# iOS 架构设计杂谈

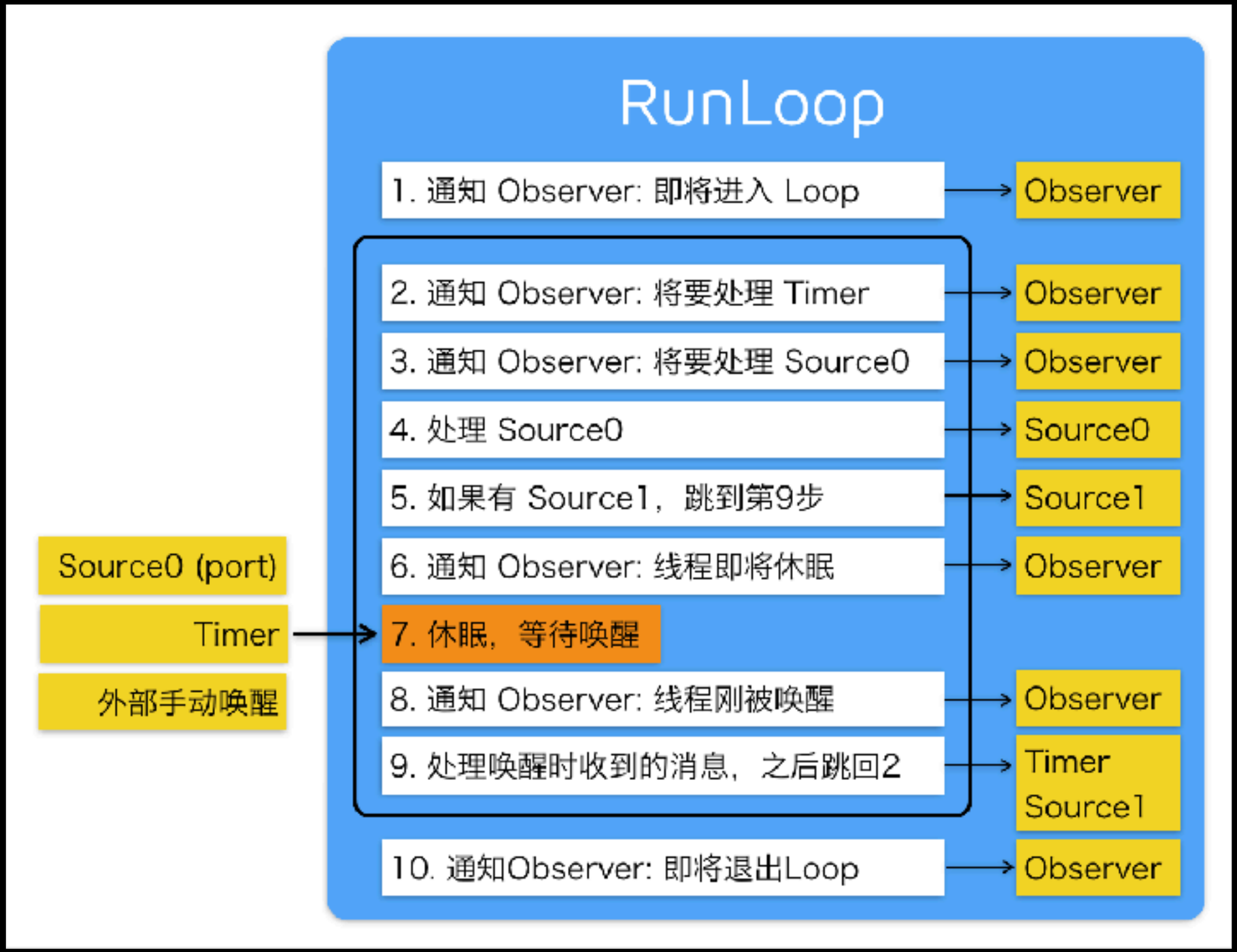
任凯

# 为什么要做架构设计?

# 所有 App 开始的地方

```
8
9  #import <UIKit/UIKit.h>
10 #import "AppDelegate.h"
11
12 int main(int argc, char * argv[]) {
13     @autoreleasepool {
14         return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
15     }
16 }
17
18
```

