21 索引类型：获取索引类型和索引值类型

更新时间：2019-06-28 11:48:42

> 横眉冷对千夫指，俯首甘为孺子牛。
>
> ——鲁迅

我们这里要讲的，可不是前面讲接口的时候讲的索引类型。在学习接口内容的时候，我们讲过可以指定索引的类型。而本小节我们讲的索引类型包含两个内容：**索引类型查询**和**索引访问**操作符。

3.8.1 索引类型查询操作符

`keyof` 操作符，连接一个类型，会返回一个由这个类型的所有属性名组成的联合类型。来看例子：

```
interface Info {
  name: string;
  age: number;
}
let infoProp: keyof Info;
infoProp = "name";
infoProp = "age";
infoProp = "no"; // error 不能将类型“"no"”分配给类型“"name" | "age"”
```

通过例子可以看到，这里的 `keyof Info` 其实相当于 `"name" | “age”`。通过和泛型结合使用，TS 就可以检查使用了动态属性名的代码：

```
function getValue<T, K extends keyof T>(obj: T, names: K[]): T[K][] { // 这里使用泛型，并且约束泛型变量K的类型是"keyof T"，也就是类型T的所有字段名组成的联合类型
  return names.map(n => obj[n]); // 指定getValue的返回值类型为T[K][]，即类型为T的值的属性值组成的数组
}
const info = {
  name: "lison",
  age: 18
};
let values: string[] = getValue(info, ["name"]);
values = getValue(info, ["age"]); // error 不能将类型“number[]”分配给类型“string[]”
```

3.8.2 索引访问操作符

索引访问操作符也就是`[]`，其实和我们访问对象的某个属性值是一样的语法，但是在 TS 中它可以用来访问某个属性的类型：

```
interface Info {
  name: string;
  age: number;
}
type NameType = Info["name"];
let name: NameType = 123; // error 不能将类型“123”分配给类型“string”
```

再来看个例子：

```
function getProperty<T, K extends keyof T>(o: T, name: K): T[K] {
  return o[name]; // o[name] is of type T[K]
}
```

这个函数中，两个参数的类型分别为泛型 T 和 K，而函数的返回值类型为`T[K]`，只要函数的返回值也是这种形式，即访问参数 o 的参数 name 属性，即可。

最后我们来看个结合接口的例子：

```
interface Obj<T> {
  [key: number]: T;
}
const key: keyof Obj<number>; // keys的类型为number
```

这里需要注意，在讲接口一节时，讲索引类型的时候我们讲过，如果索引类型为 number，那么实现该接口的对象的属性名必须是 number 类型；但是如果接口的索引类型是 string 类型，那么实现该接口的对象的属性名设置为数值类型的值也是可以的，因为数值最后还是会先转换为字符串。这里一样，如果接口的索引类型设置为 string 的话，`keyof Obj<number>`等同于类型`number | string`：

```
interface Obj<T> {
  [key: string]: T;
}
let key: keyof Obj<number>; // keys的类型为number | string
key = 123; // right
```

也可以使用访问操作符，获取索引签名的类型：

```
interface Obj<T> {
  [key: string]: T;
}
const obj: Obj<number> = {
  age: 18
};
let value: Obj<number>["age"]; // value的类型是number，也就是name的属性值18的类型
```

还有一点，我们在讲后面知识的时候会遇到，就是当tsconfig.json里`strictNullChecks`设为`false`时，通过`Type[keyof Type]`获取到的，是除去`never & undefined & null`这三个类型之后的字段值类型组成的联合类型，来看例子：

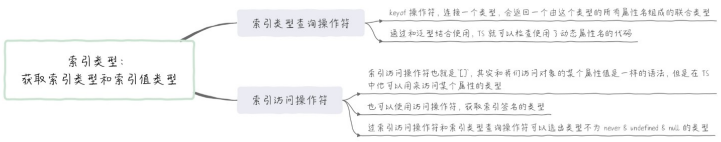
```ts
interface Type {
  a: never;
  b: never;
  c: string;
  d: number;
  e: undefined;
  f: null;
  g: object;
}
type test = Type[keyof Type];
// test的类型是string | number | object
```

这个例子中接口 Type 有几个属性，通过索引访问操作符和索引类型查询操作符可以选出类型不为 never & undefined & null 的类型。

本节小结

本小节我们学习了两个类型操作符：索引类型查询操作符 keyof，和索引访问操作符 []。通过keyof我们能够获取一个类型的所有属性名组成的联合类型，通过[]我们可以获取某个类型定义中指定字段值的类型。我们还学习了它们的组合使用方法，当tsconfig.json里 strictNullChecks 设为 false 时，我们可以通过 [keyof Type] 获取一个类型定义的所有除去 never & undefined & null 的字段值的类型组成的联合类型。

下个小节我们将学习一种新的复用现有类型定义，产生新类型定义的一种类型——映射类型。



20 this，类型？

22 使用映射类型得到新的类型