

硕 士 研 究 生 读 书 报 告



题目 函数响应式编程之ReactiveCocoa框架

作者姓名 郭志强 ＿

作者学号 21651160

指导教师 李启雷

学科专业 移动互联网和游戏开发技术 ＿

所在学院 软件学院 ＿

提交日期 二○一六 年十二月 ＿

The ReactiveCocoa Framework of Functional Reactive Programming

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Li

By

Zhiqiang Guo

Zhejiang University, P.R. China

2016

摘要

本文以iOS下的开源框架ReactiveCocoa（RAC）为起点，研究了响应式编程（RP）以及其变体函数响应式编程（FRP）。RAC基于函数响应式编程，它将传统响应事件的方式统一，为事件定义了标准接口，这使得更容易对事件流进行过滤、连接和组合。在传统的iOS开发中，都是通过action、delegate、KVO以及callback等方式来进行事件的响应，但这些方式都会让开发者过多陷入实现细节，大量的状态变量、额外的处理过程都使代码结构复杂化。RAC的主旨是让代码更简洁易懂，逻辑可以用清晰的管道、流式语法来表示，开发人员不用过多陷入实现细节而更注重业务逻辑，使得应用更容易理解。RAC试图解决三个问题：传统iOS开发过程中，状态以及状态之间依赖过多的问题；传统MVC架构的问题：Controller比较复杂，可测试性差；提供统一的消息传递机制。理解RAC的关键是理解响应式编程，在理解RP的过程中，最困难的是以RP的方式思考，意味着要放弃命令式且带状态的(Imperative and stateful)编程习惯，并且要强迫大脑以一种不同的范式来工作。如果能熟练地运用RAC，这会让iOS开发变得更快捷，因为在开发过程中更注重的是业务逻辑，业务逻辑能很简单地转化为RAC方式下的代码。特别是开发高互动性的Webapps、Mobile apps时，运用RAC框架的优势更加显著，这些应用包含大量与数据事件相关的UI事件，而RAC能轻松地用极少的代码实现复杂的UI交互。

**关键词**：iOS，ReactiveCocoa，响应式编程, FRP, RP

Abstract

Based on ReactiveCocoa (RAC) which is an open source framework in iOS, this paper studies Reactive Programming (RP) and its variant Functional Reactive Programming (FRP). RAC bases on Functional Reactive Programming that unifies the traditional way of reacting to events and defines a standard interface for events, making it easier to filter, connect, and combine event streams. In the traditional iOS development, through the action, delegate, KVO and callback and other ways to react to events. But these methods will force developers to fall into too many implementation details, a large number of state variables and additional processing procedure make the code to complex. RAC's main purpose is to make the code more simple and easy to understand, and logic can be expressed in clear pipelines and streaming syntax. Developers do not have to fall into the implementation details too much, but focus on business logic which makes the application easier to understand. RAC attempts to solve three problems: the state and too much dependence between the states problems in the traditional iOS development process; traditional MVC architecture problems: Controller is much complex and testability is poor problems; provide a unified messaging mechanism. The key to understanding RAC is to understand Reactive Programming. The most difficult way to understand RP is to think in terms of RP, which means giving up the imperative and stateful programming habit and forcing the brain to a different paradigm to work. If you can skillfully use RAC, which will make iOS development faster, because in the development process is more focused on business logic, business logic can be easily converted to RAC mode code. Especially when developing highly interactive Webapps and Mobile apps, the advantages of using the RAC framework are more remarkable. These applications contain a large number of UI events related to data events, and RAC can easily implement complex UI interaction with very little code.