# 代码背后是 RunLoop 在驱动整个 App



# 如果这是 Apple 的工程师

# 我们写按钮是这样实现的

```
12  int main(int argc, char * argv[]) {
13      @autoreleasepool {
14          while (YES) {
15              @autoreleasepool {
16                  // 按钮有没有创建，没有的话创建按钮
17
18                  // 检查当前用户点击坐标是否在按钮范围
19
20                  // 检查是否有别的东西遮挡
21
22                  // 如果用户点在按钮范围且没有遮挡，就做 XXXXX
23
24                  // 根据 UI 相关对象的属性变化绘制显示画面
25
26                  // ...
27              }
28          }
29
30          return 0;
31      }
32  }
```

# 所以架构设计目的

- 降低业务开发门槛，使得业务开发更容易
- 使工程代码易懂易维护

# 怎样做一个工程的架构

# 代码追求

- DRY - Don’t repeat yourself
- Kiss - Keep It Simple, Stupid

# 六大设计原则（Solid+LoD）

- Single responsibility principle（单一功能原则）
- Open/closed principle（开闭原则）
- Liskov substitution principle（里氏替换原则）
- Interface segregation principle（接口隔离原则）
- Dependency inversion principle（依赖反转原则）
- Law of Demeter（最少知识原则）

- 易理解的 API 设计
- 建立合理依赖关系

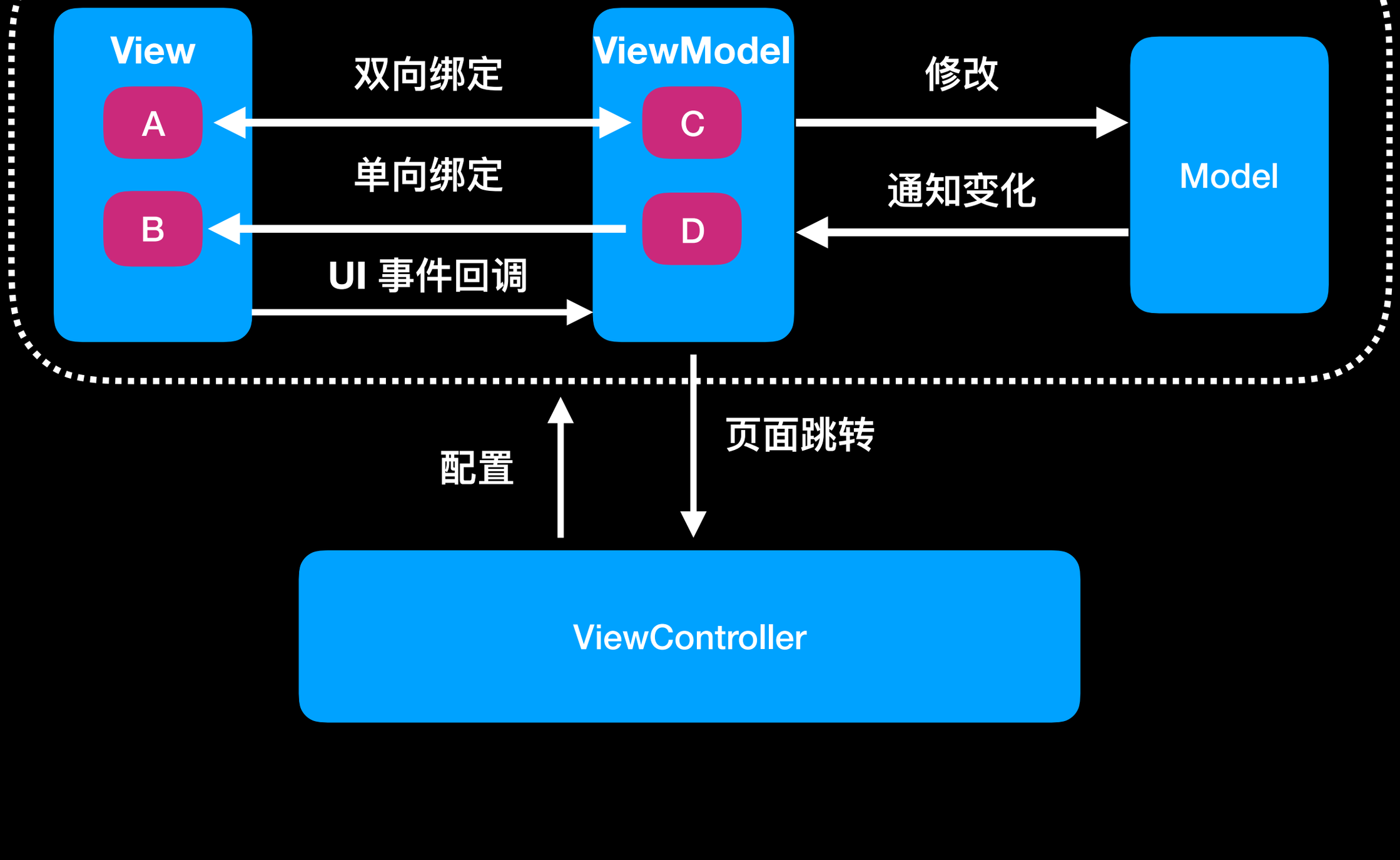
# 方法论

- MVC — View ViewController Model
- MVVM — View ViewModel Model
- Viper — View Interactor Presenter Entity Routing

