# TypeScript

Note Version : 1.0、

Test Version : ts2.4.2

Data : 2017年8月27日

## 1、前置知识

* + 理解ES5、ES6、javaScript、TypeScript的概念和关系
  + JavaScript开发经验
  + 如果懂java学习来会事半功倍

## 2、TypeScript优势

* + 支持ES6规范
  + 强大的IDE支持
  + Angluar2的开发语言

## 3、搭建TypeScript开发环境

### 3、1在线调到 ： <http://www.typescriptlang.org/play/index.html>

### 3、2本 地 : <http://www.cnblogs.com/Leo_wl/p/5751187.html>

***Visual Studio Code 工具开发***

#### 3、2、1、安装node.js

安装文件下载地址：[**Node.js Downloads**](https://nodejs.org/en/download/)。TypeScript源码需要进行编译以后才能运行，Node.js提供了编译环境。

#### 3、2、2、安装TypeScript编译工具

安装好node.js后，

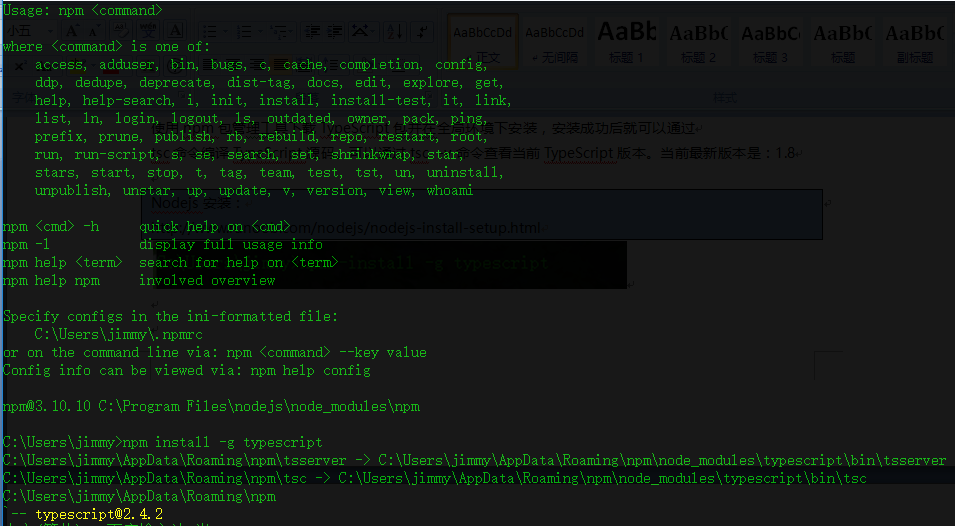
|  |
| --- |
| Nodejs安装：  http://www.runoob.com/nodejs/nodejs-install-setup.html |

cmd，输入以下命令

|  |
| --- |
| **npm install -g typescript** |

使用npm包管理工具下载TypeScript包并在全局环境下安装，安装成功后就可以通过

tsc 命令编译TypeScript源码。可以通过tsc -v 命令查看当前TypeScript版本。当前最新版本是：1.8

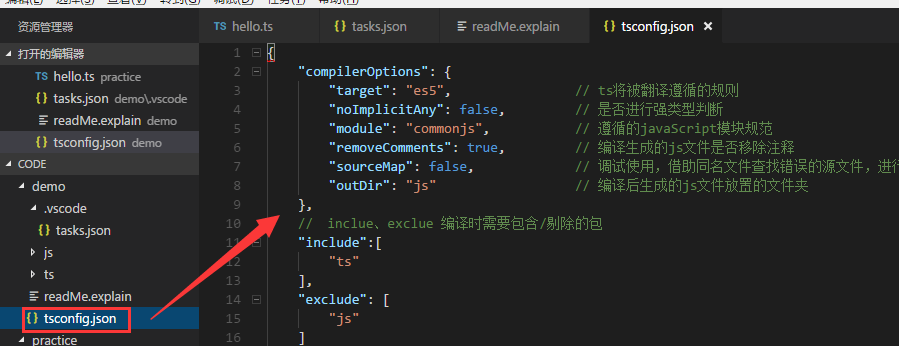


检查TypeScript版本



#### 3、3、3、使用visial studio code进行开发（待完善）

一下目录结构是一个简单的demo的结构



## 4、TS语法及特性（结合ES6标准）

### 4、1 字符串新特性（ES6模板字符串）

模板字符串由 : 反撇号（` `）

**1、支持内部换行、插值、*嵌套(ES6说可以嵌套，目前未测试出)***

|  |  |
| --- | --- |
| let html = `<div>  < p>支持换行功能</p>  </div`;  let myname = 'liangjiaming';  console.log(`${myname}插值功能${html}，  转移展示：土豪有很多\$\`\\\{转移展示结束  `);  控制台结果：   |  | | --- | | liangjiaming插值功能<div>  < p>支持换行功能</p>  </div，  转移展示：土豪有很多$`\{转移展示结束 | |

