浅谈iOS中MVVM的架构设计与团队协作

http://www.cocoachina.com/ios/20150122/10987.html

https://github.com/lizelu/MVVM

今天写这篇文章是想达到抛砖引玉的作用，想与大家交流一下思想，相互学习，博文中有不足之处还望大家批评指正。本篇文章的内容沿袭以往博客的风格，也是以干货为主，偶尔扯扯咸蛋(哈哈~不好好工作又开始发表博客啦~)。

由 于本人项目经验有限，关于架构设计方面的东西理解有限，我个人对MVVM的理解主要是借鉴于之前的用过的MVC的Web框架~在学校的时候用过 ThinkPHP框架，和SSH框架，都是MVC的架构模式，今天MVVM与传统的MVC可谓是极为相似，也可以说是兄弟关系，也就是一家人了。

说 到架构设计和团队协作，这个对App的开发还是比较重要的。即使作为一个专业的搬砖者，前提是你这砖搬完放在哪？不只是Code有框架，其他的东西都是有 框架的，比如桥梁等等神马的~在这儿就不往外扯了。一个好的工程框架不进可以提高团队的协作效率，同时还可以减少代码的冗余度和耦合性，合理的分工与系统 的架构设计是少不了的。

至于团队协作不仅仅是有SVN或者Git这些版本控制工具就行的，至于如何在iOS开发中使用SVN,请参考之前的博客（[iOS开发之版本控制（SVN）](http://www.cnblogs.com/ludashi/p/4018245.html" \t "_blank)）。一个团队可以高效的工作，本人觉得交流是最为重要的，团队中的每个人都比较和气，而且交流上没有什么障碍（不过有的团队中总有几个合不来的人），交流在团队中最为重要。至于SVN怎么用，那都不是事儿！

好了今天就以我写的一个Demo来浅谈一下iOS开发中的架构设计和团队协作，今天的咸蛋先到这儿，切入今天的话题。

为了写今天的博客我花了点时间做了个工程，这个工程后台的接口用的新浪微博的API来进行测试的，在本文的后面也会跟上GitHub的分享链接。OK~说的高大上一些就是，仁者见仁智者见智，交流思想，共同学习。

**一、小酌一下MVVM**

在这呢也不赘述什么是MVC,神马又是MVVM了,在百度上谷歌一下一抓一大把，在这儿就简单的提上一嘴。下面的Demo用的就是MVVM的架构模式。

Model层是少不了的了，我们得有东西充当DTO(数据传输对象)，当然，用字典也是可以的，编程么，要灵活一些。Model层是比较薄的一层，如果学过Java的小伙伴的话，对JavaBean应该不陌生吧。

ViewModel层，就是View和Model层的粘合剂，他是一个放置用户输入验证逻辑，视图显示逻辑，发起网络请求和其他各种各样的代码的极好的地方。说白了，就是把原来ViewController层的业务逻辑和页面逻辑等剥离出来放到ViewModel层。

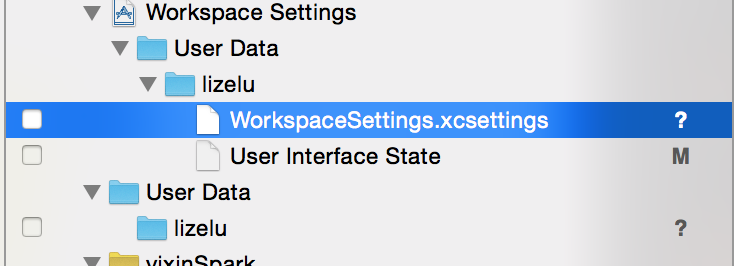
View层，就是ViewController层，他的任务就是从ViewModel层获取数据，然后显示。

