HACKING WITH SWIFT Q Search FORUMS LEARN - CAREERS - STORE - ABOUT -SUBSCRIBE

FREE TRIAL: Accelerate your app development career with Hacking with Swift+! >>

BUY OUR BOOKS

PRO SWIFT

TESTING SWIFT

SWIFT CODING CHALLENGES

SERVER-SIDE SWIFT VAPOR

ADVANCED IOS VOLUME TWO

watchOS

SERVER-SIDE SWIFT KITURA

macos spritekit

HACKING WITH iOS

ADVANCED IOS VOLUME ONE

ADVANCED IOS VOLUME THREE

tvOS

Objective-C for Swift Developers

Beyond

Code

wrapper? >

What is the @EnvironmentObject property < What is the @Published property wrapper?

What is the @ObservedObject property wrapper?

Updated for Xcode 13.2

SwiftUI gives us the @ObservedObject property wrapper so that views can watch the state of an external object, and be notified when something important has changed. It is similar in behavior to **@StateObject**, except it must *not* be used to create objects – use **@ObservableObject** only with objects that have been created elsewhere, otherwise SwiftUI might accidentally destroy the object.

For example, we might use something like this:

```
class Order: ObservableObject {
    @Published var items = [String]()
struct ContentView: View {
    @ObservedObject var order: Order
    var body: some View {
        // your code here
```

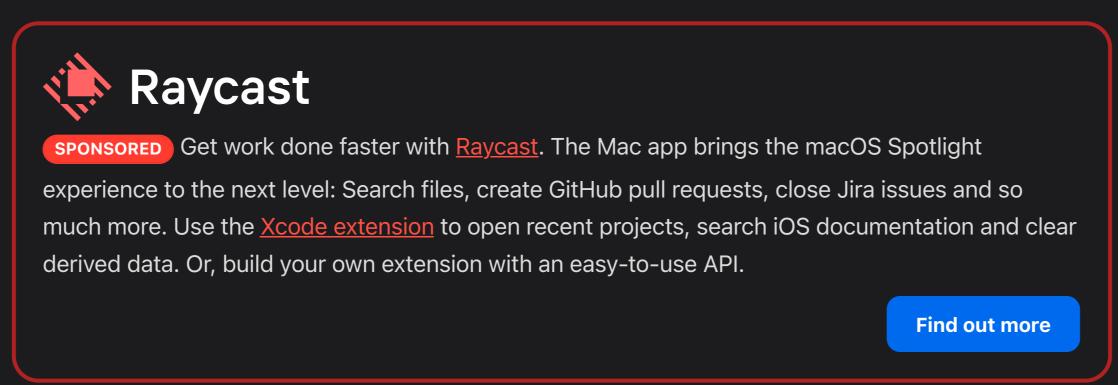
That **Order** class uses **@Published** so it will automatically send change announcements when **items** changes, and **ContentView** uses **@ObservedObject** to watch for those announcements. Without @ObservedObject the change announcements would be sent but ignored.

Although that looks straightforward enough, it's worth digging into a few specifics.

First, any type you mark with @ObservedObject must conform to the ObservableObject protocol, which in turn means it must be a class rather than a struct. This isn't optional – SwiftUI requires us to use a class here.

Second, observed objects are specifically designed for data that is external to your view, which means it might be shared across more than one view. The @ObservedObject property wrapper will automatically make sure the property is watched closely so that important changes will reload any views using it. This also means the data must be created elsewhere, then sent in to your view.

Third, not all properties in an observed object cause views to refresh – you need to decide which properties should send change notifications, either using @Published or custom announcements. Types that conform to **ObservableObject** are given a default **objectWillChange** publisher to make custom announcements as needed.



Sponsor Hacking with Swift and reach the world's largest Swift community!

Similar solutions...

- What's the difference between @ObservedObject, @State, and @EnvironmentObject?
- How to use @ObservedObject to manage state from external objects
- How to fix "Referencing initializer 'init(wrappedValue:)' on 'ObservedObject' requires that 'SomeType' conform to 'ObservableObject'"
- All SwiftUI property wrappers explained and compared
- What is the @GestureState property wrapper?

What is the @EnvironmentObject property < What is the @Published property wrapper? wrapper? >

> Was this page useful? Let us know! ****

Average rating: 4.6/5

Click here to visit the Hacking with Swift store >>



paul@hackingwithswift.com



About

Privacy Policy

Refund Policy **Update Policy** Code of Conduct

Thanks for your support, Göran Lilja!

Swift, SwiftUI, the Swift logo, Swift Playgrounds, Xcode, Instruments, Cocoa Touch, Touch ID, AirDrop, iBeacon, iPhone, iPad, Safari, App Store, watchOS, tvOS, Mac and macOS are trademarks of Apple Inc., registered in the U.S. and other countries. Pulp Fiction is copyright © 1994 Miramax Films. Hacking with Swift is ©2021 Hudson Heavy Industries.

You are not logged in

Log in or create account