ISSN(Online): 2320 - 9801
ISSN (Print) : 2320 - 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

Application of TypeScript Language: A Brief Overview

Saptarshi Bhattacharyya[*1], Asoke Nath[2]

Research Associate, Dept. of Computer Science, St. Xavier's College, Kolkata, India[1]

Associate Professor, Dept. of Computer Science, St. Xavier's College, Kolkata, India[2]

ABSTRACT: TypeScript Compiler interprets any TypeScript programs and generates output which is JavaScript. TypeScript Language has tremendous applications in Internet Webpage designing. In the present paper the authors have given thorough study on scope, applications and challenges of TypeScript language. The goal of this paper is to capture the essence of TypeScript by giving a precise definition of this type system on a core set of constructs of the language. JavaScript remains a poor language for developing and maintaining large applications. TypeScript is an extension of JavaScript intended to address this deficiency. The JavaScript generated by TypeScript language can immediately run in a huge range of execution environments. The compiler is used extensively in Microsoft to author significant JavaScript applications.

KEYWORDS: TypeScript, JavaScript, Transpilation, Transpiler, CoffeeScript.

I. INTRODUCTION

TypeScript is a free and open source programming language developed and maintained by Microsoft. It is a strict superset of JavaScript.JavaScript remains a poor language for developing and maintaining large applications. TypeScript is an extension of JavaScript intended to address this deficiency.
TypeScript is an extension of JavaScript intended to enable easier development of large-scale JavaScript applications. TypeScript adds optional static typing and class-based object-oriented programming to the java script language. The TypeScript programming language adds optional types to JavaScript, with support for interaction with existing JavaScript libraries via interface declarations.
While every JavaScript program is a TypeScript program, TypeScript offers a module system, classes, interfaces, and a rich gradual type system. The intention is that TypeScript provides a smooth transition for JavaScript programmers—well-established JavaScript programming idioms are supported without any major rewriting or annotations. One interesting consequence is that the TypeScript type system is not statically sound by design.

II. LITERATURE SURVEY

JavaScript's flexible semantics makes writing correct code hard and writing secure code extremely difficult. To address the former problem, various forms of gradual typing have been proposed, such as Closure and TypeScript. The TypeScript programming language adds optional types to JavaScript, with support for interaction with existing JavaScript libraries via interface declarations. Such declarations have been written for hundreds of libraries, but they can be difficult to write and often contain errors, which may affect the type checking and misguide code completion for the application code in IDEs. However, supporting all common programming idioms is not easy; for example, TypeScript deliberately gives up type soundness for programming convenience. In this paper, we propose various applications of gradual type system and implementation techniques that provide important safety and security guarantees. Typescript is compiled rather than interpreted. 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 java script engine.

ISSN(Online): 2320 - 9801
ISSN (Print) : 2320 - 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

III. THOROUGH STUDY

The most popular web scripting language is JavaScript because all of the browser supports it. It is powerful and flexible language. But it has also 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 authors have made through study on scope and challenges of Typescript language.

❖ TypeScript

- Paradigm: Multi-paradigm: scripting, object-oriented, structured, imperative, functional, and generic.
- Designed by: MICROSOFT
- Developed by: MICROSOFT
- First appeared: October 1,2012
- License: Apache License 2.0
- Filename extension:.ts
- Influenced by: JavaScript, Java, C#
- Influenced: AtScript.

In the present paper the authors have made special attention on followings:

A. Aim / Motto:
TypeScript enriches JavaScript with a module system, classes, interfaces, and a static type system. As TypeScript aims to provide lightweight assistance to programmers, the module system and the type system are flexible and easy to use. In particular, they support many common JavaScript programming practices. They also enable tooling and IDE experiences previously associated with languages such as C# and Java.

B. Relation between Typescript and Javascript:
TypeScript is a syntactic sugar for JavaScript. TypeScript is a compiled language, it's not interpreted at run-time. The TypeScript compiler takes TypeScript files (.ts) and compiles them in to JavaScript files (.js).TypeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, TypeScript code has to be translated into JavaScript. This operation is referred to as Transpilation and the tools that perform it are called Transpiler.

C. Live Applications:
For example, recently Microsoft gave details of two substantial Typescript projects:
1. Monaco, an online code editor, which is around 225kloc, and
2. Xbox Music, a music service, which is around 160kloc.designations.

IV. CHARACTERISTICS OF TYPESCRIPT LANGUAGE

TypeScript originated from the perceived shortcomings of JavaScript for the development of large-scale applications.

1. Type annotations
   Type signature or type annotation defines the inputs and outputs for a function, subroutine or method. A type signature includes the function's return type, the number of arguments, the types of arguments, or errors it may pass back.
2. compile-time type checking
   During compilation the typescript language compiler checks the data type.
3. Type inference
   Type inference refers to the automatic deduction of the data type of an expression in a programming language.
4. Type erasure
   In programming languages, type erasure refers to the compile-time process by which explicit type annotations are removed from a program, before it is executed at run-time. Operational semantics that do not require programs to

ISSN(Online): 2320 - 9801
ISSN (Print) : 2320 - 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

be accompanied by types are called type-erasure semantics, to be contrasted with type-passing semantics. The possibility of giving type-erasure semantics is a kind of abstraction principle, ensuring that the run-time execution of a program does not depend on type information. In the context of generic programming, the opposite of type erasure is called reification.