**Keywords：**iOS, ReactiveCocoa, Reactive Programming, FRP, RP

**目录**

[1 响应式编程概念 6](#_Toc471225650)

[1.1 什么是RP 6](#_Toc471225651)

[1.2 RP是使用异步数据流进行编程 6](#_Toc471225652)

[1.3 点击事件流 7](#_Toc471225653)

[1.4 简单的计数器数据流 7](#_Toc471225654)

[1.5 获取双击事件数据流 8](#_Toc471225655)

[1.6 RP的优点 9](#_Toc471225656)

[2 函数响应式编程框架ReactiveCocoa 10](#_Toc471225657)

[2.1 概念 10](#_Toc471225658)

[2.2 RAC的优点 10](#_Toc471225659)

[2.2.1 试图解决的问题 10](#_Toc471225660)

[2.2.2 传统iOS开发过程中，状态以及状态之间依赖过多的问题 10](#_Toc471225661)

[2.2.3 试图解决MVC框架的问题 11](#_Toc471225662)

[2.2.4 统一消息传递机制 11](#_Toc471225663)

[2.3 ReactiveCocoa类关系图 12](#_Toc471225664)

[2.4 重点类解释 12](#_Toc471225665)

[2.5 工作原理 15](#_Toc471225666)

[2.6 重要API示例 15](#_Toc471225667)

[2.6.1 Signal 15](#_Toc471225668)

[2.6.2 RACSubscriber 16](#_Toc471225669)

[2.6.3 RACDisposable 16](#_Toc471225670)

[2.6.4 RACSubject 16](#_Toc471225671)

[2.6.5 RACReplaySubject 16](#_Toc471225672)

[2.6.6 RACTuple、RACSequence 16](#_Toc471225673)

[2.6.7 RACCommand 16](#_Toc471225674)

[2.6.8 RACMulticastConnection 16](#_Toc471225675)

[2.6.9 RACScheduler 16](#_Toc471225676)

[2.6.10 RACUnit 16](#_Toc471225677)

[2.6.11 RACEvent 17](#_Toc471225678)

[2.7 常用方法 17](#_Toc471225679)

[3 小结 18](#_Toc471225680)

[参考文献 19](#_Toc471225681)

[附录 20](#_Toc471225682)

[代码-1 20](#_Toc471225683)

[代码-2 21](#_Toc471225684)

[代码-3 21](#_Toc471225685)

[代码-4 21](#_Toc471225686)

[代码-5 22](#_Toc471225687)

[代码-6 23](#_Toc471225688)

[代码-7 24](#_Toc471225689)

# 1 响应式编程概念

## 1.1 什么是RP

RP（Reactive Programming）即响应式编程，在理解RP的过程中，最困难的是以RP的方式思考，意味着要放弃命令式且带状态的(Imperative and stateful)编程习惯，并且要强迫你的大脑以一种不同的范式来工作。

网络上有一大堆对新手难以理解的解释和定义，如Wikipedia上通常都是些非常笼统和理论性的解释，而[Stackoverflow](http://stackoverflow.com/questions/1028250/what-is-functional-reactive-programming)上的一些规范的回答显然也不适合新手来参考，[Reactive Manifesto](http://www.reactivemanifesto.org/)看起来也只像是拿给你的PM或者老板看的东西，微软的[Rx术语](https://rx.codeplex.com/)"Rx = Observables + LINQ + Schedulers" 也显得太过沉重，而且充满了太多微软式的东西，反而给我们带来更多疑惑。"Reactive"和"Propagation of change"这样的术语并没有传达任何有意义的概念。

1.2 RP是使用异步数据流进行编程

RP是一种编程范式，它与异步数据流进行交互[1]。事件总线（Event buses）或者点击事件（Click Events）等本质上就是异步事件流（Asynchronous event stream），它们可以被监听处理，可以观察它们的变化做出一些反应（do some side effects）。RP的大致思路是：除了点击（Click）和悬停（Hover）等事件，其它任何事物都可以被创建数据流（Data stream）。变量、用户输入、属性、cache、数据结构等都可以被创建为一个stream。

图 1-1 “Click Button” Event Stream

在创建出数据流后，还可以用函数取处理，比如结合（combine）、创建（create）、过滤（filter）等函数。这是函数式编程，它与响应式编程结合形成一个变体—FRP（函数响应式编程）。

