Design Doc: Use JavaScript instead of TypeScript for internal Deno Code

Update June 10 2020: I saw that this design doc was being discussed more widely. Most people don't have the context to understand this narrow technical document - it is only applicable to a very particular, very technical situation in the internals of Deno. This is not at all a reflection on the usefulness of TypeScript in general. It's not a discussion about any publicly visible interface in Deno. Deno, of course, will support TypeScript forever. A website or server written in TypeScript is a very very different type of program than Deno - maybe much more so than novice programmers can appreciate - little of Deno is written in TypeScript. The target audience is the 5 to 10 people who work on this particular internal system. Please don't draw any broader conclusions.

Update July 21, 2020: Completed in https://github.com/denoland/deno/pull/6793

Problem

- Incremental compile time when changing files in cli/js takes minutes. This is crushingly slow and painful to modify.

- The typescript organization/structure that we're using in cli/js is creating runtime performance problems. As an example, we recently realized that we're unable to to get TS to generate a class with the name "Header" because it shadows the declaration in our d.ts file. So instead we name the class "HeaderImpl" and assign it to "window.Header". But that creates the problem that "Header.name" has the wrong value. So we're forced to add unnecessary runtime code

  ```
  Object.defineProperty(HeaderImpl, "name", { value: "Header" });
  ```

  to fix "Header.name". Who knows if this kicks Header out of some optimization path in V8. The optimal thing is to generate "class Header { ... }" anything less suggests a fundamental flaw in the design.

- TypeScript is supposed to be helping us organize code, but one could claim it has the opposite effect. For example, we have two independent Body classes https://github.com/denoland/deno/issues/4748. This is difficult to have visibility into because of the complexity involved with generating the runtime code.
  Ideally we would have a system where two Body classes would be obviously wrong.

- Our internal code and runtime TS declarations must already be manually kept in sync. TSC isn't helping us to generate the d.ts files - it was too much overhead and complexity when we attempted it before.

- We have two independent TS compiler hosts: one for the internal Deno code in //deno_typescript/compiler_main.js and another other for external user code in //cli/js/compiler/. These two hosts have similar goals - they seek to modify TSC to operate on Deno-style imports (e.g. with file extensions). The //deno_typescript is used at build-time while the //cli/js/compiler/ is used at runtime.

Deno should be focused on providing its users with a performant and safe runtime environment. The organization of Deno's internal code should never come at a cost to the product it delivers. Currently we are attempting to "self host", in that we organize our internal TS code just like Deno user code would look. This is a nice thing, but it's not necessary. It's becoming more and more obvious that our attempts to "self host" are hindering the performance of user code and the maintainability of the runtime.

Solution

A radical solution is to remove all build-time TS type checking and bundling for internal code. We would move ALL runtime code into a single big file cli/rt.js. This file would be dropped directly into V8 at build-time to create the snapshot. The corresponding cli/rt.d.ts would contain the type definitions and documentation.

In this setup it will be possible to write code like `class Header { }` and clearly understand what is provided and executed in the runtime.

At first glance the idea of having a 10k line javascript file sounds unappealing. But I think compared to the current system, this will allow new contributors to quickly understand how Deno bootstraps itself.

Build-time complexity and performance is obviously better with this setup. Of course we will still need to do a snapshot, but we will not have to build a special TSIsolate to first execute tsc.

Runtime complexity and performance is strictly better with this setup. It will be very clear if we have multiple definitions of a class. We will be able to see all code that is being executed clearly. We will be able to analyze and step through the code during debugging without worrying about additional source map complexity.

It's important to explicitly note that Deno user code will still be in typescript and type checked. We will have tests to make sure the code works as intended against cli/rt.d.ts

The compiler worker will need its own separate cli/compiler.js file. It does not need a corresponding d.ts. One drawback of this approach is that it will be difficult to share code between the compiler worker and the runtime itself.

Implementation

To make this refactor happen, we can take an incremental approach or a quick approach.

The incremental approach would be to start an empty cli/rt.js file and slowly move code over into it over the course of many commits.

