MVVM 在美团点评酒旅业务中的实践

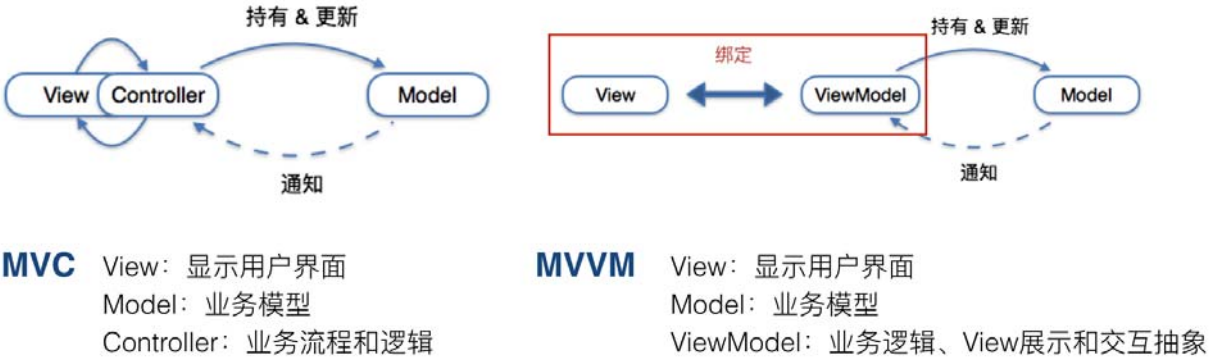
前言

MVVM 是 Model-View-ViewModel 的缩写，是一种常见的软件架构模式。MVVM 架构的由来，最早由微软的架构师提出并应用在 WPF 中，随后在各大语言中衍生出许多新的框架。MVVM 在使用中，使用绑定技术，使得 ViewModel 变化时，View 也随之变化，从而实现了将试图和逻辑分离。借助 MVVM 的这个特点，我所在的酒旅业务团队从2015年开始尝试使用 MVVM 架构进行开发，基于 MVVM 架构开展技术建设，支撑业务的快速发展。本文通过一些简化的场景，介绍 iOS 开发中 MVVM 的代码实践，帮助大家理解如何使用 MVVM，真正让 MVVM 为项目和团队带来价值。

MVVM 概述

MVC 与 MVVM

说到 MVVM，大家往往会同时提到另外一种常见的架构模式 MVC。MVC 是 iOS 官方默认使用的架构模式，接下来我们就从图例解释下两者的区别。



在 MVC 中，View 负责显示用户界面，Model 负责封装业务数据模型，Controller 这层起到业务流程和逻辑的处理操作。而在 MVVM 中，View 和 Model 和 MVC 中的定义一致，ViewModel 负责处理业务逻辑以及视图层的抽象。

借助图例，我们重点说下不同的这部分。在 MVC 中，View 和 Controller 是相互持有的，View 的操作会直接传递到 Controller 中进行逻辑处理，同时 Controller 也会直接操作 View 的属性实现展示变更。而在 MVVM 中，View 和 ViewModel 的关系是通过绑定建立起来的，基于这种绑定关系，ViewModel 只需要改变自身内部的属性，View 能对应监听到变化而产生相应的改变。

为什么使用 MVVM?

MVVM 架构的核心在于展示和逻辑的分离，通过“绑定”连接。我们从两个方面来看这个架构：

1. 由于展示和逻辑是独立的模块，意味着我们就能很方便地拆解和替换其中的模块。在酒旅业务中，经常会出现不同业务之间借鉴 UI 的情况，其中 UI 展示相近甚至完全一样，但是数据和逻辑根据业务特点单独定制。在这种情况下，我们就可以借助 MVVM 的优势，实现 UI 组件的复用。
2. 由于“绑定”关系是通过三方框架建立的，绑定的结果是可信的。因此只需要测试到 ViewModel 一层，就可以覆盖业务逻辑在 UI 上的体现。又由于 ViewModel 本身都是普通 Objective-C 对象，不需要复杂的 Mock 就可以运行，所以更容易进行测试代码的编写。

