TypeScript

一.特点

微软开发

Js的超集

ES6

Angular2框架由typeScript开发

二.内容

1.ts优势:

2.安装环境

3.ts概念语法和特性

1Ts优势:

1)支持ES6(2015年) , 客户端脚本语言的规范

2)强大的IDE支持 :类型检查\语法提示\重构(修改变量或方法的名字)

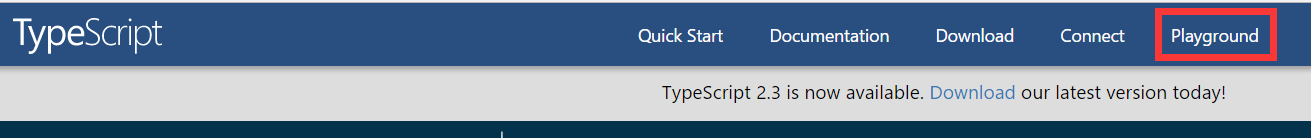
3)angular2 的开发语言

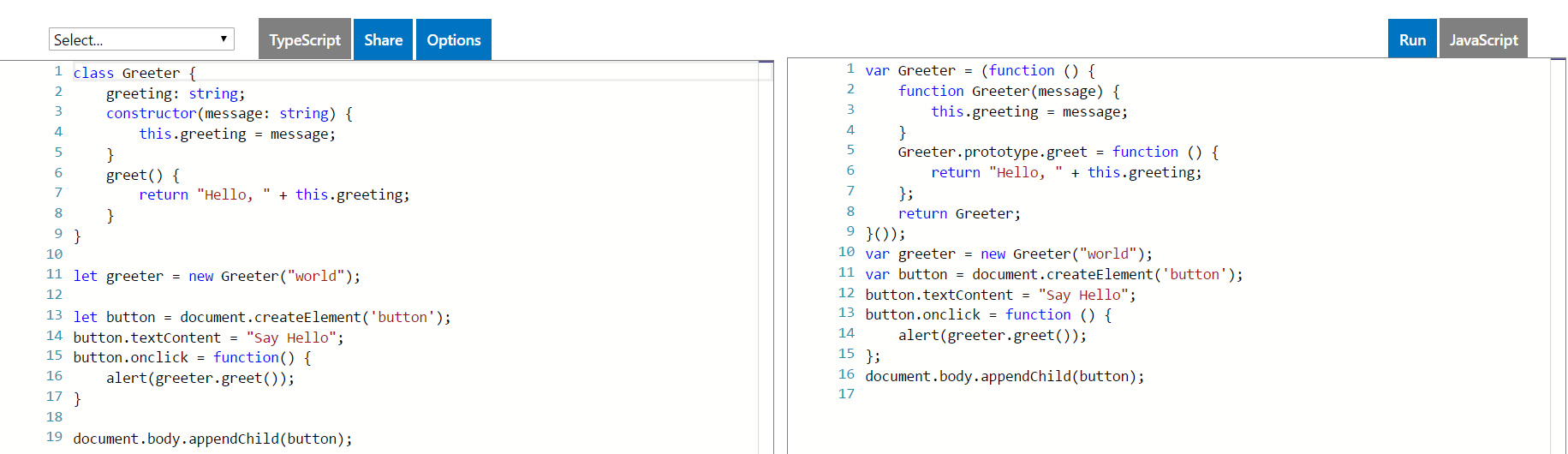
一.ts开发环境

Compiler 编译器 ts🡪js 原因 ES6 部分浏览器不支持

1.在线compiler编译

地址: <http://www.typescriptlang.org/>

选择playground



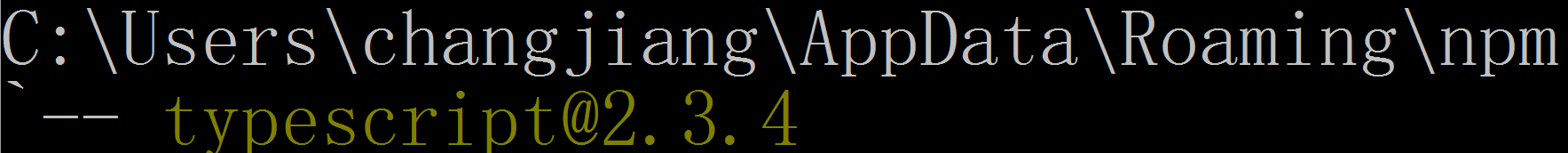
2.本地环境搭建

使用node环境下的npm安装

安装ts, 只涉及win平台 ,

安装步骤: npm install -g typescript (全局安装ts)

