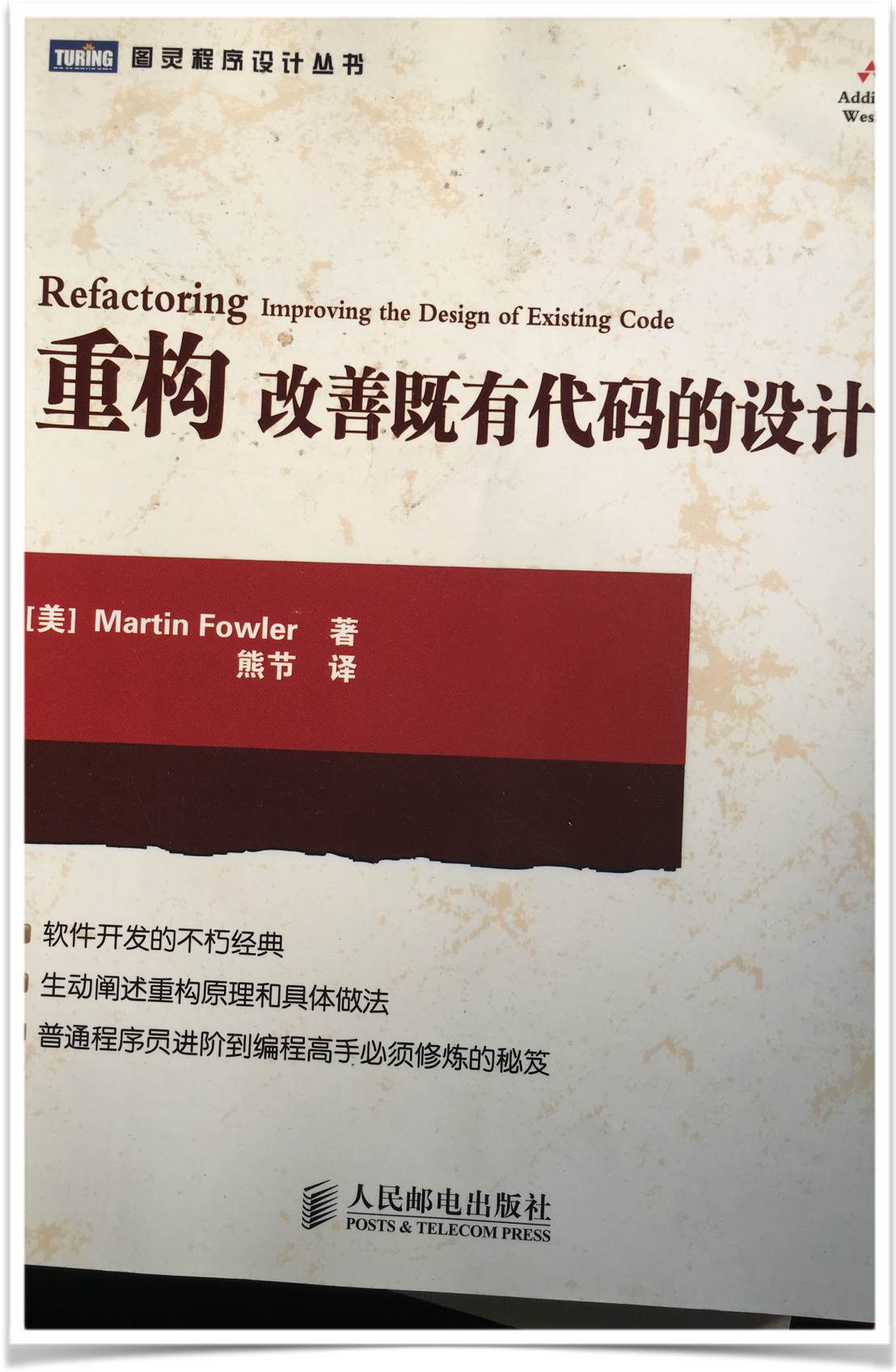
Swift 改善既有代码的设计

翁阳（酒仙）

有时候相似

不是巧合


TURING 图灵程序设计丛书
Addi
Wes
Refactoring Improving the Design of Existing Code
重构 改善既有代码的设计
[美] Martin Fowler 著
熊节 译
软件开发的不朽经典
生动阐述重构原理和具体做法
普通程序员进阶到编程高手必须修炼的秘笈
人民邮电出版社
POSTS & TELECOM PRESS

