23 前面跳过的unkown类型详解

更新时间：2019-07-02 14:25:09

> 人生的旅途，前途很远，也很暗。然而不要怕，不怕的人的面前才有路。
>
> —— 鲁迅

学习完交叉类型、联合类型、类型断言、映射类型、索引后，我们就可以补充一个基础类型中没有讲的知识了，就是 TS 在 3.0 版本新增的顶级类型 unknown。它相对于 any 来说是安全的。关于 unknown 类型，有如下几点需要注意，我们来逐个讲解和举例学习：

(1) 任何类型的值都可以赋值给 unknown 类型：

```
let value1: unknown;
value1 = "a";
value1 = 123;
```

(2) 如果没有类型断言或基于控制流的类型细化时 unknown 不可以赋值给其它类型，此时它只能赋值给 unknown 和 any 类型：

```
let value2: unknown;
let value3: string = value2; // error 不能将类型“unknown”分配给类型“string”
value1 = value2;
```

(3) 如果没有类型断言或基于控制流的类型细化，则不能在它上面进行任何操作：

```
let value4: unknown;
value4 += 1; // error 对象的类型为 "unknown"
```

(4) unknown 与任何其它类型组成的交叉类型，最后都等于其它类型：

```
type type1 = unknown & string; // type1 => string
type type2 = number & unknown; // type2 => number
type type3 = unknown & unknown; // type3 => unknown
type type4 = unknown & string[]; // type4 => string[]
```

(5) unknown 与任何其它类型组成的联合类型，都等于 unknown 类型，但只有any例外，unknown与any组成的联合类型等于any)：

```
type type5 = string | unknown; // type5 => unknown
type type6 = any | unknown; // type6 => any
type type7 = number[] | unknown; // type7 => unknown
```

(6) never 类型是 unknown 的子类型：

```
type type8 = never extends unknown ? true : false; // type8 => true
```

(7) keyof unknown 等于类型 never：

```
type type9 = keyof unknown; // type9 => never
```

(8) 只能对 unknown 进行等或不等操作，不能进行其它操作：

```
value1 === value2;
value1 !== value2;
value1 += value2; // error
```

(9) unknown 类型的值不能访问其属性、作为函数调用和作为类创建实例：

```
let value5: unknown;
value5.age; // error
value5(); // error
new value5(); // error
```

(10) 使用映射类型时如果遍历的是 unknown 类型，则不会映射任何属性：

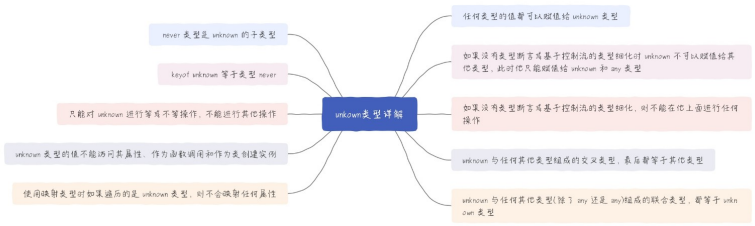
```
type Types<T> = { [P in keyof T]: number };
type type10 = Types<any>; // type10 => { [x: string]: number }
type type11 = Types<unknown>; // type10 => {}
```

我们在实际使用中，如果有类型无法确定的情况，要尽量避免使用 any，因为 any 会丢失类型信息，一旦一个类型被指定为 any，那么在它上面进行任何操作都是合法的，所以会有意想不到的情况发生。因此如果遇到无法确定类型的情况，要先考虑使用 unknown。

本节小结

本小节我们详细学习了 unknown 类型，它和 any 有相似的特点，就是制定一个类型是任意的，但是区别在于制定一个类型为 any 的话，可以在这个值上做任意操作，而 unknown 类型则不允许在没有类型断言或基于控制流的类型细化时对 unknown 类型的值做任何操作。

下个小节我们将学习条件类型，它看起来像是三元操作符的写法，其实效果确实很像，只不过它判断的是类型，返回的结果也是类型。



22 使用映射类型得到新的类型

24 条件类型，它不是三元操作符的写法吗？