support this but ban it in node_modules.

ACE: This is defensive to protect the performance of VS code. It works better with DTS.

MH: So the recommendation is that you should always publish DTS.

JGT: It's unclear which use-cases are supported and which ones are not. Directly loading is in Node. It would be helpful to see a table of in and out.

CM: Anyone here from Node? That ought to be quite practical. So banning from node_modules is not a commitment.

ACE: TS asked Node for this policy restriction.

JGT: It's very convenient to not have a build step. Maybe the answer is that this provides the ecosystem with type-stripping in node_modules?

ACE: If this went into the spec, it would show very good faith on coordination. This formalizes that cooperative mode. There would be carve-outs for where things could be added without TC39 blessing.

SYG: Sounds more like a question for TS.

ACE: Yes, this only works with cooperation.

MM: Only works if TS judge the benefits are worth the cost.

SYG: One concern before was that because of the angle-brackets, we will end up standardizing a third-thing that the ecosystem would need to move.

RPR: That is unresolved.

ACE: Only becomes an option if TS leans into and assists a transition. Like sloppy vs strict.

SYG: TS is in the best position to nudge. Seems risky. We may end up with three things.

MM: Like the XKCD 14 standards.

ACE: It seems like only Turbofish remains.

MM: I would be happy if this thing went forwards with two levels of language, with engines only being obligated to implement the lower level.