

Simi Documentation / TypeScript

Source: <https://github.com/saidake/simi-docs/tree/master/pdf>

Author: Craig Brown

Version: 1.0.0

Date: Feb 2, 2025

Typescript / Concept

File Check

Typescript: TypeScript 执行前会通过编译生成 JavaScript, 之后才能被解释执行 (都是 ES 标准)

```
// @ts-ignore    单行忽略
// @ts-nocheck   忽略全文整个文件
// @ts-check     Cancel ignore Full Text.
```

Rules

TypeScript: TS Playground: Test TypeScript rules

Function

An expression of type 'void' cannot be tested for truthiness.

```
const test:(a:number)=>void = (a:number)=>{
  console.log(a)
}
let result=test&&test(99) || false    Return type void is not allowed here.
let result=test&&test(99)              That's right
```

Typescript / Core

变量常量

let a :any=123 **let a=false** **let a:number= '-fsfs1'** as any **任意类型** (as 类型断言, 定义前方值的类型)

let a: number = 0b1010 **let a: number = 0o744** **let a: number = 6** **let a: number = 0xf00d** **数字**

let a: string = "xxx" **let a: string = `b\${ name }c`** **字符串**

let a: boolean = true **let a: boolean = -1** as any **布尔**

let a: null =null **null**

let a: undefined = undefined **undefined**

let x: never = ()=>{ throw new Error('exception')})() **never** (never 类型只能被 never 类型赋值, 在函数中它通常表现为抛出异常 或无法执行到终止点 或无限循环)

let a: string | number = 233 **混合类型**

let func: (a:number, b: boolean) => void = (a:number, b: boolean)=>{ a=a+1; return undefined } **函数类型**

readonly a: boolean=false **常量** (只能被初始
化一次)

let a: number[] = [1, 2] **let a: Array<number> = [1, 2]** **let a: { xxx: string }[] = [{xxx:"ddd"}]** **数组**

let a: [string, number, boolean] = ['Runoob', 1, false] **元组** (已知元素数量和类型的数组)

let a: { [key: string]: number } = {a: 233} **对象**, 针对未知 key 定义

let a: Person = {age: 233} **let b: Son = a** **对象**, 针对接口继承定义

类型转换

同 JS

语句运算符

同 JS

Typescript / Functionality

函数

```
function( a :string, b :string = 'xx' , c ?:string ) :string {    可重载函数, 限制参数类型和 返回值类型 (为可选参数 必须在最后, d 为默认值)
    return "fafa"
}
form?.submit()  存在调用
```

类

```
export class Person extends PureComponent<PageProps, PageState> {    限制 this.props 和 this.state
    const state :Person = {  pageNum: 1, },                        state 限制为 Person
}
```

命名空间

```
namespace Animal{    防止重名, 不同命名空间名字可以相同, 同一命名空间可以在多个文件内重复定义
    export interface DogInter { draw() }
    export class Person implements People { public draw(){} }
}
function func( aaa : Animal.DogInter ) {
    aaa.draw();
}
func(new Animal.Person() )
```

自定义类型

```
export declare type PersonType = ( ...args: any[] ) => string
```

接口

```
export interface People<T> {
    funca ? : (props: T, fields: any, allFields: any) => void;    限制函数类型
    validateMessages ? : PersonType;                            限制为自定义类型
    [messageld: string] : string | PersonType;                  不限制键的类型
}
```

枚举

```
enum color {
    red ,                默认值 0, 1, 2
    blue = 22,
    purple,
    A = getValue(),
    B = 2,               动态获取值后, 下一个值必须要初始化值, 不然编译不通过
}
let a : color = color.red    ---> 0            读取 枚举值
let a : string = color [ 22 ] ---> 'blue'      读取 枚举名称
```