面向协议编程

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

面向协议编程

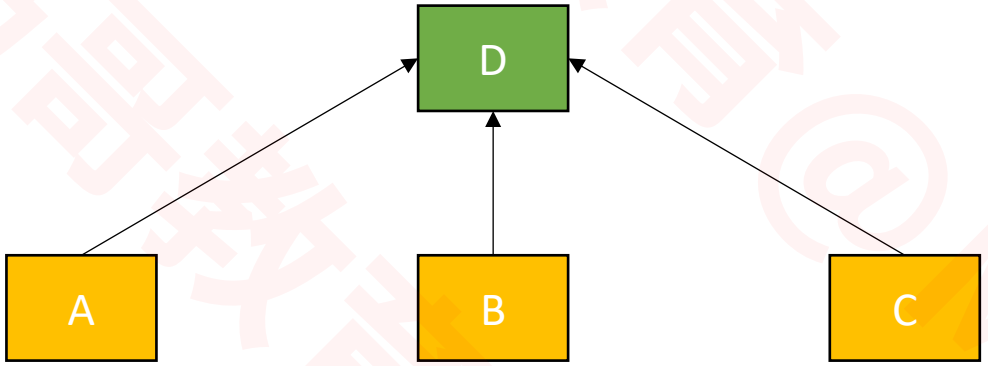
- 面向协议编程（Protocol Oriented Programming，简称POP）
  - 是Swift的一种编程范式， Apple于2015年WWDC提出
  - 在Swift的标准库中，能见到大量POP的影子

- 同时，Swift也是一门面向对象的编程语言（Object Oriented Programming，简称OOP）
  - 在Swift开发中，OOP和POP是相辅相成的，任何一方并不能取代另一方

- POP能弥补OOP一些设计上的不足

回顾OOP

- OOP的三大特性：封装、继承、多态
- 继承的经典使用场合
  - 当多个类（比如A、B、C类）具有很多共性时，可以将这些共性抽取到一个父类中（比如D类），最后A、B、C类继承D类



OOP的不足

- 但有些问题，使用OOP并不能很好解决，比如
  - 如何将 BVC、DVC 的公共方法 run 抽取出来？

```
class BVC: UIViewController {
    func run() {
        print("run")
    }
}
```

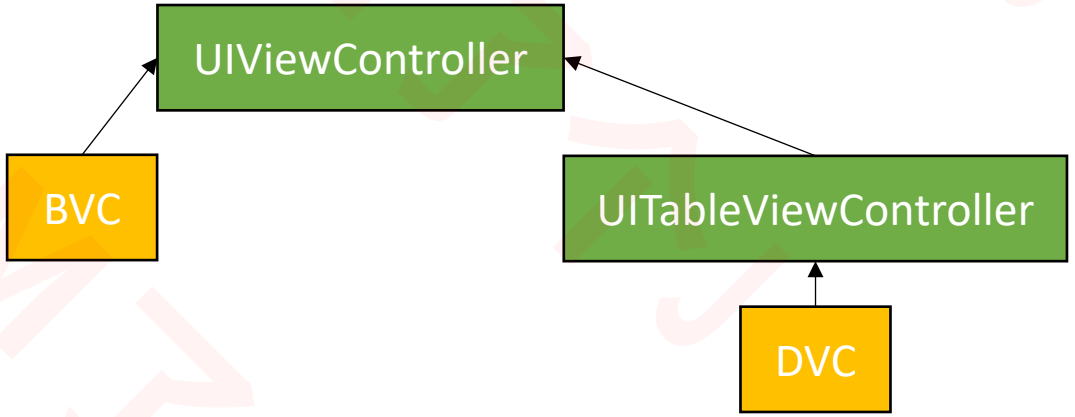
```
class DVC: UITableViewController {
    func run() {
        print("run")
    }
}
```

