面向协议编程与 Cocoa 的邂逅

MDCC 16

王巍 - 2016 年 9 月 24 日

主题

- 起·初识
- 承·相知
- 转·热恋
- 合·陪伴

主题

- 起・初识 - 什么是 Swift 协议
- 承・相知
- 转・热恋
- 合・陪伴

主题

- 起・初识 - 什么是 Swift 协议
- 承・相知 - 协议扩展和面向协议编程
- 转・热恋
- 合・陪伴

主题

- 起・初识 - 什么是 Swift 协议
- 承・相知 - 协议扩展和面向协议编程
- 转・热恋 - 在日常开发中使用协议
- 合・陪伴

主题

- 起・初识 - 什么是 Swift 协议
- 承・相知 - 协议扩展和面向协议编程
- 转・热恋 - 在日常开发中使用协议
  - Model (Networking)
  - ViewController
- 合・陪伴

起・初识

什么是 Swift 协议

Protocol

```
protocol Greetable {
    var name: String { get }
    func greet()
}
```

面向对象

Object-oriented

面向对象

```
class Animal {
    var leg: Int { return 2 }
    func eat() {
        print("eat food.")
    }
    func run() {
        print("run with \(leg) legs")
    }
}

class Tiger: Animal {
    override var leg: Int { return 4 }
    override func eat() {
        print("eat meat.")
    }
}

let tiger = Tiger()
tiger.eat() // "eat meat"
tiger.run() // "run with 4 legs"
```

ViewController → UIViewController

```
class ViewCotroller:
                    UIViewController
{
    // 继承
    // view, isFirstResponder()...

    // 新加
    func myMethod() {

    }
}
```

AnotherViewController → UITableViewController → UIViewController

```
class AnotherViewController:
                UITableViewController
{
    // 继承
    // tableView, isFirstResponder()...

    // 新加
    func myMethod() {

    }
}
```

困境之一

Cross-Cutting Concerns

横切关注点

解决方案

- Copy & Paste
- 引入 BaseViewController
- 依赖注入
- 多继承

Objective-C

Message Sending

```
ViewController *v1 = ...
[v1 myMethod];

AnotherViewController *v2 = ...
[v2 myMethod];
```

```objc
NSArray *array = @[v1, v2];
for (id obj in array) {
    [obj myMethod];
}
```

```objc
NSObject *v3 = [NSObject new]
// v3 没有实现 `myMethod`

NSArray *array = @[v1, v2, v3];
for (id obj in array) {
    [obj myMethod];
}
```

```objc
// Runtime error:
// unrecognized selector sent to instance blabla
```

困境之二

Dynamic Dispatch Safety

动态派发安全性

OOP 困境

- 动态派发安全性
- 横切关注点

协议

Protocol

Java, C#

Interface

Swift

protocol

Swift protocol

```
protocol Greetable {
    var name: String { get }
    func greet()
}
```

Swift protocol

```
protocol Greetable {
    var name: String { get }
    func greet()
}

struct Person: Greetable {
    let name: String
    func greet() {
        print("你好 \(name)")
    }
}
Person(name: "Wei Wang").greet()
```

承・相知

协议扩展和面向协议编程

OOP 困境

- 动态派发安全性
- 横切关注点

```swift
protocol Greetable {
    var name: String { get }
    func greet()
}

struct Person: Greetable {
    let name: String
    func greet() {
        print("你好 \(name)")
    }
}
```

```swift
protocol Greetable {
    var name: String { get }
    func greet()
}

struct Cat: Greetable {
    let name: String
    func greet() {
        print("meow~ \(name)")
    }
}
```

```swift
let array: [Greetable] = [
        Person(name: "Wei Wang"),
        Cat(name: "onevcat")]
for obj in array {
    obj.greet()
}
// 你好 Wei Wang
// meow~ onevcat
```

```
struct Bug: Greetable {
    let name: String
}

// Compiler Error:
// 'Bug' does not conform to protocol 'Greetable'
// protocol requires function 'greet()'
```

