



作者 峥吖 (/users/b09c3959ab3b) 2015.09.28

20:44*

✓ 正在关注 (/users/b09c3959ab3b/toggle_like)

写了43950字, 被6517人关注, 获得了2788个喜欢

最快让你上手ReactiveCocoa之进阶篇

字数6146 阅读22033 评论42 喜欢191

前言

由于时间的问题, 暂且只更新这么多了, 后续还会持续更新本文《最快让你上手ReactiveCocoa之进阶篇》, 目前只是简短的介绍了些RAC核心的一些方法, 后续还需要加上MVVM+ReactiveCocoa实战开发。

如果喜欢我的文章, 可以关注我, 微博: 吖了个峥 (<http://weibo.com/2034818060/profile?rightmod=1&wvr=6&mod=personinfo>), 欢迎交流。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

1.ReactiveCocoa常见操作方法介绍。

- 1.1 ReactiveCocoa操作须知
 - 所有的信号 (RACSignal) 都可以进行操作处理, 因为所有操作方法都定义在RACStream.h中, 因此只要继承RACStream就有了操作处理方法。
- 1.2 ReactiveCocoa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。
- 1.3 ReactiveCocoa核心方法bind

- 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。
- 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。
- 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。
- `bind` 方法简单介绍和使用。

```
// 假设想监听文本框的内容，并且在每次输出结果的时候，都在文本框的内容拼接一段文字“输出：”
```

```
// 方式一：在返回结果后，拼接。
```

```
[_textField.rac_textSignal subscribeNext:^(id x) {
```

```
    NSLog(@"输出:%@",x);
```

```
    }];
```

```
// 方式二：在返回结果前，拼接，使用RAC中bind方法做处理。
```

```
// bind方法参数：需要传入一个返回值是RACStreamBindBlock的block参数
```

```
// RACStreamBindBlock是一个block的类型，返回值是信号，参数（value,stop），因此参数的block返回值也是
```

```
// RACStreamBindBlock:
```

```
// 参数一(value):表示接收到信号的原始值，还没做处理
```

```
// 参数二(*stop):用来控制绑定Block, 如果*stop = yes,那么就会结束绑定。
```

```
// 返回值：信号，做好处理，在通过这个信号返回出去，一般使用RACReturnSignal,需要手动导入头文件RACReturnSignal.h
```

```
// bind方法使用步骤：
```

```
// 1.传入一个返回值RACStreamBindBlock的block。
```

```
// 2.描述一个RACStreamBindBlock类型的bindBlock作为block的返回值。
```

```
// 3.描述一个返回结果的信号，作为bindBlock的返回值。
```

```
// 注意：在bindBlock中做信号结果的处理。
```

```
// 底层实现：
```

```
// 1.源信号调用bind,会重新创建一个绑定信号。
```

```
// 2.当绑定信号被订阅，就会调用绑定信号中的didSubscribe, 生成一个bindingBlock。
```

```
// 3.当源信号有内容发出，就会把内容传递到bindingBlock处理，调用bindingBlock(value,stop)
```

```
// 4.调用bindingBlock(value,stop), 会返回一个内容处理完成的信号（RACReturnSignal）。
```

```
// 5.订阅RACReturnSignal, 就会拿到绑定信号的订阅者，把处理完成的信号内容发送出来。
```

```
// 注意:不同订阅者,保存不同的nextBlock,看源码的时候,一定要看清楚订阅者是哪个。
// 这里需要手动导入#import <ReactiveCocoa/RACReturnSignal.h>,才能使用RACReturnSignal。
[_textField.rac_textSignal bind:^(RACStreamBindBlock{

    // 什么时候调用:
    // block作用:表示绑定了一个信号。

    return ^RACStream *(id value, BOOL *stop){

        // 什么时候调用block:当信号有新的值发出,就会来到这个block。

        // block作用:做返回值的处理

        // 做好处理,通过信号返回出去。
        return [RACReturnSignal return:[NSString stringWithFormat:@"输出:%@",value]];
    };

}] subscribeNext:^(id x) {

    NSLog(@"%@",x);

}];
```

- 1.4ReactiveCocoa操作方法之映射(flattenMap,Map)

