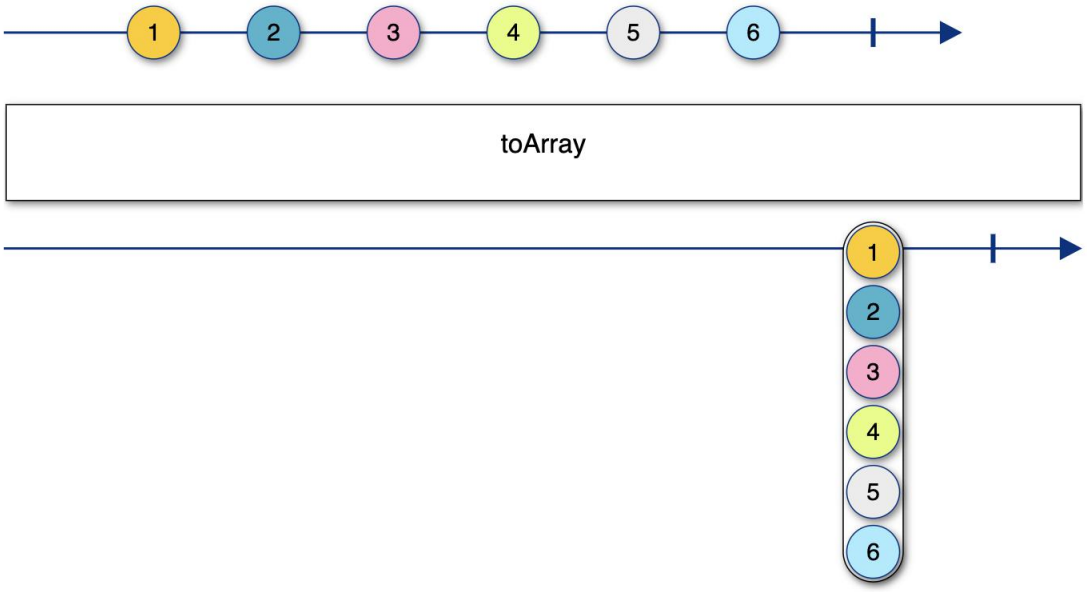
Thứ 5 ngày 13/01/2022

NOTE

I. Transforming Elements

1. toArrays
    - Công dụng: dùng để chuyển đổi 1 Observable thành 1 Single mà sẽ emit 1 mảng chứa các element trong sequence sau khi Observable kết thúc
    - Single chỉ nhận .success và .error
    - Sơ đồ marble:



2. map
    - Công dụng: Dùng 1 closure để biến đổi kiểu dữ liệu nhận được từ observable.

II. Transforming inner Observable

1. flatMap

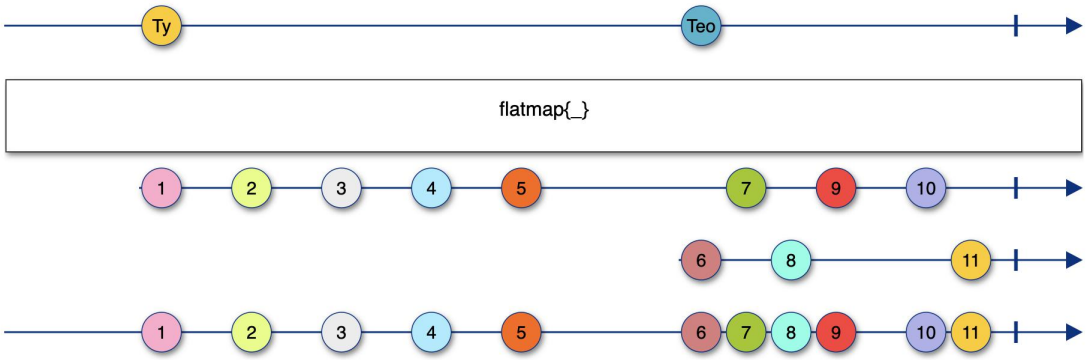
- Công dụng: Dùng để biến đổi nhiều observable thành 1 observable.
- flatMap thực hiện như sau:
    + Ta có 1 observable gốc emit đi các observable.
    + Các element trong observable gốc bắt đầu emit dữ liệu.
    + Tất cả các giá trị của element bị bắt lại bởi flatMap và cuối cùng tạo 1 observable mới với tất cả dữ liệu mà observable element phát ra.
- flatMap sẽ lấy tất cả dữ liệu không quan tâm đến thời gian mà element được emit trong observable gốc.
- Trình tự:

```
subject
    .flatMap { user in
        user.message ///Changing the user into a string
    }
    .subscribe(onNext: { msg in
        print(msg)
    })
    .disposed(by: bag)

//1
subject.onNext(cuTy)
//cuTy emit default value
//2~5
cuTy.message.onNext("Co ai o day khong?")
cuTy.message.onNext("Co 1 minh minh thoi a!")
cuTy.message.onNext("Buon vay!")
cuTy.message.onNext("...")
//6
subject.onNext(cuTeo)
//cuTeo emit default value
//7
cuTy.message.onNext("Chao Teo, ban co khoe khong?")
//8
cuTeo.message.onNext("Chao Ty, minh khoe. Con ban thi sao?")
//9-10
cuTy.message.onNext("Minh cung khoe luon")
cuTy.message.onNext("Minh dung day tu chieu ne")
//11
cuTeo.message.onNext("Ke Ty. Ahihi")
```

