TypeScript Tutorial

Introduction

JavaScript have been around for several years, but is still growing as more and more companies take advantage of the availability it provides for web development [1]. JavaScript has some issues that we have addressed in multiple weeks of this course. Anders Møller described is as dressing a rotten onion with layers, but at the core it was still the same rotten onion. JavaScript does not contain static typing, classes, modules or interfaces. It was not designed for big applications, however, many web developers have that wish. The layers of the rotten onion described by Møller could be some of the libraries we have looked at through this course, eg. JQuery. Those libraries try to hide the fact that we are actually writing JavaScript. TypeScript stays to true to the fact that you are writing JavaScript. Like they say on their website: “It starts with JavaScript and ends with JavaScript” [2]. The website also contains a Playground, where TypeScript is compiled live and you can see the JavaScript unfold as you type, and you can then execute it to watch it in action. Their open source compiler creates plain JavaScript during compilation from the provided JavaScript. It follows the standards presented in ES6 and through examples we will come back to the significance of that and the other advantages TypeScript provides. It has been developed by Microsoft and Lead Designer Anders Hejlsberg in 2012.

ES6

“TypeScript syntax is a superset of ECMAScript 6 (ES6) syntax” [3]

TypeScript tries to follow ES6 and for the moment will monitor and adapt to changes in the ECMA Script which at the moment includes classes, modules and interfaces. Even more importantly TypeScript makes it possible to translate these features into ES3 or ES5 and it therefore back compatible. A new feature in ES6 that seems quite useful is using arrows (lambda) to avoid the problems in using the right “this”. A way of capturing the right “this” is shown on their web page [2] and in the presentation [1]. The TypeScript here is an example of a usage of the lambda/arrow function in ECMAScript 6 [1]:

```
class Tracker {
        count = 0;
    start() {
        windows.onmousemove = e => {
                this.count++;
            console.log(this.count);
         }
    }
}
var t = new Tracker();
t.start();
```

Using the Playground on the web page [2] would also yield a wrong type of this (any), if we did not write the arrow function. Another cool thing you can do in TypeScript with ES6 is defining default values for functions, so if the parameter is not received it will just use the default value. A lot of work in setting up global variables inside classes can be avoided by using the inbuilt public functions, if you declare a constructor for the class. Then it automatically saves the values to be used in the class and makes them accessible, if you have access to the class [1]. It saves you from writing a lot of code. When writing big applications it can be hard to avoid using the same names in different modules, but TypeScript knows which namespace you are referring to with the correct encapsulation [1, 3]. Hejlsberg shows this in the introduction of TypeScript by demonstrating it with the TypeScript compiler (which is also written in TypeScript).

Resulting JavaScript

In the examples presented by Hejlsberg it is clear that TypeScript resolves some of those tedious extra lines of code you have to write in JavaScript if you wish to make your own classes or functions within functions. It we look at the example in Figure 1 it is clear that everyone would prefer the code on the left side (TypeScript) contra the right side (plain JavaScript). We do not really care that it is JavaScript in the end. As a developer we wish to have good looking code that works and JavaScript in the essence is a good thing.

