聊聊FRP，RAC 和 MVVM - 王韧竹

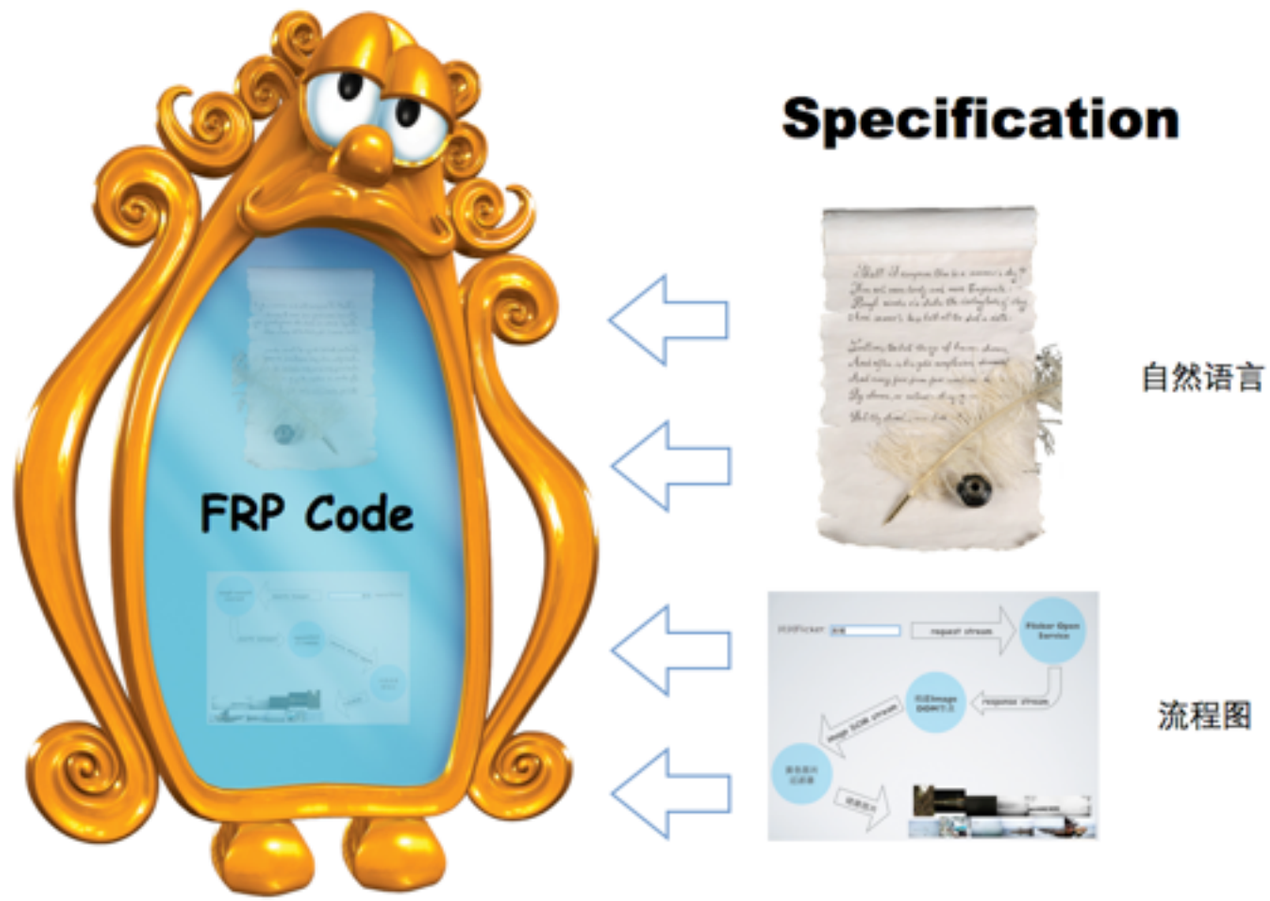
IOS 中的函数式编程 与 RAC 技术探讨

什么是FRP?