OOP 困境

- ✅ 动态派发安全性
- 横切关注点

使用协议共享代码

```
protocol P {
    func myMethod()
}
```

```
// class ViewController: UIViewController
extension ViewController: P {
    func myMethod() {
        doWork()
    }
}

// class AnotherViewController: UITableViewController
extension AnotherViewController: P {
    func myMethod() {
        doWork()
    }
}
```

Swift 2 - 协议扩展

```
protocol P {
    func myMethod()
}

extension P {
    func myMethod() {
        doWork()
    }
}
```

为 P 定义的方法提供默认实现

```
extension ViewController: P { }
extension AnotherViewController: P { }

viewController.myMethod()
anotherViewController.myMethod()
```

实现未在协议中声明的内容

```
protocol P {
    func myMethod()
}

extension P {
    func myMethod() {
        doWork()
    }

    func anotherMethod() {
        myMethod()
        someOtherWork()
    }
}

viewController.anotherMethod()
```

- 协议定义
  - 提供实现的入口
  - 遵循协议的类型需要对其进行实现
- 协议扩展
  - 为入口提供默认实现
  - 根据入口提供额外实现

OOP 困境

- ✅ 动态派发安全性
- ✅ 横切关注点

转・热恋

在日常开发中使用协议

WWDC 15 #408

Protocol-Oriented Programming in Swift

Model (Networking)

案例

基于 Protocol 的网络请求

Demo

- 基于协议
- 类型安全
- 解耦合
- 可单独测试
- 扩展性

- Networking:
  - AFNetworking
  - Alamofire
  - (ASIHTTPRequest) 😇
- Model Parser
  - SwiftyJSON
  - Argo
  - Himotoki

优先考虑使用协议

高度协议化有助于解耦，测试以及扩展

APIKit[1] + Himotoki[2]

---

[1] https://github.com/ishkawa/APIKit

[2] https://github.com/ikesyo/Himotoki

Controller

案例

分页加载

分页加载的网络请求

```
struct Pagination<T> {
    let items: [T]
    let hasNext: Bool
}
```

```
struct ChannelsResquest: Request {
    typealias Response = Pagination<Channel>
    let lastId: Int?
}
```

```
class ChannelsTableViewController: UITableViewController {
    private var lastId: Int? = nil
    private var hasNext = true

    override func viewDidLoad() {
        super.viewDidLoad()
        load()
    }

    func load() {
        if hasNext {
            client.send(ChannelsResquest(lastId: lastId)) {
                result in
            }
        }
    }
}
```

```swift
class ChannelsTableViewController: UITableViewController {
    private var lastId: Int? = nil
    private var hasNext = true

    private var data: [Channel] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        load()
    }

    func load() {
        if hasNext {
            client.send(ChannelsResquest(lastId: lastId)) {
                result in
                self.lastId = result!.items.last?.id
                self.hasNext = result!.hasNext
                self.data = result.items
                self.tableView.reloadData()
            }
        }
    }
}
```

```
extension ChannelsTableViewController: UITableViewDelegate {
    override func tableView(tableView: UITableView,
                   willDisplayCell cell: UITableViewCell,
             forRowAtIndexPath indexPath: NSIndexPath)
{
    if indexPath.row == data.count - 1 {
        load()
    }
}
```

