とりあえず投稿機能に ViewModelを 使ってみるための知識

yuzushioh

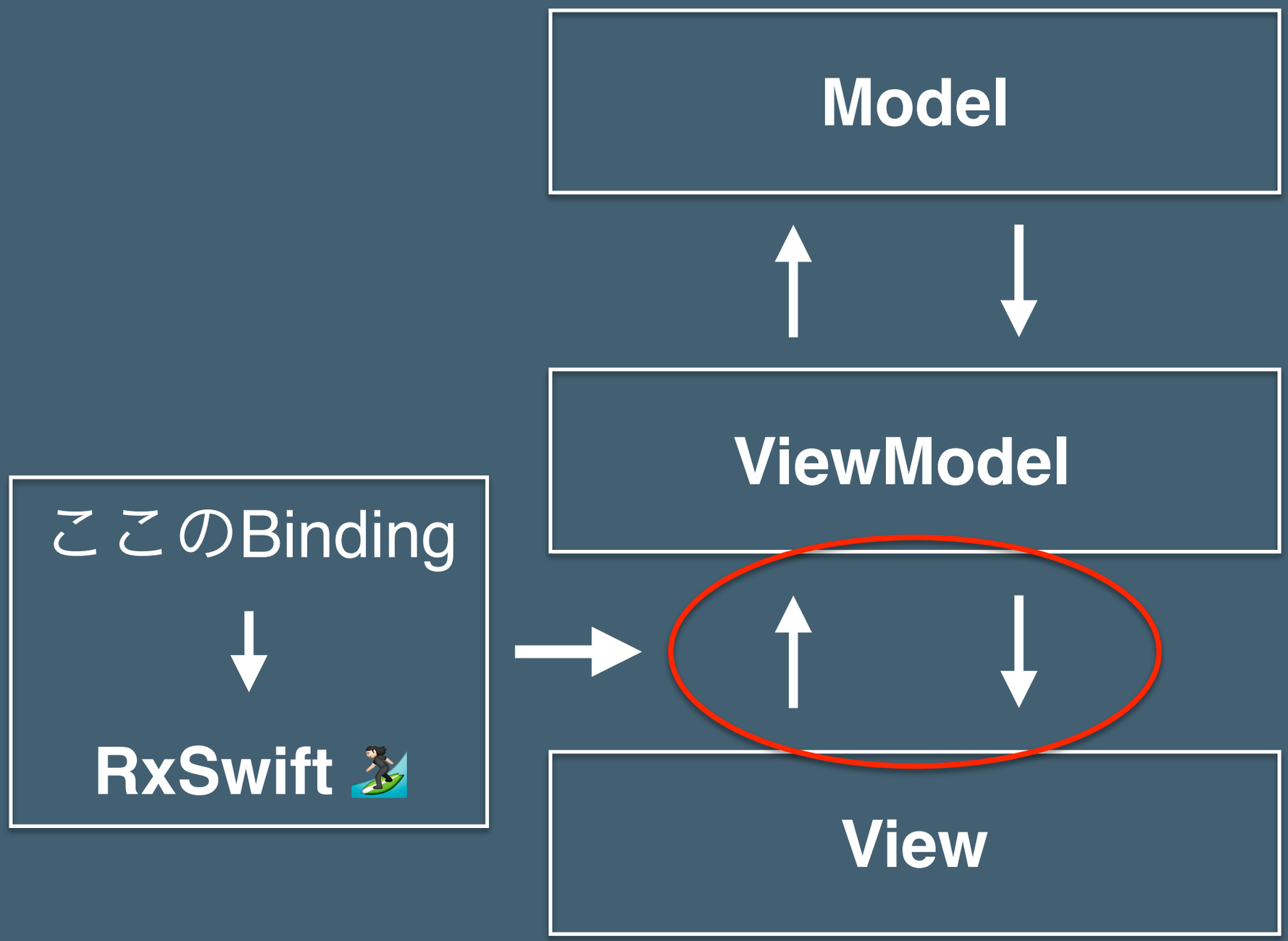
自己紹介

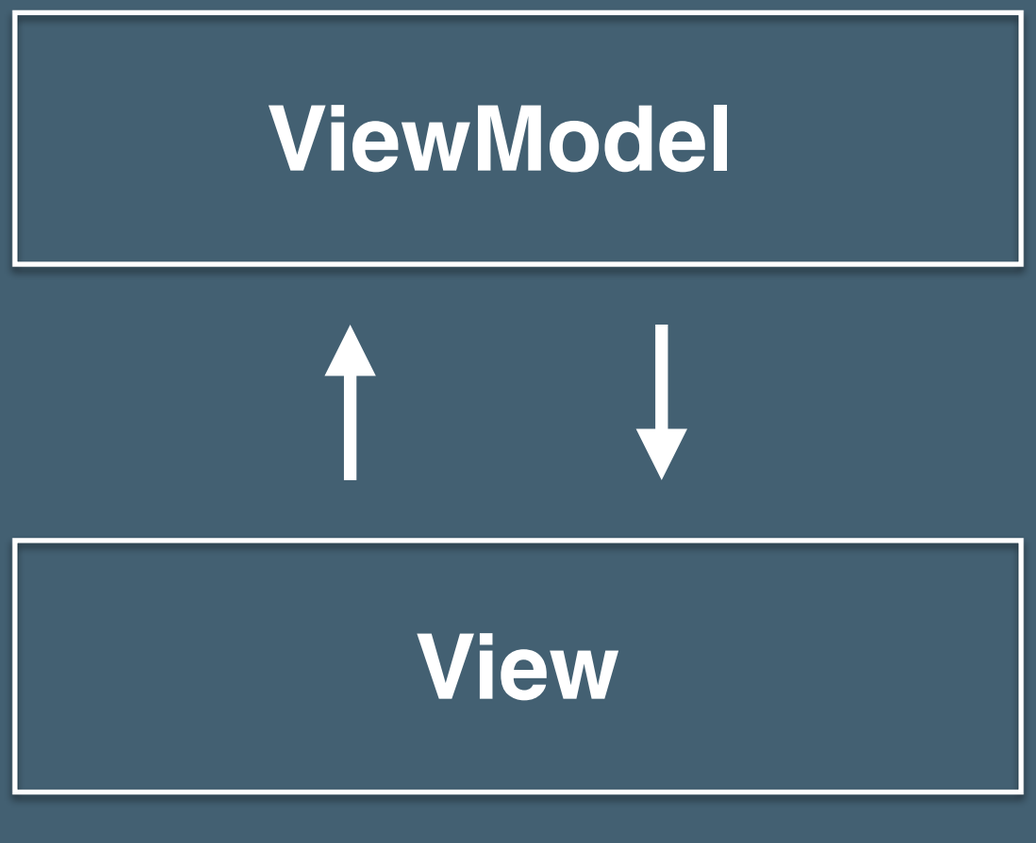
- 福田 涼介 (yuzushioh)
- 2016年1月にソウゾウに入社
  (インターン)
- 19歳の大学２年生(Swift歴１年)

.retry( )とcatchError( )を続けてます🏄🏻

とりあえずViewModelを使ってみるための知識

MVVM?





UIは更新されていく

RxSwift: UIの反映を自動化🏄‍♀️!

状態管理はViewModel🏄‍♀️

投稿機能にViewModelを使って見る 


au
16:51
100%
あげます・売ります
画像必須。タップして編集、長押しして順序を入れ替え
商品名（必須 40文字まで）
0/40
募集内容の説明（必須 1,000文字まで）
色、素材、重さ、定価、状態、注意点など
0/1000
投稿例
詳細カテゴリー
必須
価格
￥0


au
16:50
100%
あげます・売ります
画像必須。タップして編集、長押しして順序を入れ替え
ピアンキ譲ります
8/40
去年購入したピアンキを安く譲ります。購入以来ほぼ使用していないので、状態はかなり良いと思います。タイプは乗りやすいフラットバータイプです。
先日の別タイプのものを購入するなど僕自身ピアンキ大好きなので、同じように好きな方に乗っていただけるととてもうれしいです＼(^o^)／
137/1000
投稿例
詳細カテゴリー
バイク
価格
￥0