Functional Programing ＋ Reactive ＝ **FRP**

Functional Reactive Programing 函数响应式编程

主要利用函数式编程(Functional Programming)的思想和方法(函数、高阶函数)来支持Reactive Programming就是所谓的Functional Reactive Programming，简称FRP



函数式编程的起源：LISP

```
(setq a 10)
(cond ((> a 20)
    (format t "~% a is less than 20"))
(t (format t "~% value of a is ~d " a)))
```

1958年，John McCarthy设计了Lisp语言

```
value of a is 10
```

Java、Perl、Python、Swift、Haskell、Ruby、。你会发现，排在越后面的语言，越像Lisp。

编程语言现在的发展，不过刚刚赶上1958年Lisp语言的水平。

1958年的技术，怎么可能超过今天的水平呢?

这是因为John McCarthy本来没打算把Lisp设计成编程语言，至少不是我们现在意义上的编程语言。他的原意只是想做一种理论演算，用更简洁的方式定义图灵机。

因为这种语言本质上不是一种技术，而是数学。数学是不会过时的。

LISP语言诞生的时候，就包含了9种新思想

1. 条件结构（即"if-then-else"结构）

2. 函数也是一种数据类型。

3. 递归

4. 变量的动态类型。

5. 垃圾回收机制

6. 程序由表达式（expression）组成

7. 符号（symbol）类型 符号实际上是一种指针，指向储存在哈希表中的字符串。所以，比较两个符号是否相等，只要看它们的指针是否一样就行了，不用逐个字符地比较（eq 'a 'a）

8. 代码使用符号和常量组成的树形表示法（notation） 用一门语言自己的数据结构来表达该语言

9. 无论什么时候，整个语言都是可用的。Lisp并不真正区分读取期、编译期和运行期。

思想1到思想5已经被广泛接受，思想6开始在主流编程语言中出现，思想7在Python语言中有所实现，不过似乎没有专用的语法。

LISP的编程优势体现在哪里呢

语言的编程能力越强大，写出来的程序就越短（当然不是指字符数量，而是指独立的语法单位）。

如果同一个软件，一种语言写出来的代码比另一种语言长三倍，这意味着你开发它耗费的时间也会多三倍

如果使用Lisp语言，能让程序变得多短？以Lisp和C的比较为例，我听到的大多数说法是C代码的长度是Lisp的7倍到10倍。但是最近，New Architect杂志上有一篇介绍ITA软件公司的文章，里面说"一行Lisp代码相当于20行C代码"，因为此文都是引用ITA总裁的话，所以我想这个数字来自ITA的编程实践。 如果真是这样，那么我们可以相信这句话。ITA的软件，不仅使用Lisp语言，还同时大量使用C和C++，所以这是他们的经验谈。

格林斯潘第十定律

实战中运用函数式思想的框架

Masonry

```
[self.imgNew mas_makeConstraints:^(MASConstraintMaker *make) {
    make.bottom.mas_equalTo(self.uvContentView.mas_bottom).offset(-10).and.priorityHigh();
}];
```

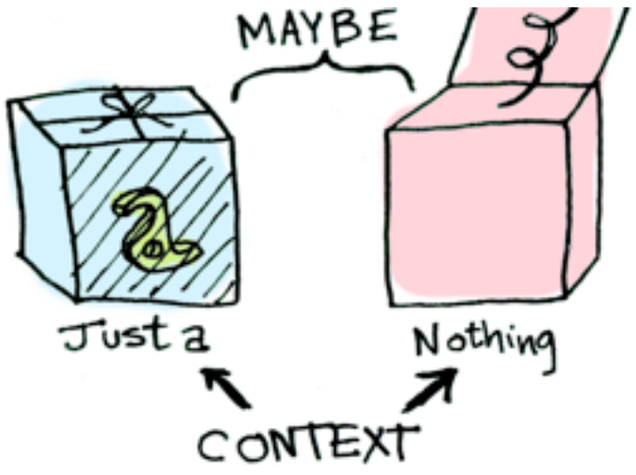
ReactiveCocoa

```
RAC(self, avatarURL) = [[RACObserve(self, username)
    map:^(NSString *username) {
        return [[OCTUser mrc_fetchUserWithRawLogin:username] avatarURL];
    }]
    distinctUntilChanged];

self.validLoginSignal = [[RACSignal
  combineLatest:@[ RACObserve(self, username), RACObserve(self, password) ]
    reduce:^(NSString *username, NSString *password) {
      return @(username.length > 0 && password.length > 0);
    }]
    distinctUntilChanged];
```