The quick approach might work better: we would simply take the generated bundle CLI_SNAPSHOT.js and COMPILER_SNAPSHOT.js and commit them into the tree. At the same time we would remove all existing TypeScript files from cli/js (excluding the tests). If we took this approach we need to ensure that we don't lose comments - it seems CLI_SNAPSHOT.js has them stripped.

> Ryan: I favor the quick approach, because I think we can get it to go green without too much trouble.
>
> Bartek: I actually favor a slower approach as generated files are wrapped in SystemJS loader which already might pose some overhead on the runtime code. I'd suggest spending an evening or two to try and create rt.js; which would effectively be one giant IIFE that adds globally accessible functions that can bootstrap main runtime and worker runtime.

After this refactor we can remove deno_typescript.

Update:

https://github.com/denoland/deno/pull/4750
https://github.com/denoland/deno/pull/4760

Discussion log (April 15, 2020)

ry
Just an update on the "removing the internal typescript" project. Bartek and I both have done tentative patches to get a feel for how this would work. It seems okay - mainly we're interested in this because we can very carefully manage what runtime code we're shipping - it also makes incremental compile time 2x faster
however, we're not going to just land these patches as is.

having a single file to mange all internal code is workable, but unnecessary. we can split it up into different files without any extra infrastructure - just by loading them individually into V8

Ideally we would use ES modules to structure this internal code - but we've experienced some problems with snapshotting ES modules before - so we need to do more experiments in this area

lucacasonato So typescript for internal code is gone?

ry So - to make a long story short - we're removing the types from internal code and making it pure JS
this reduces complexity and helps us ship a faster product. I acknowledge it's unfortunate to lose the type information, but it's really masking larger problems.
It's not going to happen immediately tho. We have a bit more work to figure out how exactly it should be done.
It's not going to be one big 9000 line file either.
But it might not be ES modules. depending on if we can make that work or not.

lucacasonato So what is the reason behind the removal of types? Just the faster compile? I don't quite understand yet what larger problems typescript is masking - could you elaborate on that?

ry
https://docs.google.com/document/d/1_WvwHl7BXUPmoiSeD8G83JmS8ypsTPqed4Btkqkn_-4/edit
Google Docs
Design Doc: Use JavaScript instead of TypeScript for internal Deno ...
Design Doc: Use JavaScript instead of TypeScript for internal Deno Code Problem Incremental compile time when changing files in cli/js takes minutes. This is crushingly slow and painful to modify. The typescript organization/structure that we're using in cli/js is creating r...

does this answer your question?

lucacasonato 1. 2. I don't get this one. Why don't we just replace the declare interface Headers + declare const Headers with an actual class Headers in the .d.ts? The interface + const is effectively a class. 3. This is just because noone took the time to check if there was an already existing implementation. I don't see how this is related to to TS. The exact same could happen in JS. 4. will get worse with this change 5. could be done by using //cli/js/compiler/ from a previous build(edited)

ry
2. doesn't work

3. yes - good point

4. I don't think so

5. I don't want the build to depend on an existing deno executable

homebrew, for example, does not allow that

nayeemrmn 2. Can we just check and strip types independently of the d.ts or something? Even that would be a massive improvement for dev. It seems like the presented issues shouldn't fundamentally stop us from checking that one local variable is assignable to another, etc.

ry @nayeemrmn i considered stripping the types at build-time and using a unit test for the type check. unfortunately we don't have a good way to strip the types without spinning up TSC
also it doesn't solve the problem that TypeScript is getting in the way of producing the bundle we need
which is the main thing

lucacasonato @nayeemrmn i considered stripping the types at build-time and using a unit test for the type check. unfortunately we don't have a good way to strip the types without spinning up TSC
@ry Can't swc do that?

ry TBH this code is a relatively small part of Deno... there are much more complex things happening. We need it to be small and fast.
@lucacasonato no
actually i think it might but last time i talked to the swc guy - he said it was under construction

lucacasonato Ah ok - because the 1.0 release suggests it can: https://swc-project.github.io/blog/2019/02/08/Introducing-swc-1.0#typescript-support, just no type checking as expected
Introducing swc 1.0 · swc
Introduction

2. doesn't work
@ry @kitsonk seemed to suggest this was possible
https://github.com/denoland/deno/issues/4682#issuecomment-611302431 and he started doing it for some things in dom_types.d.ts. I might be missing something though
GitHub
deno doc: Blob interface is not exposed · Issue #4682 · denoland/de...