数据流也可以作为另一个数据流的输入，每个数据流可以接受一个或多个数据流输入；还可以合并（merge）两个数据流；从一个数据流过滤（filter）出感兴趣的事件生成新的数据流；也可以将一个数据流的值映射（map）到新的数据流中。

1.3 点击事件流

数据流是RP的核心[2]，这里从“点击一个按钮”事件流来进行简单介绍。数据流是一个按时间排序的即将发生事件的序列（Ongoing events ordered in time），如图1-1，它可以发出（emit）三种不同的事件：某种类型的值（value）事件、错误（error）事件和完成（completed）事件。

捕获这三个事件是以异步的方式，分别为三个事件定义事件处理函数，当事件被发出便执行相应的处理函数，一般情况只需要定义处理值事件的处理函数。如何知道哪个事件被发出了，需要去监听数据流，这也被成为订阅（Subscribing）。那么定义的事件处理函数就是观察者（Observer），被订阅的数据流就是被观察者（Observable），这其实就是观察者模式（Observer Design Pattern）。

图1-1也可以用ASCII码的方式表示：

**--a---b-c---d---X---|->**

**a, b, c, d are emitted values**

**X is an error**

**| is the 'completed' signal**

**---> is the timeline**

1.4 简单的计数器数据流

接下里是一个计数器数据流的实现，它能记录一个按钮点击次数。一般的RP库中，每个数据流都包含map、filter、scan等方法。比如clickStream.map(f)，会在原clickStream的基础上返回新的数据流，不会去修改原数据流，即不可变性（immutability）。

这里进行了链式调用clickStream.map(f).scan(g)，用ASCII表示：

**clickStream: ---c----c--c----c------c-->**

**vvvvv map(c becomes 1) vvvv**

**---1----1--1----1------1-->**

**vvvvvvvvv scan(+) vvvvvvvvv**

**counterStream: ---1----2--3----4------5-->**

map(f)根据f函数把原数据流中的值映射到新数据流，这里是把每次点击全部映射为1。scan(g)根据g函数把数据流中的值聚合成新的值，new\_value = g(accumlated,current)，这里的g函数是一个累加函数。通过这两次数据流转换，最后的counterStream会在每次点击后将值发送给观察者，观察者实际得到的就是点击的总次数。

1.5 获取双击事件数据流

当需要获取包含双击事件的数据流，或许还需要考虑三击（Triple clicks），甚至是连击（Mutiple clicks）。如果用传统的命令式带状态的方式去实现，代码必定是混乱难以理解的，一定有一堆变量来保存状态，还有一堆计算时间间隔的代码。

在RP中却只需要4行代码就能实现这个功能，首先用图的方式表示这个实现过程。如图1-2，灰色的方框是用来转换数据流的函数。



图1-2 获取双击事件数据流过程

首先将点击事件分类放进列表（list），而这个分类的依据就是250ms内无事件发生（event silence，无事件时段，上一个发生事件到下一个发生事件间的时段），通过buffer(stream.throttle(250ms))将点击事件隔开，然后将每一段的一个或多个点击事件添加进列表。这一步得到图1-2中第二条数据流，其中有4个列表，每个列表间隔大于等于250ms。然后使用map('get length of list')将列表映射为整数，即每个列表的长度。最后使用filter(x >= 2)将整数过滤。

通过上述步骤，生成了需要的数据流，然后可以订阅（监听）这个数据流做出相应的反应。从这个小功能体现出RP的强大之处，它简洁、逻辑清晰，没有一堆状态变量，也没有复杂的计算代码。

1.6 RP的优点

RP提高了代码的抽象层级，在开发中只需要关注业务逻辑中那些相互依赖的事件，不用纠结于大量的实现细节，这使得RP的代码逻辑更加的清晰、更容易理解。

在开发高互动性的Webapps、Mobile apps时，RP优势更加显著，这些应用包含大量与数据事件相关的UI事件。以前的网页交互很简单，只需要在用户填完一个很长的表单后，将其内容提交给后端，在填写表单过程中交互很少。而现在的网页、app都需要大量的交互，最基本就是用户名、密码，这些东西需要实时地反应给用户。这意味着现在的应用存在大量的交互事件，如果使用命令式带状态的方式去处理，要处理太多实现细节、太多状态变量，而用RP的方式显然更加清晰、简单。