上面对MVVM就先简单的这么一说，好好的理解并应用的话，还得实战。

**二、关于工程中是否使用StoryBoard的论述**

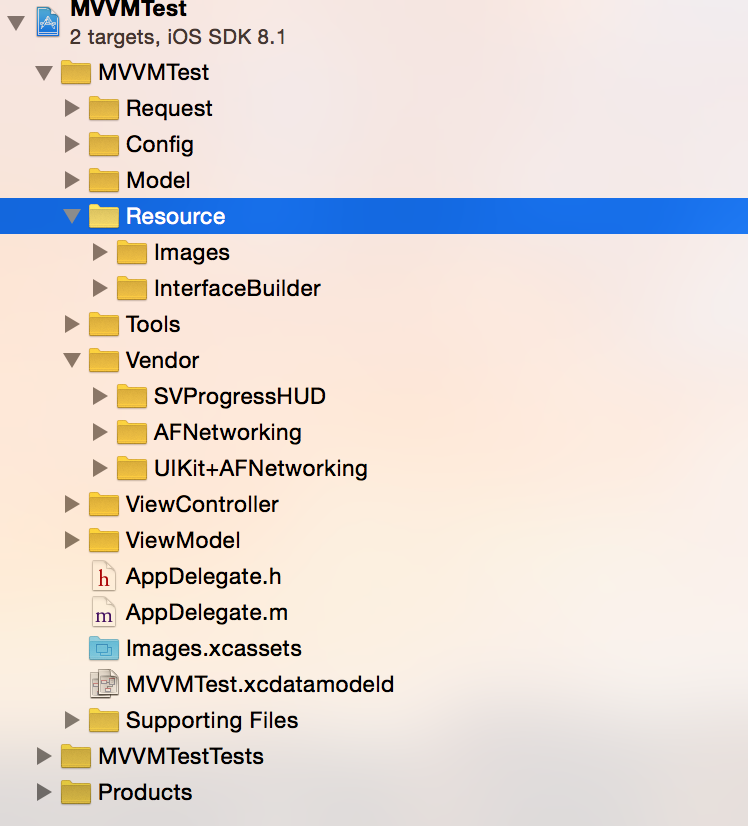
从网上经常看到说不推荐使用StoryBoard或者Xib,推荐用纯代码手写。个人认为这种观点是和苹果设计StoryBoard的初衷相悖的，在我做过的项目中是以StoryBoard为主，xib为辅，然后用代码整合每个StoryBoard.

举 一个用Storyboard好处的例子就OK了，给控件添加约束，如果用Storyboard完成那是分分秒的事情，而用代码的添加约束的话是何等的恶 心，纯代码写的话会把大量的时间花在写UI上，而且技术含量是比较低的，这个个人认为没什么必要。在团队合作中负责UI开发的小伙伴只需没人负责一个 Storyboard,各开发各的，用SVN提交时把下面的勾（如下图）去掉即可，这样用Storyboard是没有问题的。然后再用代码进行整合就OK 了。如果你在你的工程中加入了新的资源文件的话，用XCode自带的SVN提交的话需要吧Project Setting文件一并提交。



**三、实战MVVM（用Xcode创建的Group是虚拟的文件夹，为了便于维护，建议创建物理文件夹，然后再手动引入）**

1.下面通过一个实例来体会一下MVVM架构模式，下面是该工程的一级目录如下，每层之间的交互是用Block的形式来实现的



工程目录说明：

1. Request:文件夹下存储网络请求的类，下面会给出具体的实现
2. Config:就是工程的配置文件
3. Resource:就是工程的资源文件，下面有图片资源和Storyboard文件资源
4. Tools是:工具文件类，存放工具类，比如数据正则匹配等。
5. Vender:存放第三方类库
6. Model:这个就不多说了
7. ViewController:存放ViewController类资源文件，也就是View层
8. ViewModel：存放各种业务逻辑和网络请求

2.详解Request:Request负责网络请求的东西，具体如下：

81707410003870.png

NetRequestClass是存放网络请求的代码，本工程用的AF，因为本工程只是一个Demo,所以就只封装了监测网络状态，GET请求，POST请求方法，根据现实需要，还可以封装上传下载等类方法。

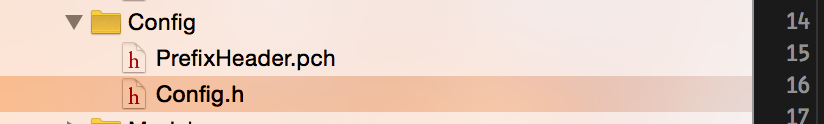
NetRequestClass.h中的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //  //  NetRequestClass.h  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/6.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import @interface NetRequestClass : NSObject    #pragma 监测网络的可链接性  + (BOOL) netWorkReachabilityWithURLString:(NSString \*) strUrl;    #pragma POST请求  + (void) NetRequestPOSTWithRequestURL: (NSString \*) requestURLString                          WithParameter: (NSDictionary \*) parameter                   WithReturnValeuBlock: (ReturnValueBlock) block                     WithErrorCodeBlock: (ErrorCodeBlock) errorBlock                       WithFailureBlock: (FailureBlock) failureBlock;    #pragma GET请求  + (void) NetRequestGETWithRequestURL: (NSString \*) requestURLString                          WithParameter: (NSDictionary \*) parameter                  WithReturnValeuBlock: (ReturnValueBlock) block                    WithErrorCodeBlock: (ErrorCodeBlock) errorBlock                      WithFailureBlock: (FailureBlock) failureBlock;    @end |

