The Path to Native TypeScript

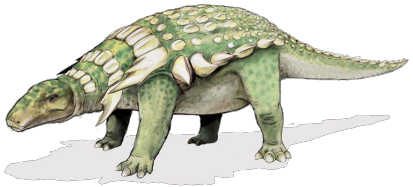
Marco Ippolito

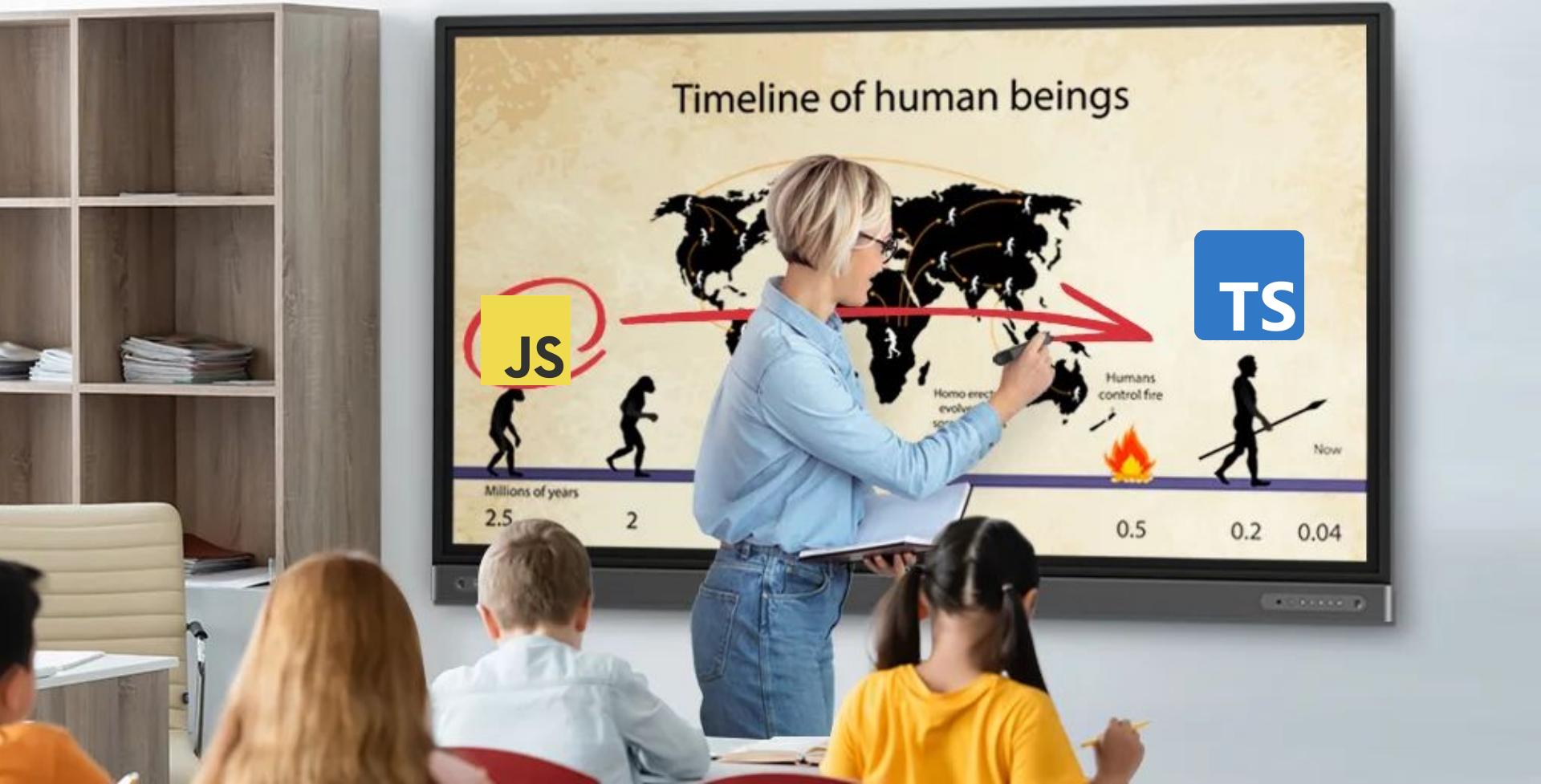
Marco Ippolito

Senior Security Engineer @HeroDevs

Node.js TSC & Releaser

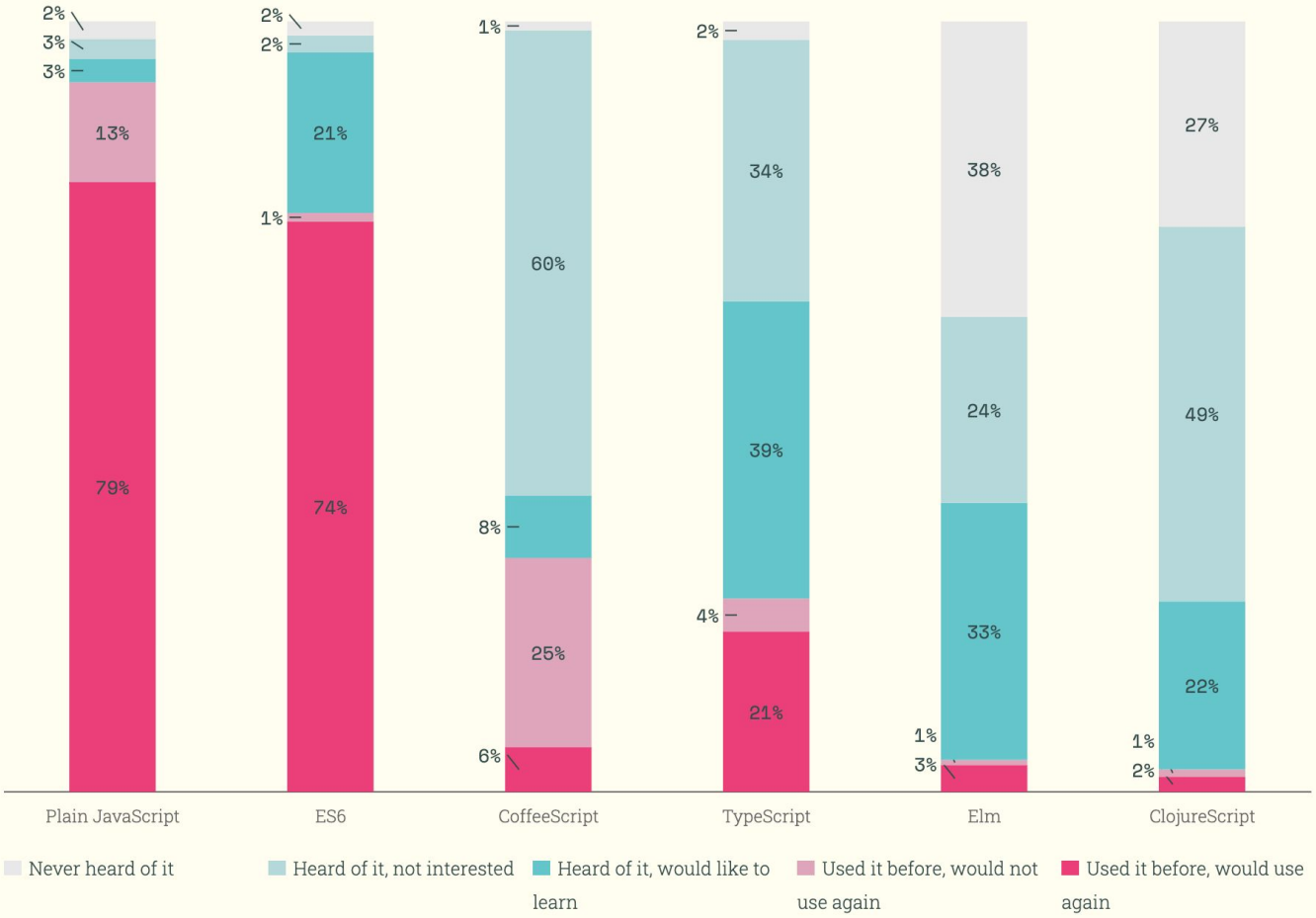
