

Symbol 类型

Symbol 是 ES2015 新引入的一种原始类型的值。它类似于字符串，但是每一个 Symbol 值都是独一无二的，与其他任何值都不相等。

Symbol 值通过 `Symbol()` 函数生成。在 TypeScript 里面，Symbol 的类型使用 `symbol` 表示。

```
let x:symbol = Symbol();
let y:symbol = Symbol();

x === y; // false
```

上例中，变量 `x` 和 `y` 的类型都是 `symbol`，且都用 `Symbol()` 生成，但是它们是不相等的，因为他们都是独一无二的。

1. unique symbol

`symbol` 类型包含所有的 Symbol 值，但是无法表示某一个具体的 Symbol 值。

比如，5 是一个具体的数值，就用 5 这个字面量来表示，这也是它的值类型。但是，Symbol 值不存在字面量，必须通过变量来引用，所以写不出只包含单个 Symbol 值的那种值类型。

为了解决这个问题，TypeScript 设计了 `symbol` 的一个子类型 `unique symbol`，它表示单个的、某个具体的 Symbol 值。

因为 `unique symbol` 表示单个值，所以这个类型的变量是不能修改值的，只能用 `const` 命令声明，不能用 `let` 声明。

```
const x:unique symbol = Symbol();
let y:unique symbol = Symbol(); // A variable whose type is a 'unique symbol' type
must be 'const'.类型为“唯一符号”类型的变量必须为“const”。
```

`const` 命令为变量赋值 Symbol 值时，变量类型默认就是 `unique symbol`，所以类型可以省略不写。

```
const x:unique symbol = Symbol();
// 等同于
const x = Symbol();
```

每个声明为 `unique symbol` 类型的变量，它们的值都是不一样的，其实属于两个值类型。

```
const a:unique symbol = Symbol();
const b:unique symbol = Symbol();
```

```
a === b // This comparison appears to be unintentional because the types 'typeof a' and 'typeof b' have no overlap.这种比较似乎是无意的，因为类型'typeof a'和'typeof b'没有重叠。
```

上例中，变量 `a` 和变量 `b` 的类型虽然都是 `unique symbol`，但其实是两个值类型。不同类型的值肯定是不相等的，所以最后一行就报错了。

而且，由于变量 `a` 和 `b` 是两个类型，就不能把一个赋值给另一个。

```
const a:unique symbol = Symbol();
const b:unique symbol = a; // 报错
```

上例中，变量 `a` 和变量 `b` 的类型都是 `unique symbol`，但是其实类型不同，所以把 `a` 赋值给 `b` 会报错。

上例变量 `b` 的类型，如果要写成与变量 `a` 同一个 `unique symbol` 值类型，只能写成类型为 `typeof a`。

```
const a:unique symbol = Symbol();
const b:typeof a = a; // 正确
```

`unique symbol` 类型是 `symbol` 类型的子类型，所以可以将前者赋值给后者，但是反过来就不行。

```
const a:unique symbol = Symbol();
const b:symbol = a; // 正确
const c:unique symbol = b; // Type 'symbol' is not assignable to type 'unique symbol'.
```

`unique symbol` 类型的一个作用，就是用作属性名，这可以保证不会跟其他属性名冲突。如果要把某一个特定的 `Symbol` 值当作属性名，那么它的类型只能是 `unique symbol`，不能是 `symbol`。

```
const x:unique symbol = Symbol();
const y:symbol = Symbol();

interface Foo {
  [x]: string; // 正确
  [y]: string; // A computed property name in an interface must refer to an
expression whose type is a literal type or a 'unique symbol' type.接口中的计算属性
名称必须引用其类型为文字类型或“唯一符号”类型的表达式。
}
```

`unique symbol` 类型也可以用作类（class）的属性值，但只能赋值给类的 `readonly static` 属性。

```
class C {
  static readonly foo:unique symbol = Symbol();
}
```

```
}
```

上例中，静态只读属性 `foo` 的类型就是 `unique symbol`。注意，这时 `static` 和 `readonly` 两个限定符缺一不可，这是为了保证这个属性是固定不变的。

2. 类型推断

如果变量声明时没有给出类型，TypeScript 会推断某个 `Symbol` 值变量的类型。

`let`命令声明的变量，推断类型为 `symbol`。

```
// 类型为 symbol  
let x = Symbol();
```

`const`命令声明的变量，推断类型为 `unique symbol`。

```
// 类型为 unique symbol  
const x = Symbol();
```

但是，`const`命令声明的变量，如果赋值为另一个 `symbol` 类型的变量，则推断类型为 `symbol`。

```
let x = Symbol();  
  
// 类型为 symbol  
const y = x;
```

`let` 命令声明的变量，如果赋值为另一个 `unique symbol` 类型的变量，则推断类型还是 `symbol`。

```
const x = Symbol();  
  
// 类型为 symbol  
let y = x;
```