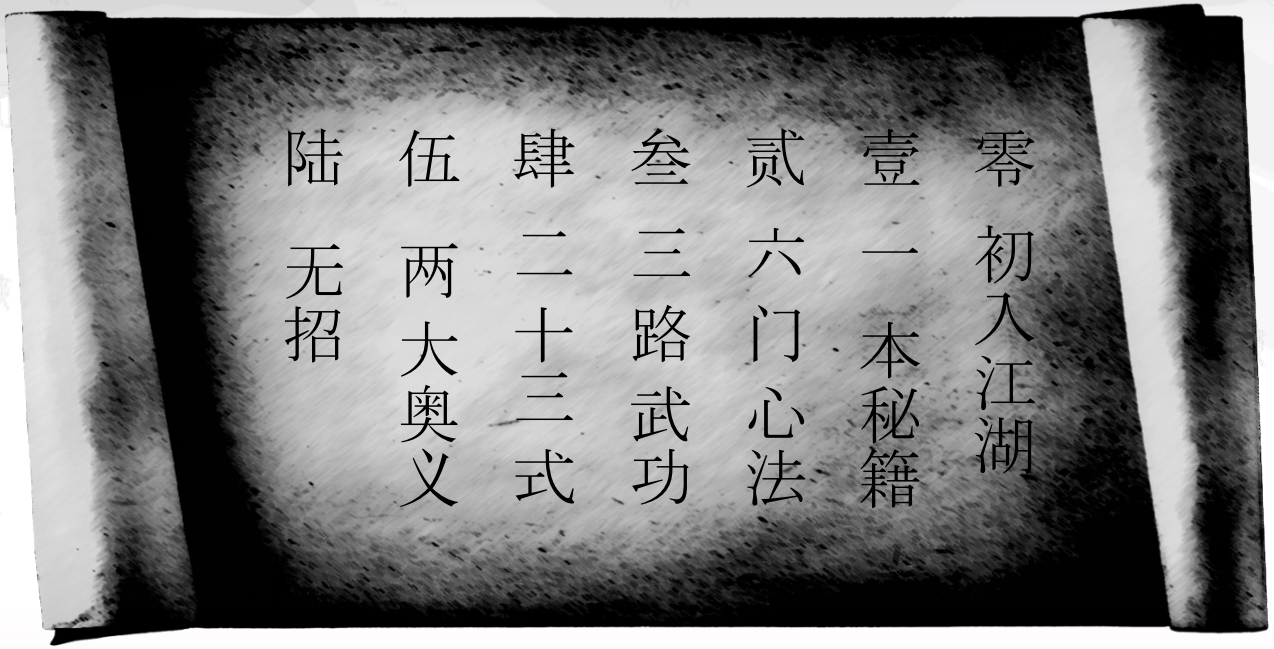
SwiftCon China 2016

www.swiftconchina.com

Swift 行走江湖指南



初入江湖

我们不想成为

Writing Code that
Nobody Else Can Read

Essential

Copying and Pasting
from Stack Overflow

Real World

Rewriting Your Front
End Every Six Weeks

生活总是充满希望

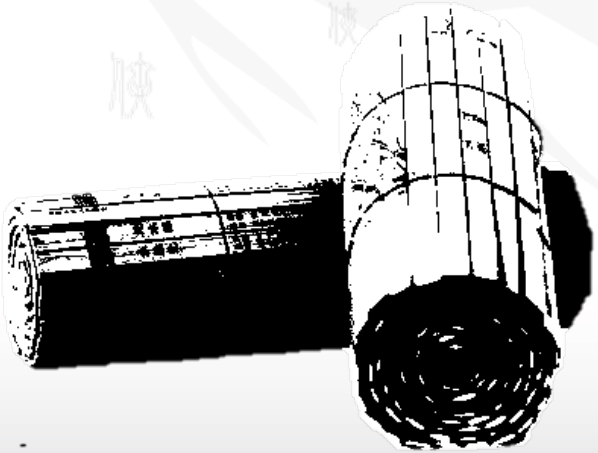
与

惊喜？

放心

不是这本

一本秘籍

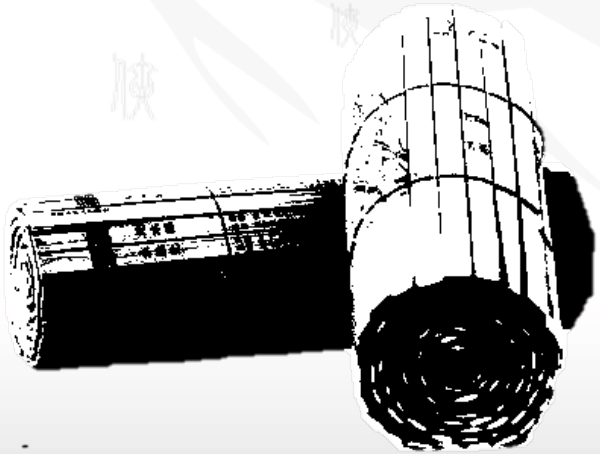


设计模式

设计模式是软件开发的哲学，它能指导你如何去设计一个优秀的架构，编写健壮的代码，解决一个复杂的需求。

有多了解就能产生多少优秀的代码和设计。

从代码工人到架构师的蜕变。



六门心法

六门心法

开闭原则

开闭原则

单一原则

里氏替换

依赖倒置

接口隔离

迪米特法则

三路武功

三路武功

创建类	结构类	行为类
工厂模式	代理模式	观察者模式
单例模式	桥梁模式	责任链模式
……	……	……

二十三式

二十三式

- 工厂模式
- 建造者
- 抽象工厂
- 单例
- 原型
- 适配器
- 桥接
- 享元
- 代理
- 责任链
- 命令
- 解释器
- 迭代器
- 中介者
- 备忘录
- 观察者
- 状态
- 策略
- 模板方法
- 访问者
- 组合模式
- 装饰器
- 外观

二十三式

单例模式

```
class 我是一个单例 {
    static let shareInstance = 我是一个单例()

    init(){

    }
}
let 单例 = 我是一个单例.shareInstance
```

```
class 还是一个单例 {
    static var shareInstance:还是一个单例?

    init(){
        if 还是一个单例.shareInstance == nil {
            还是一个单例.shareInstance = self
        }else{
            fatalError("请通过shareInstance获取该单例")
        }
    }

}
```

二十三式

工厂模式

```
protocol 大侠 {
    var 职业:String?{get set}
    var 门派:String?{get set}
    var 武器:String?{get set}
    var 绝学:String?{get set}
}


class 大侠抽象类: NSObject,大侠 {
    var 职业: String?
    var 门派: String?
    var 武器: String?
    var 绝学: String?

}
```

```
class 大侠梦工厂{
    class func 创建大侠(职业: String) -> 大侠? {
        switch 职业{
        case "贱客":
            return 贱客类()
        case "刀客":
            return 刀客类()
        default:
            return nil
        }
    }
}
```

```
class 贱客类: 大侠抽象类 {
    override init()
    {
        super.init()
        self.职业 = "贱客"
        self.门派 = "六扇门"
        self.武器 = "倚天剑"
        self.绝学 = "辟邪剑谱"
    }
}
```

```
class 刀客类: 大侠抽象类 {
    override init()
    {
        super.init()
        self.职业 = "刀客"
        self.门派 = "金刀门"
        self.武器 = "屠龙刀"
        self.绝学 = "五虎断魂刀法"
    }
}
```

