



作者 峥吖 (/users/b09c3959ab3b) 2015.09.25

23:23*



正在关注 (/users/b09c3959ab3b/toggle_like)

写了45950字, 被9517人关注, 获得了2787个喜欢

最快让你上手ReactiveCocoa之基础篇

字数5514 阅读83664 评论139 喜欢576

前言

很多blog都说ReactiveCocoa好用, 然后各种秀自己如何灵活运用ReactiveCocoa, 但是感觉真正缺少的是一篇如何学习ReactiveCocoa的文章, 小编看了很多篇都没看出怎么使用ReactiveCocoa, 于是决定自己写一遍关于学习ReactiveCocoa的文章, 本文主要针对如何从零开始学习ReactiveCocoa, 这里非常感谢3个人(支点的雷纯锋, camera360的宋潘, 以及我的小学弟何宗柱(我爱科技)), 在我研究ReactiveCocoa对我的帮助。

如果喜欢我的文章, 可以关注我, 也可以来小码哥 (<http://www.520it.com>), 了解下我们的iOS培训课程。之后还会更新《最快让你上手ReactiveCocoa之进阶篇》

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

- 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.编程思想

在开发中我们也不能太依赖于某个框架，否则这个框架不更新了，导致项目后期没办法维护，比如之前Facebook提供的 Three20框架，在当时也是神器，但是后来不更新了，也就没什么人用了。因此我感觉学习一个框架，还是有必要了解它的 编程思想 。

先简单介绍下目前咱们已知的 编程思想 。

3.1 面向过程：处理事情以过程为核心，一步一步的实现。

3.2 面向对象：万物皆对象

3.3 链式编程思想：是将多个操作（多行代码）通过点号(.)链接在一起成为一句代码,使代码可读性好。a(1).b(2).c(3)

- 链式编程特点：方法的返回值是block,block必须有返回值（本身对象），block参数（需要操作的值）
- 代表：masonry框架。
- 模仿masonry，写一个加法计算器，练习链式编程思想。

```
@class CaculatorMaker;  
@interface NSObject (Caculator)  
  
// 计算  
+ (int)makeCaculators:(void(^)(CaculatorMaker *make))caculatorMaker;  
  
@end
```

Snip20150925_2.png

```
@implementation NSObject (Calculate)  
  
+ (CGFloat)makeCalculate:(void (^)(CalculateManager *))block  
{  
    // 1.创建计算管理者  
    CalculateManager *mgr = [[CalculateManager alloc] init];  
  
    block(mgr);  
  
    return mgr.result;  
}  
@end
```

```
@interface CaculatorMaker : NSObject

@property (nonatomic, assign) int result;

// 加法
- (CaculatorMaker * (^)(int))add;

- (CaculatorMaker * (^)(int))sub;
- (CaculatorMaker * (^)(int))muilt;
- (CaculatorMaker * (^)(int))divide;

@end
```

```
@implementation CalculateManager

- (CalculateManager * (^)(int))add
{
    return ^CalculateManager *(int value){
        _result += value;
        return self;
    };
}

@end
```

```
int result = [NSObject makeCaculators:^(CaculatorMaker *make) {
    make.add(1).add(2).add(3).add(4).divide(5);
}];
```

3.4 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。

- 代表：KVO运用。

3.5 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）
- 代表：ReactiveCocoa。
- 用函数式编程实现，写一个加法计算器，并且加法计算器自带判断是否等于某个值。

```
@interface Caculator : NSObject
```

```
@property (nonatomic, assign) BOOL isEqule;
```

```
@property (nonatomic, assign) int result;
```

```
- (Caculator *)caculator:(int(^)(int result))caculator;
```

```
- (Caculator *)equle:(BOOL(^)(int result))operation;
```

```
@end
```

```

// 2 * 5 == 10
Caculator *c = [[Caculator alloc] init];

// 计算2 * 5 , 并且判断是否等于10
BOOL isqule = [[[c caculator:^(int result) {
    result += 2;
    result *= 5;
    return result;
}] equle:^(BOOL(int result) {

    return result == 10;

}] isEqule];

NSLog(@"%d", isqule);

```

Paste_Image.png

4.ReactiveCocoa编程思想

ReactiveCocoa结合了几种编程风格:

函数式编程 (Functional Programming)

响应式编程 (Reactive Programming)

