

页面编写习惯

1. 简介：每个人都有自己的页面编写习惯，下面就自己的编码习惯进行下介绍，后续，会不断进行更新，因为可能后续会学到更好的编码风格，目前主要以 mvvm 思想为基础进行编写：

2. 编码习惯 of cwn

(1).创建viewController.xib(为什么不是storyboard? 如果是storyboard的话svn或git管理开发难以避免冲突)，autolayout或者frame做好界面布局，建立outlet关联，列表设置delegate但不设置datasource为viewController

(2).创建viewControllerTableModel, 该类的基类充当列表数据源, 该类负责两个方面：

(1) 列表网络数据请求或者页面其它网络请求(比如请求广告轮播图片数组)，

(2) 其它业务逻辑处理(比如日期转换，判断某个时间是否在规定时间内)

(3).创建自定义cell，这个cell可以用xib创建(最便捷的方式)；当一个列表cell种类太多的时候，可以考虑用代码创建：基类为viewControllercell，有多少种cell，就创建多少个子类；基类创建方法loadwithdata：又子类去重写，传入参数为id类型的数据，子类可以改其类型为对应使用到的数据模型model；基类声明delegate，子类需要用到代理的，将代理方法声明都写到基类去

(4).创建自定义view，这个类用来封装页面中使用到的，不是cell的视图，如：cell中的一块view(如cell中有九宫格布局，每个格子是一个view内有多个子视图，为了便于

cell布局，将其封装起来，然后for循环创建九宫格)，基类为viewcontrollerview，基类创建方法loadwithdata：由子类去重写，传入参数为id类型的data，子类可以改其类型为对应使用到的数据模型model

(5).注意：一定要遵守一个页面一个列表的原则，避免列表嵌套列表，一个是内存消耗大，一个是controller逻辑复杂，不便代码维护。也就是说：在一个页面一个列表的前提下，关于UI样式多变的，优先考虑以section来区分，不同section建立不同样式的cell即可。如果发现有区样式特别，不是竖直排列的cell，那么别考虑用collectionview，只需通过for循环来创建封装view即可。举个例子，首页，如下所示：



一共有7个区域，每个区域的UI差别都很大，所以，采用一个列表，七个section，有的有sectionHeader(比如增值服务)，有的没有(指数列表)，之所以设置7个区的原因是，首页UI多变，可能一会儿就换个排版了，所以一个个section的划分有利于后期界面板块排列调整。那么问题来了，这个列表的数据源怎么样的？先看下我的ViewModel如下：

数据源数据

```
@property (strong, nonatomic, readonly) NSMutableArray <IconItem *> *section0Data;
@property (strong, nonatomic, readonly) NSMutableArray <StockRealTimeInfo *> *section1Data;
@property (strong, nonatomic, readonly) NSMutableArray <NSString *>*section2Data;
@property (strong, nonatomic, readonly) NSMutableArray <PuBuLiuModle *>*section3Data;
@property (strong, nonatomic, readonly) NSMutableArray <LaoShiYuYueData *>*section4Data;
@property (strong, nonatomic, readonly) NSMutableArray <ToZhiYouDaoModle *>*section5Data;
@property (strong, nonatomic, readonly) NSMutableArray <ZuiXInWenZhanJingAnLi *>*section6Data;
- (id)getItemDataWithIndex:(NSInteger)index;//获取指定section的数据
```

```
- (id)initWithCellIdentifier:(NSString *)aCellIdentifier withHeaderIdentifier:(NSString *)aHeaderIdentifier
configureCellBlock:(MYTableViewCellConfigureBlock)aConfigureCellBlock withCellClass:(NSString *)className {
    if(self = [super initWithCellIdentifier:aCellIdentifier withHeaderIdentifier:aHeaderIdentifier configureCellBlock:
        aConfigureCellBlock withCellClass:className]){
        self.data = [NSMutableArray arrayWithObjects:[NSMutableArray array], [NSMutableArray array], [NSMutableArray
            array], [NSMutableArray array], [NSMutableArray array], [NSMutableArray array], nil];
        self.zhiShuCodes = [NSMutableArray array];
    }
    return self;
}
```

数据源数据请求

```
//Controller列表数据加载
- (void)loadSection0DataWithCompletion:(MYRequestCompleteBlock)completion;//加载主功能网格图标数据，会先发
    是否延期用户请求
- (void)loadSection1DataWithCompletion:(MYRequestCompleteBlock)completion;//加载指数行情数据
- (void)loadSection2DataWithCompletion:(MYRequestCompleteBlock)completion;//加载增值服务数据
- (void)loadSection3DataWithCompletion:(MYRequestCompleteBlock)completion;//加载广告轮播数据
- (void)loadSection4DataWithCompletion:(MYRequestCompleteBlock)completion;//加载找老师轮播数据
- (void)loadSection5DataWithCompletion:(MYRequestCompleteBlock)completion;//加载股学堂数据
- (void)loadSection6DataWithCompletion:(MYRequestCompleteBlock)completion;//加载经典案例数据
```