重构 与 Swift 有啥关系

你必须面对**既有代码！**

无论是不是由你编写

这些**既有代码**你无法逃避

---

① 遗留的老项目

② Cocoa Touch

③ Cocoa Pods

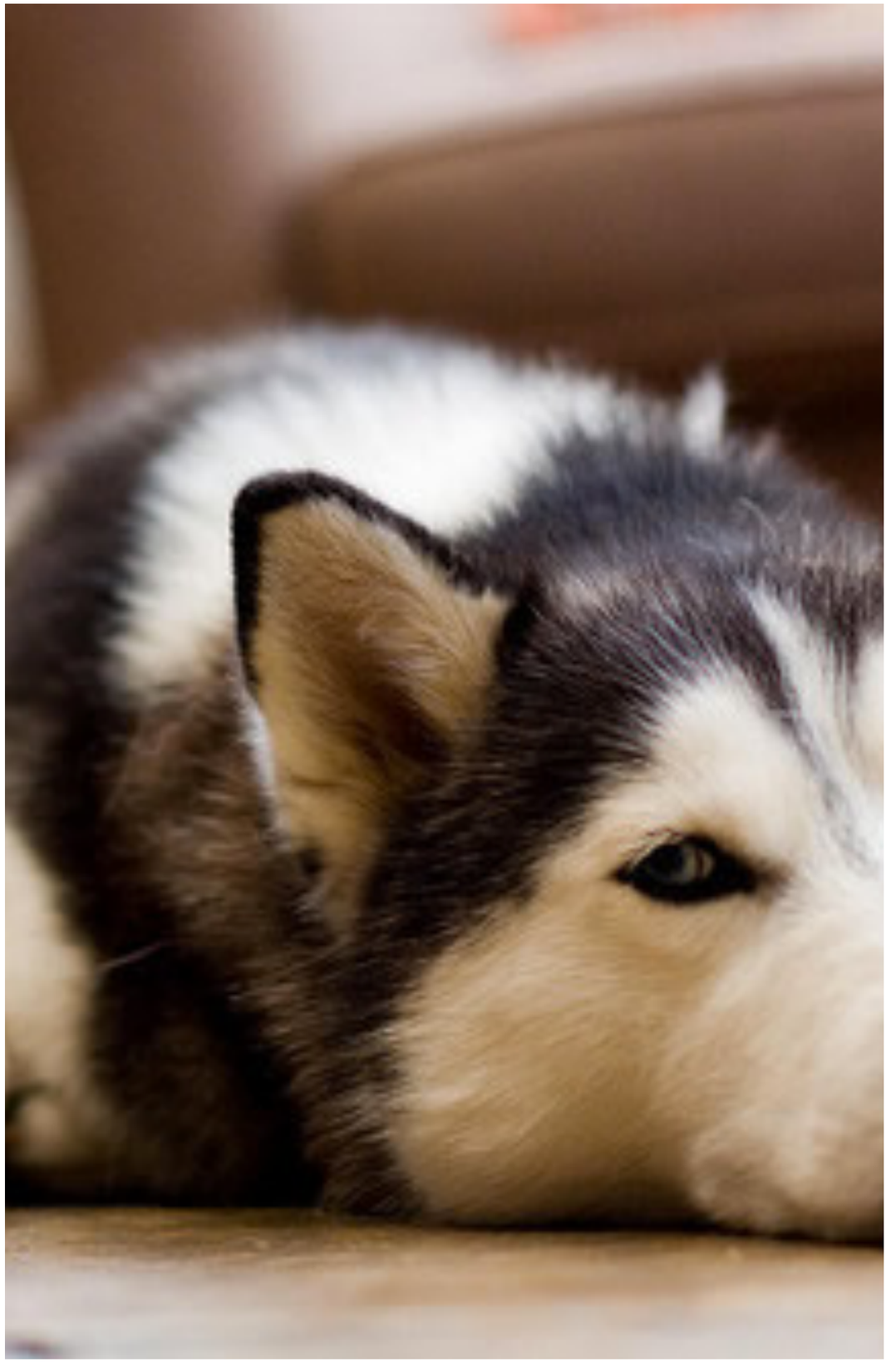
或许

Swift

和

重构

都让你尝尽苦头



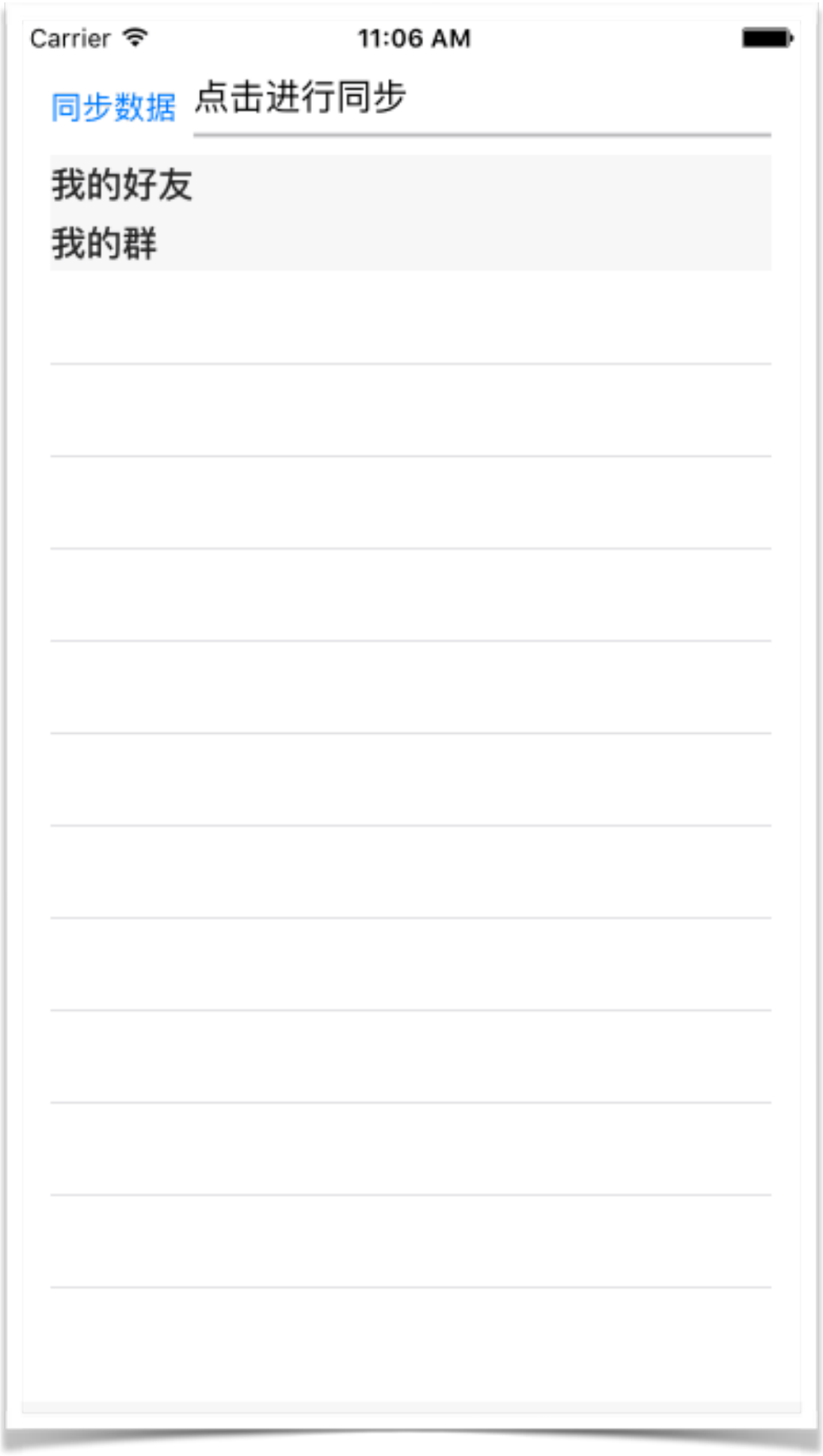
但这一切**绝对值得！**

因为我们需要改变世界

这一切要从“举个栗子”开始

这是我们的例子

一个已知，却一直未被很好解决的问题



非常常见的**既有设计**

```objc
- (IBAction)syncButtonTouched:(UIButton *)sender {
    self.statusLabel.text = @"正在同步好友...";
    [WebAPI requestBuddyList:^(NSArray<Buddy *> *buddys, NSError *error) {
        [self.buddys removeAllObjects];
        [self.buddys addObjectsFromArray:buddys];

        dispatch_async(dispatch_get_main_queue(), ^{
            self.statusLabel.text = @"正在同步群...";
            self.statusProgressView.progress = 0.5;
            [self.tableView reloadData];

            [WebAPI requestGroupList:^(NSArray<Group *> *groups, NSError *error) {
                [self.groups removeAllObjects];
                [self.groups addObjectsFromArray:groups];

                dispatch_async(dispatch_get_main_queue(), ^{
                    self.statusLabel.text = @"点击进行同步";
                    self.statusProgressView.progress = 0.0;
                    [self.tableView reloadData];
                });
            }];
        });
    }];
}
```

