第11回　CyberAgent21卒内定者iOS輪読会

by @ostk0069

今回は番外編です

The Composable Architecture について行います

今回の話では The Composable Architectureは長いので
本家でもよく出てくるTCAとして記すこととします

発表者はTCAについてプロジェクトとととしての運用経験はないのでマサカリをする際には十分に配慮した上で行っていただけると助かります

今回話すこと

- 理解する上で必要な知識
- 主な概要や誕生経緯
- 5つの特徴
- 登場するClassの役割
- 導入することのメリット・デメリット
- サンプルコードを見る

今回話さないこと

- 長期的に使うことのメリデメ
- 細かい圏論的な話

僕の経験はry-ittoとハッカソンの時に使ったことがあるぐらいです

理解する上で必要な知識

必要な知識

- Combine(UIKit+RxSwiftもある一応ある)
- DI
- iOSで用いる設計ほぼ全て
    - 特にReduxやFlux

あると理解がはやいもの

- Elm Architecture
- Vuex

比較的考えが似ている、または開発者がそれにinspireされて作ってる

主な概要

二人の有名な開発者によって作られた

- Brandon Williams
- Stephen Celis

どちらもKick Starterのmain contributerとして有名。snapshot testingとかも作ってる。

動画が多数存在するのでとりあえず見てない人は見た方がいいかも

- https://www.pointfree.co/collections/composable-architecture/a-tour-of-the-composable-architecture/ep100-a-tour-of-the-composable-architecture-part-1

TCAの理解以外にも色々考え方を学べそうなコンテンツ(英語レベル高め)

5つの特徴

- State Management
- Composition
- Side Effect
- Testable
- Ergonomics

State Management

- 状態管理をReduxやFluxのように行う

Composition

- コンポーネント思考
  - FatなReducerが作成されない(Reducerの分割)

Side Effect

- サービスにおける副作用に耐えうる設計になってる
  - 副作用(apiのfetch, 遅延処理とか)
    - (PublisheのPublisherが内部で作成されてる)
  - CombineのPublisherに準拠してるため実現できてる