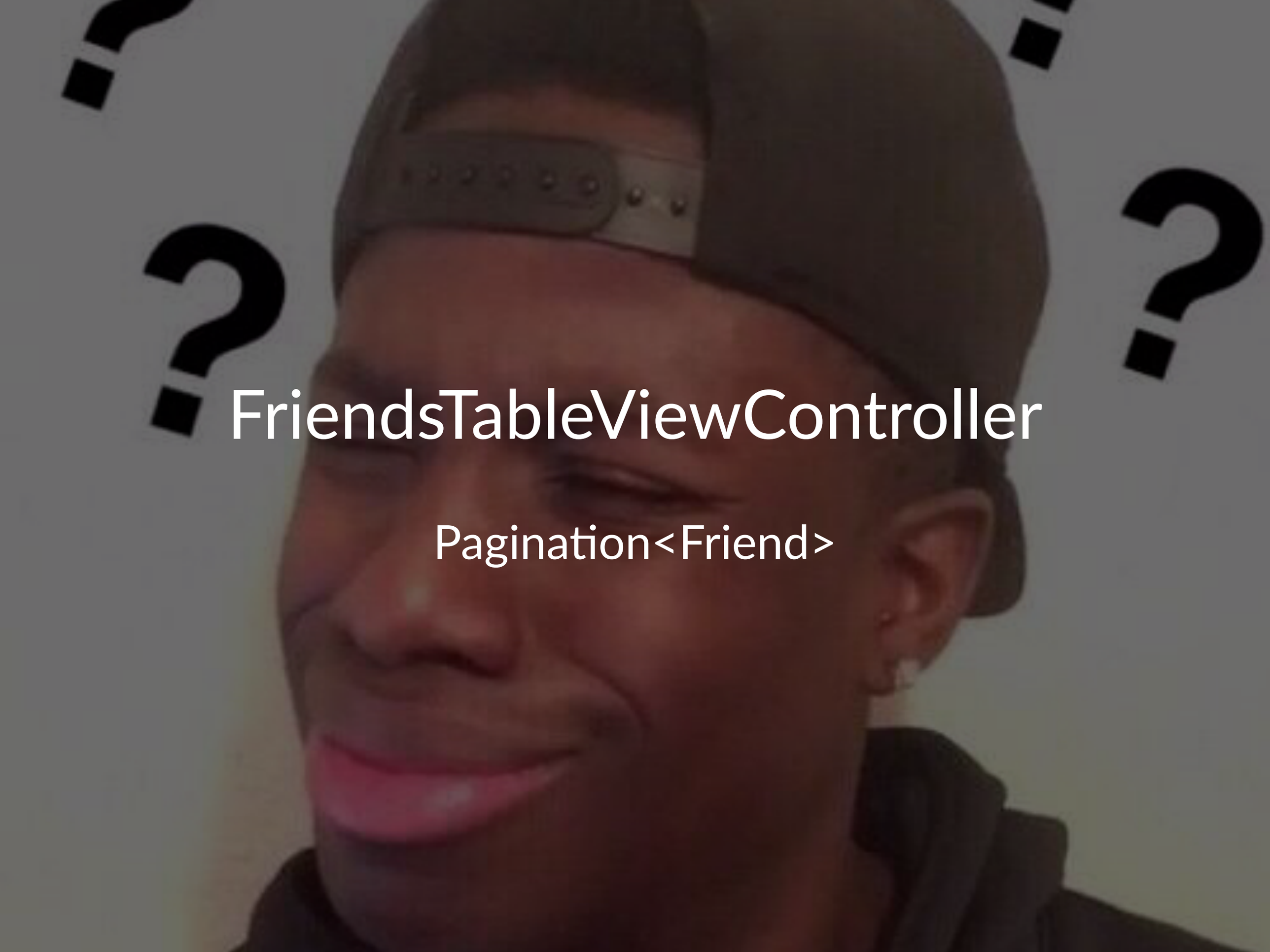
故事还没有结束...

故事还没有结束...

```
private var isLoading = false

func load() {
      if isLoading { return }
      if hasNext {
          isLoading = true
          client.send(ChannelsResquest(lastId: lastId)) {
              result in
              //...
              self.isLoading = false
          }
      }
  }
```

