Programming Typescript

Nathan Warner

Northern Illinois
University

Computer Science
Northern Illinois University
United States

Contents

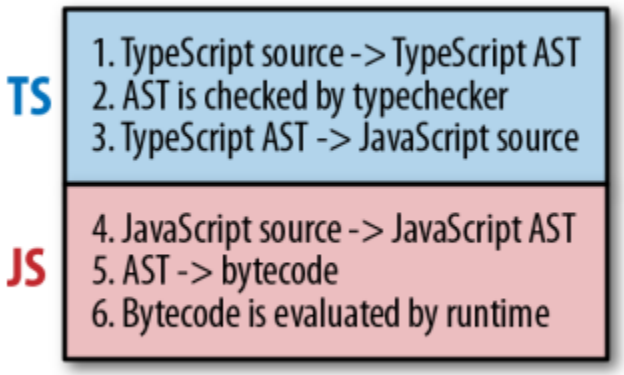
The compiler

Let's start broad: programs are files that contain a bunch of text written by you, the programmer. That text is parsed by a special program called a compiler, which trans- forms it into an abstract syntax tree (AST), a data structure that ignores things like whitespace, comments, and where you stand on the tabs versus spaces debate. The compiler then converts that AST to a lower-level representation called bytecode. You can feed that bytecode into another program called a runtime to evaluate it and get a result. So when you run a program, what you're really doing is telling the runtime to evaluate the bytecode generated by the compiler from the AST parsed from your source code. The details vary, but for most languages this is an accurate high-level view.

Where TypeScript is special is that instead of compiling straight to bytecode, Type- Script compiles to… JavaScript code! You then run that JavaScript code like you nor- mally would—in your browser, or with NodeJS, etc.

n. I actually skipped over a crucial step: after the TypeScript Compiler generates an AST for your program—but before it emits code—it typechecks your code.

Note: A *typechecker* is a special program that verifies that your code is typesafe.



Steps 1–3 are done by TSC (typescript compiler), and steps 4–6 are done by the JavaScript runtime that lives in your browser, NodeJS, or whatever JavaScript engine you're using.

In this process, steps 1–2 use your program's types; step 3 does not. That's worth reiterating: when TSC compiles your code from TypeScript to JavaScript, it won't look at your types. That means your program's types will never affect your program's gener- ated output, and are only used for typechecking. This feature makes it foolproof to play around with, update, and improve your program's types, without risking break- ing your application.

The Type System

Modern languages have all sorts of different type systems. A *type system* is a set of rules that a typechecker uses to assign types to your program.

There are generally two kinds of type systems: type systems in which you have to tell the compiler what type everything is with explicit syntax, and type systems that infer the types of things for you automatically. Both approaches have trade-offs

TypeScript is inspired by both kinds of type systems: you can explicitly annotate your types, or you can let TypeScript infer most of them for you

To explicitly signal to TypeScript what your types are, use annotations. Annotations take the form value: type and tell the typechecker, “Hey! You see this value here? Its type is type.”