关于Typescript的一些分享(进阶)

TypeScript是什么

> TypeScript
> 扩展了JavaScript语法，任何已经存在的JavaScript程序，可以不加任何改动，在TypeScript环境下运行。TypeScript只是向JavaScript添加了一些新的遵循ES6规范的语法，以及基于类的面向对象编程的这种特性。
> 其次，2016年9月底发布的Angular2框架，这个框架本身是由TypeScript编写的。Angular框架，大家都知道，它是由谷歌公司开发的，非常流行的框架。也就是说，现在TS这门语言是由微软和谷歌这两大公司在背后支持。

总结:

- 微软开发的一门编程语言
- JavaScript的超集
- 遵循最新的ES6规范

TypeScript优势?为什么会有TypeScript?

1. 解决传统JavaScript的缺陷:

> JavaScript 只是一个脚本语言，并非设计用于开发大型 Web 应用，JavaScript 没有提供类和模块的概念，而 TypeScript 扩展了 JavaScript 实现了这些特性。

2. 更好的,提前的debugger:

> 基于静态类型，用 TypeScript 编辑代码有更高的预测性，更易纠错(体现在哪?预编译阶段在编辑器上有下划线提示)。
> • 由于模块，命名空间和强大的面向对象编程支持，使构建大型复杂应用程序的代码库更加容易。
> • TypeScript在编译为JavaScript的过程中，在它到达运行时间前可以捕获所有类型的错误，并中断它们的执行。

TypeScript 主要特点总结

- TypeScript 是 JavaScript 的超集.
- TypeScript 增加了可选类型、类和模块
- TypeScript 可编译成可读的、标准的 JavaScript
- TypeScript 支持开发大规模 JavaScript 应用
- TypeScript 设计用于开发大型应用，并保证编译后的 JavaScript 代码兼容性(tsconfig配置成es3的代码)
- TypeScript 扩展了 JavaScript 的语法，因此已有的 JavaScript 代码可直接与 TypeScript一起运行无需更改
- TypeScript 文件扩展名是 ts，而 TypeScript 编译器会编译成 js 文件
- TypeScript 语法与 JScript .NET 相同

Ts怎么用

1. 搭建TypeScript开发环境

> 需要Node.js 和 Npm。
> 安装 TypeScript 最简单的方式就是通过 npm。使用以下命令行，可以全局安装 TypeScript 包，然后就可以在所有项目中使用TypeScript编译器了:

```
npm install -g typescript
```

打开终端然后运行 tsc -v 命令来查看是否正确安装了 TypeScript

```
tsc -v //Version 2.6.2
```

支持 TypeScript 的文本编辑器

> Visual Studio Code
> Sublime Text
> WebStorm
> Atom

2. 编译成JavaScript

```
TypeScript 是 写在 .ts 文件（或者 JSX的.tsx）里，不能直接在浏览器端运行，需要首先转译为xxx.js。这个编译的过程可以有多种实现方式：
```

在终端上运行前面提到的命令行工具 tsc。
1直接在 Visual Studio 或者其他 IDE 和文本编辑器上（操作）。
2使用自动化构建工具，例如 gulp。

下面的命令行把 TypeScript 文件 main.ts编译为 JavaScript 版本的 main.js。如果 main.js 已经存在的话会被覆盖。

```
tsc main.ts
```

也可以通过列出所有的文件或者使用通配符来一次编译多个文件：

```
tsc main.ts worker.ts
tsc *.ts
```

有更改的时候也可以使用 –watch 来自动编译成 TypeScript 文件：

```
tsc *.ts -watch
```

在实际项目运用中，用户也可以创建一个 **tsconfig.json** 文件，包含多种构建设置。因为配置文件在某种程度上是可以自动化进程的，所以在有许多.ts文件的大型项目中有配置文件是很方便的。可以在这里阅读到更多关于 tsconfig.json 的TypeScript 文档。
http://www.typescriptlang.org/docs/handbook/tsconfig-json.html

TS重点知识点回顾以及在项目中的实际使用

> 在线编辑器 http://www.typescriptlang.org/play/

静态类型

> TypeScript 一个很独特的特征是支持静态类型。意思就是可以声明变量的类型，（因此）编译器就可以确保赋值时不会产生类型错误。如果省略了类型声明，TypeScript 将会从代码中自动推测出正确的类型。

这有个例子。任意变量，函数自变量或者返回值在初始化时都可以定义自己的类型。

```
var burger: string = 'hamburger',      // String
    calories: number = 300,            // Numeric
    tasty: boolean = true;             // Boolean

// 也可以省略类型声明:
// var burger = 'hamburger';

// 函数参数需要string 和 integer.
// void 不return东西.

function speak(food: string, energy: number): void {
  console.log("Our " + food + " has " + energy + " calories.");
}

speak(burger, calories);
```