**Tip: 需要在模板字符串中引入字符$和{ 用反斜杠转义每一个字符：`\$`和`\{`**

**2、自动拆分字符串**

|  |  |
| --- | --- |
| function print(temple, name, age) {  console.log(`temple:${temple}`);  console.log(`name:${name}`);  console.log(`age:${age}`);  }  let myName = "liangjiaming";  function getAge() {  return 18;  }  print`hello my name is ${myName},i'm ${getAge()}`; // 表达式1  print `hello my name is ${myName},i'm `; // 表达式2报错 但解析后仍然能打印出信息  print`hello my name is ${myName},i'm ${getAge()},${}`; // 表达式3 报错    控制台结果：   |  | | --- | | 表达式1：  temple:hello my name is ,,i'm ,  name:liangjiaming  age:18  表达式2：  temple:hello my name is ,,i'm  name:liangjiaming  age:undefined  表达式3：  temple:hello my name is ,,i'm ,,,  name:liangjiaming  age:18 | |

**小结：字符串模板是支持模板内部换行以及插值（${}）**

**使用字符串模板调用方法的时候，会进行自动的拆分,传递参数用${},并且ts会进行参数个数的校验，若参数个数符合则正常拆分，若不符合个数则进行顺序赋值。**

### 4.2 参数

#### 4.2.1参数类型

**ES6中原始类型：**

*  Undefined 未定义
*  Null 空值
*  Boolean 布尔类型
*  Number 数字类型
*  String 字符串类型
*  Object 对象类型

**TS中的数据类型**

* + Boolean 布尔
  + Number 数字
  + String 字符串
  + Array 数组
  + Enum 枚举
  + Any 任意类型
  + Void 一般这种类型都是用在函数的返回值

|  |
| --- |
| // 声明变量类型  let myname: string = `jimmy`;  myname = 13 // 编译报错  let alias = 'jimmy';  alias = 14; // 编译报错  let eve: any = 'jimmy';  eve = 10;  let age: number = 11;  // 方法参数类型、返回值类型  function test(name: string): string {  return name;  } |

#### 4.2.2参数默认值

声明默认值、方法参数默认值

|  |  |
| --- | --- |
| var myname: string = `jimmy`;  // 默认值参数要声明在必选参数之后  function test(a: string, b: string, c: string = 'yoyo') {  console.log(`a:${a},b:${b},c:${c}`)  }  test(myname, 'Hi');  test(myname); // 报错  控制台打印结果   |  | | --- | | a:jimmy,b:Hi,c:yoyo  a:jimmy,b:undefined,c:yoyo | |

#### 4.2.3 可选参数

|  |  |
| --- | --- |
| var myname: string = `jimmy`;  // 可选参数必须声明在必选参数后面  function test(a: string, b?: string, c: string = 'yoyo') {  console.log(`a:${a},b:${b},c:${c}`)  // 需要单独处理不传的时候的报错异常ERROR  console.log(b.length);  }  test(myname, 'Hi');  test(myname);   |  | | --- | | a:jimmy,b:Hi,c:yoyo  2  a:jimmy,b:undefined,c:yoyo  Uncaught TypeError: Cannot read property 'length' of undefined  at test (<anonymous>:7:18)  at <anonymous>:10:1  at HTMLButtonElement.excuteButton.onclick (http://www.typescriptlang.org/play/playground.js:243) | |

### 4.3函数（支持重载）

#### 4.3.1 Rest and Spread[…]操作符（ES6不定参数）

用来声明任意数量的方法参数

|  |  |
| --- | --- |
| // 声明不定参数  function test(a, ...needle ) {  console.log(`a:${a}`);  console.log(`needle:${needle}`)  }  test('jimmy','love', 'mother');  let para = ['jimmy','love', 'mother'];  test(para);控制台结果：   |  | | --- | | a:jimmy  needle:love,mother  a:jimmy,love,mother  needle: | |

