猿题库 iOS 客户端架构设计

一种基于 MVC 和 MVVM 改进的架构

蓝晨钰 iOS 团队负责人

Model-View-Controller

MVC 优点

易学习

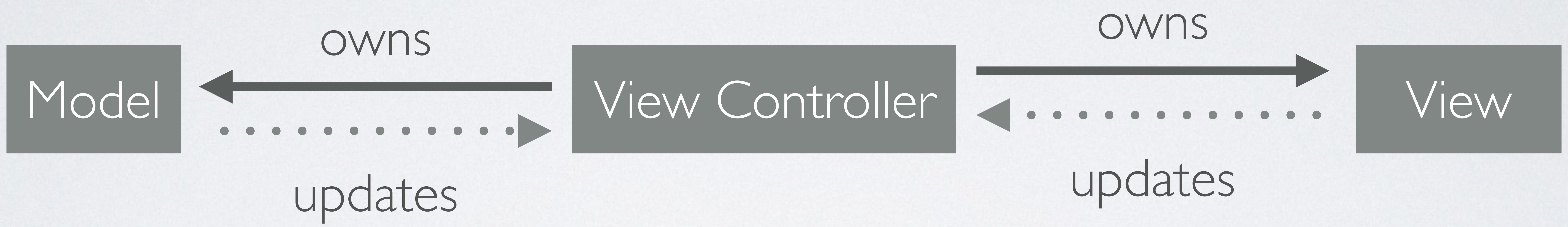
易开发

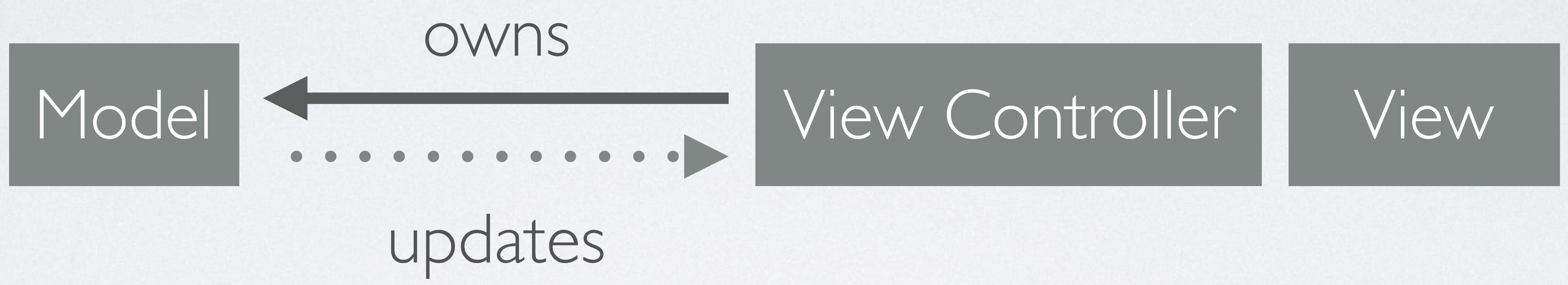
通用成熟

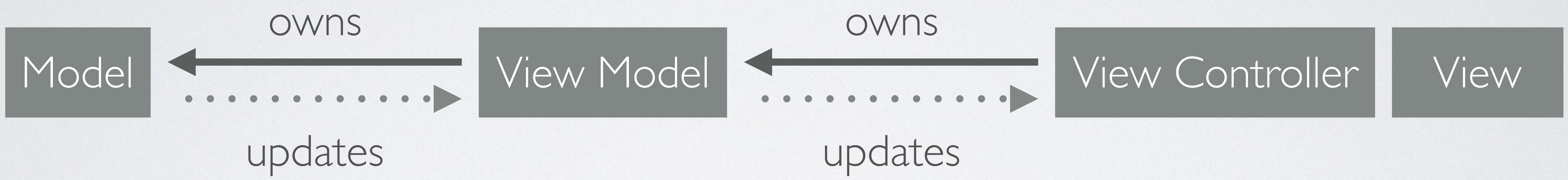
MVC 缺点

Massive View Controller

Model-View-ViewModel