- flattenMap , Map 用于把源信号内容映射成新的内容。

flattenMap 简单使用

```
// 监听文本框的内容改变，把结构重新映射成一个新值。
```

```
// flattenMap作用: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。
```

```
// flattenMap使用步骤:
```

```
// 1.传入一个block，block类型是返回值RACStream，参数value
```

```
// 2.参数value就是源信号的内容，拿到源信号的内容做处理
```

```
// 3.包装成RACReturnSignal信号，返回出去。
```

```
// flattenMap底层实现:
```

```
// 0.flattenMap内部调用bind方法实现的，flattenMap中block的返回值，会作为bind中bindBlock的返回值。
```

```
// 1.当订阅绑定信号，就会生成bindBlock。
```

```
// 2.当源信号发送内容，就会调用bindBlock(value, *stop)
```

```
// 3.调用bindBlock，内部就会调用flattenMap的block，flattenMap的block作用：就是把处理好的数据包装成
```

```
// 4.返回的信号最终会作为bindBlock中的返回信号，当做bindBlock的返回信号。
```

```
// 5.订阅bindBlock的返回信号，就会拿到绑定信号的订阅者，把处理完成的信号内容发送出来。
```

```
[[_textField.rac_textSignal flattenMap:^(RACStream *(id value) {
```

```
    // block什么时候 ： 源信号发出的时候，就会调用这个block。
```

```
    // block作用 ： 改变源信号的内容。
```

```
    // 返回值：绑定信号的内容。
```

```
    return [RACReturnSignal return:[NSString stringWithFormat:@"输出:%@",value]];
```

```
}] subscribeNext:^(id x) {
```

```
    // 订阅绑定信号，每当源信号发送内容，做完处理，就会调用这个block。
```

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

```
});
```

Map 简单使用:

// 监听文本框的内容改变，把结构重新映射成一个新的值。

// Map作用：把源信号的值映射成一个新的值

// Map使用步骤：

// 1.传入一个block,类型是返回对象，参数是value

// 2.value就是源信号的内容，直接拿到源信号的内容做处理

// 3.把处理好的内容，直接返回就好了，不用包装成信号，返回的值，就是映射的值。

// Map底层实现：

// 0.Map底层其实是调用flattenMap,Map中block中的返回的值会作为flattenMap中block中的值。

// 1.当订阅绑定信号，就会生成bindBlock。

// 3.当源信号发送内容，就会调用bindBlock(value, *stop)

// 4.调用bindBlock，内部就会调用flattenMap的block

// 5.flattenMap的block内部会调用Map中的block，把Map中的block返回的内容包装成返回的信号。

// 5.返回的信号最终会作为bindBlock中的返回信号，当做bindBlock的返回信号。

// 6.订阅bindBlock的返回信号，就会拿到绑定信号的订阅者，把处理完成的信号内容发送出来。

```
[[_textField.rac_textSignal map:^(id value) {  
    // 当源信号发出，就会调用这个block，修改源信号的内容  
    // 返回值：就是处理完源信号的内容。  
    return [NSString stringWithFormat:@"输出:%@",value];  
}] subscribeNext:^(id x) {  
  
    NSLog(@"%@",x);  
}];
```

- FlattenMap和Map的区别

- 1.FlattenMap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FlattenMap。

- 总结：signalOfsignals用FlattenMap。

```

// 创建信号中的信号
RACSubject *signal0fsignals = [RACSubject subject];
RACSubject *signal = [RACSubject subject];

[[signal0fsignals flattenMap:^(RACStream *(id value) {

    // 当signal0fsignals的信号发出信号才会调用

    return value;

}] subscribeNext:^(id x) {

    // 只有signal0fsignals的信号发出信号才会调用，因为内部订阅了bindBlock中返回的信号，也就是fla
    // 也就是flattenMap返回的信号发出内容，才会调用。

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

// 信号的信号发送信号
[signal0fsignals sendNext:signal];

// 信号发送内容
[signal sendNext:@1];

```

- 1.5 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。

```

RACSignal *signalA = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

    [subscriber sendNext:@1];

    [subscriber sendCompleted];

    return nil;
}]];
RACSignal *signalB = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

    [subscriber sendNext:@2];

    return nil;
}]];

// 把signalA拼接到signalB后, signalA发送完成, signalB才会被激活。
RACSignal *concatSignal = [signalA concat:signalB];

// 以后只需要面对拼接信号开发。
// 订阅拼接的信号, 不需要单独订阅signalA, signalB
// 内部会自动订阅。
// 注意: 第一个信号必须发送完成, 第二个信号才会被激活
[concatSignal subscribeNext:^(id x) {

    NSLog(@"%@",x);

}]];

// concat底层实现:
// 1.当拼接信号被订阅, 就会调用拼接信号的didSubscribe
// 2.didSubscribe中, 会先订阅第一个源信号 (signalA)
// 3.会执行第一个源信号 (signalA) 的didSubscribe
// 4.第一个源信号 (signalA) didSubscribe中发送值, 就会调用第一个源信号 (signalA) 订阅者的nextBlock
// 5.第一个源信号 (signalA) didSubscribe中发送完成, 就会调用第一个源信号 (signalA) 订阅者的completeBlock
// 6.订阅第二个源信号 (signalB), 执行第二个源信号 (signalB) 的didSubscribe
// 7.第二个源信号 (signalA) didSubscribe中发送值, 就会通过拼接信号的订阅者把值发送出来。