出现如下情况,ts安装完毕



3.简单测试

1)sublime新建文档

2)输入如下代码:

//ts创建类代码

export class hello{

}

保存在桌面,命名为hello.ts

3)通过命令行,编译该文件 tsc hello.ts

此时,会生成hello.js(编译后产生的代码,将会在浏览器执行的)

4.使用IDE工具

WebStorm

新建项目

新建ts文件, 输入文件名, 保存

开始书写ts代码

点击右上角OK编译,自动生成js文件

二.字符串新特性

1.多行字符串

//多行字符串使用连接

//js中

var content1 = "aaa" + "bbb" + "ccc";

//ts中 双撇号(数字1左边的键)

var content2 = `aaa

bbb

ccc`

编译后,js中显示

**var** content2 = "aaa\nbbb\nccc";

2.字符串模板

var myname = "changjiang";

var getname = function () {

return "changjiang";

}

console.log(`hello ${myname}`);

console.log(`hello ${getname()}`);

console.log(`<div>

<span>${myname}</span>

<span>${getname()}</span>

<div>xxx</div>

</div>`);

编译结果:

**var** myname = "changjiang";  
**var** getname = **function** () {  
 **return** "changjiang";  
}  
console.log(`hello ${myname}`);  
console.log(`hello ${getname()}`);  
console.log(`<div>  
<span>${myname}</span>  
<span>${getname()}</span>  
<div>xxx</div>  
</div>`);

3.自动拆分字符串

字符串模板作为函数参数,自动拆分变量

function test(template, name, age) {

console.log(template);

console.log(name);

console.log(age);

}

var myname = "chang jiang";

var getAge = function () {

return 18;

}

test`hello my name is ${myname},i'm ${getAge()}`

编译结果:

**function** test(template, name, age) {  
 console.log(template);  
 console.log(name);  
 console.log(age);  
}  
**var** myname = "chang jiang";  
**var** getAge = **function** () {  
 **return** 18;  
};  
(\_a = ["hello my name is ", ",i'm ", ""], \_a.raw = ["hello my name is ", ",i'm ", ""], test(\_a, myname, getAge()));  
**var** \_a;

三.参数

1.参数类型:在参数名称后面使用冒号来指定参数类型

var myname: string = "chang jiang";

myname = 15;//错误,报错,在js中不报错.

var abc = "xixi";

abc = 13;//错误,报错, 在第一次赋值时已经默认类型为字符串

var def: any = "haha";

def = 200; //此时不报错

var age: number = 20;//数值类型

var man: boolean = true;//布尔类型

//函数返回值定义类型

function test(): void{

return "abc";//因为上面制定了返回类型为void空, 此时返回字符串,报错;可以将上面改为string

}

//函数参数定义类型

function test1(name: string) {

}

test1("abc");

test1(12);//报错,与参数不匹配

//自定义类型

class Person{

name: string;

age: number;

}

var zhangsan: Person = new Person();

zhangsan.name = "张三";

zhangsan.age = 18;

2.默认参数:在参数声明后面用等号来指定参数的默认值

var myname: string = "chang jiang";

function test(a: string, b: string, c: string) {

console.log(a);

console.log(b);

console.log(c);

}

test("xxx", "yyy", "zzz");

//指定参数默认值

//带默认值的参数一定放在最后面

function test1(a: string, b: string, c: string ="abc" ) {

console.log(a);

console.log(b);

console.log(c);

}

test1("xxx", "yyy");

3.可选参数:在方法的参数声明后面用问号来表名此参数为可选参数

//可选参数

//b参数为可选状态

//必填写参数在可选参数前面

function test1(a: string, b?: string, c: string ="abc" ) {

console.log(a);

console.log(b);

console.log(b.length)//报错,可选参数没传时时,使用报错

console.log(c);

}

test1("xxx");

四.函数新特性

1.Rest and Spread操作符

用来声明任意数量的方法参数

//传递任意数量的参数

function test(...args) {

args.forEach(function (arg) {

console.log(arg);

});

}

test(1, 2, 3);

test(101, 102, 103, 104);

//传递固定数量参数

function test(a,b,c) {

console.log(a);

console.log(b);

console.log(c);

}

var args1 = [1, 2];

test(...args1);//此调用方法在ts中报错, 在ES6环境下没问题,编译出的js也没问题,缺少的参数显示undefined

var args2 = [7, 8, 9, 10, 11];

test(...args2);//只接受前面三个参数

2.generator函数

控制函数的执行过程, 手工暂停和恢复代码执行

function\*doSomething(){

console.log("start");

yield;

console.log("finish");

