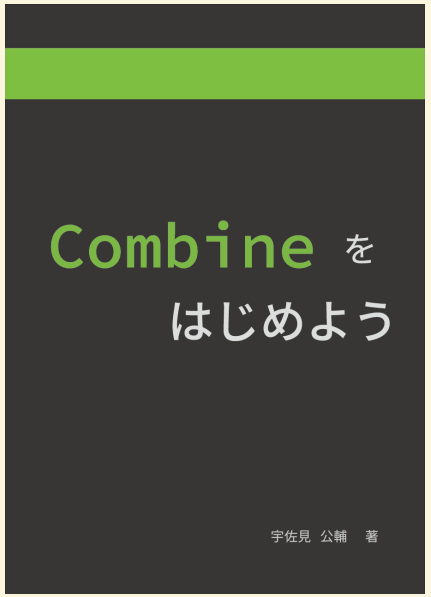
Swiftの Observationフレームワークによる 値の監視

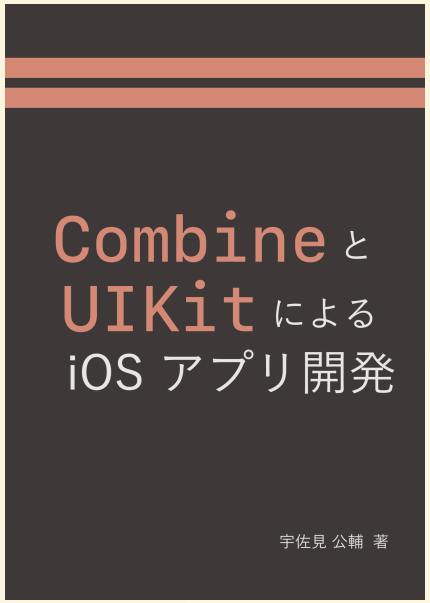
宇佐見公輔 / 株式会社ゆめみ

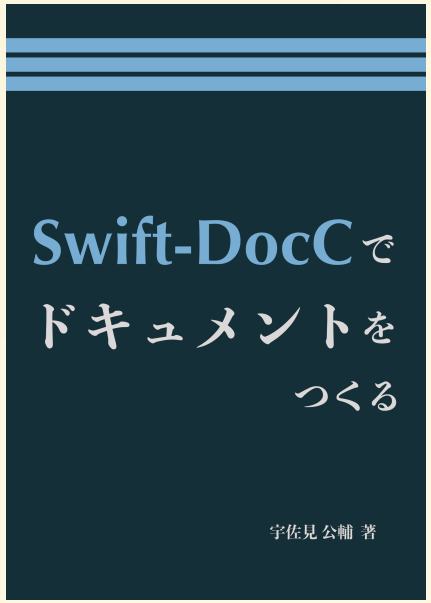
2023-12-13

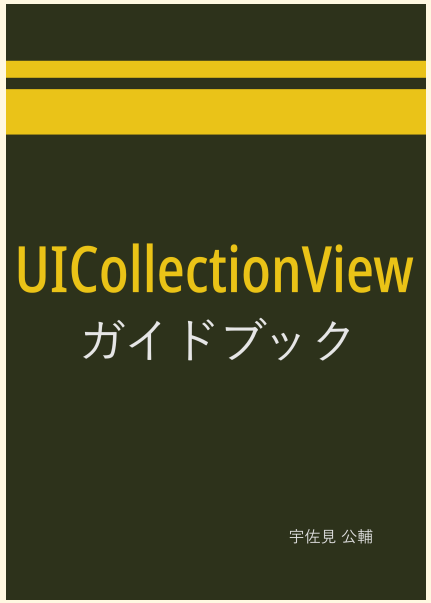
自己紹介

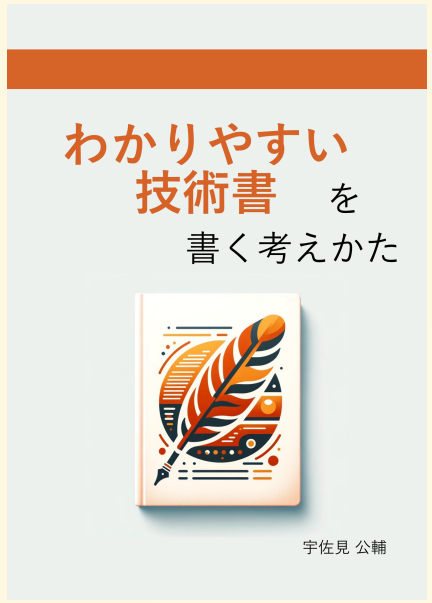
- 宇佐見公輔（うさみこうすけ）
- 株式会社ゆめみ / iOSテックリード











Observationフレームワーク

- Swiftで値の変更を監視するためのフレームワーク
- データバインディングの実現
  - とくにSwiftUIでの利用を想定
- Swift 5.9で追加された
  - iOS 17など最新OSでのみ動作する
  - 将来のデファクトスタンダード

Swiftにおける値の変更の監視

- KVO（Key-Value Observing）
  - Objective-Cからの機能
- ObservableObject
  - Combineフレームワークの機能
- @Observable
  - Observationフレームワークの機能

@Observable

- Swift 5.9で追加されたマクロ機能で実現

```
@Observable
final class Model {
    var value: Int = 0
}
```

- クラスに `@Observable` をつけると、そのクラスが監視可能になる

withObservationTracking

```
final class ViewController: UIViewController {
    private let model = Model()

    private func tracking() {
        withObservationTracking { [weak self] in
            guard let self else { return }
            print("value: \(model.value)")
        } onChange: {
            print("onChange")
        }
    }
}
```

変更の監視を開始する

```
withObservationTracking { ...
    ...
    print("value: \(model.value)")
} onChange: {
    ...
}
```

- `withObservationTracking` のクロージャで監視対象が決まる
  - `model.value` を参照 → `model.value` の値が監視対象となる

変更が通知される

```
withObservationTracking { ...
    ...
} onChange: {
    print("onChange")
}
```

- `model.value` の値が変更されると `onChange` クロージャが呼ばれる

動作の確認

- `tracking()` を呼び出すと `print("value: ...")` が呼ばれる

```
value: 0
```

- `model.value` の値を変更すると `print("onChange")` が呼ばれる

```
onChange
```

