Rx의 쓱싹해보기

Swift

https://www.notion.so/RxSwift-59081ae144bc4c33807954cb2ce36426

갓병찬선배님의 자료를 토대로 발표합니다☺️

ReactiveX란?

An API for asynchronous programming with observable streams

Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Lua: RxLua
- Ruby: Rx.rb
- Python: RxPY
- Go: RxGo
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift
- PHP: RxPHP
- Elixir: reaxive
- Dart: RxDart

ReactiveX for platforms and frameworks

- RxNetty
- RxAndroid
- RxCocoa

We use ReactiveX

Applied Duality

Microsoft

NETFLIX

GitHub

SOUNDCLOUD

Trello

treehouse

SeatGeek

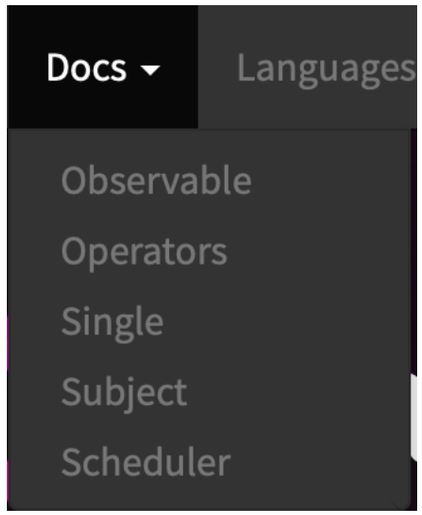
odVoo

Couchbase

futurice

O.C.TANNER

airbnb


Docs
Languages
Observable
Operators
Single
Subject
Scheduler

Observable이란?

Observable은 통로, 흐름이다.
무엇이 흐르느냐?
Event 라는 것이 흐르는 통로이다.

Event란?

RxSwift에서의 Event는 Enum타입

```
public enum Event<Element> {
    /// Next element is produced.
    case next(Element)

    /// Sequence terminated with an error.
    case error(Swift.Error)

    /// Sequence completed successfully.
    case completed
}
```

```swift
public enum Event<Element> {
    /// Next element is produced.
    case next(Element)

    /// Sequence terminated with an error.
    case error(Swift.Error)

    /// Sequence completed successfully.
    case completed
}
```

.next: 어떠한 값이 올 때

.error: 오류가 났을 때

.complete: 더이상 아무것도 오지 않을 때

Observable의 다른 모습

Single

Maybe

Completable

Single

.next or .error

Maybe

.next or .error or .complete

Completable

.error or .complete

Subscribe란?

말그대로구독

Subscribe란?

```
// 보통 Observable을 사용할 때 이런 형태로 사용합니다.
let observable: Observable<String>
let single: Single<String>
let maybe: Maybe<String>
let completable: Completable

// 위의 모두는 Event라는 것을 떨어뜨립니다.
// Subscribe의 종류 1 -> 위에서 설명한 Event가 떨어집니다.
observable.subscribe({event: Event<String> -> Void})

// Subscribe의 종류 2 -> Event의 타입에 맞추어 각각의 함수를 호출해줍니다.
observable.subscribe(
  onNext: { value: String -> Void }, // Event가 .next일 때 .next안의 Value를 반환
  onError: { error: Error -> Void }, // Event가 .error일 때 .error안의 Error를 반환
  onCompleted: { _ -> Void },        // Event가 .complete일 때 함수 호출
  onDisposed: { // 이 부분은 나중에! }
)
```

.subscribe함수가 호출되기 전까지는 Observable의 Event는 흐르지 않는다.

.subscribe함수가 호출되기 전까지 Observable은 Event 설계도(?)

클래스와 인스턴스같은 느낌적인 느낌

Subject란?

연결고리

Subject란?

```
// 보통 다음과 같이 사용됩니다.
let observable1: Observable<Int>
let observable2: Observable<Int>
let observable3: Observable<Int>
let subject = PublishSubject<Int>()

// observable의 Event를 subject가 대신 받아서
observable1.subscribe(subject)
observable2.subscribe(subject)
observable3.subscribe(subject)

// subject의 subscribe 함수로 받도록 합니다.
subject.subscribe()
```

Scheduler란?

스레드

Operator란?

Observable에 의해 방출되는 이벤트를 변환하고 처리하여 대응할 수 있게 해준다.

Operator를 크게 세그룹으로 나누면

Filtering Operator

Transforming Operator

Combining Operator

간단한 RxSwift 예제 구경하기

Q&A