函数式的核心思想FUNCTORS, APPLICATIVE FUNCTORS 与 MONADS

MAYBE



Maybe 类型封装了一个可选值。一个 Maybe a 类型的值要么包含一个 a 类型的值（用 Just a 表示）；要么为空（用 Nothing 表示）。我们可以把 Maybe 看作一个盒子，这个盒子里面可能装着一个 a 类型的值，即 Just a；也可能是一个空盒子，即 Nothing

data Maybe a = Nothing | Just a

FUNCTORS

在 Functor typeclass 中定义了一个函数 fmap ， 它将一个函数 (a -> b) 应用到一个在上下文中的值 f a ， 并返回另一个在相同上下文中的值 f b ， 这里的 f 是一个类型占位符，表示任意类型的 Functor 。

fmap (+3) (Just 2)

Just 5

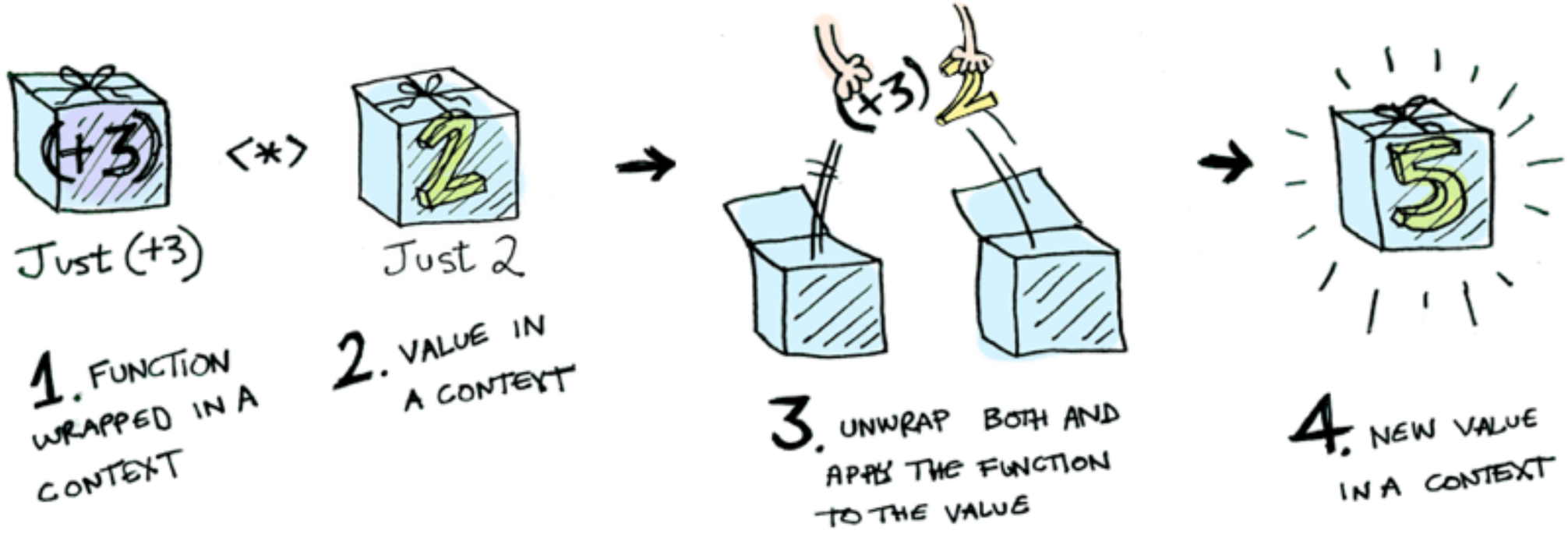
instance Functor Maybe where

fmap func (Just x) = Just (func x)