- Do cuTeo & cuTy là behaviourSubject nên khi phát đi thì sẽ phát chuỗi mặc định khi khởi tạo 2 subject đó ở bước 1 & 6.
- Output:

```
Cu Ty chao ban!
Co ai o day khong?
Co 1 minh minh thoi a!
Buon vay!
...
Cu Teo chao ban!
Chao Teo, ban co khoe khong?
Chao Ty, minh khoe. Con ban thi sao?
Minh cung khoe luon
Minh dung day tu chieu ne
Ke Ty. Ahihi
```

- Sơ đồ marble:



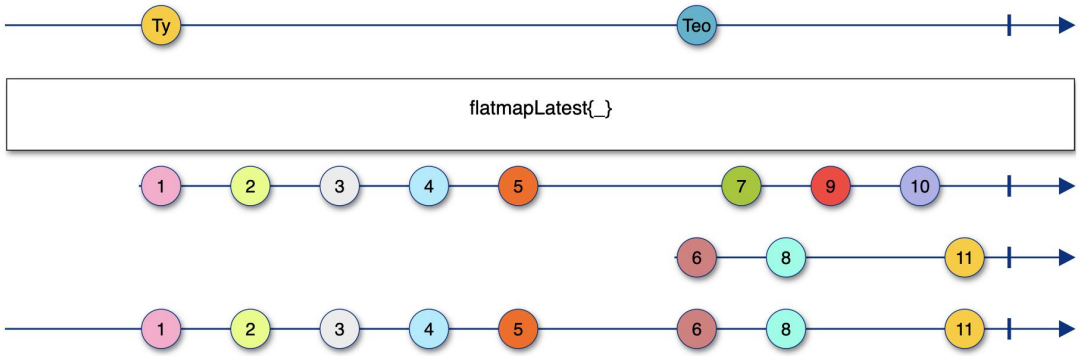
2. flatMapLatest
   - Công dụng: giống như flatMap nhưng có thêm tính năng của toán tử switchToLatest mà làm cho observable gốc chỉ nhận dữ liệu của element cuối cùng.
   - Trình tự:

```
subject
    .flatMapLatest({ $0.message })
    .subscribe{ print($0)}
    .disposed(by: bag)

//1
subject.onNext(cuTy)
//cuTy emit default value

//2~5
cuTy.message.onNext("Co ai o day khong?")
cuTy.message.onNext("Co 1 minh minh thoi a!")
cuTy.message.onNext("Buon vay!")
cuTy.message.onNext("...")
//6
subject.onNext(cuTeo)
//cuTeo emit default value

//7
///ignored
cuTy.message.onNext("Chao Teo, ban co khoe khong?")
//8
cuTeo.message.onNext("Chao Ty, minh khoe. Con ban thi sao?")
//9-10
///ignored
cuTy.message.onNext("Minh cung khoe luon")
cuTy.message.onNext("Minh dung day tu chieu ne")
//11
cuTeo.message.onNext("Ke Ty. Ahihi")
```

- Output:

```
next(Cu Ty chao ban!)
next(Co ai o day khong?)
next(Co 1 minh minh thoi a!)
next(Buon vay!)
next(...)
next(Cu Teo chao ban!)
next(Chao Ty, minh khoe. Con ban thi sao?)
next(Ke Ty. Ahihi)
```

- Sơ đồ marble:



III. Observing events

1. error

- Trường hợp error mẫu:
- Code:

```
let cuTy = User(message:BehaviorSubject(value: "Cu Ty chao ban"))
let cuTeo = User(message: BehaviorSubject(value: "Cu Teo chao ban"))

let subject = PublishSubject<User>()

let roomChat = subject.flatMap{ $0.message }

roomChat
    .subscribe(onNext: { msg in
        print(msg)
    })
    .disposed(by: bag)
```

