Thứ 3 11/01/2022

NOTE

I. Observables

1. Khái niệm

- RxSwift là dựa trên khái niệm lập trình Reactive Programming hay nói đơn giản là lập trình với dữ liệu theo luồng bất đồng bộ.
- Observable: là nguồn phát ra dữ liệu mà có thể được quan sát bởi các đối tượng và đăng ký tới.
- Sequence: là luồng dữ liệu mà nguồn phát phát đi.
- Observer: là đối tượng mà đăng ký tới 1 Observable. Đối tượng này sẽ phản ứng với các dữ liệu, luồng dữ liệu mà Observable sẽ phát ra tới.
- 1 Observable thường được ví dụ như 1 mảng bất biến mà có dữ liệu bất đồng bộ.

2. Mô hình

- Reactive programming thường được thể hiện dưới dạng 1 sơ đồ marble để làm rõ các sequence của Observable phát cho các

observer.

- Ví dụ:
- ---1---2----3----4----6----|--->
- Sequence đi từ trái sang phải.
- Các số là các giá trị được phát ra.
- | tượng trưng cho kết thúc (completed).
- X tượng trưng cho xảy ra lỗi (error).

3. Vòng đời của Observable

- Các kiểu giá trị mà Observable có thể phát:
  + Value: gía trị dữ liệu mà nguồn phát đi. Thường là kiểu Event.
  + Error: giá trị trả về nếu xảy ra lỗi. Nếu xảy ra lỗi thì sẽ loại bỏ Observable.
  + Completed: Kết thúc Observable.
- Các method sử dụng cho các sự kiện của vòng đời:
  + onNext: Khi mà Observable phát đi giá trị của luồng. Có tham số là dữ liệu mà Observable phát ra.
  + onError: Khi mà Observable gặp trục trặc. Observable sẽ ngừng gọi onNext, onCompleted và có 1 tham số hiển thị lỗi trả

về.

+ onCompleted: Khi mà Observable hoàn thành gọi onNext lần cuối và không gặp lỗi.

- Vòng đời:
  + Khởi tạo Observable
  + Gọi onNext với mỗi lần phát đi giá trị bên trong sequence.
  + Nếu gặp lỗi thì sẽ gọi onError và kết thúc.
  + Nếu ta không gặp trục trặc và hoàn thành phát hết giá trị trong sequence thì ta sẽ thực hiện gọi onCompleted và kết thúc.

4. Cách tạo Observable
   - just: Là toán tử để tạo 1 Observable mà có sequence chứa 1 phần tử duy nhất.
   - Ví dụ:

```
let observable1 = Observable<Int>.just(ios)
```

   - of: Là toán tử để tạo 1 Observable mà có sequence chứa 1 hay nhiều phần tử có cùng kiểu.
   - Ví dụ:

```
let observable2 = Observable.of(ios, android, flutter)
```

   - of cũng có thể dùng để chứa các kiểu

Collection như mảng

```
let observable3 = Observable.of([ios, android, flutter])
```

- from: cũng giống như just nhưng dành cho các kiểu thuộc protocol Sequence.
- Ví dụ:

```
let dev = [ios, android, flutter]
let observable4 = Observable.from(dev)
```

5. Subscribing to Observable
   - addObserver: Các event mà lắng nghe khi mà Observable phát đi dữ liệu được gọi là Observable như onNext, onError ,onCompleted... mà với RxSwift thì ta có thể xem được các dữ liệu mà Observable phát ra.
   - Subscribe: Để thực hiện lắng nghe Observable ta thực hiện sử dụng toán tử .subscribe.
   - Ta có thể thực hiện lắng nghe 3 event:
     + onCompleted, onError và onNext với 3 tham số trả về tương ứng cho từng trường hợp.
     + Để lấy giá trị bên trong các tham số ta phải thực hiện optional binding do khi gọi

onNext thì có thể trả về giá trị nil.

- Ví dụ:

```
observable4.subscribe { value in
    print(value)
} onError: { (error) in
    print(error.localizedDescription)
} onCompleted: {
    print("Completed")
} onDisposed: {
    print("Disposed")
}
```

6. Các dạng đặc biệt của Observable
   - Empty: Là 1 toán tử tạo 1 observable mà có sequence rỗng không phát ra phần tử nào và kết thúc ngay lập tức.
   - Never: giống như empty nhưng không bao giờ kết thúc.
   - Range: giống như lệnh for .. in cơ bản mà với tham số bắt đầu số lần lặp.

```
let observableRange = Observable<Int>.range(start: 1, count: 10)
var sum = 0
observableRange.subscribe { i in
    sum += i
} onCompleted: {
    print("Sum = \(sum)")
}
```

   + Dữ liệu mà observable phát đi sẽ được tăng lên 1 đơn vị mỗi lần phát.

+ Dữ liệu bị giới hạn là kiểu Int.

- Các điểm lưu ý:
  + Observable là 1 mảng dữ liệu bất đồng bộ mà bất biến.
  + Ta có thể truy cập các giá trị của observable emit ở nhiều thời điểm khác nhau.
  + Để thay đổi 1 Observable ta có thể sử dụng các toán tử của RxSwift để tạo 1 Observable mới với các thay đổi.
  + Observable bắt buộc phải có giá trị lúc bắt đầu.
  + Các subscriber sẽ chạy theo thứ tự mà subscriber đăng ký mỗi khi observer emit dữ liệu

II. Subject

1. Subject là gì

- Subject là 1 Observable và vừa là 1 Observer. Khi nào mà nhận 1 event .next thì sẽ phát dữ liệu cho các subscriber của nó.
- Do subject vừa là 1 observable và là 1 observer nên subject có thể subscribe cho 1 observable

và phát đi lại dữ liệu vừa được nhận và cũng có thể phát đi thêm dữ liệu mới cho các subscriber của nó.

