TypeScript 的 symbol 类型

简介

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。

typescript

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

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

上面示例中，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。

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。

typescript

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

// 报错
let y: unique symbol = Symbol();
```

上面示例中， `let` 命令声明的变量，不能是 `unique symbol` 类型，会报错。

`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。

typescript

```
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。

typescript

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

a === b; // 报错
```

上面示例中，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。

由于 Symbol 类似于字符串，可以参考下面的例子来理解。

typescript

```
const a: "hello" = "hello";
const b: "world" = "world";

a === b; // 报错
```

上面示例中，变量 `a` 和 `b` 都是字符串，但是属于不同的值类型，不能使用严格相等运算符进行比较。

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。

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

上面示例中，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` 。

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

不过我们知道，相同参数的 `Symbol.for()` 方法会返回相同的 Symbol 值。TypeScript 目前无法识别这种情况，所以可能出现多个 unique symbol 类型的变量，等于同一个 Symbol 值的情况。

```typescript
const a: unique symbol = Symbol.for("foo");
const b: unique symbol = Symbol.for("foo");
```

上面示例中，变量 `a` 和 `b` 是两个不同的值类型，但是它们的值其实是相等的。

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。

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

const b: symbol = a; // 正确

const c: unique symbol = b; // 报错
```

上面示例中，unique symbol 类型（变量 `a` ）赋值给 symbol 类型（变量 `b` ）是可以的，但是 symbol 类型（变量 `b` ）赋值给 unique symbol 类型（变量 `c` ）会报错。

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。

typescript

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

interface Foo {
  [x]: string; // 正确
  [y]: string; // 报错
}
```

上面示例中，变量 `y` 当作属性名，但是 `y` 的类型是 symbol，不是固定不变的值，导致报错。

`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。

typescript

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

上面示例中，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。

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。

typescript

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

`const` 命令声明的变量，推断类型为 unique symbol。

typescript

```typescript
// 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。

typescript

```typescript
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。

typescript

```typescript
const x = Symbol();

// 类型为 symbol
let y = x;
```


最后更新: 2023/8/13 15:25

Previous page
元组

Next page
函数