RxJS

What is RxJS?

Reactive Extensions For JavaScript
라이브러리

"Reactive Programming" *지원을 위한 확장*

이벤트 스트림을 ***"Observable"*** *이라는 객체로 표현한 후*
비동기 이벤트 기반의 프로그램 작성을 돕는다

"Observer Pattern" *으로 비동기 구현*

What is Reactive Programming?

- **"리액티브 프로그래밍"**이란 이벤트나 배열 같은 데이터 스트림을 **비동기**로 처리해 변화에 유연하게 반응하는 **프로그래밍 패러다임**

- **외부 통신하는 방식**
  a. **Pull 시나리오**
     외부에서 명령하여 응답받고 처리한다.
     데이터를 가지고 오기 위해서는 계속 호출해야 한다.

  b. **Push 시나리오**
     외부에서 명령하고 기다리지 않고, 응답이 오면 그때 반응하여 처리한다
     데이터를 가지고 오기 위해서 구독해야 한다.

- **"리액티브 프로그래밍"**은 **Push 시나리오**를 채택

RxJS는
일관된 방식으로
안전하게
데이터 흐름을
처리하는
라이브러리

하나의 방식으로
적용하자

인터페이스 단일화

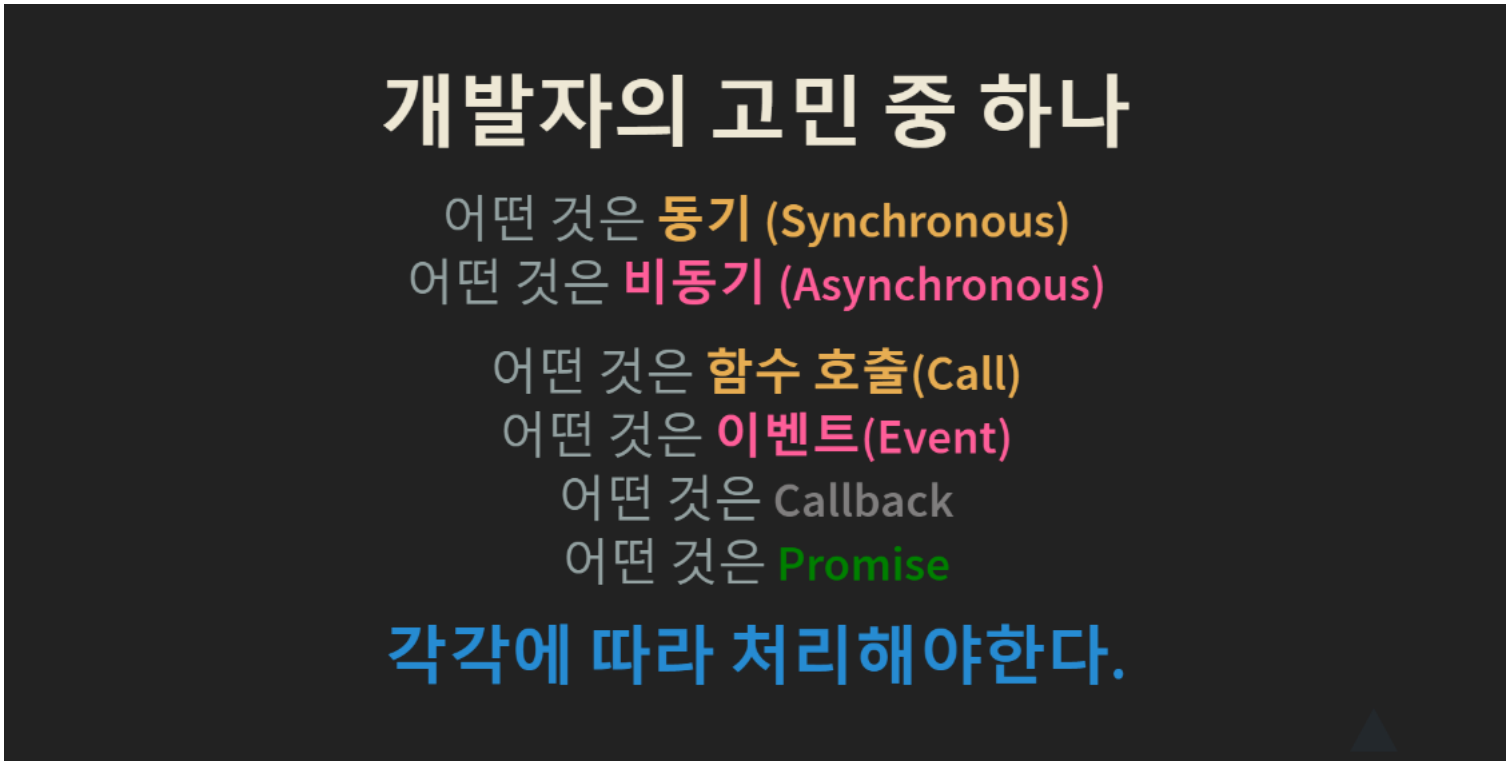
개발자가 처리하는

입력값(Input)은 어떤 것들이 있는가?

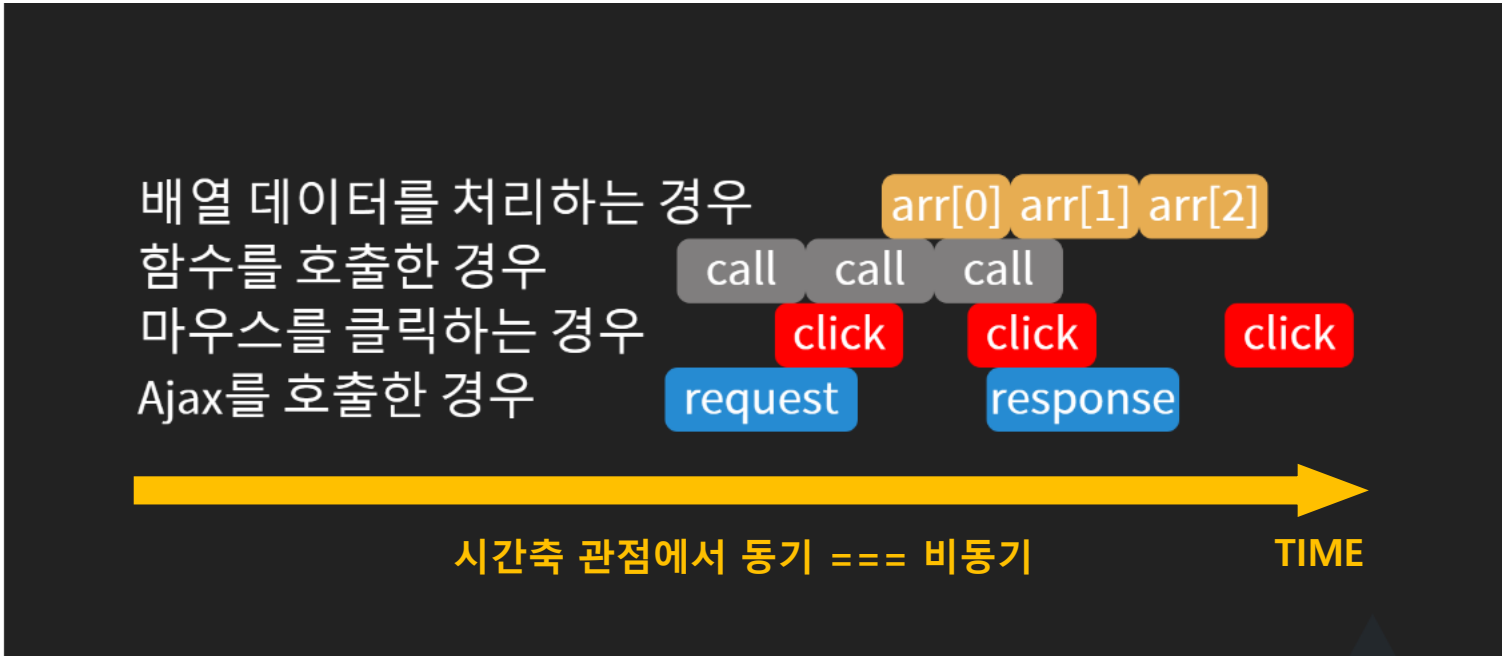
배열 데이터도 입력값으로
함수 반환값 도 입력값으로

키보드를 누르는 것도 입력값으로
마우스를 움직이는 것도 입력값으로

원격지의 데이터도 입력값으로
DB 데이터도 입력값으로


개발자의 고민 중 하나
어떤 것은 동기 (Synchronous)
어떤 것은 비동기 (Asynchronous)
어떤 것은 함수 호출(Call)
어떤 것은 이벤트(Event)
어떤 것은 Callback
어떤 것은 Promise
각각에 따라 처리해야한다.

