

A Brief Introduction to TypeScript:

Is It the Next Big Thing?

Qianyi Wu

Rice University

Abstract

JavaScript, as known as ECMA Script, is the most commonly used programming language on Earth (Rowinski, 2016). However, JavaScript does not have a type system that could perform in every way that you would expect. TypeScript, a strict superset of JavaScript that adds a static typing system and class-based objected-oriented programming features, was brought to the light by Microsoft in 2012 to aid development of large scale applications using JavaScript, as well as to fill the feature gap between the state of the art of JavaScript and JavaScript in the wild of web development (Hejlsberg, 2016). This paper focuses on the introduction of the key features that TypeScript offers. It gives an analysis why TypeScript is necessary in the world of JavaScript, and the reasons why these features could really help JavaScript to scale up in the development process of large web applications.

Keywords: JavaScript, TypeScript, static type, object-oriented, scale

A Brief Introduction to TypeScript:

Why Is It A Leap Forward?

According to the developer survey results from stackoverflow.com (2016), JavaScript is being used by more than fifty-five percent of software developers. It has become the most popular programming language on the planet. Along with the explosion of popularity of JavaScript, React and Node.js also appeared among the top ten most loved technologies by developers. JavaScript used to be a scripting language for small applications which usually contain around a few hundred lines of code. Nowadays, developers are using JavaScript to write large-scaled web applications that consists of hundreds of thousands of lines, which makes the drawbacks of JavaScript even more problematic (Hejlsberg, 2016).

JavaScript has a few very controversial characteristics. For example, its typing system is a very strange one comparing to other programming languages. Not only does it check the type of a variable at run time, it also coerces the value whose type does not fit to the “correct” type. There are some error messages that I am sure every single JavaScript developer has seen very often because of the dynamic and coercing typing system. Here are three examples:

```
// Dynamic typing examples
> var foo = null
> foo.prop
TypeError: Cannot read property 'prop' of null

// Another dynamic typing example
> var bar = {}
> bar.prop
undefined

//Coercion example
> '3' * '4'
12
```

Of course, there are more examples like this. One could argue that these features of JavaScript are what makes JavaScript a great scripting language, however, they really bring down the productivity of the developers because tools cannot figure out the type of each variable belongs

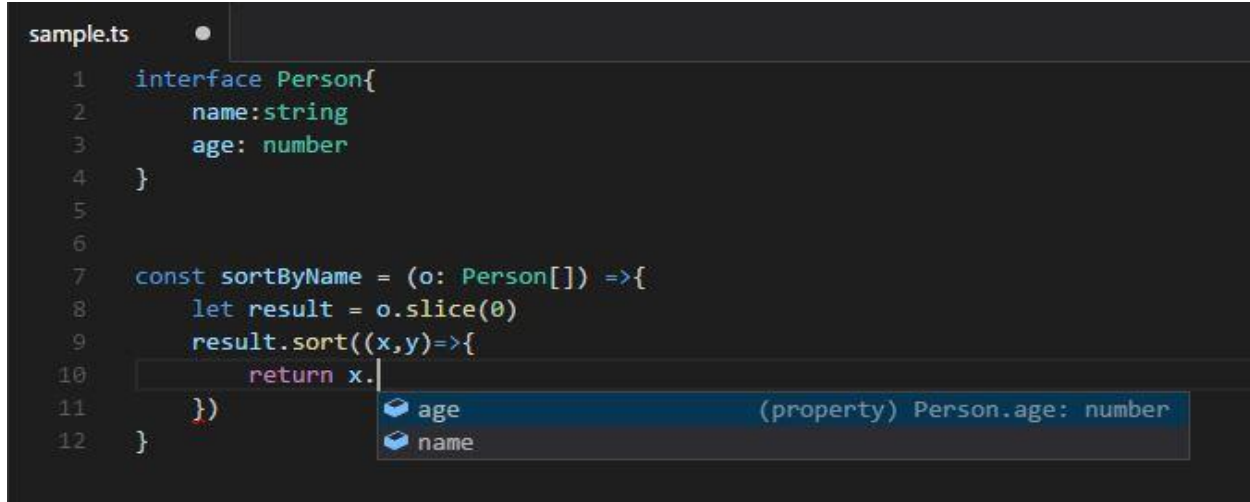
to help you at compile time. In addition, the dynamic typing system significantly increase the debugging difficulty when building a large application, since it is going to be terribly time consuming to find out where the variable you are looking for was initially defined and later modified.

There is also a huge difference between ideal and reality where JavaScript is at because of its speed of evolution, as Hejlsberg (2016), the father of TypeScript, called “the feature gap”. The state of the art JavaScript has many new features that could help the JavaScript developers manage the code, like ES6’s modules and classes. However, in the wild of web development, the support of these features are lagging behind (Hejlsberg, 2016). While it is not possible to enforce old browsers like IE, whose support had been discontinued by Microsoft, to run ES6, Developers have to use other tools like polyfillers and transcompilers to write JavaScript in the style that they like.

