What is TypeScript and why should I use it instead of JavaScript? [closed]

Asked 12 years, 2 months ago Modified 3 months ago

Viewed 679k times

1936

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 7 years ago.

Improve this question

What is the TypeScript language? What can it do that JavaScript or available libraries cannot do, that would give me reason to consider it?

javascript typescript

Share

Improve this question

Follow

edited Sep 12 at 23:25

user26627078

asked Oct 2, 2012 at 16:37

Mohammed Thabet
21.7k ● 7 ● 28 ● 43

3 Here are some thought on this: blog.priceandcost.com/development/… – Jefim Dec 10, 2015 at 20:33

4 Some notes here : basarat.gitbooks.io/typescript/content/docs/why-typescript.html – basarat May 13, 2016 at 1:51

5 Answers

Sorted by: Highest score (default)

1439

> I originally wrote this answer when TypeScript was still hot-off-the-presses. Five years later, this is an OK overview, but look at Lodewijk's answer below for more depth

1000ft view...

TypeScript is a superset of JavaScript which primarily provides optional static typing, classes and interfaces. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors *as you type the code*.

To get an idea of what I mean, watch Microsoft's introductory video on the language.

For a large JavaScript project, adopting TypeScript might result in more robust software, while still being deployable where a regular JavaScript application would run.

It is open source, but you only get the clever Intellisense as you type if you use a supported IDE. Initially, this was only Microsoft's Visual Studio (also noted in blog post from Miguel de Icaza). These days, other IDEs offer TypeScript support too.

Are there other technologies like it?

There's CoffeeScript, but that really serves a different purpose. IMHO, CoffeeScript provides readability for humans, but TypeScript also provides deep readability for *tools* through its optional static typing (see this recent blog post for a little more critique). There's also Dart but that's a full on replacement for JavaScript (though it can produce JavaScript code)

Example

As an example, here's some TypeScript (you can play with this in the TypeScript Playground)

```
class Greeter {
    greeting: string;
    constructor (message: string) {
        this.greeting = message;
    }
    greet() {
```

```
        return "Hello, " + this.greeting;
    }
}
```

