SwiftUIアプリ設計をReduxを使って開発する(Reduxの特徴をおさらいする)

♡ 6
masa7351
2019/10/17 10:40

#Redux #SwiftUI #Swift #プログラミング

はじめに

SwiftUI発表を見たときは、Swiftの発表より大きな衝撃を受けました。発表翌日はSwift界隈はザワザワしていました。長らくStoryboardと格闘していた我々iOSエンジニアとしては悲願だった技術なのではないでしょうか。
ただし、残念ながらSwiftUIを使ったアプリは、iOS13以降のみサポートするという、大きな制限があるため、本格的に使われ始めるのは2年以上時間を要すると言われております。

SwiftUIはレイアウト作成の技術のためフォーカスされておりますが、一緒に発表された、Combine Framework (※動画) と合わせて活用することが前提の作りとなっております。今回はこのCombine Frameworkに焦点を当てて記事を書きたいと思います。

今ではiOSアプリは、MVC、MVP、MVVM、Flux、Redux、Clean Architecture(VIPPER) 、Micro View Controller ...etc などの様々なアーキテクチャーを使い開発されてきておりますが、多くの現場ではMVCでのアプローチでの開発を行っております。

SwiftUI + Combine Framework の発表は、MVCからMVVMへのAppleから開発者に向けて、「2年後には、開発のメインストリームがMVVMに移るからしっかり勉強しておいてね」といったメッセージのように感じました。
MVC一択脳の方は、MVVM脳を構築する必要があります。

MVVMの記事を書こうとも考えましたが、既に世の中にSwiftUIを使ったMVVMの良質な記事をいくつか目にするのでそれは見送ります。

技術書やネット記事などで目にしたこともあるかもしれませんが、MVC開発で起こっている大きな問題の一つに、ViewControllerが肥大化する問題(Fat / Massive ViewController)というものがあります。MVVMを齧ったぐらいの知識で構築すると、Massive ViewModel化するように思えてなりません。

近年 iPhoneアプリのAndroid化が止まらないと言われるように、昔のiPhoneでは想像できないほど、iPhoneの高機能化（裏を返すと操作が難しくなってきた)してきております。

アプリを取り巻くOSやハードウェアも、iPhone端末サイズのバリエーションの増加、iPad OSの発表、Split View ...etc のように変化してきております。

データ更新のトリガーも色々あり、画面遷移、ユーザーのボタンタップ、フォアグラウンド復帰、通信状態の変化、Push通知による遷移、ポーリング、 ...etc

MVVMでこれからの問題を解決しようとするとINとOUTをしっかり設計した上で実装しないと複雑化し、保守不能な負債コード（従来のMVCより技術難易度が高すぎる分より厳しい状況）が生まれる未来が想像できます。

AppleはツールとしてXcodeやフレームワークとしてUIKit、SwiftUI、Combine Framework、...etcは提供してくれるのだが、開発ガイドライン(HIGはUIやUXなどのデザインに関するものとしてあるが)はないので、開発者のレベルや思想の違いによりアプリの構造が全く別物になっております。

実際、iOSアプリ開発で関わらせて頂いた現場によって、全く別のアプローチで開発しておりました。

今回はReduxという**開発ガイドライン**を使ってアプリ開発をしてみたいと思います。Reduxは考え方であり、特定のフレームワークに依存しないものとして捉えているため、意図的に、**開発ガイドライン** と書かせて頂きました。

Reduxとは

Reduxは、Fluxアーキテクチャー(2014年5月のF8のセッションにて発表)のアイディアに影響を受けた、 Dan Abrmovさんによって、 2015年8月にリリース(Webアプリ向け)されたものです。

Dan Abrmovさんは、複雑性の原因を、**変化(mutation)**と**非同期性(asynchronicity)**が組み合わさったものであると考えており、それを解決するためのReduxは以下の特徴を備えております。

> ・ Flux アーキテクチャの情報の伝播を1方向に制限する特徴を踏襲し、いつどのように 更新が起きるかを明瞭にする
> ・ Elm アーキテクチャの純粋関数による副作用の排除や、イミュータブルな状態表現の 制約を踏襲し、厳格で整合性のとれた状態管理を実現する

