TYPESCRIPT: WRITING SAFE AND EXPRESSIVE CODE

Shevchenko B.M.
Scientific supervisor– assistant Zybina, K.V.
Kharkiv National University of Radioelectronics
(Software Engineering Department, 14, Nauky Ave., Kharkiv, 61166, Ukraine
tel. (057) 702-14-46)
e-mail: bohdan.shevchenko1@nure.ua

According to many ratings, JavaScript is the most popular programming language in the world. JavaScript has achieved such popularity due to its simplicity, flexibility, and expressiveness. However, it has significant disadvantages including dynamic weak typing and excessive liberty. That is why transpilers were developed - programs, that transform code from the language of choice to JavaScript. TypeScript is a definitive winner in the world of transpilers. “TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.”, as the official website says [1]. TypeScript makes writing code easier, but still, there exist subtle problems that can be solved by a proper approach to coding.

TypeScript enables all the features of modern object-oriented programming languages and extends it with a functional programming type system. However, the problem with TypeScript is that the code written in it will be a plain JavaScript at last. This means we will never see these two features in TypeScript [2]:

1. Real type reflection and runtime generics.
2. Dependent types when they require runtime values. Other cases can be successfully emulated by intersection, union types and type inference [3].

Despite these two features, all other features can be successfully adopted in this flexible language. For example, the equality protocol and calculating hash codes for maps (or dictionaries) and sets.

This problem really bothered me and I spent hours searching for an existing library that could satisfy my needs, but I failed to find anything. So I came up with the solution of developing a node module, named *eq-collections*, that implements dictionaries with objects as keys and sets of object values. Both maps and sets have two implementation - one with resolving hash-code collisions and one without. Though this library is well tested, it has problems with type definitions, needed for TypeScript development. This leads to poorer code completion and intellisense [4, 5], so I will revisit this issue later.

The second thing that could make a developer to write boilerplate code for operating iterable sequences and arrays such as grouping items according to their parameters by applying very broad and generalized *reduce* and *map* operators. This means having functions similar to the LINQ framework from C# language. There exist several implementations of LINQ among packages: *linq*, *linq-collections*, *rx*, *linqjs*, *linqts* and others. In spite of being helpful LINQ has

an obscure naming convention, that is only clear if used with SQL. Besides, a developer, who is not familiar with the original LINQ API will not be able to use these packages with ease.

Fortunately, there exist *ix* package, that provides very similar to LINQ functionality and integrates nicely with TypeScript and *RxJS* library - very popular Reactive Extensions for JavaScript.

The first two libraries discussed can be helpful not only in TypeScript environment, but they are also especially useful with typing. Following tips utilize features of the TypeScript type system, particularly its functional programming features. The tips make use of the *fp-ts* library:

1. Avoid *null* and *undefined*. TypeScript compiler has a flag that makes all the types non-nullable. It means that *null* stops to be a valid value for type and nullable type must be denoted as "*type?*". We can go even further and make the code use *Option<T>* type from the *fp-ts* library. This is a container that can keep either a value or a none.

2. Exceptions are not always useful. Exceptions should be used only in an exceptional situation when the state of the program is unrecoverable or nearly unrecoverable. Other situations must be handled with *Either<T>* type from the *fp-ts*. It is a container that can store either value or error. In this regard, it is similar to *Option<T>*.

References

1. TypeScript. Available at: https://www.typescriptlang.org/ (accessed 4 February 2019).
2. Bierman G., Abadi M., Torgersen M. Understanding typescript //European Conference on Object-Oriented Programming. – Springer, Berlin, Heidelberg, 2014. – P. 257-281.
3. Richards G., Zappa Nardelli F., Vitek J. Concrete types for TypeScript //29th European Conference on Object-Oriented Programming (ECOOP 2015). – Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
4. Fischer L., Hanenberg S. An empirical investigation of the effects of type systems and code completion on api usability using typescript and javascript in ms visual studio //ACM SIGPLAN Notices. – ACM, 2015. – V. 51. – # 2. – P. 154-167.
5. Williams J. et al. Mixed Messages: Measuring Conformance and Non-Interference in TypeScript. – 2017.