因为 JavaScript 是弱类型语言（即不声明变量类型），因此 TypeScript 编译为 JavaScript 时，（变量类型的声明）全部被移除：

```
// 上面TS示例中的JavaScript代码。

var burger = 'hamburger',
    calories = 300,
    tasty = true;

function speak(food, energy) {
    console.log("Our " + food + " has " + energy + " calories.");
}

speak(burger, calories);
```

然而，如果我们试着输入非法的语句，tsc 会警告代码里有错误。例如：

```
// 给boolean类型的变量赋值一个string
var tasty: boolean = "I haven't tried it yet"; //main.ts(1,5): error TS23
22: Type 'string' is not assignable to type 'boolean'.

//如果传入错误的函数自变量也会发出警告:
function speak(food: string, energy: number): void{
  console.log("Our " + food + " has " + energy + " calories.");
}

// 参数不能匹配上函数定义的类型
speak("一颗大榴莲", "超级多");// main.ts(5,30): error TS2345: Argument of ty
pe 'string' is not assignable to parameter of type 'number'.
```

以下是一些最常用的数据类型:

- Number (数值类型) – 所有数字都是数值类型的，无论是整数、浮点型或者其他数值类型都相同。
- String（字符串类型） – 文本类型，就如 vanilla JS 字符串一样可以使用单引号或者双引号。
- Boolean（布尔类型） – true 或者 false，用 0 和 1 会造成编译错误。
- Any（任意类型） – 该类型的变量可以设定为字符串类型，数值类型或者任何其他类型。
- Arrays（数组类型） – 有两种语法：my_arr: number[];或者my_arr: Array
- Void （空类型）- 用在不返回任何值的函数中。
  可以到官方的TypeScript文档查看所有变量类型列表 – 点击这里。
  http://www.typescriptlang.org/docs/handbook/basic-types.html
  中文 点击这里。
  https://www.tslang.cn/docs/handbook/basic-types.html

Interfaces 接口

> 接口通常会根据一个对象是否符合某种特定结构来进行类型检查。通过定义一个接口我们可以命名一个特殊的组合变量，确保它们会一直一起运行。当转译成 JavaScript 时，接口会消失 – 它们唯一的目的是在开发阶段里起到辅助的作用。

在下面的例子中我们定义了一个简单的接口来对一个函数自变量进行类型检查:

```
// 简单定义一下食物的接口.
interface Food {
    name: string;
    calories: number;
}
// 接口的使用确保函数正确执行
function speak(food: Food): void{
  console.log("Our " + food.name + " has " + food.calories + " calories."
);
}

// 定义一个对象包含正确的参数
// 这里会有自动的类型推断
var ice_cream = {
  name: "ice cream",
  calories: 200
}

speak(ice_cream);
```

属性的顺序并不重要。我们只需必要的属性存在并且是正确的类型。如果哪里有遗漏，类型错误，或者命名不同的话，编译器都会报警告信息。

```
interface Food {
    name: string;
    calories: number;
}

function speak(food: Food): void{
  console.log("Our " + food.name + " has " + food.calories + " grams.");
}
// 错误输入函数的参数name as nmae.
var ice_cream = {
  nmae: "ice cream",
  calories: 200
}

speak(ice_cream);//main.ts(16,7): error TS2345: Argument of type '{ nmae:
 string; calories: number; } is not assignable to parameter of type 'Foo
d'.Property 'name' is missing in type '{ nmae: string; calories: number;
 }'.
```

官方文档:
http://www.typescriptlang.org/docs/handbook/interfaces.html

类

在搭建大型规模的应用程序时，尤其是在 Java 或 C# 当中，许多开发者会优先选择面向对象编程。TypeScript 提供一个类系统，和 Java、C# 中的非常相似，包括了继承，抽象类，接口实现，setters/getters 方法等。

值得一提的是由于最新的 JavaScript 更新（ECMAScript 2015），这些类对于 vanilla JS 来说是原生的，并且在没有TypeScript 的情况下也可以使用。这两种实现方式非常相似但是也有不同的地方，TypeScript 更加严格一些(使用类似babel编译为es5甚至es3的代码)。

继续上面的 food的 例子，这里有一个简单的TypeScript类：

```
class Menu {
  // Our properties:
  // 默认是 public, 也可以是private 或者 protected.
  items: Array<string>;
  pages: number;

  // 一个简单的构造函数。
  constructor(item_list: Array<string>, total_pages: number) {
    // The this keyword is mandatory.
    this.items = item_list;
    this.pages = total_pages;
  }

  // Methods
  list(): void {
    console.log("Our menu for today:");
    for(var i=0; i<this.items.length; i++) {
      console.log(this.items[i]);
    }
  }

}

// 创建一个Menu类的新实例
var sundayMenu = new Menu(["pancakes","waffles","orange juice"], 1);

// 使用实例内的list方法.
sundayMenu.list();
```