Model
owns
updates
View Controller
owns
updates
View


owns
Model
View Controller
View
updates


owns
owns
Model
View Model
View Controller
View
updates
updates

MVVM 优点

减轻了 VC 的负担

更可测试

强大的绑定机制

MVVM 缺点

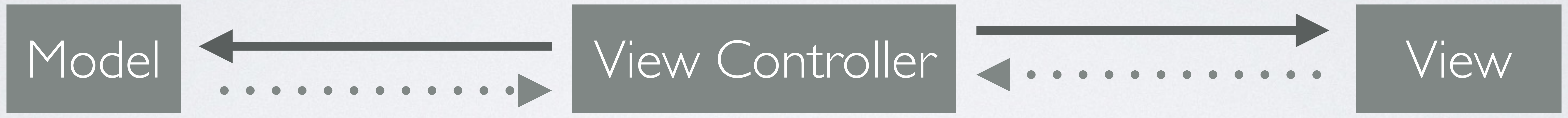
极高的学习成本和开发成本

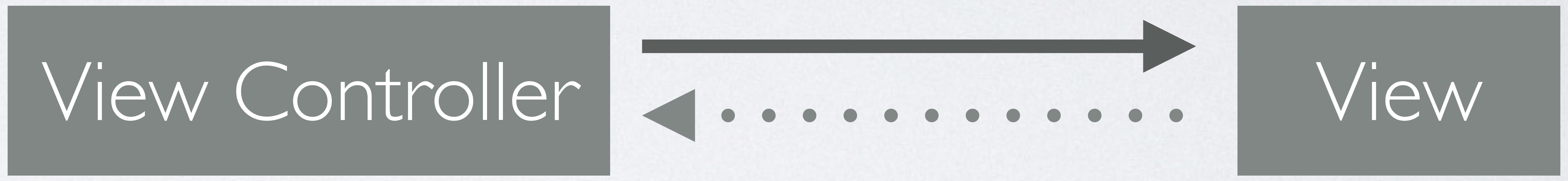