数据源适配器方法

#pragma mark -重写父类方法

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
    return [self.data count];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
    if(section == 6)
        return [[self.data objectAtIndex:section] count];
    return 1;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath{
    UITableViewCell *cell;
    switch (indexPath.section) {
        case 0:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection0Identifier forIndexPath:indexPath];
            break;
        case 1:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection1Identifier forIndexPath:indexPath];
            break;
        case 2:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection2Identifier forIndexPath:indexPath];
            break;
        case 3:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection3Identifier forIndexPath:indexPath];
            break;
        case 4:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection4Identifier forIndexPath:indexPath];
            break;
        case 5:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection5Identifier forIndexPath:indexPath];
            break;
        case 6:
            cell = [tableView dequeueReusableCellWithIdentifier:kSection6Identifier forIndexPath:indexPath];
            break;
        default:
            break;
    }

    if(self.configureCellBlock){
        self.configureCellBlock(cell, [self.itemAtIndexPath:indexPath], indexPath);
    }
}
```

注意到：第7个section的cell是个数不是1，为啥，因为它是竖直排列的一行行cell，其他的呢，看看UI就知道了，要么是水平排列的，要么是九宫格排列的。当然，这种

用collectionview来做是最快的了，那么问题来了，用collectionview会不会太耗资源了
列表嵌列表不大好是一点，还有一点，我们得明白下列表的意义：使用重用机制加载显示未知个数的数据。也就是说像首页这种好多区都是固定数据，发挥不了列表作用的，个人感觉嵌套列表就是不应该的。相比之下，for循环+懒加载到是个不错的idea来替换嵌套列表的想法。

(6).注意：controller是领导，不做基层工作，什么意思呢？换句话说controller不能有时间判断、网络请求等具体逻辑。本人认为，正确的做法应该是：界面需要显示什么数据，controller让其下某个员工viewModel去取 -> viewModel发起网络请求得到model回调给controller -> controller拿到数据，ok，丢给view，告诉view：数据更新了，这是新数据，你拿去更新下界面把 -> view对新model进行拆解，得到目的数据进行界面更新(如果view嫌controller总是把整个model直接丢过来自己不能用还得做些繁琐的处理，那ok，再招一个打工仔viewModel，让他帮忙拆解)

本人页面编写习惯总结如下：

(1).controller的生命周期->(2).页面UI搭建及数据初始化(config函数)->(3).列表数据请求及数据刷新(refresh函数)->(4).各种Delegate实现，主要目的是为了处理对应view层事件触发->(5).其他事件处理

注意：(1)、(2)相当于页面加载过程 (3)、(4)、(5)涉及数据请求业务逻辑处理等，应该将其主要实现交给viewModel层，整个(controller)主要负责(view)层事件监听->监听到事件触发后命令(viewModel)进行具体事件

处理，得到处理结果(Model)->将结果反馈给view层

[-] <-----页面生命周期----->
[-] M -dealloc
[-] M -viewDidLoad
[-] M -viewWillAppear:
[-] M -viewDidAppear:
[-] M -viewDidDisappear:
[-] <-----页面UI搭建及数据初始化(Config函数)----->
[-] M -configNavigationBar
[-] M -configTable
[-] M -configTimerManager
[-] <-----列表数据请求及数据刷新(refresh函数)----->
[-] M -refreshMessageNoReadNum
[-] M -refreshTable
[-] M -refreshSection:
[-] <-----各种代理方法事件处理----->
[-] UITableViewDelegate
[-] M -tableView:heightForRowAtIndexPath:
[-] M -tableView:heightForFooterInSection:
[-] M -tableView:heightForHeaderInSection:
[-] M -tableView:viewForHeaderInSection:
[-] M -tableView:didSelectRowAtIndexPath:
[-] 经典案例某篇文章点击事件
[-] UINavigationControllerDelegate
[-] M -navigationViewLeftDlegate
[-] 导航左边按钮点击事件
[-] M -navigationViewReghtDlegate
[-] 导航右边按钮点击事件
[-] ViewControllerCellDelegate
[-] M -viewControllerCell_Section0BtnClick:
[-] 九宫格某个按钮点击事件
[-] M -viewControllerCell_Section1BtnClick:
[-] 指数某个指数点击事件
[-] M -viewControllerCell_Section2BtnClick:
[-] 增值服务某个服务点击事件
[-] M -bannerScrollView:didClickScrollView:
[-] 轮播图某张图片点击事件
[-] NSTimerManagerDelegate
[-] M -timerDidFired:
[-] 定时器触发事件
[-] <-----其他事件处理----->
[-] M -tiaoZhuanZengZhiYeMian:withRoomId:withName:withroomId1:
[-] 增值服务跳转事件
[-] M -handleEventAfterRequestIfYouKeNeedZengZhiDown
[-] 游客是否需要增值服务接口请求结束后事件
[-] M -receiveNotificationForTuiChu:
[-] 接收到退出登录推送消息的响应事件
[-] M -receiveNotificationForChangeColor:
[-] 接收到新纸条信息推送消息的响应事件
[-] M -observeValueForKeyPath:ofObject:change:context:
[-] kvo信息未读条数变化监听事件
[-] M -didReceiveMemoryWarning