枚举

使用枚举我们可以定义一些有名字的数字常量。
枚举通过`enum`关键字来定义。

```
enum Direction {
    Up = 1,
    Down,
    Left,
    Right
}
```

一个枚举类型可以包含零个或多个枚举成员。
枚举成员具有一个数字值，它可以是*常数*或是*计算得出的值*
当满足如下条件时，枚举成员被当作是常数：

- 不具有初始化函数并且之前的枚举成员是常数。
  在这种情况下，当前枚举成员的值为上一个枚举成员的值加1。
  但第一个枚举元素是个例外。
  如果它没有初始化方法，那么它的初始值为`0`。
- 枚举成员使用*常数枚举表达式*初始化。
  常数枚举表达式是TypeScript表达式的子集，它可以在编译阶段求值。
  当一个表达式满足下面条件之一时，它就是一个常数枚举表达式：
  - 数字字面量
  - 引用之前定义的常数枚举成员（可以是在不同的枚举类型中定义的）
    如果这个成员是在同一个枚举类型中定义的，可以使用非限定名来引用。
  - 带括号的常数枚举表达式
  - `+`, `-`, `~` 一元运算符应用于常数枚举表达式
  - `+`, `-`, `*`, `/`, `%`, `<<`, `>>`, `>>>`, `&`, `|`, `^` 二元运算符，常数枚举表达式做为其一个操作对象
    若常数枚举表达式求值后为`NaN`或`Infinity`，则会在编译阶段报错。

所有其它情况的枚举成员被当作是需要计算得出的值。

```
enum FileAccess {
    // constant members
    None,
    Read    = 1 << 1,
    Write   = 1 << 2,
    ReadWrite  = Read | Write,
    // computed member
    G = "123".length
}
```

枚举是在运行时真正存在的一个对象。
其中一个原因是因为这样可以从枚举值到枚举名进行反向映射。

```
enum Enum {
    A
}
let a = Enum.A;
```

```
let nameOfA = Enum[Enum.A]; // "A"
```

编译成：

```
var Enum;
(function (Enum) {
    Enum[Enum["A"] = 0] = "A";
})(Enum || (Enum = {}));
var a = Enum.A;
var nameOfA = Enum[Enum.A]; // "A"
```

生成的代码中，枚举类型被编译成一个对象，它包含双向映射（`name -> value`）和（`value -> name`）。
引用枚举成员总会生成一次属性访问并且永远不会内联。
在大多数情况下这是很好的并且正确的解决方案。
然而有时候需求却比较严格。
当访问枚举值时，为了避免生成多余的代码和间接引用，可以使用常数枚举。
常数枚举是在`enum`关键字前使用`const`修饰符。

```
const enum Enum {
    A = 1,
    B = A * 2
}
```

常数枚举只能使用常数枚举表达式并且不同于常规的枚举的是它们在编译阶段会被删除。
常数枚举成员在使用的地方被内联进来。
这是因为常数枚举不可能有计算成员。

```
const enum Directions {
    Up,
    Down,
    Left,
    Right
}

let directions = [Directions.Up, Directions.Down, Directions.Left, Directions.R
```

生成后的代码为：

```
var directions = [0 /* Up */, 1 /* Down */, 2 /* Left */, 3 /* Right */];
```

外部枚举

外部枚举用来描述已经存在的枚举类型的形状。

```
declare enum Enum {
    A = 1,
    B,
    C = 2
}
```

外部枚举和非外部枚举之间有一个重要的区别，在正常的枚举里，没有初始化方法的成员被当成常数成员。
对于非常数的外部枚举而言，没有初始化方法时被当做需要经过计算的。