Enums 枚举

枚举在程序语言及 mysql 等数据库中广泛使用

- 不设置值时，值以 0 开始递增

下面是使用枚举设置性别

```
enum SexType {
    BOY, GIRL
}

const hd = {
    name: '后盾人',
    sex: SexType.GIRL
}
console.log(hd); //{ name: '后盾人', sex: 1 }

```

当某个字段设置数值类型的值后，后面的在它基础上递增

```
enum SexType {
    BOY = 1, GIRL
}
...
console.log(hd); //{ name: '后盾人', sex: 2 }

```

可以将值设置为其他类型

```
enum SexType {
    BOY = '男', GIRL = '女'
}
...
console.log(hd); //{ name: '后盾人', sex: '女' }

```

#as 断言

as 断言的意思就是用户断定这是什么类型，不使用系统推断的类型，说白了就是『我说是什么，就是什么』
下例中 TS 会根据函数推断变量 f 的类型是 string | number

```
function hd(arg: number) {
  return arg ? 'houdunren' : 2030
}

let f = hd(1) //let f: string | number

```

我们可以由开发者来断定（断言）这就是字符串，这就是断言

```
function hd(arg: number) {
  return arg ? 'houdunren' : 2030
}

let f = hd(1) as string //let f: string

```

也可以使用下面的断言语法

```
function hd(arg: number) {
  return arg ? 'houdunren' : 2030
}

let f = <string>hd(1) //let f: stri

```

#const 断言

#let & const

- const 保证该字面量的严格类型
- let 为通用类型比如字符串类型

```
const hd = 'houdunren' //const hd: "houdunren"
let xj = 'houdunren' //let xj: string
```

```
3
```

#const

`const`断言告诉编译器为表达式推断出它能推断出的最窄或最特定的类型，而不是宽泛的类型

- 字符串、布尔类型转换为具体值
- 对象转换为只读属性
- 数组转换成为只读元组

下面限定 user 类型为最窄类型`houdunren.com`

```
let user = '后盾人' as const
user = 'houdunren.com'

//与以下很相似
let user: 'houdunren' = 'houdunren'
user = 'houdunren'

```

对象转换为只读属性

```
let user = { name: '后盾人' } as const
user.name = 'houdunren' //因为是只读属性，所以不允许设置值

```

当为变量时转换为变量类型，具体值是转为值类型

```
let a = 'houdunren.com'
let b = 2030

let f = [a, b, 'houdurnen.com', true, 100] as const //readonly [string, number, "sina.com", true, 100]
let hd = f[0]
hd = '向军'

```

#数组赋值

变量 f 得到的类型是数组的类型 string|number，所以只要值是这两个类型都可以

```

```

```
let a = 'houdunren.com'
let b = 2039

let hd = [a, b] //let hd: (string | number)[]
let f = hd[1] //let f: string | number
f = '后盾人' //不报错，因为类型是 string | number

```

使用 const 后会得到值的具体类型，面不是数组的类型

```
let a = 'houdunren.com'
let b = 2039

let hd = [a, b] as const //let hd: readonly [string, number]
let f = hd[1] //let f: number
f = '后盾人' //报错，只能是最窄类型即变量 b 的类型 number

```

也可以使用以下语法

```
let a = 'houdunren.com'
let b = 2039

let hd = <const>[a, b] //let hd: readonly [string, number]
let f = hd[1] //let f: number
f = 199

```

#解构

下面解构得到的变量类型不是具体类型，面是数组类型，比如变量 y 的类型是 **string | (() => void)**
这在写项目时是不安全的，因为可以将 y 随时修改为字符串，同时也不会有友好的代码提示

```
function hd() {
  let a = 'houdunren.com'
  let b = (x: number, y: number): number => x + y
  return [a, b]
}
```

```
let [n, m] = hd() //变量 m 的类型为 string | (() => void)

m(1, 6) //报错：因为类型可能是字符串，所以不允许调用

```

可以断言 m 为函数然后调用

```
function hd() {
  let a = 'houdunren.com'
  let b = (x: number, y: number): number => x + y
  return [a, b]
}
let [n, m] = hd()
console.log((m as Function)(1, 2))
//使用以下类型声明都是可以的
console.log((m as (x: number, y: number) => number)(1, 5)

```

可以在调用时对返回值断言类型

```
function hd() {
  let a = 'houdunren.com'
  let b = (x: number, y: number): number => x + y
  return [a, b]
}

let [n, m] = hd() as [string, (x: number, y: number) => number]
console.log(m(9, 19))

