# 浅谈TypeScript语法

## Angular是用TypeScript构建的

TypeScript并不是一门全新的语言，而是ES6的超集。所有的ES6代码都是完全有效且可编译的TypeScript代码。



说明一下：什么是ES5？什么是ES6？ES5是ECMAScript5的缩写，也被称为“普通的JavaScript”。ES5就是大家熟知的JavaScript，它能够运行在大部分浏览器上。ES6则是下一个版本的JavaScript。

现如今支持ES6的浏览器还比较少，更不用说TypeScript了，我们用转译器来解决这个问题。TypeScript转译器能把TypeScript代码转换为几乎所有浏览器都支持的ES5代码。

TypeScript是Microsoft和Google之间的官方合作项目。有这两家强有力的科技巨头在背后支撑，对于我们来说是个好消息，因为这表示TypeScript将会得到长期的支持。这两家公司都承诺全力推动Web技术的发展，我们这些开发人员显然会受益匪浅。

需要指出的是：TypeScript并不是开发Angular应用的必选语言。我们同样可以使用ES5代码（即“普通”JavaScript）来开发Angular应用。Angular也为全部功能提供了ES5 API。那么为什么我们还要使用TypeScript呢？这是因为TypeScript有不少强大的功能，能极大简化开发。

## TypeScript提供了哪些特性

TypeScript相对于ES5有五大改善：

* 类型
* 类
* 注解
* 模块导入
* 语言工具包（比如，结构）

接下来我们逐个介绍。

## 类型

相对于ES6，TypeScript最大的改善是增加了类型系统。

有些人可能会觉得，缺乏类型检查正是JavaScript这些弱类型语言的优点。但是对于类型检查，也有诸多好处：

1. 有助于代码的编写，因为它可以在编译期预防bug：
2. 有助于代码的阅读，因为它能清晰地表明你的意图。

另外值得一提的是，TypeScript中的类型是可选的。如果希望写一些快速代码或功能原型，可以首先忽略类型，然后再随着代码日趋成熟逐渐加上类型。

TypeScript的基本类型与我们平时所写JavaScript代码中用的隐式类型一样，包括字符串、数字、布尔值等。

直到ES5，我们都在用var关键字定义变量，比如var name;。

TypeScript的新语法是从ES5自然演化而来的，仍沿用var来定义变量，但现在可以同时为变量名提供可选的变量类型了：

var name : string;

在声明函数时，也可以为函数参数和返回值指定类型：

function greetText(name : string) : string {

return “Hello” + name;

}

上述函数的定义和java，c语法都不太一样，但对于编程人员，应该都能看懂，只是一个习惯问题。

## 尝试REPL

REPL(Read Eval Print Loop:交互式解释器),表示一个电脑的环境，类似 Window 系统的终端或 Unix/Linux shell，我们可以在终端中输入命令，并接收系统的响应。

为了运行本章中的例子，我们可以先安装一个小工具，名为TSUN(TypeScript Upgraded Node，支持TypeScript的升级版Node）：

$ npm install -g tsun

在安装这个工具之前，一定要先安装TypeScript环境：npm install -g typescript

接着启动它：

$ tsun

TSUN : TypeScript Upgraded Node

type in TypeScript expression to evaluate

type :help for commands in repl

>

这个小小的>是一个命令提示符，表示TSUN已经准备好接收命令了。

在这个终端中，可以运行一些TypeScript的例子，去开启我们语法知识的学习之路。

对于node.js的安装和npm的使用，可以参照我的这两篇博客：

<http://blog.csdn.net/csdn_ds/article/details/72625928>

<http://blog.csdn.net/csdn_ds/article/details/72626839>

## 内置对象

### 字符串

字符串包含文本，声明为string类型：

var username: string = ‘张三’;

### 数字

无论整数还是浮点，任何类型的数字都属于number类型。在TypeScript中，所有的数字都是用浮点数表示的，这些数字的类型就是number：

var age ：number =36;

### 布尔类型

布尔类型（boolean）以true（真）和false（假）为值。

var married : boolean =true;

### 数组

数组用Array类型表示，然而，因为数组是一组相同类型的集合，所以我们还需要为数组中的条目指定一个类型。

我们可以用Array<type>或者type[]语法来为数组条目指定元素类型：

var jobs : Array<string> = ['IBM','Miscrosoft','Google'];

var jobs:string[] = ['Apple','Dell','HP'];

数字型数组的声明与之类似：

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

var jobs : number[] = [4,5,6];

### 枚举

枚举是一组可命名数值的集合。比如，如果我们想拿到某人的一系列角色，可以这样写：

enum Role {Employee,Manager,Admin};

var role : Role =Role.Employee;