数据绑定使得 Bug 更难调适

View Model 的职责仍然很重

在两种架构中权衡而产生的架构

MVVM without BINDING with DATA CONTROLLER

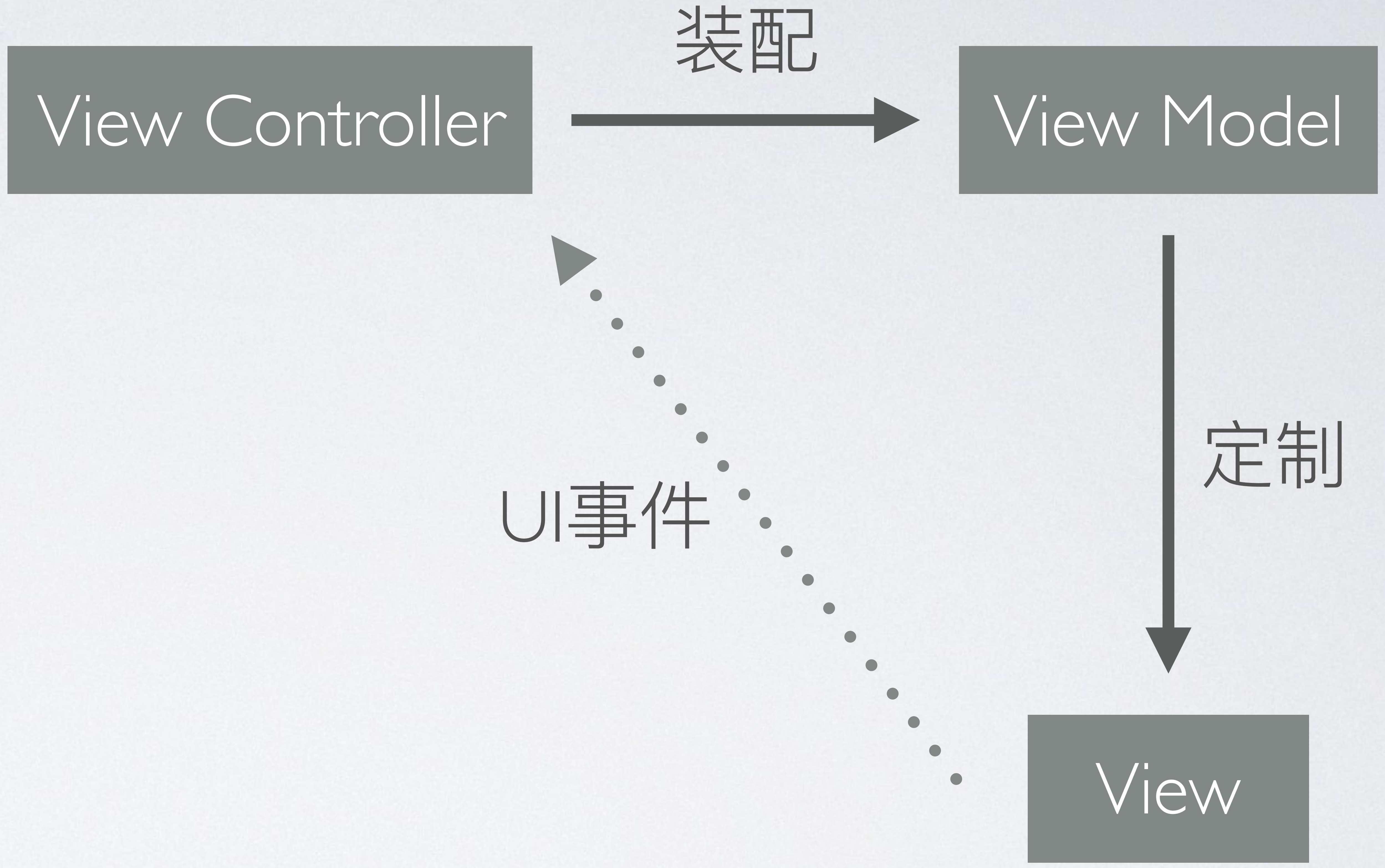

Model
View Controller
View


View Controller
View

每一个V 都有一个对应的VM，V 的数据展示和样式都由其定制

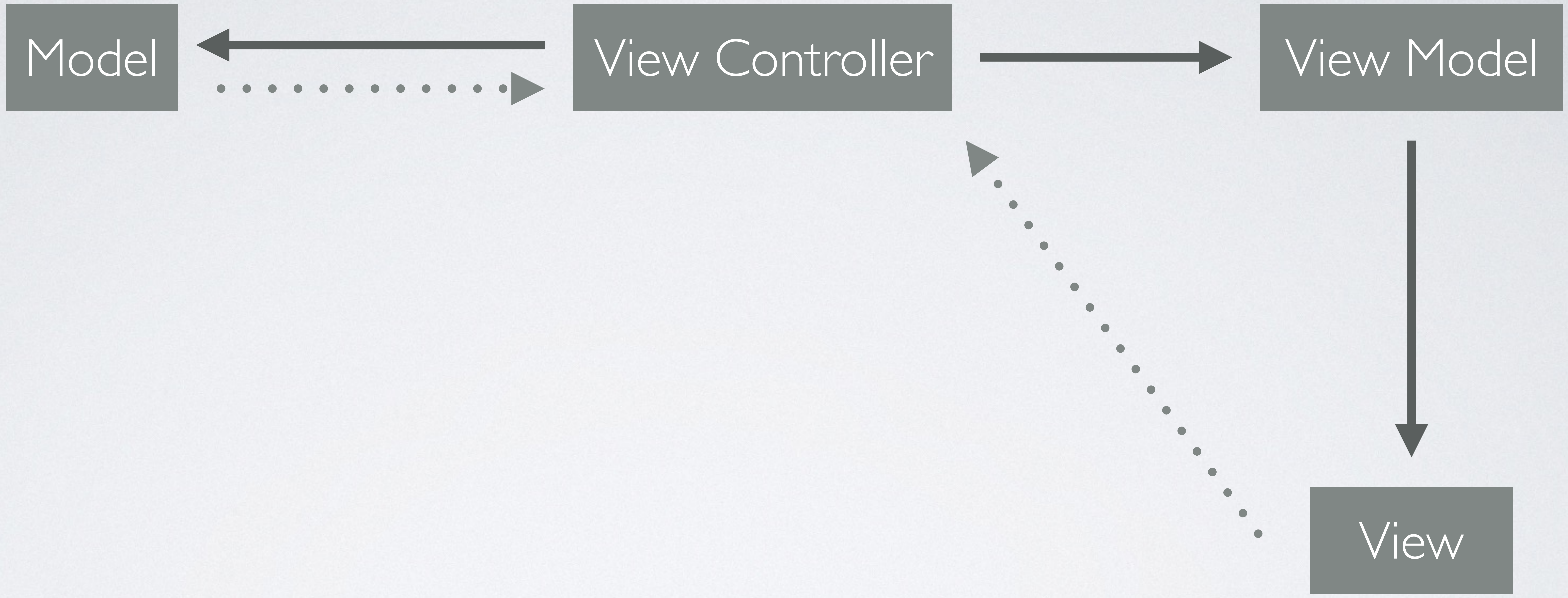