手順🏄‍♀️

１. 投稿の要素はViewModelで管理

２. UIの更新はViewModelを反映して自動的に

３. ViewModelにTriggerを与える

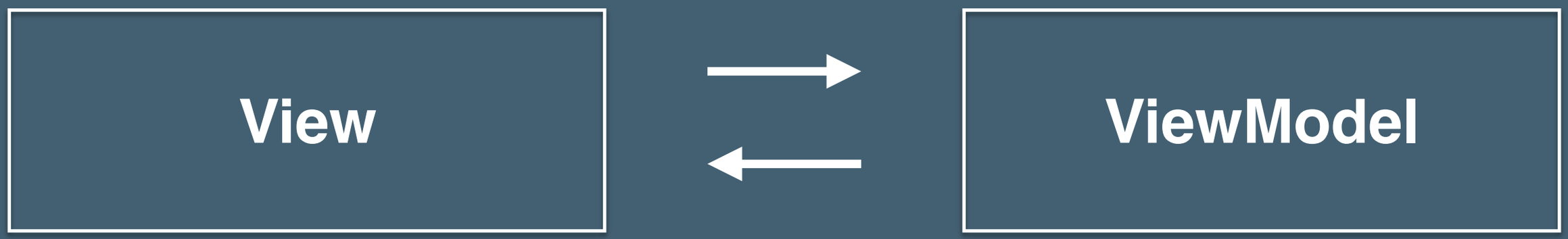
４. ViewModelからのoutputとerrorをViewで処理

PostViewModel🏄のゴール

手順１・２

UIに入力された値のストリーム

(TitleText, DescriptionText…)



画面の状態を表すストリーム

postButtonEnabled: Observable<Bool>

textViewPlaceholderHidden: Observable<Bool>

1. 投稿の要素はViewModelで管理

```
class PostViewModel {
    let postTrigger = PublishSubject<Void>()
    let requestCompleted = PublishSubject<Post>()
    let error = PublishSubject<ErrorType>()

    let title = PublishSubject<String>()
    let description = PublishSubject<String>()
    let category = PublishSubject<Category>()
    let price = PublishSubject<Int>()
    let image = PublishSubject<UIImage>()

    var loading: Observable<Bool>
    var textViewPlaceholderHidden: Observable<Bool>
    var postButtonEnabled: Observable<Bool>
}
```

2. UIの更新はViewModelを反映して自動的に

```
var textViewPlaceholderHidden: Observable<Bool> {
    return description.map { !$0.isEmpty }
}

var postButtonEnabled: Observable<Bool> {
    return Observable
        .combineLatest(title, description, category, price, image) { title, description, category, price, image -> Bool in
            return !title.isEmpty && !description.isEmpty && price != 0
        }
        .startWith(false)
}
```

```
private var postRequest: Observable<PostService.PostRequest> {
    return Observable
        .combineLatest(title, description, category, price) { title, description, category, price in
            return PostService.PostRequest(
                title: title,
                description: description,
                category: category.id,
                price: price,
                mediaId: ""
            )
        }
}
```