And here's the JavaScript it would produce

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
})();
```

Notice how the TypeScript defines the type of member variables and class method parameters. This is removed when translating to JavaScript, but used by the IDE and compiler to spot errors, like passing a numeric type to the constructor.

It's also capable of inferring types which aren't explicitly declared, for example, it would determine the `greet()` method returns a string.

Debugging TypeScript

Many browsers and IDEs offer direct debugging support through sourcemaps. See this Stack Overflow question for more details: [Debugging TypeScript code with Visual Studio]

Want to know more?

I originally wrote this answer when TypeScript was still hot-off-the-presses. Check out Lodewijk's answer to this question for some more current detail.

Share Improve this answer

Follow

edited Nov 26, 2019 at 18:56

Westy92

21.2k ● 4 ● 77 ● 64

answered Oct 2, 2012 at 16:41

Paul Dixon

300k ● 54 ● 314 ● 349

117 WebStorm offers nice IntelliSense on TypeScript now and is multiplatform. – Radek Jul 3, 2013 at 11:52

5 @Erwinus: Did you see analogy here? You can also program sometimes with javascript, in all other cases use typescript: it will help you check for types, will provide some intellisense. Why not? ;-) – VikciaR Sep 4, 2015 at 10:46

13 @Erwinus You're making assumptions about how TypeScript works. You can write plain JavaScript as TypeScript and have the compiler only do compile time type checking. There's no performance loss by doing that. – thoughtrepo Sep 11, 2015 at 15:30

8 No, @Erwinus, that code is optional. You can just as easily write the plain JavaScript version in TypeScript. – thoughtrepo Sep 11, 2015 at 17:32

47 @Erwinus The point of TypeScript is to provide compile time type checking. If you don't see the value in that, that's perfectly fine. TypeScript has been refereed to as "your first unit test". There are many resources that discuss whether or not optional type checking has value, and are more detail

than what we can do here. I'm not trying to convince you of anything, just correcting a misconception. – thoughtrepo Sep 11, 2015 at 17:49

1203

TypeScript's relation to JavaScript

> TypeScript is a typed superset of JavaScript that compiles to plain JavaScript - typescriptlang.org.

JavaScript is a programming language that is developed by ECMA's Technical Committee 39, which is a group of people composed of many different stakeholders. TC39 is a committee hosted by ECMA: an internal standards organization. JavaScript has many different implementations by many different vendors (e.g. Google, Microsoft, Oracle, etc.). The goal of JavaScript is to be the lingua franca of the web.

TypeScript is a superset of the JavaScript language that has a single open-source compiler and is developed mainly by a single vendor: Microsoft. The goal of TypeScript is to help catch mistakes early through a type system and to make JavaScript development more efficient.

Essentially TypeScript achieves its goals in three ways:

1. **Support for modern JavaScript features** - The JavaScript language (not the runtime) is standardized through the ECMAScript standards. Not all browsers and JavaScript runtimes support all features of all ECMAScript standards (see this overview). TypeScript allows for the use of many of the latest ECMAScript features and translates them to older ECMAScript targets of your choosing (see the list of compile targets under the `--target` compiler option). This means that you can safely use new features, like modules, lambda functions, classes, the spread operator and destructuring, while remaining backwards compatible with older browsers and JavaScript runtimes.
2. **Advanced type system** - The type support is not part of the ECMAScript standard and will likely never be due to the interpreted nature instead of compiled nature of JavaScript. The type system of TypeScript is incredibly rich and includes: interfaces, enums, hybrid types, generics, union/intersection types, access modifiers and much more. The official website of TypeScript gives an overview of these features. Typescript's type system is on-par with most other typed languages and in some cases arguably more powerful.
3. **Developer tooling support** - TypeScript's compiler can run as a background process to support both incremental compilation and IDE integration such that you can more easily navigate, identify problems, inspect possibilities and refactor your codebase.

TypeScript's relation to other JavaScript targeting languages

TypeScript has a unique philosophy compared to other languages that compile to JavaScript. JavaScript code is valid TypeScript code; TypeScript is a superset of JavaScript. You can almost rename your `.js` files to `.ts` files and start using TypeScript (see "JavaScript interoperability" below). TypeScript files are compiled to readable JavaScript, so that migration back is possible and understanding the compiled TypeScript is not hard at all. TypeScript builds on the successes of JavaScript while improving on its weaknesses.

On the one hand, you have future proof tools that take modern ECMAScript standards and compile it down to older JavaScript versions with Babel being the most popular one. On the other hand, you have languages that may totally differ from JavaScript which target JavaScript, like CoffeeScript, Clojure, Dart, Elm, Haxe, Scala.js, and a whole host more (see this list). These languages, though they might be better than where JavaScript's future might ever lead, run a greater risk of not finding enough adoption for their futures to be guaranteed. You might also have more trouble finding experienced developers for some of these languages, though the ones you will find can often be more enthusiastic. Interoperability with JavaScript can also be a bit more