不引入双向绑定机制或观察机制，而是通过传统的代理回调或通知将UI事件传给外界

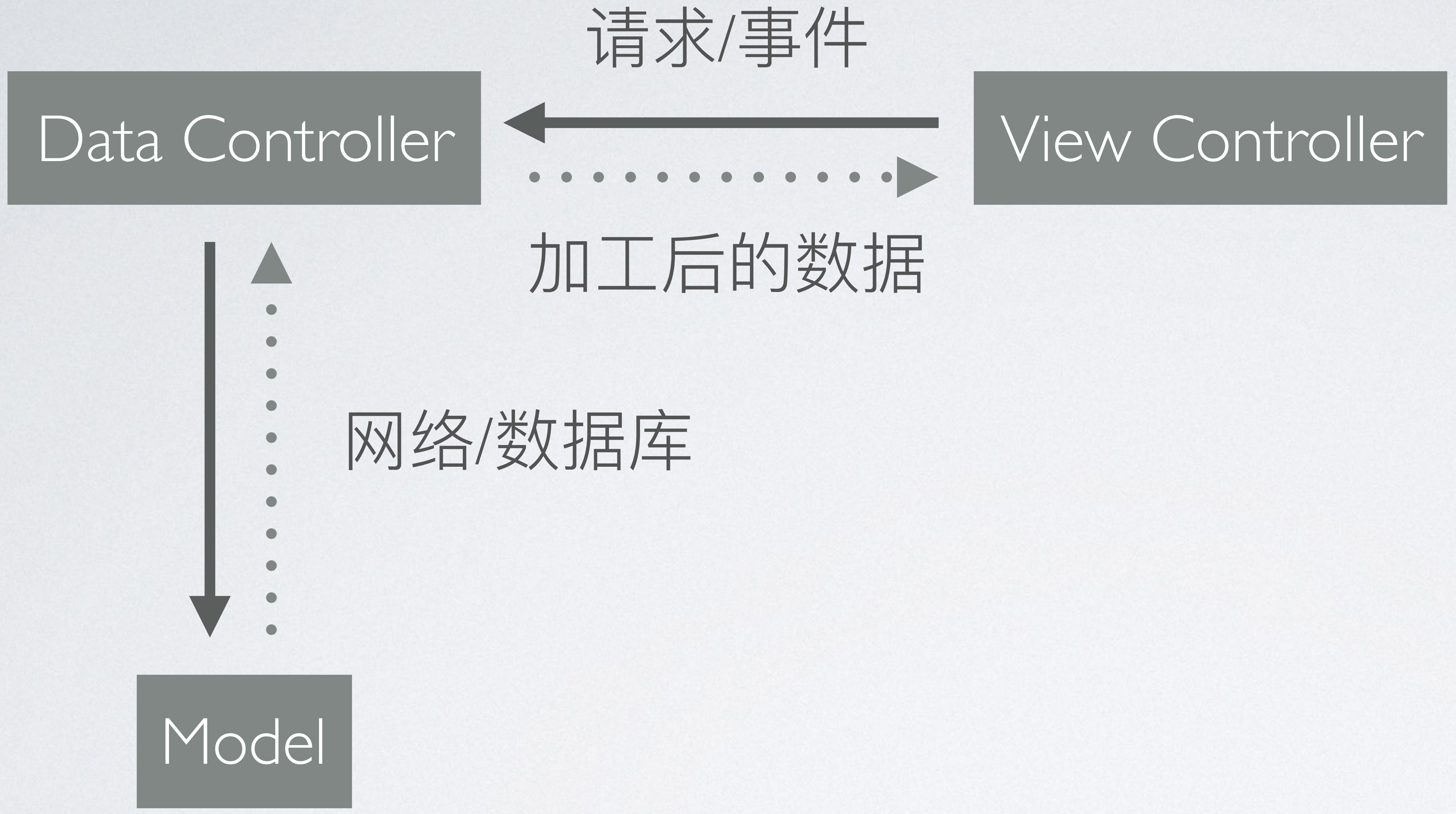
VC只负责将VM装配给V，接受UI事件



好处

- View 可以完全解耦，只需要确定好 View Model 和回调接口即可
- View Controller 层可以尽可能少的和 View 的具体表现打交道，将这部分职责转给了 View Model，减轻了 View Controller 的负担
- 使用传统的回调机制，学习成本低，数据和事件流入和流出易观察和可控，降低维护和调试成本