NetRequestClass.m中的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115 | //  //  NetRequestClass.m  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/6.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import "NetRequestClass.h"    @interface NetRequestClass ()    @end      @implementation NetRequestClass  #pragma 监测网络的可链接性  + (BOOL) netWorkReachabilityWithURLString:(NSString \*) strUrl  {      \_\_block BOOL netState = NO;        NSURL \*baseURL = [NSURL URLWithString:strUrl];        AFHTTPRequestOperationManager \*manager = [[AFHTTPRequestOperationManager alloc] initWithBaseURL:baseURL];        NSOperationQueue \*operationQueue = manager.operationQueue;        [manager.reachabilityManager setReachabilityStatusChangeBlock:^(AFNetworkReachabilityStatus status) {          switch (status) {              case AFNetworkReachabilityStatusReachableViaWWAN:              case AFNetworkReachabilityStatusReachableViaWiFi:                  [operationQueue setSuspended:NO];                  netState = YES;                  break;              case AFNetworkReachabilityStatusNotReachable:                  netState = NO;              default:                  [operationQueue setSuspended:YES];                  break;          }      }];        [manager.reachabilityManager startMonitoring];        return netState;  }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   在这做判断如果有dic里有errorCode   调用errorBlock(dic)   没有errorCode则调用block(dic   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    #pragma --mark GET请求方式  + (void) NetRequestGETWithRequestURL: (NSString \*) requestURLString                         WithParameter: (NSDictionary \*) parameter                  WithReturnValeuBlock: (ReturnValueBlock) block                    WithErrorCodeBlock: (ErrorCodeBlock) errorBlock                      WithFailureBlock: (FailureBlock) failureBlock  {      AFHTTPRequestOperationManager \*manager = [[AFHTTPRequestOperationManager alloc] init];        AFHTTPRequestOperation \*op = [manager GET:requestURLString parameters:parameter success:^(AFHTTPRequestOperation \*operation, id responseObject) {          NSDictionary \*dic = [NSJSONSerialization JSONObjectWithData:responseObject options:NSJSONReadingAllowFragments error:nil];          DDLog(@"%@", dic);            block(dic);        } failure:^(AFHTTPRequestOperation \*operation, NSError \*error) {          failureBlock();      }];        op.responseSerializer = [AFHTTPResponseSerializer serializer];        [op start];    }    #pragma --mark POST请求方式    + (void) NetRequestPOSTWithRequestURL: (NSString \*) requestURLString                          WithParameter: (NSDictionary \*) parameter                   WithReturnValeuBlock: (ReturnValueBlock) block                     WithErrorCodeBlock: (ErrorCodeBlock) errorBlock                       WithFailureBlock: (FailureBlock) failureBlock  {      AFHTTPRequestOperationManager \*manager = [[AFHTTPRequestOperationManager alloc] init];        AFHTTPRequestOperation \*op = [manager POST:requestURLString parameters:parameter success:^(AFHTTPRequestOperation \*operation, id responseObject) {          NSDictionary \*dic = [NSJSONSerialization JSONObjectWithData:responseObject options:NSJSONReadingAllowFragments error:nil];            DDLog(@"%@", dic);            block(dic);          /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*           在这做判断如果有dic里有errorCode           调用errorBlock(dic)           没有errorCode则调用block(dic           \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/        } failure:^(AFHTTPRequestOperation \*operation, NSError \*error) {          failureBlock();      }];        op.responseSerializer = [AFHTTPResponseSerializer serializer];        [op start];    }          @end |