```

- 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。

```

// 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
// 注意使用then，之前信号的值会被忽略掉。
// 底层实现：1、先过滤掉之前的信号发出的值。2.使用concat连接then返回的信号
[[[RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

    [subscriber sendNext:@1];
    [subscriber sendCompleted];
    return nil;
}] then:^(RACSignal *{
    return [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {
        [subscriber sendNext:@2];
        return nil;
    }]];
}] subscribeNext:^(id x) {

    // 只能接收到第二个信号的值，也就是then返回信号的值
    NSLog(@"%@@",x);
}]];

```

- `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


```

// merge:把多个信号合并成一个信号
//创建多个信号
RACSignal *signalA = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber)

    [subscriber sendNext:@1];

    return nil;
}]];

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

    [subscriber sendNext:@2];

    return nil;
}]];

// 合并信号,任何一个信号发送数据, 都能监听到.
RACSignal *mergeSignal = [signalA merge:signalB];

[mergeSignal subscribeNext:^(id x) {

    NSLog(@"%@ ",x);

}]];

// 底层实现:
// 1.合并信号被订阅的时候, 就会遍历所有信号, 并且发出这些信号。
// 2.每发出一个信号, 这个信号就会被订阅
// 3.也就是合并信号一被订阅, 就会订阅里面所有的信号。
// 4.只要有一个信号被发出就会被监听。

```

- `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。

```

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

    [subscriber sendNext:@1];

    return nil;
}];

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

    [subscriber sendNext:@2];

    return nil;
}];

// 压缩信号A, 信号B
RACSignal *zipSignal = [signalA zipWith:signalB];

[zipSignal subscribeNext:^(id x) {

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

// 底层实现：
// 1.定义压缩信号，内部就会自动订阅signalA, signalB
// 2.每当signalA或者signalB发出信号，就会判断signalA, signalB有没有发出个信号，有就会把最近发出的信号

```

- `combineLatest` :将多个信号合并起来，并且拿到各个信号的最新的值,必须每个合并的信号至少都有过一次`sendNext`，才会触发合并的信号。

```

RACSignal *signalA = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscrib

    [subscriber sendNext:@1];

    return nil;
}];

RACSignal *signalB = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscribe

    [subscriber sendNext:@2];

    return nil;
}];

// 把两个信号组合成一个信号,跟zip一样, 没什么区别
RACSignal *combineSignal = [signalA combineLatestWith:signalB];

[combineSignal subscribeNext:^(id x) {

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

// 底层实现:
// 1.当组合信号被订阅, 内部会自动订阅signalA, signalB,必须两个信号都发出内容, 才会被触发。
// 2.并且把两个信号组合成元组发出。

```

- **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

```

RACSignal *signalA = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscribe

    [subscriber sendNext:@1];

    return nil;
}];

RACSignal *signalB = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscribe

    [subscriber sendNext:@2];

    return nil;
}];

// 聚合
// 常见的用法，（先组合在聚合）。combineLatest:(id<NSFastEnumeration>)signals reduce:(id (^))
// reduce中的block简介：
// reduceblock中的参数，有多少信号组合，reduceblock就有多少参数，每个参数就是之前信号发出的内容
// reduceblock的返回值：聚合信号之后的内容。
RACSignal *reduceSignal = [RACSignal combineLatest:@[signalA,signalB] reduce:^(id(NSNumber

    return [NSString stringWithFormat:@"%d %d",num1,num2];

}];

[reduceSignal subscribeNext:^(id x) {

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

// 底层实现：
// 1.订阅聚合信号，每次有内容发出，就会执行reduceblock，把信号内容转换成reduceblock返回的值。

```

◦ 1.6 ReactiveCocoa操作方法之过滤。

◦ filter :过滤信号，使用它可以获取满足条件的信号。

```

// 过滤：
// 每次信号发出，会先执行过滤条件判断。
[_textField.rac_textSignal filter:^(BOOL(NSString *value) {
    return value.length > 3;
}];

```

- `ignore`:忽略完某些值的信号.

```
// 内部调用filter过滤, 忽略掉ignore的值
[_textField.rac_textSignal ignore:@"1"] subscribeNext:^(id x) {

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

- `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。

```
// 过滤, 当上一次和当前的值不一样, 就会发出内容。
// 在开发中, 刷新UI经常使用, 只有两次数据不一样才需要刷新
[_textField.rac_textSignal distinctUntilChanged] subscribeNext:^(id x) {

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

- `take`:从开始一共取N次的信号

```
// 1、创建信号
RACSubject *signal = [RACSubject subject];

// 2、处理信号, 订阅信号
[[signal take:1] subscribeNext:^(id x) {

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

// 3.发送信号
[signal sendNext:@1];

[signal sendNext:@2];
```

