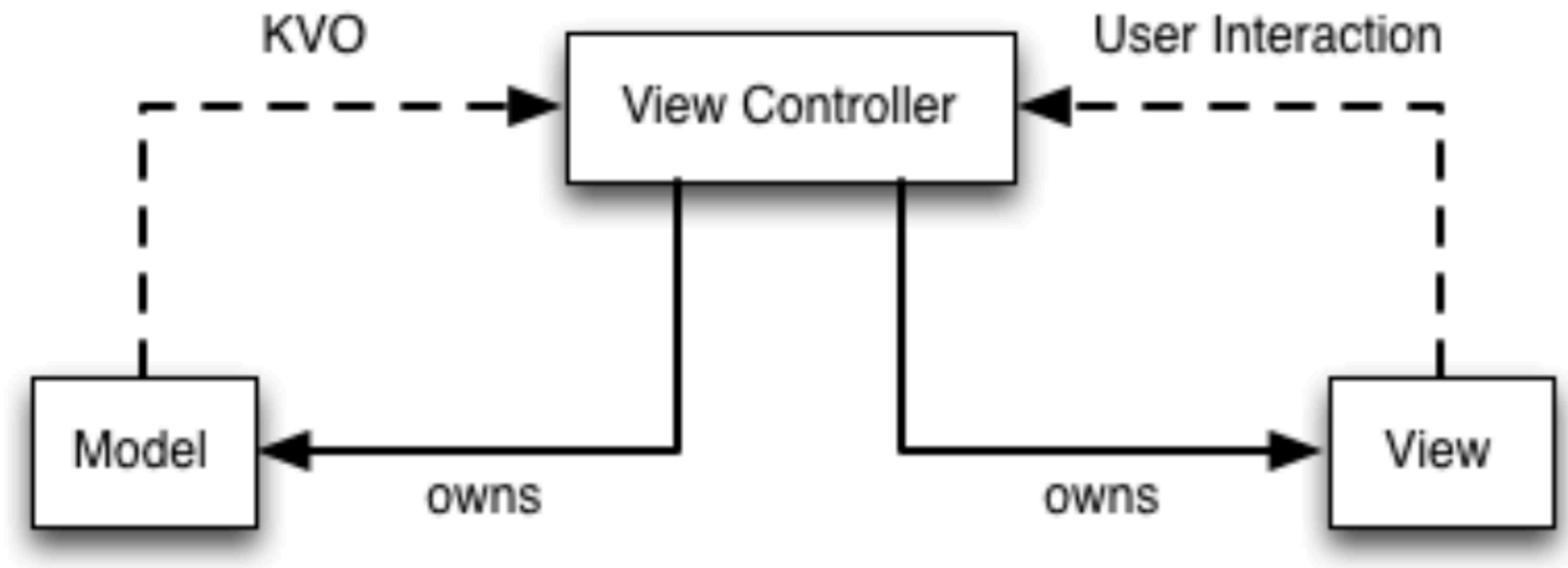
1. ReactiveCocoa 简介（作者、作用、为什么要用）
2. ReactiveCocoa 作用
3. ReactiveCocoa 工作原理和编程思想
4. ReactiveCocoa 框架思维导图
5. ReactiveCocoa 常见类
6. ReactiveCocoa 常见宏
7. ReactiveCocoa 常见用法
8. ReactiveCocoa 常见方法
9. UI - Category（常用汇总）
10. Foundation - Category （常用汇总）

