12.泛型

2021年10月9日 16:26

泛型： 不预先确定的数据类型，具体的类型在使用的时候才能确定。

1. 泛型函数

```
// 泛型函数
function log<T>(str: T): T {
    console.log(typeof str, str);
    return str;
}
```

上述的写法保证了：

1. 输入和输出的类型是一致的
2. 输入的类型是any

调用泛型函数：

```
log<number>(12)
log<string>('log')
log([12, 'logging'])  // 运用了类型推断直接传参
```

```
// 类型别名
type Log12 = <T>(str: T) => T;
let myLog12: Log12 = log;
```

2. 泛型接口

可以将泛型也理解为一个特殊的参数，只不过这个参数是代表类型的，是用来约束其他变量的，可以用在函数的任何地方。

```
// 泛型接口 —— 指定默认类型为string
interface Log12Interace<T = string> {
    (value: T): T
}
let myLog12_2: Log12Interace = log;
myLog12_2('1');
let myLog12_3: Log12Interace<number> = log;
myLog12_3(1);
```

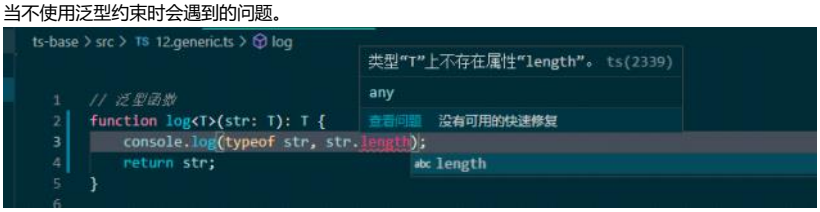
泛型优点：

1. 函数和类可以轻松地支持多种类型，增强程序的扩展性
2. 不必写多条函数重载，冗长的联合类型声明，增强代码可读性
3. 灵活控制类型之间的约束

3. 泛型类

4. 泛型约束

当不使用泛型约束时会遇到的问题。



使用泛型约束

```
interface Length {
    length: number
}
// 泛型约束 de 泛型函数
function log14<T extends Length>(str: T): T {
    console.log(typeof str, str.length);
    return str;
}
```