Volume 6, Issue 5, May 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Issues and Scope in Typescript Language

Supriya Maji, Kanij Fatema Aleya, Madhumita Santra, Asoke Nath
Department of Computer Science, St. Xavier's College (Autonomous)
Kolkata, India

Abstract: ***The most popular web scripting language is JavaScript because all of the browser supports it. It is powerful and flexible language. But it also has some shortcomings. As the complexities of JavaScript applications increases it's important to keep code under control before it spirals in to a mess. For this reason, over the last few years many different web scripting languages have developed and they give the solutions to the shortcomings of JavaScript. One of these types of language is typescript language. The Typescript compiler is itself written in Typescript, compiled to JavaScript. In the present paper the author will discuss the different features of Typescript language.***

Keywords: ECMA, IDE, OOP,

I. INTRODUCTION

JavaScript applications such as web email, documents editing and collaboration tools are becoming an increasingly important part of the everyday computing. Typescript comes to meet the needs of the JavaScript programming teams that build and maintain large JavaScript programs. Typescript is a free and open source programming language developed and maintained by Microsoft. Typescript is a superset of JavaScript. Typescript is an extension of JavaScript intended to enable easier development of large-scale JavaScript applications. While every JavaScript program is a Typescript program, Typescript offers a module system, classes, interfaces, and a rich gradual type system. The in-tension is that Typescript provides a smooth transition for JavaScript programmers. Typescript starts from the same syntax and semantics that millions of JavaScript developers know today. Use existing JavaScript code, incorporate popular JavaScript libraries, and call Typescript code from JavaScript. Typescript compiles to clean, simple JavaScript code which runs on any browser, in Node.js, or in any JavaScript engine that supports ECMAScript 3 (or newer). The Typescript programming language adds optional types to JavaScript, with support for interaction with existing JavaScript libraries via interface declarations. As the name implies, Typescript associates a strongly typed layer in conjunction with JavaScript. Typescript also associates an object oriented layer with JavaScript

II. HISTORY

Typescript was first made public in October 2012 (at version 0.8), after two years of internal development at Microsoft. But criticized the lack of mature IDE support apart from Microsoft Visual Studio, which is not available on Linux and OS X. As of 2013 there is support in other IDEs, particularly in Eclipse The most publicly recognizable name behind Typescript is Microsoft Technical Fellow Anders Hejlsberg, the father of C# and Turbo Pascal.[5] But Hejlsberg isn't the one who came up with the idea for Typescript. Typescript is actually the product of a team of about 50 people, headed by Microsoft Technical Fellow Steve Lucco.

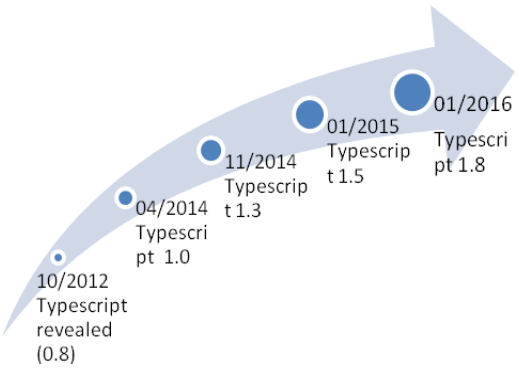

Fig1: Different version of Typescript Language

III. WHAT IS TYPESCRIPT

Typescript is a syntactic sugar for JavaScript. Typescript syntax is a superset of JavaScript. Any existing JS code is the TypeCcript code. Typescript offers optional type annotations and static typing. It has classes, explicit interfaces and easier modules exports. Classes enable programmers to express common object-oriented patterns in a standard way, making features like inheritance more readable and interoperable.

IV. HOW IT WORKS?

Typescript is compiled, rather than interpreted. [5] Typescript has a compiler tsc that is written in Typescript compiles the Typescript code and generates the idiomatic Java-Script that can execute in any host (browser) or in any JavaScript engine.

V. BASIC TYPES

Typescript, support much the same types as we expected in JavaScript. Some of the basic types are numbers, strings, structures, boolean. New addition is enum. The Typescript type system enables programmers to express limits on the capabilities of JavaScript objects, and to use tools that enforce this limits. Without adding any type hinting, variables in Typescript are of the any type, which means they are allowed to contain any type of data.