ES6还支持一下调用方式，TS目前不支持这种语法

|  |  |
| --- | --- |
| function test(a, b, c) {  console.log(`a:${a} b:${b} c:${c}`)  }  var para = ['jimmy', 'love', 'daddy'];  var args = ['jimmy', 'deepLove', 'daddy', 'mother', 'lover'];  test(...para);  test(...args);  控制台结果：   |  | | --- | | a:jimmy b:love c:daddy  a:jimmy b:deepLove c:daddy | |

#### 4.3.2 generator函数（ES6 生成器generators）

控制函数的执行，手工暂定和恢复代码执行。

关键字：yield

|  |  |
| --- | --- |
| function\* doSomething(){  yield console.log("start");  yield console.log("Hi! I am jimmy");  yield console.log('finish');    }  let iter = doSomething();  控制台结果：   |  | | --- | | iter.next();  start  {value: undefined, done: false}  iter.next();  Hi! I am jimmy  {value: undefined, done: false}  iter.next();  finish  {value: undefined, done: true} | |

代码片段2:

|  |  |
| --- | --- |
| function\* getStockPrice(stock) {  while (true) {  yield Math.random()\*100;  }  }  var priceGenerator = getStockPrice('IBM');  let limitPrice = 30;  var price = 100;  while (price > limitPrice) {  price = priceGenerator.next().value;  console.log(`price:${price}`);  }  console.log(`buy IBM at ${price}`);  控制台结果：   |  | | --- | | price:92.3768788310275  price:77.94496783533138  price:55.34506698096238  price:60.72895857265179  price:36.88825285966353  price:41.414616729629316  price:57.18981655074251  price:15.125107572596107  buy IBM at 15.125107572596107 | |

#### 4.3.3理解generator

首先看如下代码：

|  |  |
| --- | --- |
| function\* doSomething(){  yield Math.random()\*100;  yield Math.random()\*100;  yield Math.random()\*100;  }  let iter = doSomething();  控制台结果：   |  | | --- | | iter.next();  {value: 90.16063004650843, done: false}  done : false  value : 90.16063004650843  \_\_proto\_\_:Object | |

运行iter.next()方法的时候查看返回值

*{value: 90.16063004650843, done: false}*

*done : false*

*value : 90.16063004650843*

*\_\_proto\_\_:Object*

value 为yield后面表达是（语句块）返回值。

done 为generator的是否存在下步false表示有 true表示没有

|  |
| --- |
| **ES6：**调用generators（ES6概念生成器）的时候，它不是立即执行，而是返回一个已暂停的生成器对象，  当调用生成器对象的.next()方法时，函数调用将其自身解冻并一直运行到下一个 yield（若有java基础可以理解为java的阻塞） 表达式，再次暂停。调用最后一个 iter.next()时，我们最终抵达生成器函数的末尾，所以返回结果中 done的值为 true。抵达函数的末尾意味着没有返回值，所以返回结果中 value 的值为undefined  **Tip：生成器不是线程**  当生成器运行时，它和调用者处于同一线程中，拥有确定的连续执行顺序，永不并发  生成器就是迭代器！ |

扩展思考：ES6所提及的promise编程方式

### 4.4.3析构表达式(ES6解构 Destructuring)

例子1：

|  |  |
| --- | --- |
| function getStock() {  return {  code : 'IBM',  price: 100,  contact:{  phone: 13100000000,  tel : '0752-7895642'  }    }  }  let { code, price,contact } = getStock();  console.log(`code:${code} price:${price} contact:${contact}`)  控制台结果：   |  | | --- | | code:IBM price:100 contact:[object Object] | |

**解析：数组、对象、其他**

数组：

|  |  |
| --- | --- |
| let [a, b, c] = [1, 2, 3];  let arr1 = ['jimmy', 'see', 'Lily', 'hit', 'Tom'];  let [myName, opt, ...para] = arr1;  console.log(`${myName}${opt}${para}`)控制台结果：   |  | | --- | | jimmyseeLily,hit,Tom | |

对象：

|  |  |
| --- | --- |
| let obj = {  a: 1,  b: 2,  c: 3,  d: 4,  arr: [  'Yo.',  {  sone:'typeScript'  }  ]  }  let { a, b: B } = obj;  console.log(`a:${a} B:${B}`);  let c = 0;  ({ c, d,e=1 } = obj); // ts编译报错  console.log(`c:${c} d:${d} e:${e}`);  let { arr: [greeting, { sone }] } = obj; // ts编译报错  console.log(`${greeting} ${sone}`);  控制台结果：   |  | | --- | | a:1 B:2  c:3 d:4 e:1  Yo. typeScript | |