- 基于OOP想到的一些解决方案？

1. 将run方法放到另一个对象A中，然后BVC、DVC拥有对象A属性
   - 多了一些额外的依赖关系

2. 将run方法增加到UIViewController分类中
   - UIViewController会越来越臃肿，而且会影响它的其他所有子类

3. 将run方法抽取到新的父类，采用多继承？（C++支持多继承）
   - 会增加程序设计复杂度，产生菱形继承等问题，需要开发者额外解决



小码哥教育 SEEMYGO

POP的解决方案

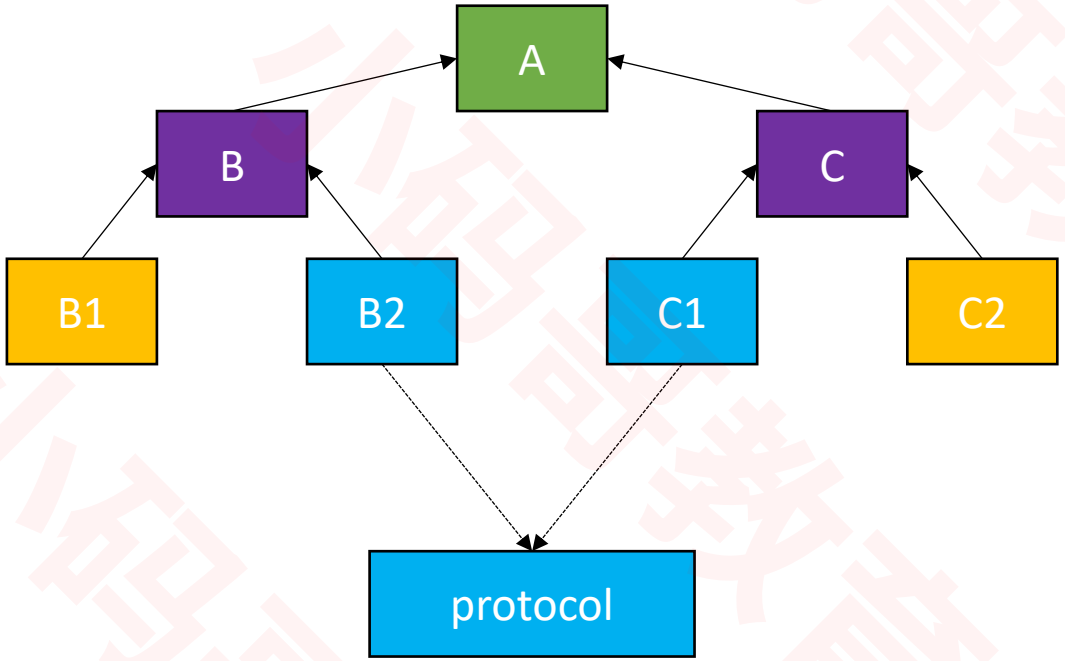
```
protocol Runnable {
    func run()
}

extension Runnable {
    func run() {
        print("run")
    }
}
```

```
class BVC: UIViewController, Runnable {}
class DVC: UITableViewController, Runnable {}
```

小码哥教育 SEEMYGO

再举例



POP的注意点

- 优先考虑创建协议，而不是父类（基类）
- 优先考虑值类型（struct、enum），而不是引用类型（class）
- 巧用协议的扩展功能
- 不要为了面向协议而使用协议

利用协议实现前缀效果

```swift
var string = "123fdsf434"
print(string.mj.numberCount())
```

```swift
struct MJ<Base> {
    let base: Base
    init(_ base: Base) {
        self.base = base
    }
}

protocol MJCompatible {}
extension MJCompatible {
    static var mj: MJ<Self>.Type {
        get { MJ<Self>.self }
        set {}
    }
    var mj: MJ<Self> {
        get { MJ(self) }
        set {}
    }
}
```

```swift
extension String: MJCompatible {}
extension MJ where Base == String {
    func numberCount() -> Int {
        var count = 0
        for c in base where ("0"..."9").contains(c) {
            count += 1
        }
        return count
    }
}
```

Base: 类

```
class Person {}
class Student: Person {}

extension Person: MJCompatible {}
extension MJ where Base: Person {
    func run() {}
    static func test() {}
}
```

```
Person.mj.test()
Student.mj.test()

let p = Person()
p.mj.run()

let s = Student()
s.mj.run()
```

Base: 协议

```swift
var s1: String = "123fdsf434"
var s2: NSString = "123fdsf434"
var s3: NSMutableString = "123fdsf434"
print(s1.mj.numberCount())
print(s2.mj.numberCount())
print(s3.mj.numberCount())
```

```swift
extension String: MJCompatible {}
extension NSString: MJCompatible {}
extension MJ where Base: ExpressibleByStringLiteral {
    func numberCount() -> Int {
        let string = base as! String
        var count = 0
        for c in string where ("0"..."9").contains(c) {
            count += 1
        }
        return count
    }
}
```

小码哥教育 SEEMYGO

利用协议实现类型判断

```
func isArray(_ value: Any) -> Bool { value is [Any] }
isArray( [1, 2] )
isArray( ["1", 2] )
isArray( NSArray() )
isArray( NSMutableArray() )
```

```
protocol ArrayType {}
extension Array: ArrayType {}
extension NSArray: ArrayType {}
func isArrayType(_ type: Any.Type) -> Bool { type is ArrayType.Type }
isArrayType([Int].self)
isArrayType([Any].self)
isArrayType(NSArray.self)
isArrayType(NSMutableArray.self)
```