@satanacchio




Timeline of human beings
JS
TS
Humans control fire
Now
Millions of years
2.5
2
0.5
0.2
0.04

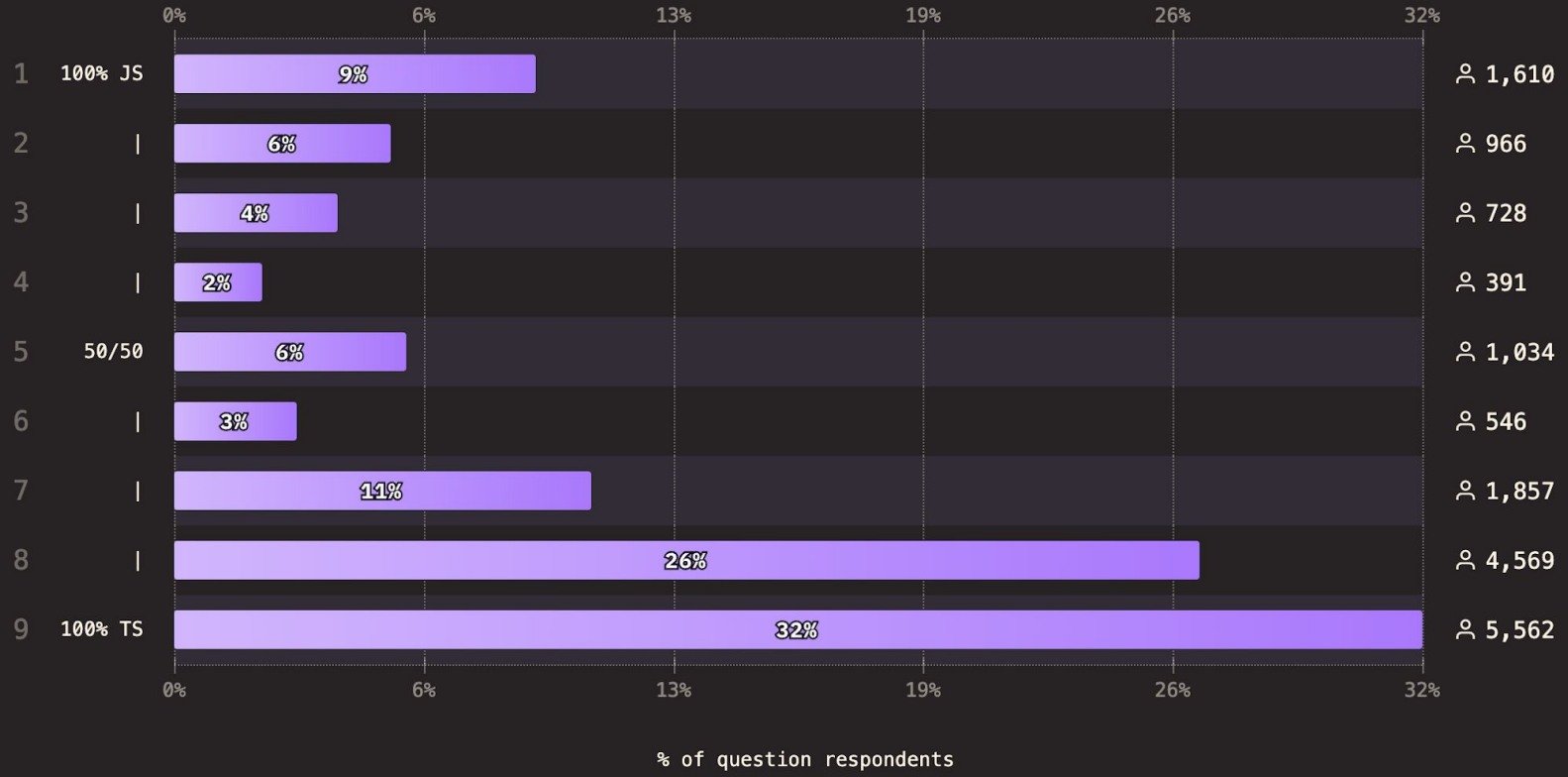
Filter: All | Interest | Satisfaction Show: Percents | Numbers