ChannelsTableViewController

Pagination<Channel>


FriendsTableViewController
Pagination<Friend>

方案一：复制粘贴

珍藏本

汉译世界学术名著丛书

自杀论

〔法〕埃米尔·迪尔凯姆 著

方案一：复制粘贴

商務印書館
The Commercial Press

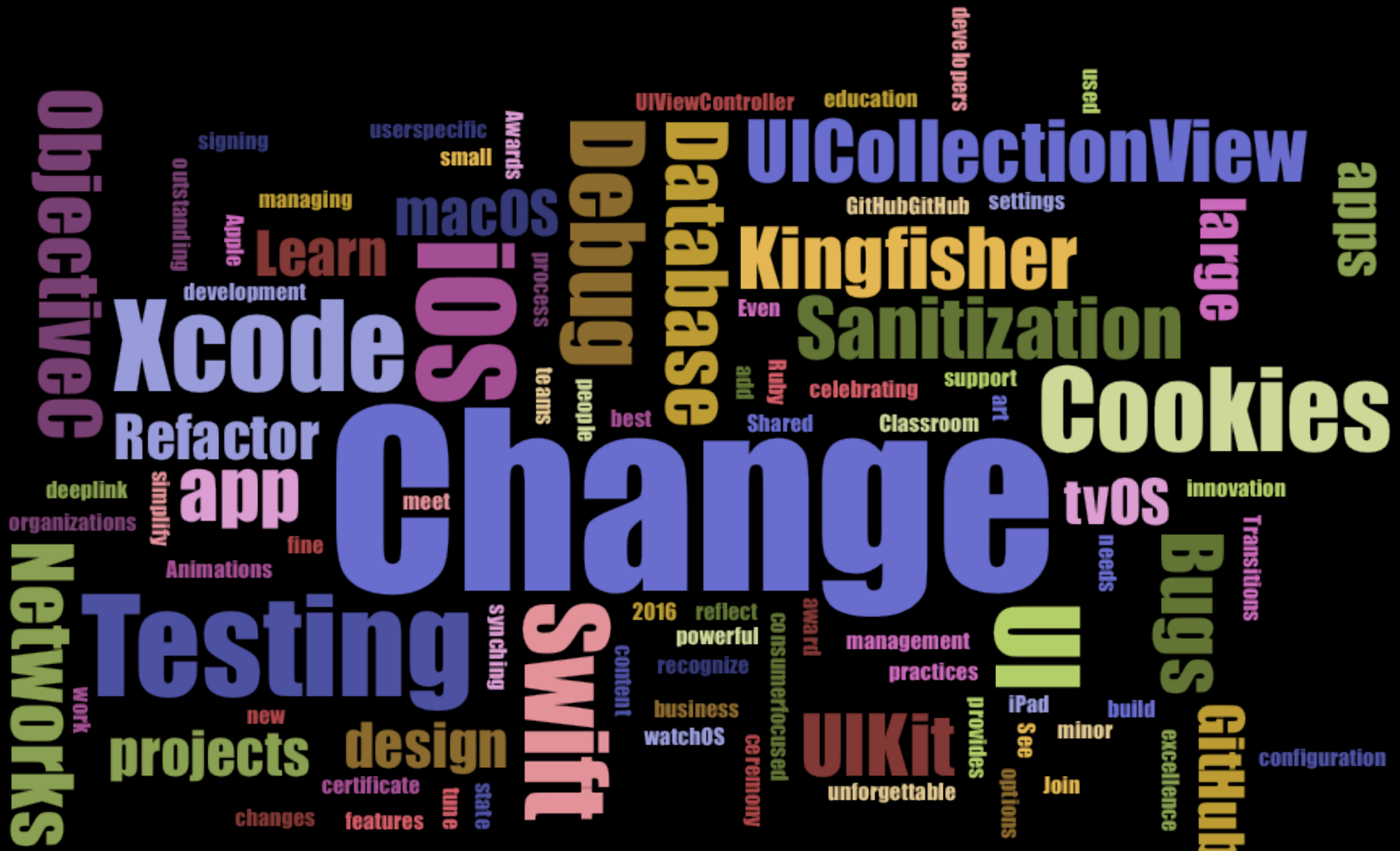
方案二：父类

BaseTableViewController

```
class BaseTableViewController: UITableViewController {
    var lastId: Int? = nil
    var hasNext = true
    var isLoading = false

    func loadNext() {
        if isLoading { return }
        if hasNext {
            isLoading = true
            doLoad {result in
                self.lastId = //...
                self.hasNext = //...
            }
        }
    }

    func doLoad(handler: (Any?)->Void) {
        // ??
    }
}
```

```
class BaseTableViewController: UITableViewController {
    var lastId: Int? = nil
    var hasNext = true
    var isLoading = false

    func loadNext() {
        if isLoading { return }
        if hasNext {
            isLoading = true
            doLoad {result in
                self.lastId = //...
                self.hasNext = //...
            }
        }
    }

    func doLoad(handler: (Any?)->Void) {
        fatalError("You should implement it in subclass!")
    }
}
```