fmap func Nothing = Nothing

APPLICATIVE

```
class Functor f => Applicative f where
    pure :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```



Just (+3) <*> Just 2 == Just 5

MONADS

> Just 20 >>= half >>= half >>= half

Nothing

现在，我们已经知道 Monad 是什么了，它就是一种实现了 Monad typeclass 的数据类型。那么它有什么具体的应用呢？你总不能让我们都来做理论研究吧。既然如此，那我们就只好祭出 Objective-C 中的神器，ReactiveCocoa ，它就是根据 Monad 的概念搭建起来的。

ReactiveCocoa 是一个 iOS 中的函数式响应式编程框架，它受 Functional Reactive Programming 的启发，是 Justin Spahr-Summers 和 Josh Abernathy 在开发 GitHub for Mac 过程中的一个副产品，它提供了一系列用来组合和转换值流的 API 。

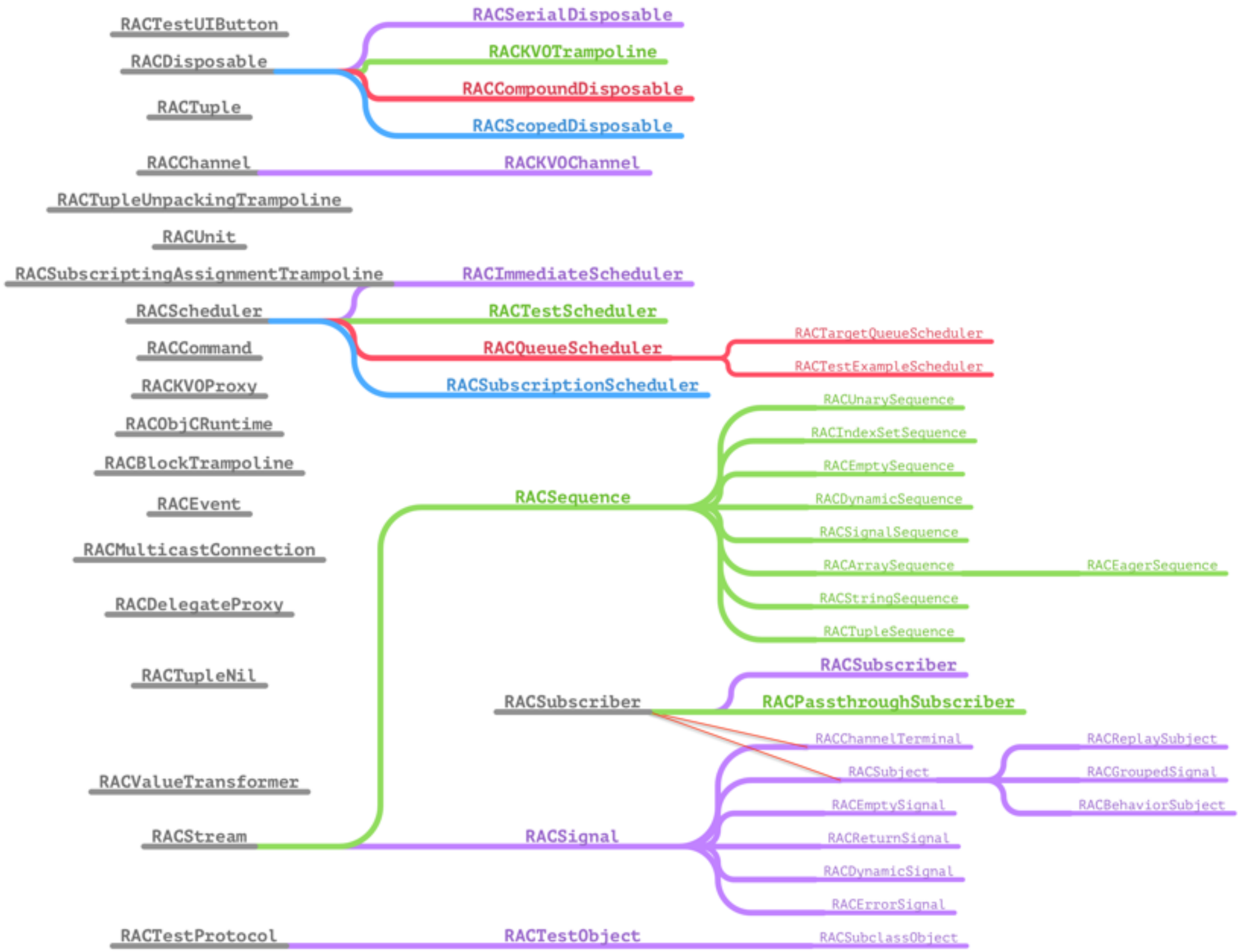
> Breaking from a tradition of covering Apple APIs exclusively, this edition of NSHipster will look at an open source project that exemplifies this brave new era for Objective-C.。
> Johnny Appleseed

Mattt Thompson

https://github.com/mattt

他认为 ReactiveCocoa 打破了苹果 API 排他性的束缚，勇敢地开创了 Objective-C 的新纪元，具有划时代的意义。不得不说，这对于一个第三方框架来说，已经是非常高的评价了。

ReactiveCocoa 是一个非常复杂的框架，在正式开始介绍它的核心组件前，我们先来看看它的类图，以便从宏观上了解它的层次结构：



信号源：RACStream 及其子类；

订阅者：RACSubscriber 的实现类及其子类；

调度器：RACScheduler 及其子类；

清洁工：RACDisposable 及其子类。

Delegate