State of JS 2016



JavaScript/TypeScript Balance

State of JS 2023



17,263 % 73% 5.7 7

Share Chart… Export Data…

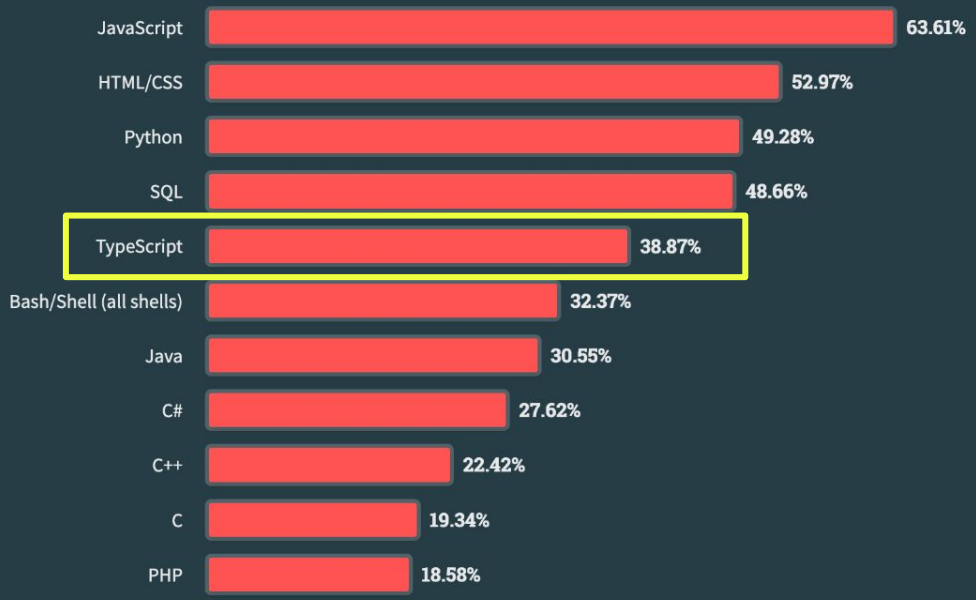
Most popular technologies

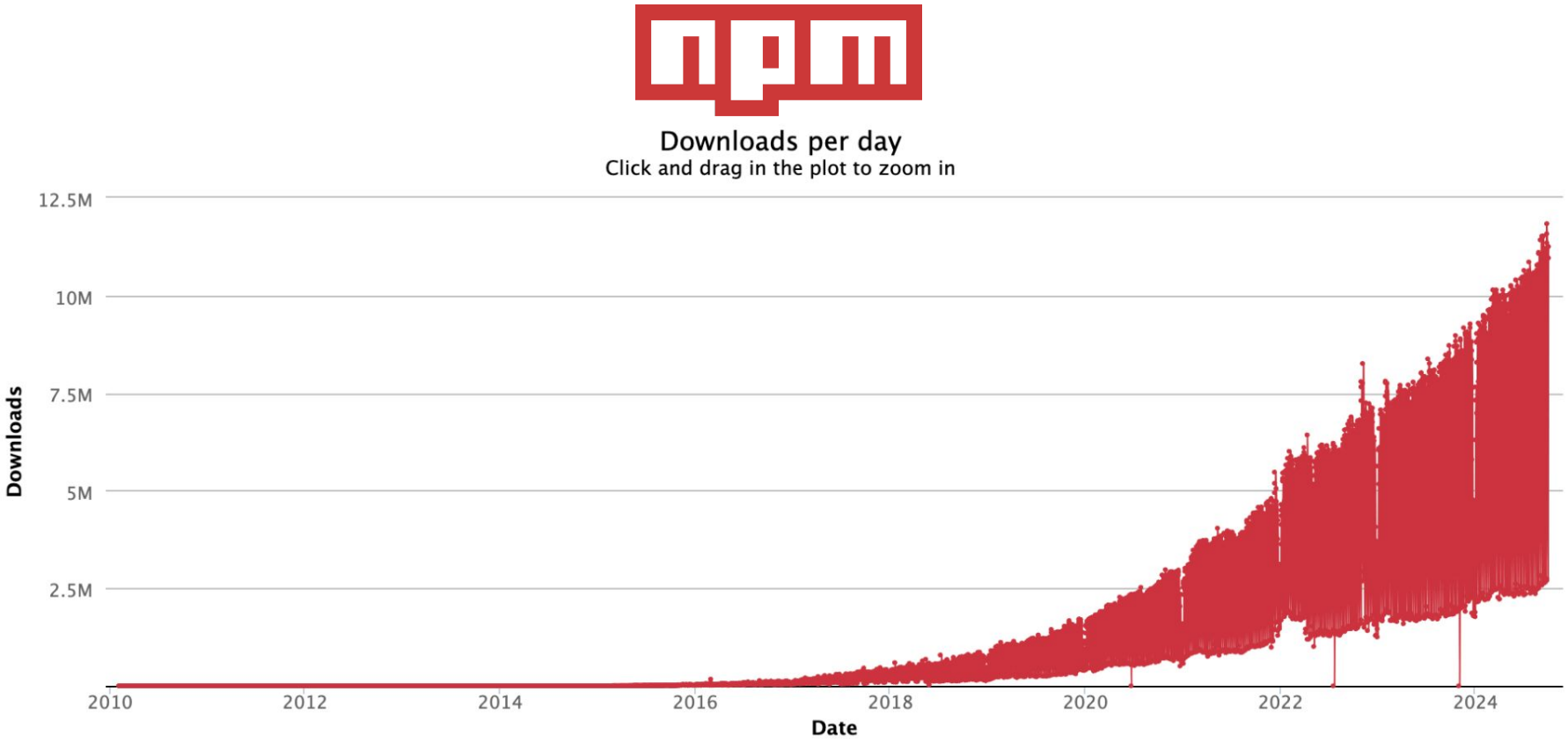
Stack Overflow 2023

This year, we're comparing the popular technologies across three different groups: All respondents, Professional Developers, and those that are learning to code.