}

var func1 = doSomething();

func1.next();

func1.next();

function\* getStockPrice(stock){

while(true){

yield Math.random()\*100;

}

}

var priceGenerator = getStockPrice("IBM");

var limitPrice = 15;

var price = 100;

while(price>limitPrice){

price = priceGenerator.next().value;

console.log(`the generator return ${price}`);

}

3.destructuring析构表达式

通过表达式将对象或数组拆解成任意数量的变量

function getStock(){

return {

code: "IBM",

price: 100

}

}

//ES5里面

var stock = getStock();

var code = stock.code;

var price = stock.price;

//ES6

//此时的变量必须与方法中的一样

var { code, price } = getStock();

console.log(code);

console.log(price);

var { codex, price } = getStock();//报错

var { code: codex, price } = getStock();

多层对象中

function getStock(){

return {

code: "IBM",

price: {

price1: 200,

price2:300

},

aaa: "xixi",

bbb: "haha"

}

}

//ES6

//此时打印是对象

var { code, price } = getStock();//直接打印是对象

var { code, price:{price2} } = getStock();//打印出该值

console.log(code);

console.log(price2);

//及时内容多了, 也不影响析构表达式

数组中

var array = [1, 2, 3, 4];

var [number1, number2] = array;

console.log(number1);

console.log(number2);

var [, , number3, number4] = array;

console.log(number3);

console.log(number4);

var [number1, , ...others] = array;

console.log(others);

var array1 = [5, 6, 7, 8];

function doSomething([number5, , ...others1]) {

console.log(number5);

console.log(others1);

}

doSomething(array1);

五.表达式

1.箭头表达式

用来声明匿名函数,消除传统匿名函数的this指针问题

//匿名函数单行

var sum = (arg1, arg2) => arg1 + arg2;

console.log(sum(1, 2));

//匿名函数多行

var sum1 = (arg1, arg2) => {

return arg1 + arg2;

}

//无参数的箭头表达式

var sum2 = () => {

}

var myArray = [1, 2, 3, 4, 5];

console.log(myArray.filter(value =>value % 2 == 0))

2. 消除this关键字问题

function getStock(name: string) {

this.name = name;

setInterval(function () {

console.log("name is:" + this.name);

}, 1000);

}

var stock = new getStock("IBM");//setTimeout执行的上下文环境为window,使得add方法中的this脱离了原上下文而指向了window.

function getStock2(name: string) {

this.name = name;

setInterval(()=> {

console.log("name is:" + this.name);

}, 1000);

}

var stock = new getStock2("LAMCO");

六.for of循环

var myArray = [1, 2, 3, 4];

myArray.desc = "four number";//给数组指定属性,ts中报错,js中无问题

//使用foreach循环,循环出数值,忽略数组属性

//不能跳出, break不行

myArray.forEach(value => console.log(value));

//数组下标与数组属性

for (var n in myArray) {

console.log(n);//打印键

console.log(myArray[n]);//打印值

}

//for of 类似于for each

//可以break掉

for (var m of myArray) {

if (m > 2) break;

console.log(m);

}

//用于任何对象, 集合,数组,map, set,字符串上

for (var h of "four number") {

console.log(h);

}

七.面向对象特性

1.类

类是typeScript的核心, 使用ts开发时, 大部分代码都是写在类里面的.

类的定义,构造函数,类的继承

1. 类的声明

//类的声明

class Person {

name;

eat() {

console.log("eating");

}

}

//实例化

var p1 = new Person();

p1.name = "batman";

p1.eat();

var p2 = new Person();

p2.name = "supername";

p2.eat();

访问变量修饰符 类名首字母同样大写，但是方法不需要表明类型，直接写方法名加（）即可；属性也不要声明类型；直接写属性名即可；

但是可以加上访问控制符public（类外类内都可访问），protected（类内和子类里可访问），private（类内访问）；

用protected和private修饰的构造函数，不能被实例化；但是用protected修饰的可以被继承；

1. 构造函数

//类的声明

class Person {

//构造函数,外部访问不到

//实例化时调用

// constructor() {

// console.log("haha");

// }

//实例化人的时候必须指定名字

constructor(public name: string) {

this.name = name;

}

name;

eat() {

console.log(this.name );

}

}

//实例化

var p1 = new Person();

p1.name = "batman";

p1.eat();

var p2 = new Person("superwoman");

p2.eat();