Model
View Controller
View Model
View



将处理数据和获取数据的职责从传统 MVVM 的 VM 中抽离出来，成为 DC

VC 请求数据和将一些数据修改的事件（可以是UI事件触发）传递给 DC

DC 接收到 VC 的请求后，向 M 获取数据和更新数据，并将加工后的数据返回

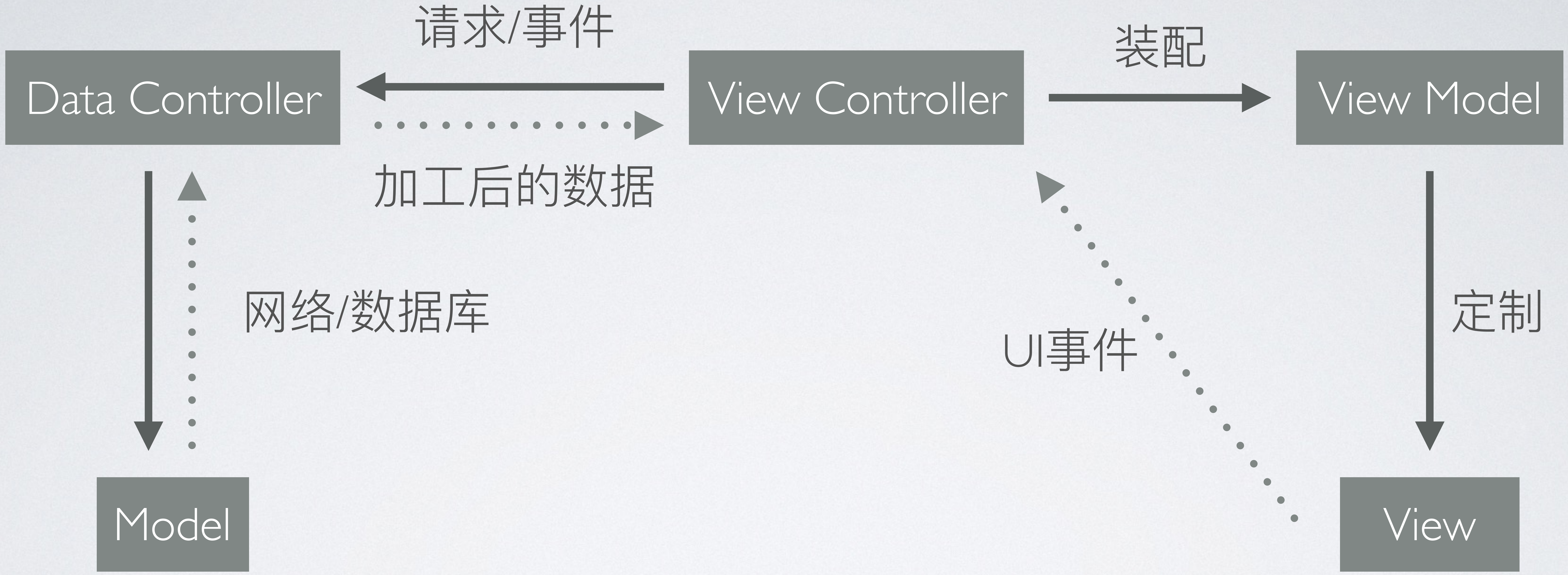
DC 还负责网络层和持久层的逻辑

好处

- 避免了传统 MVVM 架构 VM 层有可能变得臃肿的情况，更清晰的模块职责

- 业务逻辑解耦，数据的加工和处理都放在 Data Controller 中，View Controller 不再关心数据如何获得，如何处理，Data Controller 不再需要关心界面如何展示，如何交互