RxJS


배열 데이터를 처리하는 경우
함수를 호출한 경우
마우스를 클릭하는 경우
Ajax를 호출한 경우
arr[0]
arr[1]
arr[2]
call
call
call
click
click
click
request
response
시간축 관점에서 동기 === 비동기
TIME

개발자의 고민 중 하나

~~어떤 것은 동기 (Synchronous)~~
~~어떤 것은 비동기 (Asynchronous)~~
~~어떤 것은 함수 호출(Call)~~
~~어떤 것은 이벤트(Event)~~
~~어떤 것은 Callback~~
~~어떤 것은 Promise~~

모두 Observable로 처리한다

Observer Pattern을 적용하자

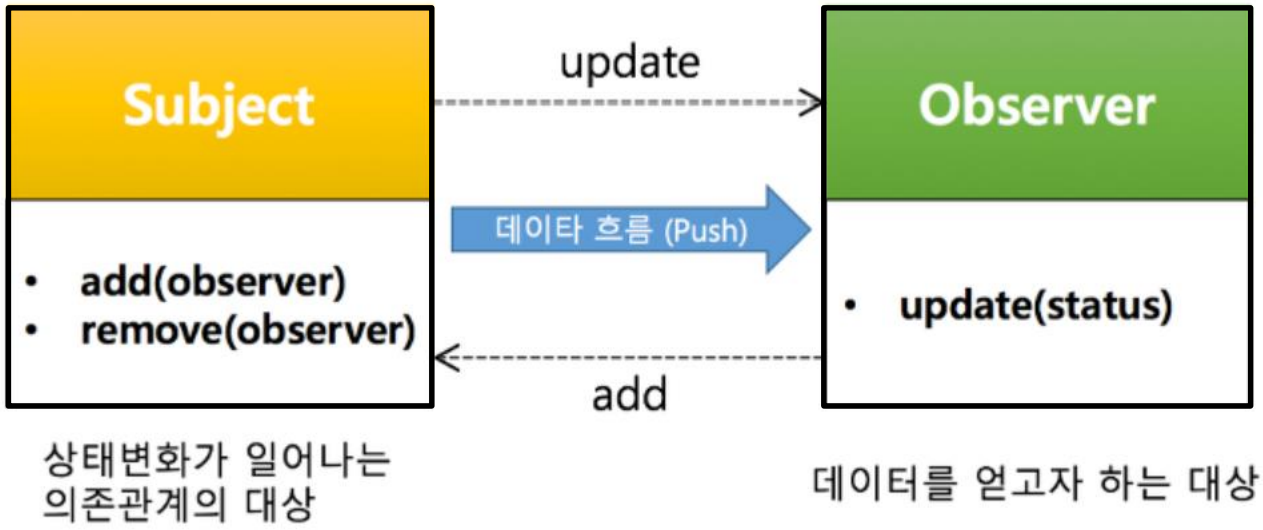
상태 자동 전파
Loosely Coupling

RxJS

일반적인 비동기 이벤트

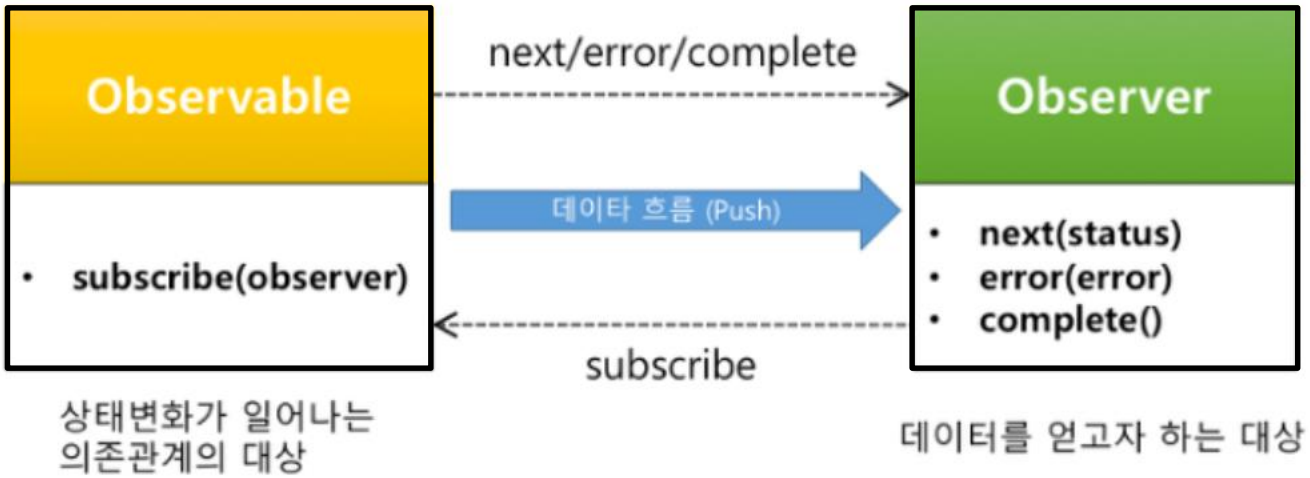
```
// javascript
function event() {
// ...
};

const e = document.getElementsById('Button');
e.addEventListener('click', event)
```