# 2 函数响应式编程框架ReactiveCocoa

2.1 概念

利用函数式编程（Functional Programming）的思想和方法（函数、高阶函数）来支持RP就是函数响应式编程（FRP，Functional Reactive Programming），而iOS开发中常用的框架ReactiveCocoa便支持FRP。

ReactiveCocoa (RAC)是一个Objective-C的框架,由Github开源的一个应用于iOS和OS开发的新框架，它的灵感来自函数响应式编程（FRP）。RAC提供了单一且统一的方法来处理异步行为，包括delegate方法、block回调、target-action机制、notification和KVO

RAC常用于处理异步事件、驱动数据源，作为一个iOS开发者，每一行代码几乎都是在响应某个事件，例如按钮的点击、收到网络消息、属性的变化或者用户位置的变化（通过CoreLocation）。但是这些事件都用不同的方式来处理，比如action、delegate、KVO、callback等。RAC为事件定义了一个标准接口（信号signal，用RACSignal类表示），从而可以使用一些基本工具来更容易的连接、过滤和组合。

RAC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，这样非常方便管理，就不需要跳到对应的方法里。在处理时不需要考虑调用顺序，直接考虑结果，把每一次操作都写到一系列嵌套的方法中，使代码高聚合方便管理。

RAC的主旨是让代码更简洁易懂，逻辑可以用清晰的管道、流式语法来表示，很容易理解这个应用到底干了什么。

2.2 RAC的优点

### 2.2.1 试图解决的问题

RAC试图解决：传统iOS开发过程中，状态以及状态之间依赖过多的问题；传统MVC架构的问题：Controller比较复杂，可测试性差；提供统一的消息传递机制。

### 2.2.2 传统iOS开发过程中，状态以及状态之间依赖过多的问题

开发过程中，一个界面元素的状态很可能受多个其它界面元素或后台状态的影响。例如，在用户帐户的登录界面，通常会有2个输入框（分别输入帐号和密码）和一个登录按钮。如果要加入一个限制条件：当用户输入完帐号和密码，并且登录的网络请求还未发出时，确定按钮才可以点击。通常情况下，需要监听这两个输入框的状态变化以及登录的网络请求状态，然后修改另一个控件的enabled状态。

* 传统的写法见附录1中[代码-1](#_代码-1)

RAC通过引入信号（Signal）的概念，来代替传统iOS开发中对于控件状态变化检查的target-action模式或代理模式（delegate）。因为RAC的信号是可以组合（combine）的，所以可以轻松地构造出另一个新的信号出来，然后将按钮的enabled状态与新的信号绑定。

* 代码见附录1中[代码-2](#_代码-2)

可以看到，在引入RAC之后，以前散落在action-target或KVO的回调函数中的判断逻辑被统一到了一起，从而使得登录按钮的enabled状态被更加清晰地表达了出来。

除了组合（combine）之外，RAC的信号还支持链式（chaining）和过滤（filter)，以方便将信号进行进一步处理。

### 2.2.3 试图解决MVC框架的问题

对于传统的Model-View-Controller的框架，Controller很容易变得比较庞大和复杂。由于Controller承担了Model和View之间的桥梁作用，所以Controller常常与对应的View和Model的耦合度非常高，这同时也造成对其做单元测试非常不容易，对iOS工程的单元测试大多都只在一些工具类或与界面无关的逻辑类中进行。

RAC的信号机制很容易将某一个Model变量的变化与界面关联，所以非常容易应用Model-View-ViewModel 框架。通过引入ViewModel层，然后用RAC将ViewModel与View关联，View层的变化可以直接响应ViewModel层的变化，这使得Controller变得更加简单，由于View不再与Model绑定，也增加了View的可重用性。