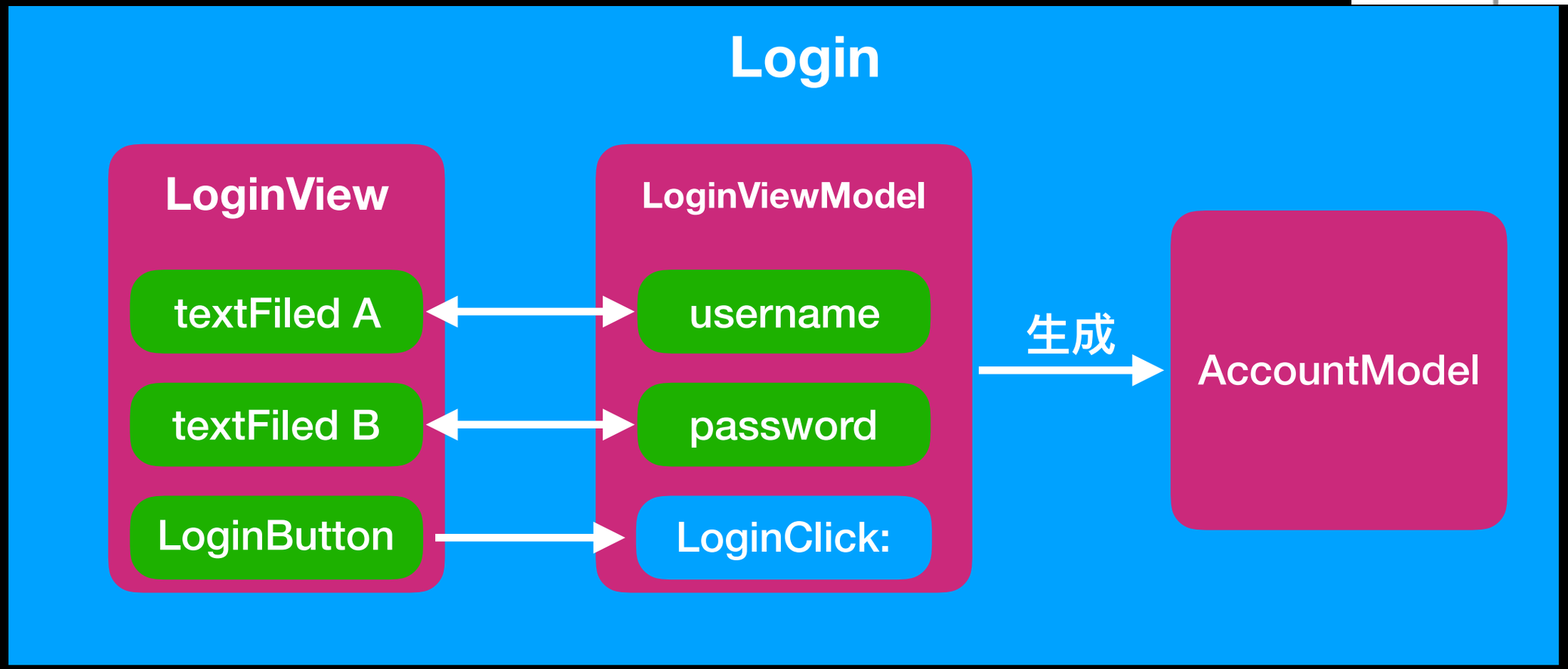
# 案例

以 **MVVM** 的设计方法设计一个登陆功能

# iOS MVVM 实现方案

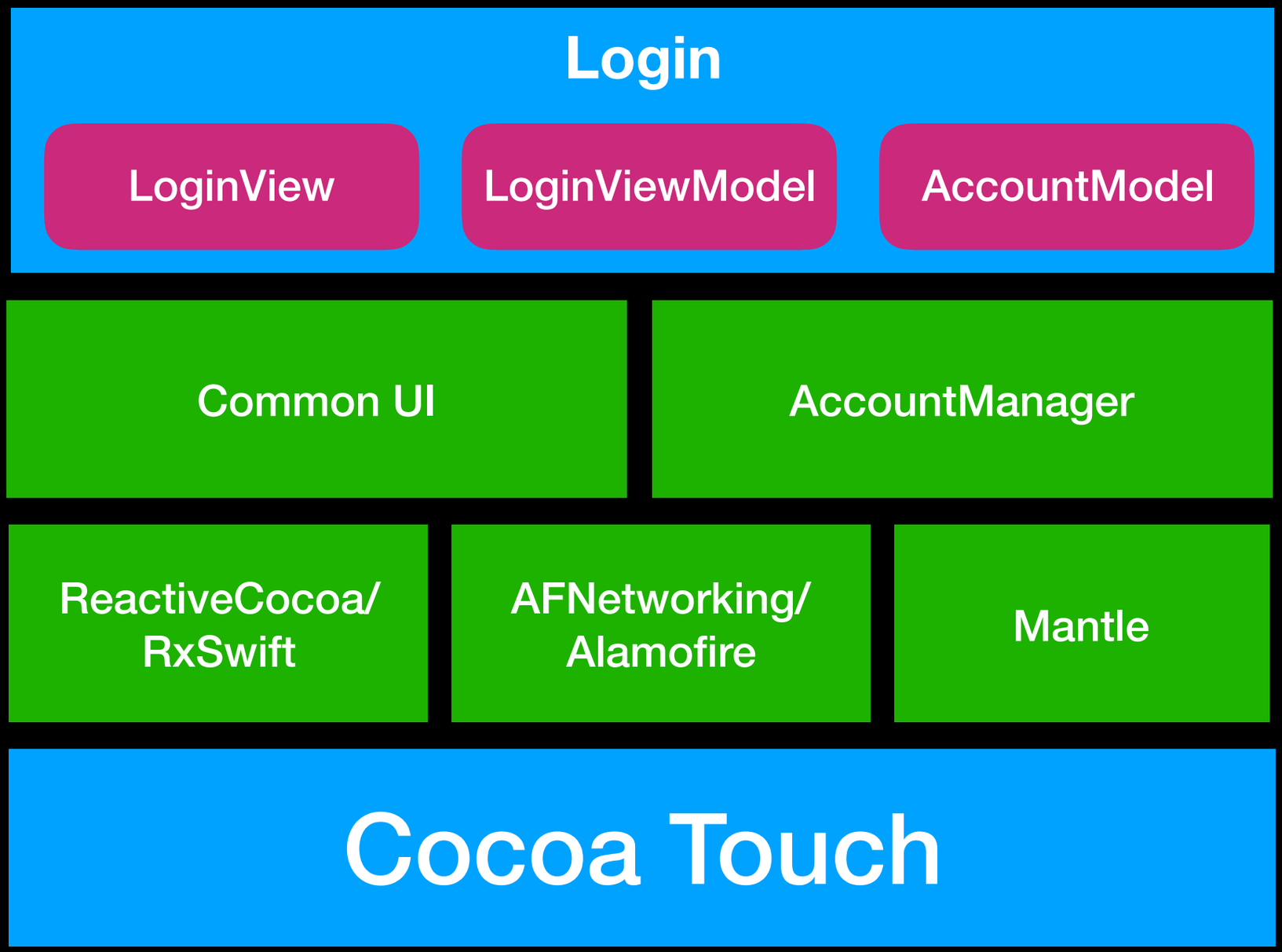