Observer Pattern



출처: http://sculove.github.io/blog/2017/10/21/shoulduserxjs/

RxJS



출처: http://sculove.github.io/blog/2017/10/21/shoulduserxjs/

고차 함수를
제공한다

"함수형 프로그래밍"을
지향한다
(순수함수, 일급함수)

Take a look

RxJS

RxJS 개발 방법

1. 데이터 소스를 Observable로 생성

2. Observable의 operator를 사용
   데이터를 변경. 추출. 합침. 분리

3. 원하는 데이터를 받아 처리하는 Observer를 만든다.

4. Observable의 subscribe를 통해 Observer를 등록한다.

5. Observable 구독을 정지하고 자원을 해지한다.

RxJS

RxJS 예제 코드

```
// javascript
Observable.create((observer) => { ← Observable 생성
  try {
    observer.next('item'); ← callback to receive noti of 'next'
  } catch(e) {
    observer.error(e); ← callback to receive noti of 'error'
  } finally {
    observer.complete(); ← callback to receive noti of 'complete'
  }
}).subscribe( ← 구독 시, Observer 전달
  (x) => console.log(x),
  (err) => console.error(err),
  () => console.log('complete')
)
```

RxJS Observer

- **observer.next()** :
  *Observable 구독자에게 **데이터를 전달**한다.*

- **observer.complete()** :
  *Observable 구독자에게 **완료** 되었음을 알린다. next는 더 이상 데이터를 전달하지 않는다.*

- **observer.error()** :
  *Observable 구독자에게 **에러를 전달**한다. 이후에 next 및 complete 이벤트가 발생하지 않는다.*

RxJS Observable

- **생성**
  Observable.create()
  생성시점에는 어떠한 이벤트도 발생되지 않는다.

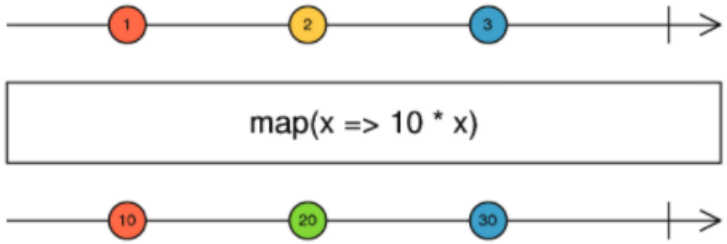
- **구독**
  Observable.subscribe()
  구독시점에 이벤트를 구독할 수 있다.

- **구독 해제**
  Observable.unsubscribe()
  구독 해제 시점에 구독하고 있는 모든 대상의 구독을 종료한다.

rxjs/operator

https://www.learnrxjs.io/learn-rxjs/operators

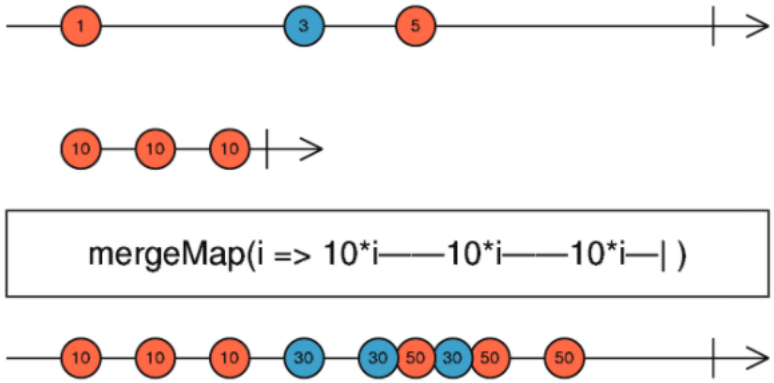
RxJS Observable operator (map)



map operator의 동작 과정

map operator는 아래 예제와 같이 발생하는 값을 다른 값으로 변환, 대체해야 하는 경우에 적합

RxJS Observable operator (flatMap, mergeMap)