- `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.

```
// 1、创建信号
RACSubject *signal = [RACSubject subject];

// 2、处理信号，订阅信号
[[signal takeLast:1] subscribeNext:^(id x) {

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

// 3.发送信号
[signal sendNext:@1];

[signal sendNext:@2];

[signal sendCompleted];
```

- `takeUntil:(RACSignal *)`:获取信号直到某个信号执行完成

```
// 监听文本框的改变直到当前对象被销毁
[_textField.rac_textSignal takeUntil:self.rac_willDeallocSignal];
```

- `skip:(NSUInteger)`:跳过几个信号,不接受。

```
// 表示输入第一次，不会被监听到，跳过第一次发出的信号
[_textField.rac_textSignal skip:1] subscribeNext:^(id x) {

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

- `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。

```

RACSubject *signalOfSignals = [RACSubject subject];
RACSubject *signal = [RACSubject subject];

// 获取信号中信号最近发出信号，订阅最近发出的信号。
// 注意switchToLatest：只能用于信号中的信号
[signalOfSignals.switchToLatest subscribeNext:^(id x) {

    NSLog(@"%@",x);
}];
[signalOfSignals sendNext:signal];
[signal sendNext:@1];

```

• 1.7 ReactiveCocoa操作方法之秩序。

- 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

```

[[[RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {
    [subscriber sendNext:@1];
    [subscriber sendCompleted];
    return nil;
}] doNext:^(id x) {
// 执行[subscriber sendNext:@1];之前会调用这个Block
    NSLog(@"doNext");;
}] doCompleted:^(
    // 执行[subscriber sendCompleted];之前会调用这个Block
    NSLog(@"doCompleted");;

}] subscribeNext:^(id x) {

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

```

• 1.8 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。

• 1.9 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。

```
RACSignal *signal = [[RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {
    return nil;
}] timeout:1 onScheduler:[RACScheduler currentScheduler]];

[signal subscribeNext:^(id x) {

    NSLog(@"%@",x);
} error:^(NSError *error) {
    // 1秒后会自动调用
    NSLog(@"%@",error);
}];
```

- `interval` 定时：每隔一段时间发出信号

```
[[RACSignal interval:1 onScheduler:[RACScheduler currentScheduler]] subscribeNext:^(id x) {

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

- `delay` 延迟发送next。

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

    [subscriber sendNext:@1];
    return nil;
}] delay:2] subscribeNext:^(id x) {

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

• 1.9 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。


```

__block int i = 0;
[[[RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

    if (i == 10) {
        [subscriber sendNext:@1];
    }else{
        NSLog(@"接收到错误");
        [subscriber sendError:nil];
    }
    i++;

    return nil;

}] retry] subscribeNext:^(id x) {

    NSLog(@"%@",x);

} error:^(NSError *error) {

}

}]];

```

- replay 重放： 当一个信号被多次订阅,反复播放内容

```

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

    [subscriber sendNext:@1];
    [subscriber sendNext:@2];

    return nil;
}] replay];

[signal subscribeNext:^(id x) {

    NSLog(@"第一个订阅者%@",x);

}]];

[signal subscribeNext:^(id x) {

    NSLog(@"第二个订阅者%@",x);

}]];

```

- 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。

```
RACSubject *signal = [RACSubject subject];

_signal = signal;

// 节流, 在一定时间 (1秒) 内, 不接收任何信号内容, 过了这个时间 (1秒) 获取最后发送的信号内容发出。
[[signal throttle:1] subscribeNext:^(id x) {

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

2.介绍MVVM架构思想。

2.1 程序为什么要架构: 便于程序员开发和维护代码。

2.2 常见的架构思想:

- MVC M:模型 V:视图 C:控制器
- MVVM M:模型 V:视图+控制器 VM:视图模型
- MVCS M:模型 V:视图 C:控制器 C:服务类
- VIPER V:视图 I:交互器 P:展示器 E:实体 R:路由

PS:VIPER架构思想 (<http://www.cocoachina.com/ios/20140703/9016.html>)

2.3 MVVM介绍

- 模型 (M):保存视图数据。
- 视图+控制器 (V):展示内容 + 如何展示
- 视图模型 (VM):处理展示的业务逻辑, 包括按钮的点击, 数据的请求和解析等等。

3.ReactiveCocoa + MVVM 实战一: 登录界面

• 3.1需求+分析+步骤

/* 需求：1.监听两个文本框的内容，有内容才允许按钮点击
2.默认登录请求。

用MVVM：实现，之前界面的所有业务逻辑

分析：1.之前界面的所有业务逻辑都交给控制器做处理
2.在MVVM架构中把控制器的业务全部搬去VM模型，也就是每个控制器对应一个VM模型。

步骤：1.创建LoginViewModel类，处理登录界面业务逻辑。
2.这个类里面应该保存着账号的信息，创建一个账号Account模型
3.LoginViewModel应该保存着账号信息Account模型。
4.需要时刻监听Account模型中的账号和密码的改变，怎么监听？
5.在非RAC开发中，都是习惯赋值，在RAC开发中，需要改变开发思维，由赋值转变为绑定，可以在一开始初始化的
6.每次Account模型的值改变，就需要判断按钮能否点击，在VM模型中做处理，给外界提供一个能否点击按钮的信
7.这个登录信号需要判断Account中账号和密码是否有值，用KVO监听这两个值的改变，把他们聚合成登录信号。
8.监听按钮的点击，由VM处理，应该给VM声明一个RACCommand，专门处理登录业务逻辑。
9.执行命令，把数据包装成信号传递出去
10.监听命令中信号的数据传递
11.监听命令的执行时刻

*/

• 3.2 控制器的代码

