20 this，类型？

更新时间：2019-06-27 18:00:07

> 每个人都是自己命运的主宰。
>
> ——斯蒂尔斯

在 JavaScript 中，this 可以用来获取对全局对象、类实例对象、构建函数实例等的引用，在 TypeScript 中，this 也是一种类型，我们先来看个计算器 Counter 的例子：

```
class Counter {
  constructor(public count: number = 0) {}
  add(value: number) { // 定义一个相加操作的方法
    this.count += value;
    return this;
  }
  subtract(value: number) { // 定义一个相减操作的方法
    this.count -= value;
    return this;
  }
}
let counter = new Counter(10);
console.log(counter.count); // 10
counter.add(5).subtract(2);
console.log(counter.count); // 13
```

我们给 Counter 类定义几个方法，每个方法都返回 this，这个 this 即指向实例，这样我们就可以通过链式调用的形式来使用这些方法。这个是没有问题的，但是如果我们要通过类继承的形式丰富这个 Counter 类，添加一些方法，依然返回 this，然后采用链式调用的形式调用，在过去版本的 TypeScript 中是有问题的，先来看我们继承的逻辑：

```
class PowCounter extends Counter {
  constructor(public count: number = 0) {
    super(count);
  }
  pow(value: number) { // 定义一个幂运算操作的方法
    this.count = this.count ** value;
    return this;
  }
}
let powcounter = new PowCounter(2);
powCounter
  .pow(3)
  .subtract(3)
  .add(1);
console.log(powCounter.count); // 6
```

我们定义了 PowCounter 类，它继承 Counter 类，新增了 pow 方法用来求值的幂次方，这里我们使用了 ES7 新增的幂运算符 **。我们使用 PowCounter 创建了实例 powcounter，它的类型自然是 PowCounter，在该实例上调用继承来的 subtract 和 add 方法。如果是在过去，就会报错，因为创建实例 powcounter 的类 PowCounter 没有定义这两个方法，所以会报没有这两个方法的错误。但是在 1.7 版本中增加了 this 类型，TypeScript 会对方法返回的 this 进行判断，就不会报错了。

对于对象来说，对象的属性值可以是一个函数，那么这个函数也称为方法，在方法内如果访问this，默认情况下是对这个对象的引用，this类型也就是这个对象的字面量类型，如下：

```
// 例3.7.1
let info = {
  name: 'Lison',
  getName () {
     return this.name // "Lison" 这里this的类型为 { name: string; getName(): string; }
  }
}
```

但是如果显式地指定了this的类型，那么this的类型就改变了，如下：

```
// 例3.7.2
let info = {
  name: "Lison",
  getName(this: { age: number }) {
    this; // 这里的this的类型是{ age: number }
  }
};
```

如果我们在 tsconfig.json 里将 noImplicitThis 设为 true，这时候有两种不同的情况：

(1) 对象字面量具有 ThisType<T> 指定的类型，此时 this 的类型为 T，来看例子：

```
type ObjectDescriptor<D, M> = { // 使用类型别名定义一个接口，这里用了泛型，两个泛型变量D和M
  data?: D; // 这里指定data为可选字段，类型为D
  // 这里指定methods为可选字段，类型为M和ThisType<D & M>组成的交叉类型；
  // ThisType是一个内置的接口，用来在对象字面量中键入this，这里指定this的类型为D & M
  methods?: M & ThisType<D & M>;
}

// 这里定义一个mackObject函数，参数desc的类型为ObjectDescriptor<D, M>
function makeObject<D, M>(desc: ObjectDescriptor<D, M>): D & M {
  let data: object = desc.data || {};
  let methods: object = desc.methods || {};
  // 这里通过...操作符，将data和methods里的所有属性、方法都放到了同一个对象里返回，这个对象的类型自然就     是D & M，因为他同时包含D和M
两个类型的字段
  return { ...data, ...methods } as D & M;
}

let obj = makeObject({
  data: { x: 0, y: 0 }, // 这里data的类型就是我们上面定义ObjectDescriptor<D, M>类型中的D
  methods: { // 这里methods的类型就是我们上面定义ObjectDescriptor<D, M>类型中的M
    moveBy(dx: number, dy: number) {
      this.x += dx;  // 所以这里的this是我们通过ThisType<D & M>指定的，this的类型就是D & M
      this.y += dy;
    }
  }
});

obj.x = 10;
obj.y = 20;
obj.moveBy(5, 5);
```

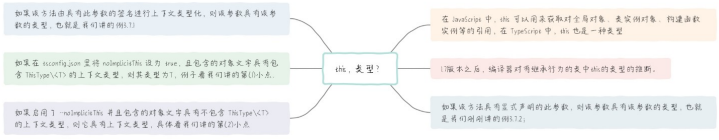
(2) 不包含 ThisType<T> 指定的上下文类型，那么此时 this 具有上下文类型，也就是普通的情况。你可以试着把上面使用了 ThisType<T> 的例子中，ObjectDescriptor<D, M>类型中指定methods的类型中的 `& ThisType<D & M>` 去掉，你会发现 `moveBy` 方法中 `this.x` 和 `this.y` 报错了，因为此时 `this` 的类型是`methods` 这个对象字面量的类型。

本节小结

本小节我们学习了this类型的相关知识，我们通过计数器的例子，学习了在1.7版本之后，编译器对有继承行为的类中this的类型的推断。还学习了对于对象的方法中，this指向的相关知识。更多的关于this类型的知识，可以看一下这个PR中的介绍及例子，这里面完整地写了this的类型的规则。不过我们上面都举例学习了，总结一下：

- 如果该方法具有显式声明的此参数，则该参数具有该参数的类型，也就是我们刚刚讲的例3.7.2；
- 否则，如果该方法由具有此参数的签名进行上下文类型化，则该参数具有该参数的类型，也就是我们讲的例3.7.1；
- 否则，如果在 tsconfig.json 里将 noImplicitThis 设为 true，且包含的对象文字具有包含 ThisType<T> 的上下文类型，则其类型为T，例子看我们讲的第(1)小点.
- 否则，如果启用了 --noImplicitThis 并且包含的对象文字具有不包含 ThisType<T> 的上下文类型，则它具有上下文类型，具体看我们讲的第(2)小点。
- 否则，this 的类型为 any 任何类型。

下个小节我们将学习索引类型，这里说的索引类型，并不是前面我们讲接口的时候，给接口中字段名设置类型，我们将学习获取索引类型和索引值类型。



19 使用可辨识联合并保证每个case都被处理

21 索引类型：获取索引类型和索引值类型