Using TypeScript on the Command Line

Type check and transpile TypeScript files into JavaScript files on the command line.

TypeScript's command line interface (CLI) type checks and transpiles TypeScript files with one command: `tsc`. In this article, we'll learn how to install `tsc` globally, what configurations it applies by default, and how to configure it with flags. Let's dive in!

Installing TypeScript Globally

We can use `tsc` to run TypeScript's type-checking and transpilation features from the command line against a TypeScript file.

To install `tsc`, run:

```
npm install --global typescript
```

This command will install `tsc` globally. Installing TypeScript globally is convenient when first getting started with TypeScript since we can run `tsc` in any directory, against any TypeScript file, without any prior setup.

Using `tsc`

Run the following commands in the terminal to create a folder called `tsc-clock` and a file `clock.ts`:

```
mkdir tsc-clock
touch tsc-clock/clock.ts
```

Then open up the file with VSCode or your editor of choice. Once open, we'll write a program that outputs the current time to the console. Copy and paste the following code in `clock.ts`:

```
function logTime(date: Date): void {
  console.log(`The time is ${date.toLocaleTimeString()}`);
}

logTime(new Date());
```

Save the file. Then `cd` into `tsc-clock` and run the following command in the terminal:

```
tsc clock.ts
```

This command will transpile the `clock.ts` file into a new file named `clock.js`. When we open the newly created file, we see:

```
function logTime(date) {
    console.log("The time is ".concat(date.toLocaleTimeString()));
}
logTime(new Date());
```

When TypeScript transpiled `clock.ts` into `clock.js`, it made a few alterations to our program:

- The transpilation removed the types from the `logTime()` function's argument and return type.
- The transpilation also replaced the template literal syntax with the `concat()` function.

Now that TypeScript compiled `clock.ts` into `clock.js`, we can run the program with the following command in the terminal:

```
node clock.js
```

Our program will output the current time to the command line.

Thinking back to the transpilation step, it makes sense that `tsc` removed the TypeScript types from our program since JavaScript does not support types. With that said, why did TypeScript remove the template literal syntax and replace it with the `.concat()` method?

Configuring `tsc`

By default, `tsc` uses a set of default configurations that will work on most modern-day computers. `tsc` over a hundred options which we can see in the TypeScript CLI reference. Under "Compiler flags", there's a list of all of `tsc`'s options and their default values.

Let's focus on one of them: `target`. By default, `target` is set to transpile TypeScript code into ES3 JavaScript code. ES3, short for ECMAScript 3, was released in December 1999. `tsc` transpiles TypeScript into ES3 since nearly all computers that can run JavaScript can execute the syntax in specified in ES3.

One disadvantage of transpiling to ES3 is that it's more verbose than modern JavaScript specifications. This verbosity results in larger JavaScript file sizes. If we send these JavaScript files to users, like when they visit a website, we'd send them more bytes of code than they need to run our program. As a result, our website would be slower than it needs to be.

`tsc` allows us to set the transpilation target when running the `tsc` command. Let's set the `--target` flag and see how it changes our transpiled JavaScript:

```
tsc clock.ts --target esnext
```

Now our transpiled code inside `clock.js` looks like:

```
function logTime(date) {
    console.log(`The time is ${date.toLocaleTimeString()}`);
}
logTime(new Date());
```

Notice that there is no longer the `.concat()` method call, since modern versions of JavaScript support template literals. While this change makes our transpiled code slightly smaller, when transpiling large programs, the file size savings can be significant.

The `--target` flag is just one example of how we can alter how `tsc` transpiles our files. To learn about all the flags, check out the TypeScript CLI reference.

Wrap Up

We can use the TypeScript CLI, `tsc`, to quickly type check and transpile files. Installing `tsc` globally and then passing flags to alter its output is a fantastic way to try out TypeScript and all of its features. With this tool in our belt, we're ready to start writing TypeScript programs locally and to contribute to larger TypeScript projects.