因为引入了ViewModel层，所以单元测试可以在ViewModel层进行，iOS工程的可测试性也大大增强了。

### 2.2.4 统一消息传递机制

iOS开发中有着各种消息传递机制，包括KVO、Notification、delegation、block以及target-action方式，各种消息传递机制使得开发者在做具体选择时感到困惑。RAC将传统的UI控件事件进行了封装，使得以上各种消息传递机制都可以用RAC来完成。

* 示例代码见附录1中[代码-3](#_代码-3)

2.3 ReactiveCocoa类关系图

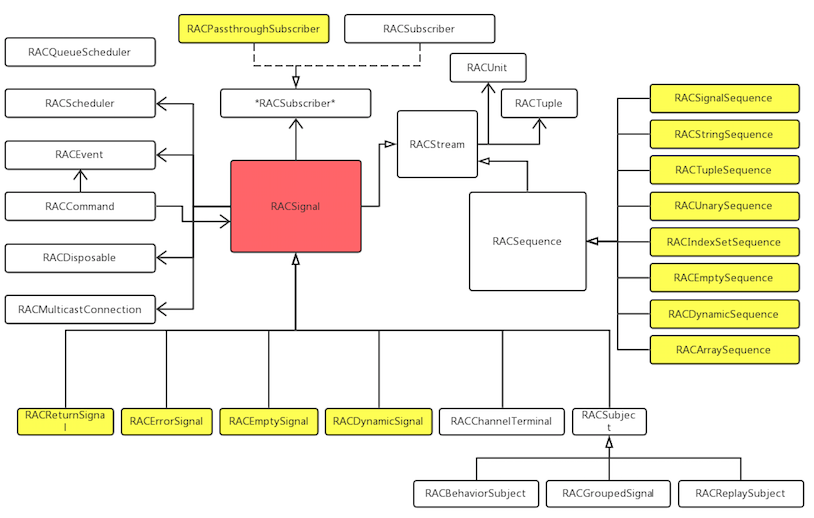
图2-1 ReactiveCocoa类图

图2-1中红色类RACSignal是最重要的类，而黄色类是私有类。

2.4 重点类解释

* Streams—RACStream

streams代表任意的值，其值会随着事件发生变化，由RACStream类表示。值可能马上可用，或者在将来某一段时间可用，但必须按顺序获取，也就是说在获取到第一个值之前，是不可能获取到第二个值的。

streams 是一个构造因果关系的结构（ [Monad](http://www.apple.com.cn)），它可以实现在基本的初始值上进行复杂的操作运算（ filter， map， reduce等）。

streams不会被经常使用，大多情况下表现为signal和sequences，即Signal和Sequence是由stream继承。

* Signals—RACSignal

Signals由RACSignal类表示，Signals一般表示将来被传递的数据。当信号接收到数据时，值就会通过signal被发送出去，它推送数据给订阅者，用户必须订阅信号才能获取到它的值。

Signals发送3种不同类型的事件给它的订阅者：

1.next：next事件是stream提供了一个新值。而RACStream类的方法也只能在这个值上进行操作运算。和cocoa的集合不同的是，它可以包含一个nil值。

2.error：error事件表示在信号完成之前发生了错误。这个事件包含了一个NSError类型的错误，这个值不会在RACStream类里存储。

3.completed：表示信号已经成功地完成了。这个值也不会在RACStream类里存储。

* Subscription—RACSubscriber

subscriber（订阅者）表示等待或者能够等待信号发送事件的任意对象。在框架中使RACSubscriber 协议表示，也即任意实现了RACSubscriber协议的对象都可以是订阅者。

可以通过调用 -subscribeNext: error: completed：方法来创建订阅， RACStream 和 RACSignal类的大多操作也会自己创建订阅。

订阅会对Signals对象引用计数加1，当信号发送错误或者完成事件后，会自动被处理，不需要用户关心内存管理。当然，用户也可以手动处理。

* Subjects—RACSubject

subject由RACSubject类表示，它是可以手动控制的信号。subject可以想成是signal的变体，就像NSMutableArray相对于NSArray一样。它们是非RAC的代码和RAC代码之间的桥梁，因此，非常有用。