```
- (void) didUpdateUsername:(NSString *)username;
```

Block Callbacks

```
- (void) updateUsernameWithCompletionHandler: (void (^)(id result))completionHandler;
```

Target Action

```
[textfield addTarget:self
              action:@selector(textDidChange:)
    orControlEvents:UIControlEventValueChanged];
```

KVO

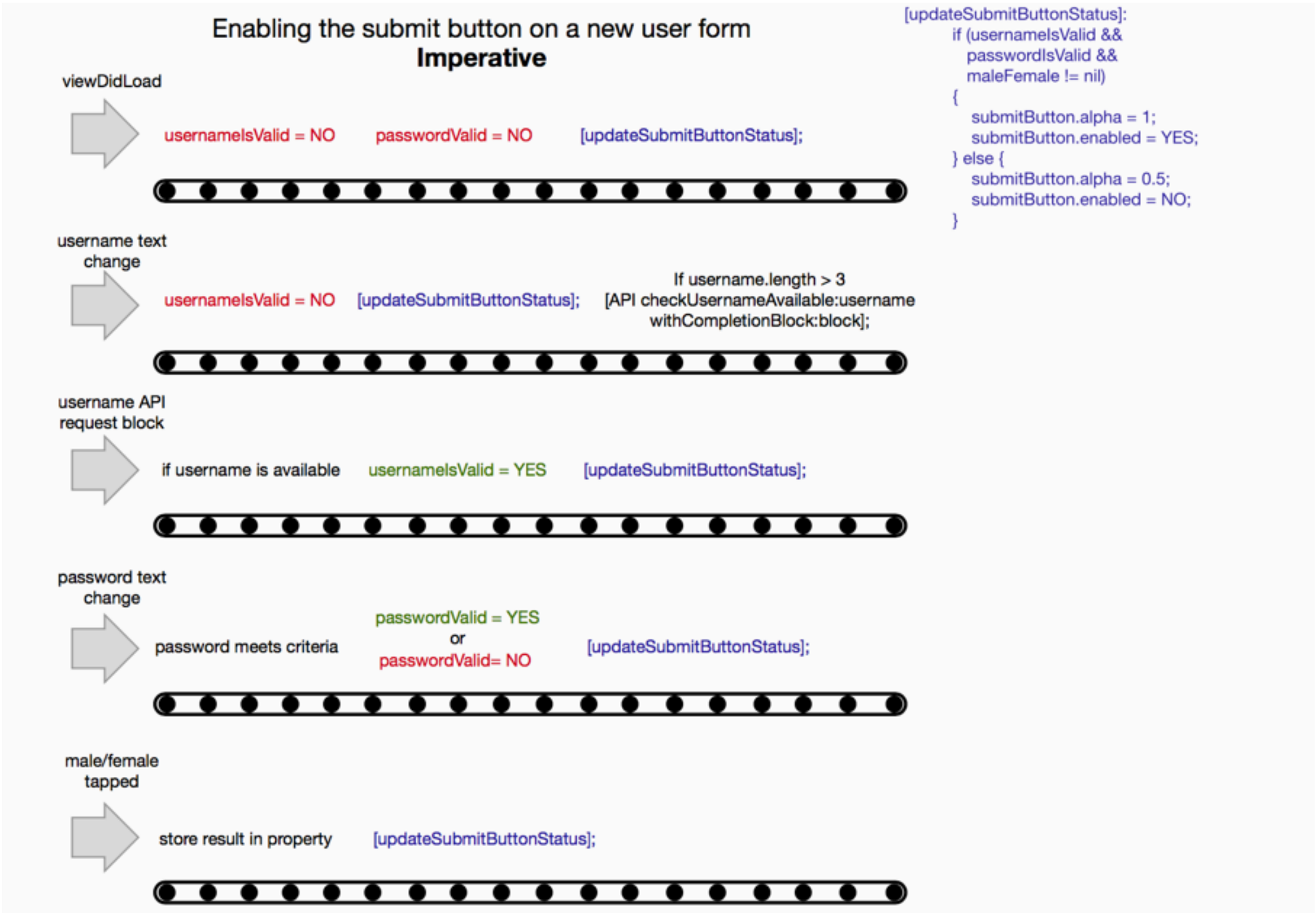
```
[viewModel addObserver:self