1. 类的继承

//类的声明

class Person {

//实例化人的时候必须指定名字

constructor(name: string) {

this.name = name;

}

eat() {

console.log(this.name);

}

}

//雇员类

class Employee extends Person{

code: string;

work() {

}

}

//实例化

var e1 = new Employee("name");

e1.eat();

e1.work();

e1.code = "101";

super关键字

//类的声明

class Person {

//实例化人的时候必须指定名字

constructor(public name: string) {

console.log("haha");

}

eat() {

console.log("eating");

}

}

//雇员类

class Employee extends Person{

//子类的构造函数,调用父类的构造函数

constructor(name: string, code: string) {

super(name);

console.log("xixi");

this.code = code;

}

code: string;

work() {

super.eat();

this.dowork();

}

dowork() {

console.log("working");

}

}

//实例化

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

e1.work();

e1.dowork();//老板希望的

//在dowork前加上private

2.泛型

参数化的类型,一般用来限制集合的内容

//类的声明

class Person {

//实例化人的时候必须指定名字

constructor(public name: string) {

console.log("haha");

}

eat() {

console.log("eating");

}

}

//雇员类

class Employee extends Person{

//子类的构造函数,调用父类的构造函数

constructor(name: string, code: string) {

super(name);

console.log("xixi");

this.code = code;

}

code: string;

work() {

super.eat();

this.dowork();

}

dowork() {

console.log("working");

}

}

//实例化

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

e1.work();

e1.dowork();//老板希望的

//在dowork前加上private

//person就是这个数组的泛型

var workers: Array<Person> = [];

workers[0] = new Person("zhangsan");

workers[1] = new Employee("lisi", "2");

workers[2] = 10;//报错

3.接口Interface

用来建立某种代码约定,使得其他开发者在调用某个方法或创建新的类时必须遵循接口所定义的代码约定,js中没有接口概念

1. 可以声明方法（可以是构造方法也可是非构造方法）的参数类型：检查传入方法的参数是否满足接口所声明的所有属性；

//声明接口

interface IPerson{

name: string;

age: number;

}

class Person{

//接收IPerson类型的参数

//作为方法的参数的类型声明

constructor(public config: IPerson) {

}

}

var p1 = new Person({

name: "zhangsan",

age: 18

//多写少写属性都会报错

});

2) 用接口来声明方法,并且实现这个接口的类必须实现接口中的方法；

interface Animal{

eat();

}

//类实现接口

class Sheep implements Animal{

//必须实现接口方法

eat() {

console.log("i eat grass");

}

}

class Tiger implements Animal{

eat() {

console.log("i eat meat");

}

}

4.模块

模块：模块可以帮助开发者将代码分割为可重用的单元。开发者可以自己决定将哪些资源（类，方法，变量）暴露出去供外部使用，哪些只在模块内部使用；

所谓模块就是文件，一个模块就是一个文件；并没有特殊的标志来标记什么是模块；

WebStorm中两个文件

a.ts文件中

//暴露出

export var prop1;

var prop2;

export function func1(){

}

function func2(){

}

export class Clazz1{

}

class Clazz2{

}

b.ts文件中

//使用

console.log(prop1);//只暴露prop1, prop2没有, 上面自动生成

func1();

new Clazz1();

5.注解

注解annotation是为程序的元素（类，方法，变量）加上更直观明了的说明；这些说明信息与程序的业务逻辑无关，而是供指定的工具或框架使用的；

以@开头的是注解的名称：@注解名称（{}）；

括号里面有的是一些属性，例如：某个类的上面有个注解，则该注解中的属性就是该类加载时需要做的处理（如加载注解注解中的文件等）；

例子:angular2的代码

![C:\Users\changjiang\AppData\Roaming\Tencent\Users\2064534286\QQ\WinTemp\RichOle\Z6~Z9C9EZ6L6C7G~G{](T)A.png](data:image/png;base64,)

6.类型定义文件

类型定义文件（\*.d.ts）来源是Definitely Typed:

有时需要在typescript中使用一些框架及其他工具什么的，如jQuery，此时就要用到类型定义文件；

类型定义文件用来帮助开发者在typescript中使用已有的javascript工具包，如jQuery等；

每个类型定义文件就是一个typescript模块，

在刚才a.ts文件中使用jquery方法

function func2(){

//报错

$('#xxx').hide();

}

引入jq , 引入的是.d.ts文件,

Github中搜索 DefinitelyTyped