二十三式

建造者模式

```
class 大侠类{
    var 内功:String?
    var 轻功:String?
    var 兵器:String?
    init(){}
}
```

```
class 大侠建造者抽象类{
    private var 大侠:大侠类 = 大侠类()
    func 学习内功(){
        fatalError("不执行")
    }
    func 学习轻功(){
        fatalError("不执行")
    }
    func 选取兵器(){
        fatalError("不执行")
    }
    init(){}
    func 获取大侠() -> 大侠类{
        return 大侠
    }
}
```

```
class 少林派大侠建造者:大侠建造者抽象类{
    override func 学习内功() {
        大侠.内功 = "易筋经"
    }
    override func 学习轻功() {
        大侠.轻功 = "一苇渡江"
    }
    override func 选取兵器() {
        大侠.兵器 = "扫把"
    }
}
```

```
class 造侠梦工厂 {
    init(){}
    func 创建侠客(造侠者:大侠建造者抽象类?){
        造侠者?.学习内功()
        造侠者?.学习轻功()
        造侠者?.选取兵器()
    }
}
```

```
let 梦工厂 = 造侠梦工厂()

let 少林派 = 少林派大侠建造者()
梦工厂.创建侠客(少林派)
let 扫地僧 = 少林派.获取大侠()
扫地僧.内功
扫地僧.轻功
扫地僧.兵器
```

二十三式

原型模式

```
class 大侠 {
    var 职业:String?
    var 门派:String?
    var 武器:String?
    var 绝学:String?
    init(职业:String?,门派:String?,武器:String?,绝学:String?){
        self.职业 = 职业
        self.门派 = 门派
        self.武器 = 武器
        self.绝学 = 绝学
    }
    func 复制() -> 大侠{
        return 大侠(职业:self.职业, 门派: self.门派, 武器: self.武器, 绝学: self.绝学)
    }
}
```

二十三式

适配器模式

二十三式

二十三式

桥梁模式

```
protocol 总指挥协议{
    var 旗部:五行旗部队协议{get set}
    func 下令攻击()

}

class 总指挥类:总指挥协议 {
    var 旗部:五行旗部队协议
    func 下令攻击(){
        旗部.执行攻击()
    }
    init(旗部:五行旗部队协议){
        self.旗部 = 旗部
    }
}
```

```
protocol 五行旗部队协议{
    func 执行攻击()
}


class 锐金旗:五行旗部队协议 {
    func 执行攻击(){
        print("射箭，投标前，扔短飞斧")
    }
}
```

```
var 总指挥 = 总指挥类(旗部: 锐金旗())
总指挥.下令攻击()
```

二十三式

二十三式

外观模式

```
class 外观鲁有脚 {
    var 左右护法:左右护法系统
    var 长老团:长老团系统
    var 分舵舵主:分舵舵主系统
    var 分舵护法:分舵护法系统
    var 底层帮众:底层帮众系统
    init(){
        左右护法 = 左右护法系统()
        长老团 = 长老团系统()
        分舵舵主 = 分舵舵主系统()
        分舵护法 = 分舵护法系统()
        底层帮众 = 底层帮众系统()

    }
```

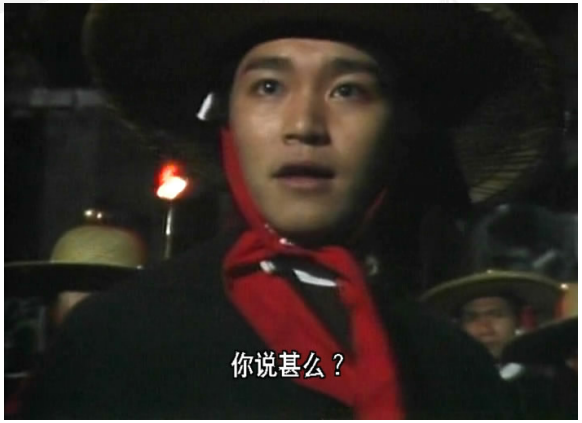
```
    func 召开丐帮大会(){
        print("都去洞庭湖君山开年会")
        左右护法.护送帮主前往洞庭湖君山()
        长老团.向各地分舵传达命令()
        分舵舵主.向手下帮众传达命令()
        分舵护法.护送分舵主前往洞庭湖君山()
        底层帮众.沿路乞讨滚去洞庭湖君山()
    }
    func 丐帮大会落幕(){
        print("年会开完，黄蓉正式接任丐帮帮主")
    }
}
```

```
class 左右护法系统{
    func 护送帮主前往洞庭湖君山(){
        print("左右护法,护送帮主前往洞庭湖君山")

    }
}

class 长老团系统{
    func 向各地分舵传达命令(){
        print("左右长老团,向各地分舵传达命令")
    }
}

class 分舵舵主系统{
    func 向手下帮众传达命令(){
        print("分舵舵主,向手下帮众传达命令")
    }
}
```

```
class 分舵护法系统{
    func 护送分舵主前往洞庭湖君山(){
        print("分舵护法,护送分舵主前往洞庭湖君山")
    }
}

class 底层帮众系统{
    func 沿路乞讨滚去洞庭湖君山(){
        print("底层帮众,沿路乞讨滚去洞庭湖君山")
    }
}
```

```
let 鲁有脚 = 外观鲁有脚()
鲁有脚.召开丐帮大会()
鲁有脚.丐帮大会落幕()
```