2. UIの更新はViewModelを反映して自動的に

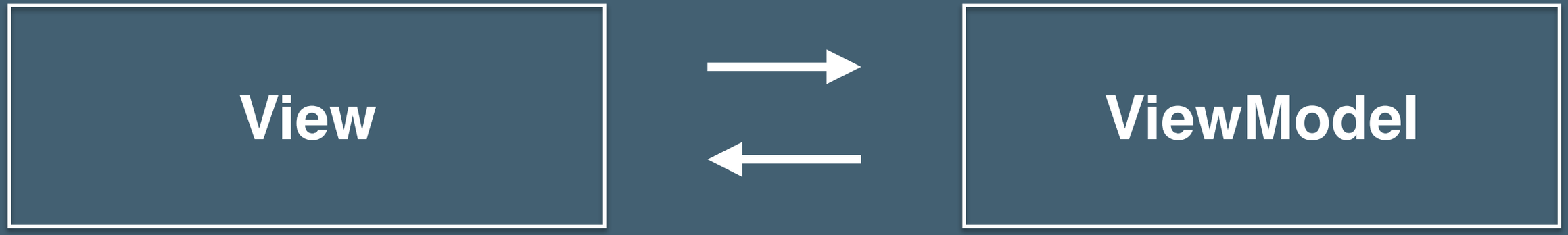
```
private func bindUI() {
    viewModel.postButtonEnabled
        .bindTo(postButton.rx_enabled)
        .addDisposableTo(disposeBag)

    viewModel.textViewPlaceholderHidden
        .bindTo(descriptionPlaceholder.rx_hidden)
        .addDisposableTo(disposeBag)

    titleTextField.rx_text
        .bindTo(viewModel.title)
        .addDisposableTo(disposeBag)

    viewModel.title
        .bindTo(titleTextField.rx_text)
        .addDisposableTo(disposeBag)
}
```

PostViewModel🏄のゴール

手順３・４

投稿に必要な値

postTrigger: Observable<Void>



error: Observable<ErrorType>

loading: Observable<Bool>

.....

３. ViewModelにTriggerを与える

APIリクエストをバインドする

Triggerを元に画像をUP

↓

取得したmedialdをリクエストの
パラメーターに追加

↓

リクエストを送りレスポンスを取得

↓

requestCompletedをonNext()

3. ViewModelにTriggerを与える

実際のコード！

```
private func bindPostRequest() {
    let mediaId = postTrigger
        .withLatestFrom(image)
        .flatMap { image in
            // ここで画像をAPIに送り、idを受け取る
            return Observable.just("media id comes here")
        }

    let request = mediaId
        .withLatestFrom(postRequest) { $0 }
        .map { id, request -> PostService.PostRequest in
            var request = request
            request.mediaId = id

            return request
        }
        .shareReplay(1)

    let response = request
        .flatMap { request in
            return Session.sharedSession.rx_responseFrom(request)
                .doOnError { [weak self] error in
                    self?.error.onNext(error)
                }
                .catchError { _ in Observable.empty() }
        }
        .shareReplay(1)

    response
        .bindTo(requestCompleted)
        .addDisposableTo(disposeBag)
```

４. ViewModelからのoutputとerrorをViewで処理

通信中は上がonNext🏄‍♀️され

投稿成功時は中がonNext🏄‍♀️され、

エラーの場合は下がonNext🏄‍♀️される

```
viewModel.loading
    .subscribeNext { loading in
        // ここでこのloadingをindicatorViewなどのrx_animatingなどにbindする。
    }
    .addDisposableTo(disposeBag)

viewModel.requestCompleted
    .subscribeNext { post in
        // 投稿が成功した時の処理を行う
    }
    .addDisposableTo(disposeBag)

viewModel.error
    .subscribeNext { error in
        // 投稿の通信でerrorが出てしまった場合の処理を行う
    }
    .addDisposableTo(disposeBag)
```

おさらい🏄🏻

1. 投稿の要素はViewModelで管理

2. UIの更新はViewModelを反映して自動的に

3. ViewModelにTriggerを与える

4. ViewModelからのoutputとerrorをViewで処理

+α

既存の投稿の編集の際も同じ

PostViewModelを使う🏄

既存のElementの値をonNext( )🏄‍♀️

ViewとViewModelで双方Bindingする

```
class PostViewModel {

    init(post: Post?) {
        setDefaultValueFromPost(post)
        bindPostRequest()
    }

    private func setDefaultValueFromPost(post: Post?) {
        guard let post = post else { return }

        title.onNext(post.title)
        description.onNext(post.description)
        category.onNext(Category.findCategoryWithId(post.category))
        price.onNext(post.price)
    }
```

是非参考にしてみてください🏄🏻‍♀️

レポジトリー

https://github.com/yuzushioh/sandbox/tree/master/RxSwift%2BPostViewModel

RxSwiftのオペレーターサンプル

https://github.com/yuzushioh/RxOperator