- Data Controller 由于界面无关，所以可以有更好的可测试性和可复用性

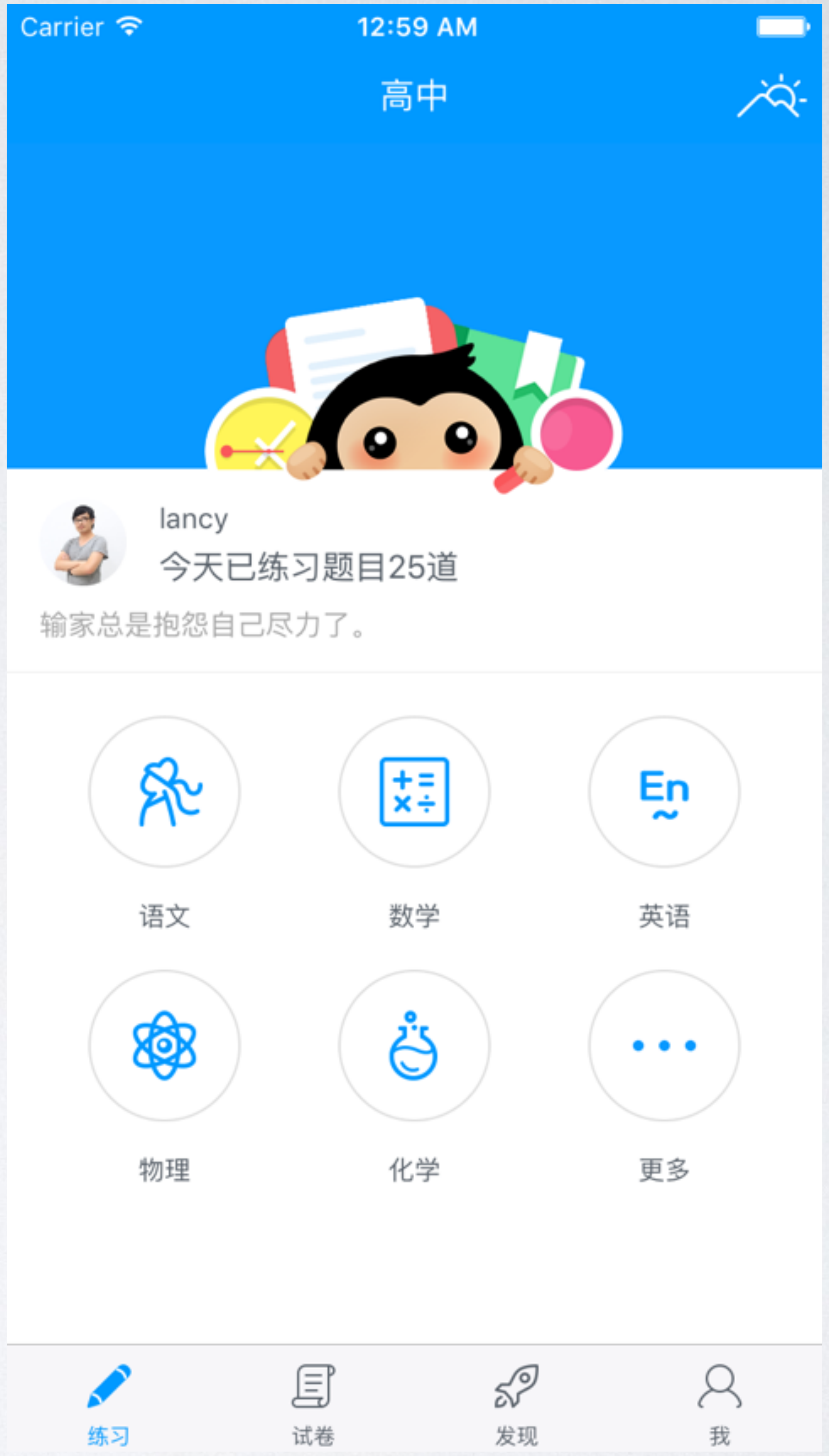

请求/事件
Data Controller
View Controller
装配
View Model
加工后的数据
网络/数据库
定制
UI事件
Model
View

"Talk is Cheap, Show me the Code."

–Linus Torvalds

怎么做?

猿题库的主页面


Carrier
12:59 AM
高中
lancy
今天已练习题目25道
输家总是抱怨自己尽力了。
语文
数学
英语
物理
化学
更多
练习
试卷
发现
我

View Controller

- 每一个 View Controller 会有一个对应的 Data Controller
- 把界面拆分成几个单独的 View： BannerView， Activity View， SubjectView

```
@interface APEHomePracticeViewController () <APEHomePracticeSubjectsViewDelegate,
APEBannerCycleImageViewDelegate, APECoverAdViewDelegate, APEHomePracticeActivityViewDelegate>

@property (nonatomic, strong, nullable) UIScrollView *contentView;

@property (nonatomic, strong, nullable) APEHomePracticeBannerView *bannerView;
@property (nonatomic, strong, nullable) APEHomePracticeActivityView *activityView;
@property (nonatomic, strong, nullable) APEHomePracticeSubjectsView *subjectsView;

@property (nonatomic, strong, nullable) APEHomePracticeDataController *dataController;

@end
```

