RX로 복잡해진 비동기 코드 정리하기

- hckrmoon

“

WHY

WHY

예전의 웹은 지금과는 사뭇 다른 형태였다.

WHY

- *example*

Reactive programming

From Wikipedia, the free encyclopedia

In computing, **reactive programming** is a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.

For example, in an *imperative* programming setting, $a := b + c$ would mean that a is being assigned the result of $b + c$ in the instant the expression is evaluated, and later, the values of b and c can be changed with no effect on the value of a.

However, in reactive programming, the value of a would be automatically updated based on the new values.

A modern spreadsheet program is an example of reactive programming. Spreadsheet cells can contain literal values, or formulas such as "=B1+C1" that are evaluated based on the current value of other cells. Whenever the value of the other cells change, the value of the cell containing the formula is automatically updated.

Another example is a hardware description language such as Verilog. In this case, reactive programming allows changes to be modeled as they propagate through a circuit.

Reactive programming has foremost been proposed as a way to simplify the creation of interactive user interfaces, animations in real time systems, but is essentially a general programming paradigm.

For example, in a Model-view-controller architecture, reactive programming can allow changes in the underlying model to automatically be reflected in the view, and vice versa.[1]

Contents [hide]

1 Concepts
1.1 Degrees of explicitness
1.2 Static or Dynamic
1.3 Higher-order reactive programming

WHY

Static

Dynamic

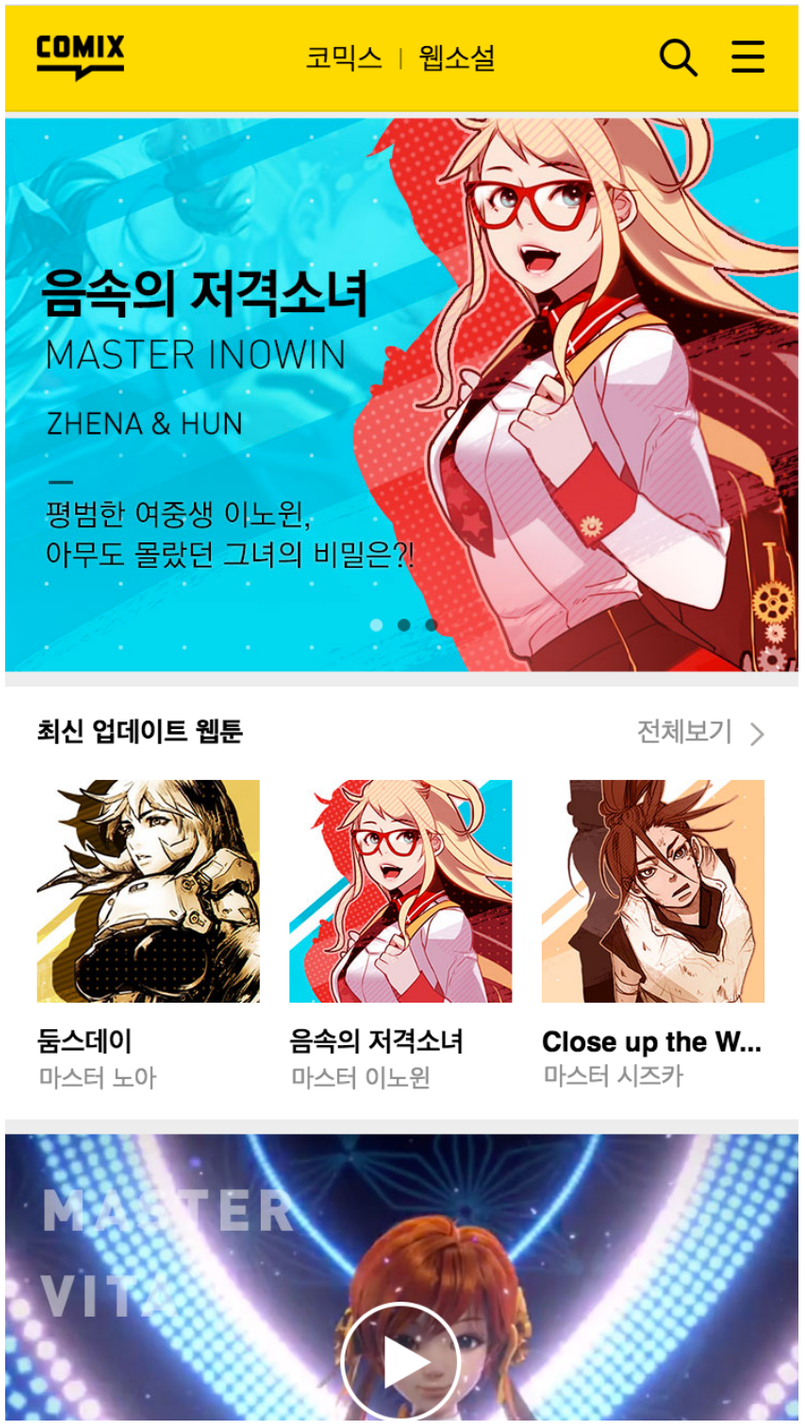
Better UI/UX

User Interaction

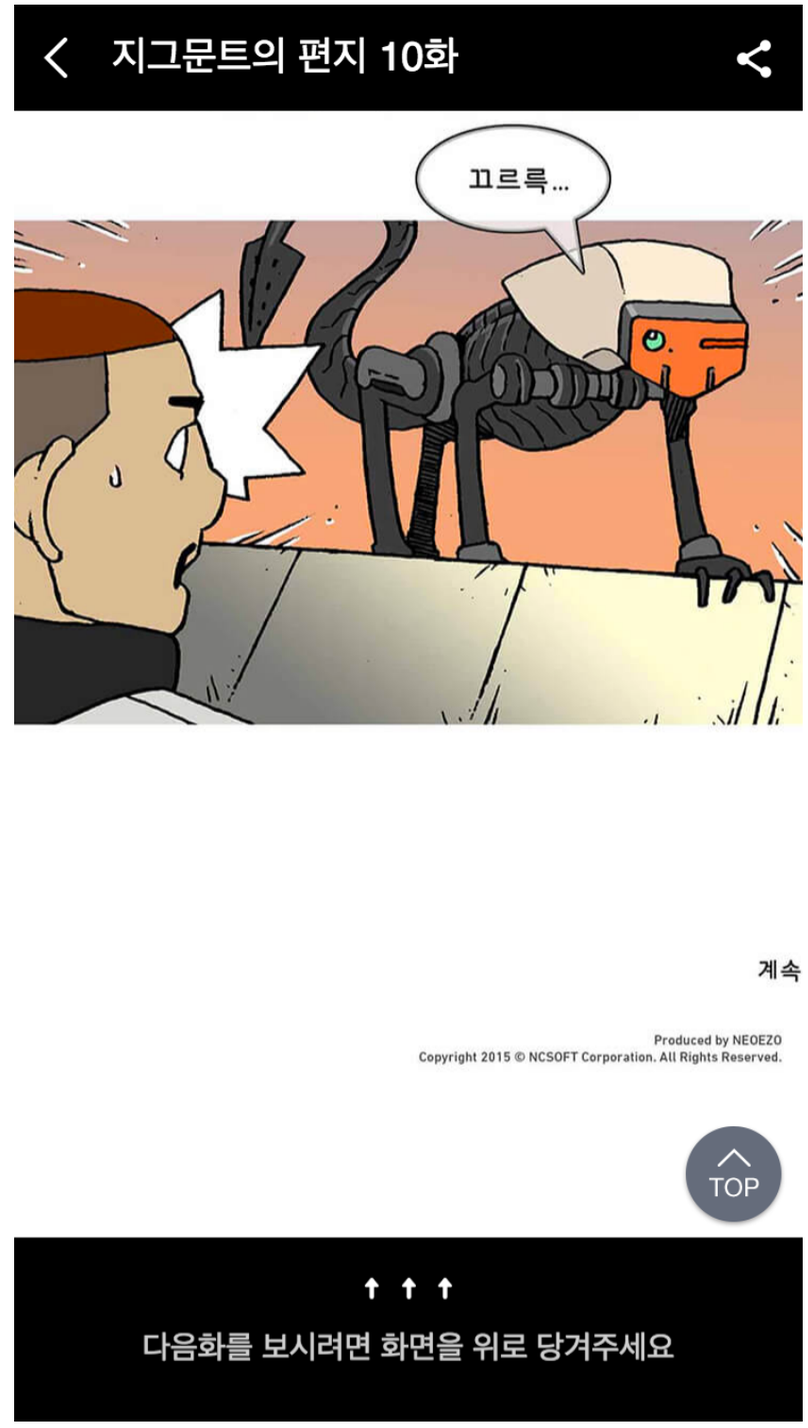
WHY

WHY

Rich UI/UX



User Interaction



(자료제공: 엔씨코믹스)

WHY

자연스럽게 Client 에서 처리해야되는 것도 많아졌다.

- User Input (DOM Event)
- Animations
- AJAX
- Web Socket

WHY

그리고 이것들은 모두 비동기적으로 처리된다.

WHY

ECMAScript 5 까지는 모두 Callback 패턴에 의존했고 많은 로직들이
해당 함수에서 처리된다.

```
1  var btn = document.getElementById('button');
2
3  btn.addEventListener('click', function () {
4    console.log('Hello, Universe');
5  });
```

WHY

하지만 Callback 함수에서 실행해야되는 로직 중에는 특정 조건을 거쳐야 되는 경우가 종종있다.

```
var btn = document.getElementById('button');

btn.addEventListener('click', function () {
  if (window.__VALID_USER__) {
    console.log('Hello, Universe');
  }
});
```

WHY

생각해보면 Callback 함수가 실행되는 시점은 아직 정제되지 않은 시점이라고 할 수 있다.

WHY

예를 들면, 우리가 처리해야 되는 시점이 “scroll, touchmove 이벤트가 발생하면서 현재 스크롤 위치가 맨 위가 아닐 때”라고 하면 다음과 같은 코드가 나오고 `realCallback` 함수가 실행되는 시점이 바로 우리가 처리해야 되는 시점이다.