默认情况下,枚举类型的初始值是0。我们也可以调整初始值的范围：

enum Role {Employee = 3,Manager,Admin};

var role : Role =Role.Employee;

在上面的代码，Employee的初始值被设置为3而不是0。枚举中其他项的值是依次递增的，意味着Manager的值为4，Admin的值为5。同样，我们也可以单独为枚举中的每一项指定值：

enum Role {Employee = 3,Manager = 5,Admin = 7};

var role : Role =Role.Employee;

还可以从枚举的值来反查它的名字：

enum Role {Employee,Manager,Admin};

console.log('Role:',Role[0],',',Role[1],'and',Role[2]);

### 任意类型

如果我们没有为变量指定类型，那它的默认类型就是any。在TypeScript中，any类型的变量能够接收任意类型的数据：

var something : any ='as thing';

something =1;

something = [1,2,3];

### “无”类型

void意味着我们不期望那里有类型。它通常用作函数的返回值，表示没有任何返回值：

function setName(name:string):void{

this.name=name;

}

## 类

JavaScript ES5采用的是基于原型的面向对象设计。这种设计模型不使用类，而是依赖于原型。

JavaScript社区采纳了大量最佳实践，以弥补JavaScript缺少类的问题。在ES6中，我们终于有内置的类了。

用class关键字来定义一个类，紧随其后的是类名和类的代码块：

class Vehicle{

}

类可以包含属性、方法以及构造函数。

### 属性

属性定义了类实例对象的数据。比如名叫Person的类可能有first\_name、last\_name和age属性。

类中的每个属性都可以包含一个可选的类型。比如，我们可以把first\_name和last\_name声明为字符串类型（string），把age声明为数字类型（number）。

Pserson类的声明是这样的：

class Person {

first\_name : string;

last\_name : string;

age : number;

}

### 方法

方法是运行在类对象实例上下文中的函数。再调用对象的方法之前，必须要有这个对象的实例。要实例化一个类，我们使用new关键字，比如new Person()会创建一个Person类的实例对象。

如果我们希望问候某个Person，就可以这样写：

class Person {

first\_name : string;

last\_name : string;

age : number;

greet(){

console.log('hello',this.first\_name);

}

}

注意，借助this关键字，我们能用this.first\_name表达式来访问Person类的first\_name属性。

如果没有显式声明过方法的返回类型和返回值，就会假定它可能返回任何东西（即any类型）。然而，因为这里没有任何显式的return语句，所以实际返回的类型是void。

注意：void类型也是一种合法的any类型。

实例化方法：var p : Person = new Person();

假如我们希望Person类有一个带返回值的方法。比如，要获取某个Person在数年后的年龄，我们可以这样写：

class Person {

first\_name : string;

last\_name : string;

age : number;

greet(){

console.log(‘Hello’,this.first\_name);

}

ageInYears(years:number):number{

return this.age + years;

}

}

//instantiate a new Person instance

var p:Person = new Person();

//set initial age

p.age=6;

//how old will he be in 10 year?

p.ageInYears(10);

//->16

### 构造函数

构造函数是当类进行实例化时执行的特殊函数。通常会在构造函数中对新对象进行初始化工作。

构造函数必须命名为constructor。因为构造函数是在类被实例化时调用的，所以它们可以有输入参数，但不能由任何返回值。

当类没有显式地定义构造函数时，将自动创建一个无参构造函数：

class Vehicle{

}

var v =new Vehicle();

它等价于：

class Vehicle{

constructor();

}

var v =new Vehicle();

说明一下：在TypeScript中，每个类只能有一个构造函数。这是违背ES6标准的。在ES6中，一个类可以拥有不同参数数量的多个构造函数重载实现。

我们可以使用带参数的构造函数来将对象的创建工作参数化。

比如，我们可以对Person类使用构造函数来初始化它的数据：

class Person {

first\_name : string;

last\_name : string;

age : number;

constructor(first\_name:string,last\_name:string,age:number){

this.first\_name = first\_name;

this.last\_name = last\_name;

this.age = age;

}

greet(){

console.log(‘Hello’,this.first\_name);

}

ageInYears(years:number):number{

return this.age + years;

}

}

用下面的这种方法重写前面的例子要容易些：

var p:Person = new Person('Felipe','Coury',36);

p.greet();

当创建这个对象的时候，其姓名、年龄都会被初始化。

### 继承

面向对象的另一个重要特性就是继承。继承表明子类能够从父类得到它的行为。然后，我们就可以在这个子类中重写、修改以及添加行为。

TypeScript是完全支持继承特性的，并不像ES5那样要靠原型链实现。继承是TypeScript的核心语法，用extends关键字实现。

要说明这一点，我们来创建一个Report类：

class Report{

data:Array<string>;

constructor(data:Array<string>){

this.data = data;