所以, 你可能听说过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。

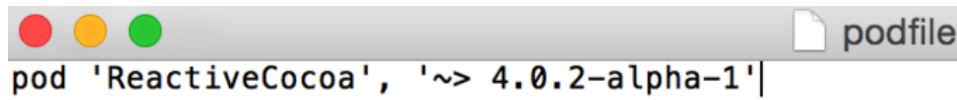
5.如何导入ReactiveCocoa框架

通常都会使用CocoaPods (用于管理第三方框架的插件) 帮助我们导入。

PS:CocoaPods教程 (<http://code4app.com/article/cocoapods-install-usage>)

注意：

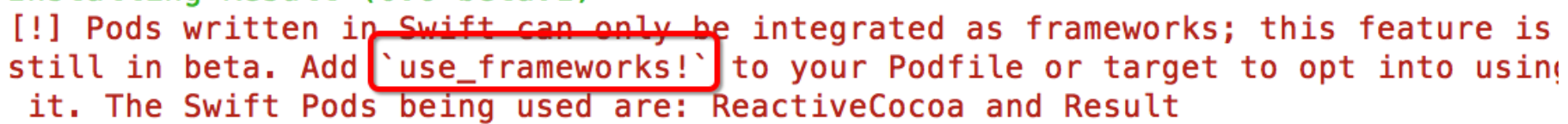
- podfile如果只描述pod 'ReactiveCocoa', '~> 4.0.2-alpha-1', 会导入不成功。



```
podfile
pod 'ReactiveCocoa', '~> 4.0.2-alpha-1'
```

Snip20150926_1.png

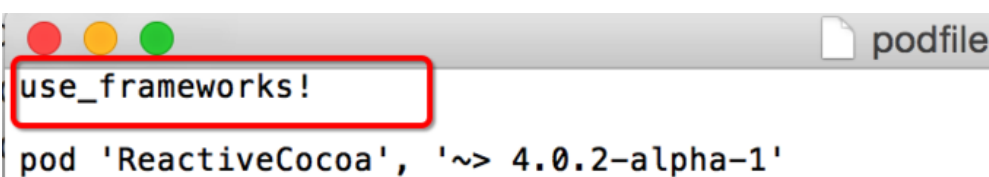
- 报错提示信息



```
[!] Pods written in Swift can only be integrated as frameworks; this feature is
still in beta. Add `use_frameworks!` to your Podfile or target to opt into using
it. The Swift Pods being used are: ReactiveCocoa and Result
```

Snip20150926_2.png

- 需要在podfile加上use_frameworks, 重新pod install 才能导入成功。



```
podfile
use_frameworks!
pod 'ReactiveCocoa', '~> 4.0.2-alpha-1'
```

Snip20150926_3.png

6.ReactiveCocoa常见类。

学习框架首要之处: 个人认为先要搞清楚框架中常用的类, 在RAC中最核心的类 RACSignal, 搞定这个类就能用ReactiveCocoa开发了。

6.1 RACSignal: 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:

- 信号类(RACSignal), 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。
- RACSignal简单使用：


```

// RACSignal使用步骤：
// 1.创建信号 + (RACSignal *)createSignal:(RACDisposable * (^)(id<RACSubscriber> subscribe
// 2.订阅信号,才会激活信号。 - (RACDisposable *)subscribeNext:(void (^)(id x))nextBlock
// 3.发送信号 - (void)sendNext:(id)value

// RACSignal底层实现：
// 1.创建信号，首先把didSubscribe保存到信号中，还不会触发。
// 2.当信号被订阅，也就是调用signal的subscribeNext:nextBlock
// 2.2 subscribeNext内部会创建订阅者subscriber，并且把nextBlock保存到subscriber中。
// 2.1 subscribeNext内部会调用signal的didSubscribe
// 3.signal的didSubscribe中调用[subscriber sendNext:@1];
// 3.1 sendNext底层其实就是执行subscriber的nextBlock

// 1.创建信号
RACSignal *signal = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscribe

    // block调用时刻：每当有订阅者订阅信号，就会调用block。

    // 2.发送信号
    [subscriber sendNext:@1];

    // 如果不在发送数据，最好发送信号完成，内部会自动调用[RACDisposable disposable]取消订阅信号。
    [subscriber sendCompleted];

    return [RACDisposable disposableWithBlock:^(

        // block调用时刻：当信号发送完成或者发送错误，就会自动执行这个block,取消订阅信号。

        // 执行完Block后，当前信号就不在被订阅了。

        NSLog(@"信号被销毁");

    }]];
}];

// 3.订阅信号,才会激活信号。
[signal subscribeNext:^(id x) {
    // block调用时刻：每当有信号发出数据，就会调用block。
    NSLog(@"接收到数据:%@", x);
}];