1. ReactiveCocoa简介

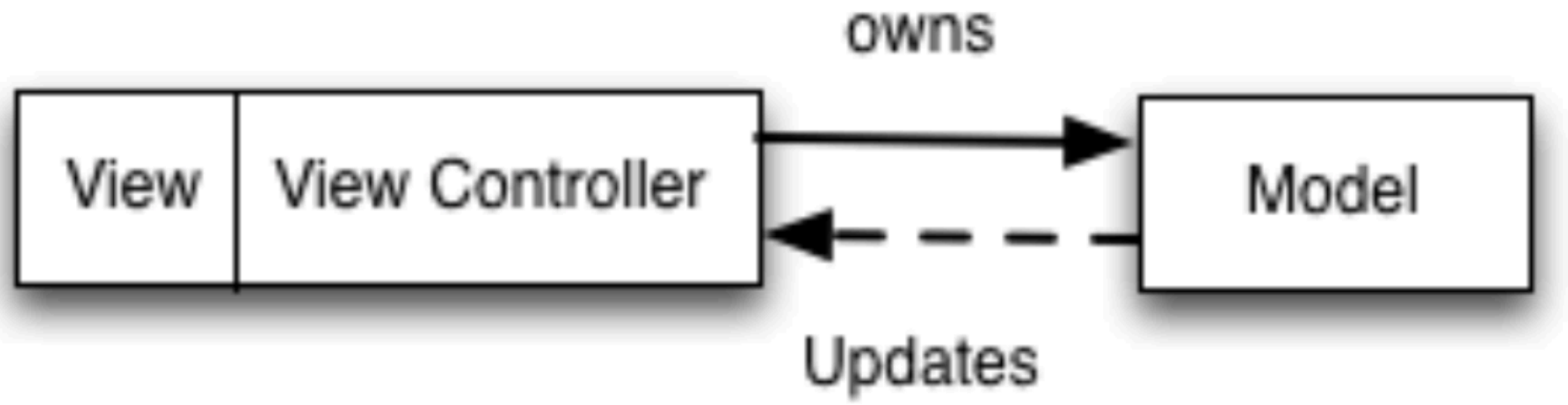
ReactiveCocoa（简称为RAC），是由Github开源的一个应用于iOS和OS开发的新框架，Cocoa是苹果整套框架的简称，因此很多苹果框架喜欢以Cocoa结尾。

2. ReactiveCocoa作用

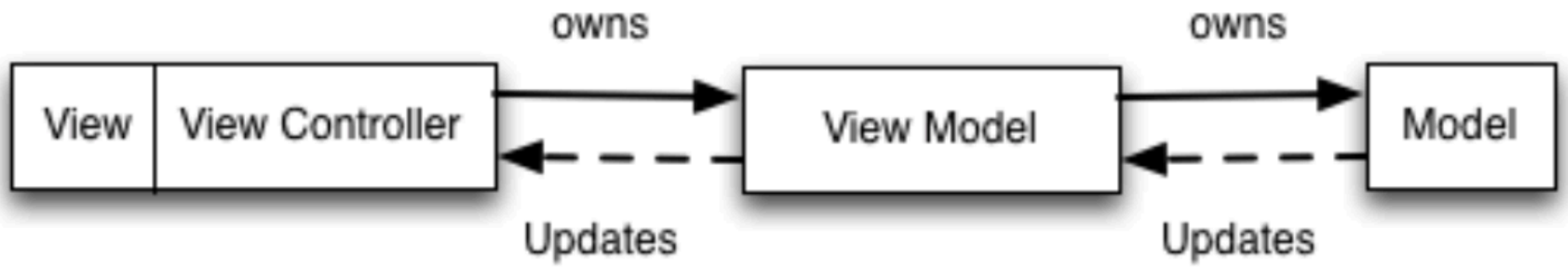
谈到RAC的作用，首先来看一下一般 iOS 工程的构建方法。
MVC，一个典型的 iOS 工程的构建形式，Model 呈现数据，View 呈现用户界面，而 View Controller 调节它两者之间的交互。