            forKeyPath:@“userFullName”
               options:NSKeyValueObservingOptionInitial context:context];
```

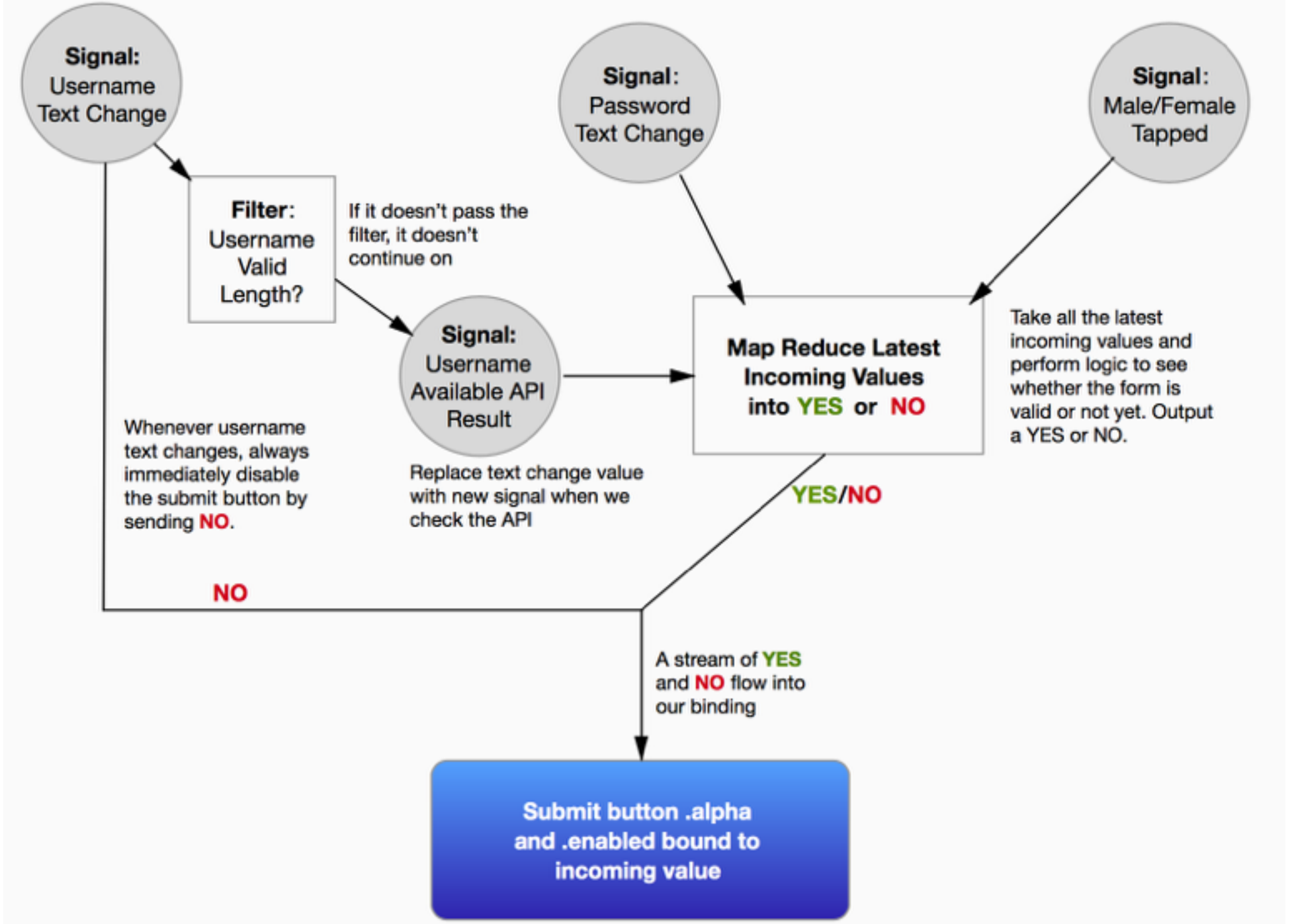
Notifications

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(usernameIsValidChanged:)
                                             name:@“UsernameIsValidChanged”
                                           object:nil];
```

