Integrate TypeScript in your Vue project

Posted on Jun 28 2017 · Vue · TypeScript

You have a Vue project and you heard about all things TypeScript can help you with. You decide to start using TypeScript. Here you'll see how to do that in an approachable and pragmatic way.

In this article we'll assume you're using SFC (Single File Components), but it could work as well even if you're splitting them into multiple files. So... let's get started!

Integrating TypeScript

We're gonna start from Vue CLI Webpack's template, but it would work as well with the amazing PWA template. Don't choose to use ESLint when you're ask to:

```
vue init webpack my-app # or: vue init pwa my-app
cd my-app
```

We have to do 4 steps:

1. Create a `tsconfig.json` file

Let's start with something very simple, later we'll get back to the TypeScript configuration.

```
{
  "compilerOptions": {
    "lib": ["dom", "es5", "es2015"],
    "target": "es5",
    "module": "es2015",
    "moduleResolution": "node",
    "sourceMap": true,
    "allowSyntheticDefaultImports": true
  }
}
```

The most important part is the `allowSyntheticDefaultImports` setting. Since Vue types doesn't use ES2015 default exports, this setting must be set to by-pass that. You can see more info in this VSCode docs page.

Setting `"module": "es2015"` would make the code tree-shakeable by producing ESM (EcmaScript Modules).

2. Add `ts-loader` and webpack tweaks

Install `typescript` an `ts-loader` with npm:

```
npm i -O typescript ts-loader
```

Then open `build/webpack.base.conf.js`, and place the following code at **the beginning** of `module.rules`, right before than `vue-loader`:

```
module: {
    rules: [
      {
        test: /\.ts$/,
        exclude: /node_modules|vue\/src/,
        loader: "ts-loader",
        options: {
          appendTsSuffixTo: [/\.vue$/]
        }
      },
    ...
```

In there, rename the entry to `.ts` and add it to the extensions:

```
...
entry: {
  app: './src/main.ts'
},
...
resolve: {
    extensions: ['.ts', '.js', '.vue', '.json'],
...
```

3. Add `es-module: true` to `build/vue-loader.conf.js`

That will tell vue-loader to use ES instead of CJS (CommonJS) modules, as describe in vue-loader docs:

```
module.exports = {
  loaders: utils.cssLoaders({
    sourceMap: isProduction
      ? config.build.productionSourceMap
      : config.dev.cssSourceMap,
    extract: isProduction
  }),
  esModule: true
}
```

4. Use TypeScript in files

So you must do 2 things here:

- Rename `.js` to `.ts` extensions within the `src` folder
- Use `lang="ts"` on the `script` tag of you Vue file. For example in `App.vue`:

```
<script lang="ts">
export default {
  name: 'app'
}
</script>
```

Troubleshooting

If your editor is yelling at the line `import App from './App'` in `main.js` file about not finding the App module, you can add a `vue-shim.d.ts` file to your project with the following content:

```
declare module "*.vue" {
  import Vue from 'vue'
  export default Vue
}
```

I'm using VSCode 1.13.1 and not seeing it, but I've seen it before.

TSLint, Prettier... Goddess!

I've recorded a 2 min video on Egghead where I explain how you can set up TSLint with Prettier without any conflicts. Go check it out!

Ok, I can use TypeScript... so now what's next?

At this point, TypeScript could already point you out to some errors you haven't noticed before by using built-in and third-party types and gives you a better dev experience by using type inference, as explained in TypeScript at Slack, an article telling how Slack moved their codebase to TypeScript.

Still, you must add your own types by using interfaces, types, enums, classes and whatever you need. That way you'll add more type coverage that TypeScript will use to apply static typing, assuring type safety.

Ideally, you'd use the TypeScript 2.3 `strict` compiler option in your `tsconfig.json` because it'll bring you the most type safety. Marius Schulz has a well explained article about this. By TypeScript 2.3, the strict option is a group of 4 options, but in future releases it could add more:

- `strictNullChecks`
- `noImplicitAny`
- `noImplicitThis`
- `alwaysStrict`

However, if you have a medium/large codebase, using strict option will make you spend really a huge effort and time solving all the type shortage.

For that case, a good way to integrate TypeScript is to start with the most flexible configuration, and as you add more type coverage to your codebase, start to enable individually the flags mentioned above, till you get to the point that you can apply the `strict` option, so you can approach it in a pragmatic way.

Conclusion

TypeScript brings you type safety to your codebase from the very start, but the more types you define, the more easy to detect bugs and errors and to make the code maintainable. Vue from version 2 is shipped with typings, so we can make use of them. They become more powerful when you use OOP (Object Oriented Programming) in Vue, but will see that in another post.

If you like it, please go and share it! You can follow me on this blog or on twitter as @alexjoverm. Any questions? Shoot!

NEXT POST →