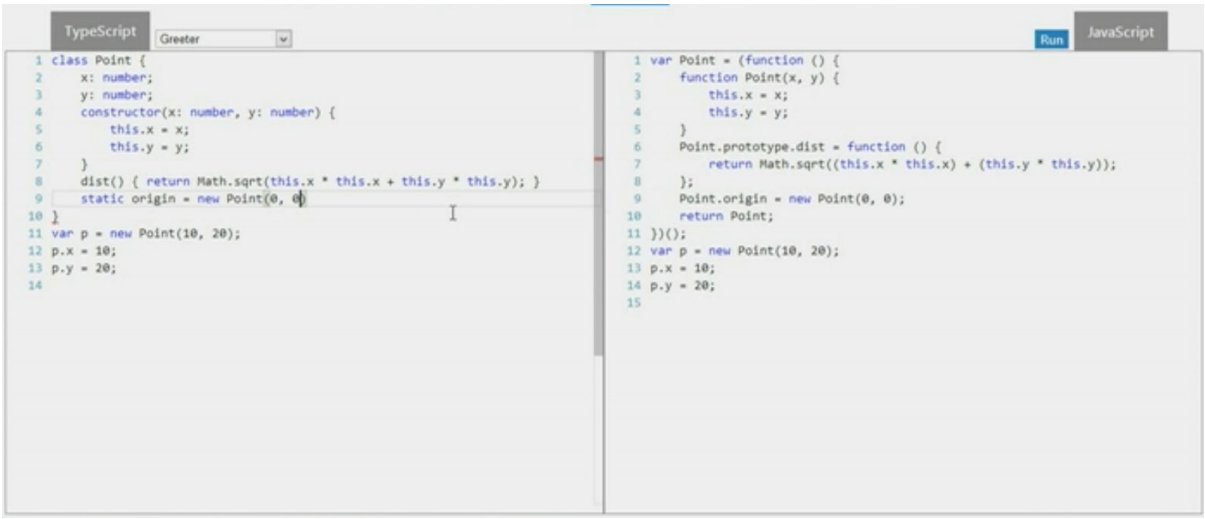

Figure 1: Classes and Functions with TypeScript [1]

Type Checking

Type checking is something that is not included in the regular JavaScript. TypeScript fixes this by allowing you to declare types and the rest of the code will treat the types correctly. This does not change the fact that JavaScript does not have static typing, but again it really only necessary to monitor, as the code is being written to keep the code reliable. A type is declared like this:

```
var height: number = 6;
```

When using something like WebStorm, Eclipse or VisualStudio you would then receive warnings that height is a number, if you are trying to violate that type. It can also be useful to declare return types for a function like this:

```
function add(x: number, y: number): number {
    return x+y;
}
```

The add function here expects two numbers (x and y). It then returns the result of addition between the two. Again if you try to parse a string as x, you would receive and error. And you tried to return something else than a number, you would violate the type as well. If we do not know what type we wish the function or variable to have, we can declare it the type "any". This means that you can use type checking where it is relevant and leave it otherwise (this opt-in/opt-out makes it very situation based and easy to implement).

Types also works for arrays and can be defined in two ways

```
var list:number[] = [1, 2, 3];
```

or with generic array types

```
var list:Array<number> = [1, 2, 3];
```

Again it just makes it easier to handle bigger applications and it also makes a lot more readable without having to write a lot of comments. Everything can be done in-line and in the end save lines of code. Hejlsberg also mention that other tools exists for having comment blocks to describe the typing rules, but it is just more libraries and tools you add to fill up your code space [1]. Using other libraries is also easy in TypeScript as you can make references to other libraries and other files by loading them in your main TypeScript file (.ts):

```
///<reference path="jquery.d.ts" />
require(['jquery'], function ($) {
    $(document).ready(() => {
        alert('Your code executes after jQuery has been loaded.');
    });
});
```

This requires a definition file that is the documentation of the types in JQuery. Some of these files are developed by Microsoft but they are open source, so people can help to update them, and the same goes for other libraries, where people want type checking included. Microsoft has some collaborations with other firms such as AngularJS to keep this definition files up to date.

Conclusion

Generally TypeScript makes JavaScript human-readable. It keeps your code relevant for JavaScript of tomorrow. Type checking makes the code more readable and helps you avoid

errors and spend more time debugging something and modern IDE can tell you as you type. It is one of those layers that actually makes sense without providing more complexity. It can't do all the fancy things we see in AngularJS, but it can be used with AngularJS. It does not exclude working with other tools. As we do not see major changes in web development we have to assume that JavaScript will be here for years to come and more bigger applications will be written for the web and TypeScript is a valuable tool to maintain those.

References

[1] Hejlsberg Introducing TypeScript, https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript, Visited 07/12-2015

[2] TypeScript web page, http://www.typescriptlang.org, Visited 07/12-2015

[3] TypeScript documentation, https://github.com/Microsoft/TypeScript/blob/master/doc/spec.md, Visited 07/12-2015