```

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。

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。

6.4 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。

- **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,即只缓存最新的几个值。
- **RACSubject**和**RACReplaySubject**简单使用:

```
// RACSubject使用步骤
// 1.创建信号 [RACSubject subject], 跟RACSignal不一样, 创建信号时没有block。
// 2.订阅信号 - (RACDisposable *)subscribeNext:^(void (^)(id x))nextBlock
// 3.发送信号 sendNext:(id)value

// RACSubject:底层实现和RACSignal不一样。
// 1.调用subscribeNext订阅信号, 只是把订阅者保存起来, 并且订阅者的nextBlock已经赋值了。
// 2.调用sendNext发送信号, 遍历刚刚保存的所有订阅者, 一个一个调用订阅者的nextBlock。

// 1.创建信号
RACSubject *subject = [RACSubject subject];

// 2.订阅信号
[subject subscribeNext:^(id x) {
    // block调用时刻: 当信号发出新值, 就会调用。
    NSLog(@"第一个订阅者%@",x);
}];
[subject subscribeNext:^(id x) {
    // block调用时刻: 当信号发出新值, 就会调用。
    NSLog(@"第二个订阅者%@",x);
}];
```

```

    }];

// 3.发送信号
[subject sendNext:@"1"];

// RACReplaySubject使用步骤：
// 1.创建信号 [RACSubject subject]，跟RACSignal不一样，创建信号时没有block。
// 2.可以先订阅信号，也可以先发送信号。
// 2.1 订阅信号 - (RACDisposable *)subscribeNext:(void (^)(id x))nextBlock
// 2.2 发送信号 sendNext:(id)value

// RACReplaySubject：底层实现和RACSubject不一样。
// 1.调用sendNext发送信号，把值保存起来，然后遍历刚刚保存的所有订阅者，一个一个调用订阅者的nextBlock。
// 2.调用subscribeNext订阅信号，遍历保存的所有值，一个一个调用订阅者的nextBlock

// 如果想当一个信号被订阅，就重复播放之前所有值，需要先发送信号，在订阅信号。
// 也就是先保存值，在订阅值。

// 1.创建信号
RACReplaySubject *replaySubject = [RACReplaySubject subject];

// 2.发送信号
[replaySubject sendNext:@1];
[replaySubject sendNext:@2];

// 3.订阅信号
[replaySubject subscribeNext:^(id x) {

    NSLog(@"第一个订阅者接收到的数据%@",x);
}];

// 订阅信号
[replaySubject subscribeNext:^(id x) {

    NSLog(@"第二个订阅者接收到的数据%@",x);
}];

```

- RACSubject替换代理

```
// 需求：
// 1.给当前控制器添加一个按钮，modal到另一个控制器界面
// 2.另一个控制器view中有个按钮，点击按钮，通知当前控制器
```

步骤一：在第二个控制器.h，添加一个RACSubject代替代理。

```
@interface TwoViewController : UIViewController
```

```
@property (nonatomic, strong) RACSubject *delegateSignal;
```

```
@end
```

步骤二：监听第二个控制器按钮点击

```
@implementation TwoViewController
```

```
- (IBAction)notice:(id)sender {
    // 通知第一个控制器，告诉它，按钮被点了

    // 通知代理
    // 判断代理信号是否有值
    if (self.delegateSignal) {
        // 有值，才需要通知
        [self.delegateSignal sendNext:nil];
    }
}
@end
```

步骤三：在第一个控制器中，监听跳转按钮，给第二个控制器的代理信号赋值，并且监听。

```
@implementation OneViewController
```

```
- (IBAction)btnClick:(id)sender {

    // 创建第二个控制器
    TwoViewController *twoVc = [[TwoViewController alloc] init];

    // 设置代理信号
    twoVc.delegateSignal = [RACSubject subject];

    // 订阅代理信号
    [twoVc.delegateSignal subscribeNext:^(id x) {

        NSLog(@"点击了通知按钮");
    }];

    // 跳转到第二个控制器
    [self presentViewController:twoVc animated:YES completion:nil];

}
@end
```