基于以上两方面的考虑，MVVM 在酒旅业务中广泛使用起来。

- 美团从2015年初开始初步尝试 MVVM，并结合 RAC（ReactiveCocoa）实施 FRP。
- 2016年美团点评广泛使用 MVVM，酒旅的各业务通过 MVVM 实现代码高效复用。
- 2017年酒旅广泛实践基于 MVVM 的自动化测试，酒旅交易业务逻辑覆盖率达80%。

MVVM 实战

在 MVVM 模式中，绑定是其中最重要的环节，我们根据实际场景抽象出以下几种形式：

- 单向数据绑定
- 集合数据绑定
- 双向数据绑定
- 执行过程绑定
- 错误处理

接下来，我们将通过具体的 Demo，来讲解这些绑定在 iOS 中代码写法。

在 Demo 中定义的功能如下：

- 第一行展示酒店的名称，并且也会展示在输入框下面的文字区。
- 用户输入入住人和手机号时，输入框下面的文字会同步展示输入的信息。
- 用户点击清空输入按钮时，入住人和手机号文本框被清空。
- 用户点击提交按钮时，会校验入住人和手机号是否为空；若其中任何一项为空，输入框下的文字会提示错误信息。

单向数据绑定

单向数据绑定指的是从 View 到 ViewModel，或者是 ViewModel 到 View 单向的一个过程。比如说，我们把一个 ViewModel 的属性显示到 View 上，这就是一个单向数据绑定的过程。单向数据绑定一般应用在展示的场景中，通过将 View 的属性和 ViewModel 中对应的数据关联起来，后续对 UI 的操作直接通过修改 ViewModel 中对应的属性来实现，而不通过修改 View 的方式。以当前 Demo 为例，酒店名称的展示用到了单向数据绑定。

我们首先看下 View 和 ViewModel 的关键属性定义。在 HotelTitleCell 中，定义了一个 UI 组件 titleLabel；在 HotelTitleViewModel 中定义的 label 需要展示的内容，属性命名为 title。

```
// UI定义
@interface HotelTitleCell : MVVMBaseCell
@property (strong, nonatomic) UILabel *titleLabel;
@end

// ViewModel定义
@interface HotelTitleViewModel : BaseViewModel
@property (strong, nonatomic) NSString *title;
@end
```


接下来是绑定部分的代码。数据绑定的方法操作，一般放在对应的 View 中，也就是 HotelTitleCell 中执行绑定的方法。

```
@implementation HotelTitleCell

// 数据绑定
- (void)bindViewModel:(HotelTitleViewModel *)viewModel
{
    RAC(self.titleLabel, text) = RACObserve(viewModel, title);
}

@end
```

在绑定的流程里，通过等号赋值将两个信号关联起来。等号左侧通过 RAC() 宏将 self.titleLabel 的 text 属性封装成一个信号；等号右侧则是 RACObserve() 宏，这个宏的含

义是当 viewModel 的 title 属性被赋值时产生的信号。通过这样的绑定方式，当外部在设置 viewModel 的 title 值时，self.titleLabel 的 text 值也会随之发生改变。

```
// 单向数据绑定-UI绑定
HotelTitleCell *hotelTitleCell = [[HotelTitleCell alloc] init];

HotelTitleViewModel *hotelTitleVM = [[HotelTitleViewModel alloc]
init];
hotelTitleVM.title = @"美团酒店-望京国际研发园店";
[hotelTitleCell bindViewModel:hotelTitleVM];
```

最后看下 View 和 ViewModel 的创建操作。在创建 ViewModel 的时候设置了 title 的值，接下来进行绑定操作时 title 的值会传递到 label 的设置 text 属性，从而实现 label 的文本显示我们想要的值。后面的代码如果再次设置 title 值时，label 都会显示相应的值。

双向数据绑定

双向数据绑定稍微复杂一点，它存在两个方向的数据流动。一方面，它会将 ViewModel 中的属性显示在 View 上；另一方面，也会通过 View 采集数据并同步更新 ViewModel 里的属性。双向数据绑定常见于表单的情况，比如：输入框、开关操作等等。以当前 Demo 为例，入住人和手机号两个模块均用到了双向数据绑定，我们以其中一个为例。