3.详解Config:创建pch文件，和Config.h文件



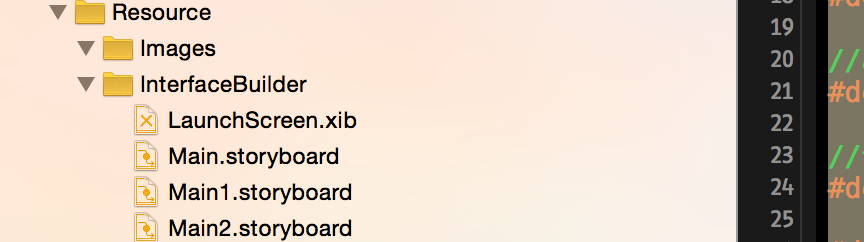
pch文件引入常用的头文件，内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | //  //  PrefixHeader.pch  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/6.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #ifndef MVVMTest\_PrefixHeader\_pch  #define MVVMTest\_PrefixHeader\_pch    #import"AFNetworking.h"  #import "UIKit+AFNetworking.h"  #import "Config.h"    #import "NetRequestClass.h"  #import "SVProgressHUD.h"  #endif |

Config.h中就是各种宏定义和各种枚举类型和block类型，代码如下：

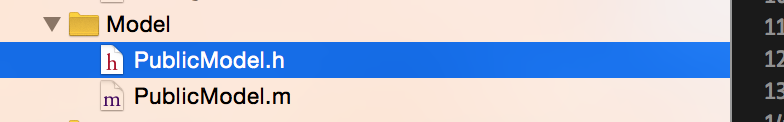
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | //  //  Config.h  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/6.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #ifndef MVVMTest\_Config\_h  #define MVVMTest\_Config\_h    //定义返回请求数据的block类型  typedef void (^ReturnValueBlock) (id returnValue);  typedef void (^ErrorCodeBlock) (id errorCode);  typedef void (^FailureBlock)();  typedef void (^NetWorkBlock)(BOOL netConnetState);    #define DDLog(xx, ...)  NSLog(@"%s(%d): " xx, \_\_PRETTY\_FUNCTION\_\_, \_\_LINE\_\_, ##\_\_VA\_ARGS\_\_)    //accessToken  #define ACCESSTOKEN @"你自己的access\_token"    //请求公共微博的网络接口  #define REQUESTPUBLICURL @"<https://api.weibo.com/2/statuses/public_timeline.json>"    #define SOURCE @"source"  #define TOKEN @"access\_token"  #define COUNT @"count"    #define STATUSES @"statuses"  #define CREATETIME @"created\_at"  #define WEIBOID @"id"  #define WEIBOTEXT @"text"  #define USER @"user"  #define UID @"id"  #define HEADIMAGEURL @"profile\_image\_url"  #define USERNAME @"screen\_name"    #endif |

4.详解资源文件Resource，结构如下图：



Image中就存放各种图片（3x,2x等），InterfaceBuider里面就是放一些Xib和Storyboard文件，每个负责UI的开发人员负责一个Storyboard

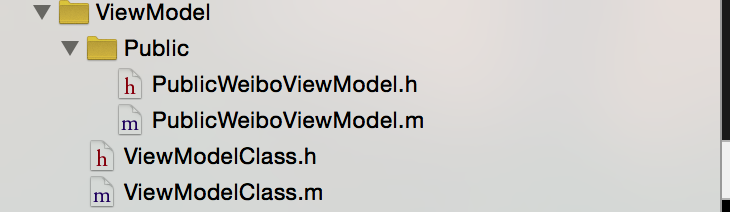
5.详解Model:本工程用的是请求公共微博接口我们需要在页面上现实用户的头像，用户名，发布日期，博文，已经隐式的用户ID和微博ID,文件目录结构如下：



PublicModel中的内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | //  //  PublicModel.h  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/8.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import @interface PublicModel : NSObject  @property (strong, nonatomic) NSString \*userId;  @property (strong, nonatomic) NSString \*weiboId;  @property (strong, nonatomic) NSString \*userName;  @property (strong, nonatomic) NSURL \*imageUrl;  @property (strong, nonatomic) NSString \*date;  @property (strong, nonatomic) NSString \*text;    @end |