6.6 RACTuple :元组类,类似NSArray,用来包装值.

6.7 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。

使用场景： 1.字典转模型

RACSequence和RACTuple简单使用

```
// 1.遍历数组
NSArray *numbers = @[1,2,3,4];

// 这里其实是三步
// 第一步：把数组转换成集合RACSequence numbers.rac_sequence
// 第二步：把集合RACSequence转换RACSignal信号类,numbers.rac_sequence.signal
// 第三步：订阅信号，激活信号，会自动把集合中的所有值，遍历出来。
[numbers.rac_sequence.signal subscribeNext:^(id x) {

    NSLog(@"%@",x);
}];

// 2.遍历字典,遍历出来的键值对会包装成RACTuple(元组对象)
NSDictionary *dict = @{@"name":@"xmg",@"age":18};
[dict.rac_sequence.signal subscribeNext:^(RACTuple *x) {

    // 解包元组，会把元组的值，按顺序给参数里面的变量赋值
    RACTupleUnpack(NSString *key,NSString *value) = x;

    // 相当于以下写法
    //     NSString *key = x[0];
    //     NSString *value = x[1];

    NSLog(@"%@ %@",key,value);

}];

// 3.字典转模型
// 3.1 OC写法
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"flags.plist" ofType:nil];

NSArray *dictArr = [NSArray arrayWithContentsOfFile:filePath];

NSMutableArray *items = [NSMutableArray array];
```

```

for (NSDictionary *dict in dictArr) {
    FlagItem *item = [FlagItem flagWithDict:dict];
    [items addObject:item];
}

// 3.2 RAC写法
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"flags.plist" ofType:nil];

NSArray *dictArr = [NSArray arrayWithContentsOfFile:filePath];

NSMutableArray *flags = [NSMutableArray array];

_flags = flags;

// rac_sequence注意点: 调用subscribeNext, 并不会马上执行nextBlock, 而是会等一会。
[dictArr.rac_sequence.signal subscribeNext:^(id x) {
    // 运用RAC遍历字典, x: 字典

    FlagItem *item = [FlagItem flagWithDict:x];

    [flags addObject:item];

}];

NSLog(@"%@", NSStringFromCGRect([UIScreen mainScreen].bounds));

// 3.3 RAC高级写法:
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"flags.plist" ofType:nil];

NSArray *dictArr = [NSArray arrayWithContentsOfFile:filePath];
// map:映射的意思, 目的: 把原始值value映射成一个新值
// array: 把集合转换成数组
// 底层实现: 当信号被订阅, 会遍历集合中的原始值, 映射成新值, 并且保存到新的数组里。
NSArray *flags = [[dictArr.rac_sequence map:^(id value) {

    return [FlagItem flagWithDict:value];

}] array];

```

6.8 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

```

// 一、RACCommand使用步骤：
// 1.创建命令 initWithSignalBlock:(RACSignal * (^)(id input))signalBlock
// 2.在signalBlock中，创建RACSignal，并且作为signalBlock的返回值
// 3.执行命令 - (RACSignal *)execute:(id)input

// 二、RACCommand使用注意：
// 1.signalBlock必须要返回一个信号，不能传nil。
// 2.如果不想传递信号，直接创建空的信号[RACSignal empty];
// 3.RACCommand中信号如果数据传递完，必须调用[subscriber sendCompleted]，这时命令才会执行完毕，否则
// 4.RACCommand需要被强引用，否则接收不到RACCommand中的信号，因此RACCommand中的信号是延迟发送的。

// 三、RACCommand设计思想：内部signalBlock为什么要返回一个信号，这个信号有什么用。
// 1.在RAC开发中，通常会把网络请求封装到RACCommand，直接执行某个RACCommand就能发送请求。
// 2.当RACCommand内部请求到数据的时候，需要把请求的数据传递给外界，这时候就需要通过signalBlock返回的信号

// 四、如何拿到RACCommand中返回信号发出的数据。
// 1.RACCommand有个执行信号源executionSignals，这个是signal of signals(信号的信号)，意思是信号发出
// 2.订阅executionSignals就能拿到RACCommand中返回的信号，然后订阅signalBlock返回的信号，就能获取发

// 五、监听当前命令是否正在执行executing

// 六、使用场景,监听按钮点击，网络请求

// 1.创建命令
RACCommand *command = [[RACCommand alloc] initWithSignalBlock:^(RACSignal *(id input)) {

    NSLog(@"执行命令");

    // 创建空信号,必须返回信号
    //      return [RACSignal empty];

    // 2.创建信号,用来传递数据
    return [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

        [subscriber sendNext:@"请求数据"];

        // 注意：数据传递完，最好调用sendCompleted，这时命令才执行完毕。
        [subscriber sendCompleted];

        return nil;
    }]];

}];