View 和 ViewModel 的关键部分代码如下：

```
// UI定义
@interface LabelAndTextFieldCell : MVVMBaseCell
...
@property (strong, nonatomic) UITextField *textField;
@end

// ViewModel定义
@interface LabelAndTextFieldViewModel : BaseViewModel
```

```
...
@property (strong, nonatomic) NSString *inputText;
@end

// 数据双向绑定
- (void)bindViewModel:(LabelAndTextFieldViewModel *)viewModel
{
...
    RACChannelTerminal *channelTerminal =
self.textField.rac_newTextChannel;
    RACChannelTerminal *channelTerminal2 =
RACChannelTo(viewModel, inputText);

    [channelTerminal subscribe:channelTerminal2];
    [channelTerminal2 subscribe:channelTerminal];
}
```

我们重点解读下双向绑定的代码：双向绑定我们会用到 RACChannelTerminal 这样一种信号，RACChannelTerminal 遵循 RACSubscriber 协议，可以订阅别的信号，一旦它所订阅的信号产生事件，它也会产生一个相同的事件，无论是值事件还是错误事件、完成事件。

我们分别将 textField 输入内容变动、viewModel 中对应属性 inputText 变动分别封装成两个 RACChannelTerminal 信号，这里注意与单向数据绑定不一致的地方，两个信号的绑定不是通过等号赋值，而是使用 subscribe 方法相互订阅。当一方产生变化时，另一方都会得到相应的值。

这里有用到一个 RACChannelTo() 的宏，补充解释下。这个宏的基本使用格式是 RACChannelTo(target, keyPath)，该宏创建一个 RACChannelTerminal 对象，并与 target 的 keyPath 形成绑定关系，一旦 RACChannelTerminal 产生一个新值，就把该值赋给 target.keyPath。

值得一提的是，RACChannelTo() 既可以作为左值，又可以作为右值：

- 当它作为右值时，它将产生的 RACChannelTerminal 返回，该对象可订阅别的信号，或者被别的订阅者订阅。
- 当它作为左值时，必须为它赋值 RACChannelTerminal 对象，与其自身产生的 RACChannelTerminal 形成绑定关系（即互相订阅）。

集合数据绑定

集合数据绑定常见于多个组件之间的联动，换句话说操作的不是单个的数据源，而是多个数据源合并处理的结果。以当前 Demo 为例，下方的展示区域文字，会汇总酒店名称、入住人姓名、手机号的内容，并统一展示出来；如果输入内容发生变化，下方的展示区域的文字也会随之变化。

View 和 ViewModel 的定义在先前的讲解中已经提到，这儿不过多展开，以下重点关注集合数据绑定的关键代码：

```
// 信号组合
- (void)combineHotelTitleViewModel:(HotelTitleViewModel
*)hotelTitleVM
                 nameInputViewModel:(NameInputViewModel
*)nameInputVM
                phoneInputViewModel:(PhoneInputViewModel
*)phoneInputVM
{
...
    NSArray *signals = @[hotelTitleVM.titleSignal,
nameInputVM.inputChannel, phoneInputVM.inputChannel];

    _textSignal = [RACSignal combineLatest:signals
                                    reduce:^id (NSString *title,
NSString *name, NSString *phone) {
        return [NSString stringWithFormat:@"%@ 将入住 %@\n联系电
话：%@",
                name? : @"",
                title,
                phone? : @""];
                       }];

}
```

在集合数据绑定中，我们使用到 RAC 中的 combineLatest 操作，解释下这个方法的作用。我们汇总了三个信号，包括酒店的名称信号、输入的姓名信号、输入的手机号信号，这三个信号都发生过赋值操作后，再有任意一个信号发出时触发处理的代码。在相应信号变化的操作中，我们将三个文字内容合并在一起，拼接成一个内容字符串。

执行过程绑定

执行过程的绑定通常会和按钮、Cell 的点击过程有关，我们会把按钮点击这样的行为和对应操作的代码进行关联绑定，这就是一个执行过程的绑定。以 Demo 为例，当用户点