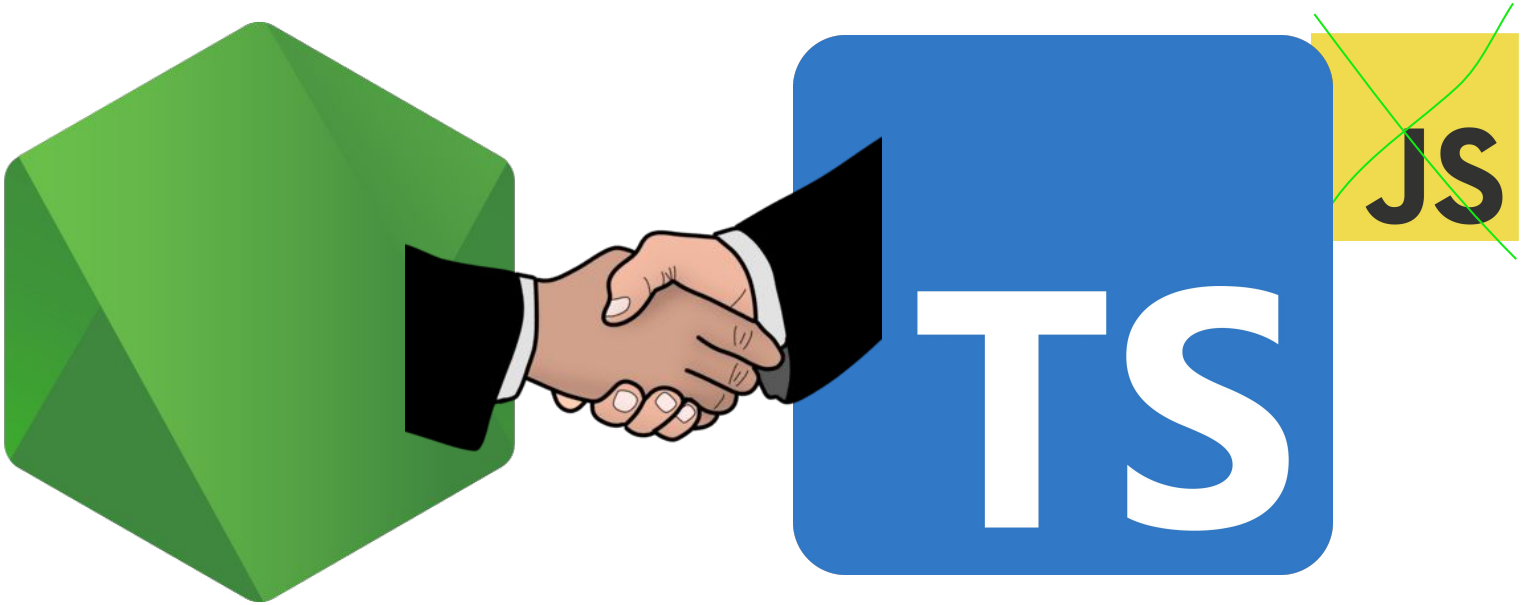
All Respondents | Professional Developers | Learning to Code | Other Coders

87,585 responses




Downloads per day
Click and drag in the plot to zoom in
12.5M
10M
7.5M
5M
2.5M
Downloads
2010
2012
2014
2016
2018
2020
2022
2024
Date

TYPESCRIPT SUPPORT IN NODE.JS



TypeScript Support #43816

Closed avin-kavish

avin-kavish com

What is the p

With TypeScript
proposition to u

Currently, users
process, progra
doesn't use the
speed boost.

It's more ergonc

What is the f

About native support for typescript!! #24101

Closed **ahmedkamalio** opened this issue on Nov 5, 2018 · 14 comments

How could we support typescript without vendoring it? #43818

Open **mcollina** opened this issue on Jul 13, 2022 · 95 comments

mcollina commented on Jul 13, 2022 · edited — Member

I would like Node.js to support the following user experience

```
$ node script.ts
Typescript support is missing, install it with:
npm i --location=global typescript-node-core
$ npm i --location=global typescript-node-core
...
$ node script.ts
"Hello World"
```

(I picked the typescript-node-core name at random, I would be extremely happy if that could be ts-node)

Note that `script.ts` could evaluate to either cjs or mjs depending on the `tsconfig.json` file.

Originally posted by @mcollina in #43408 (comment)

`ode main.ts`
`es6` and `es7` features not supported, I also know that it's
`an be done with` `node` and it's going to be great feature of

typescript like `node foo.ts` in

ent

support for

Closed insinfo

Types/TypeScript - Follow on mini-summit - July 28 at 10:30 ET #149

Closed **mhdawson** opened this issue on Jul 21, 2022 · 15 comments

insinfo commented on Oct 27, 2021

support for native typescript like deno

2

just for convinient

What alternatives have you considered?

No response

1

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Past attempts died on the table

“Node.ts” nodejs/loaders#164 was present in Bilbao in September 2023 openjs-foundation/summit#368 to positive reception and multi-collaborator buy-in, but it eventually bleed out nodejs/node#49704.

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Challenges

- Very hard to get consensus
- Different stability guarantees between Node and TS
- tsconfig
- Pick a tool that works out of the box
- cjs/esm interop
- source-maps
- No champion

Geoffrey Booth Jun 24th at 9:06 PM

(i don't want to post this too publicly): along the lines of adding sqlite, is there a native library we could include that handles stripping typescript types from javascript? so not typechecking, not converting module formats, maybe not supporting the few transforms that typescript has such as enums and legacy decorators, maybe not supporting reading a `tsconfig.json` file: really just stripping the types. i feel like there are low-level libraries for this like that esbuild uses, some rust-based tools use; could any of them be added to our codebase? (edited)

35 replies

Marco Ippolito Jun 24th at 9:10 PM

AFAIK Deno uses SWC

Geoffrey Booth Jun 24th at 9:11 PM

which is in rust, i think? how feasible would it be to add a rust library to our codebase?

Marco Ippolito Jun 24th at 9:17 PM

I read that now Rust has moved Solaris into Tier 2 so I think most of our platforms/architecture (need to check) are supported, the issue might be to add another toolchain to our build

Geoffrey Booth Jun 24th at 9:25 PM

we would need to build swc ourselves rather than including some pre-built c++ binary? i'm assuming yes, because of the variety of platforms?

👍 1

Marco Ippolito Jun 24th at 9:26 PM

Unless there is a wasm version 🙂

Geoffrey Booth Jun 24th at 9:26 PM

there is, actually: https://swc.rs/docs/usage/wasm

Type stripping

- No type checking
- No tsconfig
- Remove inline types
- No source maps
- Stable across TS versions

Type stripping

- No type checking
- No tsconfig
- Remove inline types
- No source maps
- Stable across TS versions

Type stripping

- No type checking
- No tsconfig
- Remove inline types
- No source maps
- Stable across TS versions

Type stripping

- No type checking
- No tsconfig
- Remove inline types
- No source maps
- Stable across TS versions

Type stripping

- No type checking
- No tsconfig
- Remove inline types
- No source maps
- Stable across TS versions

Type stripping


```ts
// .ts

interface LabeledValue {
  label: string;
}

function printLabel(labeledObj: LabeledValue) {
  console.log(labeledObj.label);
}
```

Type stripping


```ts
// .ts

interface LabeledValue {
  label: string;
}

function printLabel(labeledObj: LabeledValue) {
  console.log(labeledObj.label);
}
```

Type stripping

```ts
// .ts
```


```ts

function printLabel(labeledObj: LabeledValue) {
  console.log(labeledObj.label);
}
```

Type stripping

```ts
// .ts

function printLabel(labeledObj: LabeledValue) {
  console.log(labeledObj.label);
}
```