```

```
// 强引用命令，不要被销毁，否则接收不到数据
```

```
_command = command;
```

```
// 3. 订阅RACCommand中的信号
```

```
[command.executionSignals subscribeNext:^(id x) {
```

```
    [x subscribeNext:^(id x) {
```

```
        NSLog(@"%@",x);
```

```
    }];
```

```
    }];
```

```
// RAC高级用法
```

```
// switchToLatest:用于signal of signals, 获取signal of signals发出的最新信号,也就是可以直接拿到
```

```
[command.executionSignals.switchToLatest subscribeNext:^(id x) {
```

```
    NSLog(@"%@",x);
```

```
    }];
```

```
// 4. 监听命令是否执行完毕,默认会来一次,可以直接跳过, skip表示跳过第一次信号。
```

```
[[command.executing skip:1] subscribeNext:^(id x) {
```

```
    if ([x boolValue] == YES) {
```

```
        // 正在执行
```

```
        NSLog(@"正在执行");
```

```
    }else{
```

```
        // 执行完成
```

```
        NSLog(@"执行完成");
```

```
    }
```

```
    }];
```

```
// 5. 执行命令
```

```
[self._command execute:@1];
```

6.9 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-multicast:方法创建。

RACMulticastConnection简单使用：


```
// RACMulticastConnection使用步骤：
// 1.创建信号 + (RACSignal *)createSignal:(RACDisposable * (^)(id<RACSubscriber> subscriber))
// 2.创建连接 RACMulticastConnection *connect = [signal publish];
// 3.订阅信号,注意：订阅的不在是之前的信号，而是连接的信号。 [connect.signal subscribeNext:^(id x) {}]
// 4.连接 [connect connect]
```

```
// RACMulticastConnection底层原理：
// 1.创建connect, connect.sourceSignal -> RACSignal(原始信号) connect.signal -> RACSubject
// 2.订阅connect.signal, 会调用RACSubject的subscribeNext, 创建订阅者，而且把订阅者保存起来，不会执行
// 3.[connect connect]内部会订阅RACSignal(原始信号)，并且订阅者是RACSubject
// 3.1.订阅原始信号，就会调用原始信号中的didSubscribe
// 3.2 didSubscribe, 拿到订阅者调用sendNext, 其实是调用RACSubject的sendNext
// 4.RACSubject的sendNext,会遍历RACSubject所有订阅者发送信号。
// 4.1 因为刚刚第二步，都是在订阅RACSubject, 因此会拿到第二步所有的订阅者，调用他们的nextBlock
```

```
// 需求：假设在一个信号中发送请求，每次订阅一次都会发送请求，这样就会导致多次请求。
// 解决：使用RACMulticastConnection就能解决。
```

```
// 1.创建请求信号
```

```
RACSignal *signal = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber))
```

```
    NSLog(@"发送请求");
```

```
    return nil;
```

```
}};
```

```
// 2.订阅信号
```

```
[signal subscribeNext:^(id x) {
```

```
    NSLog(@"接收数据");
```

```
}};
```

```
// 2.订阅信号
```

```
[signal subscribeNext:^(id x) {
```

```
    NSLog(@"接收数据");
```

```
}};
```

```
// 3.运行结果，会执行两遍发送请求，也就是每次订阅都会发送一次请求
```

```
// RACMulticastConnection:解决重复请求问题
```

```
// 1.创建信号
```

```
RACSignal *signal = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber))
```

```
        NSLog(@"发送请求");
        [subscriber sendNext:@1];

        return nil;
    }];

// 2.创建连接
RACMulticastConnection *connect = [signal publish];

// 3.订阅信号,
// 注意: 订阅信号, 也不能激活信号, 只是保存订阅者到数组, 必须通过连接, 当调用连接, 就会一次性调用所有订阅者
[connect.signal subscribeNext:^(id x) {

    NSLog(@"订阅者一信号");

}];

[connect.signal subscribeNext:^(id x) {

    NSLog(@"订阅者二信号");

}];

// 4.连接, 激活信号
[connect connect];
```

6.10 RACScheduler :RAC中的队列, 用GCD封装的。

6.11 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.