有些subject提供了额外的功能，特别地， RACReplaySubject用于缓存事件，将来有订阅者时，可以将缓存的数据发送给订阅者，比如，当请求一个网络时，服务器数据已经返回，但其它的数据还没有准备好，这时，先要将网络请求结果缓存，等其它数据准备好时，再订阅，而不至于网络请求数据丢失。

* Commands—RACCommand

command由RACCommand 类表示， 它创建并订阅响应action的信号。通常command是由UI触发，比如一个按钮被点击时。当command被触发时，控件会自动被禁用。

* Connections—RACMulticastConnection

connection由RACMulticastConnection类表示。它是在任意数量的订阅者之间共享订阅。默认情况下，信号没有订阅时，是冷信号；当有订阅者被添加进来后，信号才会开始工作，变为热信号。

对于每个订阅，数据值都会被延迟重新计算。否则，如果信号有副作用或者任务开销很大时，可能就会产生问题。

connection通过RACSignal的publish或者multicast方法创建，一旦连接，信号就变为热信息，订阅会保持激活状态直到所有的订阅连接被取消。

* Sequences—RACSequence

sequence由RACSequence 类表示。Sequence是一种集合，类似于NSArray。和数组不同，为了改进性能，在sequence里的值会延迟计算，只有需要输出值时，结果才会被计算。 sequences不能包含nil值。

对于大多数据的集合类， RAC添加了-rac\_sequence方法来生成RACSequence类进行操作。

* Disposables—RACDisposable

disposable由RACDisposable 类表示，用于取消订阅或者清理资源。Disposable

大多情况下用于取消信号的订阅 。

* Schedulers—RACScheduler

scheduler由RACScheduler类表示，它是信号执行任务时所在的队列（ queue）或者信号执行完后将结果放到队列里执行，可以认为就是gcd里的queues。

Scheduler支持取消操作，而且它总是串行地执行任务，这有利于避免死锁。RACScheduler有时候也类似NSOperationQueue，但它不允许任务间相互依赖。

* Value types

RAC提供了以下类用于表示各种值：

1.RACTuple： 一个比较少数量值，大小固定的，可以包含nil的集合。一般用于表示多个streams的聚合值。

2.RACUnit ： 一个单例的空值，表示stream不包含有意义的值。

3.RACEvent ： 代表任意的信号事件。

* Asynchronous Backtraces

由于基于RAC的代码经常和异步相关，因此，为了更方便调试， RAC支持捕获当前的asynchronous backtraces（异步调用栈）。

在OS X系统上，支持捕获所有代码，包括系统库。而iOS，只支持RAC及工程内的代码。

2.5 工作原理

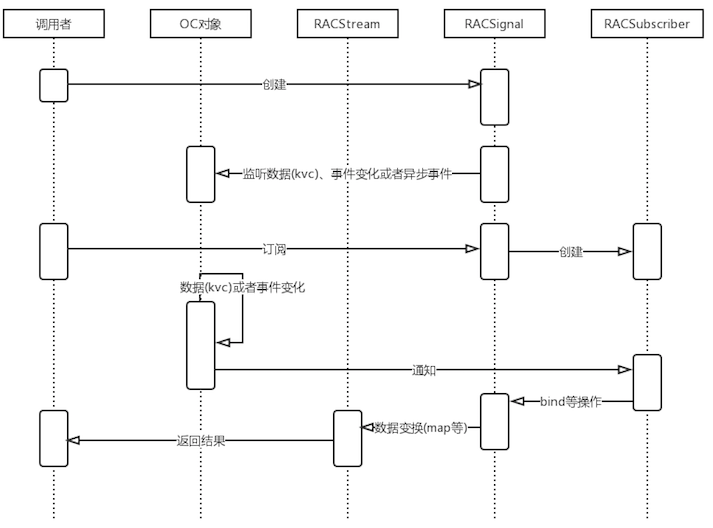


图2-2 工作原理

2.6 重要API示例

### 2.6.1 Signal

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。

注意：

1.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。

2.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。

3.如何订阅信号：调用信号RACSignal的subscribeNext就能订阅

* 示例代码见附录1中[代码-4](#_代码-4)