View Controller

- 在 viewDidLoad 的时候，初始化各个 View，并设置好布局

```
- (void)setupContentView {
   self.contentView = [[UIScrollView alloc] init];
   [self.view addSubview:self.contentView];

    self.bannerView = [[APEHomePracticeBannerView alloc] init];
    self.bannerView.cycleImageView.delegate = self;
    self.activityView = [[APEHomePracticeActivityView alloc] init];
    self.activityView.delegate = self;
    self.subjectsView = [[APEHomePracticeSubjectsView alloc] init];
    self.subjectsView.delegate = self;

    [self.contentView addSubview:self.bannerView];
    [self.contentView addSubview:self.activityView];
    [self.contentView addSubview:self.subjectsView];

    // Layout Views
    ...
}
```

以 SubjectView 为例

- 向 DataController 请求 Subjects 的数据
- 请求完成后，用获得的数据生成 ViewModel，并将其装配给 SubjectView

```
- (void)fetchSubjectData {
    [self.dataController requestSubjectDataWithCallback:^(NSError *error) {
        if (error == nil) {
            [self renderSubjectView];
        }
    }];
}

- (void)renderSubjectView {
    APEHomePracticeSubjectsViewModel *viewModel =
        [APEHomePracticeSubjectsViewModel viewModelWithSubjects:self.dataController.openSubjects];
    [self.subjectsView bindDataWithViewModel:viewModel];
}
```

Subject 相关数据结构

- APESubject：学科，包含 id 和 name 等属性
- APEUserSubject：用户学科信息，包含用户是否开启某个学科等属性

```
@interface APESubject : MTLModel<MTLJSONSerializing>

@property (nonatomic, strong, nullable) NSNumber *id;
@property (nonatomic, strong, nullable) NSString *name;

@end

@interface APEUserSubject : MTLModel <MTLJSONSerializing>

@property (nonatomic, strong, nullable) NSNumber *id;
@property (nonatomic, strong, nullable) NSNumber *updatedTime;
///  On or Off
@property (nonatomic) APEUserSubjectStatus status;

@end
```

Data Controller

- 每一个 ViewController 有一个对应的 DataController，包含了这个页面上所有数据相关逻辑，我们称其为 View Related Data Controller
- 为了显示 SubjectView 需要一个用户开启的科目列表，定义为 openSubjects
- 定义一个接口请求这个数据

```
@interface APEHomePracticeDataController : APEBaseDataController

///  Subjects that are open with current phase.
@property (nonatomic, strong, nonnull, readonly) NSArray<APESubject *> *openSubjects;

///  Request subject data and call callback when finished.
///
///  @param callback Completion callback block.
- (void)requestSubjectDataWithCallback:(nonnull APECompletionCallback)callback;

@end
```

- DataController 可以复用更小的 DataController，通常只包含纯粹的 Model 相关逻辑，例如网络请求，数据库请求，或是基本的数据加工。我们称其为 Model Related Data Controller

- 这类 DataController 经常提供正交的数据。例如 SubjectDataController，提供了所有的 allSubjects （APESubject类） 和用户开启的 userSubjects （APEUserSubject类） 。将这些正交数据加工成界面最终需要的数据 openSubjects （APESubject类）

```
@interface APEHomePracticeDataController ()
@property (nonatomic, strong, nonnull) APESubjectDataController *subjectDataController;
@end

@implementation APEHomePracticeDataController
- (void)requestSubjectDataWithCallback:(nonnull APECompletionCallback)callback {
    APEDataCallback dataCallback = ^(NSError *error, id data) {
        callback(error);
    };
    [self.subjectDataController requestAllSubjectsWithCallback:dataCallback];
    [self.subjectDataController requestUserSubjectsWithCallback:dataCallback];
}
@end

- (nonnull NSArray<APESubject *> *)openSubjects {
    return self.subjectDataController.openSubjectsWithCurrentPhase ?: @[];
}
```