6.详解ViewModel层，本层是最为重要的一层，下面是本层的详细截图，ViewModeClass是所有ViewMode的父类，其中存储着共同部分



ViewModelClass.h中的内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | //  //  ViewModelClass.h  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/8.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import @interface ViewModelClass : NSObject    @property (strong, nonatomic) ReturnValueBlock returnBlock;  @property (strong, nonatomic) ErrorCodeBlock errorBlock;  @property (strong, nonatomic) FailureBlock failureBlock;      //获取网络的链接状态  -(void) netWorkStateWithNetConnectBlock: (NetWorkBlock) netConnectBlock WithURlStr: (NSString \*) strURl;    // 传入交互的Block块  -(void) setBlockWithReturnBlock: (ReturnValueBlock) returnBlock                   WithErrorBlock: (ErrorCodeBlock) errorBlock                 WithFailureBlock: (FailureBlock) failureBlock;  @end |

ViewModelClass.m中的内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //  //  ViewModelClass.m  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/8.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import "ViewModelClass.h"  @implementation ViewModelClass    #pragma 获取网络可到达状态  -(void) netWorkStateWithNetConnectBlock: (NetWorkBlock) netConnectBlock WithURlStr: (NSString \*) strURl;  {      BOOL netState = [NetRequestClass netWorkReachabilityWithURLString:strURl];      netConnectBlock(netState);  }    #pragma 接收穿过来的block  -(void) setBlockWithReturnBlock: (ReturnValueBlock) returnBlock                   WithErrorBlock: (ErrorCodeBlock) errorBlock                 WithFailureBlock: (FailureBlock) failureBlock  {      \_returnBlock = returnBlock;      \_errorBlock = errorBlock;      \_failureBlock = failureBlock;  }    @end |

PublicWeiboViewModel.h中的内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | //  //  PublicWeiboViewModel.h  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/8.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import "ViewModelClass.h"  #import "PublicModel.h"    @interface PublicWeiboViewModel : ViewModelClass  //获取围脖列表  -(void) fetchPublicWeiBo;    //跳转到微博详情页  -(void) weiboDetailWithPublicModel: (PublicModel \*) publicModel WithViewController: (UIViewController \*)superController;  @end |