击清空输入按钮的时候，输入框里的内容会被清空，此处的点击操作即用到了执行过程的绑定操作。

执行过程的信号通常会创建一个 RACCommand，在信号创建时填入具体的操作内容，最后将 View 中控件的 command 信号与 ViewModel 的 command 信号用等号绑定起来，具体的代码如下。RACCommand 的特点是能够手动触发信号。


```
// UI定义
@interface OrderDetailCell : MVVMBaseCell
...
@property (strong, nonatomic) UIButton *clearButton;
@end

// ViewModel定义
@interface OrderDetailViewModel : BaseViewModel
...
@property (strong, nonatomic) RACCommand *clearCommand;
@end

// 执行信号创建
_clearCommand = [[RACCommand alloc]
initWithSignalBlock:^RACSignal *(id input) {
        nameInputVM.inputText = nil;
        phoneInputVM.inputText = nil;
        return [RACSignal empty];
    }];

// 操作绑定
self.clearButton.rac_command = viewModel.clearCommand;
```

> 这儿延伸解释下 RACCommand 这个关键字，它在创建时会通过传进一个用于构建 RACSignal 的 block 参数，而 block 中的参数 input 为执行 command 时传入的数

> 据。当这个 command 执行时，这个用来构建 command 的 signal 的 block 中的内容就会执行。

错误处理

错误处理指的是，我们会把一些不同来源、不同原因的错误汇总到一起，呈现到 View 层显示出来。以 Demo 为例，点击提交按钮时会校验入住人和手机号是否为空，如果任何一项为空，展示错误的提示信息。

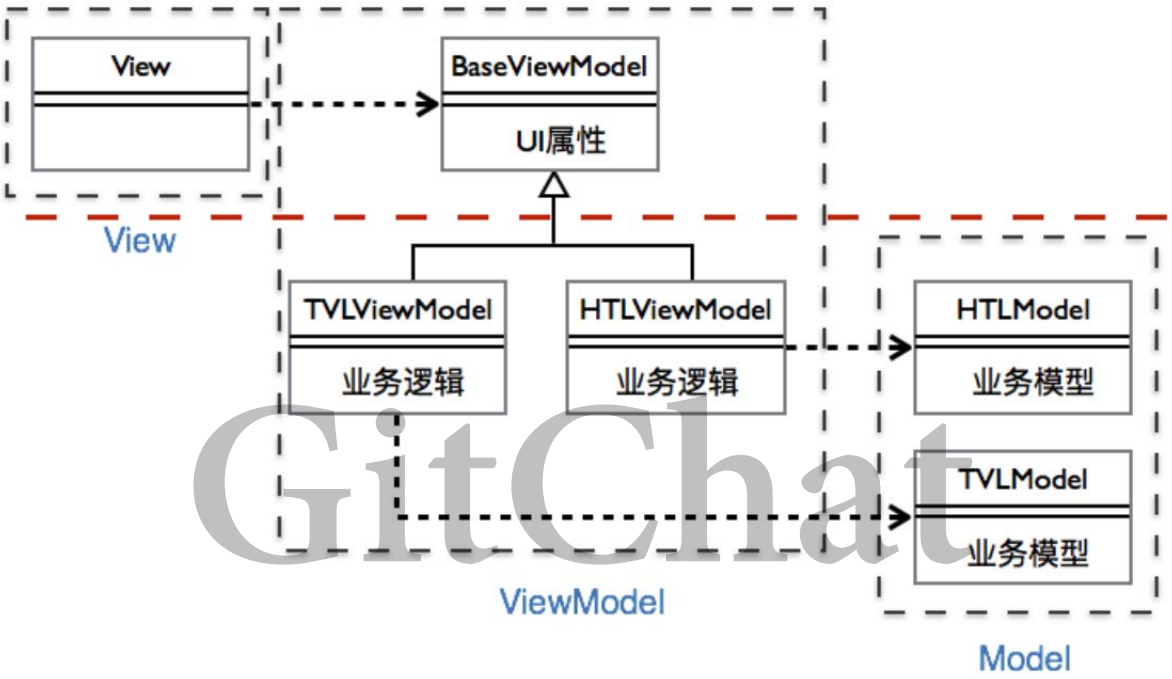
下面是该例子的关键代码，分别展示了错误的抛出以及对应的处理操作。


```
// 错误信号创建
_submitCommand = [[RACCommand alloc]
initWithSignalBlock:^RACSignal *(id input) {
    if (nameInputVM.inputText == nil) {
        NSDictionary *userInfo = @{NSLocalizedDescriptionKey:
@"入住人不能为空！"};
        NSError *error = [NSError
errorWithDomain:NSCocoaErrorDomain code:-1 userInfo:userInfo];
        return [RACSignal error:error];
    } else if (phoneInputVM.inputText == nil) {
        NSDictionary *userInfo = @{NSLocalizedDescriptionKey:
@"手机号不能为空！"};
        NSError *error = [NSError
errorWithDomain:NSCocoaErrorDomain code:-1 userInfo:userInfo];
        return [RACSignal error:error];
    }

   return [RACSignal empty];
}];

// 信号绑定
self.submitButton.rac_command = viewModel.submitCommand;
@weakify(self)
[self.submitButton.rac_command.errors subscribeNext:^(NSError *x)
{
    @strongify(self)
    self.textView.text = x.localizedDescription;
}];
```

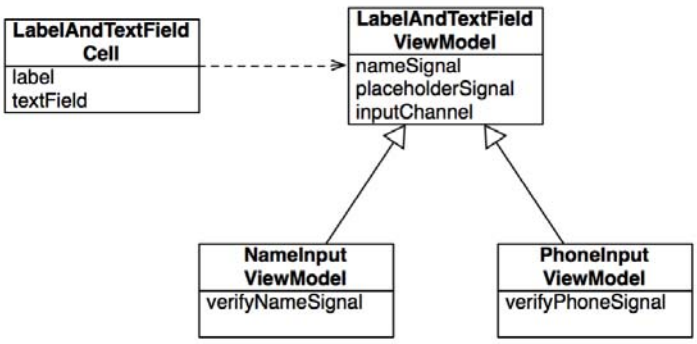
跟执行过程绑定中的代码类似，我们同样创建了一个提交按钮操作的 RACCommand，在这个信号中会校验入住人和手机号的输入内容，如果其中任何一项为空，返回一个错误的信号。在绑定操作信号时，会监听 errors 信号产生的操作，这儿用到 subscribeNext 的方法，在对应的 block 中，我们根据收到的错误内容进行相应的展示处理。

MVVM 如何进行代码复用

刚才我们通过代码实例，讲解了 View 和 ViewModel 之间如何进行绑定的。针对代码复用这一诉求，我们总结了一种标准实践。原理如下：



从图例上，我们定义了两个业务复用同一个 UI 组件。View 层是公共的，ViewModel 层分解为两个部位，在公共的 ViewModel 中，定义了 UI 相关的属性；在各自业务的子类 ViewModel 中，分别定义了各自特有的业务逻辑。我们以先前的 Demo 为例说明：



入住人和手机号两项输入用到了 UI 的组件复用。针对 View 的内容，我们定制了对应的 ViewModel，包含 UI 属性的信号：

- 输入框名字
- 输入框提示
- 输入的内容

这些信号与 View 里对应的控件进行绑定。在子类中，输入入住人 ViewModel 中实现校验姓名的逻辑，输入手机号 ViewModel 中实现校验手机号的逻辑。

MVVM 编写自动化测试

通过上面的 Demo，覆盖了5种典型的绑定场景，可以解决项目中遇到的绝大多数问题。接下来我们讨论一下针对 ViewModel 层如何做单元测试。

先前提到基于“绑定”，只需要测试到 ViewModel，就可以覆盖业务逻辑在 UI 上的体现。我们来看具体两个例子。

（1）信号逻辑校验例子

在这个例子中，我们尝试对输入手机号的合法性校验环节进行测试。具体的测试代码如下：


```
_verifyPhoneSignal = [RACObserve(self, inputText) map:^id
(NSString *phone) {
            NSString *phoneRegexp = @"^1(3[0-9]|5[0-35-9]|8[0-25-
9])\\d{8}$";
            NSPredicate *regextestmobile = [NSPredicate
predicateWithFormat:@"SELF MATCHES %@", phoneRegexp];
            return @((BOOL)[regextestmobile
evaluateWithObject:phone]);
        }];

// 测试“校验手机号”逻辑是否正确
- (void)testVerifyPhone
{
    // 步骤一
    PhoneInputViewModel *viewModel = [[PhoneInputViewModel alloc]
init];
    // 步骤二
    RACChannel *channel = [[RACChannel alloc] init];
    [viewModel.inputChannel subscribe:channel.leadingTerminal];
    [channel.leadingTerminal subscribe:viewModel.inputChannel];
    [channel.followingTerminal sendNext:@"18612345678"]; // 模拟从
文本框输入 18612345678
    // 步骤三
    NSNumber *verifyPhoneResult = [viewModel.verifyPhoneSignal
first];
    XCTAssertEqualObjects(verifyPhoneResult, @(YES));
    // 步骤四
    [viewModel setValue:@"13810001000" forKey:@"inputText"]; //
```

```
模拟ViewModel更新phone值为13810001000
    XCTAssertEqualObjects([channel.followingTerminal first],
@"13810001000"); // 检验文本框内容是否为13810001000
}
```