```

也可以在函数体内声明返回类型

```
function hd() {
  let a = 'houdunren.com'
  let b = (x: number, y: number): number => x + y
  return [a, b] as [typeof a, typeof b]
}

let [n, m] = hd()
```

```
console.log(m(9, 19))

```

使用 as const 就可以很高效的解决上面的问题，可以得到具体的类型，来得到更安全的代码，同时会有更好的代码提示

```
function hd() {
  let a = '后盾人'
  let b = (): void => {}
  return [a, b] as const
}

const [x, y] = hd() //变量 y 的类型为 () => void

```

因为 const 是取值的类型，下面代码虽然不报错，但此时 b 的类型已经是 字符串或函数，所以像下面一样在函数调用时 as const 没有意义

```
function hd() {
  let a = 'houdunren.com'
  let b = (x: number, y: number): number => x + y
  return [a, b]
}

const [n, m] = [...hd()] as const

```

> 也可以使用泛型来处理，我们会在泛型章节介绍

#null/undefined

默认情况下 null 与 undefined 可以赋值给其他类型

```
let hd: string = 'houdunren.com'
hd = null
hd = undefined

```

当我们修改 tsconfig.json 配置文件的 strictNullChecks 字段为 true（默认即为 true） 时，则不能将 null、undefined 赋值给其他类型

```
"strictNullChecks": true
```

除非向下面一样明确指定类型

```
let hd: string |undefiend|null = 'houdunren.com'
hd = null
hd = undefined
```

#非空断言

下面的示例获取的值可能为 HTMLDivElement 或 null，所以直接分配类型“HTMLDivElement”将报错误

> 下例操作需要开启 tsconfig.json 的配置项 strictNullChecks 字段为 true

```
const el: HTMLDivElement = document.querySelector('.hd')
console.log(el.id);
```

可以使用 as 断言类型

```
const el: HTMLDivElement = document.querySelector('.hd') as HTMLDivElement
console.log(el.id);
```

在值后面使用 `!` 来声明值非 null

```
const el: HTMLDivElement = document.querySelector('.hd')!
console.log(el.id);
```

#DOM

为了演示示例我们创建 html 文件如下

- 下面的操作需要开启 tsconfig.json 的配置项 strictNullChecks 字段为 true
- 有关 DOM 的原型知识请在后盾人网站 (opens new window)学习 DOM 章节

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>后盾人</title>
        <script src="1.js" defer></script>
    </head>
    <body>
        <div class="hd">houdunren.com</div>
        <button id="bt">插入元素</button>
    </body>
</html>

```

#类型推断

对于获取的标签对象可能是为 null 也可能是该标签类型

- body 等具体标签可以准确标签类型或 null
- 根据 class 等获取不能准确获取标签类型，推断的类型为 Element|null

```
const body = document.querySelector('body') //const body: HTMLBodyElement|null

```

下面是根据 class 获取标签结果是 ELement 并不是具体的标签，因为根据 class 无法确定标签类型

```
const el = document.querySelector('.hd') //const el: Element | null

```

#null 处理

针对于其他标签元素，返回值可能为 null，所以使用 as 断言或！处理

```
let div = document.querySelector('div') as HTMLDivElement//const div:
HTMLDivElement
//或使用
div = document.querySelector('div')! //非空断言
console.log(div.id);

```

#断言处理

使用`as` 将类型声明为 `HTMLAnchorElement` 则 TS 会将其按 a 链接类型处理

- 现在所有的提示将是 a 链接属性或方法

```
const el = document.querySelector('.hd') as HTMLAnchorElement //const el: HTMLAnchorElement
console.log(el.href);

```

下例中的 DOM 类型会报错，因为.hd 是 Element 类型，而构建函数参数 el 的类型是 `HTMLDivElement`

```
class Hd {
    (el: HTMLDivElement) {
    }
}
const el = document.querySelector('.hd'); //el: Element
new Hd(el)

```

这时可以使用 as 断言处理，明确告之获取的值类型是 `HTMLDivElement`

```
class Hd {
    (el: HTMLDivElement) {
    }
}
const el = document.querySelector('.hd') as HTMLDivElement;
new Hd(el)

```

#事件处理

下面提取按钮元素并添加事件，实现插入元素的功能

```
const body = document.querySelector('body')
const bt = document.querySelector('#bt') as HTMLButtonElement

bt.addEventListener('click', (e: Event) => {
    e.preventDefault(); //因为设置了 e 的类型，所以会有完善的提示
```

```
    body.insertAdjacentHTML('beforeend', "<div>后盾人-向军</div>")
})

```