```
el.addEventListener('scroll', function () {
  if (pageYOffset !== 0) {
    realCallback();
  }
});

el.addEventListener('touchmove', function () {
  if (pageYOffset !== 0) {
    realCallback();
  }
});
```

WHY

DOM 이벤트 뿐만 아니라 ajax callback 에서의 처리도 마찬가지다.
우리는 `showSomethingCool` 함수가 호출되는 시점말고는 관심이 없다.

```
$.get('http://get-user-info.com').then(function(user) {
  if (user.didPay) {
    showSomethingCool();
  }
});
```

WHY

그럼 `realCallback/showSomethingCool` 함수가 실행되는 시점을
선언적으로 표현할 수 있으면 좋지 않을까 ?

WHY

가령 이렇게 말이다.

(물론 커스텀이벤트들을 만들어서 사용할 순 있지만 그러지말자.)

```
el.addEventListener('scrollWhenScrollNotAtTop', realCallback);
$.ajax('http://get-user-info.com').ifUserDidPay(showSomethingColl);
```

“

Refine

REFINE

물론 현실세계에서는 앞의 조건보다 훨씬 복잡한 조건들이 들어간다.
그렇다면 이를 좀 더 편하게 정제할 순 없을까?

REFINE

다시 생각해보면 비동기적인 시점이 아니라 일반 데이터를 정제할 때
우리는 어떻게 했는지 되새겨 볼 필요가 있다.

```
var arr = [1, 2, 3, 4, 5, 6];

var filteredArr = arr.filter(function(item) {
  return item % 2 === 0;
});

var mappedFilteredArr = filteredArr.map(function (item) {
  return item * 2;
});
```

REFINE

Array Data를 Array Method를 통해 선언적이고 효율적으로 처리하고 있다.
비동기적으로 발생하는 이벤트도 하나의 Data 단위로 생각하고 처리하면
똑같이 선언적이고 효율적으로 처리할 수 있지 않을까?