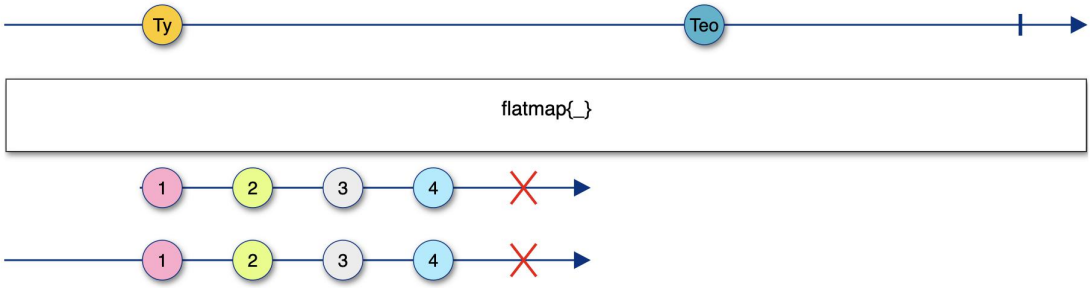
- Trình tự:

```
//1
subject.onNext(cuTy)

//2~4
cuTy.message.onNext("Ty: A")
cuTy.message.onNext("Ty: B")
cuTy.message.onNext("Ty: C")

//5
cuTy.message.onError(MyError.anError)
//terminated.

cuTy.message.onNext("Ty: D")
cuTy.message.onNext("Ty: E")

subject.onNext(cuTeo)

cuTeo.message.onNext("Teo: 1")
cuTeo.message.onNext("Teo: 2")
```

- Output:

```
Cu Ty chao ban
Ty: A
Ty: B
Ty: C
Unhandled error happened: anError
```

- Sơ đồ marble:



2. materialize

- Công dụng: Thường dùng để biến đổi các element bên trong observable thành các event như onNext, onError, onCompleted.
- Toán tử này giúp ta bắt các event của các observable element khi sử dụng với flatMap.
- Sử dụng:

```
let roomChat = subject.flatMap{ $0.message.materialize() }
```

- Trình tự:

```
//1
subject.onNext(cuTy)

//2~4
cuTy.message.onNext("Ty: A")
cuTy.message.onNext("Ty: B")
cuTy.message.onNext("Ty: C")

//5
///terminate cuTy Observable
cuTy.message.onError(MyError.anError)

//6-7
///terminated so this part is ignored
cuTy.message.onNext("Ty: D")
cuTy.message.onNext("Ty: E")

//8
subject.onNext(cuTeo)

//9-10
cuTeo.message.onNext("Teo: 1")
cuTeo.message.onNext("Teo: 2")
```

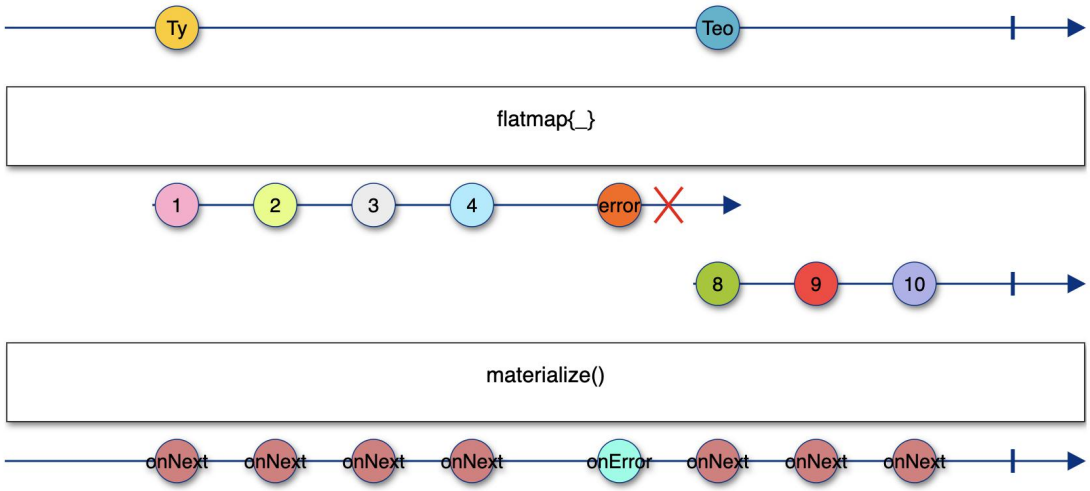
- Output:

```
next(Cu Ty chao ban)
next(Ty: A)
next(Ty: B)
next(Ty: C)
error(anError)
next(Cu Teo chao ban)
next(Teo: 1)
next(Teo: 2)
```

- Chú ý: do cuTeo có phát ra error nên bị

terminated bỏ qua bước 6 & 7.

- Sơ đồ marble:



3. dematerialize

- Công dụng: Thường dùng để biến đổi giá trị event được thay đổi do toán tử materialize lại thành giá trị ban đầu.
- Bắt onError mà observable phát ra sử dụng filter { } :

```
roomChat
    .filter { event in
        guard event.error == nil else { //skipping an onError event
            print("Error: \(event.error!)")
            return false
        }

        return true
    }
```

- Lấy giá trị bên trong event:

```
    .dematerialize() //converting the materialized event back into element.
    .subscribe(onNext: { msg in
        print(msg)
    })
    .disposed(by: bag)
```

- Output:

```
Cu Ty chao ban
Ty: A
Ty: B
Ty: C
Error: anError
Cu Teo chao ban
Teo: 1
Teo: 2
```

- Sơ đồ marble: