




ReactiveX

 Reactive Programming by @kiboom

저자분께 허가를 받아 수정 및 사용하였습니다.

허가받지 않은 복제(복사), 전송, 수정 및 배포를 금합니다.



비동기적인 데이터 흐름을 다루는 프로그래밍

Reactive Programming이란?

- Rx를 알려면 **Reactive Programming**이 뭔지 먼저 알아야!
 - **Rx**: Reactive Extension
- **Reactive Programming**
 - 연속적이고 비동기적으로 전달되는 데이터에
 - **즉각 반응**할 수 있도록 하는 프로그래밍 기법

 유튜브 구독과 알림을 떠올리면 된다.



유튜브 특징	구독과 알림을 누르면
• 연속적이다 : 새로운 영상들이 계속 도착한다.	• 관심 채널에서 새로운 영상 을 계속 받아볼 수 있다.
• 비동기적이다 : 다음 영상이 언제 도착할지 모른다.	• 새로운 영상이 올라오면 즉각 알림 을 받아볼 수 있다.

- **Reactive Programming** 은
 - **연속적(Data Stream)**이고 **비동기적(Asynchronous)**으로 변화하는 데이터들에
 - **잘 반응하기 위한 (Reactive)** 프로그래밍 기법이다.
 - 이름이 **Reactive Programming** 인 이유는, 이런 종류의 데이터에 즉각 반응할 수 있기 때문.

연속적이고 비동기적인 데이터란?

- **Reactive Programming** 을 유튜브에 비유했는데, 실제로는 어떤 사례가 있을까?
 - **API 응답**: 응답값이 비동기적으로 온다.
 - **음성영상 스트리밍**: 미디어 데이터가 연속적으로 온다.
 - **사용자 입력**: 마우스 제스처나 키보드 입력이 비동기적으로 연속적으로 발생한다.
 - **앱 내 이벤트**: 앱에서 각종 이벤트가 계속 발생한다. (검색, 필터링 등)

Reactive Programming 구현법

▼ 구현 원리

- **연속적이고 비동기적인** 데이터 흐름에 즉각 반응하려면?
 - 유튜브의 구독과 알림의 원리를 떠올려보자!
- 데이터를 구독함으로써, 새로운 소식이 도착할 때마다, 특정 동작을 즉시 실행하면 된다.

▼ 연속적인 데이터

- **구독** 을 통해 새로운 이벤트를 **연속적으로** 받아보자!
 - 여러 소식들을 조합해서 새로운 데이터 흐름을 만들 수도 있다.

▼ 비동기적인 데이터

- **알림**을 통해 새로운 이벤트에 **즉각 대응**할 수 있다.
 - 새로운 이벤트가 발생할 때까지 마냥 기다리지 않아도 된다. (비동기 처리)

▼ 옛날에는 어떻게 처리했을까? (feat. 콜백 방식)

- 비동기적 이벤트에는 대응 가능하나,
 - 여러 이벤트를 조합하기 어려움 (callback hell)

```
asyncCallOne {  
  asyncCallTwo {  
    asyncCallThree {  
      // ...  
    }  
  }  
}
```

- 연속적인 이벤트에는 대응 하기 어려움.
 - 특히 여러 개의 데이터 흐름을 제어하기에 매우 어려움. (ex. 사용자 입력 이벤트)

▼ 관련 프레임워크

- **Stream**: Dart에서 자체 제공하는 **문법**
- **Rx**: Reactive Programming을 도와주는 **써드파티 프레임워크**
 - 단순히 데이터의 흐름을 처리하는 것 뿐만 아니라,
데이터의 흐름을 조작할 수 있음. (밑에서 살펴볼 예정)
 - **RxDart**: Dart에서 제공하는 Stream을 더욱 확장한 프레임워크

Reactive Programming 개념들

Reactive Programming 개념들을 익혀보자.

▼ **Stream**

▼ **구독 가능한 데이터 흐름**

- Rx에서는 Observable이라는 클래스로 표현함.

- Dart에서는 Stream이라는 클래스가 이미 있어서,
Rx Dart에서는 Observable을 따로 안 만듦.

▼ 기능1: 연속적인 데이터를 표현함

- **Broadcast Stream**: 구독을 하기 전부터 이미 데이터가 송출되고 있음.
 - ex) 라이브 방송
 - Rx) *Hot Observable*
- **Single Stream**: 구독을 요청하면 비로소 데이터가 송출됨.
 - ex) VOD 재생
 - Rx) *Cold Observable*
- Rx) *Observable*

▼ 기능2: 구독해서 받아볼 수 있음

- **listen**: 구독 시작!
 - **onData**: 신문을 잘 배송 받았을 때 동작 😊
 - **onError**: 신문 배송에 문제가 생겼을 때 동작 😬
 - **onDone**: 신문이 폐간됐을 때 동작 😊
 - Rx) *subscribe*
- **cancel**: 구독 취소!
 - **구독보다 더 중요한 건 구독 취소!!!**
 - 신문 구독을 안 끊으면 돈이 계속 나가듯이,
스트림 구독도 제 때 종료하지 않으면 메모리 누수가...
 - Rx) *dispose*

▼ Future vs Stream

- 둘 다 비동기적으로 도착하는 데이터를 다루기 위한 API
- Future는 데이터 하나만 다루고, Stream은 연속적인 데이터를 다룸
- 둘 다 **FutureBuilder**와 **StreamBuilder**가 있음.

▼ **Stream 연산자**

▼ 데이터 흐름을 조작함

- Rx에서 제공해주는 기능이자, Rx를 사용하는 가장 큰 이유!
- RxDart를 설치하면 Stream의 데이터 흐름을 조작할 수 있다!

▼ 기능1: 띄엄띄엄 받아볼 수 있다

- `take`: 처음 몇 개만 받고 구독 종료! (ex. 처음 3개만 받고 종료)
- `skip`: 처음 몇 개는 건너뛰고 받아보기 (ex. 처음 2개는 건너뛰기)
- `filter`: 조건에 맞는 애들만 받아보기 (ex. 0 이상의 값만 받기)
- `debounce`: 마지막 이벤트로부터 몇 초 동안 이벤트가 더 없으면, 마지막 이벤트만 반환.
- `throttle`: 첫 이벤트로부터 몇 초 후에, 무조건 마지막 이벤트만 반환.

▼ 기능2: 개조해서 받아볼 수 있다

- `map`: 원하는 형태로 개조해서 받아보기 (ex. Int를 String으로 바꿔서 받기)

▼ 기능3: 조합해서 받아볼 수 있다

- 여러 물줄기를 하나로 합치는 것처럼,
여러 데이터 스트림을 조합해서 하나의 스트림으로 받아볼 수 있다.
- `merge`: 두 스트림의 이벤트를 순서대로 합치기
- `join(zip)`: 두 스트림의 이벤트를 각각 짝지어서 합치기

▼ 참고 사이트

RxMarbles: Interactive diagrams of Rx Observables

Learn, build, and test Rx functions on Observables

🔗 <https://rxmarbles.com/>

▼ StreamController

▼ 데이터 흐름을 제어함

- Stream이 신문 구독자라면, StreamController는 신문사
 - Stream이 데이터 수신자라면, StreamController는 데이터 제공자 관점에서 API가 제공됨
- Rx에서는 `Subject` 라는 클래스로 표현함
 - 실제로 RxDart에서 Subject는 StreamController의 하위 클래스임

- (밑에서도 살펴보겠지만) StreamController보다 훨씬 풍부한 기능을 제공한다

▼ 기능1: 데이터의 흐름을 제어한다

- `add`: 새로운 데이터를 추가할 수 있다.
- `close`: 데이터를 더이상 흘려보내지 않는다. (신문 폐간)
- Rx) `Subject`

▼ 기능2: 구독이 발생했을 때 데이터를 배송해준다

- Rx를 사용하면 첫 배송을 어떻게 할 지 설정할 수 있다!
 - `PublishSubject`: “다음달 호부터 배송해드릴게요!”
 - `BehaviorSubject`: “이번달 호부터 먼저 받아가세요!”
 - `RelaySubject`: “최근 3개월 호 먼저 받아가세요!”

▼ Stream vs StreamController

- 둘 다 연속적인 데이터를 다루는 클래스
- Stream은 신문을 받아보는 `구독자 관점`, StreamController는 신문을 배송하는 `신문사 관점`에서 API 제공.
- StreamController로부터 Stream 생성 가능.
- Observable vs Subject

▼ StreamController vs Subject

- StreamController는 Dart에서 기본 제공하는 클래스, Subject는 RxDart에서 제공하는 클래스.
- StreamController의 기능을 더욱 확장한 것이 Subject
 - 구독이 발생했을 때 첫 배송을 어떻게 해줄 것인지 상세하게 설정할 수 있음!

GetX에서는 Rx 자체 구현

- `obs`
- `StreamController`
- `ObxValue`
- `Obx`

Reactive Programming 사례

어디에 적용해보면 좋을까?

▼ 비동기적인 변화에 즉각 대응하고 싶을 때

- **Presenter**
 - 모델에 변화가 있을 때, 뷰에 즉각 알려줌.
- **EventBus**
 - 앱에서 발생하는 각종 이벤트를 구독할 수 있음.
여러 화면에서 동시에 대응 가능.
 - ex) 스크롤, 메뉴 이동, 프로필 정보 수정
- **StreamBuilder**
 - Stream으로부터 전달 받는 데이터에 따라 위젯의 모양을 변경함.

▼ 연속적인 데이터 흐름을 제어하고 싶을 때

- 토글 버튼(ON/OFF) 상태에 따른 API 조회
- [토스] 주식 검색 화면 키워드 적용


참고

ReactiveX

 <https://reactivex.io/>

RxMarbles: Interactive diagrams of Rx Observables

Learn, build, and test Rx functions on Observables

 <https://rxmarbles.com/>