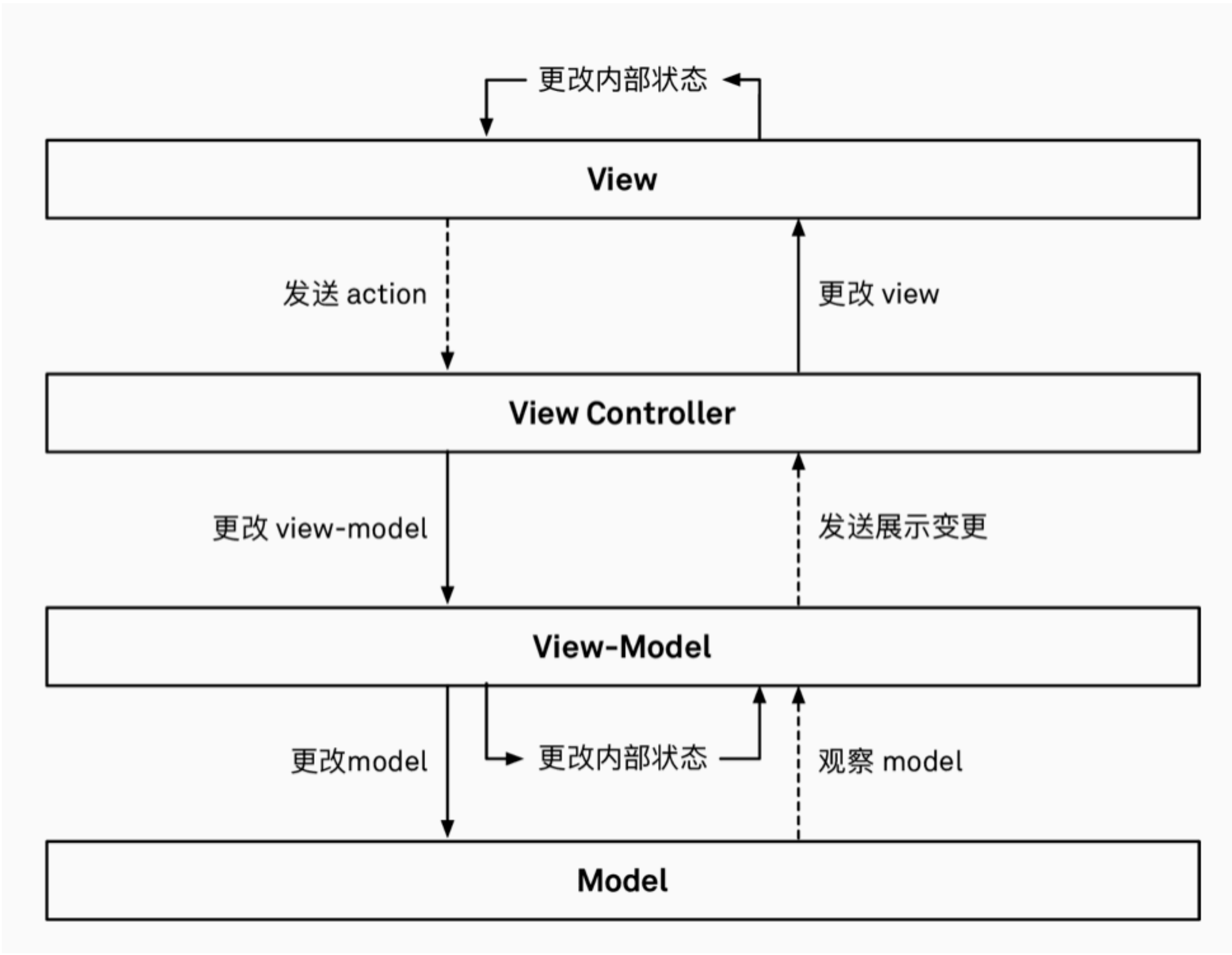


然而，虽然 View 和 View Controller 是技术上不同的组件，但它们几乎总是成对的。
下面这张图准确地描述了我们可能已经编写的 MVC 代码。但它并没有做太多事情来解决 iOS 应用中C层臃肿的的问题。在典型的 `MVC` 应用里，许多逻辑被放在 `View Controller` 里 。它们中的一些确实属于 View Controller，但更多的是所谓的“表示逻辑（presentation logic）”，以 MVVM 的术语来说，就是那些将 Model 数据转换为 View 可以呈现的东西的事情，例如将一个 NSDate 转换为一个格式化过的 NSString。