involved, since they are farther removed from what JavaScript actually is.

TypeScript sits in between these two extremes, thus balancing the risk. TypeScript is not a risky choice by any standard. It takes very little effort to get used to if you are familiar with JavaScript, since it is not a completely different language, has excellent JavaScript interoperability support and it has seen a lot of adoption recently.

Optionally static typing and type inference

JavaScript is dynamically typed. This means JavaScript does not know what type a variable is until it is actually instantiated at run-time. This also means that it may be too late. TypeScript adds type support to JavaScript and catches type errors during compilation to JavaScript. Bugs that are caused by false assumptions of some variable being of a certain type can be completely eradicated if you play your cards right (how strict you type your code or if you type your code at all is up to you).

TypeScript makes typing a bit easier and a lot less explicit by the usage of type inference. For example: `var x = "hello"` in TypeScript is the same as `var x : string = "hello"`. The type is simply inferred from its use. Even it you don't explicitly type the types, they are

still there to save you from doing something which otherwise would result in a run-time error.

TypeScript is optionally typed by default. For example `function divideByTwo(x) { return  x / 2 }` is a valid function in TypeScript which can be called with *any* kind of parameter, even though calling it with a string will obviously result in a *runtime* error. Just like you are used to in JavaScript. This works, because when no type was explicitly assigned and the type could not be inferred, like in the divideByTwo example, TypeScript will implicitly assign the type `any`. This means the divideByTwo function's type signature automatically becomes `function divideByTwo(x : any) : any`. There is a compiler flag to disallow this behavior: `--noImplicitAny`. Enabling this flag gives you a greater degree of safety, but also means you will have to do more typing.

Types have a cost associated with them. First of all, there is a learning curve, and second of all, of course, it will cost you a bit more time to set up a codebase using proper strict typing too. In my experience, these costs are totally worth it on any serious codebase you are sharing with others. A Large Scale Study of Programming Languages and Code Quality in Github suggests that *"statically typed languages, in general, are less defect prone than the dynamic types, and that strong typing is better than weak typing in the same regard".*

It is interesting to note that this very same paper finds that TypeScript is less error-prone than JavaScript:

> For those with positive coefficients we can expect that the language is associated with, ceteris paribus, a greater number of defect fixes. These languages include C, C++, *JavaScript*, Objective-C, Php, and Python. The languages Clojure, Haskell, Ruby, Scala, and *TypeScript*, all have negative coefficients implying that these languages are less likely than the average to result in defect fixing commits.

Enhanced IDE support

The development experience with TypeScript is a great improvement over JavaScript. The IDE is informed in real-time by the TypeScript compiler on its rich type information. This gives a couple of major advantages. For example, with TypeScript, you can safely do refactorings like renames across your entire codebase. Through code completion, you can get inline help on whatever functions a library might offer. No more need to remember them or look them up in online references. Compilation errors are reported directly in the IDE with a red squiggly line while you are busy coding. All in all, this allows for a significant gain in productivity compared to working with JavaScript. One can spend more time coding and less time debugging.

There is a wide range of IDEs that have excellent support for TypeScript, like Visual Studio Code, WebStorm, Atom

and Sublime.

Strict null checks

Runtime errors of the form `cannot read property 'x' of undefined` or `undefined is not a function` are very commonly caused by bugs in JavaScript code. Out of the box TypeScript already reduces the probability of these kinds of errors occurring, since one cannot use a variable that is not known to the TypeScript compiler (with the exception of properties of `any` typed variables). It is still possible though to mistakenly utilize a variable that is set to `undefined`. However, with the 2.0 version of TypeScript you can eliminate these kinds of errors all together through the usage of non-nullable types. This works as follows:

With strict null checks enabled (`--strictNullChecks` compiler flag) the TypeScript compiler will not allow `undefined` to be assigned to a variable unless you explicitly declare it to be of nullable type. For example, `let x : number = undefined` will result in a compile error. This fits perfectly with type theory since `undefined` is not a number. One can define `x` to be a sum type of `number` and `undefined` to correct this: `let x : number | undefined = undefined`.

Once a type is known to be nullable, meaning it is of a type that can also be of the value `null` or `undefined`, the TypeScript compiler can determine through control

flow based type analysis whether or not your code can safely use a variable or not. In other words when you check a variable is `undefined` through for example an `if` statement the TypeScript compiler will infer that the type in that branch of your code's control flow is not anymore nullable and therefore can safely be used. Here is a simple example:

```
let x: number | undefined;
if (x !== undefined) x += 1; // this line will compile
x += 1; // this line will fail compilation, because x
```

During the build, 2016 conference co-designer of TypeScript Anders Hejlsberg gave a detailed explanation and demonstration of this feature: video (from 44:30 to 56:30).

Compilation

To use TypeScript you need a build process to compile to JavaScript code. The build process generally takes only a couple of seconds depending of course on the size of your project. The TypeScript compiler supports incremental compilation (`--watch` compiler flag) so that all subsequent changes can be compiled at greater speed.

The TypeScript compiler can inline source map information in the generated .js files or create separate .map files. Source map information can be used by

debugging utilities like the Chrome DevTools and other IDE's to relate the lines in the JavaScript to the ones that generated them in the TypeScript. This makes it possible for you to set breakpoints and inspect variables during runtime directly on your TypeScript code. Source map information works pretty well, it was around long before TypeScript, but debugging TypeScript is generally not as great as when using JavaScript directly. Take the `this` keyword for example. Due to the changed semantics of the `this` keyword around closures since ES2015, `this` may actually exists during runtime as a variable called `_this` (see this answer). This may confuse you during debugging but generally is not a problem if you know about it or inspect the JavaScript code. It should be noted that Babel suffers the exact same kind of issue.

There are a few other tricks the TypeScript compiler can do, like generating intercepting code based on decorators, generating module loading code for different module systems and parsing JSX. However, you will likely require a build tool besides the Typescript compiler. For example, if you want to compress your code you will have to add other tools to your build process to do so.

There are TypeScript compilation plugins available for Webpack, Gulp, Grunt and pretty much any other JavaScript build tool out there. The TypeScript documentation has a section on integrating with build tools covering them all. A linter is also available in case you would like even more build time checking. There are also a great number of seed projects out there that will

get you started with TypeScript in combination with a bunch of other technologies like Angular 2, React, Ember, SystemJS, Webpack, Gulp, etc.

JavaScript interoperability

Since TypeScript is so closely related to JavaScript it has great interoperability capabilities, but some extra work is required to work with JavaScript libraries in TypeScript. TypeScript definitions are needed so that the TypeScript compiler understands that function calls like `_.groupBy` or `angular.copy` or `$.fadeOut` are not in fact illegal statements. The definitions for these functions are placed in `.d.ts` files.

The simplest form a definition can take is to allow an identifier to be used in any way. For example, when using Lodash, a single line definition file `declare var _ : any` will allow you to call any function you want on `_`, but then, of course, you are also still able to make mistakes: `_.foobar()` would be a legal TypeScript call, but is, of course, an illegal call at run-time. If you want proper type support and code completion your definition file needs to to be more exact (see lodash definitions for an example).

Npm modules that come pre-packaged with their own type definitions are automatically understood by the TypeScript compiler (see documentation). For pretty much any other semi-popular JavaScript library that does not include its own definitions somebody out there has

already made type definitions available through another npm module. These modules are prefixed with "@types/" and come from a Github repository called DefinitelyTyped.

There is one caveat: the type definitions must match the version of the library you are using at run-time. If they do not, TypeScript might disallow you from calling a function or dereferencing a variable that exists or allow you to call a function or dereference a variable that does not exist, simply because the types do not match the run-time at compile-time. So make sure you load the right version of the type definitions for the right version of the library you are using.

To be honest, there is a slight hassle to this and it may be one of the reasons you do not choose TypeScript, but instead go for something like Babel that does not suffer from having to get type definitions at all. On the other hand, if you know what you are doing you can easily overcome any kind of issues caused by incorrect or missing definition files.