方法：

|  |  |
| --- | --- |
| let { floor, pow } = Math;  console.log(floor(1.9));  console.log(pow(2,3));  控制台结果：   |  | | --- | | 1  8 | |

其他：

|  |  |
| --- | --- |
| let { length } = 'Yo.';  console.log(length)  let [ a,b,c ]= 'Yo.';  console.log(`${a} ${b} ${c}`)  控制台结果：   |  | | --- | | 3  Y o . | |

### 4.5箭头表达式 =>(ES6箭头函数 Arrow Functions)

|  |  |
| --- | --- |
| let myArray = [1, 2, 3, 4, 5];  console.log(myArray.filter(value => value % 2 == 0));  控制台结果：   |  | | --- | | Array(2)0: 21: 4length: 2\_\_proto\_\_: Array(0) | |

解放匿名函数中this关键字的问题

|  |  |
| --- | --- |
| function getStock(name:string) {  this.name = name;  setInterval(function () {  console.log("ES5"+this.name);  },1000);  }  var stock = getStock('IBM');  function getStock2(name:string) {  this.name = name;  setInterval( () =>{  console.log("ES6"+this.name);  });  }  var stock2 = getStock('IBM');  控制台结果：   |  | | --- | |  | |

### 4.6.表达式和循环（for… of）

|  |  |
| --- | --- |
| var myArray = [10, 20, 30, 40];  myArray.desc='ES5写法，ts不要这么写'  // forEach 循环  myArray.forEach(value => console.log("forEach循环："+value));  for (var v in myArray) {  console.log("for in循环："+ v);  }  for (var a of myArray) {  console.log("for of循环：" + v);  }  控制台结果：   |  | | --- | |  | |

### 4.7、类（ES6 class）

Java程序员的福音

知识点：

类的定义、构造、属性、方法

访问控制符（public private protected）

继承（extends super）

#### 4.7.1定义、构造、属性、方法:

|  |  |
| --- | --- |
| class Person{    private age;  protected sex;  constructor(public myName :string) {  console.log("hi");  this.eat();  }  eat() {  console.log(`${this.myName} is eating`);  }  work() {  this.eat();  console.log("then working");  }  }  let p = new Person(`jimmy`);  控制台结果：   |  | | --- | | hi  jimmy is eating | |

#### 4.7.2访问控制符（public private protected）

Public 类内部和外部均可以访问

Protected 类内部以及其子类均可以访问

Private 只有类内部可以访问

#### 4.7.3继承

|  |  |
| --- | --- |
| class Person{    private age = 18;  protected sex = '男';  constructor(public myName: string) {  //this.sex = '男';  console.log("hi");  this.eat();  }  eat() {  console.log(`${this.myName} is eating`);  }  protected work() {  this.eat();  console.log("then working");  }  }  class Emplyee extends Person{    }  let e = new Emplyee("Arui");  console.log(e.sex); // ts报错  控制台结果：   |  | | --- | | hi  Arui is eating  男 | |

Tip: 继承不会继承private的属性和方法，子类的构造方法必须调用父类的构造方法。

Super的理解只能调用父类的方法（包括构造方法和普通方法），TS不支持多继承

|  |  |
| --- | --- |
| class Person{    private age = 18;  protected sex = '男';  constructor(public myName: string) {  //this.sex = '男';  console.log("hi");  this.eat();  }  eat() {  console.log(`${this.myName} is eating`);  }  protected work() {  this.eat();  console.log("then working");  }  }  class Emplyee extends Person{  constructor(public myName:string,public hobby:string) {  super(myName);  console.log(`${this.myName}性别${this.sex}喜欢${this.hobby}`);  }  }  let e = new Emplyee("Arui","女");  控制台结果：   |  | | --- | | hi  Arui is eating  Arui性别男喜欢女 | |

### 4.8.泛型

采用java的概念解释即为参数化类型

|  |  |
| --- | --- |
| /\*\*  \* 没有泛型，我们要么必须给身份功能的特定类型  \*/  function identity1(arg: number): number {  return arg;  }  /\*\*  \* 或者:我们可以描述使用“任意”类型的标识功能：  \*/  function identity2(arg: any): any {  return arg;  }  console.log(identity1('jimmy')); // ts报错  console.log(identity2('jimmy'));  控制台结果：   |  | | --- | | jimmy  jimmy | |