我们图解里缺少某些东西，那些使我们可以把所有表示逻辑放进去的东西。我们打算将其称为 `“View Model”` —— 它位于 View/Controller 与 Model 之间：

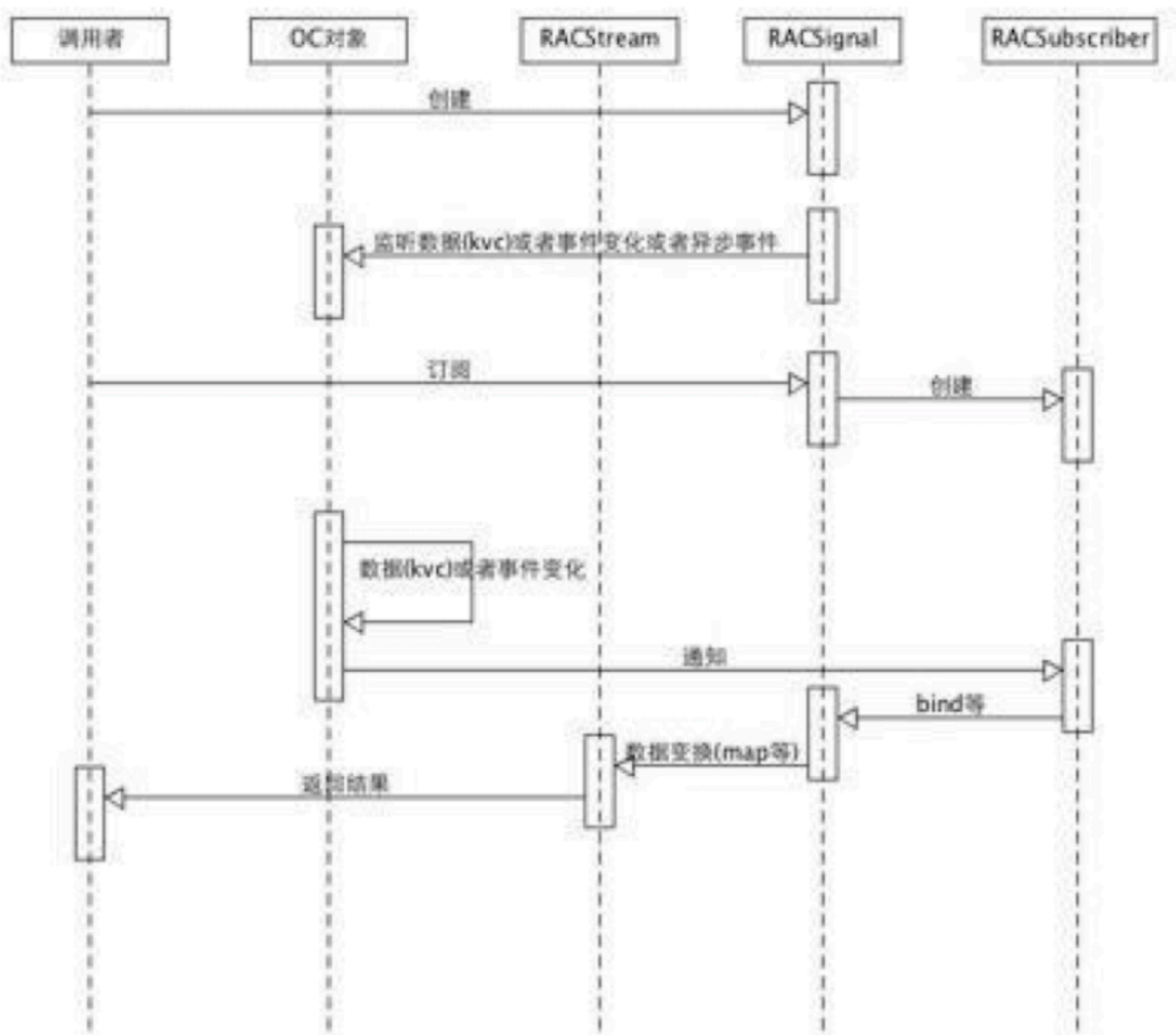




这个图解准确地描述了什么是 MVVM：一个 MVC 的增强版，我们正式连接了视图和控制器，并将表示逻辑从 Controller 移出放到一个新的对象里，即 View Model。MVVM 本质上就是一个精心优化的 MVC 架构。对我们来说，就是它能减少 `ViewController` 的复杂性并使得表示逻辑更易于测试 。

RAC的作用就在于它提供了很好的API可以很好的配合了MVVM的使用。
在我们iOS开发过程中，当某些事件响应的时候，需要处理某些业务逻辑，这些事件都用不同的方式来处理。比如按钮的点击使用action，ScrollView滚动使用delegate，属性值改变使用KVO等系统提供的方式。其实这些事件，都可以通过RAC处理。
ReactiveCocoa为事件提供了很多处理方法，而且利用RAC处理事件很方便，可以把要处理的事情，和监听的事情的代码放在一起，这样非常方便我们管理，就不需要跳到对应的方法里。非常符合我们开发中高聚合，低耦合 的思想。

3. ReactiveCocoa工作原理和编程思想



1. ReactiveCocoa编程思想：

   - 函数式编程：
     百度百科： 简单说，"函数式编程"是一种"编程范式"（programming paradigm），也就是如何编写程序的方法论。它属于"结构化编程"的一种，主要思想是把运算过程尽量写成一系列嵌套的函数调用。
     函数式编程思想：是把操作尽量写成一系列嵌套的函数或者方法调用。
     函数式编程特点：每个方法必须有返回值（本身对象）,把函数或者Block当做参数,block参数（需要操作的值）block返回值（操作结果）。
     在RAC代码块的复用。也就是信号量的复用。
   - 响应式编程：