这段代码有哪些**问题**

- UI逻辑和数据逻辑混合
- 嵌套层次太深
- 违背了OCP

那就用 **Swift** 吧

```
@IBAction func syncButtonTouched(sender: UIButton) {
    self.statusLabel.text = "正在同步好友..."
    WebAPI.requestBuddyList { (buddys, error) -> Void in
        self.buddys.removeAll()
        self.buddys.appendContentsOf(buddys)

        dispatch_async(dispatch_get_main_queue(), { () -> Void in
            self.statusLabel.text = "正在同步群..."
            self.statusProgressView.progress = 0.5
            self.tableView.reloadData()

            WebAPI.requestGroupList({ (groups, error) -> Void in
                self.groups.removeAll()
                self.groups.appendContentsOf(groups)

                dispatch_async(dispatch_get_main_queue(), { () -> Void in
                    self.statusLabel.text = "点击进行同步"
                    self.statusProgressView.progress = 0.0
                    self.tableView.reloadData()
                })
            })
        })
    }
}
```

没有任何意义，**人肉翻译**

- UI逻辑和数据逻辑混合

回归问题，**解决问题**

- 嵌套层次太深

- 违背了OCP

剥离UI和数据逻辑

提取同步方法，隔离 UI 逻辑

```
func syncBuddys(completion: () -> Void) {
    WebAPI.requestBuddyList { (buddys, error) -> Void in
        self.buddys.removeAll()
        self.buddys.appendContentsOf(buddys)

        completion()
    }
}


func syncGroups(completion: () -> Void) {
    WebAPI.requestGroupList { (groups, error) -> Void in
        self.groups.removeAll()
        self.groups.appendContentsOf(groups)

        completion()
    }
}
```

满满的只剩 UI 逻辑，但是...

```
@IBAction func syncButtonTouched(sender: UIButton) {
    self.statusLabel.text = "正在同步好友..."
    self.syncBuddys {
        dispatch_async(dispatch_get_main_queue()) {
            self.statusLabel.text = "正在同步群..."
            self.statusProgressView.progress = 0.5
            self.tableView.reloadData()
            self.syncGroups {
                dispatch_async(dispatch_get_main_queue()) {
                    self.statusLabel.text = "点击进行同步"
                    self.statusProgressView.progress = 0.0
                    self.tableView.reloadData()
                }
            }
        }
    }
}
```

Don't Repeat Yourself

使用嵌套函数，有效的局部封装

```
@IBAction func syncButtonTouched(sender: UIButton) {

    func updateStatus(text: String, progress: Float = 0.0, reload: Bool = true) {
        self.statusLabel.text = text
        self.statusProgressView.progress = progress
        if reload { self.tableView.reloadData() }
    }

    updateStatus("正在同步好友...", reload: false)
    self.syncBuddys {
        dispatch_async(dispatch_get_main_queue()) {
            updateStatus("正在同步群...", progress: 0.5)
            self.syncGroups {
                dispatch_async(dispatch_get_main_queue()) {
                    updateStatus("点击进行同步")
                }
            }
        }
    }
}
```

展开嵌套层次

自定义操作符，展开嵌套的神器

```
infix operator |> { associativity left precedence 140 }
```

自定义操作符，展开嵌套的神器

```
typealias Action = () -> Void
typealias AsyncTask = (Action) -> Void

func |>(lhs: AsyncTask, rhs: Action) -> AsyncTask {
    return { (action) -> Void in
        lhs { rhs(); action() }
    }
}
```

```
func |>(lhs: AsyncTask, rhs: Action) -> Action {
    return {
        lhs { rhs() }
    }
}

func |>(lhs: AsyncTask, rhs: AsyncTask) -> AsyncTask {
    return { (action) -> Void in
        lhs { rhs(action) }
    }
}
```

