类型

```
function addOne(n: number) {
  return n + 1;
}
//表明n只能用number数值，传入其他类型 的值会报错
addOne("hello"); //error
```

JavaScript 语言就没有这个功能，不会检查类型对不对。开发阶段很可能发现不了这个问题，代码也许就会原样发布，导致用户在使用时遇到错误。

作为比较，TypeScript 是在开发阶段报错，这样有利于提早发现错误，避免使用时报错。另一方面，函数定义里面加入类型，具有提示作用，可以告诉开发者这个函数怎么用。

静态类型

```
let x = 1;
x = "fpp"; //error，typesciript已经推断了类型，后面不允许修改

let x = { foo: 1 };
delete x.foo; //error
x.bar = 2; //error。对象的属性是静态的，不允许随意增删
```

希望一旦报错就停止编译，不生成编译产物 `--noEmitOnError`

```
tsc --noEmitOnError app.ts
```

类型声明

在 javascript 变量上添加了类型声明：

```
let foo: string;

//1.函数参数和返回值，也是这样来声明类型
function toString(num: number): string {
  return String(num);
}

//2.变量只有赋值后才能使用
let x: number;
console.log(x); //error
```

编译

typescript 编译成 javascript - tsc 编译器

```
//安装(全局)，也可以在项目中 将tsc安装为一个依赖模块
npm install -g typescript

//检查版本
tsc -v

//编译脚本
tsc app.ts

//一次编译多个,在当前目录生成3个脚本文件
tsc app.ts bpp.ts cpp.ts

//将多个typescript脚本编译成javascript文件
tsc file1.ts file2.ts --outFile

//将结果保存到其他目录
tsc app.ts --outDir dist

//指定编译后的javascript版本
tsc --target es2015 app.ts
```

tsconfig.json 存储配置文件

TypeScript 允许将 `tsc` 的编译参数，写在配置文件 `tsconfig.json` 。只要当前目录有这个文件，`tsc` 就会自动读取，所以运行时可以不写参数。

```bash
$ tsc file1.ts file2.ts --outFile dist/app.js
```

上面这个命令写成 `tsconfig.json` ，就是下面这样。

```json
{
  "files": ["file1.ts", "file2.ts"],
  "compilerOptions": {
    "outFile": "dist/app.js"
  }
}
```

有了这个配置文件，编译时直接调用 `tsc` 命令就可以了。

```bash
$ tsc
```

ts-node 模块直接运行 typescript 代码

```
//全局安装
npm install -g ts-node

//安装完可以直接运行ts脚本
ts-node script.ts

//
```

any 类型 / Unknown 类型 / never 类型 / 联合类型

关闭类型检查，尽量不用

类型推断

没有指定类型，则认为该变量类型为 any

```
function add(x, y) {
  return x + y;
}

add(1, [2, 3]);
//terrible! 尽量不用，一定要显式声明
```

unknown与 any 类似，区别：

- 不能直接调用 unknown 类型的变量的方法和属性

```
let v: unknown = 11;

let v1: boolean = v; //error
```

- 不能直接调用 unknown 类型变量的方法和属性

```
let v1: unknown = { foo: 123 };
v1.foo; // 报错

let v2: unknown = "hello";
v2.trim(); // 报错

let v3: unknown = (n = 0) => n + 1;
v3(); // 报错
```

never 类型 空类型

联合类型 如果一个变量有多种类型，需要使用分支处理每种类型 此时，处理所有可能的类型后，剩余的情况就属于 never 类型。

```
function fn(x:string | number){
    if(type x === "string"){

    }
    else if(type x === "number"){

    }
    else {
```

```
        x
    }
}
```

为什么 `never` 类型可以赋值给任意其他类型呢？这也跟集合论有关，空集是任何集合的子集。TypeScript 就相应规定，任何类型都包含了 `never` 类型。因此，`never` 类型是任何其他类型所共有的，TypeScript 把这种情况称为“底层类型”（bottom type）。

总之，TypeScript 有两个“顶层类型”（`any` 和 `unknown`），但是“底层类型”只有 `never` 唯一一个。

类型系统

js 值 8 种类型

```
boolean
string
number
bigint
symbol
object
undefined
null
```