선언형 UI

선언형 구문

시스템에 절차적으로 UI를 생성하도록 하는 명령형 UI 프레임워크와 달리 SwiftUI는 선언형 구문을 사용한다. SwiftUI에서 개발자는 UI가 주어진 상황에서 어떻게 보여야 하는지를 기술하고 SwiftUI는 그것을 그려내는데 집중한다.

View

SwiftUI에서는 모든 것이 뷰이다. 한 개의 뷰는 View 프로토콜을 채택해 유저 인터페이스를 묘사하는 구조체이다. 뷰는 다른 뷰를 포함할 수 있으며 복잡한 UI를 빌드하기 위해 구성된다.

상태 관리

반응형(Reactive)

SwiftUI는 반응형 원칙을 기반으로 구축된다. 따라서 UI가 자동적으로 앱의 상태 변화를 업데이트한다. 이런 방식은 @State, @Binding, @ObservedObject, @EnvironmentObject와 같은 프로퍼티 래퍼에 의해 수행된다.

Single Source of Truth

SwiftUI는 단 하나의 데이터 소스를 사용하도록 권장한다. 이것은 모든 데이터가 앱의 한 장소에 저장되어 있어야 한다는 것을 의미한다. 앱의 다른 영역에서는 이 공간을 참조하거나 이 공간의 데이터 변화를 지켜봐야 한다.

레이아웃 시스템

Modifiers

수정자를 사용해 뷰를 커스터마이징할 수 있다. 여기서 수정자란 색깔, 패딩, 폰트 등의 특징을 변환해 뷰를 반환하는 메소드를 말한다.

Contatiners

SwiftUI는 뷰의 레이아웃을 위해 HStack, VStack, ZStack, List와 같은 컨테이너를 활용한다. 이 컨테이너들은 그것들의 자식 뷰를 어떻게 정렬하고 렌더링하는지 정의한다.

Adaptive

SwiftUI의 레이아웃은 본질적으로 적응형이다. 이는 SwiftUI가 아이폰, 아이패드, 맥 등 다양한 크기의 디바이스에서 동작하는 UI를 고려하기 때문이다.

Swift와의 통합

스위프트의 특징과 통합

SwiftUI는 스위프트와 긴밀하게 통합되어 스위프트의 특징적인 구조체, 열거형, 프로토콜을 강력하고 효율적으로 사용할 수 있게 한다.

스위프트UI의 라이프 사이클

새 프로젝트의 라이프 사이클 관리를 위해 SwiftUI App life cycle을 활용할 수 있다. UIKit AppDelegate와 비교했을 때 훨씬 간결하다.

퍼포먼스와 효율성

효율적인 렌더링

SwiftUI는 UI에서 업데이트가 필요한 부분만 다시 렌더링한다. 작은 데이터 변경 시 전체를 전부 다시 그리는 것보다 훨씬 효율적인 방식이다.

컴파일 언어

SwiftUI의 뷰는 고전적인 명령형 UI 세팅보다 훨씬 효율적인 방식으로 컴파일 된다.

스위프트 앱 라이프 사이클

엔트리 포인트

SwiftUI 앱에서 엔트리 포인트는 @main으로 선언된다. 일반적으로 @main은 App 프로토콜을 채택한 구조체에 선언된다.

App Protocol

앱 프로토콜은 SwiftUI의 일부이며 앱의 근간이다. SwiftUI 앱을 구축할 때, 개발자는 이 프로토콜을 채택한 구조체를 정의하고 body 프로퍼티를 통해 이를 구현해야 한다.

선언형 구문

앱 구조체의 body 프로퍼티는 SwiftUI의 선언형 구문을 사용해 앱의 초기 뷰를 지정한다.

```
@main
struct MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

인스턴스 생성

SwiftUI 앱을 켜면 시스템은 자동으로 @main이 붙은 구조체를 인스턴스화한다. SwiftUI는 body 프로퍼티에 선언된 뷰의 라이프 사이클을 관리하고 이를 렌더링한다.

View 구조체들

SwiftUI의 뷰들은 클래스가 아니라 구조체이다. 객체지향 패러다임에서는 클래스를 객체화해서 다루지만 스위프트에서는 구조체를 인스턴스화 해서 다루는 게 아니다. 대신에 개발자는 View 구조체를 선언하고 그것들의 관계로서 UI를 선언한다.

View 렌더링

뷰를 body 속성에 선언하면, SwiftUI는 뷰의 인스턴스화와 렌더링을 관리한다. 프레임워크는 뷰의 라이프 사이클을 생성하고 관리하는데 책임이 있다.

상태와 데이터 흐름

SwiftUI는 UI를 업데이트할 때 데이터 주도적인 접근 방식을 취한다. 뷰들은 state variables에 의존적이다. 그리고 state가 변화할 때, 뷰는 새로운 상태를 반영해 부분적으로 다시 렌더링된다.

스프링과의 비교

스프링에서는 메인 함수를 실행해 스프링 컨테이너를 부트스트랩하고 스프링 컨테이너가 객체의 생명주기를 담당한다. 이와 다르게 SwiftUI는 declarative UI definitions와 data state에 기반한 view의 라이프 사이클과 인스턴스화를 담당한다. 뷰의 수동적인 인스턴스화나 그를 통한 직접적인 라이프 사이클 관리는 일반적으로 존재하지 않는다.

결론

SwiftUI에서는 뷰의 인스턴스화나 생명주기 관리를 추상화해 프레임워크가 책임지고 개발자의 UI 개발을 단순화한다. 개발자는 UI가 어떤 일을 해야 하는지 기술하기만 하면 된다. UI가 어떻게 조직되고 관리될지 거의 신경 쓸 필요가 없다. 프레임워크가 알아서 하기 때문이다. 개발자는 대신 가독성이 좋고 정확한 코드를 쓰는데 더 신경을 쓰면 된다.