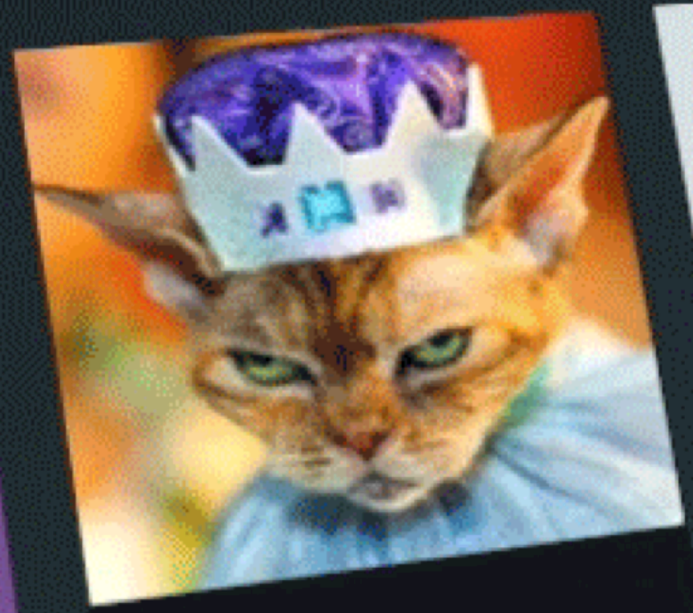
```
class FriendsTableViewController: BaseTableViewController {
    private var data: [Friend] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        loadNext()
    }

    override func doLoad(handler: (Any?)->Void) {
        client.send(FriendsRequest(lastId: lastId)) {
            result in
            handler(result)
            // ...
            self.data = result!.items
            self.tableView.reloadData()
        }
    }
}
```


Change
UICollectionView
Kingfisher
Sanitization
Cookies
Database
Debug
Objectivec
Xcode
iOS
Learn
macOS
Refactor
app
Testing
Networks
Swift
projects
design
UIKit
UI
Bugs
GitHub
tvOS
Cookies
UIViewController
education
developers
used
userspecific
signing
Awards
small
managing
Apple
outstanding
development
process
Even
large
apps
Ruby
add
celebrating
support
art
people
best
Shared
Classroom
teams
deeplink
simplify
organizations
meet
fine
Animations
innovation
Transitions
needs
2016
reflect
powerful
recognize
award
consumerfocused
content
business
watchOS
ceremony
management
practices
provides
options
iPad
See
minor
build
excellence
configuration
Join
unforgettable
syncing
new
work
certificate
changes
features
tune
state

