Demystify SwiftUI

스유, 수수께끼를 풀다

Stat 2023. 5. 26.

Concepts of SwiftUI

- Identity
- Lifetime
- Dependencies

Identity

Different views?
Same view?

Identity

Different views or Same View

- 두 View에 해당하는 data의 identity에 따라서 결정됨
- identity가 동일하면 same view
  - 두 View의 차이는 state 변화에 따른 결과
- identity가 서로 다르면 different view
  - 두 View의 차이는 새로운 View가 생성된 결과
- different view냐 same view냐에 따라 transition이 다르게 표현됨

Identity

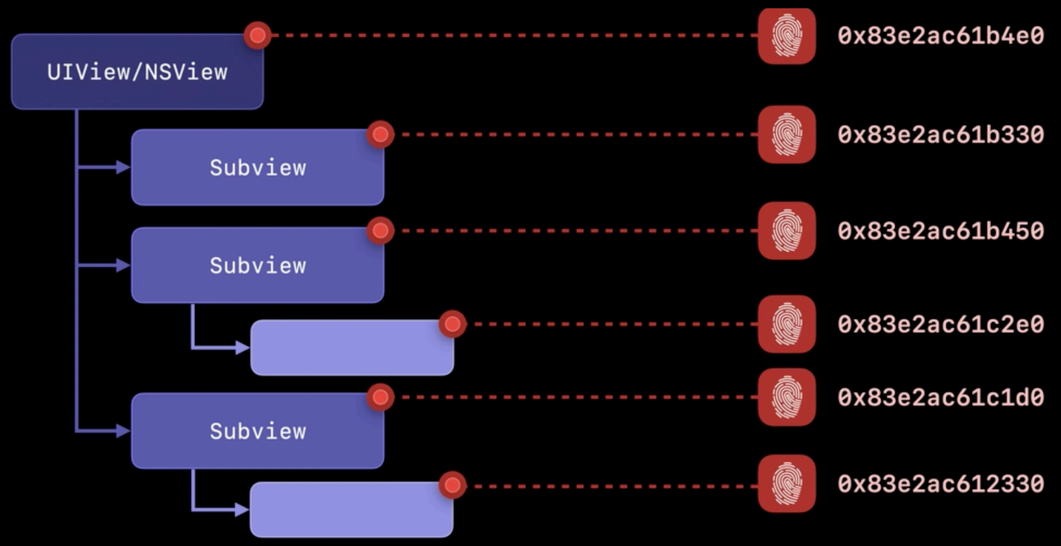
Types of identity used by SwiftUI

- Explict identity
- Structural identity

Identity

Explict identity in UIKit or AppKit

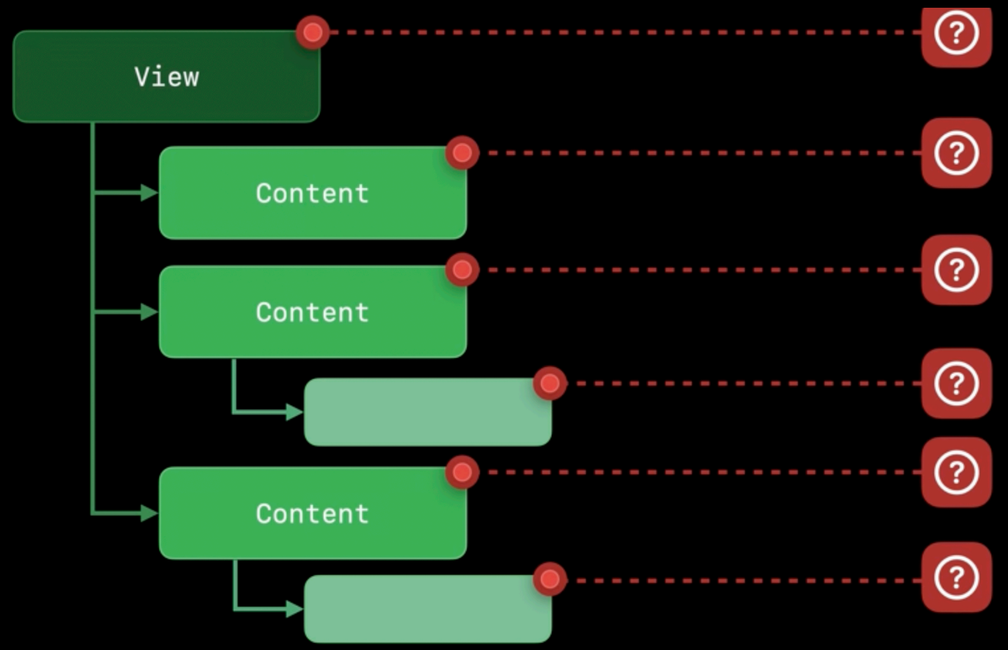
- UIView/NSView는 class로 정의 되어 있음
  - View 계층 구조에서 각각의 View는 unique pointer를 가진다
  - pointer가 explict identity로써 역할을 하게 됨
- pointer가 같으면 same view 다르면 different view