Testable

- 書くべきテストコードの明確化
  - Action -> Reducerをテストすることがメインになる
- サンプルコードも豊富

Ergonomics

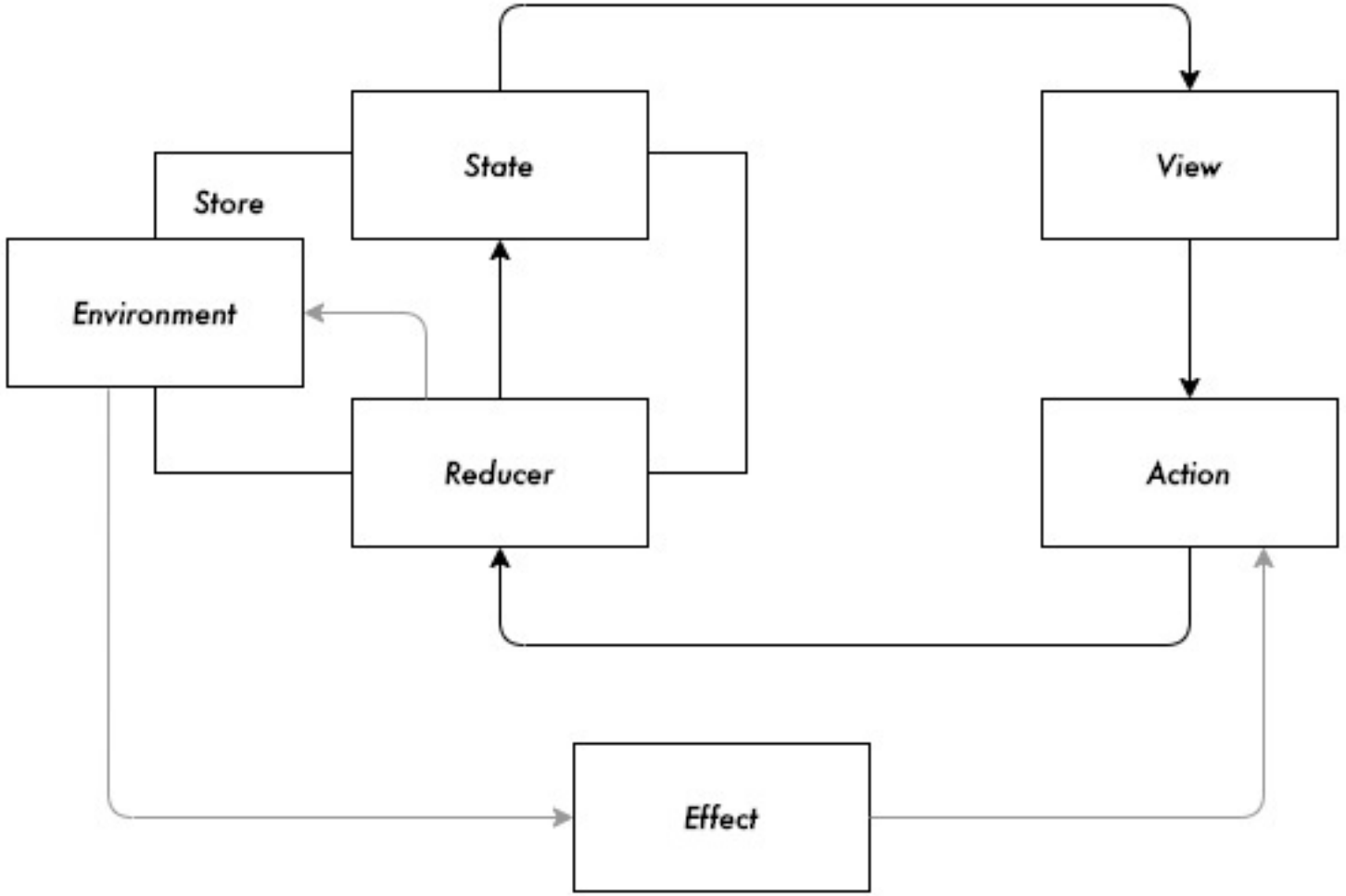
- 簡略化(?)
    - 僕はまだこれを実感できてない

登場するClassの役割

- **Action**
  - enumで定義されたユーザーのアクション
- **Reducer**
  - Actionを受け取りEffectを呼び出すorStateに流す役割
- **Effect**
  - CombineのPublisher
  - Actionへ接続する
- **State**
  - 受け取った値をViewに反映させる
- **Store**
  - Action, Reducer, Effect, Stateの集合体
- **Environment**
  - Dependencyを保持する

Build an iTunes Search App with SwiftUI and The Composable Architecture

by Glenn Gonda



導入することのメリット・デメリット

メリット

- 状態管理の明確な責務の切り分け
  - 設計に関する議論がチームで行いやすくなる
- modular architectureの適用のしやすさ
- 採用につながる
- Combine理解してなくてもなんとなくで書き始められる
- テストを書くことの難易度が下がる

デメリット

- 設計として制約が多い
- ライブラリとしての依存をしないといけない(自作でも問題ない)
  - Combineをラップしてるのでの理解とiOS13以上ではないと使えない
    - UIKitのサンプルも存在する
- 実務でやるのであればCombineの理解(ソースコードの理解)は必須
- 別の設計への乗り換えはそこそこしんどそう
- 長期プロジェクトになることが決まってないサービス
- チームのレベル感次第では崩壊する
  - 特性を理解せずに導入するにはむしろマイナスが発生する可能性も

サンプルコードを見る

- **Basic Usage**
  - https://github.com/pointfreeco/swift-composable-architecture#basic-usage
  - Effectがどんな感じか理解しやすい
  - TestStoreも見てみると良さそう
- **Todos**
  - https://github.com/pointfreeco/swift-composable-architecture/tree/main/Examples/Todos
  - 親コンポーネントと子コンポーネントの関係性が見れるので良さそう

- **WithViewStore**
  - SwiftUIのView
  - storeを初期値に持ち、viewStoreに変換する
- **ViewStore**
  - Observed Object
  - ViewStoreを購読してViewを変更できる

その他

結構他にも似たような設計を提唱しているものはある

- Harvest
- bow-arch
- VueFlux

質問 & 議論

次回は 第13章 & CAの設計 です！