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

7.ReactiveCocoa开发中常见用法。

7.1 代替代理:

- rac_signalToSelector : 用于替代代理。

7.2 代替KVO :

- rac_valuesAndChangesForKeyPath : 用于监听某个对象的属性改变。

7.3 监听事件:

- `rac_signalForControlEvents` : 用于监听某个事件。

7.4 代替通知:

- `rac_addObserverForName` :用于监听某个通知。

7.5 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。

7.6 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

- `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。

7.7 代码演示

```
// 1.代替代理
// 需求: 自定义redView,监听红色view中按钮点击
// 之前都是需要通过代理监听, 给红色View添加一个代理属性, 点击按钮的时候, 通知代理做事情
// rac_signalForSelector:把调用某个对象的方法的信息转换成信号, 就要调用这个方法, 就会发送信号。
// 这里表示只要redV调用btnClick:,就会发出信号, 订阅就好了。
[[redV rac_signalForSelector:@selector(btnClick:)] subscribeNext:^(id x) {
    NSLog(@"点击红色按钮");
}];

// 2.KVO
// 把监听redV的center属性改变转换成信号, 只要值改变就会发送信号
// observer:可以传入nil
[[redV rac_valuesAndChangesForKeyPath:@"center" options:NSKeyValueObservingOptionNew observe:^(id x, NSDictionary *dict) {
    NSLog(@"%@",x);
}];

// 3.监听事件
// 把按钮点击事件转换为信号, 点击按钮, 就会发送信号
[[self.btn rac_signalForControlEvents:UIControlEventTouchUpInside] subscribeNext:^(id x)
```

```

        NSLog(@"按钮被点击了");
    }];

// 4.代替通知
// 把监听到的通知转换信号
[[NSNotificationCenter defaultCenter] rac_addObserverForName:UIKeyboardWillShowNotification
        NSLog(@"键盘弹出");
    }];

// 5.监听文本框的文字改变
[_textField.rac_textSignal subscribeNext:^(id x) {

    NSLog(@"文字改变了%@",x);
}];

// 6.处理多个请求，都返回结果的时候，统一做处理。
RACSignal *request1 = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

    // 发送请求1
    [subscriber sendNext:@"发送请求1"];
    return nil;
}]);

RACSignal *request2 = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {
    // 发送请求2
    [subscriber sendNext:@"发送请求2"];
    return nil;
}]);

// 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。
[self rac_liftSelector:@selector(updateUIWithR1:r2:) withSignalsFromArray:@[request1,request2]];

}

// 更新UI
- (void)updateUIWithR1:(id)data r2:(id)data1
{
    NSLog(@"更新UI%@", %@",data,data1);
}

```

8.ReactiveCocoa常见宏。

8.1 RAC(TARGET, [KEYPATH, [NIL_VALUE]]) :用于给某个对象的某个属性绑定。

```
// 只要文本框文字改变, 就会修改label的文字
RAC(self.labelView,text) = _textField.rac_textSignal;
```

8.2 RACObserve(self, name):监听某个对象的某个属性,返回的是信号。

```
[RACObserve(self.view, center) subscribeNext:^(id x) {

    NSLog(@"%@",x);
}];
```

8.3 @weakify(Obj)和@strongify(Obj),一般两个都是配套使用,在主头文件(ReactiveCocoa.h)中并没有导入,需要自己手动导入, RACEXTScope.h 才可以使用。但是每次导入都非常麻烦,只需要在主头文件自己导入就好了。

8.4 RACTuplePack: 把数据包装成RACTuple (元组类)

```
// 把参数中的数据包装成元组
RACTuple *tuple = RACTuplePack(@10,@20);
```

8.5 RACTupleUnpack: 把RACTuple (元组类) 解包成对应的数据。

```
// 把参数中的数据包装成元组
RACTuple *tuple = RACTuplePack(@"xmg",@20);

// 解包元组, 会把元组的值, 按顺序给参数里面的变量赋值
// name = @"xmg" age = @20
RACTupleUnpack(NSString *name,NSNumber *age) = tuple;
```

联系方式

如果你喜欢这篇文章, 可以继续关注我, 微博: 吖了个峥 (<http://weibo.com/2034818060/profile?rightmod=1&wvr=6&mod=personinfo>), 欢迎交流。