Identity

Explict identity in SwiftUI

- SwiftUI View는 value type
  - pointer ❌ - value type은 메모리 위치를 가리키지 않고 값의 복사본을 저장하기 때문에 포인터 비교가 불가능



Identity

Explict identity in SwiftUI - Examples

Structure

ForEach

A structure that computes views on demand from an underlying collection of identified data.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

```
init(Data, id: KeyPath<Data.Element, ID>, content: (Data.Element) -> Content)
```

Creates an instance that uniquely identifies and creates views across updates based on the provided key path to the underlying data's identifier.

Available when `Data` conforms to `RandomAccessCollection`, `ID` conforms to `Hashable`, and `Content` conforms to `View`.

Identity

Explict identity in SwiftUI - Examples

Instance Method

id(_:)

Binds a view's identity to the given proxy value.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

```
// Explicit Identity in SwiftUI

ScrollViewReader { proxy in
  ScrollView {
    HeaderView(rescueDog)
      .id(headerID)

    Text(rescueDog.backstory)

    Button("Jump to Top") {
      withAnimation {
        proxy.scrollTo(headerID)
      }
    }
  }
}
```

Identity

Explict identity in SwiftUI - Examples

We don't have to explicitly identify every view

-> View에 명시적으로 identity를 부여하지 않았다고 해서 identity가 없는것은 🤦‍♂️

-> structural identity를 가짐

```
// Explicit Identity in SwiftUI

ScrollViewReader { proxy in
  ScrollView {
    HeaderView(rescueDog)
      .id(headerID)

    Text(rescueDog.backstory)

    Button("Jump to Top") {
      withAnimation {
        proxy.scrollTo(headerID)
      }
    }
  }
}
```

Identity

Structural identity

👎

```
VStack {
  if dog.isGood {
    PawView(tint: .green)
    Spacer()
  } else {
    Spacer()
    PawView(tint: .red)
  }
}
```

condition에 따라 다른 View를 반환
-> 서로 다른 strucural identity를 가짐
-> Different Views

👍

```
PawView(tint: dog.isGood ? .green : .red)
  .frame(
    maxHeight: .infinity,
    alignment: dog.isGood ? .top : .bottom)
```

condition에 따라 single view가 수정됨
-> structural identity 유지
-> Same View

Identity

Structural identity - ConditionalContent View

```
var body: some View {
  if rescueDogs.isEmpty {
    AdoptionDirectory(selection: $rescueDogs)
  } else {
    DogList(rescueDogs)
  }
}
```

- 조건문으로 분기된 View는 SwiftUI 내부적으로는 generic type으로 치환하여 처리
  - _ConditionalContent<AViewType, BViewType...>
- true / false에 따라 다른 identity를 부여하는 로직이 _ConditionalContent에 구현되어 있는듯함

Concepts of SwiftUI

- Identity
- Lifetime
- Dependencies

Lifetime

View value != view lifetime

```
struct PurrDecibelView: View {
    var intensity: Double

    var body: some View {
        // …
    }
}
```

```
var body: some View {
    PurrDecibelView(intensity: 25)
}
```

```
var body: some View {
    PurrDecibelView(intensity: 50)
}
```

- body 가 invoke 되면 view value 생성, 기존 view value는 copy를 가지고 있다가 new view value와 비교 후 destory
- 이렇게 view value는 일시적이기 때문에 view의 수명이 view value에 의존해서는 안됨
- identity만 같다면 동일한 뷰로 인식함

Lifetime

State lifetime == View lifetime

```
struct CatRecorder: View {
    @State var title = ""

    @StateObject var mic = Microphone()

    var body: some View {
        VStack {
            TextField("Title:", text: $title)
            MicView(mic)
        }
    }
}
```

- @State 또는 @StateObject의 내부 date storage는 View lifetime과 동일하게 유지

Lifetime

State lifetime == View lifetime

```
struct CatRecorder: View {
    @State var title = ""

    @StateObject var mic = Microphone()

    var body: some View {
        VStack {
            TextField("Title:", text: $title)
            MicView(mic)
        }
    }
}
```

