**TypeScript笔记**

1. **介绍**
   1. **Typescript 是 javascript 的一个超集遵循ES5 ES6 的规范 扩展了Javascript的语法。**
   2. **是由微软 Microsoft 开发的一款开源的编程语言。**
   3. **Typescript 更像java c# 这样的后端语言 能够使javascript开发大型的项目**
   4. **Angular2.x 就是基于Typescript 语法**
   5. **最新的Vue React也可以集成Typescript 语法**
2. **安装 编译**
   1. **安装命令 : cnpm install --global typescript**
   2. **编译 tsc helloworld.ts**
   3. **浏览器不能够编译ts文件 所以需要编译**
3. **在 vscode 中配置自动编译ts文件**
   1. **执行命令 tsc --init 生成 tsconfig.json 配置文件**
   2. **修改文件里面的的outDir为不注释 值为你要编译到的路径 默认为当前路径**
   3. **点击菜单 任务(Terminal) - 运行任务(run Task) 点击监视 tsc:监视-tsconfig.json 然后就可以自动生成代码。**
4. **TS中的数据类型**
   1. **Typescript为了增加代码的规范度 特地增加了类型校验**
   2. **写ts代码必须指定类型**
   3. **数据类型：**
      1. **Number 数字**
      2. **Boolean 布尔**
      3. **String 字符串**
      4. **Array 数组**
      5. **Tuple 元组**
      6. **Enum 枚举**
      7. **Any 任意类型**
      8. **Null 和 undefined**
      9. **Never类型**
   4. **例如： let person:string=”人” 、let flag:boolean=false;**
   5. **不能够随意变化类型值**
5. **创建数组 【有两种形式】**
   1. **Let arr = [1,2,3,4,5];**
   2. **Let arr:Array<number> = [1,2,3,4,5]; 指明了类型是什么 只能保存这一种类型的数据**
   3. **小例子**

**let** arr:Array<number> = [1,2,3,4,5,6];

**let** itemDocument = document.createElement("ul");

for(**let** item in arr){

**let** itemDocmentLi = document.createElement("li");

itemDocmentLi.innerText = String(arr[item]);

itemDocument.appendChild(itemDocmentLi);}

**let** body = document.body || document.documentElement;

body.appendChild(itemDocument)

1. **元组类型**
   1. **可以给数组的每一个位置指定类型**
   2. **例子**

**let** tuple:[number,string] = [123,'大家好'];

1. **枚举类型**
   1. **使用 enum 声明**
   2. **例子**
      1. **Enum Color {**
      2. **Success = 1,**
      3. **Error = 2,**
      4. **Paeding = 3**
      5. **}**
      6. **Let color:Color = Color.Success; 结果是1 color是 Color枚举类型**
2. **任意类型**
   1. **抵制警告**
   2. **当我们获取一个DOM节点的时候 调用里面的方法在里面很可能报错**
   3. **那么这时候我们在哪个DOM节点的变量后面加上:any就不会警告报错了**
   4. **在浏览器上运行是可以的**
   5. **例如**
      1. **Let oBox:any = documet.getElementById(‘box’);**
      2. **oBox.style.color = “red”;**
3. **Void 无返回类型常用于修饰函数返回值**
   1. **例子**
      1. **Function Person():void{ }**
4. **函数**
   1. **可以限制传入的类型**
   2. **可以设置返回的类型**
   3. **可以设置可选参数**
   4. **可以设置默认参数**
   5. **可以重载**
   6. **例子**

**function** Person(name:string,age?:number):string{

return "我的名字是 ："+name+"，我的年龄是"+age

}

console.log(Person("李文祥",18));

**function** Animal(...result:number[]):any{

return result;;

}

console.log(Animal(1,2,3,4,5,6,9,8,7))

**function** Cat(name:string="波斯猫"){

return name;

}

1. **类**

**class** Persons{

**public** name:string = "lienxaing";

**protected** nics = 123;

**private** love = "爱好";

**constructor**(name:string){

this.name = name;

}

run(){

alert(this.name);

}

}

**class** Web **extends** Persons{

**constructor**(name:string){

super(name);

}

run(){

super.run();

}

}

**let** web = new Web("lee");

web.run();

1. **抽象类**

**abstract** **class** queryAnimal{

**constructor**(){

}

**abstract** eat():void;

}

**class** smallAnimal **extends** queryAnimal{

**constructor**(){

super();

}

eat(){

console.log(123);

}

}

1. **接口**
   1. **是一种规范**
   2. **例子**
      1. **函数类型接口 传递的参数属性必须是接口内指定的属性**

**interface** inter{

name:string;

age:number

}

**function** fn(params:inter){

alert(params.name + params.age);

}

fn({name:"liwenxaing",age:18});

类接口

**interface** classed {

name:string;

eat():void;

}

**class** ex **implements** classed{

name:string;

**constructor**(name:string){

this.name = name;

};

eat():void{

alert(this.name);

}

}

**let** e = new ex("大家伙");

e.eat();

1. **泛型**
   1. **可以更方便的转换数据类型**
   2. **函数泛型**
      1. **例子**

**function** fx<T>(value:T):T{

return value;

}

alert(fx<string>("大妹子~"));

* 1. **类泛型**
     1. **例子**

**class** Ani<T>{

**constructor**(){

}

eat(value:T){

alert(value);

}

}

**Let** ani = new Ani<string>();

ani.eat("大家伙阿达大蒋大为爱打架");