你会发现TypeScript和Java 或者 C#它们在语法上非常相似。继承也是一样：

```
class HappyMeal extends Menu {
  // Properties are inherited属性继承

  // 必须定义一个新的构造函数。
  constructor(item_list: Array<string>, total_pages: number) {
    // 在这种情况下我们用了相同的 constructor
    // 使用super() 继承父类中的方法
    super(item_list, total_pages);
  }


  //  也可以重写方法来覆盖list()
  list(): void{
    console.log("Our special menu for children:");
    for(var i=0; i<this.items.length; i++) {
      console.log(this.items[i]);
    }

  }
}

// Create a new instance of the HappyMeal class.
var menu_for_children = new HappyMeal(["candy","drink","toy"], 1);

// This time the log message will begin with the special introduction.
menu_for_children.list();
```

```
```

另一个例子

```
//类的声明
class Person{
    constructor(public name:string){
        this.name = name;
        consloe.log("haha"):
    }// 类的构造函数，只在实例化时被调用，而且被调用一次；

    eat(){
        console.log("im eating");
    }
}
```

```
//子类拥有父类的属性和方法,还可以自定义自己属性和方法
class Emplyee extends Person{
    code: string:
    constructor(name: string, code: string){
        super(name);//子类构造函数必选调用父类构造函数
        consloe.log("xixi"):
        this.code = code;
    }

    work(){
        super.eat();//调用父类方法
        this.doWork();
    }
    private doWork(){
        console.log("im working");
    }
}
//类的实例化

var e1 = new Employee("name", "1");
e1.work();

var p1 = new Person("batman");
p1.eat();

var p2 = new Person("superman");
p2.eat();
```

类的访问控制符：

- private:只能中类内部访问
- public:类的外部和内部都可以访问
- protected:类内部和子类可以访问

更深入了解类，可以阅读 TypeScript 文档 – 点击这里。
https://www.tslang.cn/docs/handbook/classes.html

泛型

> 泛型（Generics）是允许同一个函数接受不同类型参数的一种模板。相比于使用 any 类型，使用泛型来创建可复用的组件要更好，因为泛型会保留参数类型。

一段简单的脚本例子，传入一个参数，返回一个包含了同样参数的数组。

```
//  函数名后的< T >象征着这是一个泛型函数。
// 当我们调用这个函数,每一个T的实例将被替换为实际提供的类型。
//  接收一个参数T类型,
// 返回一个数组类型的T。

function genericFunc<T>(argument: T): T[] {
  var arrayOfT: T[] = [];    // Create empty array of type T.
  arrayOfT.push(argument);   // Push, now arrayOfT = [argument].
  return arrayOfT;
}

var arrayFromString = genericFunc<string>("beep");
console.log(arrayFromString[0]);          // "beep"
console.log(typeof arrayFromString[0])    // String

var arrayFromNumber = genericFunc(42);
console.log(arrayFromNumber[0]);          // 42
console.log(typeof arrayFromNumber[0])    // number
```

第一次调用函数的时候，我们将类型手动设置成字符串。第二次及以后再次调用的时候就不必这样做了，因为编译器会判断传递过什么参数并且自动决定哪种类型最适合。虽然不是强制性的，但是由于编译器在众多复杂环境中确定正确类型的时候可能会失败，所以每次都传入类型是好的做法。

TypeScript 文档里包含了一些比较新的例子，包括泛型类，泛型类与接口绑定等等，更多请点击这里。
https://www.tslang.cn/docs/handbook/generics.html

在开发大型应用时，另一个重要的概念是模块化。与一个有 10000 行代码的文件相比，把代码分成多个可复用组件，这样可以帮助项目保持条理性和易懂性。

TypeScript 介绍了导入和导出模块的语句，但是并不能解决文件间的真正连接。TypeScript 依赖于第三方函数库来加载外部模块：用于浏览器应用程序的 require.js 和用于 Node.js 的 CommonJS。我们来看一个简单的带有 require.js 的TypeScript 模块例子：

我们会有两个文件。一个是导出函数，另一个是导入并调用函数。

```
exporter.ts
var sayHi = function(): void {
    console.log("Hello!");
}
 importer.ts

import sayHi = require('./exporter');
sayHi();
export = sayHi;
```

现在我们需要下载 require.js，包含在一个script标签里 – 如何设置请点击这里 http://requirejs.org/docs/start.html
最后一步是编译这两个 .ts 文件。需要添加一个额外的参数来告诉 TypeScript，我们是为 require.js 创建模块的（也被称为AMD），而不是 CommonJS。

```
tsc --module amd *.ts
```

第三方声明文件

在使用一个常规 JavaScript 库时，我们需要用到一个声明文件来使这个库是否和 TypeScript 兼容。一个声明文件包含 .d.ts 扩展名和关于该库的多种信息，还有API。

TypeScript的声明文件通常是手写的，但是极有可能你需要的库中已经有了一个由其他人创建的 .d.ts 文件。DefinitelyTyped 是最大的公共存储库，包括1000多个库文件。也有一个用来管理 TypeScript 定义的 Node.js 流行模块，叫 Typings。

如果仍需要亲自写声明文件，可以从这里开始 (http://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html)。

Angular中对TypeScript的使用

https://github.com/maskhb/ng5Demo.git