Sane Asynchronous Programming with RxJS

~~Sane Asynchronous Programming with RxJS~~

Introduction to Reactive Programming with RxJS

whoami

Luís Gabriel Lima

- BSc and MSc in CS at CIn/UFPE
- Former Software Engineer at INDT
- Functional Programming enthusiast
- RxJS 5 Contributor


...


@_luisgabriel :: @luisgabriel

Reactive Programming

"Reactive programming is programming with **asynchronous data streams**"

Reactive Programming

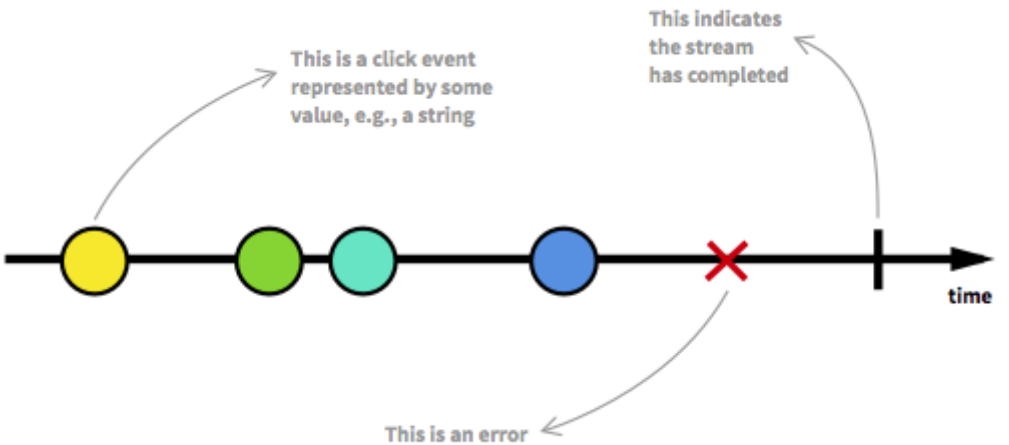
"Reactive programming is programming with **asynchronous data streams**"

A **stream** is a **sequence** of ongoing events **ordered in time**

Reactive Programming

"Reactive programming is programming with **asynchronous data streams**"

A **stream** is a **sequence** of ongoing events **ordered in time**



source: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

Reactive Extensions (Rx)

A common **idiom** to express RP across **different platforms**

Reactive Extensions (Rx)

A common **idiom** to express RP across **different platforms**

Rx.NET

RxJS

RxJava

RxSwift

...

Observable

An **object** that represents a **data stream** in Rx

Observable

An **object** that represents a **data stream** in Rx

```
const numberStream = Rx.Observable.range(1, 8);

numberStream.subscribe({
  next(x) { console.log(x); },
  error(e) { console.error(e); },
  complete() { console.log('done'); }
});

// => 1
// => 2
// => 3
// => 4
// => 5
// => 6
// => 7
// => 8
// => done
```

An Observable represents a **collection** of **async values**

Operators

Methods that perform **calculations** on the **values** emitted by an Observable

Operators

Methods that perform **calculations** on the **values** emitted by an Observable

Collection operations

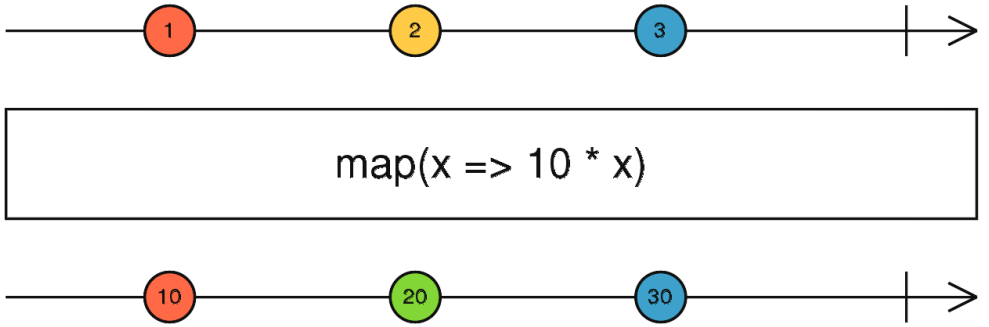
- map
- reduce
- filter
- concat
- merge
- zip
- flatMap
- take
- ...