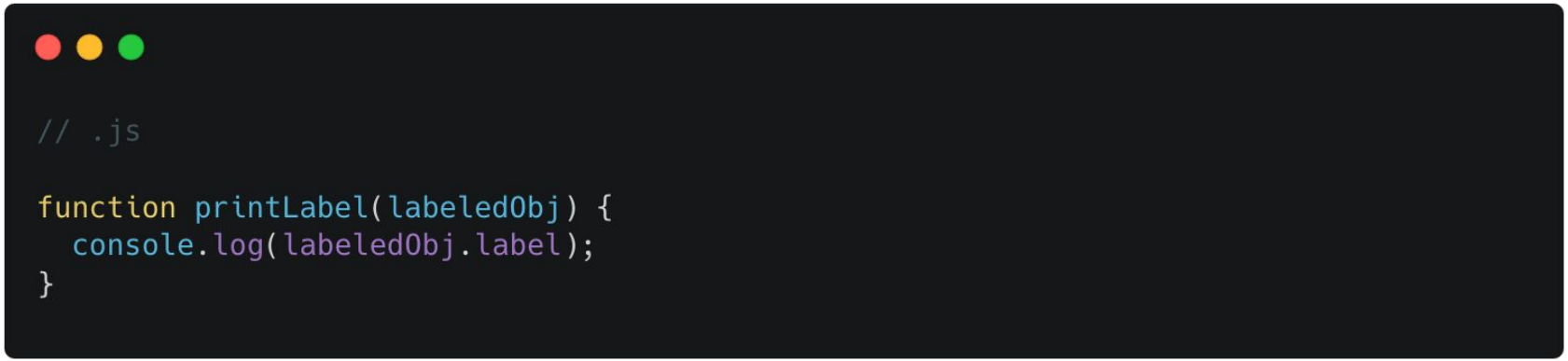
Type stripping

```ts
// .ts

function printLabel(labeledObj
  console.log(labeledObj.label);
}
```


```ts
) {
```

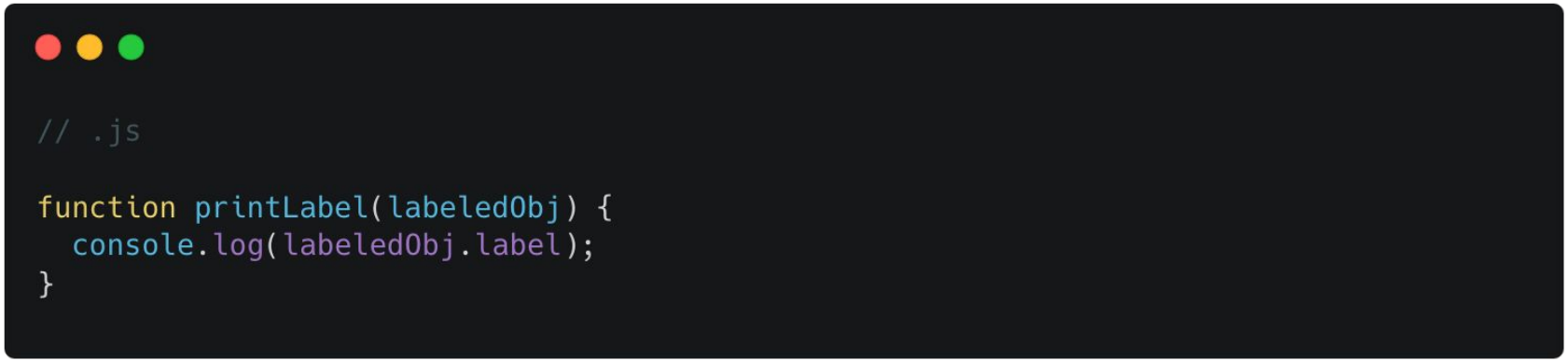
Type stripping



```
// .js

function printLabel(labeledObj) {
  console.log(labeledObj.label);
}
```

Type stripping



```
// .js

function printLabel(labeledObj) {
  console.log(labeledObj.label);
}
```

Can we try to preserve locations?

acutmore commented on Jul 4

TypeScript type-stripping is something I have been actively working on at work (for internal purposes, not for landing in Node) so I'd like to just pass on some of the knowledge I gained from that.

It might be useful in both the implementation and in particular to make the validation easier. To be clear, this is just for passing on info that might be helpful to the folk doing the work here.

https://gist.github.com/acutmore/27444a9dbfa515a10b25f0d4707b5ea2

marco-ippolito commented on Jul 4 • edited

Member Author

> TypeScript type-stripping is something I have been actively working on at work (for internal purposes, not for landing in Node) so I'd like to just pass on some of the knowledge I gained from that.
>
> It might be useful in both the implementation and in particular to make the validation easier. To be clear, this is just for passing on info that might be helpful to the folk doing the work here.
>
> https://gist.github.com/acutmore/27444a9dbfa515a10b25f0d4707b5ea2

This is gold, thank you.
I think white spacing is a very good alternative to source map that could work for us **@kdy1** wdyt?
Source mapping since its expensive behind a flag, white spacing by default so stack traces are in the right place

Blank space

```
// .js



function printLabel(labeledObj              ) {
  console.log(labeledObj.label);
}
```

@swc/wasm-typescript

- Wasm
- Battle-tested
- Type stripping support
- Fast
- Support from maintainers

Install

```
> npm i @swc/wasm-typescript
```

Repository

github.com/swc-project/swc

Homepage

github.com/swc-project/swc#readme

Weekly Downloads

204

Version	License
1.7.35	Apache-2.0

Unpacked Size	Total Files
3.56 MB	5

Issues	Pull Requests
350	26

Last publish

5 days ago

@swc/wasm-typescript

- Wasm
- Battle-tested
- Type stripping support
- Fast
- Support from maintainers

Install

```
> npm i @swc/wasm-typescript
```

Repository

github.com/swc-project/swc

Homepage

github.com/swc-project/swc#readme

Weekly Downloads

204

Version	License
1.7.35	Apache-2.0

Unpacked Size	Total Files
3.56 MB	5

Issues	Pull Requests
350	26

Last publish

5 days ago

@swc/wasm-typescript

- Wasm
- Battle-tested
- Type stripping support
- Fast
- Support from maintainers

Install

```
> npm i @swc/wasm-typescript
```

Repository

github.com/swc-project/swc

Homepage

github.com/swc-project/swc#readme

Weekly Downloads

204

Version	License
1.7.35	Apache-2.0

Unpacked Size	Total Files
3.56 MB	5

Issues	Pull Requests
350	26

Last publish

5 days ago

@swc/wasm-typescript

- Wasm
- Battle-tested
- Type stripping support
- Fast
- Support from maintainers