> deno doc https://github.com/denoland/deno/releases/download/v0.40.0/lib.deno.d.ts | rg Blob const Blob: { prototype: Blob } The interface associated with that type is not in the documentat...

ry manually managing the d.ts files has been great - it's very much part of the public interface - we need to have 100% control over it
we do not want random compiler-generated __namespace0123 in there
This is the same situation - but for the bundle
It's not good that we do not have visibility nor control over what code is in there.
for example, the TS compiler worker, has 15k lines of code in its bundle - EXCLUDING typescript.js
TS compiler worker is about 500 line thing that interfaces between ops and TSC
this is caused by our attempt to "self host" the internal code
self hosting is cute - but if it's effecting performance and the ability to make improvements - it's an unnecessary feature
https://gist.github.com/ry/a6d4b1466158d82750a6447342fd3af4
^--- here is what we ship for the compiler worker
we call this code every time deno runs

lucacasonato Ok, wow. I hadn't seen this. That changes things...

ry does the compiler worker really need to generate a UUID?
https://gist.github.com/ry/a6d4b1466158d82750a6447342fd3af4#file-compiler_snapshot-js-L9982
(no)
yes, this is fixable within the current system .. but the point is that we're abstracting away the important parts
V8 does not run typescript. It runs JS. We need to have a good handle on the JS we ship
If we don't try to self-host, we don't need a sourcemap for the internal code. There's 1mb off the executable right there.
The point is the internal typescript comes at a cost - and that cost is too high.

lucacasonato So the actual issue is the bundling and opaqueness that that brings with it. It makes sense to stick to pure JS in that case. Its annoying that type checking is gone, but youre right. This is not the kind of code that should be in the runtime.

ry not like the code isn't full of "!" anyway. TS is great, but it gives a false sense of security.

It would be one thing if the internal code was very complicated - like we were implementing parsers/compilers
but it's basically glue code between user code and rust
I've toyed with the idea of removing all internal functionality and only exposing raw ops - requiring all functionality to be done in external code
i think that's a bit extreme tho.

kitsonk Most of it is actually implementing browser standards.

ry yea the bulk of the internal code is going to be streams/events/fetch

lucacasonato Moving away from ts for internal code should also make #2180 significantly easier. (If we don't put everything in one file)(edited)

kitsonk Thinking about it over night my time, my feelings are 1. We get the ability to snapshot modules. 2. We create a type stripper with swc, it does it already. 3. We have a minimal type checker that is an optional part of the build toolchain. 4. We figure out a way to validate against the runtime types.

lucacasonato I think that the problems @ry mentioned could be solved by just not doing any form of bundling for internal code - so a ts file gets stripped and the output js file is used as is by V8. No bundling. That way you get direct insight into what the runtime looks like - so no weird namespaces - without having to leave typescripts typechecking behind. Yeah typescript isn't always great, but in most cases it does help.

kitsonk That means we would be authoring ES Modules and running ES Modules, just with types which get stripped out, so the control of the runtime code is far more explicit. If we did anything, I think the lack of modules is my biggest concern. It is insane to have to maintain a 10k file and try to not leak all sorts of internal guts. That is like going back 20 years.

ry we're not going to do the literal bundle file
(but i also think it would be fine if we had to)
we can have a directory of scripts which are loaded into V8 directly
(note that there's no source map necessary to get good stack traces - V8 knows the name of each file)

kitsonk I get that, but we hide a lot of internals in the scope of a module. It would mean a lot of work to restructure that.A load of script files is just as bad. Collisions, hoisting. There are reasons we have modules.

ry This is a prototype - but something like this
https://github.com/ry/deno/tree/f4bc35e3588a9f689ca6c174db566b2ae0b78b52/cli/js/rt
It would be great if we could do ES modules. We're going to do some research on that.

we have to remember - this is not a website - this is not end user code(edited)
this is part of a runtime executable
sometimes in the engine room you have to wear a hard hat