Async operations

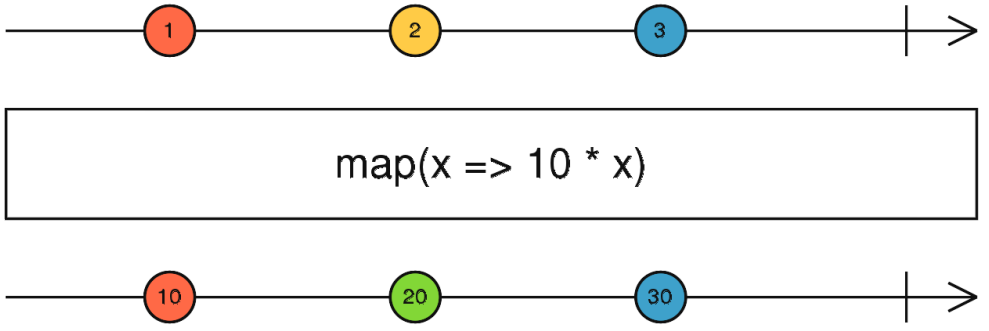
- buffer
- window
- combineLatest
- switchMap
- delay
- throttle
- race
- ...


WHAA

map



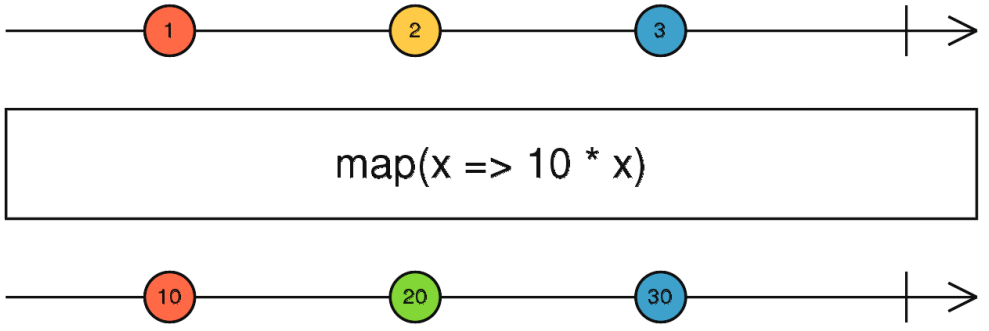
map




```
const numberStream = Rx.Observable.range(1, 3)
    .map(x => x * 10);

numberStream.subscribe(x => console.log(x));
// 10
// 20
// 30
```

map




```
const numberStream = Rx.Observable.range(1, 3)
    .map(x => x * 10);

numberStream.subscribe(x => console.log(x));
// 10
// 20
// 30
```

More examples and diagrams of each operator on reactivex.io/rxjs

Creating an Observable

...that generates a single random number

```
let randomNumber$ = Rx.Observable.create((observer) => {
    const id = setTimeout(() => {
        observer.next(Math.random());
        observer.complete();
    }, 1000);
    return () => clearTimeout(id);
});

randomNumber$.subscribe({
    next(x) { console.log(x); },
    error(e) { console.error(e); },
    complete() { console.log('done'); }
});

// 0.1619301338497363
// done
```

Creating an Observable

...that generates a multiple random numbers

```
let randomNumber$ = Rx.Observable.create((observer) => {
    const id = setInterval(() => {
        observer.next(Math.random());
    }, 1000);
    return () => clearInterval(id);
});

let sub = randomNumber$.subscribe({
    next(x) { console.log(x); },
    error(e) { console.error(e); },
    complete() { console.log('done'); }
});

setTimeout(() => {
    sub.unsubscribe();
}, 2000);

// 0.20407358744031567
// 0.9618393938889027
```

Observables are Lazy

- Execute code upon **subscription** to set up the underlying data stream
- Execute code upon **disposal** to teardown the underlying data stream

Observables are Lazy

- Execute code upon **subscription** to set up the underlying data stream
- Execute code upon **disposal** to teardown the underlying data stream

So you can do things like...

