Swift 协议

协议规定了用来实现某一特定功能所必需的方法和属性。

任意能够满足协议要求的类型被称为遵循(conform)这个协议。

类，结构体或枚举类型都可以遵循协议，并提供具体实现来完成协议定义的方法和功能。

```
protocol classa {

    var marks: Int { get set }
    var result: Bool { get }

    func attendance() -> String
    func markssecured() -> String

}

protocol classb: classa {

    var present: Bool { get set }
    var subject: String { get set }
    var stname: String { get set }

}

class classc: classb {
    var marks = 96
    let result = true
    var present = false
    var subject = "Swift 协议"
    var stname = "Protocols"

    func attendance() -> String {
        return "The \(stname) has secured 99% attendance"
    }

    func markssecured() -> String {
        return "\(stname) has scored \(marks)"
    }
}

let studdet = classc()
```

```
studdet.stname = "Swift"
studdet.marks = 98
studdet.markssecured()
```

委托

```
//定义一个协议
protocol LogManagerDelegate {
    func writeLog()
}

//用户登录类
class UserController {
    var delegate : LogManagerDelegate?

    func login() {
        //查看是否有委托，然后调用它
        delegate?.writeLog()
    }
}

//日志管理类
class SqliteLogManager : LogManagerDelegate {
    func writeLog() {
        print("将日志记录到 sqlite 数据库中")
    }
}


//使用
let userController = UserController()
userController.login()   //不做任何事

let sqliteLogManager = SqliteLogManager()
userController.delegate = sqliteLogManager
userController.login()   //输出 “将日志记录到 sqlite 数据库中”
```

泛型

Swift 提供了泛型让你写出灵活且可重用的函数和类型。

```
func swapTwoStrings(_ a: inout String, _ b: inout String) {
    let temporaryA = a
    a = b
    b = temporaryA
}

func swapTwoDoubles(_ a: inout Double, _ b: inout Double) {
    let temporaryA = a
    a = b
    b = temporaryA
}

// 定义一个交换两个变量的函数
func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

swapTwoValues 后面跟着占位类型名（T），并用尖括号括起来（<T>）。这个尖括号告诉 Swift 那个 T 是 swapTwoValues(_:_:) 函数定义内的一个占位类型名，因此 Swift 不会去查找名为 T 的实际类型。

泛型的栈

```
struct Stack<Element> {
    var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
}



var stackOfStrings = Stack<String>()
print("字符串元素入栈: ")
stackOfStrings.push("google")
stackOfStrings.push("runoob")
print(stackOfStrings.items);

let deletetos = stackOfStrings.pop()
print("出栈元素: " + deletetos)

var stackOfInts = Stack<Int>()
```

```
print("整数元素入栈: ")
stackOfInts.push(1)
stackOfInts.push(2)
print(stackOfInts.items);
```