二十三式



二十三式

享元模式

二十三式

五岳盟主

小岳岳

二十三式

代理模式

二十三式

二十三式

责任链模式

```
protocol 杀手协议 {
    var 下一位杀手:杀手协议?{get set}
    func 链接下一位杀手(下一位杀手:杀手协议)
    func 接生意(金额:Int)
}
```

```
class 穿鞋杀手类:杀手协议{
……
class 光脚杀手类:杀手协议{
……
```

```
class 梁朝伟类:杀手协议{
    var 下一位杀手:杀手协议?

    func 链接下一位杀手(下一位杀手:杀手协议){
        self.下一位杀手 = 下一位杀手
    }
    func 接生意(金额:Int){
        if 金额 < 100 {
            下一位杀手!.接生意(金额)
        }else{
            print("梁朝伟接下这笔生意")
        }
    }
}
```

```
let 梁朝伟:梁朝伟类 = 梁朝伟类()
let 穿鞋杀手:穿鞋杀手类 = 穿鞋杀手类()
let 光脚杀手:光脚杀手类 = 光脚杀手类()

梁朝伟.链接下一位杀手(穿鞋杀手)
穿鞋杀手.链接下一位杀手(光脚杀手)
梁朝伟.接生意(200)
```

二十三式

土豪撩妹

二十三式

命令模式

```
class 厨师类 {
    func 做四干果(){
        print("厨师做了四干果")
    }
    func 做四鲜果(){
        print("厨师做了做四鲜果")
    }
    func 做两咸酸(){
        print("厨师做了做两咸酸")
    }
    func 做四蜜饯(){
        print("厨师做了做四蜜饯")
    }
}
```

```
protocol 点菜命令协议{
    var 菜名:String?{get set}
    var 厨师:厨师类?{get set}
    func 执行命令()
}
class 点菜命令抽象类:NSObject,点菜命令协议{
    var 菜名:String?
    var 厨师:厨师类?
    init(厨师: 厨师类?) {
        self.厨师 = 厨师
    }
    func 执行命令() {

    }
}
```

```
class 做四干果命令:点菜命令抽象类
{
    override func 执行命令() {
        厨师?.做四干果()
    }
}

class 做四鲜果命令:点菜命令抽象类
{
    override func 执行命令() {
        厨师?.做四鲜果()
    }
}
```

```
class 做两咸酸命令:点菜命令抽象类{
    override func 执行命令() {
        厨师?.做两咸酸()
    }
}

class 做四蜜饯命令:点菜命令抽象类{
    override func 执行命令() {
        厨师?.做四蜜饯()
    }
}
```

```
class 小二类 {
    private var 命令数组 = [点菜命令抽象类]()
    func 记录订单(点菜命令:点菜命令抽象类){
        print("下单:",点菜命令.菜名!)
        命令数组.append(点菜命令)
    }
    func 取消订单(点菜命令:点菜命令抽象类){
        print("撤单:",点菜命令.菜名!)
        命令数组.removeAtIndex(命令数组.indexOf(点菜命令)!)
    }
    func 正式下订单(){
        for 命令 in 命令数组{
            命令.执行命令()
        }
    }
}
```

```
let 厨师:厨师类? = 厨师类()
let 小二 = 小二类()
let 做四干果 = 做四干果命令(厨师: 厨师)
做四干果.菜名 = "四干果"
let 做四鲜果 = 做四鲜果命令(厨师: 厨师)
做四鲜果.菜名 = "四鲜果"
let 做两咸酸 = 做两咸酸命令(厨师: 厨师)
做两咸酸.菜名 = "两咸酸"
let 做四蜜饯 = 做四蜜饯命令(厨师: 厨师)
做四蜜饯.菜名 = "四蜜饯"

小二.记录订单(做四干果)
小二.记录订单(做四鲜果)
小二.记录订单(做两咸酸)
小二.记录订单(做四蜜饯)
小二.取消订单(做四鲜果)
小二.正式下订单()
```