所以，展开后是这样的

```
let switchToMainThread: AsyncTask = { (action) -> Void in
    dispatch_async(dispatch_get_main_queue(), action)
}

updateStatus("正在同步好友...", reload: false)

let task: Action = syncBuddys |> switchToMainThread |> {
    updateStatus("正在同步群...", progress: 0.5)
} |> syncGroups |> switchToMainThread |> {
    updateStatus("点击进行同步")
}

task()
```

Functional **Swift**

细分执行步骤，增强可读性

```
let beginSyncBuddys: AsyncTask = switchToMainThread |> {
    updateStatus("正在同步好友...", reload: false)
}

let beginSyncGroups: AsyncTask = switchToMainThread |> {
    updateStatus("正在同步群...", progress: 0.5)
}

let endSync: Action = switchToMainThread |> {
    updateStatus("点击进行同步")
}


let task: Action = beginSyncBuddys |> syncBuddys |>
    beginSyncGroups  |> syncGroups |> endSync

task()
```

提取高阶函数，简化代码

updateStatus:

```
(String, Float, Bool) -> Void
```

```
(String, Float, Bool) -> () -> Void
```

提取高阶函数，简化代码

```
let beginSyncBuddys: AsyncTask = switchToMainThread |>
    updateStatus("正在同步好友...", reload: false)

let beginSyncGroups: AsyncTask = switchToMainThread |>
    updateStatus("正在同步群...", progress: 0.5)

let endSync: Action = switchToMainThread |>
    updateStatus("点击进行同步")
```

最后一个问题：OCP

POP vs OOP vs ISP

提炼基础协议，达到最小依赖

```
func syncBuddys(completion: Action) {
    WebAPI.requestBuddyList { (buddys, error) -> Void in
        self.buddys.removeAll()
        self.buddys.appendContentsOf(buddys)

        completion()
    }
}

func syncGroups(completion: Action) {
    WebAPI.requestGroupList { (groups, error) -> Void in
        self.groups.removeAll()
        self.groups.appendContentsOf(groups)

        completion()
    }
}
```

提炼基础协议，达到最小依赖

```
protocol Synchronizable {
    typealias Element

    var items: [Element] { get }

    func synchronize(completion: Action)
}
```

封装同步逻辑

```
class BuddyDataProvider: Synchronizable {
    typealias Element = Buddy

    private(set) var items = [Buddy]()

    func synchronize(completion: Action) {
        WebAPI.requestBuddyList { (buddys, error) -> Void in
            self.items.removeAll()
            self.items.appendContentsOf(buddys)

            completion()
        }
    }
}

class GroupDataProvider: Synchronizable {
    typealias Element = Group
    ......
}
```

解耦 UI 与 数据 的依赖

提炼 UI 数据协议

- Section
- Section.Name
- Section.RowCount
- Row.Name

提炼 UI 数据协议

```
protocol TableViewSectionDataSource {
    var sectionName: String { get }
    var rowCount: Int { get }

    subscript(i: Int) -> String { get }
}
```

扩展原有实现，更好的接口隔离

```
extension BuddyDataProvider: TableViewSectionDataSource {
    var sectionName: String {
        return "我的好友"
    }

    var rowCount: Int {
        return self.items.count
    }

    subscript(i: Int) -> String {
        return self.items[i].displayName
    }
}

extension GroupDataProvider: TableViewSectionDataSource {
    ...
}
```

定义扩展点

```
func addTableViewSection<T: TableViewSectionDataSource
    where T: Synchronizable>(section: T, updatingText: String)
```

实现扩展点

```
self.tableViewSections.append(section)

if let preSyncTask = self.syncTask {
    let index = self.tableViewSections.count - 1
    let updateProgress: Action = { [unowned self] in
        let progress = Float(index) / Float(self.tableViewSections.count)
        self.statusProgressView.progress = progress
    }
    self.syncTask = preSyncTask |>
                    switchToMainThread |>
                    updateStatus(updatingText) |>
                    updateProgress |>
                    section.synchronize
} else {
    self.syncTask = switchToMainThread |>
                    updateStatus(updatingText, reload: false) |>
                    section.synchronize
}
```

那么，最后的按钮响应是这样

```
@IBAction func syncButtonTouched(sender: UIButton) {
    guard let performSync = self.syncTask else {
        return
    }

    let resetProgress: Action = {
        [unowned self] in
        self.statusProgressView.progress = 0.0
    }

    let endSync: Action = switchToMainThread |> resetProgress |>
        updateStatus("点击进行同步")

    let task: Action = performSync |> endSync

    task()
}
```