Converting from JavaScript to TypeScript

Any `.js` file can be renamed to a `.ts` file and ran through the TypeScript compiler to get syntactically the same JavaScript code as an output (if it was syntactically

correct in the first place). Even when the TypeScript compiler gets compilation errors it will still produce a `.js` file. It can even accept `.js` files as input with the `--allowJs` flag. This allows you to start with TypeScript right away. Unfortunately, compilation errors are likely to occur in the beginning. One does need to remember that these are not show-stopping errors like you may be used to with other compilers.

The compilation errors one gets in the beginning when converting a JavaScript project to a TypeScript project are unavoidable by TypeScript's nature. TypeScript checks *all* code for validity and thus it needs to know about all functions and variables that are used. Thus type definitions need to be in place for all of them otherwise compilation errors are bound to occur. As mentioned in the chapter above, for pretty much any JavaScript framework there are `.d.ts` files that can easily be acquired with the installation of DefinitelyTyped packages. It might, however, be that you've used some obscure library for which no TypeScript definitions are available or that you've polyfilled some JavaScript primitives. In that case, you must supply type definitions for these bits for the compilation errors to disappear. Just create a `.d.ts` file and include it in the tsconfig.json's `files` array, so that it is always considered by the TypeScript compiler. In it declare those bits that TypeScript does not know about as type `any`. Once you've eliminated all errors you can gradually introduce typing to those parts according to your needs.

Some work on (re)configuring your build pipeline will also be needed to get TypeScript into the build pipeline. As mentioned in the chapter on compilation there are plenty of good resources out there and I encourage you to look for seed projects that use the combination of tools you want to be working with.

The biggest hurdle is the learning curve. I encourage you to play around with a small project at first. Look how it works, how it builds, which files it uses, how it is configured, how it functions in your IDE, how it is structured, which tools it uses, etc. Converting a large JavaScript codebase to TypeScript is doable when you know what you are doing. Read this blog for example on converting 600k lines to typescript in 72 hours). Just make sure you have a good grasp of the language before you make the jump.

Adoption

TypeScript is open-source (Apache 2 licensed, see GitHub) and backed by Microsoft. Anders Hejlsberg, the lead architect of C# is spearheading the project. It's a very active project; the TypeScript team has been releasing a lot of new features in the last few years and a lot of great ones are still planned to come (see the roadmap).

Some facts about adoption and popularity:

- In the 2017 StackOverflow developer survey TypeScript was the most popular JavaScript transpiler (9th place overall) and won third place in the most loved programming language category.
- In the 2018 state of js survey TypeScript was declared as one of the two big winners in the JavaScript flavors category (with ES6 being the other).
- In the 2019 StackOverflow developer survey TypeScript rose to the 9th place of most popular languages amongst professional developers, overtaking both C and C++. It again took third place amongst most the most loved languages.
- In the 2020 StackOverflow developer survey TypeScript was the second most loved technology.

Share Improve this answer

Follow

edited Feb 1, 2023 at 9:14

Cœur

38.6k ● 26 ● 202 ● 276

answered Jan 27, 2016 at 21:23

Lodewijk Bogaards

20k ● 3 ● 30 ● 54

38 "JavaScript code is valid TypeScript code" - this is actually not always true. I mean code like if(1 === '1') {} gives you an error in TS and in JS don't. But most of the time, if JS code is well-written it's true. – Maciej Bukowski Mar 16, 2017 at 23:21

5 If you have lost your precious productive time fretting over a missing semi colon, writing in Typescript would be a life

saver. – SoSufi Mar 18, 2017 at 8:22

3 Typings has been deprecated, and the current best practice is to just `npm` (or `yarn`) `install @types/foo`. Can you update your answer? – Jed Fox Dec 21, 2017 at 22:09

13 @MaciejBukowski Even if TypeScript actually complains in this case, you still get a valid JS output (which will be the JS code that you compiled/transpiled with TypeScript). So it's a "compile with error", and **not** invalid TypeScript. It's written down in TypeScript specification that any valid JS code must be valid TS code. – Pac0 Jan 7, 2019 at 14:58

