16 使用类型保护让TS更聪明

更新时间：2019-06-21 09:38:41

> 世界上最宽阔的是海洋，比海洋更宽阔的是天空，比天空更宽阔的是人的胸怀。
>
> ——雨果

这个小节我们来学习类型保护，在学习前面知识的时候我们有遇到过需要告诉编译器某个值是指定类型的场景，当时我们使用的是类型断言，这一节我们来看一个不同的场景：

```
const valueList = [123, "abc"];
const getRandomValue = () => {
  const number = Math.random() * 10; // 这里取一个[0, 10)范围内的随机值
  if (number < 5) return valueList[0]; // 如果随机数小于5则返回valueList里的第一个值，也就是123
  else return valueList[1]; // 否则返回"abc"
};
const item = getRandomValue();
if (item.length) {
  // error 类型“number”上不存在属性“length”
  console.log(item.length); // error 类型“number”上不存在属性“length”
} else {
  console.log(item.toFixed()); // error 类型“string”上不存在属性“toFixed”
}
```

上面这个例子中，getRandomValue 函数返回的元素是不固定的，有时返回数值类型，有时返回字符串类型。我们使用这个函数生成一个值 item，然后接下来的逻辑是通过是否有 length 属性来判断是字符串类型，如果没有 length 属性则为数值类型。在 js 中，这段逻辑是没问题的，但是在 TS 中，因为 TS 在编译阶段是无法知道 item 的类型的，所以当我们在 if 判断逻辑中访问 item 的 length 属性的时候就会报错，因为如果 item 为 number 类型的话是没有 length 属性的。

这个问题我们可以先采用类型断言的方式来解决。类型断言我们学习过，就是相当于告诉 TS，这个值就是制定的类型，我们只需要修改判断逻辑即可，来看怎么写：

```
if ((<string>item).length) {
  console.log((<string>item).length);
} else {
  console.log((<number>item).toFixed());
}
```

3.3.1 自定义类型保护

上面的代码不报错是因为我们通过使用类型断言，告诉 TS 编译器，if 中的 item 是 string 类型，而 else 中的是 number 类型。这样做虽然可以，但是我们需要在使用 item 的地方都使用类型断言来说明，显然有些繁琐，所以我们就可以使用类型保护来优化。

我们先来看，本小节开头这个问题，如何使用**自定义类型保护**来解决：

```
const valueList = [123, "abc"];
const getRandomValue = () => {
  const number = Math.random() * 10; // 这里取一个[0, 10)范围内的随机值
  if (number < 5) return valueList[0]; // 如果随机数小于5则返回valueList里的第一个值，也就是123
  else return valueList[1]; // 否则返回"abc"
};
function isString(value: number | string): value is string {
  const number = Math.random() * 10
  return number < 5;
}
const item = getRandomValue();
if (isString(item)) {
  console.log(item.length); // 此时item是string类型
} else {
  console.log(item.toFixed()); // 此时item是number类型
}
```

我们看到，首先定义一个函数，函数的参数 value 就是要判断的值，在这个例子中 value 的类型可以为 number 或 string，函数的返回值类型是一个结构为 `value is type` 的类型谓语，value 的命名无所谓，但是谓语中的 value 名必须和参数名一致。而函数里的逻辑则用来返回一个布尔值，如果返回为 true，则表示传入的值类型为`is`后面的 type。

使用类型保护后，if 的判断逻辑和代码块都无需再对类型做指定工作，不仅如此，既然 item 是 string 类型，则 else 的逻辑中，item 一定是联合类型两个类型中另外一个，也就是 number 类型。

3.3.2 typeof 类型保护

但是这样定义一个函数来用于判断类型是字符串类型，难免有些复杂，因为在 JavaScript 中，只需要在 if 的判断逻辑地方使用 typeof 关键字即可判断一个值的类型。所以在 TS 中，如果是基本类型，而不是复杂的类型判断，你可以直接使用 typeof 来做类型保护：

```
if (typeof item === "string") {
  console.log(item.length);
} else {
  console.log(item.toFixed());
}
```

这样直接写也是可以的，效果和自定义类型保护一样。但是在 TS 中，对 typeof 的处理还有些特殊要求：

- 只能使用`=`和`!`两种形式来比较
- type 只能是`number`、`string`、`boolean`和`symbol`四种类型

第一点要求我们必须使用这两种形式来做比较，比如你使用`(typeof item).includes('string')`也能做判断，但是不行的。

第二点要求我们要比较的类型只能是这四种，但是我们知道，在 JS 中，`typeof xxx`的结果还有`object`、`function`和`undefined`。但是在 TS 中，只会把对前面四种类型的 typeof 比较识别为类型保护，你可以使用`typeof {} == 'object'`，但是这里它只是一条普通的 js 语句，不具有类型保护具有的效果。我们可以来看例子：

```
const valueList = [{}, () => {}];
const getRandomValue = () => {
  const number = Math.random() * 10;
  if (number < 5) {
    return valueList[0];
  } else {
    return valueList[1];
  }
};
const res = getRandomValue();
if (typeof res === "object") {
  console.log(res.toString());
} else {
  console.log(ress()); // error 无法调用类型缺少调用签名的表达式。类型“{}”没有兼容的调用签名
}
```

3.3.3 instanceof 类型保护

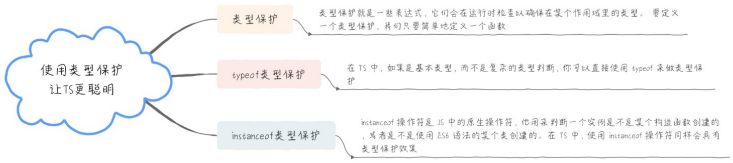
`instanceof`操作符是 JS 中的原生操作符，它用来判断一个实例是不是某个构造函数创建的，或者是不是使用 ES6 语法的某个类创建的。在 TS 中，使用 instanceof 操作符同样会具有类型保护效果，来看例子：

```
class CreateByClass1 {
  public age = 18;
  constructor() {}
}
class CreateByClass2 {
  public name = "lison";
  constructor() {}
}
function getRandomItem() {
  return Math.random() < 0.5 ? new CreateByClass1() : new CreateByClass2(); // 如果随机数小于0.5就返回CreateByClass1的实例，否则返回CreateByClass2的实例
}
const item = getRandomItem();
if (item instanceof CreateByClass1) { // 这里判断item是否是CreateByClass1的实例
  console.log(item.age);
} else {
  console.log(item.name);
}
```

这个例子中 if 的判断逻辑中使用 instanceof 操作符判断了 item 。如果是 CreateByClass1 创建的，那么它应该有 age 属性，如果不是，那它就有 name 属性。

本节小结

本小节我们学习了类型保护，通过使用类型保护可以更好地指定某个值的类型，可以把这个指定理解为一种强制转换，这样编译器就能知道我们这个值是我们指定的类型，从而符合我们的预期。**typeof** 和 **instanceof** 是 JavaScript 中的两个操作符，用来判断某个值的类型和一个值是否是某个构造函数的实例，它们在 TypeScript 中会被当做类型保护。我们也可以自定义类型保护，通过定义一个返回值类型是"参数名 is type"的语句，来指定传入这个类型保护函数的某个参数是什么类型。如果只是简单地要判断某个值是什么类型，使用 typeof 类型保护就可以了。



15 类型兼容性，开放心态满足灵活的JS

17 使用显式复制断言给TS一个你一定会赋值的承诺

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

慕粉0010189952

自定义类型保护里面的isString函数内部逻辑为什么是又写了一遍随机数判断？这和前面的函数能取到同一个随机值吗？

1 回复 4天前