     百度百科：响应式编程是一种面向数据流和变化传播的编程范式。这意味着可以在编程语言中很方便地表达静态或动态的数据流，而相关的计算模型会自动将变化的值通过数据流进行传播。
     响应式编程思想：不需要考虑调用顺序，只需要知道考虑结果，类似于蝴蝶效应，产生一个事件，会影响很多东西，这些事件像流一样的传播出去，然后影响结果，借用面向对象的一句话，万物皆是流。
     关于响应式编程，RAC将我们关注的点放到绑定。每一步都在操作和处理数据，但却把操作点从以前的数据本身转移到函数方法上边去。将变量之间建立绑定，这就是响应式编程的内容。

   ReactiveCocoa被描述为函数响应式编程（FRP） 框架。
   以后使用RAC解决问题，就不需要考虑调用顺序，直接考虑结果，把每一次操作都写成一系列嵌套的方法中，使代码高聚合，方便管理。

2. 操作思想

   - 运用的是Hook（钩子）思想，Hook是一种用于改变API(应用程序编程接口：方法)执行结果的技术.
   - Hook用处：截获API调用的技术。
   - Hook原理：在每次调用一个API返回结果之前，先执行你自己的方法，改变结果的输出。

4. ReactiveCocoa思维导图

5. ReactiveCocoa常见类

学习框架首先搞清楚框架中常用的类，在RAC中最核心的类RACSiganl。

- **RACSiganl**:信号类,一般表示将来有数据传递，只要有数据改变，信号内部接收到数据，就会马上发出数据。

> `RACEmptySignal` ： 空信号，用来实现 RACSignal 的 +empty 方法；

> `RACReturnSignal` ： 一元信号，用来实现 RACSignal 的 +return: 方法；
> `RACDynamicSignal` ： 动态信号，使用一个 block - 来实现订阅行为，我们在使用 RACSignal 的 +createSignal: 方法时创建的就是该类的实例；
> `RACErrorSignal` ： 错误信号，用来实现 RACSignal 的 +error: 方法；
> `RACChannelTerminal` ： 通道终端，代表 RACChannel 的一个终端，用来实现双向绑定。