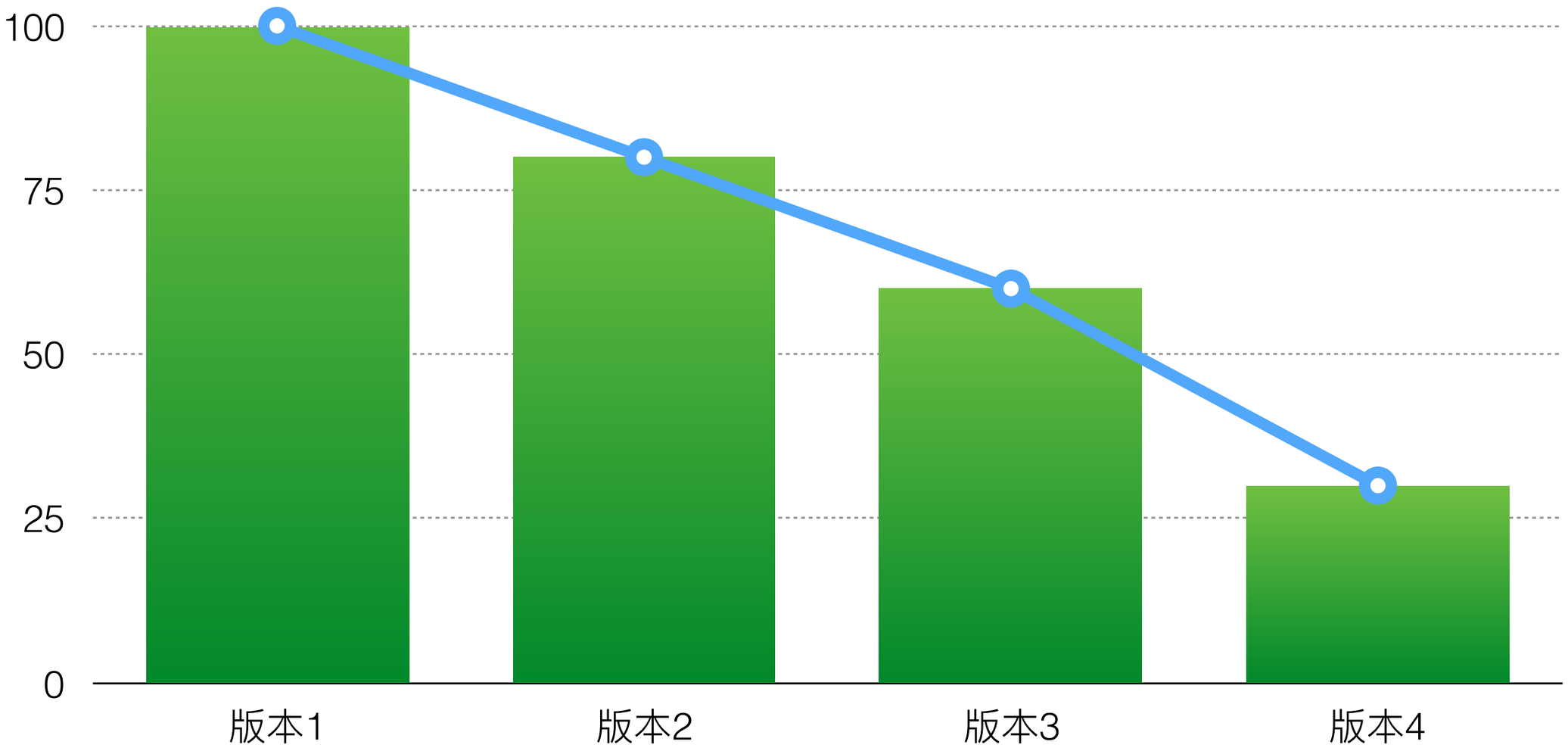
所以面对变化时，这样扩展

```
self.addTableViewSection(BuddyDataProvider(), updatingText: "正在同步好友...")
self.addTableViewSection(GroupDataProvider(), updatingText: "正在同步群...")
self.addTableViewSection(DiscussDataProvider(), updatingText: "正在同步讨论组...")
```

最终我们**解决了**所有问题

但是…

别人的**认知度**是这样



为什么？

优良的设计是...

从无到有

再**从有到无**的过程

Swift 的确改善了设计，但是

- 权衡团队的综合实力
- 传播思想，分享理念
- 稳定住整体结构，不要过度纠结细节
- 接受不完美的完美

谢谢