(PS: 另外咱们公司小码哥, 诚邀IT届有事业心, 有能力, 有拼劲, 有干劲各路英豪加盟一起创业, 详情可以点击小码哥 (<http://www.520it.com>), 小码哥官方微博 (<http://weibo.com/u/5596623481?topnav=1&wvr=6&topsug=1>), 或者微博私聊我)

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

¥ 打赏支持





(/user(\$/50df42631088b1199)

♥ 5喜6次

🐼 🗨️
更多分享 ▼

139条评论 (按时间正序 · 按时间倒序 · 按喜欢排序) ✎ 添加新评论

 我是乔忘记疯狂 (/users/c9c816980018)
2 楼 2015.09.26 00:57 (/p/87ef6720a096/comments/652773#comment-652773)
(/users/c9c816980018)
精品！👍
♡ 喜欢(0) 回复

 巴黎有条狗 (/users/ddf82c5e7477)
3 楼 2015.09.26 01:28 (/p/87ef6720a096/comments/652818#comment-652818)
(/users/ddf82c5e7477)
使用cocopods导入RAC框架的时候会提示有个metaornsy.h file non found，阿崢知道怎么解决吗？
♡ 喜欢(0) 回复

- 崢吖 (/users/b09c3959ab3b): @ONECode (/users/ddf82c5e7477) 知道 我等会补上去 哈哈
2015.09.26 06:33 (/p/87ef6720a096/comments/652947#comment-652947) 回复
- 淇淇一毛不啊吧 (/users/bf9196eeebfc): 同求，也遇到这个问题
2015.09.28 14:04 (/p/87ef6720a096/comments/661063#comment-661063) 回复
- magic3584 (/users/1d47fb725d7e): 大神求解决这个问题啊。。。开始都卡了。。。
2015.09.28 21:46 (/p/87ef6720a096/comments/662449#comment-662449) 回复



鹏志_Model (/users/bd623c44b32f)

4 楼 · 2015-09-26 09:15 (/p/87ef6720a096/comments/653241#comment-653241)

(/users/bd623c44b32f)

直接pod reactiveCocoa...

♡ 喜欢(0)

回复



KevinMK (/users/fce038424835)

5 楼 · 2015-09-26 13:14 (/p/87ef6720a096/comments/653955#comment-653955)

(/users/fce038424835)

啊争666

♡ 喜欢(0)

回复



伏特加 (/users/85e9774a482d)

6 楼 · 2015-09-26 14:46 (/p/87ef6720a096/comments/654202#comment-654202)

(/users/85e9774a482d)

非常好的教程

♡ 喜欢(0)

回复



EvenCoder (/users/2104d205620a)

7 楼 · 2015-09-27 21:15 (/p/87ef6720a096/comments/658475#comment-658475)

(/users/2104d205620a)

峥哥， 666

♡ 喜欢(0)

回复



山山大王 (/users/c102bbc3bd5a)

8 楼 · 2015-09-28 00:44 (/p/87ef6720a096/comments/659406#comment-659406)

(/users/c102bbc3bd5a)

这篇必须得马啊

♡ 喜欢(0)

举报

回复



BabyWong (/users/b7a42d345441)

9 楼 · 2015-09-28 02:26 (/p/87ef6720a096/comments/659502#comment-659502)

(/users/b7a42d345441)

国庆有时间再看 🍷

♡ 喜欢(0)

回复



AlphaYu (/users/3edde6b75fcc)

10楼 2015.09.28 09:01 (/p/87ef6720a096/comments/659897#comment-659897)

哥，有个单词拼错了，calculate

♡ 喜欢(1)

回复

峥吁 (/users/b09c3959ab3b): @AlphaYu (/users/3edde6b75fcc) 谢谢 哈哈

2015.09.28 10:10 (/p/87ef6720a096/comments/660122#comment-660122)

回复

甲方JiaFang (/users/bc3ecc38a76f): @AlphaYu (/users/3edde6b75fcc) 你看的真细。。。

2016.02.16 17:18 (/p/87ef6720a096/comments/1465803#comment-1465803)

回复

✎ 添加新回复



d123c03ba701 (/users/d123c03ba701)

11楼 2015.09.28 10:51 (/p/87ef6720a096/comments/660273#comment-660273)

真是个好东东

♡ 喜欢(1)

回复



2008慕玉 (/users/70fd779f77ba)