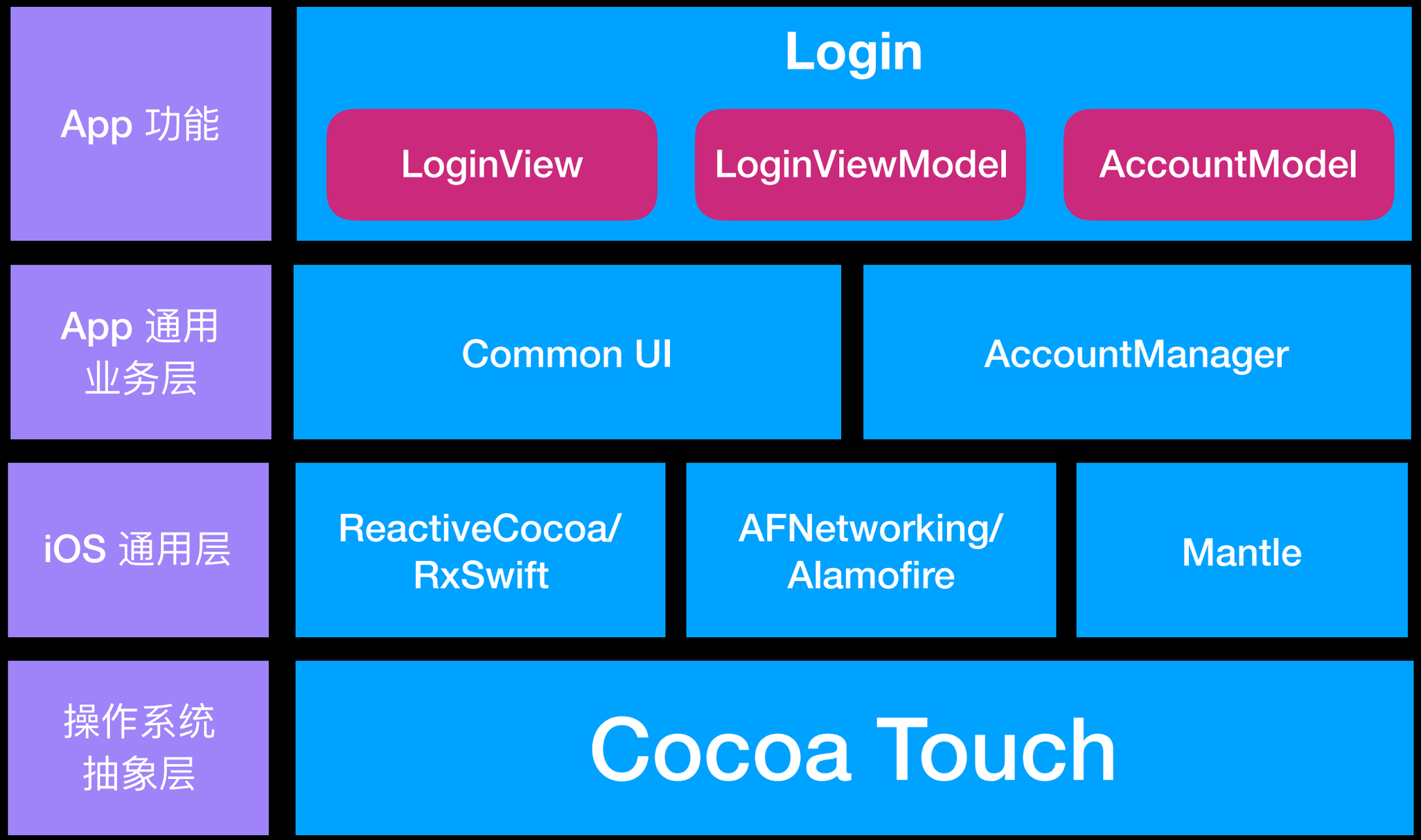
# 初步设计



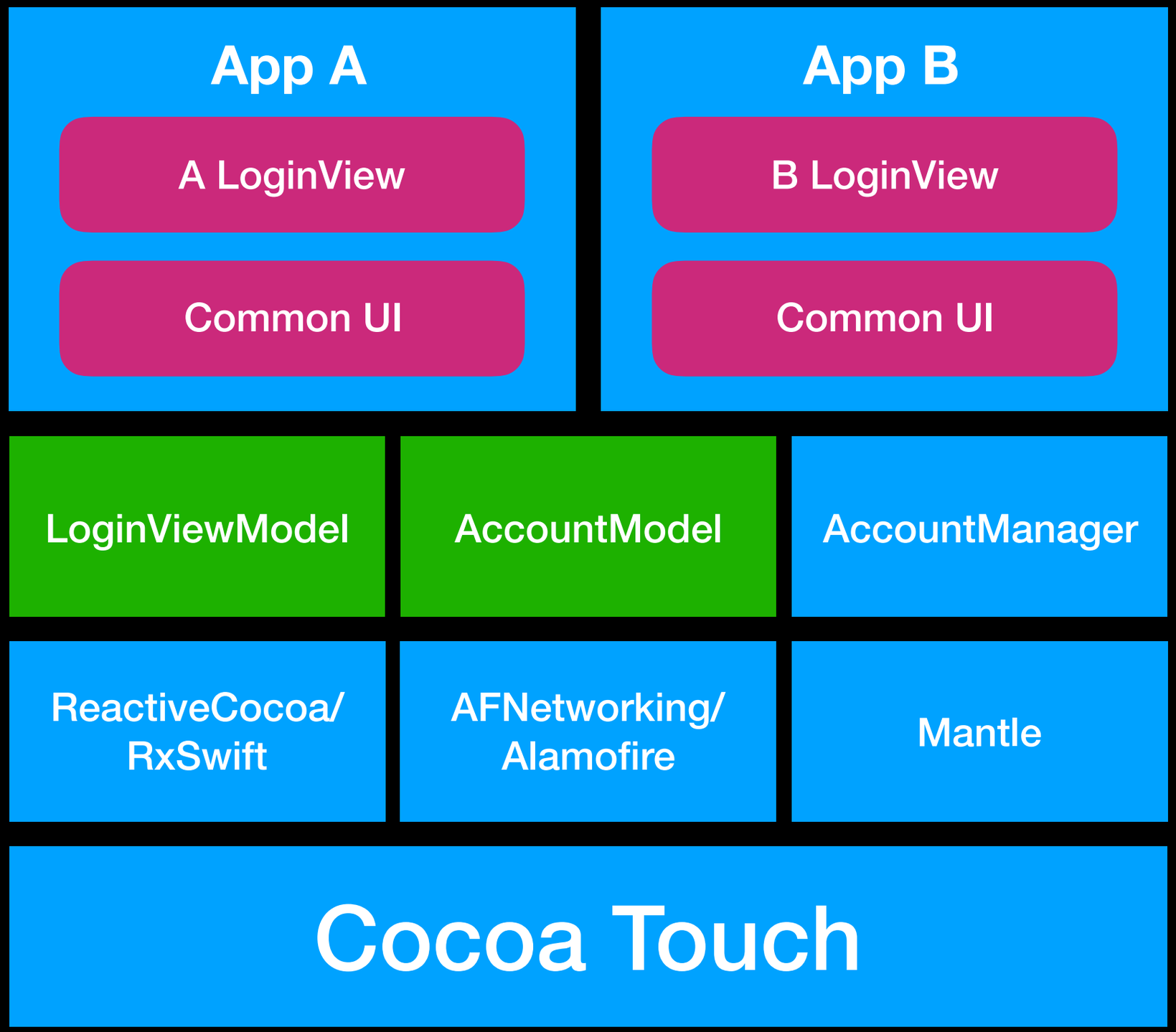

Cocoa Touch

