

RxSwift Basics – Day 2

Younghwan Kim



RxSwift Basics

- Day 1 – Observable, Operator (Filter, Transform, Combine)
- **Day 2 – Subject (flatMap, flatMapFirst, flatMapLatest)**
- Day 3 – Two VCs communications with Subject, RxCocoa (Button)
- Day 4 – Sequential, Merged Observable Calls
- Day 5 – RxCocoa, UI Binding (Button, TextField, Label, TableView)



Advanced RxSwift

- Day 1 – Protocol-Oriented Programming, Protocol Extension, Associatetype
- Day 2 – Network Call, Generic Enum
- Day 3 – Binding Track Activity (show / hide ‘Loading’)
- Day 4 – Adding a Reactive Extension to Custom UI Element,
2 Way Binding, Advanced TableView – RxDataSources
- Day 5 – Schedulers (observeOn, subscribeOn),
Unit Test (RxTest, RxBlocking)



Subject

Observable

Observer

PublishSubject, BehaviorSubject, ReplaySubject

PublishRelay, BehaviorRelay (Variable)



PublishSubject

```
func publishSubjectTest() {  
    let subject = PublishSubject<String>()  
    subject.onNext("Is anyone listening?")  
  
    let subscriptionOne = subject  
        .subscribe(onNext: { string in  
            print(string)  
        })  
  
    subject.on(.next("1"))  
    subject.onNext("2")  
  
    let subscriptionTwo = subject  
        .subscribe { event in  
            print(event)  
        }  
  
    subject.onNext("3")  
    subscriptionOne.dispose()  
    subject.onNext("4")
```

```
// 1  
subject.onCompleted()  
// 2  
subject.onNext("5")  
// 3  
subscriptionTwo.dispose()  
let disposeBag = DisposeBag()  
  
// 4  
subject  
    .subscribe {  
        print($0)  
    }  
    .disposed(by: disposeBag)  
  
subject.onNext("?")  
}
```



BehaviorSubject

```
func behaviorSubjectTest() {  
    let subject = BehaviorSubject(value: "Initial value")  
  
    let disposeBag = DisposeBag()  
  
    subject.onNext("X")  
    subject.asObservable()  
        .subscribe {  
            print($0)  
        }  
        .disposed(by: disposeBag)  
    // 1  
    subject.onError(MyError.anError)  
  
    // 2  
    subject  
        .subscribe {  
            print($0)  
        }  
        .disposed(by: disposeBag)  
}
```



ReplaySubject

```
func replaySubjectTest() {  
    // 1  
    let subject = ReplaySubject<String>.create(bufferSize: 2)  
  
    let disposeBag = DisposeBag()  
  
    // 2  
    subject.onNext("1")  
    subject.onNext("2")  
    subject.onNext("3")  
  
    // 3  
    subject  
        .subscribe {  
            print($0)  
        }  
        .disposed(by: disposeBag)
```

```
        subject  
            .subscribe {  
                print($0)  
            }  
            .disposed(by: disposeBag)  
  
    subject.onNext("4")  
    subject.onError(MyError.anError)  
    subject.dispose()  
  
    subject  
        .subscribe {  
            print($0)  
        }  
        .disposed(by: disposeBag)  
}
```



Lab - 1

BehaviorSubject, PublishSubject, ReplaySubject



flatMap, flatMapFirst, flatMapLatest - 1

```
func flatMapOneObservableTest() {  
    struct Player {  
        var score: Int  
    }  
  
    var 🧑 = Player(score: 80)  
    var 🧒 = Player(score: 90)  
  
    let player = BehaviorRelay(value: 🧑)  
  
    player.asObservable()  
        .map { $0.score }  
        .subscribe(onNext: { print($0) })  
        .disposed(by: self.disposeBag)  
  
    🧑.score = 85  
    player.accept(🧒)  
    🧑.score = 95  
    🧒.score = 100  
}
```



flatMap, flatMapFirst, flatMapLatest - 2

```
func flatMapRelayTest() {  
    struct Player {  
        var score: BehaviorRelay<Int>  
    }  
  
    let 🧑 = Player(score: BehaviorRelay(value: 80))  
    let 🧑 = Player(score: BehaviorRelay(value: 90))  
  
    let player = BehaviorRelay(value: 🧑)  
  
    player.asObservable()  
        .flatMap { $0.score.asObservable() }  
        .subscribe(onNext: { print($0) })  
        .disposed(by: self.disposeBag)  
  
    🧑.score.accept(85)  
    player.accept(🧑)  
    🧑.score.accept(95)  
    🧑.score.accept(100)  
}
```



flatMap, flatMapFirst, flatMapLatest - 3

```
func flatMapFirstTest() {  
    struct Player {  
        var score: BehaviorRelay<Int>  
    }  
  
    let 🧑 = Player(score: BehaviorRelay(value: 80))  
    let 🧒 = Player(score: BehaviorRelay(value: 90))  
  
    let player = BehaviorRelay(value: 🧑)  
  
    player.asObservable()  
        .flatMapFirst { $0.score.asObservable() }  
        .subscribe(onNext: { print($0) })  
        .disposed(by: self.disposeBag)  
  
    🧑.score.accept(85)  
    player.accept(🧒)  
    🧑.score.accept(95)  
    🧒.score.accept(100)  
}
```



flatMap, flatMapFirst, flatMapLatest - 4

```
func flatMapLatestTest() {  
    struct Player {  
        var score: BehaviorRelay<Int>  
    }  
  
    let 🧑 = Player(score: BehaviorRelay(value: 80))  
    let 🧑 = Player(score: BehaviorRelay(value: 90))  
  
    let player = BehaviorRelay(value: 🧑)  
  
    player.asObservable()  
        .flatMapLatest { $0.score.asObservable() }  
        .subscribe(onNext: { print($0) })  
        .disposed(by: self.disposeBag)  
  
    🧑.score.accept(85)  
    player.accept(🧑)  
    🧑.score.accept(95)  
    🧑.score.accept(100)  
}
```



Lab -2

`flatMap`, `flatMapFirst`, `flatMapLatest`