Yalantis 9:41 AM 100%

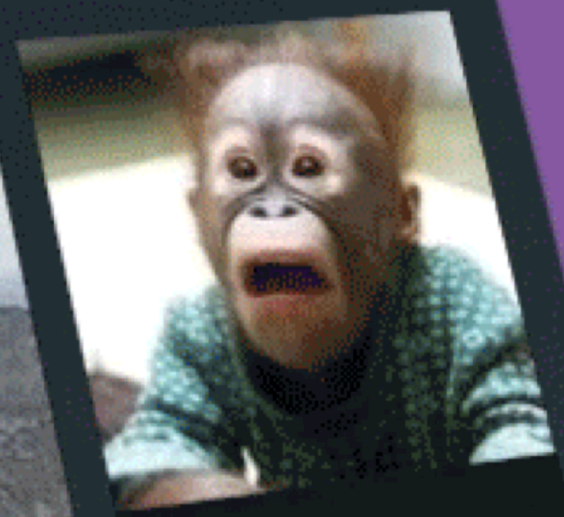
Friends

Search

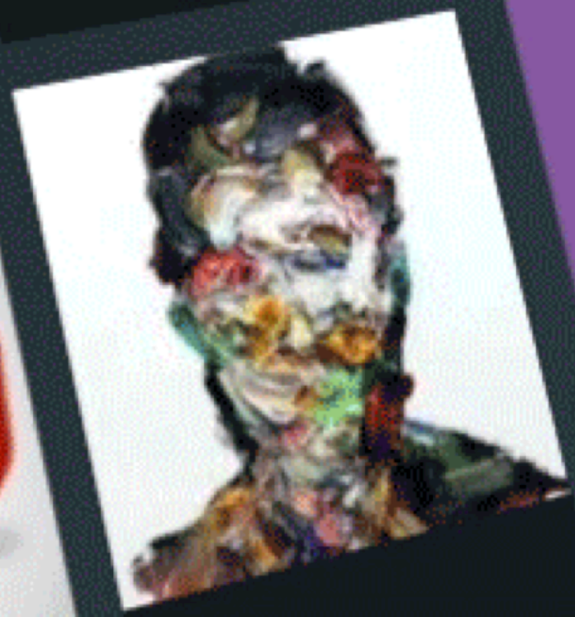


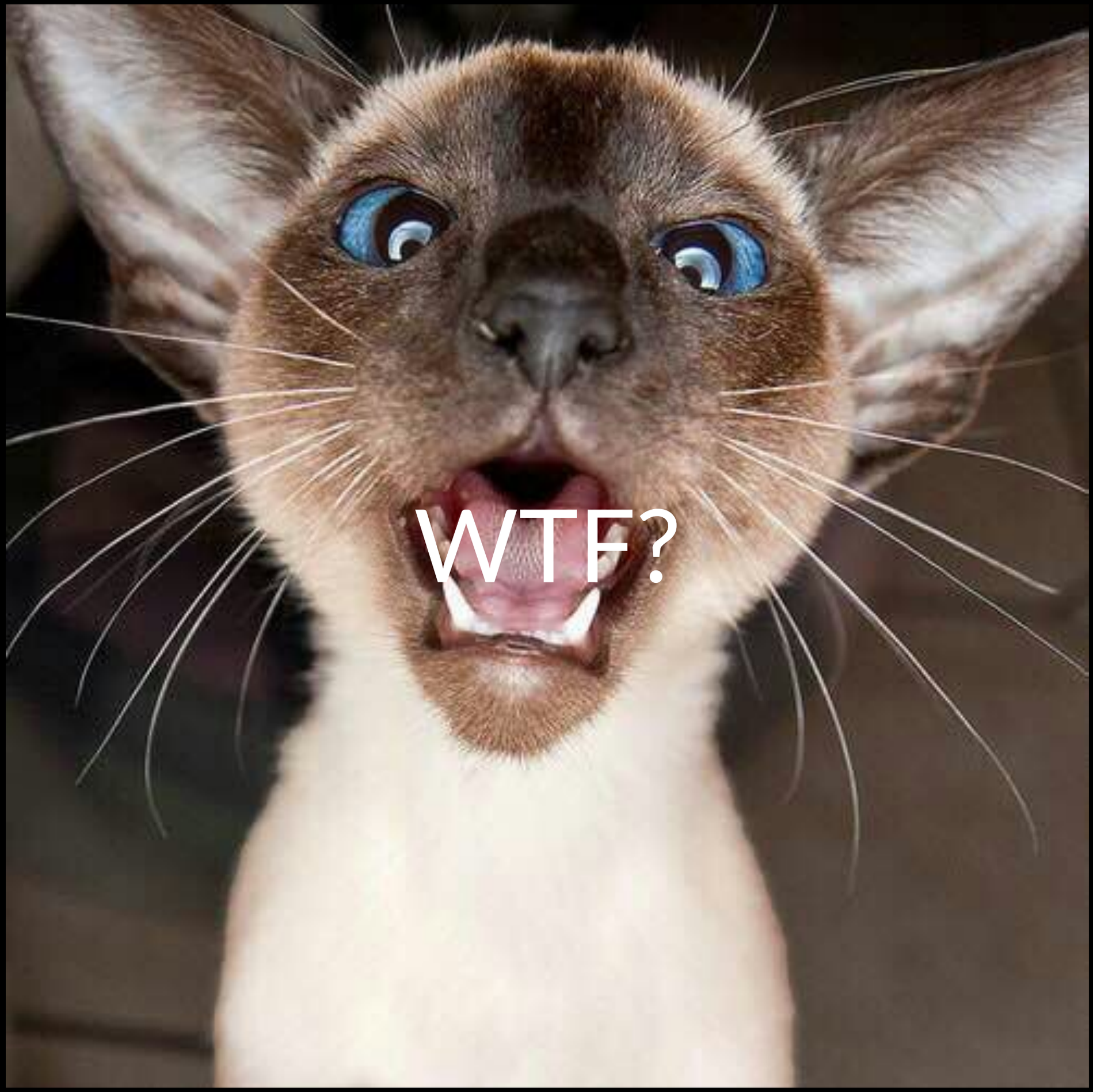
Grace Simpson

George Barrett



Joan Gonzales




WTF?

- FriendsTableViewController → FriendsCollectionViewController

- FriendsTableViewController → FriendsCollectionViewController
- BaseTableViewController → BaseCollectionViewController

- FriendsTableViewController → FriendsCollectionViewController
- BaseTableViewController → BaseCollectionViewController

复制粘贴?

- FriendsTableViewController → FriendsCollectionViewController
- BaseTableViewController → BaseCollectionViewController

方案三：协议

```
struct NextPageState<T> {
    private(set) var hasNext: Bool
    private(set) var isLoading: Bool
    private(set) var lastId: T?

    init() {
        hasNext = true
        isLoading = false
        lastId = nil
    }

    mutating func reset() {
        hasNext = true
        isLoading = false
        lastId = nil
    }

    mutating func update(hasNext: Bool, isLoading: Bool, lastId: T?) {
        self.hasNext = hasNext
        self.isLoading = isLoading
        self.lastId = lastId
    }
}
```

```
protocol NextPageLoadable: class {
    associatedtype DataType
    associatedtype LastIdType

    var data: [DataType] { get set }
    var nextPageState: NextPageState<LastIdType> { get set }
    func performLoad(
        successHandler: (_ rows: [DataType],
                         _ hasNext: Bool,
                         _ lastId: LastIdType?) -> (),
        failHandler: () -> ()
    )
}
```

```
protocol NextPageLoadable: class {
    associatedtype DataType
    associatedtype LastIdType

    var data: [DataType] { get set }
    var nextPageState: NextPageState<LastIdType> { get set }
    func performLoad(
        successHandler: (_ rows: [DataType],
                         _ hasNext: Bool,
                         _ lastId: LastIdType?) -> (),
        failHandler: () -> ()
    )
}

extension NextPageLoadable where Self: UITableViewController {
    func loadNext() {
        guard nextPageState.hasNext else { return }
        if nextPageState.isLoading { return }
```

```
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() {
        guard nextPageState.hasNext else { return }
        if nextPageState.isLoading { return }

        nextPageState.isLoading = true
        performLoad(successHandler: { rows, hasNext, lastId in
            self.data += rows
            self.nextPageState.update(hasNext: hasNext,
                                      isLoading: false,
                                      lastId: lastId)
```

```
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() {
        guard nextPageState.hasNext else { return }
        if nextPageState.isLoading { return }

        nextPageState.isLoading = true
        performLoad(successHandler: { rows, hasNext, lastId in
            self.data += rows
            self.nextPageState.update(hasNext: hasNext,
                                      isLoading: false,
                                      lastId: lastId)

            self.tableView.reloadData()

        }, failHandler: {
            //..
        })
    }
}
```

```
class FriendTableViewController: UITableViewController {
    var nextPageState = NextPageState<Int>()
    var data: [Friend] = []
}

extension FriendTableViewController: NextPageLoadable {
    func performLoad(
                successHandler: ([String], Bool, Int?) -> (),
                failHandler: () -> ())
{
        client.send(FriendsRequest()) {
                result in
                if let result = result {
                    successHandler(result.items,
                                   result.hasNext,
                                   result.items.last.id)
                } else {
                    failHandler()
                }
        }
    }
}
```

```
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() { ... }
}
```

```
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() { ... }
}

extension FriendTableViewController: NextPageLoadable { ... }
class FriendTableViewController: UITableViewController {
    //...
    override func viewDidLoad() {
        super.viewDidLoad()
        loadNext()
    }

    override func tableView(tableView: UITableView,
                  willDisplayCell cell: UITableViewCell,
            forRowAtIndexPath indexPath: NSIndexPath)
    {
        if indexPath.row == data.count - 1 {
            loadNext()
        }
    }
}
```

```
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() { ... }
}
```

UICollectionView 怎么办?

```
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() { ... }
}
```

UICollectionView 怎么办?