```

@interface ViewController ()

@property (nonatomic, strong) LoginViewModel *loginViewModel;

@property (weak, nonatomic) IBOutlet UITextField *accountField;
@property (weak, nonatomic) IBOutlet UITextField *pwdField;

@property (weak, nonatomic) IBOutlet UIButton *loginBtn;

@end

- (LoginViewModel *)loginViewModel
{
    if (_loginViewModel == nil) {
        _loginViewModel = [[LoginViewModel alloc] init];
    }
    return _loginViewModel;
}

// 视图模型绑定
- (void)bindModel
{
    // 给模型的属性绑定信号
    // 只要账号文本框一改变，就会给account赋值
    RAC(self.loginViewModel.account, account) = _accountField.rac_textSignal;
    RAC(self.loginViewModel.account, pwd) = _pwdField.rac_textSignal;

    // 绑定登录按钮
    RAC(self.loginBtn,enabled) = self.loginViewModel.enableLoginSignal;

    // 监听登录按钮点击
    [[_loginBtn rac_signalForControlEvents:UIControlEventTouchUpInside] subscribeNext:^(id x

        // 执行登录事件
        [self.loginViewModel.LoginCommand execute:nil];
    }]];
}

```

• 3.3 VM的代码

```

@interface LoginViewModel : NSObject

@property (nonatomic, strong) Account *account;

```

```

// 是否允许登录的信号
@property (nonatomic, strong, readonly) RACSignal *enableLoginSignal;

@property (nonatomic, strong, readonly) RACCommand *LoginCommand;

@end

@implementation LoginViewModel
- (Account *)account
{
    if (_account == nil) {
        _account = [[Account alloc] init];
    }
    return _account;
}
- (instancetype)init
{
    if (self = [super init]) {
        [self initialBind];
    }
    return self;
}

// 初始化绑定
- (void)initialBind
{
    // 监听账号的属性值改变，把他们聚合成一个信号。
    _enableLoginSignal = [RACSignal combineLatest:@[RACObserve(self.account, account), RACObserve(self.pwd, pwd)]];

    return @(account.length && pwd.length);
}

// 处理登录业务逻辑
_LoginCommand = [[RACCommand alloc] initWithSignalBlock:^(RACSignal *(id input) {

    NSLog(@"点击了登录");
    return [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

        // 模仿网络延迟
        dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(0.5 * NSEC_PER_SEC)), dispatch_get_main_queue());

        [subscriber sendNext:@"登录成功"];

        // 数据传送完毕，必须调用完成，否则命令永远处于执行状态
        [subscriber sendCompleted];
    }]);
}];

```

```

    });

    return nil;
}];

// 监听登录产生的数据
[_LoginCommand.executionSignals.switchToLatest subscribeNext:^(id x) {

    if ([x isEqualToString:@"登录成功"]) {
        NSLog(@"登录成功");
    }
}];

// 监听登录状态
[[_LoginCommand.executing skip:1] subscribeNext:^(id x) {
    if ([x isEqualToNumber:@(YES)]) {

        // 正在登录ing...
        // 用蒙版提示
        [MBProgressHUD showMessage:@"正在登录..."];