二十三式

二十三式

解释器模式

二十三式

迭代器模式

二十三式

五 岳 剑 派

二十三式

中介者模式

```
protocol 五岳剑派协议 {
    func 发消息(中介:五岳盟主协议,消息:String)
    func 接收消息(消息:String)
}

class 五岳剑派类:五岳剑派协议{
    var 名字:String
    init(名字:String){
        self.名字 = 名字
    }
    func 发消息(中介:五岳盟主协议,消息:String){
        中介.广播消息(self, 消息: 消息)
    }
    func 接收消息(消息:String){
        print("\(self.名字) 接收到消息：\(消息)")
    }
}
```

```
protocol 五岳盟主协议 {
    var 五岳剑派数组:[五岳剑派协议]?{get}
    func 广播消息(发送者:五岳剑派协议,消息:String)
    func 加入联盟(剑派:五岳剑派协议)
}

class 五岳盟主类:五岳盟主协议{
    var 五岳剑派数组:[五岳剑派协议]? = [五岳剑派类]()
    func 广播消息(发送者:五岳剑派协议,消息:String){
        for 剑派 in 五岳剑派数组! {
            if 剑派 as! 五岳剑派类 !== 发送者 as! 五岳剑派类 {
                剑派.接收消息(消息)
            }
        }
    }
    func 加入联盟(剑派:五岳剑派协议){
        五岳剑派数组?.append(剑派)
    }
}
```

```
let 泰山派 = 五岳剑派类(名字: "泰山派")
let 衡山派 = 五岳剑派类(名字: "衡山派")
let 华山派 = 五岳剑派类(名字: "华山派")
let 恒山派 = 五岳剑派类(名字: "恒山派")
let 嵩山派 = 五岳剑派类(名字: "嵩山派")

let 五岳盟主 = 五岳盟主类()
五岳盟主.加入联盟(泰山派)
五岳盟主.加入联盟(衡山派)
五岳盟主.加入联盟(华山派)
五岳盟主.加入联盟(恒山派)
五岳盟主.加入联盟(嵩山派)

嵩山派.发消息(五岳盟主, 消息: "好久没一起完了，来嵩山再选一次五岳盟主吧")
```

二十三式


9.30 一起哈哈哈
夏洛
特烦恼
开心麻花
闫非 彭大魔
Goodbye Mr. Loser


我自己平常也创作
I'm writing too.

二十三式

备忘录模式

二十三式


www.impdb.com
infernal affairs
無間道
INFERNAL AFFAIRS
陳永仁（梁朝偉）

二十三式

观察者模式

二十三式

二十三式

状态模式

```
class 东方不败类{
    private var 性别:性别协议!
    init(性别:性别协议){
        self.性别 = 性别
    }
    func 说话(){
        性别.说话(self)
    }

}
```

```
protocol 性别协议 {
    func 说话(东方不败:东方不败类)
}
class 性别男:性别协议{
    func 说话(东方不败:东方不败类){
        print("可男，气宇轩昂")
        东方不败.性别 = 性别女()

    }
}
class 性别女:性别协议 {
    func 说话(东方不败:东方不败类){
        print("可女，倾国倾城")
        东方不败.性别 = 性别男()
    }
}
```

```
let 性别 = 性别男()
let 东方不败 = 东方不败类(性别: 性别)
//每说一次话就转换一次性别
东方不败.说话()
东方不败.说话()
东方不败.说话()
东方不败.说话()
```

二十三式

模板方法模式

二十三式

访问者模式

两大奥义

两大奥义

MVC

两大奥义

手中无招
心中有招

无招

手中无招
心中也也无招

谢谢观赏