View Model

- 每个 View 都会有一个对应的 View Model
- View Model 包含了展示这个 View 所需要的所有数据
- 用工厂方法来创建 View Model，这个方法不再需要关心传递的是所有的 Subjects 还是用户开启的 Subjects

```
@interface APEHomePracticeSubjectsViewModel : NSObject

@property (nonatomic, strong, nonnull) NSArray<APEHomePracticeSubjectsCollectionCellViewModel *>
*cellViewModels;
@property (nonatomic, strong, nonnull) UIColor *backgroundColor;

+ (nonnull APEHomePracticeSubjectsViewModel *)viewModelWithSubjects:(nonnull NSArray<APESubject *>
  *)subjects;

@end
```

View Model

- View Model 可以包含子 View Model，就像 View 可以有 Subview

- SubjectView 内部由 UICollectionView 实现，将 Cell 也对应的设计一个 View Model

```
@interface APEHomePracticeSubjectsCollectionCellViewModel : NSObject

@property (nonatomic, strong, nonnull) UIImage *image;
@property (nonatomic, strong, nonnull) UIImage *highlightedImage;
@property (nonatomic, strong, nonnull) NSString *title;
@property (nonatomic, strong, nonnull) UIColor *titleColor;
@property (nonatomic, strong, nonnull) UIColor *backgroundColor;

+ (nonnull APEHomePracticeSubjectsCollectionCellViewModel *)viewModelWithSubject:(nonnull
APESubject *)subject;
+ (nonnull APEHomePracticeSubjectsCollectionCellViewModel *)viewModelForMore;

@end
```

View

- 定义好装配 ViewModel 的接口
- 定义好 UI 回调事件

```
@protocol APEHomePracticeSubjectsViewDelegate <NSObject>

- (void)homePracticeSubjectsView:(nonnull APEHomePracticeSubjectsView *)subjectView
             didPressItemAtIndex:(NSInteger)index;

@end

@interface APEHomePracticeSubjectsView : UIView

@property (nonatomic, strong, nullable, readonly) APEHomePracticeSubjectsViewModel *viewModel;
@property (nonatomic, weak, nullable) id<APEHomePracticeSubjectsViewDelegate> delegate;

- (void)bindDataWithViewModel:(nonnull APEHomePracticeSubjectsViewModel *)viewModel;

@end
```

View

- 使用 View Model 的数据来渲染界面
- Subview 也可以使用 View Model

```
- (void)bindDataWithViewModel:(nonnull APEHomePracticeSubjectsViewModel *)viewModel {
    self.viewModel = viewModel;
    self.backgroundColor = viewModel.backgroundColor;
    [self.collectionView reloadData];
    [self setNeedsUpdateConstraints];
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView cellForItemAtIndexPath:
(NSIndexPath *)indexPath {
    APEHomePracticeSubjectsCollectionViewCell *cell = [collectionView
dequeueReusableCellWithReuseIdentifier:@"Cell" forIndexPath:indexPath];
    if (0 <= indexPath.row && indexPath.row < self.viewModel.cellViewModels.count) {
        APEHomePracticeSubjectsCollectionCellViewModel *vm =
self.viewModel.cellViewModels[indexPath.row];
        [cell bindDataWithViewModel:vm];
    }
    return cell;
}
```

View Controller

```
@interface APEHomePracticeViewController () <APEHomePracticeSubjectsViewDelegate,
APEBannerCycleImageViewDelegate, APECoverAdViewDelegate, APEHomePracticeActivityViewDelegate>

@property (nonatomic, strong, nullable) UIScrollView *contentView;

@property (nonatomic, strong, nullable) APEHomePracticeBannerView *bannerView;
@property (nonatomic, strong, nullable) APEHomePracticeActivityView *activityView;
@property (nonatomic, strong, nullable) APEHomePracticeSubjectsView *subjectsView;

@property (nonatomic, strong, nullable) APEHomePracticeDataController *dataController;

@end
```

```
- (void)fetchSubjectData {
    [self.dataController requestSubjectDataWithCallback:^(NSError *error) {
        if (error == nil) {
            [self renderSubjectView];
        }
    }];
}

- (void)renderSubjectView {
    APEHomePracticeSubjectsViewModel *viewModel =
        [APEHomePracticeSubjectsViewModel viewModelWithSubjects:self.dataController.openSubjects];
    [self.subjectsView bindDataWithViewModel:viewModel];
}
```

总结

层次清晰，职责明确，耦合度低，复用性高，测试性高

低学习成本，低开发成本

高实施性，无需整体重构

Q&A
THANKS

国内最有价值的在线教育公司 **猿题库** 还在持续招人中

Web前端、服务器端、iOS、Android、Windows、算法研究、音视频、运维等

蓝晨钰 lancy@fenbi.com