TypeScript brings solutions to these major problems. It provides a static typing system so that it is easier for tools to support type checking and IntelliSense at compile time to make software more reliable. It also fills in the gap by offering the ability to transcompile TypeScript code to any JavaScript of version ES3 or later.

Beyond this, TypeScript shines the light to those programmers who have not written any JavaScript before, especially to .NET developers. TypeScript brings the familiar Object-Oriented Programming (OOP) constructs, such as classes and interfaces to JavaScript, which means it could implement all four principles, namely, Encapsulation, Inheritance, Abstraction and

Polymorphism.



```
sample.ts
1  interface Person{
2      name:string
3      age: number
4  }
5
6
7  const sortByNames = (o: Person[]) =>{
8      let result = o.slice(0)
9      result.sort((x,y)=>{
10         return x.
11     })
12 }
```

IntelliSense popup for `return x.`:

- age (property) Person.age: number
- name

Figure 1 Code Snippet showing IntelliSense and support of latest JavaScript features in TypeScript

Why would we use TypeScript instead of other languages with similar offerings? First, TypeScript provides technology decoupling, which means you could go back to Vanilla JavaScript at any time. Since TypeScript is a strict superset of JavaScript, the generated code looks very much like the original code written in TypeScript, therefore you could work directly on the generated code. The counter examples are Dart and CoffeeScript, their generated JavaScript code looks nothing like the original ones because they have their own syntaxes. Second, in building large scale applications, the type safety that TypeScript offers could be enforced on all API implementations to avoid confusion and misuse. At the meantime, it could provide freedom for programmers to work with no types at all since type safety is optional (Amran, 2015). In addition, I think that the Object-Oriented Programming concepts brought by TypeScript could be used to minimize the code that needs to be written and make the system more agile to potential changes, like using design patterns for decoupling.

There are also downsides of using TypeScript. The most important one that a programmer should notice before jumping in is that TypeScript is still not very popular despite all the heat it has drawn recently. Therefore, there could be less help that one could get from the community. Secondly, there is a large chance that a JavaScript programmer uses some kind of external libraries during the development. Although there is a project called *Definitely Typed* that provides definition files for a lot of popular libraries, you may not be able to find the definition file you need if the library that you are going to use is not that popular. Also, as Amran (2015) stated in his article, “Working with types, classes, and interfaces can lead to over-engineering. Many Java developers feel that they waste too much time on boilerplate code just to define a class. This can also happen in TypeScript if you are not careful”, which is definitely true. In order not to abuse the convenience that TypeScript provided, a programmer has to be always aware of the trade-offs. While there are plugins for TypeScript in all major IDEs, they could be buggy because of the length of the time TypeScript has been around.

In conclusion, TypeScript is a newly developed language that enables Object-Oriented programmers, like .NET developers, to have easier access to JavaScript. It does provide very cool features, like static typing, great tool support and fundamental OOP elements, which could lead to very robust, large-scale software written in JavaScript. Software written in TypeScript could be much easier to be maintained and extended, and also much easier to be switched back to plain JavaScript. However, TypeScript as a programming language is still far away from being perfect. Just before TypeScript 2.0 release, it had a major issue dealing with non-nullable types. The most issues that present in TypeScript is because of its lack of popularity and length of existence, thus community support. If TypeScript could keep drawing attention from web

developers and survive the fast-changing world of JavaScript, there would be a better using experience and more help from the fellow developers.

Reference List

Amran, G. (2015, September 30). Developing Large-Scale Applications with TypeScript.

Retrieved November 26, 2016, from <http://blog.wix.engineering/2015/09/30/developing-large-scale-applications-with-typescript/>

B., R. (n.d.). Why Does JavaScript Suck? | A Terrible Language with a Bright Future. Retrieved

November 26, 2016, from <https://whydoesitsuck.com/why-does-javascript-suck/>

Hejlsberg, A. (2016). What's New in TypeScript – Build2016 Conference. Retrieved November

26, 2016, from http://video.ch9.ms/ch9/4ae3/062c336d-9cf0-498f-ae9a-582b87954ae3/B881_mid.mp4

Rowinski, D. (2016). It's Official: JavaScript Is The Most Commonly Used Programming

Language On Earth - ARC. Retrieved November 26, 2016, from

<https://arc.applause.com/2016/03/22/javascript-is-the-worlds-dominant-programming-language/>

Developer Survey Results. Stackoverflow.com. Retrieved November 26, 2016, from

<http://stackoverflow.com/research/developer-survey-2016>