Install

```
> npm i @swc/wasm-typescript
```

Repository

github.com/swc-project/swc

Homepage

github.com/swc-project/swc#readme

Weekly Downloads

204

Version	License
1.7.35	Apache-2.0

Unpacked Size	Total Files
3.56 MB	5

Issues	Pull Requests
350	26

Last publish

5 days ago

@swc/wasm-typescript

- Wasm
- Battle-tested
- Type stripping support
- Fast
- Support from maintainers 

Install

```
> npm i @swc/wasm-typescript
```

Repository

github.com/swc-project/swc

Homepage

github.com/swc-project/swc#readme

Weekly Downloads

204

Version	License
1.7.35	Apache-2.0

Unpacked Size	Total Files
3.56 MB	5

Issues	Pull Requests
350	26

Last publish

5 days ago

bnoordhuis commented on Jul 14, 2022 Member

More radical: bundle swc and just run the code. :-)

(Caveat emptor: swc is a transpiler, not a type checker. But on the flip side, it also handles JSX and TSX.)

amaro

0.1.9 • Public • Published 5 days ago

Readme | Code Beta | 0 Dependencies | 1 Dependents | 10 Versions

Amaro

Amaro is a wrapper around `@swc/wasm-typescript`, a WebAssembly port of the SWC TypeScript parser. It's currently used as an internal in Node.js for **Type Stripping**, but in the future it will be possible to be upgraded separately by users. The main goal of this package is to provide a stable API for TypeScript parser, which is unstable and subject to change.

> Amaro means "bitter" in Italian. It's a reference to **Mount Amaro** on whose slopes this package was conceived.

How to Install

To install Amaro, run:

```
npm install amaro
```

How to Use

Install

```
> npm i amaro
```

Repository

github.com/nodejs/amaro

Homepage

github.com/nodejs/amaro#readme

Weekly Downloads

1,552

Version	License
0.1.9	MIT

Unpacked Size	Total Files
3.57 MB	7

module: add --experimental-strip-types #53725

Edit

Merged nodejs-github-bot merged 8 commits into `nodejs:main` from `marco-ippolito:feat/typescript-support` on Jul 24

Conversation 272 | Commits 8 | Checks 29 | Files changed 89 | +2,190 −25

marco-ippolito commented on Jul 4 • edited by benjamingr — Member

Moderation Note: This PR has been posted on several social network platforms and thus attracts a lot of mostly off-topic comments.

If you've used this feature and ran into issues or have specific feedback to provide - please open a new issue
https://github.com/nodejs/node/issues/new/choose

It is possible to execute TypeScript files by setting the experimental flag `--experimental-strip-types`.
Node.js will transpile TypeScript source code into JavaScript source code.
During the transpilation process, no type checking is performed, and types are discarded.

Roadmap

Refs: nodejs/loaders#217

Motivation

I believe enabling users to execute TypeScript files is crucial to move the ecosystem forward, it has been requested on all the surveys, and it simply cannot be ignored. We must acknowledge users want to run `node foo.ts` without installing external dependencies or loaders.

> There is a TC39 proposal for type annotations

Reviewers

- VoltrexKeyva
- RedYetiDev
- ChALkeR
- aduh95
- GeoffreyBooth
- privatenumber
- benjamingr
- karlhorky
- jakebailey
- wojtekmaj
- RyanCavanaugh
- magic-akari
- DanielRosenwasser
- targos
- aymen94
- legendecas

Motivation

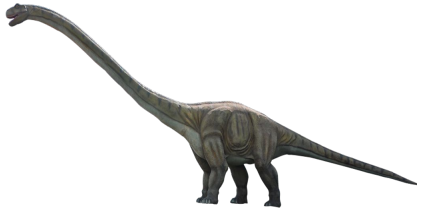
I believe enabling users to execute TypeScript files is crucial to move the ecosystem forward, it has been requested on all the surveys, and it simply cannot be ignored. We must acknowledge users want to run `node foo.ts` without installing external dependencies or loaders.

Experimental TypeScript support

Node v22.6.0 features experimental TypeScript support with the flag **--experimental-strip-types**

Limitations

- No TypeScript features that require transformation
- No downleveling
- No .js imports for .ts file
- Type keyword required for type imports
- No running .ts in node_modules



TypeScript features that require transformation

```ts
// .ts

namespace MathUtil {
    export const add = (a: number, b: number) => a + b;
}

enum Direction {
  Up = 1,
  Down,
  Left,
  Right,
}
```

Node v23.8.0 supports type only namespaces

```
namespace MathUtil {
  export type Circle = {
    x: number;
    y: number;
    radius: number;
  }
}
```

Node v22.7.0 adds TypeScript features support with the flag **--experimental-transform-types**