    }else
    {
        // 登录成功
        // 隐藏蒙版
        [MBProgressHUD hideHUD];
    }
}];
}

```

4.ReactiveCocoa + MVVM 实战二：网络请求数据

- 4.1 接口：这里先给朋友介绍一个免费的网络数据接口，[豆瓣](#)。可以经常用来练习一些网络请求的小Demo.
- 4.2 需求+分析+步骤

/*

需求：请求豆瓣图书信息，url:https://api.douban.com/v2/book/search?q=基础

分析：请求一样，交给VM模型管理

步骤：

1. 控制器提供一个视图模型（requestViewModel），处理界面的业务逻辑
2. VM提供一个命令，处理请求业务逻辑
3. 在创建命令的block中，会把请求包装成一个信号，等请求成功的时候，就会把数据传递出去。
4. 请求数据成功，应该把字典转换成模型，保存到视图模型中，控制器想用就直接从视图模型中获取。
5. 假设控制器想展示内容到tableView，直接让视图模型成为tableView的数据源，把所有的业务逻辑交给视图模型

*/

• 4.3 控制器代码

```

@interface ViewController ()

@property (nonatomic, weak) UITableView *tableView;

@property (nonatomic, strong) RequestViewModel *requestViewModel;

@end

@implementation ViewController
- (RequestViewModel *)requestViewModel
{
    if (_requestViewModel == nil) {
        _requestViewModel = [[RequestViewModel alloc] init];
    }
    return _requestViewModel;
}

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // 创建tableView
    UITableView *tableView = [[UITableView alloc] initWithFrame:self.view.bounds];
    tableView.dataSource = self.requestViewModel;

    [self.view addSubview:tableView];

    // 执行请求
    RACSignal *requestSignal = [self.requestViewModel.requestCommand execute:nil];

    // 获取请求的数据
    [requestSignal subscribeNext:^(NSArray *x) {

        self.requestViewModel.models = x;

        [self.tableView reloadData];

    }];
}

@end

```

- 4.4视图模型(VM)代码


```

@interface RequestViewModel : NSObject<UITableViewDataSource>

// 请求命令
@property (nonatomic, strong, readonly) RACCommand *reugesCommand;

//模型数组
@property (nonatomic, strong, readonly) NSArray *models;

@end

@implementation RequestViewModel

- (instancetype)init
{
    if (self = [super init]) {

        [self initialBind];
    }
    return self;
}

- (void)initialBind
{
    _reugesCommand = [[RACCommand alloc] initWithSignalBlock:^(RACSignal *(id input) {

        RACSignal *requestSignal = [RACSignal createSignal:^(RACDisposable *(id<RACSubscriber> subscriber) {

            NSMutableDictionary *parameters = [NSMutableDictionary dictionary];
            parameters[@"q"] = @"基础";

            // 发送请求
            [[AFHTTPRequestOperationManager manager] GET:@"https://api.douban.com/v2/book/search?keyword=基础"];
            NSLog(@"%@", responseObject);

            // 请求成功调用
            // 把数据用信号传递出去
            [subscriber sendNext:responseObject];

            [subscriber sendCompleted];

        } failure:^(AFHTTPRequestOperation * _Nonnull operation, NSError * _Nonnull error) {
            // 请求失败调用

```

```
    }];  
  
    return nil;  
}];
```

```
// 在返回数据信号时，把数据中的字典映射成模型信号，传递出去  
return [requestSignal map:^id(NSDictionary *value) {  
    NSMutableArray *dictArr = value[@"books"];  
  
    // 字典转模型，遍历字典中的所有元素，全部映射成模型，并且生成数组  
    NSArray *modelArr = [[dictArr.rac_sequence map:^id(id value) {  
  
        return [Book bookWithDict:value];  
    }] array];  
  
    return modelArr;  
}];
```

```
    }];
```

```
}
```

```
#pragma mark - UITableViewDataSource
```

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section  
{  
    return self.models.count;  
}
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)  
{  
    static NSString *ID = @"cell";  
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:ID];  
    if (cell == nil) {  
  
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle reuseIdentifier:  
    }  
  
    Book *book = self.models[indexPath.row];  
    cell.detailTextLabel.text = book.subtitle;  
    cell.textLabel.text = book.title;  
  
    return cell;  
}
```

联系方式

(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)

➕ 推荐拓展阅读

📄 举报文章 © 著作权归作者所有

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



(/users/7e4b9f21752bc19525ea05cd19)


♡ 1 喜欢



更多分享 ▼

42条评论 （ 按时间正序 · 按时间倒序 · 按喜欢排序 ）

✎ 添加新评论

 Azen (/users/5a86c7f0d18b)


2楼 2015-09-29 09:04 (/p/e10e5ca413b7/comments/663629#comment-663629)

(/users/5a86c7f0d18b)

哟~~要上课啦~~~~

♡ 喜欢(0)

回复

 di3 (/users/4e2423a6bd69)


3楼 2015-09-29 10:33 (/p/e10e5ca413b7/comments/664014#comment-664014)

(/users/4e2423a6bd69)

支持 👍


♡ 喜欢(0)

回复

 鹏志_Model (/users/bd623c44b32f)
4 楼 · 2015-09-30 19:46 (/p/e10e5ca413b7/comments/668237#comment-668237)


