**Управление состоянием и действия в TCA**

Управление состоянием — это фундаментальная часть любой архитектуры приложения. В **The Composable Architecture (TCA)** состояние управляется централизованно, что упрощает отладку и улучшает предсказуемость изменений. Сегодня мы подробно рассмотрим, что такое состояние, как его описывать, работать с действиями и применять их в редюсерах для изменения состояния.

**Состояние (State) в TCA**

Состояние (State) представляет данные, которые описывают текущее состояние вашего приложения. Оно может включать:

• Информацию, которую пользователь видит на экране.

• Флаги для управления отображением (например, “загрузка данных”).

• Временные данные, такие как текст в текстовом поле.

**Ключевые особенности состояния в TCA:**

1. Состояние всегда должно быть структурированным. Разделение данных на отдельные структуры упрощает поддержку приложения.

2. Все изменения состояния происходят через **действия** (Action) и **редюсеры** (Reducer).

Пример состояния для простого приложения “Счётчик”:

struct CounterState: Equatable {

    var count: Int = 0

    var isLoading: Bool = false

}

• count представляет текущее значение счётчика.

• isLoading показывает, выполняется ли асинхронная операция.

**Почему** Equatable**?**

Тип состояния должен быть совместим с протоколом Equatable, чтобы TCA могла сравнивать состояния и определять, нужно ли обновлять пользовательский интерфейс.

**Действия (Action) в TCA**

Действия (Action) в TCA представляют события, которые могут изменять состояние. Это может быть:

• Нажатие кнопки пользователем.

• Ответ сервера.

• Системные события, такие как завершение загрузки данных.

Каждое действие в TCA описывается как перечисление (enum), чтобы их можно было четко определить.

enum CounterAction: Equatable {

    case increment

    case decrement

    case reset

    case loadData

    case dataLoaded(Result<String, Error>)

}

• increment, decrement, reset — это пользовательские действия.

• loadData и dataLoaded — системные события для управления загрузкой данных.

**Зачем использовать перечисления?**

Перечисления позволяют чётко описать все возможные события, которые может обработать приложение. Это упрощает тестирование и предотвращает ошибки.

**Reducer: связующее звено между состоянием и действиями**

Редюсер (Reducer) — это центральная функция в TCA, которая управляет изменением состояния. Она принимает текущее состояние, действие и внешнюю среду (если требуется), а затем возвращает новое состояние.

**Пример редюсера для приложения “Счётчик”:**

let counterReducer = Reducer<CounterState, CounterAction, Void> { state, action, \_ in

    switch action {

    case .increment:

        state.count += 1

        return .none

    case .decrement:

        state.count -= 1

        return .none

    case .reset:

        state.count = 0

        return .none

    case .loadData:

        state.isLoading = true

        return .none

    case .dataLoaded(let result):

        state.isLoading = false

        switch result {

        case .success(let message):

            print("Data loaded: \(message)")

        case .failure(let error):

            print("Error: \(error)")

        }

        return .none

    }

}

**Что делает редюсер?**

1. **Обрабатывает действия:** Каждое действие имеет свою ветвь switch, которая определяет, как именно изменяется состояние.

2. **Не изменяет состояние напрямую:** TCA работает с неизменяемыми данными, и редюсер создаёт новое состояние на основе текущего.

3. **Может возвращать эффекты:** Например, асинхронные задачи.

**Асинхронные действия и эффекты в редюсере**

Эффекты позволяют обработать асинхронные операции, такие как сетевые запросы или взаимодействие с базой данных. Они описываются как значения, которые могут быть возвращены редюсером.

**Пример эффекта для загрузки данных:**

case .loadData:

    return .task {

        let result = await fetchDataFromAPI()

        return .dataLoaded(result)

    }

**Как это работает?**

1. Когда пользователь инициирует действие loadData, редюсер запускает эффект.

2. Эффект выполняется асинхронно.

3. После завершения эффект возвращает новое действие (dataLoaded), которое снова обрабатывается редюсером.

**Совмещение нескольких редюсеров**

В больших приложениях состояние может быть разделено на модули, например:

• Состояние профиля пользователя.

• Состояние настроек.

• Состояние текущего экрана.

Каждый модуль может иметь свой редюсер. TCA позволяет комбинировать их в один с помощью метода combine.