We can replace all these async information flow control tools with:

RACSignal


Enabling the submit button on a new user form
Imperative
viewDidLoad
usernameIsValid = NO
passwordValid = NO
[updateSubmitButtonStatus];
username text change
usernameIsValid = NO
[updateSubmitButtonStatus];
If username.length > 3
[API checkUsernameAvailable:username withCompletionBlock:block];
username API request block
if username is available
usernameIsValid = YES
[updateSubmitButtonStatus];
password text change
password meets criteria
passwordValid = YES
or
passwordValid= NO
[updateSubmitButtonStatus];
male/female tapped
store result in property
[updateSubmitButtonStatus];
[updateSubmitButtonStatus]:
if (usernameIsValid &&
passwordIsValid &&
maleFemale != nil)
{
submitButton.alpha = 1;
submitButton.enabled = YES;
} else {
submitButton.alpha = 0.5;
submitButton.enabled = NO;
}




```
RACSignal *usernameIsValidSignal = RACObserve(self.viewModel, isUsernameValid);
    RAC(self.goButton, enabled) = usernameIsValidSignal;
    RAC(self.goButton, alpha) = [usernameIsValidSignal
        map:^id(NSNumber *usernameIsValid) {
            return usernameIsValid.boolValue ? @1.0 : @0.5;
        }];
```

MVC

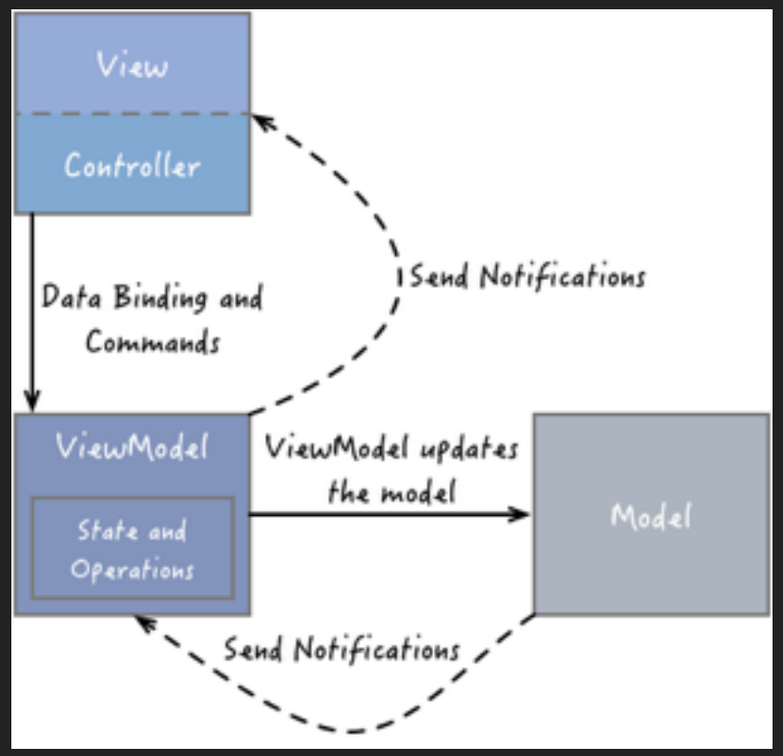
MVC，全称是 Model View Controller，是模型 (model)－视图 (view)－控制器 (controller) 的缩写。它表示的是一种常见的客户端软件开发框架。

现在，MVC 已经成为主流的客户端编程框架，在 iOS 开发中，系统为我们实现好了公共的视图类：UIView，和控制器类：UIViewController

但是，几十年过去了，我们对于 MVC 这种设计模式真的用得好吗？其实不是的，MVC 这种分层方式虽然清楚，但是如果使用不当，很可能让大量代码都集中在 Controller 之中，让 MVC 模式变成了 Massive View Controller 模式。

坦白说，有一部分逻辑确实是属于 controller 的，但是也有一部分逻辑是不应该被放置在 controller 中的。比如，将 model 中的 NSDate 转换成 view 可以展示的 NSString 等。在 MVVM 中，我们将这些逻辑统称为展示逻辑