很清晰

♡ 喜欢(0) 回复

 Bool (/users/b9e1fcbe62d5)
5 楼 · 2015-10-01 02:38 (/p/e10e5ca413b7/comments/671169#comment-671169)

感谢


♡ 喜欢(0) 回复


 偷偷学很多东西 (/users/fcb3e6f01134)
6 楼 · 2015-10-05 07:14 (/p/e10e5ca413b7/comments/688039#comment-688039)

方便出个block所有使用场景的文章？

♡ 喜欢(1) 回复


洲洲哥 (/users/1338683b18e0): @偷偷学很多东西 (/users/fcb3e6f01134) <http://www.jianshu.com/p/83c7581bf8e3> (<http://www.jianshu.com/p/83c7581bf8e3>) 看这里吧
2016.08.11 19:46 (/p/e10e5ca413b7/comments/3576059#comment-3576059) 回复

 添加新回复

 _mry (/users/49c9990ff26c)
7 楼 · 2015-10-06 15:19 (/p/e10e5ca413b7/comments/694590#comment-694590)

不错

♡ 喜欢(0) 回复

 FMG (/users/a934172a1ef5)
8 楼 · 2015-10-07 13:44 (/p/e10e5ca413b7/comments/699373#comment-699373)

哈哈，在这里听你讲课， 峥哥， 🙌

♡ 喜欢(0) 回复



daydreamsan (/users/e791dcaffc46)

9 楼 · 2015.10.07 20:39 (/p/e10e5ca413b7/comments/701117#comment-701117)

非常实用

♡ 喜欢(0)

回复



留什么白 (/users/0ededc5397dc)

10 楼 · 2015.10.08 10:30 (/p/e10e5ca413b7/comments/703474#comment-703474)

有一段markdown没处理好。。。

♡ 喜欢(0)

回复



saitjr (/users/8947226a0458)

11 楼 · 2015.10.20 23:35 (/p/e10e5ca413b7/comments/769765#comment-769765)

给两个个人小意见:

1. replay的例子，可能在信号中使用下面的代码，更能看出replay的效果

```
static int a = 1;
```

```
[subscriber sendNext:@(a)];
```

```
a ++;
```

这样两次输出可以清楚的看到都是输出的1，而不是每次都是1， 2

2. 最后一个TableView加载数据的例子，个人认为DataSource写在VM中不是很好，要不然提出来单独写，要不然写在VC里面，然后把cell的配置用RAC单独放在cell里面。直接写在VM中和以前写在VC中没什么区别啊，这样VM看起来依然像百宝箱。（顺便说下我自己的见解，讨论下：VM处理的应该是View上要显示的数据，保证View能直接显示，不应该在VM中出现View)

♡ 喜欢(0)

回复

峥吶 (/users/b09c3959ab3b): @TangJR (/users/8947226a0458) 恩 后面在课程改了，非常感谢 😊

2015.10.21 08:23 (/p/e10e5ca413b7/comments/770468#comment-770468)

回复

zhao1zhihui (/users/e227c7735856): @峥吶 (/users/b09c3959ab3b) dataSource 写在哪里比较好

2016.08.03 00:25 (/p/e10e5ca413b7/comments/3421554#comment-3421554)

回复

✎ 添加新回复



ljysdfz (/users/9658b4413ae7)

12 楼 2015.10.23 10:46 (/p/e10e5ca413b7/comments/782009#comment-782009)

有个小建议

takeUntil解释为参数signalTrigger发送next或completed时导致self发生completed

♡ 喜欢(0)

回复

峥吖 (/users/b09c3959ab3b): @ljysdfz (/users/9658b4413ae7) 谢谢

2015.10.23 11:08 (/p/e10e5ca413b7/comments/782179#comment-782179)

回复

79000c5098e5 (/users/79000c5098e5): RACSignal *signalA = [RACSignal createSignal:^(RACDisposable * (id<RACSubscriber> subscriber) {

```
[subscriber sendNext:@88];
[subscriber sendCompleted];
return nil;
}];
```

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

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

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

```
[subscriber sendNext:@11];
[subscriber sendCompleted];
return nil;
}] takeUntil:signalA] subscribeNext:^(id x) {
```

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

signalA都完成了，为什么没有打印？

2016.03.12 15:08 (/p/e10e5ca413b7/comments/1695174#comment-1695174)

回复

添加新回复



ljysdfz (/users/9658b4413ae7)

13 楼 2015.10.23 12:04 (/p/e10e5ca413b7/comments/782422#comment-782422)

(/users/9658b4413ae7)