Source-maps are enabled by default

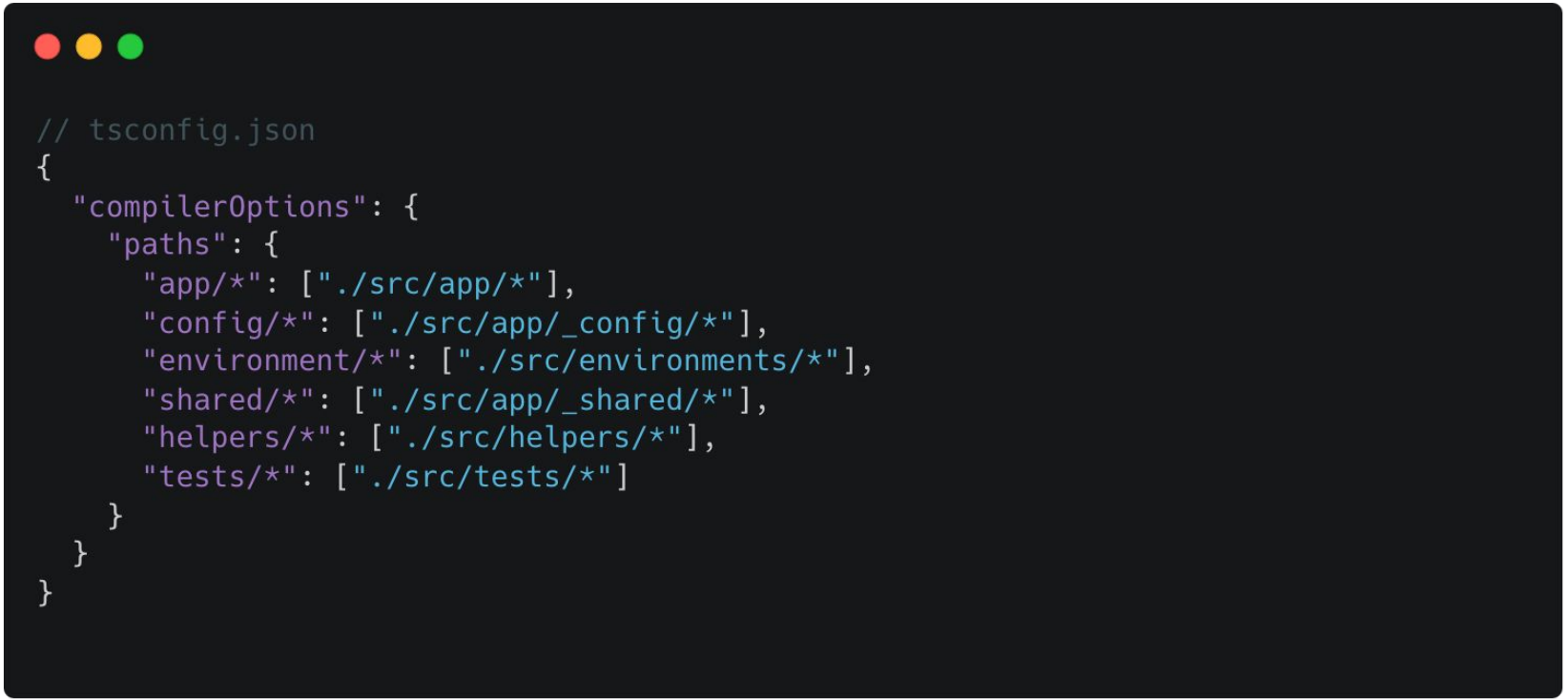
TypeScript features that require transformation

```
// .js

(function(MathUtil) {
    MathUtil.add = (a, b)=>a + b;
})(MathUtil || (MathUtil = {}));

var Direction = function(Direction) {
    Direction[Direction["Up"] = 1] = "Up";
    Direction[Direction["Down"] = 2] = "Down";
    Direction[Direction["Left"] = 3] = "Left";
    Direction[Direction["Right"] = 4] = "Right";
    return Direction;
}(Direction || {});
var MathUtil;
```

TypeScript custom paths are not supported



```
// tsconfig.json
{
  "compilerOptions": {
    "paths": {
      "app/*": ["./src/app/*"],
      "config/*": ["./src/app/_config/*"],
      "environment/*": ["./src/environments/*"],
      "shared/*": ["./src/app/_shared/*"],
      "helpers/*": ["./src/helpers/*"],
      "tests/*": ["./src/tests/*"]
    }
  }
}
```

But you can use Node.js subpath imports

```
// package.json
{
  "imports": {
    "#dep": {
      "node": "dep-node-native",
      "default": "./dep-polyfill.js"
    }
  }
}
```

Limitations

- No TypeScript features that require transformation
- No downleveling
- No .js imports for .ts file
- Type keyword required for type imports
- No running .ts in node_modules

`using` Declarations and Explicit Resource Management

TypeScript 5.2 adds support for the upcoming Explicit Resource Management feature in ECMAScript. Let's explore some of the motivations and understand what the feature brings us.

ECMAScript Explicit Resource Management TC 39

> **NOTE:** This proposal has subsumed the Async Explicit Resource Management proposal. This proposal repository should be used for further discussion of both sync and async of explicit resource management.

This proposal intends to address a common pattern in software development regarding the lifetime and management of various resources (memory, I/O, etc.). This pattern generally includes the allocation of a resource and the ability to explicitly release critical resources.

For example, ECMAScript Generator Functions and Async Generator Functions expose this pattern through the `return` method, as a means to explicitly evaluate `finally` blocks to ensure user-defined cleanup logic is preserved:

Decorators

Stage: 3

Decorators are a proposal for extending JavaScript classes which is widely adopted among developers in transpiler environments, with broad interest in standardization. TC39 has been iterating on decorators proposals for over five years. This document describes a new proposal for decorators based on elements from all past proposals.

This README describes the current decorators proposal, which is a work in progress. For previous iterations of this proposal, see the commit history of this repository.

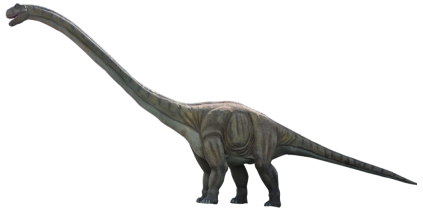
Introduction

Decorators are *functions* called on classes, class elements, or other JavaScript syntax forms during definition.

```
@defineElement("my-class")
class C extends HTMLElement {
  @reactive accessor clicked = false;
}
```

Limitations

- No TypeScript features that require transformation
- No downleveling
- No .js imports for .ts file
- Type keyword required for type imports
- No running .ts in node_modules



TypeScript allows the ‘.ts’ extension in imports with the flag **--allowImportingTsExtensions**.
But is only with **--noEmit** or **--emitDeclarationOnly**.

```
// index.ts

import foo from "foo.ts"
```

```
tsc index.ts

TSError: × Unable to compile TypeScript: error TS5096: Option
'allowImportingTsExtensions' can only be used when either 'noEmit' or
'emitDeclarationOnly' is set.
```

Announcing TypeScript 5.7

Path Rewriting for Relative Paths

There are several tools and runtimes that allow you to run TypeScript code "in-place", meaning they do not require a build step which generates output JavaScript files. For example, ts-node, tsx, Deno, and Bun all support running `.ts` files directly. More recently, Node.js has been investigating such support with `--experimental-strip-types` (soon to be unflagged!) and `--experimental-transform-types`. This is extremely convenient because it allows us to iterate faster without worrying about re-running a build task.

There is some complexity to be aware of when using these modes though. To be maximally compatible with all these tools, a TypeScript file that's imported "in-place" **must** be imported with the appropriate TypeScript extension at runtime. For example, to import a file called `foo.ts`, we have to write the following in Node's new experimental support:

Copy

```
// main.ts

import * as foo from "./foo.ts"; // <- we need foo.ts here, not foo.js
```

With the new flag
--rewriteRelativeImportExtensions
TypeScript will rewrite the '.ts' extension to '.js'.

```
// Under --rewriteRelativeImportExtensions...

// these will be rewritten.
import * as foo from "./foo.ts";
import * as bar from "../someFolder/bar.mts";

// these will NOT be rewritten in any way.
import * as a from "./foo";
import * as b from "some-package/file.ts";
import * as c from "@some-scope/some-package/file.ts";
import * as d from "#/file.ts";
import * as e from "./file.js";
```

But what about existing source code?