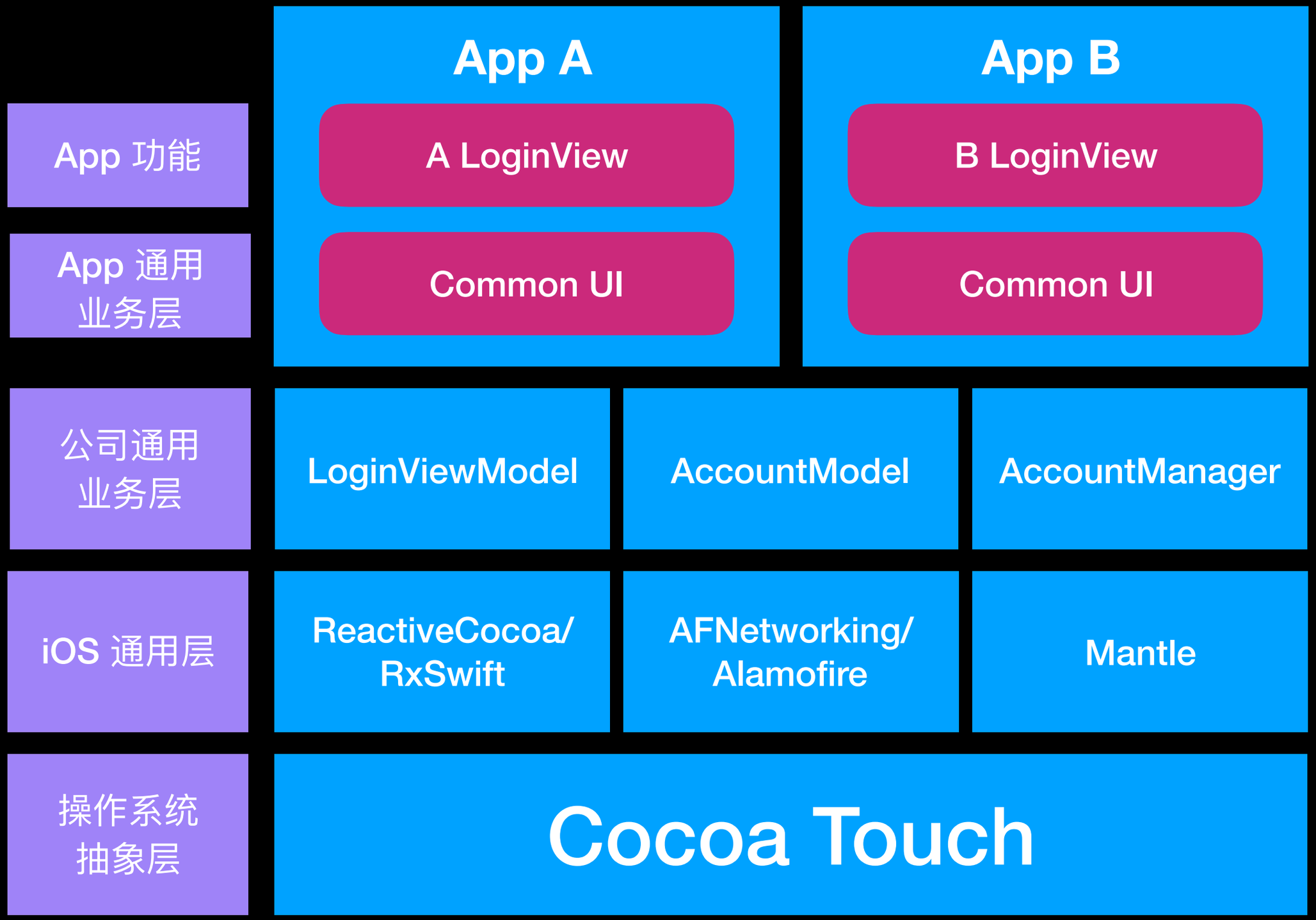
# 为了 DRY，以及开发功能容易，结构改变为





# 当新的 App 需要登陆功能





- MVVM 指导了 UI 与业务逻辑组件拆分
- UI 与业务逻辑的解耦使得不同 App 间的登陆功能有共用的组件
- 丰富的 iOS 通用层组件使绑定、网络请求、数据反序列化变得更容易实现

架构设计需要要产出的重要结果是三个通用层有丰富工具、模块
使大部分业务开发可以通过组合通用层的模块工具实现

App 功能

App 通用业务层

公司通用业务层

iOS 通用层

操作系统抽象层

# 如何能设计好模块

- 坚持设计原则
- 了解常用模块设计方法
- 不断反思改进

# 架构如何在项目中执行

- 建立合理的工程文件结构
- 达成团队架构设计共识

# 人人都是架构师


# The End