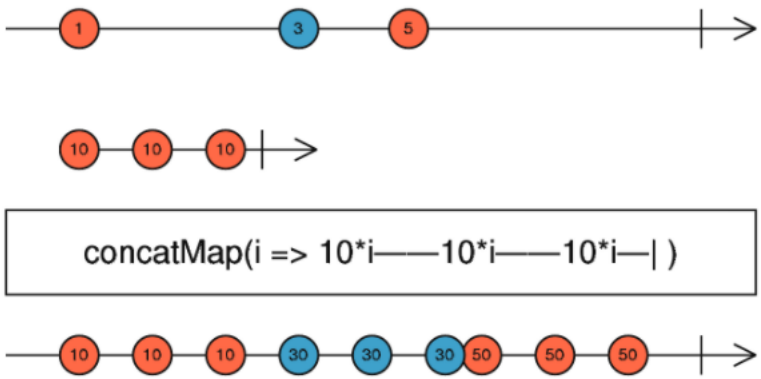


mergeMap operator의 동작 과정

두 번째 Observable의 요소 방출이 다 끝나기도 전에 세 번째 Observable의 요소가 방출되면 Observable 의 순서와 상관 없이 먼저 발생한 요소 순서로 방출된다.

```
const letters = of(1, 3, 5);
const intervalTime = { 1: 100, 3: 300, 5: 300 };

letters.pipe(
  mergeMap(x => interval(intervalTime[x]).pipe(map(i => x * 10), take(3))),
  toArray()
).subscribe(x => console.log(x));
```

RxJS Observable operator (concatMap)

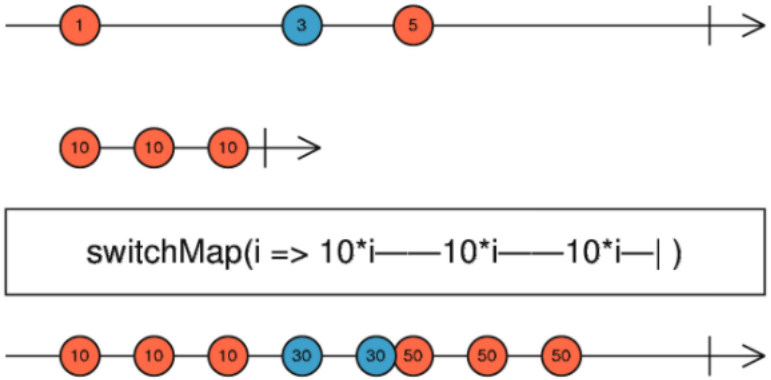


concatMap operator의 동작 과정

그렇다면 요소의 순서가 섞이지 않고 Observable 의 순서대로 요소를 방출하고 싶으면 어떻게 해야 할까? 그 경우 concatMap 을 사용하면 된다. concatMap은 Observable의 순서대로 요소가 방출된다.

```
const letters = of(1, 3, 5);
const intervalTime = { 1: 100, 3: 300, 5: 300 };

letters.pipe(
  concatMap(x => interval(intervalTime[x]).pipe(map(i => x * 10), take(3))),
  toArray()
).subscribe(x => console.log(x));
```

RxJS Observable operator (switchMap)



switchMap의 동작 과정

구독중이던 Observable이 끝나기 전에 새로운 Observable을 구독하게 되면 이전에 구독중이던 Observable 의 구독을 취소하고 다음 Observable 구독을 시작

```
const delayTime = { 1: 100, 3: 300, 5: 200 };
const letters = of(1, 3, 5).pipe(concatMap(x => of(x).pipe(delay(delayTime[x]))));

letters.pipe(
  switchMap(x => interval(100).pipe(map(_ => x * 10), take(3))),
  toArray()
).subscribe(console.log);
```