复制粘贴?

```
extension NextPageLoadable where Self: UICollectionViewController {
    func loadNext() { ... }
}
```

```swift
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() {
        guard nextPageState.hasNext else { return }
        if nextPageState.isLoading { return }

        nextPageState.isLoading = true
        performLoad(successHandler: { rows, hasNext, lastId in
            self.data += rows
            self.nextPageState.update(hasNext: hasNext, isLoading: false, lastId: lastId)

            self.tableView.reloadData()

            }, failHandler: {
                // Failed when first loading
                if self.nextPageState.lastId == nil {
                    self.data = []
                    self.nextPageState.reset()
                }
        })
    }
}
```

```
tableView.reloadData()
colletionView.reloadData()
```

```
tableView.reloadData()
colletionView.reloadData()

protocol ReloadableType {
    func reloadData()
}

extension UITableView: ReloadableType {}
extension UICollectionView: ReloadableType {}
```

```
extension NextPageLoadable
    where Self: UITableViewController {
    func loadNext() {

        //...

        self.tableView.reloadData()

        //...
    }
}
```

```
extension NextPageLoadable {

    func loadNext(view: ReloadableType) {

        //...

        view.reloadData()

        //...
    }
}
```

```swift
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() {
        loadNext(reloadView: tableView)
    }
}
```

```swift
extension NextPageLoadable where Self: UITableViewController {
    func loadNext() {
        loadNext(reloadView: tableView)
    }
}

extension NextPageLoadable where Self: UICollectionViewController {
    func loadNext() {
        loadNext(reloadView: collectionView)
    }
}
```

- FriendsTableViewController → FriendsCollectionViewController

```
class FriendTableViewController: UITableViewController {
    var nextPageState = NextPageState<Int>()
    var data: [Friend] = []
}

extension FriendTableViewController: NextPageLoadable {
    func performLoad(
                successHandler: ([String], Bool, Int?) -> (),
                failHandler: () -> ())
{
        client.send(FriendsRequest()) {
                result in
                if let result = result {
                    successHandler(result.items,
                                   result.hasNext,
                                   result.items.last.id)
                } else {
                    failHandler()
                }
        }
    }
}
```

```
class FriendCollectionViewController: UITableViewController {
    var nextPageState = NextPageState<Int>()
    var data: [Friend] = []
}

extension FriendCollectionViewController: NextPageLoadable {
    func performLoad(
                successHandler: ([String], Bool, Int?) -> (),
                failHandler: () -> ())
{
        client.send(FriendsRequest()) {
                result in
                if let result = result {
                    successHandler(result.items,
                                   result.hasNext,
                                   result.items.last.id)
                } else {
                    failHandler()
                }
        }
    }
}
```

ViewController 测试

使用协议来组织 ViewController

- 保持简单的 ViewController 继承 (减少继承)

使用协议来组织 ViewController

- 保持简单的 ViewController 继承 (减少继承)
- 使用协议来「拼装」ViewController 所需要的功能

使用协议来组织 ViewController

- 保持简单的 ViewController 继承 (减少继承)
- 使用协议来「拼装」ViewController 所需要的功能
- 将代码和责任分离出 ViewController

使用协议来组织 ViewController

- 保持简单的 ViewController 继承 (减少继承)
- 使用协议来「拼装」ViewController 所需要的功能
- 将代码和责任分离出 ViewController
- 使用重构的方法将协议逐渐通用化

合・陪伴

使用协议帮助改善代码设计

推荐资料

- Protocol-Oriented Programming in Swift - WWDC 15 #408
- Protocols with Associated Types - @alexisgallagher
- Protocol Oriented Programming in the Real World - @_matthewpalmer
- Practical Protocol-Oriented-Programming - @natashatherobot

谢谢聆听

FAQ

Email: onev@onevcat.com, GitHub: onevcat