注意

> 信号类(RACSiganl)，只是表示当数据改变时，信号内部会发出数据，它本身不具备发送信号的能力，而是交给内部一个订阅者去发出。
> 默认一个信号都是冷信号，也就是值改变了，也不会触发，只有订阅了这个信号，这个信号才会变为热信号，值改变了才会触发。
> 如何订阅信号：调用信号RACSignal的subscribeNext就能订阅

- **RACSubscriber**:表示订阅者的意思，用于发送信号，这是一个协议
- **RACDisposable**:用于取消订阅或者清理资源，当信号发送完成或者发送错误的时候，就会自动触发它。

> `RACSerialDisposable` ： 作为 disposable 的容器使用，可以包含一个 disposable 对象，并且允许将这个 disposable 对象通过原子操作交换出来
> `RACKVOTrampoline` ： 代表一次 KVO 观察，并且可以用来停止观察
> `RACCompoundDisposable` ： 它可以包含多个 disposable 对象，并且支持手动添加和移除 disposable 对象
> `RACScopedDisposable` ： 当它被 dealloc 的时候调用本身的 -dispose 方法。

- **RACSubject**

> `RACSubject`： 信号提供者，自己可以充当信号，又能发送信号
> 使用场景:通常用来代替代理，有了它，就不必要定义代理了
> `RACReplaySubject`： 重复提供信号类，RACSubject的子类
> `RACGroupedSignal` ： 分组信号，用来实现 RACSignal 的分组功能
> `RACBehaviorSubject` ： 重演最后值的信号，当被订阅时，会向订阅者发送它最后接收到的值
> `RACReplaySubject` ： 重演信号，保存发送过的值，当被订阅时，会向订阅者重新发送这些值

> RACReplaySubject与RACSubject区别:
> RACReplaySubject可以先发送信号，在订阅信号，RACSubject就不可以。

> 使用场景:
> 使用场景一:如果一个信号每被订阅一次，就需要把之前的值重复发送一遍，使用重复提供信号类。
> 使用场景二:可以设置capacity数量来限制缓存的value的数量,即只缓充最新的几个值`

- **RACSequence**:RAC中的集合类，用于代替NSArray,NSDictionary,可以使用它来快速遍历数组和字典
- **RACCommand**:RAC中用于处理事件的类，可以把事件如何处理,事件中的数据如何传递，包装到这个类中，他可以很方便的监控事件的执行过程。

> 使用场景:监听按钮点击，网络请求

- RACMulticastConnection:用于当一个信号，被多次订阅时，为了保证创建信号时，避免多次调用创建信号中的block，造成副作用，可以使用这个类处理。

> 使用注意:RACMulticastConnection通过RACSignal的-publish或者-muticast:方法创建

- **RACTuple**:元组类,类似NSArray,用来包装值
- **RACScheduler**:RAC中的队列，用GCD封装的

> `RACImmediateScheduler` ： 立即执行调度的任务，这是唯一一个支持同步执行的调度器
> `RACQueueScheduler` ： 一个抽象的队列调度器，在一个 GCD 串行列队中异步调度所有任务
> `RACTargetQueueScheduler` ： 继承自 RACQueueScheduler ，在一个以一个任意的 GCD 队列为 target 的串行队列中异步调度所有任务
> `RACSubscriptionScheduler` ： 一个只用来调度订阅的调度器

- **RACUnit** :表示stream不包含有意义的值,也就是看到这个，可以直接理解为nil
- **RACEvent**: 把数据包装成信号事件(signal event)。它主要通过RACSignal的-materialize来使用

6. ReactiveCocoa常见宏

1. `RAC(TARGET, [KEYPATH, [NIL_VALUE]])`:用于给某个对象的某个属性绑定。
2. `RACObserve(self, name)`:监听某个对象的某个属性,返回的是信号。
3. `RACTuplePack`： 把数据包装成RACTuple（元组类）
4. `RACTupleUnpack`： 把RACTuple（元组类）解包成对应的数据