REFINE

해보자!

```
var PromiseArray = [];

el.addEventListener('scroll', function(e) {
  PromiseArray.push(new Promise(...));
  forEachForPromise();
});

function forEachForPromises() {
  PromiseArray.filter(function(promise) {
    promise.then(function(data) {
      if (...) {}
    });

    return ???;
  });
}
```

REFINE

… 그만해보도록하자.

비동기적으로 들어온 data 를 Promise 로 감싸고 있을 뿐,

data 간의 시간을 Array Type 으로 표현하지는 못한다.

REFINE

이럴 땐 항상 새로운 개념이 필요하기 마련이다.

“

Observable

OBSERVABLE

Observable 은 비동기적인 Event 나 Action 을 제어한다.

OBSERVABLE

observer 가 Observable 객체가 뱉어내는 값에 반응하는 구조로 만들어져있다.

```
var observableStream = Rx.Observable.create(function(observer) {
  observer.onNext(1);
});

observableStream.subscribe(function(data) {
  console.log('data', data);
});
```

OBSERVABLE

이 구조 어딘가 익숙하다.

OBSERVABLE

우리가 자주 사용하던 Promise 와 다를바 없어 보인다.

```
1  var promise = new Promise(function(resolve) {
2    resolve(1);
3  });
4
5  promise.then(function(data) {
6    console.lod('data', data);
7  });
```

OBSERVABLE

중요한 차이점이라고 한다면, Observable 은 하나 이상의 값을 계속 뱉어낼 수 있다.

```
var observableStream = Rx.Observable.create(function (observer) {
  var url1 = 'http://hello-universe.com';
  var url2 = 'http://hello-world.com';
  var defaultValue = {};

  observer.onNext(defaultValue);

  $.get(url1).then(function(data) {
    observer.onNext(data);
    return $.get(url2);
  }).then(function(data) {
    observer.onNext(data);
    observer.onCompleted();
  });
});
```

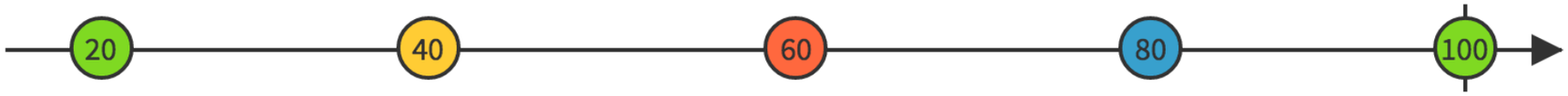
OBSERVABLE

마치 Array 가 하나 이상의 아이템을 가지듯,
하나 이상의 아이템을 비동기적으로 뱉어내는 Observable 객체

OBSERVABLE

Observable 객체가 뱉어내는 값을 타임라인으로 그려보면 배열처럼 보인다.

그럼 어떻게하면 Observable 객체를 정제할 수 있을까?



OBSERVABLE

Rx 에 바로 이 Observable 객체를 배열 데이터를 다루듯
정제할 수 있는 유용한 Operator 들이 존재한다.

OBSERVABLE

그렇다면 간단한 operator 를 통해서 DOM 이벤트를 정제해보자.

OBSERVABLE

그 전에 간단한 요구사항이 있으면 좋을 것 같다.

OBSERVABLE

- 화면에는 버튼이 있다.
- 그냥 버튼을 클릭하면 아무 반응을 하지 않는다.
- alt 키를 누른채 버튼을 클릭하면 콘솔창에 이벤트의 clientX 값을 찍는다.
- 그리고 다시 버튼을 클릭하면 해당 이벤트를 해제한다.

(무슨 프로그램인지는 모르겠다. 그냥 만들기로 한다.)

OBSERVABLE

먼저 기존의 방법으로 코드를 짜보았다.

```
var button = document.getElementById('btn');
var count = 0;

btn.addEventListener('click', function clickHandler(e) {
  if (count < 1)  {
    if (e.altKey) {
      console.log('e.clientX', e.clientX);
      count++;
    }
  } else {
    btn.removeEventListener('click', clickHandler);
  }
});
```

OBSERVABLE

Callback 함수 외부에 State(Side Effect 유발) 가지고 있고
조금만 더 복잡해지면 쉽게 따라가지 못할 것 같다.