2. Các loại subject:
    - Publish subject: Khởi đầu empty và chỉ emit các element mới cho các subscriber của nó.
    - Behaviour subject: Khởi đầu với 1 giá trị khởi tạo và sẽ relay lại element cuối cùng của chuỗi đang phát cho subscriber mới subscribe. Nếu không có giá trị thì sẽ sử dụng giá trị mặc định là giá trị khởi tạo.
    - Replay subject: Khởi tạo với 1 cache có kích thước cố định. Sau đó sẽ emit lại các element trong cache chosubscriber.
        + Replay subject được sử dụng để replay lại các element mà được emit bởi Observable gốc cho các subscriber của subject mà không để ý đến thời điểm mà subscriber subscribe.
        + Lưu ý: cache chỉ lưu lại các element gần nhất do nếu như vượt qua dung lượng của cache thì sẽ loại bỏ các element cũ.

- Publish relay & Behaviour relay: Các subject mà được bọc lại (wrap) mà chỉ nhận sự kiện .next. Thường dùng cho sự kiện mà không bao giờ kết thúc như .Never.
- Async subject: Là 1 subject mà chỉ emit ra dữ liệu trong .next cuối cùng của sequence chỉ khi mà trả về là .completed.

3. Hello subject
   - Publish subject:
     + Ví dụ mẫu:

```
//making a subject.
let subject = PublishSubject<String>()
//emitting a value.
subject.onNext("Hello World")
//subscribing to the subject.
let subscription1 = subject.subscribe { (value) in
    print(value)
}
//emitting again this time with a subscriber.
subject.onNext("Hello again")
subject.onNext("Hello again thrice")
subject.onNext("Bye")
```

   - Các điểm lưu ý:
     + Với publishSubject ta không cần phải có giá trị lúc bắt đầu do lúc khởi đầu thì publish subject là empty.
     + Với publish subject thì sẽ chỉ phát gía trị

mới cho subscriber. Có nghĩa là nếu ta mà phát giá trị lúc mà chưa có subscriber nào thì giá trị đó sẽ mất.

+ Subject khác với Observable ở việc không cần phải có gía trị ban đầu và có thể emit ở mọi nơi và mọi thời điểm.
+ Nếu publish subject nào mà thực hiện xong (.error hay .completed) thì các subscriber nào mà subscribe sau thời điểm đó thì sẽ chỉ nhận .error hay .completed .

III. Sự khác biệt giữa Struct và Class

1. Điểm giống nhau
   - Cả class và struct đều có thể chứa method/property
   - Hỗ trợ subscript synctax để truy vấn giá trị theo subscript.
   - Có thể được mở rộng bằng các extension để thêm các method/computed property cho các kiểu giá trị cơ bản thường dùng như Int, Double, String.
   - Có thể tạo init riêng để khởi tạo đúng theo mục đích của lập trình

- Có thể thêm protocol để thêm các tính năng cho của protocol-oriented programming như sử dụng được với các toán tử, thực hiện delegation pattern, chia sẻ code giữa các kiểu mà áp dụng protocol...

2. Điểm khác
   - Struct:
     + Là kiểu giá trị (Value type). Mỗi instance sẽ là 1 struct riêng. Khi mà ta gán 1 instance của 1 struct cho 1 giá trị khác hoặc đưa vào 1 trong hàm thì ta sẽ tạo 1 instance mới giống như struct
     + Struct có hàm khởi tạo mặc định (Memberwise init) mà có thể sử dụng nếu ta không định nghĩa 1 hàm khởi tạo riêng.
   - Class:
     + Là kiểu tham chiếu (Reference type). Mỗi instance sẽ chia sẻ 1 bản sao của dữ liệu. Khi mà ta thực hiện copy hoặc gán 1 instance của 1 class sang 1 class khác thì ta đang chia sẻ bản sao của dữ liệu sang hằng/biến class khác.

+ Class có thể kế thừa 1 class khác để sử dụng lại các thuộc của superclass
+ Class là kiểu tham chiếu nên ta có thể thực hiện thay đổi các property mà không cần sử dụng mác mutating
+ Class có thể sử dụng type casting is, để ép kiểu 1 class.
+ Class có thể sử dụng toán tử === và !== để kiểm tra 2 đối tượng có chỉ đến cùng 1 instance hay không.

3. Sử dụng struct khi nào
   - Struct thường được sử dụng nhiều do tính an toàn của kiểu giá trị và giúp cho sử dụng bộ nhớ ít hơn.
   - Nên sử dụng Struct nếu mà kiểu mà ta tạo có nhiều property kiểu cơ bản như Int, String, Double...
   - Sử dụng cho công việc đa luồng do có thể copy giá trị từ luồng này sang luồng khác 1 cách an toàn hơn.
   - Dễ quản lý sự thay đổi của đối tượng hơn do nếu ta muốn thay đổi thì phải gọi thẳng tới đối

tượng đó.

4. Sử dụng class khi nào
    - Class thường được sử dụng khi nào mà ta cần các tính năng mà Struct không có.
    - Class thường dùng nếu ta cần phải làm việc với thư viện Objective-C của
    - Trường hợp mà copy dữ liệu là không cần thiết như chỉ có 1 đối tượng active tại 1 thời điểm, tạo nhiều instance mà cùng chỉ đến 1 dữ liệu.