TypeScript：强类型的JS，ES6超集

变量　　let a

常量　　const b=1;

类型注解 const isBoolean:boolean;

元组

```
let x: [string, number];
x = ['hello', 10]; // ok
```

任意类型any

```
let notSure: any;
notSure = 4;
notSure = 'aaaa';
```

any也能用于数组

```
const list: any[] = [1, true, 'aaa'];
list[1] = 100;
```

枚举enum

```
enum Color {Red = 1, Green = 2, Blue = 3}
const c: Color = Color.Blue;
```

函数中使用类型约束带返回值为string类型的函数

```
function greeting(person: string): string {
  return 'Hello, ' + person;
```

```
}
```

void类型 没有返回值函数的约定

```
function warnUser(): void {
  alert('aaaaaaaa');
}
```

接口 interface：约束类型的结构

```
interface Person {
  firstName: string;
  lastName: string;
}
function greeting2(person: Person) {
  return 'hello, ' + person.firstName + ' ' + person.lastName;
}
const myname = greeting2({firstName: 'tom', lastName: 'cruise'});
console.log(myname);
```

类 class: 类中三种成员：属性、构造函数、方法

修饰符：public 公共、private 私有、protected受保护

```
 class Greeter {     声明一个类
      greeting: string; // 属性
      constructor(msg: string) { // 构造函数：通常用于属性初始化
            this.greeting = msg;
       }
```

```
    greet() { // 方法
        return 'Hello, ' + this.greeting;
    }
}
```

使用类方法

```
const greeter = new Greeter('world');
console.log(greeter.greet());
```

类的继承 extends：

```
class Dog extends Animal   {
    constructor(theName: string, readonly age: number) {
        super(theName); // 使用super()调用父类构造函数
    }
}
```

静态成员 static：

```
class Grid {
    static origin = {x: 0, y: 0}    静态成员
}
```

存储器get / set方法：

```
get fullName(): string {
    return this._fullName;
}
set fullName(value: string) {
```

```
        console.log('管理员修改了雇员名称');
        this._fullName = value;
    }
```

函数 声明：

ts中函数的参数是必须的

```
    function buildName(first: string,next:string) {
        return first + last;
    }
    buildName('tom', 'jerry');
```

函数的可选参数next? (变量参数名加一个问号，那么该参数传值可选不传)

```
    function buildName(first: string,next?:string) {
        return first + last;
    }
    buildName('tom');
```

函数参数的默认值

```
    function buildName(first: string = 'James', last: string = 'Harden') {
        return first + last;
    }
    buildName(); // 默认值
```

泛型：可以使用泛型Generic来创建可重用组件，一个组件可以支持多种类型的数据

泛型类 class Result<T> { }

泛型接口 interface Result<T, U> { }