OBSERVABLE

사실 해당 버튼을 alt 키와 함께 클릭한 이벤트 한 번만 받으면 되는 건데
뭔가 복잡해 보인다.

OBSERVABLE

그냥 선언적으로 나타내면 다 알아서 해줬으면 좋겠다.
마치 SQL 명령어처럼 ?

OBSERVABLE

처음에 SQL 명령어를 익히는데 시간은 들지만 익히고 나면 이보다 편할 수가 없었다.

이벤트도 마치 SQL 처럼 선언적으로 사용할 수 없을까 ? 이렇게 ?

“**SELECT** clientX **FROM** click **WHERE** altKey = TRUE **LIMIT** 1;”

OBSERVABLE

"Your Mouse is a Database"

사실 이 말은 Rx 를 만든 Erik Meijer 가 한 말이다.

http://queue.acm.org/detail.cfm?id=2169076

OBSERVABLE

결국 우리가 다뤄야할 Data에 정체에 대해서 다시 한번 생각해보게 된다.
메모리에 이미 존재하는 Data 뿐만 아니라 앞으로 비동기적으로
생겨날 Data에 대해서도 같은 방법의 선언적인 표현법이 필요한 것이다.

OBSERVABLE

선언적이라하면 C = A + B 라고 적어넣고 A의 값이 바뀌면 C = A + B 라고
명령을 내리는 것이 아니라 C와 A,B 의 관계를 그냥 선언하고
컴퓨터에게 맡기는 것이다.

OBSERVABLE

서론이 좀 길었지만 본론으로 바로 들어가보자.
앞의 프로그램을 이제 Rx 를 이용하여 만들어볼거다.

먼저 DOM 선택하여 특정 이벤트를 Observable 객체로 만들자.
Rx 의 fromEvent method 를 이용하면 이를 쉽게 만들어 낼 수 있다.

```
var button = document.getElementById('btn');
var clickStream = Rx.Observable.fromEvent(button, 'click');
```

OBSERVABLE

그리고 Array method 를 쿨하게 써내려가듯 이벤트를 filter method 를 이용하여 altKey 가 눌린 값만 받고, 그 중 하나만 받을거니 take(1) 으로 처리한다.
그리고 넘어온 이벤트의 clientX 값을 찍어주면 끝난다.

```
clickStream
  .filter(function (e) {
    return e.altKey;
  })
  .take(1)
  .subscribe(function(e) {
    console.log('e.clientX', e.clientX);
  });
```

OBSERVABLE

뭔가 코드가 굉장히 깔끔해진건 사실이다.
근데 clickStream ? Stream 이라는 표현이 사실 좀 낯설다.

“

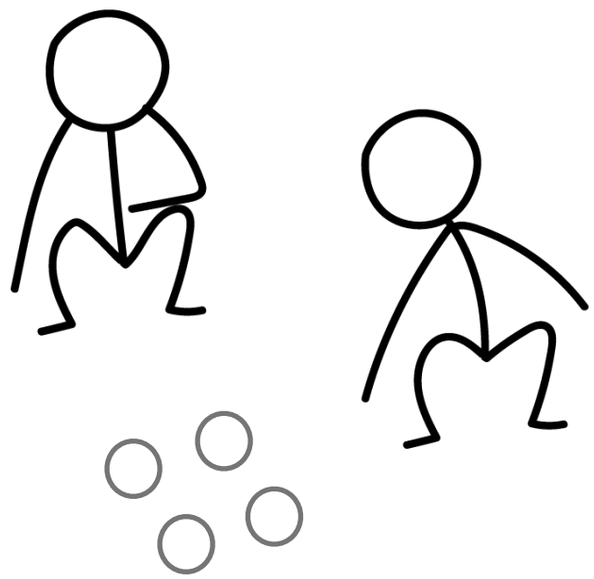
Stream

STREAM

이것을 공을 가지고 노는 것과 비교를 해본다면…

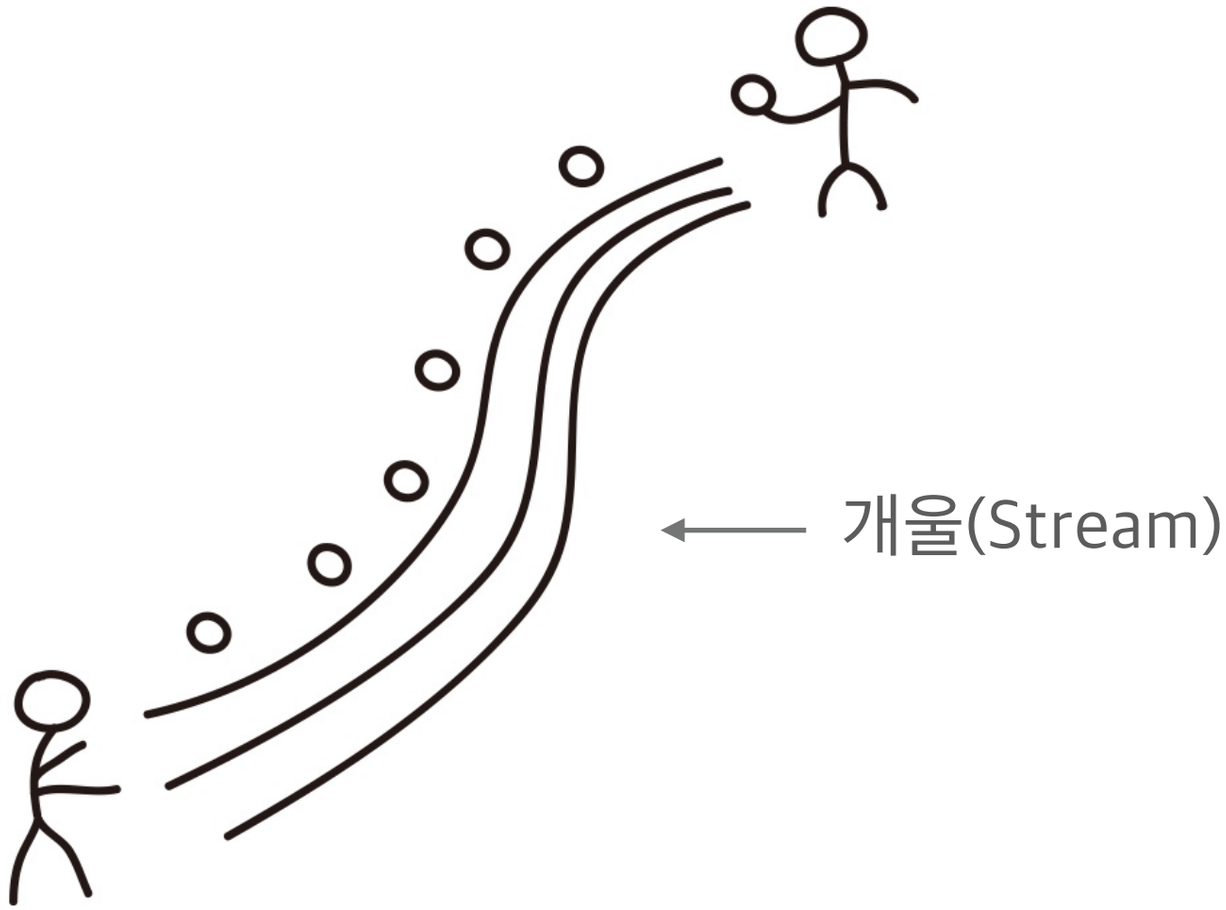
STREAM

Array Method 로 Array 을 다루는 것은 마치 제자리에서 공을 가지고 노는 것과 같다.