PublicWeiboViewModel.m中的内容如下：

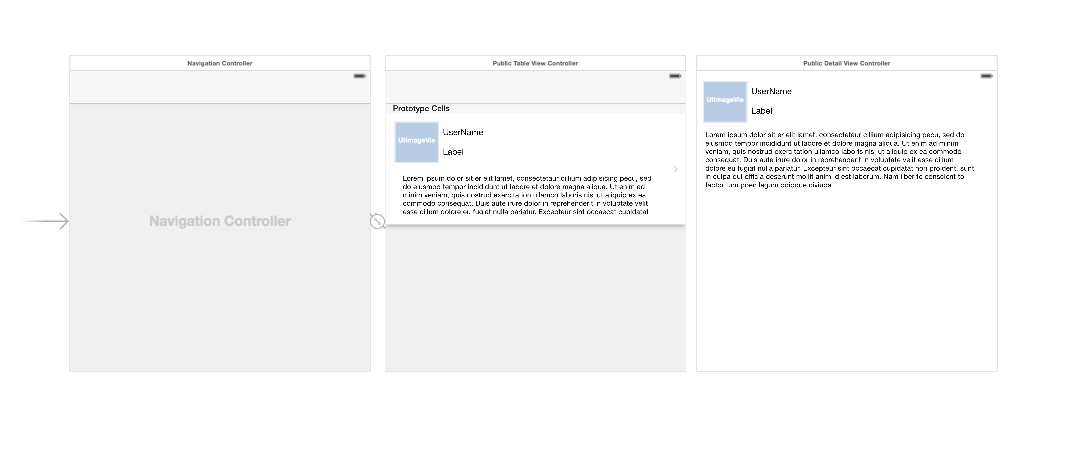
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101 | //  //  PublicWeiboViewModel.m  //  MVVMTest  //  //  Created by 李泽鲁 on 15/1/8.  //  Copyright (c) 2015年 李泽鲁. All rights reserved.  //    #import "PublicWeiboViewModel.h"  #import "PublicDetailViewController.h"    @implementation PublicWeiboViewModel    //获取公共微博  -(void) fetchPublicWeiBo  {      NSDictionary \*parameter = @{TOKEN: ACCESSTOKEN,                                  COUNT: @"100"                                  };      [NetRequestClass NetRequestGETWithRequestURL:REQUESTPUBLICURL WithParameter:parameter WithReturnValeuBlock:^(id returnValue) {            DDLog(@"%@", returnValue);          [self fetchValueSuccessWithDic:returnValue];        } WithErrorCodeBlock:^(id errorCode) {          DDLog(@"%@", errorCode);          [self errorCodeWithDic:errorCode];        } WithFailureBlock:^{          [self netFailure];          DDLog(@"网络异常");        }];    }        #pragma 获取到正确的数据，对正确的数据进行处理  -(void)fetchValueSuccessWithDic: (NSDictionary \*) returnValue  {      //对从后台获取的数据进行处理，然后传给ViewController层进行显示        NSArray \*statuses = returnValue[STATUSES];      NSMutableArray \*publicModelArray = [[NSMutableArray alloc] initWithCapacity:statuses.count];        for (int i = 0; i < statuses.count; i ++) {          PublicModel \*publicModel = [[PublicModel alloc] init];            //设置时间          NSDateFormatter \*iosDateFormater=[[NSDateFormatter alloc]init];          iosDateFormater.dateFormat=@"EEE MMM d HH:mm:ss Z yyyy";            //必须设置，否则无法解析          iosDateFormater.locale=[[NSLocale alloc]initWithLocaleIdentifier:@"en\_US"];          NSDate \*date=[iosDateFormater dateFromString:statuses[i][CREATETIME]];            //目的格式          NSDateFormatter \*resultFormatter=[[NSDateFormatter alloc]init];          [resultFormatter setDateFormat:@"MM月dd日 HH:mm"];            publicModel.date = [resultFormatter stringFromDate:date];          publicModel.userName = statuses[i][USER][USERNAME];          publicModel.text = statuses[i][WEIBOTEXT];          publicModel.imageUrl = [NSURL URLWithString:statuses[i][USER][HEADIMAGEURL]];          publicModel.userId = statuses[i][USER][UID];          publicModel.weiboId = statuses[i][WEIBOID];            [publicModelArray addObject:publicModel];        }        self.returnBlock(publicModelArray);  }    #pragma 对ErrorCode进行处理  -(void) errorCodeWithDic: (NSDictionary \*) errorDic  {      self.errorBlock(errorDic);  }    #pragma 对网路异常进行处理  -(void) netFailure  {      self.failureBlock();  }      #pragma 跳转到详情页面，如需网路请求的，可在此方法中添加相应的网络请求  -(void) weiboDetailWithPublicModel: (PublicModel \*) publicModel WithViewController:(UIViewController \*)superController  {      DDLog(@"%@,%@,%@",publicModel.userId,publicModel.weiboId,publicModel.text);      UIStoryboard \*storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];      PublicDetailViewController \*detailController = [storyboard instantiateViewControllerWithIdentifier:@"PublicDetailViewController"];      detailController.publicModel = publicModel;      [superController.navigationController pushViewController:detailController animated:YES];    }      @end |

7.ViewController层的目录结构如下：



上面的代码就不一一粘了（主要是手按command + C 按累了），后面的链接会有源码

8.storybord中的结构如下：



运行的最终效果：



9.完整目录结构，页面间的业务逻辑，和网络的请求数据是放在ViewModel层的，当然了这也不是绝对的，要灵活把握。我个人是特别喜欢编程的，因为编程灵活起来就会很有乐趣。

10.文章快结束了，在这儿在总结一下SVN使用频率比较高的而且比较重要的命令吧（当然，本人平时主要是用XCode自带的SVN可视化管理~）

（1）、更新本地代码命令

* svn  up 更新版本
* svn info 查看当前版本信息

（2）、代码的提交

* svn info 查看当前本地版本信息
* svn up 更新到最新版本信息
* svn st 查看文件状态 M-修改  D-删除 A-添加 U-更新  ?-未知状态  !-警告  C-冲突
* svn add fileName: 如果出现？一般是添加文件时出的问题，在把文件svn add fileName一下后，该文件的状态会改成A
* svn del fileName:如果出现！一般是删除文件时会出现的警告需呀执行svn del fileName命令后，该文件的状态会改成D
* svn ci -m"提交的原因"

在文章的最后呢附上DEMO的GitHub下载地址：[https://github.com/lizelu/MVVM](https://github.com/lizelu/MVVM" \t "_blank)  
（本文作者：青玉伏案）