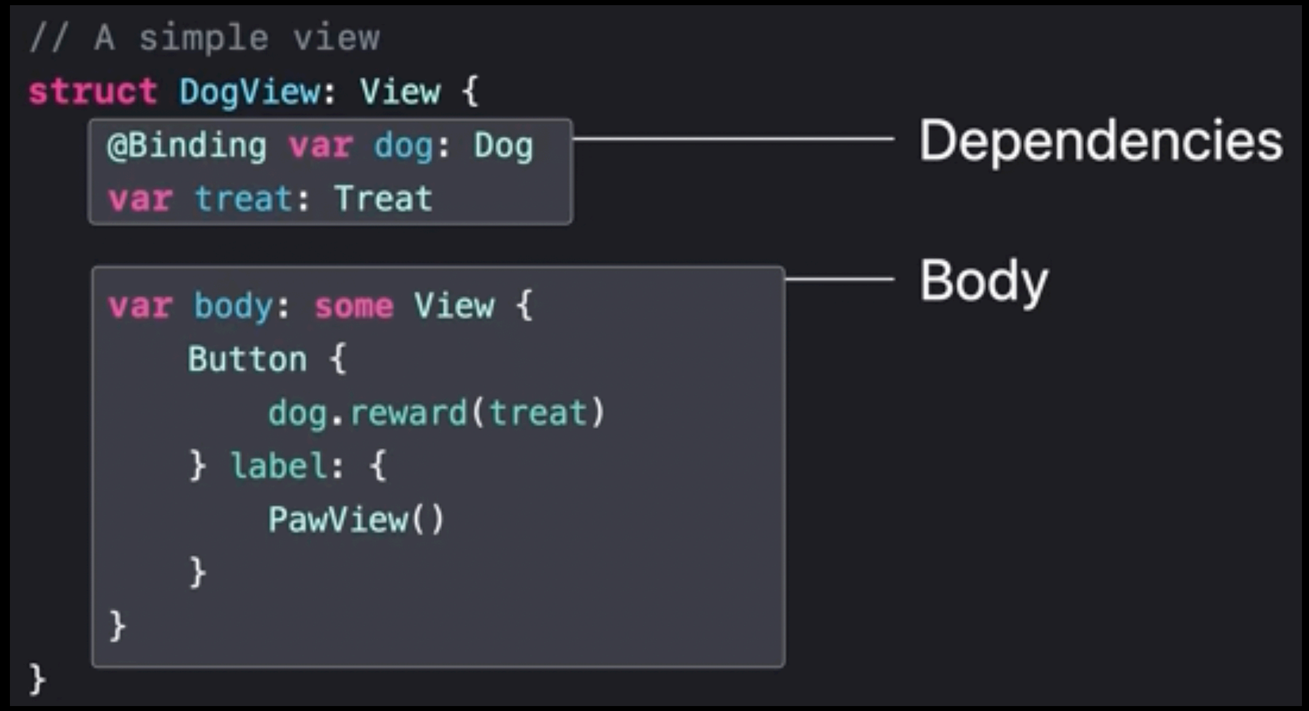
```
var body: some View {

    if dayTime {


        CatRecorder()


    } else {


        CatRecorder()
            .nightTimeStyle()


    }
}
```

- dayTime이 바뀌면 -> structural identity 바뀜 -> 기존 state storage 제거 및 신규 생성

Concepts of SwiftUI

- Identity
- Lifetime
- Dependencies

Dependencies

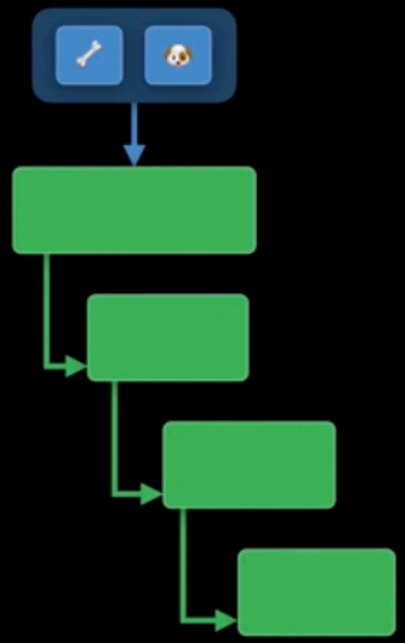


- A dependency is just an input to the view.
- When a dependency changes, the view is required to produce a new body.