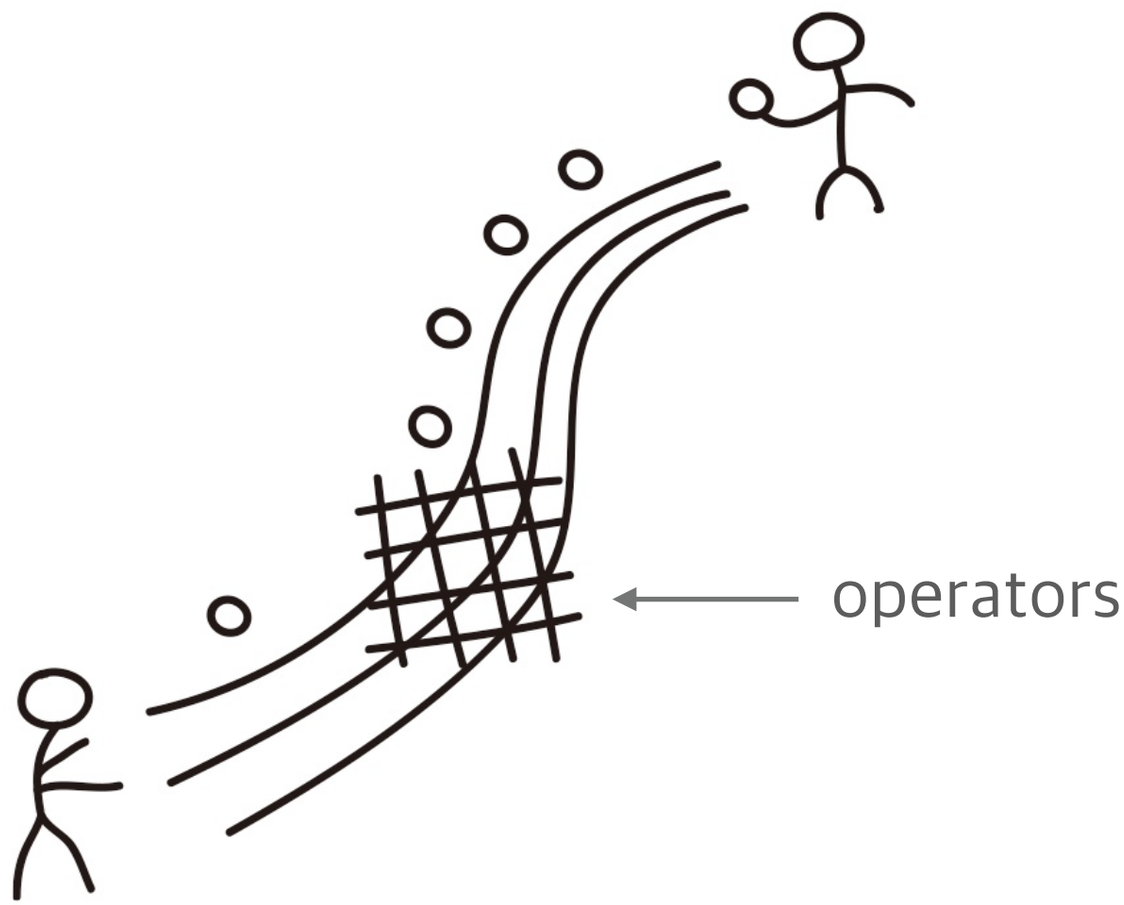
STREAM

하지만 Observable 객체를 해당 operator 로 다루는 것은 상류에서 하류로 보내는 공을 가지고 노는 것과 같다.



STREAM

Operator 를 굳이 표현하자면 필요한 공으로 걸러주는 망(?) 같은 것이라고 할까?



Conclusion

CONCLUSION

끝으로 앞의 clickStream에 operator 을 chaining 한 형태를 변수에 알맞은 이름으로 mapping 하면 앞서 우리가 맞이했던 문제를 푼 것만 같다.

CONCLUSION

`onceClickWithAltKey` 라는 Stream 이 복잡한 비동기코드를
선언적으로 표현해주고 있다.

```
var onceClickWithAltKey = clickStream.filter(function (e) {
  return e.altKey;
}).take(1)

onceClickWithAltKey.subscribe(function(e) {
  console.log('e.clientX', e.clientX);
});
```

CONCLUSION

다음의 사이트들에서 많은 정보를 얻을 수 있다.

CONCLUSION

Official Source Code

https://github.com/Reactive-Extensions/RxJS

Rx 와 관련된 리소스

http://reactivex.io/

Observable 을 시각화해서 볼 수 있는 곳

http://rxmarbles.com/

CONCLUSION

End.