12楼 2015.09.28 10:59 (/p/87ef6720a096/comments/660308#comment-660308)

哪些上线应用 使用他了？

♡ 喜欢(0)

回复

抽筋的鱼 (/users/89f4069db08c): @2008慕玉 (/users/70fd779f77ba) 飞常准

2016.01.20 14:50 (/p/87ef6720a096/comments/1279163#comment-1279163)

回复

Rico (/users/05e2e0157bb4): @2008慕玉 (/users/70fd779f77ba) OSChina

2016.05.23 14:22 (/p/87ef6720a096/comments/2499584#comment-2499584)

回复

男神nick (/users/ba7977e09b58): @_Rico_ (/users/05e2e0157bb4) :你说的是不是开源中国app?

2016.08.19 17:33 (/p/87ef6720a096/comments/3722644#comment-3722644)

回复



borjigeen_narsu (/users/0ea4355199ae)

13 楼 · 2015-09-28 11:32 (/p/87ef6720a096/comments/660452#comment-660452)

正在学习。。。。

♡ 喜欢(0)

回复



何宗柱 (/users/7f15f43e0918)

14 楼 · 2015-09-28 13:09 (/p/87ef6720a096/comments/660849#comment-660849)

学习受用了，表示很棒！！👏

♡ 喜欢(0)

回复



叫我学霸 (/users/633b9315e29b)

15 楼 · 2015-09-28 13:33 (/p/87ef6720a096/comments/660933#comment-660933)

先评论再学习,峥哥辛苦

♡ 喜欢(0)

回复



大慈大悲大熊猫 (/users/97b27815acdd)

16 楼 · 2015-09-28 13:41 (/p/87ef6720a096/comments/660963#comment-660963)

很棒谢谢，但很想看看加法计算器的源码....

♡ 喜欢(0)

回复

yehot (/users/d0002bd5b272): @大慈大悲大熊猫 (/users/97b27815acdd) 链式计算器的代码，这样写就可以了

```
- (CaculatorMaker ^(int))add {
```

```
    return ^(int num) {
        self.result = self.result + num;
        return self;
    };
}
```

```
- (CaculatorMaker ^(int))sub {
```

```
return ^(int num) {  
    self.result = self.result - num;  
    return self;  
};  
}
```

– (CaculatorMaker *(^)(int))muilt {

```
return ^(int num) {  
    self.result = self.result * num;  
    return self;  
};  
}
```

– (CaculatorMaker *(^)(int))divide {

```
return ^(int num) {  
    self.result = self.result / num;  
    return self;  
};  
}
```

2015.09.30 15:41 (/p/87ef6720a096/comments/669244#comment-669244)

回复

大慈大悲大熊猫 (/users/97b27815acdd): @yehot (/users/d0002bd5b272) 已经知道了 还是谢谢你

2015.09.30 15:42 (/p/87ef6720a096/comments/669248#comment-669248)


回复


爷操 (/users/f68bcb6da716): @yehot (/users/d0002bd5b272) + (int)makeCaculators:(void(^)(CaculatorMaker *make))caculatorMaker这个方法该怎么实现?

2015.12.24 14:23 (/p/87ef6720a096/comments/1092990#comment-1092990)

回复

还有 4 条评论, 展开查看

 添加新回复

加载更多  (/notes/2183536/comments?max_id=3825351&order=asc&page=2)


写下你的评论...

发表



⌘+Return 发表

被以下专题收入，发现更多相似内容：




iOS Developer (/collection/3233d1a249ca)

分享 iOS 开发的知识，解决大家遇到的问题，讨论iOS开发的前沿，欢迎大家投稿~

12056篇文章 (/collection/3233d1a249ca) · 25079人关注

[+ | 添加关注 \(/collections/1276/subscribe\)](/collections/1276/subscribe)




iOS开发技巧 (/collection/19dbe28002a3)

【简介】 专题内容主要包括OC、swift等涉及到iOS开发进阶的内容。swift可以关注下我的另一个专题swift开发

1049篇文章 (/collection/19dbe28002a3) · 19301人关注

[+ | 添加关注 \(/collections/6919/subscribe\)](/collections/6919/subscribe)



iOS Development (/collection/ee25d429d275)

沟通想法，分享知识，欢迎大家分享一些好的文章。

396篇文章 (/collection/ee25d429d275) · 8012人关注

[+ | 添加关注 \(/collections/3106/subscribe\)](/collections/3106/subscribe)