Reduxを構成する要素

> ・Action:
> Storeに送られる情報です。**情報であり、処理は持ちません。**
>
> ・State:
> アプリケーションの状態を表現するデータ集合です。**あくまでデータなので処理は持ちません。**
>
> ・Reducer:
> Action と Stateの入力を受けて、新たなStateを出力する関数です。**純粋関数として記述されます。**
>
> ・Store:
> StateとReducerを保持するアプリケーションの**単一インスタンスです。**

Action、Stateはただのデータで、Reducerは純粋関数、じゃあAPI通信や処理はどこにいくのだ？ とReduxを学んだ時に疑問に思いました。Reduxでは、非同期な処理は、Redux-thunkというmiddle wareに処理をまかせます。このRedux-thunkはActionをReducerの間に来る処理となります。

Reduxの守るべき原則

> ・Single source of truth（信頼できる唯一の情報源）
> ・State in read-only（stateは読み取り専用にする）
> ・Changes are made with pure functions（変更はすべて純粋関数で行われる）

Reduxを構成する要素と守る原則を交えて解説していきます。

Single source of truth（信頼できる唯一の情報源）

Reduxでは、 Storeの中に Stateを保持しておりますが、このStateは、アプリケーションの中でユニークなものになります。

MVCなどのアプリで、記事購読アプリを作ろうとした場合は、記事一覧、ユーザー情報、ユーザーのお気に入り情報などの情報を別々のデータとして管理していた思いますが、Reduxを使うと、以下のようなツリーを一つStateで管理出来るようになります。

```
Root ー 記事一覧 ―――――――――――― 記事タイトル
          |                      ┣   記事サマリー
          |                      ┗   著者
          ┣ ユーザー情報
          ┗ ユーザーのお気に入り情報
```

Fluxでは、画面毎や機能毎にStateを持つのがReduxと大きく異る点です。

State in read-only（stateは読み取り専用にする）

> 「State をイミュータブルで表現する」とは、作成された State が値を変えることのできない不変なインスタンスであることを意味しています。
> Reducer により新たな State が生成されるまでの間、
> View レイヤーで参照している現在の State はまったく変更されないことが保証されます。

Stateは、ViewやViewControllerから直接変更することはできません、Reducerによってのみ生み出されます。
Reducerは、State と Actionを引数に新たなStateを返します。
Stateはイミュータブルなため、Reducer に記述されたビジネスロジックの実行結果をコピーした State に適用し、新たな Stateを作成します。

少し話が脱線するが、Swiftには”値渡し”と”参照渡し”がありますが、画面表示に表示するデータは"値渡し"のものを利用したほうが良いでしょう。参照渡しだと、参照していた奥深くのデータが知らぬ間に書き換わっていて期待した表示にならないという不具合に遭遇することにもなります。

Stateを生み出すReducerについて補足させてもらうと
Reducerは、**純粋関数**で書くことが求められます。

純粋関数の特性を次に列挙します。

> ・ 与えられた要素や関数外の要素を変化させず、戻り値以外の出力を行わない(副作用の排除)
> ・ 取り扱うすべての要素が引数として宣言されている(引数以外の要素を参照しない)
> ・ 入力に対して出力が常に一意である(同じ入力には常に同じ出力を返す)

Reduxの全体像

View (UIKitのViewController、SwiftUIのView) は画面を表示するための情報としてStateツリー、アクション(ボタンタップなど)としてActionのディスパッチを行いますが、その裏にあるActionの実行+Reducerの存在は意識しません。
Actionが実行され、Reducerにより新たなStateが作成されたら、その値が購読している※1 Viewに引き渡されます。
※1 購読するには、RxSwiftのデータバインドや、NotificationCenter　が使われてますが、SwiftUIでは @ObservableObject を使用します。

次の記事ではいよいよ実際のコードを交えてReduxによる実装方法解説していきます。

#プログラミング #swift #swiftui #Redux

masa7351
フォロー

このクリエイターの人気記事
SwiftUIアプリ設計をReduxを使って開発する(コードを使っての説明)
♡ 9 masa7351

SwiftUIアプリ設計をReduxを使って開発する(コードを使っての説明) ›