- 因此，一种可以很好地解决 Massive View Controller 问题的办法就是将 controller 中的展示逻辑抽取出来，放置到一个专门的地方，而这个地方就是 viewModel 。其实，我们只要在上图中的 M-VC 之间放入 VM ，就可以得到 MVVM 模式的结构图



每个层之间彼此解耦，从而有以下特点：

每个View/ViewModel 可以彼此交换

内部代码实现的改变不会影响外部

每个层可以独立运作

独立的单元测试

1.View

- 轻量级的视图，包含布局，动画，提供控件动画的操作方式，以及UI相关的事件。不包含业务逻辑

2.Binder

- 负责将UI的事件针对特定ViewModel的Command 进行Binding，同时提供ViewModel的数据源于View层的单向/双向绑定。并不包含任何逻辑进提供View 于 ViewModel之间对应的Binding关系

3.ViewModel

- 实现相应的Command 业务逻辑，监听Model层的数据流包装，透过Binder通知View变更。不包含，不引用任何View。ViewModel层具有独立可测试性不依赖视图层存在

4.Model

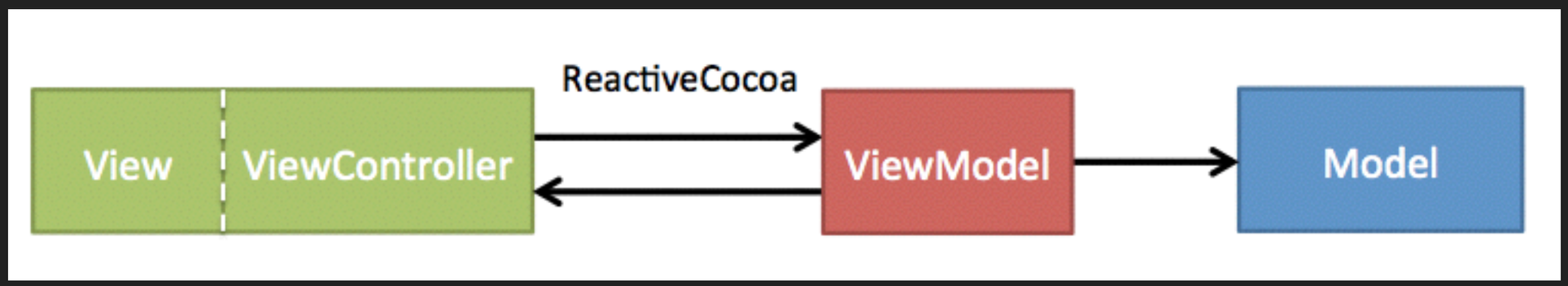
- 模型是指任何类型模型，它代表实际数据 ，或对数据访问中间层。同样模型层具有独立可测试性，其不依赖ViewModel而存在

- The Benefits of MVVM
- MVVM enables a great developer-designer workflow, providing these benefits:
- During the development process, developers and designers can work more independently and concurrently on their components. The designers can concentrate on the view, and if they are using Expression Blend, they can easily generate sample data to work with, while the developers can work on the view model and model components.
- The developers can create unit tests for the view model and the model without using the view. The unit tests for the view model can exercise exactly the same functionality as used by the view.
- It is easy to redesign the UI of the application without touching the code because the view is implemented entirely in XAML. A new version of the view should work with the existing view model.
- If there is an existing implementation of the model that encapsulates existing business logic, it may be difficult or risky to change. In this scenario, the view model acts as an adapter for the model classes and enables you to avoid making any major changes to the model code.

来源：https://msdn.microsoft.com/en-us/library/hh848246.aspx

- 尽管，在 iOS 开发中，系统并没有提供类似的框架可以让我们方便地实现 Binder 功能，不过，值得庆幸的是，GitHub 开源的 RAC ，给了我们一个非常不错的选择。

- 在 iOS 的 MVVM 实现中，我们可以使用 RAC 来在 View 和 ViewModel 之间充当 Binder 的角色，优雅地实现两者之间的同步。此外，我们还可以把 RAC 用在 Model 层，使用 Signal 来代表异步的数据获取操作，比如读取文件、访问数据库和网络请求等。说明，RAC 的后一个应用场景是与 MVVM 无关的，也就是说，我们同样可以在 MVC 的 Model 层这么用



Talk is cheap, show me the code

LINUS TORVALDS