- なお、この通知は1回だけ来る

監視を継続する

```
    private func tracking() {
        withObservationTracking { ...
            ...
        } onChange: {
            Task { @MainActor [weak self] in
                guard let self else { return }
                tracking()
            }
        }
    }
```

- 再帰的に呼び出すことで、変更の監視を継続できる

UIKitでの利用例

```
private func tracking() {
    withObservationTracking { [weak self] in
        guard let self else { return }
        label.text = "value: \(model.value)"
    } onChange: {
        Task { @MainActor [weak self] in
            guard let self else { return }
            tracking()
        }
    }
}
```

SwiftUIの場合

```
struct MyView: View {
    var model = Model()

    var body: some View {
        Text(model.value)
    }
}
```

- これだけで継続的に値の変更を監視できる
  - `body` 内で参照されているプロパティが監視対象になる
  - `withObservationTracking` の記述は不要

不要な監視は発生しない

```swift
@Observable
final class Model {
    var value1: Int = 0
    var value2: Int = 0
}

struct MyView: View {
    var model = Model()

    var body: some View {
        Text(model.value1) // value2 の変更では再描画されない
    }
}
```

コレクションの監視

```
struct LibraryView: View {
    @State private var books = [Book(), Book(), Book()]

    var body: some View {
        List(books) { book in
            Text(book.title)
        }
    }
}
```

- `books` への要素の追加や削除を監視してViewを更新できる
- `book.title` の変更では、該当Viewだけが更新される

Observableフレームワークの制約

- 構造体には `@Observable` をつけられない
  - 値型（構造体）では使えず、参照型（クラス）で使える
  - Swiftは構造体の値変更を監視するようには設計されてない
- バックポートには対応しない
  - 古いバージョンのSwiftUIを対応させるのが難しい

おわりに

- Observationは既存の方法よりもシンプルで効率的
- SwiftUIとの相性が良い
- 古いOSで使えないのでまだ採用できないが、将来のスタンダード