Let‘s take an example

```
var x: any = 10;
x = "a";
x = true;
```

Typescript take the data type of the variable as” any”. We can assign any value to it or we can pass it to any function.

The basic data types number, string, Boolean are same as JavaScript. Typescript, like JavaScript, allows us to work with arrays of values. Array types can be written in one of two ways. In the first, we use the type of the elements followed by '[]' to denote an array of that element type

A helpful addition to the standard set of data types from JavaScript is the 'enum'. Like languages like C#, an enum is a way of giving more friendly names to sets of numeric values.

```
enum Color {Red, Green, Blue};
var c: Color = Color.Green;
```

By default, enums begin numbering their members starting at 0. We can change this by manually setting the value of one its members. For example, we can start the previous example at 1 instead of 0:

```
enum Color {Red = 1, Green, Blue};
var c: Color = Color.Green;
```

VI. TYPESCRIPT AND OBJECT ORIENTED PROGRAMING

There are four main principles to Object Oriented Programming: Encapsulation, Inheritance, Abstraction, and Polymorphism. Typescript can implement all four of them with its smaller and cleaner syntax.

A. Core Object

There are many objects that are the part of its core for example there are objects like Math, Object, Array, and String .Lets take an example

```
alert(Math.random());
```

B. Classes

Traditional JavaScript focuses on functions and prototype-based inheritance as the basic means of building up reusable components. It has no class. It just has objects whose blueprints are that of a dictionary of data and functions. When a new object is created in JavaScript, it has an empty dictionary user can fill with anything. In Typescript, it allows this technique and compiles them down to JavaScript that works across all major browser and platforms. Three main causes where we need class regardless of the language are

- Creating multiple new instances
- Using inheritance
- Singleton objects

The reason Typescript’s classes are valuable is not because we can’t do them today, it’s because they make it so much easier to do these things.

Class members:

- Properties and fields to store data
- Methods to define behaviour
- Events to provide interactions between different objects .Lets take an example [2]

```
classBankAccount
{
        balance = 0;
        deposit(credit: number)
        {
                this.balance += credit;
                returnthis.balance;
        }
}
```


C. Inheritence

In Typescript, we can use common object-oriented patterns. Of course, one of the most fundamental patterns in class-based programming is inheritance. Through inheritance, one object can inherit the characteristics of another object; this allows an existing object to be extended and similar objects to share properties and behaviours. In Typescript inheritance is achieved by simply using the ―extend keyword to extend from the base class.

```
class Animal {
name:string;
constructor(theName: string) { this.name = theName; }
move(meters: number = 0) {
alert(this.name + " moved " + meters + "m.");
   }
}
class Snake extends Animal {
constructor(name: string) { super(name); }
move(meters = 5) {
alert("Slithering...");
super.move(meters);
   }
}
```

This example covers quite a bit of the inheritance features in Typescript that are common to other languages. Here we see using the 'extends' keywords to create a subclass. You can see this where ‘Snake’ is the subclass of the base class 'Animal' and gain access to its features. The example also shows off being able to override methods in the base class with methods that are specialized for the subclass. Here ‘Snake’ create a 'move' method that overrides the 'move' from 'Animal', giving it functionality specific to each class.

D. Interface

Interfaces define a set of properties, methods, and events but do not provide their implementation. We cannot instantiate an interface as we can a class. Classes inherit from interfaces and must implement each item in the interface exactly as defined. For example; we could define an interface for a Car class such that every car must have an engine and a colour like this.

```
interfaceICar{
engine: string;
color: string;
}
class Car implements ICar {
constructor (public engine: string, public color: string) {
   }
}
```

The Car class adheres to the interface ICar because it implements ICar[4]

VII. WHY WE MOVE FROM JAVASCRIPT TO TYPESCRIPT

A. No Module

Module development means separating the functionality of a program into independent parts. In JavaScript there is no import statement so no name space. Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the export form .So to consume a variable a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the import forms.

There are other benefits to using modules

- Scoping of variables (out of global scope)
- Code re-use
- Encapsulation
- Don't Repeat Yourself
- Easier for testing

  Implicit Internal Module:

  Implicit internal modules are the easiest to work with because we simply just write source code without worrying about module

```
classTestClass
 {
private a = 2;
public b = 4;
};
var t = new TestClass();
```

  The entire set of code is in an implicit module that joins the global module / global scope.