### 2.6.2 RACSubscriber

表示订阅者的意思，用于发送信号，这是一个协议，不是一个类，只要遵守这个协议，并且实现方法才能成为订阅者。通过create创建的信号，都有一个订阅者，帮助他发送数据。

### 2.6.3 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。使用场景：不想监听某个信号时，可以通过它主动取消订阅信号。

### 2.6.4 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。

### 2.6.5 RACReplaySubject

重复提供信号类，RACSubject的子类。RACReplaySubject与RACSubject区别: RACReplaySubject可以先发送信号，再订阅信号，RACSubject就不可以。

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。

* RACSubject和RACReplaySubject简单使用见附录1中[代码-5](#_代码-5)
* RACSubject替换代理见附录1中[代码-6](#_代码-6)

### 2.6.6 RACTuple、RACSequence

RACTuple：元组类，类似NSArray，用来包装值。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。

### 2.6.7 RACCommand

RAC中用于处理事件的类，可以把事件处理、事件中的数据传递过程包装到这个类中，他可以很方便的监控事件的执行过程。

### 2.6.8 RACMulticastConnection

在创建信号时，为避免多次调用创建信号中的block而造成副作用，可以使用这个类解决。使用注意：RACMulticastConnection通过RACSignal的-publish或者-muticast:方法创建。

### 2.6.9 RACScheduler

RAC中的队列，用GCD封装的。

### 2.6.10 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。

### 2.6.11 RACEvent

把数据包装成信号事件(signal event)，它主要通过RACSignal的-materialize来使用。

2.7 常用方法

* 代替代理：rac\_signalForSelector；
* 代替KVO：rac\_valuesAndChangesForKeyPath，用于监听对象属性的改变；
* 监听事件：rac\_signalForControlEvents；
* 代替通知：rac\_addObserverForName；
* 监听文本框文字改变：rac\_textSignal，只要文本框发出改变就会发出这个信号；
* 合并多个signal：rac\_liftSelector:withSignalsFromArray:signals，多次请求获得数据后才进行操作。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。有几个信号，参数一方法就几个参数，每个参数对应信号发出的数据；
* 示例代码见附录1中[代码-7](#_代码-7)

# 3 小结

掌握ReactiveCocoa框架的核心是理解响应式编程，抛弃命令式且带状态的编程习惯，习惯使用异步数据流进行编程。在理解RP的基础上，以清晰地管道、流式语法来开发。RAC将传统的事件响应方式统一，提供了统一的接口，以signal的方式使得代码简洁易懂、逻辑清晰。

使用RAC不用花过多精力关注实现细节、状态变量，当对业务逻辑完成分析后可以轻松地转换为RAC框架下的代码，这让开发更加快捷。面对高交互性应用开发，RAC比传统开发更具优势，因为RAC能以很少的代码清晰地完成复杂交互。

熟练运用RAC会在目前的iOS开发中发挥很大作用，使用的过程中也会加深对响应式编程的理解，响应式编程是一个更加接近自然语言的编程方式，所以花精力去学习RAC框架肯定大有益处。

# 参考文献

[1] Czaplicki E. Elm: Concurrent FRP for Functional GUIs[J]. 2012.

[2] Czaplicki E, Chong S. Asynchronous functional reactive programming for GUIs[C]// ACM SIGPLAN Symposium on Programming Language Design & Implementation. ACM, 2013:411-422.

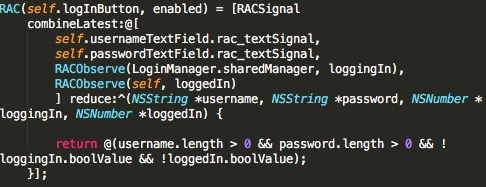
# 附录

附录1：代码示例

## 代码-1



## 代码-2



## 代码-3



## 代码-4



## 代码-5





## 代码-6



## 代码-7