7. ReactiveCocoa常见用法

- `rac_signalForSelector`： 用于替代代理。
- `rac_valuesAndChangesForKeyPath`： 代替KVO，用于监听某个对象的属性改变。
- `rac_signalForControlEvents`： 用于监听某个事件。
- `rac_addObserverForName`:用于监听某个通知。
- `rac_textSignal`:监听文本框文字改变。
- `rac_liftSelector:withSignalsFromArray:Signals`:当传入的Signals(信号数组)，每一个signal都至少sendNext过一次，就会去触发第一个selector参数的方法。
  (使用注意：几个信号，参数一的方法就几个参数，每个参数对应信号发出的数据。处理当界面有多次请求时，需要都获取到数据时，才能展示界面：)

8. ReactiveCocoa常见方法

1. ReactiveCocoa操作的核心方法是bind（绑定）,而且RAC中核心开发方式，也是绑定，之前的开发方式是赋值，而用RAC开发，应该把重心放在绑定，也就是可以在创建一个对象的时候，就绑定好以后想要做的事情，而不是等赋值之后在去做事情。
2. 列如：把数据展示到控件上，之前都是重写控件的setModel方法，用RAC就可以在一开始创建控件的时候，就绑定好数据。
3. 在开发中很少使用bind方法，bind属于RAC中的底层方法，RAC已经封装了很多好用的其他方法，底层都是调用bind，用法比bind简单(flattenMap、skip、take、takeUntilBlock、skipUntilBlock、distinctUntilChanged等function都用到了bind方法。我们可以发现很多地方都继承和重写bind方法).
4. 修改bindBlock 从而做到修改返回

- ReactiveCocoa操作方法之映射

> `Map` :把源信号的值映射成一个新的值
> `flattenMap` :把源信号的内容映射成一个新的信号，信号可以是任意类型。

> `FlatternMap和Map的区别`
> 1.FlatternMap中的Block返回信号。
> 2.Map中的Block返回对象。
> 3.开发中，如果信号发出的值不是信号，映射一般使用Map
> 4.开发中，如果信号发出的值是信号，映射一般使用FlatternMap。

- ReactiveCocoa操作方法之组合

> `concat` :按一定顺序拼接信号，当多个信号发出的时候，有顺序的接收信号
> `then` :用于连接两个信号，当第一个信号完成，才会连接then返回的信号
> `merge` :把多个信号合并为一个信号，任何一个信号有新值的时候就会调用
> `zipWith` :把两个信号压缩成一个信号，只有当两个信号同时发出信号内容时，并且把两个信号的内容合并成一个元组，才会触发压缩流的next事件
> `combineLatest` :将多个信号合并起来，并且拿到各个信号的最新的值,必须每个合并的signal至少都有过一次sendNext，才会触发合并的信号
> `reduce(聚合)`:用于信号发出的内容是元组，把信号发出元组的值聚合成一个值

- ReactiveCocoa操作方法之过滤

> `filter`： 过滤，每次信号发出，会先执行过滤条件判断
> `ignore`： 内部调用filter过滤，忽略掉ignore的值
> `distinctUntilChanged`： 当上一次的值和当前的值有明显的变化就会发出信号，否则会被忽略掉
> `take`： 从开始一共取N次的信号
> `takeLast` :取最后N次的信号,前提条件，订阅者必须调用完成，因为只有完成，就知道总共有多少信号
> `takeUntil` :(RACSignal *):获取信号直到某个信号执行完成
> `skip` :(NSUInteger):跳过几个信号,不接受
> `switchToLatest` :用于signalOfSignals（信号的信号），有时候信号也会发出信号，会在signalOfSignals中，获取signalOfSignals发送的最新信号

- 执行顺序

> `doNext` : 执行Next之前，会先执行这个Block
> `doCompleted` : 执行sendCompleted之前，会先执行这个Block

