//协议：方法、属性或一段功能的预定义

protocol AProtocol{

}

protocol BProtocol{

}

//遵从协议，多个协议之间可用","分隔

struct Astruct: AProtocol, BProtocol{

}

class Name{

}

class GivenName: Name, AProtocol{

}

//结构体、枚举就可以继承了一些功能。把功能抽取，变成小的功能，灵活。

//属性协议，要求遵从者实现以指定的名称实现属性，但具体实现是实例属性或类型属性并不关心

//可以指定要求实现getter或getter+setter，属性必须定义为变量，协议中的属性必须跟{get}或者{get set}

protocol FullName{

var fName: String{ get }

var gName: String{ get }

}

//遵守协议，用:

struct Student: FullName{

var gName: String

var fName: String

}

struct Teacher: FullName{

var gName: String

var fName: String

}

var student1 = Student(gName: "三丰", fName: "张")

student1.gName

student1.fName

class SomeBody: FullName{

var title: String? //optional，可选类型

var name: String

init(title: String?, name: String) {

self.title = title

self.name = name

}

var gName: String{

return name

}

var fName: String{

//title有值，返回值，为nil时，返回默认值“普通员工”

return title ?? "普通员工"

}

}

var somebody1 = SomeBody(title: nil, name: "李四")

somebody1.gName

somebody1.fName

//方法协议，定义时没有花括号执行体，可以通过协议将方法分解为更小的组合

//类型方法协议：前缀只能是static

protocol AMethod{

static func foo()

}

class A:AMethod{

static func foo() {

print("Hello!")

}

}

//实例方法协议，协议名称一般可以用形容词，用来描述协议能做什么

protocol RandomGeneratable{

//arc4random()生成随机数，返回值类型是UInt32

func randomNumber() -> Int

}

struct RandomNumber: RandomGeneratable{

func randomNumber() -> Int {

//arc4random\_uniform(6),返回值是0...5

return Int(arc4random())%6+1

}

}

struct RandomNumberInSix: RandomGeneratable{

func randomNumber() -> Int {

return Int(arc4random\_uniform(6))

}

}

let random1 = RandomNumber()

random1.randomNumber()

let random2 = RandomNumberInSix()

random2.randomNumber()

//协议作为类型使用：可用于参数类型/返回类型，变量/常量/属性，集合类型中的元素类型

struct Dice{

var sides: Int

//协议作为数据类型使用

var randomNumber: RandomGeneratable

func play() -> Int {

return self.randomNumber.randomNumber() % sides + 1

}

}

//randomNumber传递了遵守协议的结构体RandomNumber()

let aDice = Dice(sides: 6, randomNumber: RandomNumber())

aDice.play()

mutating主要用在Swift中的：结构体（struct）、枚举（enum）、扩展（extension）、协议（protocol）中修饰方法。

Swift结构体（struct）、枚举（enum） 中均可以包含类方法和实例方法。

在使用 class 来实现带有 mutating 的方法的协议时，方法前面是不需要加 mutating 修饰的，因为 class 可以随意更改自己的成员变量。因此在协议里用 mutating 修饰方法，对于 class 的实现是完全透明，可以当作不存在的。

//枚举的协议

protocol Switchable{

mutating func onOff()

}

//注意：在修改self的值时，要在func前加mutating

enum MySwitch: Switchable{

case on, off

mutating func onOff() {

switch self{

case .off:

self = .on

default:

self = .off

}

}

}

var myswitch = MySwitch.off

myswitch.onOff()