5. Interfaces
   In object-oriented programming, a protocol or interface is a common means for unrelated objects to communicate with each other. These are definitions of methods and values which the objects agree upon in order to co-operate.
6. Enumerated type
   In computer programming, an enumerated type (also called enumeration or enum, or factor in the R programming language, and a categorical variable in statistics) is a data type consisting of a set of named values called elements, members or enumerators of the type. The enumerator names are usually identifiers that behave as constants in the language. A variable that has been declared as having an enumerated type can be assigned any of the enumerators as a value. In other words, an enumerated type has values that are different from each other, and that can be compared and assigned, but which are not specified by the programmer as having any particular concrete representation in the computer's memory; compilers and interpreters can represent them arbitrarily.
7. Mixin
   In object-oriented programming languages, a mixin is a class that contains a combination of methods from other classes. How such a combination is done depends on the language. If a combination contains all methods of combined classes, it is equivalent to multiple inheritance. Mixins are sometimes described as being "included" rather than "inherited". Mixins encourage code reuse and can be used to avoid the inheritance ambiguity that multiple inheritance can cause (the "diamond problem"), or to work around lack of support for multiple inheritance in a language. A mixin can also be viewed as an interface with implemented methods.
8. Generic
   Generic programming is a style of computer programming in which algorithms are written in terms of types to-be-specified-later that are then instantiated when needed for specific types provided as parameters.
9. Namespaces
   Namespace is a set of symbols that are used to organize objects of various kinds, so that these objects may be referred to by name. Prominent examples include: File systems are namespaces that assign names to files. Programming languages organize their variables and subroutines in namespaces.Computer networks and distributed systems assign names to resources, such as computers, printers, websites, (remote) files, etc.
10. Tuple
    A tuple is a finite ordered list of elements. In mathematics, an n-tuple is a sequence (or ordered list) of n elements, where n is a non-negative integer. There is only one 0-tuple, an empty sequence. An n-tuple is defined inductively using the construction of an ordered pair. Tuples are usually written.

V. TYPESCRIPT TYPES

The TypeScript programming language adds optional types to JavaScript, with support for interaction with existing JavaScript libraries via interface declarations.

The list of primitive types available in TypeScript

1 Number: the “number” is a primitive number type in TypeScript. There is no different type for float or double in TypeScript
2 Boolean: The “boolean” type represents true or false condition
3 String: The “string” represent sequence of characters similar to C#
4 Null: The “null” is a special type which assigns null value to a variable
5 Undefined: The “undefined” is also a special type and can be assigned to any variable.

VI. TYPESCRIPT EXAMPLES

- Example 1:
  The following is the source code for a simple calculator written in TypeScript:
  function add(arg1: number, arg2: number):
  number

ISSN(Online): 2320 - 9801
ISSN (Print) : 2320 - 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

```
{
   return arg1 + arg2;
}
function subtract(arg1: number, arg2: number):
number
{
   return arg1 - arg2;
}
function multiply(arg1: number, arg2: number):
number
{
   return arg1 * arg2;
}
function divide(arg1: number, arg2: number):
number
{
   return arg1 / arg2;
}
```

- Example 2:
  In the example below we have a piece of code in the TypeScript. It contains two classes: Person and Main. The first one has a constructor with an optional name parameter and a public method getName(). The other class Main uses a Person class and initiates two instances of it. Pay attention to the static Person.defaultName member, which is used if Person's constructor is named without a name parameter.

```
Class person
{
  private static default name: string=”Saptarshi”;
  private name: string = null;
  constructor(name: string = null)
  {
            if(name)
            {
                  this.name = name;
            }
            else
            {
                  this.name = person.defaultName;
            }
  }

  public getName(): string
  {
     return this.name;
  }
}

Class main
{
  private unknown: person;
  private tim: person;
  private tom: person;
  constructor()
  {
            this.unknown = new person();
```

ISSN(Online): 2320 - 9801
ISSN (Print) : 2320 - 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

```
            this.tim = new person("Tim");
            this.tom = new person("Tom");

            console.log(this.tim.getName());
            console.log(this.tom.getName());
    }
}
```

- Example 3:

  TypeScript does not try to infer types for the parameters of a function expression.

```
var fact = function (x)
{
        if (x == 0) { return 1; }
        else { return x * fact(x - 1); }
}; // infers type { (x:any): number }
```

  Both rules run into an "awful" inherited from JavaScript: all variables declared in a function body are in scope regardless of nesting levels, order, or even how many times they are declared. The whole function body (except for functions nested inside it) is treated as a flat declaration space. In other words, JavaScript does not have block scoping.