- ReactiveCocoa操作方法之时间

> `timeout`： 超时，可以让一个信号在一定的时间后，自动报错
> `interval(定时)`： 每隔一段时间发出信号
> `delay`： 延迟发送next

- ReactiveCocoa操作方法之重复

> `retry(重试)` ： 只要失败，就会重新执行创建信号中的block,直到成功
> `replay(重放)`： 当一个信号被多次订阅,反复播放内容
> `throttle(节流)`:当某个信号发送比较频繁时，可以使用节流，在某一段时间不发送信号内容，过了一段时间获取信号的最新内容发出

- ReactiveCocoa操作方法之线程

> `deliverOn` : 内容传递切换到制定线程中，副作用在原来线程中,把在创建信号时block中的代码称之为副作用
> `subscribeOn` : 内容传递和副作用都会切换到制定线程中

9. UI - Category（常用汇总）

1. rac_prepareForReuseSignal： 需要复用时用
   相关UI: MKAnnotationView、UICollectionReusableView、UITableViewCell、UITableViewHeaderFooterView
2. rac_buttonClickedSignal：点击事件触发信号
   相关UI：UIActionSheet、UIAlertView
3. rac_command：button类、刷新类相关命令替换
   相关UI：UIBarButtonItem、UIButton、UIRefreshControl
4. rac_signalForControlEvents: control event 触发
   相关UI：UIControl
5. rac_gestureSignal UIGestureRecognizer 事件处理信号

相关UI：UIGestureRecognizer

6. rac_imageSelectedSignal 选择图片的信号
   相关UI：UIImagePickerController
7. rac_textSignal
   相关UI：UITextField、UITextView
8. 可实现双向绑定的相关API
   rac_channelForControlEvents: key: nilValue:
   相关UI：UIControl类
   rac_newDateChannelWithNilValue:
   相关UI：UIDatePicker
   rac_newSelectedSegmentIndexChannelWithNilValue:
   相关UI：UISegmentedControl
   rac_newValueChannelWithNilValue:
   相关UI：UISlider、UIStepper
   rac_newOnChannel
   相关UI：UISwitch
   rac_newTextChannel
   相关UI：UITextField

10. Foundation - Category （常用汇总）

1. NSArray
   rac_sequence 信号集合
2. NSData
   rac_readContentsOfURL: options: scheduler: 比oc多出线程设置
3. NSDictionary
   rac_sequence 不解释
   rac_keySequence key 集合
   rac_valueSequence value 集合
4. NSEnumerator
   rac_sequence 不解释
5. NSFileHandle
   rac_readInBackground 见名知意
6. NSIndexSet
   rac_sequence 不解释
7. NSInvocation
   rac_setArgument: atIndex: 设置参数
   rac_argumentAtIndex 取某个参数
   rac_returnValue 所关联方法的返回值
8. NSNotificationCenter
   rac_addObserverForName: object:注册通知
9. NSObject
   rac_willDeallocSignal 对象销毁时发动的信号
   rac_description debug用
   rac_observeKeyPath: options: observer: block:监听某个事件
   rac_liftSelector: withSignals: 全部信号都next在执行
   rac_signalForSelector: 代替某个方法
   rac_signalForSelector:(SEL)selector fromProtocol:代替代理
10. NSOrderedSet
    rac_sequence 不解释
11. NSSet
    rac_sequence 不解释
12. NSString
    rac_keyPathComponents 获取一个路径所有的部分
    rac_keyPathByDeletingLastKeyPathComponent 删除路径最后一部分
    rac_keyPathByDeletingFirstKeyPathComponent 删除路径第一部分
    rac_sequence 不解释 (character)
    rac_readContentsOfURL: usedEncoding: scheduler: 比之OC多线程调用
13. NSURLConnection
    rac_sendAsynchronousRequest 发起异步请求
14. NSUserDefaults
    rac_channelTerminalForKey 用于双向绑定，此乃一端