有个小问题

对于switchToLatest的使用源码如下

```
RACSubject *signalOfSignals = [RACSubject subject];
RACSubject *signal = [RACSubject subject];
[signalOfSignals sendNext:signal];
[signal sendNext:@1];

// 获取信号中信号最近发出信号， 订阅最近发出的信号。
// 注意switchToLatest： 只能用于信号中的信号
[signalOfSignals.switchToLatest subscribeNext:^(id x) {

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

这样做显然是不行的

看到使用RACSubject不用看下面就知道操作基本是先订阅再发送信号

否则subscriber都没有缓存发送信号也没有订阅者接收处理

所以应该使用如下的顺序

```
RACSubject *signalOfSignals = [RACSubject subject];
RACSubject *signal = [RACSubject subject];

// 获取信号中信号最近发出信号， 订阅最近发出的信号。
// 注意switchToLatest： 只能用于信号中的信号
[signalOfSignals.switchToLatest subscribeNext:^(id x) {

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

[signalOfSignals sendNext:signal];
[signal sendNext:@1];
```

这样就可以打印出结果

当然 如果不想改变顺序

合理使用RACSubject的子类RACReplaySubject

也是一个好办法，

其在父类subscriber cacheable的基础上加入valuesReceived的双缓存机制

修改类型声明后如下

```
RACReplaySubject *signalOfSignals = [RACReplaySubject subject];
RACReplaySubject *signal = [RACReplaySubject subject];

[signalOfSignals sendNext:signal];
[signal sendNext:@1];

// 获取信号中信号最近发出信号， 订阅最近发出的信号。
// 注意switchToLatest: 只能用于信号中的信号
[signalOfSignals.switchToLatest subscribeNext:^(id x) {

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

这样也可以打印出结果

因为这种顺序不恰当的情况在这两篇RAC的博文里多次出现
猜想应该是叶峥做了对比调试尚未改过来的原因
不过对于RAC 执行顺序是重中之重 还望多多检查
如有失言 海涵

♡ 喜欢(0)


回复

峥叶 (/users/b09c3959ab3b): @ljysdfz (/users/9658b4413ae7) 嗯嗯 当时没注意 我上课的时候已经改好了 我回头把顺序改改

2015.10.23 12:26 (/p/e10e5ca413b7/comments/782503#comment-782503)

回复

✎ 添加新回复

 叶孤城__ (/users/b82d2721ba07)
14 楼 2015.10.23 14:53 (/p/e10e5ca413b7/comments/783095#comment-783095)

(/users/b82d2721ba07)

有空去广州小码哥参观一下。 😊

♡ 喜欢(0)

回复

峥叶 (/users/b09c3959ab3b): @叶孤城__ (/users/b82d2721ba07) 来来来 欢迎

2015.10.23 15:31 (/p/e10e5ca413b7/comments/783230#comment-783230)

回复

南栀倾寒 (/users/cc1e4faec5f7): @叶孤城__ (/users/b82d2721ba07) 叫上我


2015.10.23 15:35 (/p/e10e5ca413b7/comments/783253#comment-783253)


回复

叶孤城__ (/users/b82d2721ba07): @南栀倾寒 (/users/cc1e4faec5f7) ok

2015.10.23 16:14 (/p/e10e5ca413b7/comments/783416#comment-783416)

回复

 添加新回复

 握握手好朋友 (/users/d48a9e6695a6)

15 楼 2015.10.26 15:14 (/p/e10e5ca413b7/comments/797165#comment-797165)

(/users/d48a9e6695a6)

觉得在VM中出现View，这样似乎不是很友好，直接在VC中bind 数据源，当VM中得数据fetch ed 时候，这样很自然的就会在VC中刷新了。这样感觉更符合数据驱动视图的思想。

♡ 喜欢(0)


回复


恩底弥翁之鹰 (/users/84d882ef5267): @握握手好朋友 (/users/d48a9e6695a6) 同感，vm中应该尽量不出

现ui相关的东东，

2015.12.25 01:04 (/p/e10e5ca413b7/comments/1096706#comment-1096706)

回复

 添加新回复

 玩泥巴的孩子 (/users/69e2e1652f2b)

16 楼 2015.11.03 08:09 (/p/e10e5ca413b7/comments/825557#comment-825557)

(/users/69e2e1652f2b)

学习了


♡ 喜欢(0)

回复

加载更多 ⬇ (/notes/2191743/comments?max_id=3819359&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：



程序员 (/collection/NEt52a)

如果你是程序员，或者有一颗喜欢写程序的心，喜欢分享技术干货、项目经验、程序员日常囧事等等，欢迎投稿《程序员》专题。 专题主编：小...

22285篇文章 (/collection/NEt52a) · 158731人关注

+ | 添加关注 (/collections/16/subscribe)



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)



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) · 8013人关注

+ | 添加关注 (/collections/3106/subscribe)