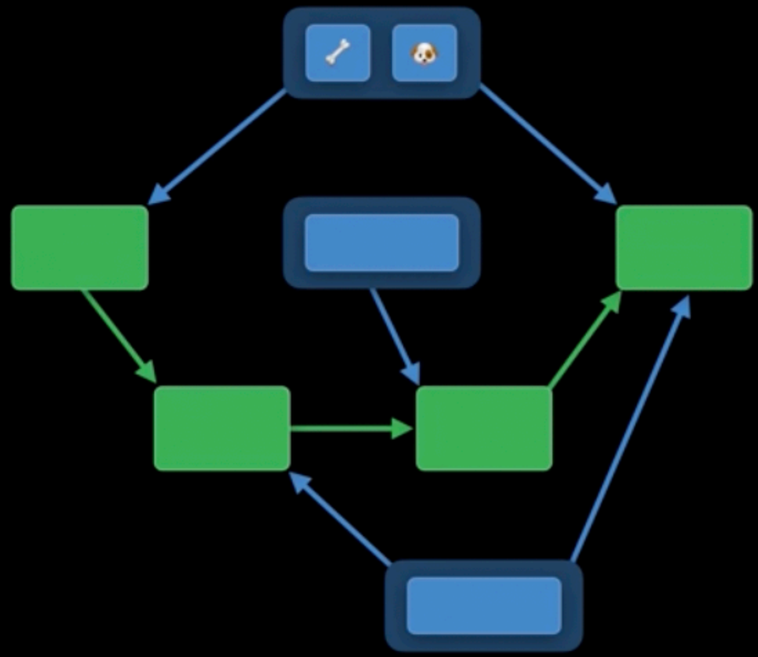
Dependencies

dependency structure

like tree



like graph



Dependencies
dependency graph

- 그래프 형태의 의존 구조를 가질 때도 identity을 활용하여 효율적으로 UI 업데이트 수행
- identifier stability -> 적절한 identity 부여하는 것이 중요함
  - view lifetime을 필요 이상으로 짧게 하는것 방지
  - 최소 연산을 통한 performance 향상
  - state storage lifetime에도 영향을 미치므로 state 로스를 막을 수 있음

Dependencies

explict identity - Identifier stability

👎

```
enum Animal { case dog, cat }

struct Pet: Identifiable {
    var name: String
    var kind: Animal
    var id: UUID { UUID() }
}

struct FavoritePets: View {
    var pets: [Pet]
    var body: some View {
        List {
            ForEach(pets) {
                PetView($0)
            }
        }
    }
}
```

👎

```
struct FavoritePets: View {
    var pets: [Pet]
    var body: some View {
        List {
            ForEach(pets.indices, id: \.self) {
                PetView(pets[$0])
            }
        }
    }
}
```

Dependencies

explict identity - uniqueness

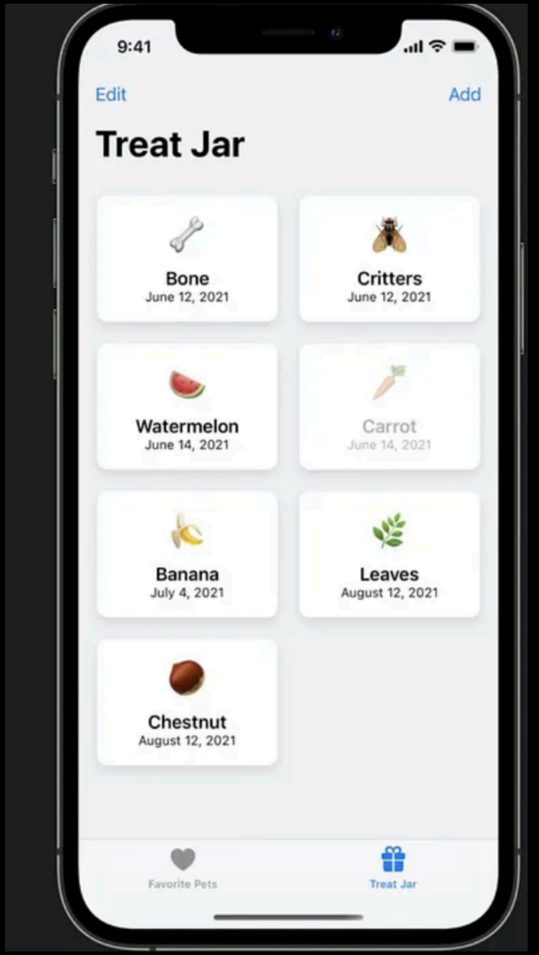
- improves animations
- performance
- Correntlly reflects dependencies

Dependencies

explict identity - identifier uniqueness

👎

```
struct TreatJar: View {
    var treats: [Treat]

    var body: some View {
        ScrollView {
            LazyVGrid(...) {
                ForEach(treats, id: \.name) {
                    TreatCell($0)
                }
            }
        }
    }
}
```



Dependencies

structural identity - example

👎

```
ForEach(treats, id: \.serialNumber) { treat in
    TreatCell(treat)
        .modifier(ExpirationModifier(date: treat.expiryDate))
}

struct ExpirationModifier: ViewModifier {
    var date: Date
    func body(content: Content) -> some View {
        if date < .now {
            content.opacity(0.3)
        } else {
            content
        }
    }
}
```

Branch