2 @Maciej Bukowski if(1 === '1') {} does not give you an error but it does give you a notice that your statement is always false (This condition will always return 'false' since the types '1' and '"1"' have no overlap.) I mean if a JavaScript dev writes code like that they should return back to basics, funny thing is I work with senior developers that write if statements like if(1+1) or if(true) and this people make a ton of money... – Avram Virgil Apr 26, 2019 at 14:47

96

TypeScript does something similar to what less or sass does for CSS. They are super sets of it, which means that every JS code you write is valid TypeScript code. Plus you can use the other goodies that it adds to the language, and the transpiled code will be valid js. You can even set the JS version that you want your resulting code on.

Currently TypeScript is a super set of ES2015, so might be a good choice to start learning the new js features and transpile to the needed standard for your project.

Share Improve this answer

Follow

edited Feb 21, 2017 at 15:28

wonea

4,959 ● 17 ● 89 ● 143

answered Feb 11, 2016 at 20:32

lebobbi

2,277 ● 18 ● 22

6 Best TL;DR is TS's page: "TypeScript is a typed superset of JavaScript that compiles to plain JavaScript." – Ruan Mendes Dec 22, 2016 at 23:36

1 It doesn't answer the "why should I use it" though. Here the tl;dr would be: 1) To add optional static types to JavaScript. Types can help to catch bugs at compile time and better document your program. 2) You can write the **new** JavaScript (ES6 / ES7 / ESnext) and compile it back to ES5, which is necessary to support older browsers; I've elaborated a bit more at tsmean.com/articles/vs/typescript-vs-javascript for those interested in more than a tl;dr – bersling Sep 24, 2017 at 15:56

1 "The transpiled code will be valid JS" - and that's TypeScript's Achilles heel if you ask me. It means they can't add several very useful features to JS; most notably *runtime* type checking. It's a bit annoying to have compiler-time type safety only to lose it for any runtime data that is read in from I/O, or any time your transpiled code is called unsafely from other JS. – Jez Apr 3, 2018 at 13:50

53

"**TypeScript Fundamentals**" -- a Pluralsight video-course by **Dan Wahlin** and **John Papa** is a really good, presently (March 25, 2016) updated to reflect TypeScript 1.8, introduction to Typescript.

For me the really good features, beside the nice possibilities for intellisense, are the *classes*, *interfaces*, *modules*, the ease of implementing AMD, and the possibility to use the Visual Studio Typescript debugger when invoked with IE.

To summarize: If used as intended, Typescript can make JavaScript programming more reliable, and easier. It can increase the productivity of the JavaScript programmer significantly over the full SDLC.

Share Improve this answer Follow

edited May 4, 2016 at 22:54

answered Dec 27, 2015 at 4:18

Dimitre Novatchev

243k ● 27 ● 303 ● 436

9 what are SDLC? AMD? – Oooogi Jan 13, 2016 at 9:00

19 @Oooogi, SDLC == Software Development Life Cycle. AMD == Asynchronous Module Definition. The latter is specific to JavaScript, while the former is rather generic in scope. – Dimitre Novatchev Jan 13, 2016 at 15:17

1 @confusedpunter More precisely: increase in **developer's** productivity. This is not only my personal impression, based on experience. Many thousands of experienced developers have started using Typescript since its creation. One example is the Angular developers team -- for what other reason would you think a leading Google team would completely rewrite their codebase in Typescript? And thousands of types for almost any JS framework have been

created at github.com/DefinitelyTyped/DefinitelyTyped – Dimitre Novatchev Jun 28, 2022 at 0:10

Thanks I'll read it – confusedpunter Jun 30, 2022 at 10:41

19

Ecma script 5 (ES5) which all browser support and precompiled. ES6/ES2015 and ES/2016 came this year with lots of changes so to pop up these changes there is something in between which should take cares about so TypeScript.

• TypeScript is Types -> Means we have to define datatype of each property and methods. If you know C# then Typescript is easy to understand.

• Big advantage of TypeScript is we identify Type related issues early before going to production. This allows unit tests to fail if there is any type mismatch.

Share Improve this answer

Follow

edited Oct 9, 2020 at 6:31

Tintin81

10.2k • 20 • 91 • 181

answered Jun 6, 2016 at 12:14

Mayank Jain

235 • 2 • 6

2 It's not every year buddy !.. they have changed the spec after very long wait – Subham Tripathi Oct 5, 2016 at 10:19

2 @SubhamTripathi It very much *is* every year. ES2015, ES2016, ES2017, and on from now until the language dies. It *wasn't* every year, before 2015, but it is now. Go search for

"TC39 process" to learn more. – daemone Oct 25, 2017 at 13:23