Fix it automatically: /c. correct-ts-specifiers

Before

```ts
import { URL } from 'node:url';

import { bar } from '@dep/bar';
import { foo } from 'foo';

import { Bird } from './Bird';
import { Cat } from './Cat.ts';
import { Dog } from '…/Dog/index.mjs';
import { baseUrl } from '#config.js';

export { Zed } from './zed';

// should be unchanged

const nil = await import('./nil.js');
```

After

```ts
import { URL } from 'node:url';

import { bar } from '@dep/bar';
import { foo } from 'foo';

import { Bird } from './Bird/index.ts';
import { Cat } from './Cat.ts';
import { Dog } from '…/Dog/index.mts';
import { baseUrl } from '#config.js';

export type { Zed } from './zed.d.ts';

// should be unchanged

const nil = await import('./nil.ts');
```

JakobJingleheimer/correct-ts-specifiers

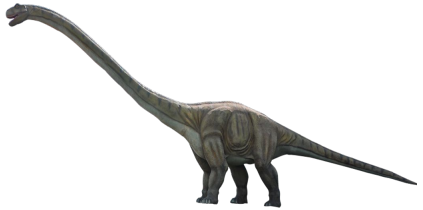
Known limitation (WIP)

`correct-ts-specifiers` can't yet update type imports missing the `type` keyword when resolving via `"types"` in a `package.json` with no implementation (eg, no `"exports"` or `"main"` fields, or `./index.js`).

```
{
  "name": "i-dont-work-yet",
  "types": "./index.d.ts"
}
```

Limitations

- No TypeScript features that require transformation
- No decorators
- No .js imports for .ts file
- Type keyword required for type imports
- No running .ts in node_modules

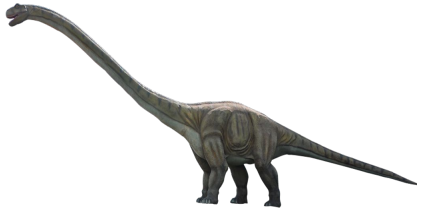


```ts
import type { Type1, Type2 } from './module.ts';
import { fn, type FnParams } from './fn.ts';
```

```ts
import { Type1, Type2 } from './module.ts';
import { fn, FnParams } from './fn.ts';
```

Limitations

- No TypeScript features that require transformation
- No decorators
- No .js imports for .ts file
- Type keyword required for type imports
- No running .ts in node_modules



```
node --experimental-strip-types ./node_modules/foo/index.ts
```

ERR_UNSUPPORTED_NODE_MODULES_TYPE_STRIPPING

Added in: v22.6.0

Type stripping is not supported for files descendent of a `node_modules` directory.

Running ts in node_modules

- Discourage authors from publishing `.ts` in libraries
- IDE needs to typecheck node_modules
- Compatibility with consumers flag
- Issues with compatibility between ts version

Node v23.6.0 enables **--experimental-strip-types** by default.

```
node file.ts

Hello World!
(node:96326) ExperimentalWarning: Type Stripping is an experimental feature and
might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
```

TypeScript syntax detection in eval

```
node --eval "import util from 'util'; const foo: string = 'Hello World!'; console.log(foo);"
Hello World!
```

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

TypeScript syntax detection in eval

```
node --eval "import util from 'util'; const foo         = 'Hello World!'; console.log(foo);"
Hello World!
```

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

How eval syntax detection works

1. Compile as **commonjs** module
2. If step 1 fails, compile as **esm** module
3. If step 2 fails with a **Syntax Error**, strip types
4. If step 3 fails with **Unsupported Syntax** or **Invalid Syntax** throw error from step 2, plus the TS error message, else compile as **commonjs**
5. If step 4 fails, compile as **esm** module

You can skip syntax detection by passing --input-type=**module-typescript** or **commonjs-typescript**

```
node --input-type module-typescript --eval "import util from 'util'; const foo: string = 'Hello World!'; console.log(foo);"
Hello World!
```

Syntax Ambiguity

```
node -p "fetch<Object>('http://example.com')"
```

Syntax Ambiguity

```
node -p "fetch<Object>('http://example.com')"
false
```

Syntax Ambiguity

```
node -p "fetch<Object>('http://example.com')"
false
```

```
> fetch<Object>('http://example.com');
< false
```

Syntax Ambiguity

v5.7.3 ▾ Run Export ▾ Share ⇥

```
1   fetch<Object>('http://example.com')
```

```
"use strict";
fetch('http://example.com');
```

```
node --input-type commonjs-typescript -p "fetch<Object>('http://example.com')"

Promise {
  Response {
    status: 200,
    statusText: 'OK',
    headers: Headers {
      'accept-ranges': 'bytes',
      'content-type': 'text/html',
      etag: '"84238dfc8092e5d9c0dac8ef93371a07:1736799080.121134"',
      'last-modified': 'Mon, 13 Jan 2025 20:11:20 GMT',
      'content-encoding': 'gzip',
      'cache-control': 'max-age=957',
      date: 'Thu, 13 Feb 2025 09:18:08 GMT',
      'content-length': '648',
      connection: 'keep-alive',
      vary: 'Accept-Encoding'
    },
    body: ReadableStream { locked: false, state: 'readable', supportsBYOB: true },
    bodyUsed: false,
    ok: true,
    redirected: false,
    type: 'basic',
    url: 'http://example.com/'
  }
}
```

Announcing TypeScript 5.8 Beta

The `--erasableSyntaxOnly` Option

Recently, Node.js 23.6 unflagged experimental support for running TypeScript files directly; however, only certain constructs are supported under this mode. Node.js has unflagged a mode called `--experimental-strip-types` which requires that any TypeScript-specific syntax cannot have runtime semantics. Phrased differently, it must be possible to easily *erase* or "strip out" any TypeScript-specific syntax from a file, leaving behind a valid JavaScript file.

That means constructs like the following are not supported:

- `enum` declarations
- `namespace`s and `module`s with runtime code
- parameter properties in classes
- `import` aliases


{
"compilerOptions": {
"target": "esnext",
"module": "nodenext",
"allowImportingTsExtensions": true,
"rewriteRelativeImportExtensions": true,
"verbatimModuleSyntax": true,
"erasableSyntaxOnly": true
}
}

Next steps

- Bug fix
- Make it faster
- Unflag in v22
- Make it stable

Feedback

- Give it a try
- @nodejs/typescript repo
- bi weekly meetings, check node calendar

Thanks for listening!!!