}

run(){

this.data.forEach(function(line){console.log(line);});

}

}

这个Report类有一个字符串数组类型的data的属性。当我们调用run方法时，它会循环这个data数组中的每一项数据，然后用console.log打印出来。

给RePort增加几行数据，并调用run把这些数据打印到控制台：

var r:Report = new Report(['First line','Second line']);

r.run();

运行结果如下：

First line

Second line

现在，假设我们希望有第二个报表，它需要增加一些头信息和数据，但我们仍想复用现有Report类的run方法来向用户展示数据。

为了复用Report类的行为，要使用extends关键字来继承它：

class TabbedReport extends Report {

headers : Array<string>;

constructor(headers:string[],values:string[]){

super(values);

this.headers = headers;

}

run(){

console.log(this.headers);

super.run();

}

}

var headers : string[] = ['Name'];

var data : string[] = ['Alice Green','Paul Pfifer','Louis Blakenship'];

var tr : TabbedReport = new TabbedReport(headers,data);

tr.run();

执行结果如下：

[ 'Name' ]

Alice Green

Paul Pfifer

Louis Blakenship

## 工具

ES6和TypeScript提供了许多语法特性，让编码成为一种享受。其中最重要的两点是：

* 胖箭头函数语法
* 模板字符串

### 胖箭头函数

胖箭头（=>）函数是一种快速书写函数的简洁语法。

在ES5中，每当我们要用函数作为方法参数时，都必须用function关键字和紧随其后的花括号（{}）表示，就像这样:

//ES5-like example

var data : string[] = ['Alice Green','Paul Pfifer','Louis Blakenship'];

data.forEach(function(line){console.log(line);});

现在我们可以用=>语法来重写它了：

//Typescript example

var data : string[] = ['Alice Green','Paul Pfifer','Louis Blakenship'];

data.forEach((line)=>console.log(line));

当只有一个参数时，圆括号可以省略。箭头（=>）语法可以用作表达式：

var events = [2,4,6,8];

var odds = events.map(v => v+1);

也可以用作语句：

data.forEach(line => console.log(line.toUpperCase()));

=>语法还有一个重要的特性，就是它和环绕它的外部代码共享同一个this。这是它和普通function写法最重要的不同点。通常，我们用function声明的函数有它自己的this。有时候在JavaScript中能看见如下代码：

var nate = {

name:'Nate',

guitars:['Gibson','Martin','Taylor'],

printGuitars:function(){

var self =this;

this.guitars.forEach(function(g){

console.log(self.name +' play a '+g);

});

}

};

由于胖箭头会共享环绕它的外部代码的this，我们可以这样改写：

var nate = {

name:'Nate',

guitars:['Gibson','Martin','Taylor'],

printGuitars:function(){

this.guitars.forEach((g) => {

console.log(this.name +' play a '+g);

});

}

};

可见，胖箭头函数是处理内联函数的好办法。这也让我们在JavaScript中更容易使用高阶函数。

### 模板字符串

ES6引入了新的模板字符串语法，它有两大优势：

1. 可以在模板字符串中使用变量（不必被迫使用+来拼接字符串）
2. 支持多行字符串
3. 字符串中的变量

这种特性也叫字符串插值（stirng interpolation）。你可以在字符串中插入变量，做法如下：

var firstName = "Nate";

var lastName = "Murray";

var greeting = `Hello ${firstName} ${lastName}`;

console.log(greeting);

注意，字符串插值必须使用反引号，不能使用单引号或双引号。

1. 多行字符串

反引号字符串的另一个优点是允许多行文本：

var template = `

<div>

<h1>Hello</h1>

<p>This is a great website</p>

</div>

`;

用tsun输出template，结果如下（带有格式的字符串）：

'\n<div>\n\t<h1>Hello</h1>\n\t<p>This is a great website</p>\n</div>\n'

当我们要插入模板这样的长文本字符串时，多行字符串会非常有帮助。

## 总结

在TypeScript和ES6中还有很多其它的优秀语法特性，如：

* 接口
* 泛型
* 模块的导入、导出
* 标注
* 解构

如果大家对TypeScript以及Angular感兴趣的话，可以继续深入学习TypeScript的语法，笔者很是看好它们的未来，将后端的思想融入到前端，让前端也拥有了框架，设计模式，模块等概念，一门语言或者思想好不好，实践最重要，如果经得起时间和项目的考验，沉淀下来的肯定都是值得我们拥有的。