- Setup a WebSocket on subscription
- Close the WebSocket on disposal
- Send an AJAX request
- Abort the AJAX request on disposal
- Setup an event listener
- Remove the event listener on disposal

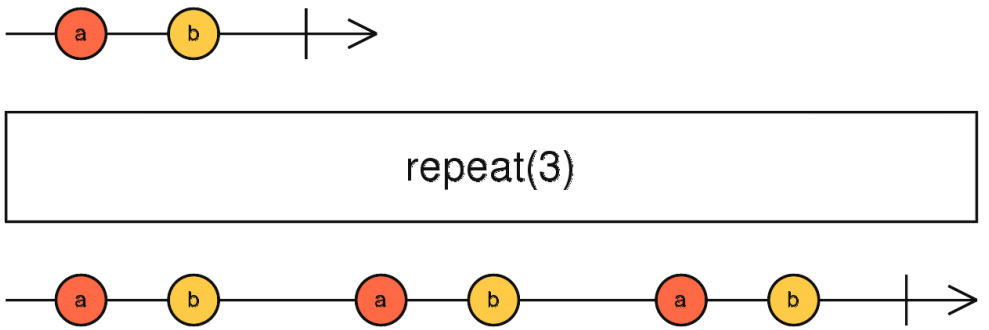
Observables are Lazy

- Execute code upon **subscription** to set up the underlying data stream
- Execute code upon **disposal** to teardown the underlying data stream

So you can do things like...

- Setup a WebSocket on subscription
- Close the WebSocket on disposal
- Send an AJAX request
- Abort the AJAX request on disposal
- Setup an event listener
- Remove the event listener on disposal

Observables can be **repeated** and **retried**




```
const repeated$ = Rx.Observable.range(0, 1)
    .repeat(3);

repeated$.subscribe(value => console.log(value));
// 0
// 1
// 0
// 1
// 0
// 1
```

Code Example!

Application Development

How does your data flow through your app?

How does your data flow through your app?

What events cause variables to be produced

How those variables are used and for what?

Which variables are updated and by what?

How does your data flow through your app?

What events cause variables to be produced

How those variables are used and for what?

Which variables are updated and by what?

RxJS makes this easier to reason about

How does your data flow through your app?

What events cause variables to be produced

How those variables are used and for what?

Which variables are updated and by what?

RxJS makes this easier to reason about

(with practice)

Thinking in Streams

Any variable in your system which its value changes over time

Track what changes its value -- an event? another variable?

Thinking in Streams

Any variable in your system which its value changes over time

Track what changes its value -- an event? another variable?

```
// ...
let c = a + b;
doSomething(c);
// ...
```

Thinking in Streams

Any variable in your system which its value changes over time

Track what changes its value -- an event? another variable?

```
// ...
let c = a + b;
doSomething(c);
// ...
```

```
// ...
const c$ = a$.combineLatest($b, (a, b) => a + b);
c$.subscribe(doSomething);
// ...
```

Thinking in Streams

Any variable in your system which its value changes over time

Track what changes its value -- an event? another variable?

```
// ...
let c = a + b;
doSomething(c);
// ...
```

```
// ...
const c$ = a$.combineLatest($b, (a, b) => a + b);
c$.subscribe(doSomething);
// ...
```

Identify your dependencies → go backwards creating streams

What Operators Should I Use?

1. Use the operator guide at reactivex.io/rxjs

2. Remember you don't have to Rx everything

3. Use the operators you know and do the rest imperatively

What Operators Should I Use?

1. Use the operator guide at reactivex.io/rxjs

2. Remember you don't have to Rx everything

3. Use the operators you know and do the rest imperatively

Start with these:

map, filter, scan, flatMap, switchMap, combineLatest, concat, do

This talk just scratches the surface...

Hot vs Cold Observables

Error Handling and Propagation

Multicast

Subjects

Schedulers

Custom Operators

References

RxJS and Reactive Programming by Ben Lesh

RxJS 5 Thinking Reactively by Ben Lesh

The introduction to Reactive Programming you've been missing by André Staltz

RxJS 5 official documentation

References

RxJS and Reactive Programming by Ben Lesh

RxJS 5 Thinking Reactively by Ben Lesh

The introduction to Reactive Programming you've been missing by André Staltz

RxJS 5 official documentation

Thanks Ben!

Resources

Your Mouse is a Database

Deprecating the observer pattern

End to End Reactive Programming at Netflix

Functional Reactive Programming with RxJava

Learning Observable By Building Observable

RxJS Evolved

RxJS Version 5

RxJS 5 Thinking Reactively

RxJS + Redux + React = Amazing!

Who to Follow

@headinthebox - Erik Meijer (Rx)

@jhusain - Jafar Husain (RxJS, TC-39)

@mattpodwysocki - Matthew Podwysocki (RxJS)

@benlesh - Ben Lesh (RxJS 5)

@trxcllnt - Paul Taylor (RxJS 5)

@andrestaltz - André Staltz (RxJS 5, xstream, Cycle.js)

@_jayphelps - Jay Phelps (RxJS 5, redux-observable)

@benjchristensen - Ben Christensen (RxJava)

Thanks!