

17 使用显式复制断言给TS一个你一定会赋值的承诺

更新时间：2019-06-24 17:58:39



“

人不可有傲气，但不可无傲骨。

——徐悲鸿

”

在讲解本小节的主要内容之前，我们先来补充两个关于`null`和`undefined`的知识点：

(1) 严格模式下`null`和`undefined`赋值给其它类型值

当我们在 `tsconfig.json` 中将 `strictNullChecks` 设为 `true` 后，就不能再将 `undefined` 和 `null` 赋值给除它们自身和`void`之外的任意类型值了，但有时我们确实需要给一个其它类型的值设置初始值为空，然后再进行赋值，这时我们可以自己使用联合类型来实现 `null` 或 `undefined` 赋值给其它类型：

```
let str = "lison";
str = null; // error 不能将类型"null"分配给类型"string"
let strNull: string | null = "lison"; // 这里你可以简单理解为，string | null即表示既可以是string类型也可以是null类型
strNull = null; // right
strNull = undefined; // error 不能将类型"undefined"分配给类型"string | null"
```

注意，TS 会将 `undefined` 和 `null` 区别对待，这和 JS 的本意也是一致的，所以在 TS 中，`string|undefined`、`string|`
`null` 和 `string|undefined|null` 是三种不同的类型。

(2) 可选参数和可选属性

如果开启了 `strictNullChecks`，可选参数会被自动加上 `|undefined`，来看例子：

```
const sum = (x: number, y?: number) => {
    return x + (y || 0);
};
sum(1, 2); // 3
sum(1); // 1
sum(1, undefined); // 1
sum(1, null); // error Argument of type 'null' is not assignable to parameter of type 'number | undefined'
```

可以根据错误信息看出，这里的参数 `y` 作为可选参数，它的类型就不仅是 `number` 类型了，它可以是 `undefined`，所以它的类型是联合类型 `number | undefined`。

TS 对可选属性和对可选参数的处理一样，可选属性的类型也会被自动加上 `|undefined`。

```
interface PositionInterface {
    x: number;
    b?: number;
}
const position: PositionInterface = {
    x: 12
};
position.b = "abc"; // error
position.b = undefined; // right
position.b = null; // error
```

3.4.1 显式赋值断言

接下来我们来看显式赋值断言。当我们开启 `strictNullChecks` 时，有些情况下编译器是无法在我们声明一些变量前知道一个值是否是 `null` 的，所以我们需要使用类型断言手动指明该值不为 `null`。这可能不好理解，接下来我们就来看一个编译器无法推断出一个值是否是 `null` 的例子：

```
function getSplicedStr(num: number | null): string {
    function getRes(prefix: string) { // 这里在函数getSplicedStr里定义一个函数getRes，我们最后调用getSplicedStr返回的值实际是getRes运行后的返回值
        return prefix + num.toFixed().toString(); // 这里使用参数num，num的类型为number或null，在运行前编译器是无法知道在运行时num参数的实际类型的，所以这里会报错，因为num参数可能为null
    }
    num = num || 0.1; // 但是这里进行了赋值，如果num为null则会将0.1赋给num，所以实际调用getRes的时候，getRes里的num拿到的始终不为null
    return getRes("lison");
}
```

这个例子中，因为有嵌套函数，而编译器无法去除嵌套函数的 `null`（除非是立即调用的函数表达式），所以我们需要使用显式赋值断言，写法就是在不为 `null` 的值后面加个 `!`。来看上面的例子该怎么改：

```
function getSplicedStr(num: number | null): string {
    function getLength(prefix: string) {
        return prefix + num!.toFixed().toString();
    }
    num = num || 0.1;
    return getLength("lison");
}
```

这样编译器就知道了，`num` 不为 `null`，即便 `getSplicedStr` 函数在调用的时候传进来的参数是 `null`，在 `getLength` 函数中的 `num` 也不会是 `null`。

本节小结

本小节我们补充学习了两个关于`null`和`undefined`的知识点。一个是如何在严格模式，也就是在`tsconfig.json`中将`strictNullChecks`设为`true`的情况下，将`null`或`undefined`赋值给除它们自身和`void`之外的类型的值；另一个知识点是当将`strictNullChecks`设为`true`后，编译器对可选参数和可选属性类型定义的处理，效果相当于在我们指定的类型后面加上`|undefined`。最后我们学习了如何使用显式赋值断言，它的作用就是告诉编译器某个值确实不为`null`，这个我们在实际开发中常会用到，我们在实战章节中用到时会再次学习。

下个小节我们将学习类型别名和字面量类型。类型别名我们在前面简单接触过，它的语法类似赋值语句，只不过赋的不是具体的值，而是一个类型；字面量类型我们称它为单一的类型，它包含数字字面量类型和字符串字面量类型两种，下个小节我们来详细学习。



精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

斯芬克斯01

一个是如何在严格模式，也就是在`tsconfig.json`中将`strictNullChecks`设为`true`的情况下，将`null`或`undefined`赋值给除它们自身和`void`之外的类型的 (剩下的全部的) 值 是不是更容易理解???或者更通顺?

0 回复

2019-07-01