1. Enum类型简介

Enum是ts新增的一种数据结构和类型，称为枚举。

相当于一个容器，用来存放常量，因为开发中经常需要定义一组相关的常量。使用时就跟对象属性写法一样。

```
enum Color {
  Red,     // 0
  Green,   // 1
  Blue     // 2
}
let c = Color.Green; // 1
// 等同于
let c = Color['Green']; // 1
let c:Color = Color.Green; // 正确
let c:number = Color.Green; // 正确
```

Enum 结构本身也是一种类型。比如，上例的变量`c`等于`1`，它的类型可以是 Color，也可以是`number`。

Enum 结构的**特别之处在于，它既是一种类型，也是一个值。绝大多数 ts 语法都是类型语法，编译后会全部去除，但是 Enum 结构是一个值，编译后会变成 JavaScript 对象，留在代码中。**

```
// 编译前
enum Color {
  Red,     // 0
  Green,   // 1
  Blue     // 2
}

// 编译后
let Color = {
  Red: 0,
  Green: 1,
  Blue: 2
};
```

Enum结构编译后是一个对象，所以不能有同名的对象、函数和类等。

2. Enum成员的值

Enum 成员默认不必赋值，系统会从零开始逐一递增，按照顺序为每个成员赋值，比如0、1、2.....也可以为 Enum 成员显式赋值，可以是任意数值，但不能是大整数(Bigint)。

成员的值可以相同。如果只设置第一个成员的值，后面的成员的值默认递增。成员的值也可以使用计算式。所有成员的值都是只读的，不能修改。

因为成员的值不能修改，所以可以加上const修饰，表示常量，加上const编译后不会转成对象，而是变成对应的常量。如果加上const，还想转成对象，可以将编译选项**preserveConstEnums**打开。

```ts
const enum Color {
  Red,
  Green,
  Blue
}

const x = Color.Red;
const y = Color.Green;
const z = Color.Blue;

// 编译后
const x = 0 /* Color.Red */;
const y = 1 /* Color.Green */;
const z = 2 /* Color.Blue */;
```

3. 同名Enum的合并

就是多个同名的Enum结构会自动合并

```ts
enum Foo {
  A,
}

enum Foo {
  B = 1,
}

enum Foo {
  C = 2,
}

// 等同于
enum Foo {
  A,
  B = 1,
  C = 2
}
```

合并规则

- 只允许其中一个的首成员省略初始化，否则会报错，就是多个同名的，只能有一个的首成员可以省略初始值，其他的首成员必须初始化值，对于不是首成员的不管。
- 合并时不能有同名成员，否则报错
- 必须同为const或这没有const，不能混合

特点：补充外部定义的Enum结构

4. 字符串Enum

Enum成员不仅可以设为数值，也能设为字符串。也就是一组相关的字符串集合。

对于字符串枚举的成员必须定义值，要是不定义值默认为数值，并且需要在显示设置的值的成员前面。

成员可以是字符串也是数值，不允许使用其他类型的值。

如果变量类型是字符串Enum，就不能在进行赋值为字符串，跟数值Enum不一样。

```
enum MyEnum {
  One = 'One',
  Two = 'Two',
}

let s = MyEnum.One;
s = 'One'; // 报错
```

因为变量类型为字符串Enum时，不能再进行修改，所以如果函数的参数类型是字符串Enum时，直接传入字符串会报错，可以起到限定函数参数的作用。

字符串Enum的成员值不能使用表达式赋值。

```
enum MyEnum {
  A = 'one',
  B = ['T', 'w', 'o'].join('') // 报错
}
```

5. keyof运算符

keyof运算符可以取出Enum结构的所有成员名，返回联合类型。

在使用的使用必须使用typeof，因为Enum类型本质是number和string的一种变体。如果不使用typeof就相当于keyof number,而有了typeof，typeof会把一个值转为对象类型，然后keyof运算符返回该对象的所有属性名。

```
enum MyEnum {
  A = 'a',
  B = 'b'
}

// 'A'|'B'
type Foo = keyof typeof MyEnum;
```

如果要返回Enum所有的成员值，可以使用in运算符

```
enum MyEnum {
  A = 'a',
  B = 'b'
}

// { a: any, b: any }
type Foo = { [key in MyEnum]: any };
```

6. 反向映射

对于数值Enum可以通过成员值获得成员名。

```
enum Weekdays {
  Monday = 1,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday
}

console.log(Weekdays[3]) // Wednesday
```