### 4.9、接口（Interface）

用法一: 作为方法参数的约束

|  |  |
| --- | --- |
| interface IPerson{  name: string;  age: number;  }  class Person{  constructor(public config: IPerson) {  console.log(this.config);  }  }  var p = new Person({  name: 'jimmy',  age:18  })  var p = new Person({  name: 50, // ts报错  age1:18 // ts报错  })  控制台结果：   |  | | --- | | Object age: 18 name: "jimmy"\_\_proto\_\_: Object  Object age1: 18 name: 50\_\_proto\_\_: Object | |

用法二：类似java的接口使用，定义一些抽象方法，实现类中必须实现。甚至原则都一样，接口中的属性和方法必须是public的。

|  |  |
| --- | --- |
| interface IPerson{  private age: number; // ts 报错  protected sex: string; // ts 报错  eat(); // 无实现  private sing(); // ts 报错  }  class Man implements IPerson{  eat() {  console.log(`吃的多`)  }  }  class Women implements IPerson{ // 未实现接口方法则ts报错    }  let m = new Man();  m.eat();控制台结果：   |  | | --- | |  | |

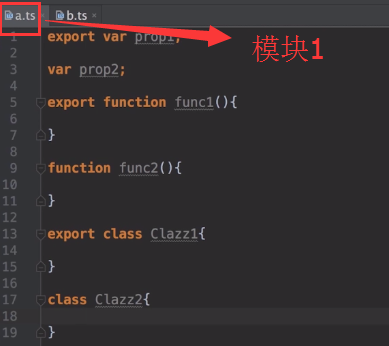
### 4.10模块（ES6 Moudles）

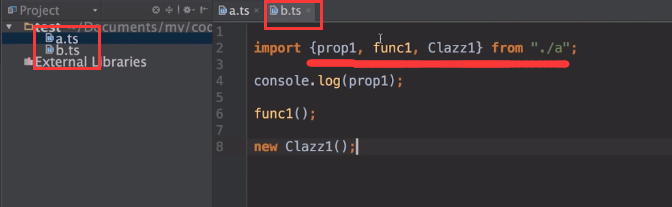
关键字：export import

类似于java的包概念，但略有不同

1、 export 可以选择对外暴漏哪些属性和方法

Import 引用其他模块的属性或者方法





**ES6补充：**

**Export列表**

不需要标记每一个被导出的特性，你只需要在花括号中按照列表的格式写下你想

导出的所有名称

export {detectCats, Kittydar};

// 此处不需要 `export`关键字

function detectCats(canvas, options) { ... }

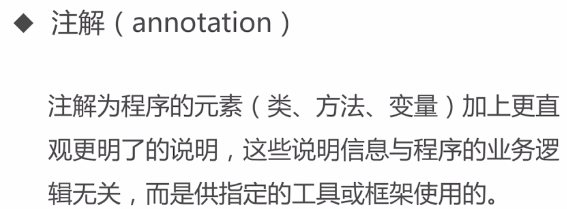
class Kittydar { ... }

**重命名 import 和 和 export**

|  |
| --- |
| // 这两个模块都会导出以`flip`命名的东西。  // 要同时导入两者，我们至少要将其中一个的名称改掉。  import {flip as flipOmelet} from "eggs.js";  import {flip as flipHouse} from "real-estate.js"; |

|  |
| --- |
| // unlicensed\_nuclear\_accelerator.js - 无 DRM（数字版权管理）的媒体流  // （这不是一个真实存在的库，但是或许它应该被做成一个库）  function v1() { ... }  function v2() { ... }  export {  v1 as streamV1,  v2 as streamV2,  v2 as streamLatestVersion  }; |

### 4.11注解（annotation）



**Angular2学习中在进行详细学习**

### 4.12类型定义文件

在TS中使用JS的第三方的框架如jqeury等。

类型定义文件 (\***.d.ts**)

## 5、总结

一、TS的环境搭建 需要安装nodeJS 使用npm命令在线安装

二、TS的特性

对ES6大部分的新特性进行了支持和扩展、借鉴了一些java的特性

1. 字符串模板（``）
2. 参数默认值、可选参数、不定参数（参数类型）
3. 生成器、解构、箭头函数/表达式
4. 类（class）
5. for..of 循环
6. 继承
7. 接口
8. 泛型
9. 模块、注解

Tip：本笔记初学者查看可以了解个大致的概念，写的不是很详细。大神若看到了。还请指正

Tip\_2:建议浏览一遍ES6的基本特性在学习TS若有java基础学习TS相对会简单。