  Thus the following (buggy) JavaScript code:

```
var scope = function (p)
{
   var y = 1;
   var a = [y, x, z];
   var x = 2;
   if (test(p)) { var z = 3; }
   else { var z = 4; var w = 5; }
   return w + a[2];
};
```

  is treated as if it had instead been written as follows.

```
var scope = function (p)
{
   var y = 1;
   var x; var z; var w; // implicit
   var a = [y, x, z];
   var x = 2;
   if (test(p)) { var z = 3; }
   else { var z = 4; var w = 5; }
   return w + a[2];
};
```

  At the level of typing this means that when typing a function expression, we need two phases: first, we find the types of all the local variables declared in the function body; and second, we then type the function body using the type environment extended with the types determined from the first phase. There is a further complication as TypeScript also infers types of local variables with initializers. Furthermore, TypeScript, again following JavaScript, supports mutually recursive variable declarations

ISSN(Online): 2320 - 9801
ISSN (Print) : 2320 - 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

VII. COMPARISON

The TypeScript compiler checks TypeScript programs and emits JavaScript.So the programs can immediately run in a huge range of execution environments CoffeeScript makes a lot of changes to JavaScript's syntax and uses JavaScript's object-oriented capabilities in a particular way, but doesn't fundamentally alter the way the language works.
There's also Dartfrom Google, but that's a full on replacement for JavaScript. Dart's goals aren't identical to TypeScript's. Google thinks that some of the semantics of JavaScript are fundamentally problematic, so Dart changes these semantics.
However, both Dart and TypeScript are designed to aid in large-scale application development using module systems and optional static type checking as the fundamental tools to support that, and both can be compiled to JavaScript.

VIII. CONCLUSIONS AND FUTURE SCOPE

This paper describes and analyses the core of the TypeScript language, and in particular its type system. The work that it represents has been useful in resolving ambiguities in the language definition, and in identifying minor unintended inconsistencies and mistakes in the language implementation. It provides a basis for partial soundness theorems, and it isolates and accounts for sources of unsoundness in the type system. Beyond the details of this work (which are specific to TypeScript, and which may perhaps change, as TypeScript develops further), we hope that our results will contribute to the principled study of deliberate unsoundness. In this direction, we believe that there are various opportunities for intriguing further research.

In particular, to the extent that any type system expresses programmer intent, we would expect that it could be useful in debugging, despite its unsoundness. Research on blame might be helpful in this respect. It may also be worthwhile to codify programmer guidance that would, over time, reduce the reliance on dangerous typing rules. Static analysis tools may support this guidance and complement a type system. These and related projects would aim to look beyond sound language fragments: the principles of programming languages may also help us understand and live with unsoundness.

ACKNOWLEDGMENT

The authors sincerely express their gratitude to Department of Computer Science for providing necessary help and assistance. The authors are very much grateful to Fr. Dr. John Felix Raj, Principal of St. Xavier's College(Autonomous), kolkata for giving opportunity to work in the field of Typescript Language.

REFERENCES

[1] Microsoft Corporation. TypeScript Language Specification, 0.9.5 edition, 2014. http://typescriptlang.orgS. M. Metev and V. P. Veiko, Laser Assisted Microtechnology, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.
[2] M. Abadi and L. Cardelli. A theory of objects. Springer Verlag, 1996.
[3] C. Anderson, P. Giannini, and S. Drossopoulou. Towards type inference for JavaScript. In Proceedings of ECOOP, 2005.
[4] G. Bierman, M. Parkinson, and A. Pitts. MJ: An imperative core calculus for Java and Java with effects. Technical Report 563, University of Cambridge Computer Laboratory, 2003.
[5] R. Chugh, D. Herman, and R. Jhala. Dependent types for JavaScript. In Proceedings of OOSLA, 2012. 5. D. Crockford. JavaScript: The good parts. O'Reilly, 2008.
[6] V. Gapeyev, M. Levin, and B. Pierce. Recursive subtyping revealed. JFP, 12(6):511–548, 2002.
[7] P. Gardner, S. Maffeis, and G. Smith. Towards a program logic for JavaScript. In Proceedings of POPL, 2013.
[8] Google. Dart programming language. http://www.dartlang.org.
[9] A. Guha, C. Saftoiu, and S. Krisnamurthi. Typing local control and state using flow analysis. In Proceedings of ESOP, 2011.
[10] A. Igarashi, B. Pierce, and P. Wadler. Featherweight Java: A minimal core calculus for Java and GJ. ACM TOPLAS, 23(3):396–450, 2001. 11.

BIOGRAPHY

Saptarshi Bhattacharyya is a Research Assistant in the Department Of Computer Science, at St. Xavier's College (Autonomous), Kolkata. He has immense interest in research, and has contributed considerably in research endeavours. His major research interest areas are Data Structure, Visual Cryptography, Data Hiding, Image Processing, Green Computing, various computer languages etc..

Dr. Asoke Nath is an Associate Professor in Department of Computer Science, St. Xavier's College (Autonomous), Kolkata. His major research areas are Cryptography and Network Security, Visual Cryptography, Data Hiding, Image Processing, Green Computing, e-learning, Li-fi Technology, Cognitive Radio, Mathematical modelling of Social Networks, MOOCs and many more. He is the life member of MIR LABS(USA) and CSI Kolkata Chapter.