When we use the module keyword we are using (named) internal modules. All variables defined within the module are scoped to the module and removed from the global scope.

```
module Shapes
 {
          class Rectangle
           {
                    constructor ( public height: number,  public width: number)
                     {
                    }
           }
   // This works!
var rect1 = new Rectangle(10, 4);
}
// this won't!!
var rect2 = Shapes._________
```

In this example it is clear that the code below shows that the rectangle class and the rect variable are both scoped to the Shapes module. They cannot be accessed outside the class.

Exports

If We could take that previous example and export what you want out of the module. In this example, Rectangle is exported from the module and accessible to code outside of the module.

```
module Shapes {
export class Rectangle {
constructor (
public height: number,
public width: number) {
   }
   }
}
// This works!
varrect = Shapes.Rectangle(10, 4);
```

import

Importing is just about as easy as exporting from a module. Let there is a class Zip Code Validator. Importing an exported declaration is done through using one of the import forms below:

```
import { ZipCodeValidator } from "./ZipCodeValidator";
letmyValidator = new ZipCodeValidator();
```

B. No Visibility Control

In JavaScript access modifier are not there, no private, protect and public modifiers like in Java that helps in hiding, reduce dependencies, understanding which code belongs to which class, and less risk of accidental side-effects etc. it helps in modularity as well. In Typescript, class members are public by default but can be made private by including the private access modifier. And it is enforced by the compiler so you get some static checking about this organization and you do not have to manage by yourself. The access modifiers increase security of the class members and prevent them to be invalid use or miss use.

If no access modifier to be set, Typescript sets public access modifier to all class members (functions or properties) by default and everything in a module is private unless export keyword is used.

- Private members are only accessible within their declaring class.
- Public members can be accessed anywhere.

C. Type System

In JavaScript there is no type system. While defining a variable, it is not necessary to specify the type of the variable. In JavaScript it does not have type system, therefore it is hard to pick up the other peoples code and look at the interface and it is difficult to see what the functions do. Typescript doesn't have many built-in data types you can use to declare variables—just string, number and Boolean. The type system in typescript is designed to be optional so that your JavaScript is typescript. Typescript does not block JavaScript emit in the presence of Type Errors, allowing to progressively update JS to TS.

D. No Static Checking

Its main goals are to perform type checking at compile time and to support the object-oriented paradigm. In Typescript, we can use types to annotate static parts of the code. This enables advanced editing, refactoring, and early error detection, which improves the dev experience in the large, without losing what is great about JavaScript.

Typescript compiler will check the type (to surface more typing errors at compiling time)

```
var name: string;
name = 2; // type error, assign a number to a string type variable
```



```
function foo(value: number) {}
foo('); // type error, use a number as a string type parameter
```

E. Maintainability Is Hard

JavaScript is increasingly used to develop the large and complex application. The problem is maintainability of such application because it should have a well-defined structure. JavaScript has features that make the development of large applications somewhat more challenging. For example, JavaScript has no built-in module system. This is a particular annoyance for programs. There is no boundary between one module to other. So maintainability is hard for large system.

VIII. CONCLUSION AND FUTURE SCOPE

There are many pitfalls and ugly sides of JavaScript such as development and maintainability of large scale complex applications etc. so There is a huge need for JavaScript improvements so many of the scripting language came. One of those scripting language is Typescript. It is also easy to learn .It support OOP features. Typescript is just a tool on top of JavaScript which provides us with more robust code structure, type safety. Typescript is included as a first-class programming language in Microsoft Visual Studio 2013 Update 2 and later, beside C# and other Microsoft languages.

REFERENCES

[1] TypeScript Classes& Object-Oriented Programming code Belt www.codebelt.com com [May 5, 2016.]
[2] Typescript Revealed by Dan Maharry
[3] www.typescriptlang.org/Handbook [on May 9 2016]
[4] https://johnpapa.net/typescriptpost3/
[5] AANSA ALI EVALUATION OF ALTERNATE PROGRAMMING LANGUAGES TO JAVASCRIPT Masters of Science Thesis
[6] https://davefancher.com/2014/07/11/typescript-bringing-sanity-to-javascript/