第一部分给出这个手机号校验信号的逻辑，第二部分在测试代码中模拟用户的输入来看校验规则是否生效。这儿可以看到所有的测试代码是针对 ViewModel 的，我们一步步解说下测试代码的操作：

- 步骤一，创建对应的 ViewModel。
- 步骤二，创建一个 RACChannel 信号并与 ViewModel 中的属性绑定，模拟文本框的输入。这儿的代码看起来有点复杂，延展解释一下。首先，创建一个 RACChannel 信号去代表 UI 中 textField 的信号，channel 包含两个 channelTerminal，一个是输出的 channelTerminal 对外发送内容，一个是输入的 channelTerminal 接受外部得到的内容。
- 步骤三，基于刚才模拟文本框数据的内容，校验信号中的操作会触发，在这儿检测校验的结果。XCTAssertEqualObjects() 是 XCTest 中提供的宏，用于比较两个对象是否一致；若不一致，及测试失败。
- 步骤四，测试 ViewModel 中属性的改变，是否会同步到文本框中。

（2）控件操作验证例子

在上一个例子中，我们展示了双向绑定的测试方式，大家对测试的代码有一定的熟悉了。接下来重点解读下按钮的点击操作测试，代码如下：


```
- (void)testClearCommand
{
    HotelTitleViewModel *hotelTitleVM = [[HotelTitleViewModel
alloc] init];
    [hotelTitleVM setValue:@"美团酒店" forKey:@"title"];
    XCTAssertEqualObjects([hotelTitleVM valueForKey:@"title"],
@"美团酒店");

    NameInputViewModel *nameInputVM = [[NameInputViewModel alloc]
init];
    [nameInputVM setValue:@"Zhuo" forKey:@"inputText"];
    XCTAssertEqualObjects([nameInputVM valueForKey:@"inputText"],
@"Zhuo");

    PhoneInputViewModel *phoneInputVM = [[PhoneInputViewModel
alloc] init];
    [phoneInputVM setValue:@"18610001000" forKey:@"inputText"];
    XCTAssertEqualObjects([phoneInputVM
valueForKey:@"inputText"], @"18610001000");

    // 模拟按钮点击操作进行测试
    OrderDetailViewModel *viewModel = [[OrderDetailViewModel
```

```
alloc] init];
    [viewModel combineHotelTitleViewModel:hotelTitleVM
nameInputViewModel:nameInputVM phoneInputViewModel:phoneInputVM];
    [viewModel.clearCommand execute:nil];
    XCTAssertEqualObjects([nameInputVM valueForKey:@"inputText"],
nil);
    XCTAssertEqualObjects([phoneInputVM
valueForKey:@"inputText"], nil);
}
```

首先创建了几个需要用到的 ViewModel，并进行关联和绑定。在用户点击操作 UI 时会触发 command 的信号，为了模拟点击操作此处用到 excute 方法去执行 command，触发 command 中相应操作的逻辑。触发信号后，继续通过 XCTAssertEqualObjects() 宏检查对应的属性是否设置为 nil。

通过以上两个比较有代表性的例子，我们就可以对于 ViewModel 中的手机号输入逻辑，以及点击清空按钮的点击逻辑进行测试。

总结

MVVM 实现了将视图和逻辑分离，从而可以很方便的进行代码复用以及自动化测试。本文从代码实例的角度，介绍了 iOS MVVM 五种常见的绑定写法；以一些 Demo 场景介绍了针对 MVVM 架构如何做代码复用，如何编写自动化测试。希望通过本文中的例子，启发大家在项目中真正意义上将 MVVM 使用起来，为项目和团队带来价值。

> 发现文章有错误、对内容有疑问，欢迎在读者圈留言，同时可以关注美团点评技术团队微信公众号（meituantech），在后台给我们留言。