**释意接口**

- 类名要表达出对象

- 对属性方法不要使用缩写

- 尽量表达出接口的用途，而不表达以何种方式

**无副作用函数**

返回结果而不产生副作用的操作称为函数

let arr = [1, 2, 3, 4]

// filter 是无副作用函数

let newArr = arr.filter(e=>e > 2);

// push 不是无副作用函数

arr.push(5);

如果一个操作既有读又有写，那么应该重构为两个函数，读写分离

注：从上述2条规则我们可以看出，无副作用函数即为读函数

值对象的操作均为函数，应为值对象不能改变，想改变一个值对象，只有替换，如果值对象想改变自身属性，则应该在函数中返回一个新的值对象，然后调用者替换掉这个值对象，基于上述方法，我们可以将一些复杂的概念封装为值对象，如下Color便是一个值对象

// 将油桶颜色混合

paint.Color = Color.MixedWith(otherPaint.Color)

**概念轮廓**

在连续的重构中，对象的概念应该越来越清晰，如果在一次重构中，对象的概念又变的模糊起来，那么应该思考是否重构错误

**独立的类**

- 提取一个概念会增加系统的复杂性（了解一个系统，就必须先了解这个概念），隐式概念（在建模中提到的概念，但在模型中没有表现出来）不提取出来，同样也会增加系统复杂性

- 如果提取的概念是是一个独立的类（不依赖于其他类），会让系统变得更清晰，增加的复杂性也是很少（理解一个独立的类是秒秒中的事情）

- 如果几个类互相依赖，那么我们是否可以将依赖的部分提取为一个独立的类，让这几个类依赖于独立的类

**闭合操作**

- 执行一个类的操作，传入的参数类型与当前类类型相同，返回与当前类类型相同的类型

- 闭合操作多用于值对象，当执行值对象的操作时，返回一个新的值对象

半个闭合操作：

传入的参数类型与当前类类型不同或返回的类型与当前类类型不同

半个闭合操作同样也具有闭合操作的优点

如我们所熟悉的集合类的过滤函数等均属于闭合操作

注：闭合操作属于无副作用函数

**声明式设计**

即我们声明了一个东西，让系统自己去应用这个东西，如

- 我们声明了一个授权类，但我们不将他添加到系统中，系统启动时会自动检测并添加进去

- 我们在方法上面标志一个特性，系统执行时会自动添加事务

我们通常使用反射实现声明式设计