Reactive Programming with RxJS

An API for asynchronous programming with observable streams

Paddy Corr

Senior Software Engineer

Paddy Corr

Senior Software Engineer

Reactive Programming

Reactive Programming

"Reactive programming is programming with asynchronous data streams."

Reactive Programming

- Observable

"Reactive programming is programming with asynchronous data streams."

Reactive Programming

- Observable

"Reactive programming is programming with asynchronous data streams."

A stream may emit 3 things.

Reactive Programming

- Observable

"Reactive programming is programming with asynchronous data streams."

A stream may emit 3 things.

A value.

An error.

A completed signal.

Reactive Programming - Observer

Functions that will react to items or sequences of items the Observable emits.

Reactive Programming - Observer

Functions that will react to items or sequences of items the Observable emits.

```
value => {
  const listItem = document.createElement('li').innerHTML = value;
  document.querySelector('#member-list').append(listItem);
},
```

Reactive Programming - Observer

Functions that will react to items or sequences of items the Observable emits.

```
value => {
  const listItem = document.createElement('li').innerHTML = value;
  document.querySelector('#member-list').append(listItem);
},
```

```
err => {
  const listItem = document.createElement('li').innerHTML = err.message;
  document.querySelector('#member-list').innerHTML = row;
},
```

Reactive Programming - Observer

Functions that will react to items or sequences of items the Observable emits.

```
value => {
  const listItem = document.createElement('li').innerHTML = value;
  document.querySelector('#member-list').append(listItem);
},
```

```
err => {
  const listItem = document.createElement('li').innerHTML = err.message;
  document.querySelector('#member-list').innerHTML = row;
},
```

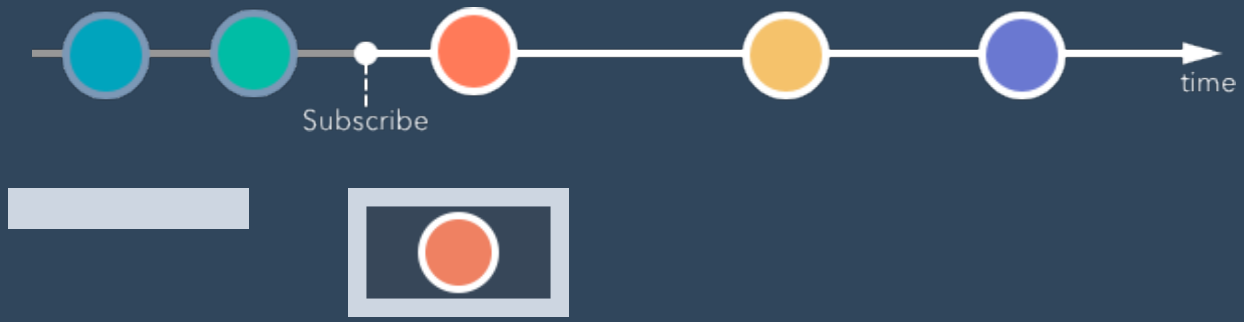
```
function () {
  document.querySelector('#member-list').innerHTML = '';
}
```

Reactive Programming - Subscribing

An Observer subscribes to an Observable.

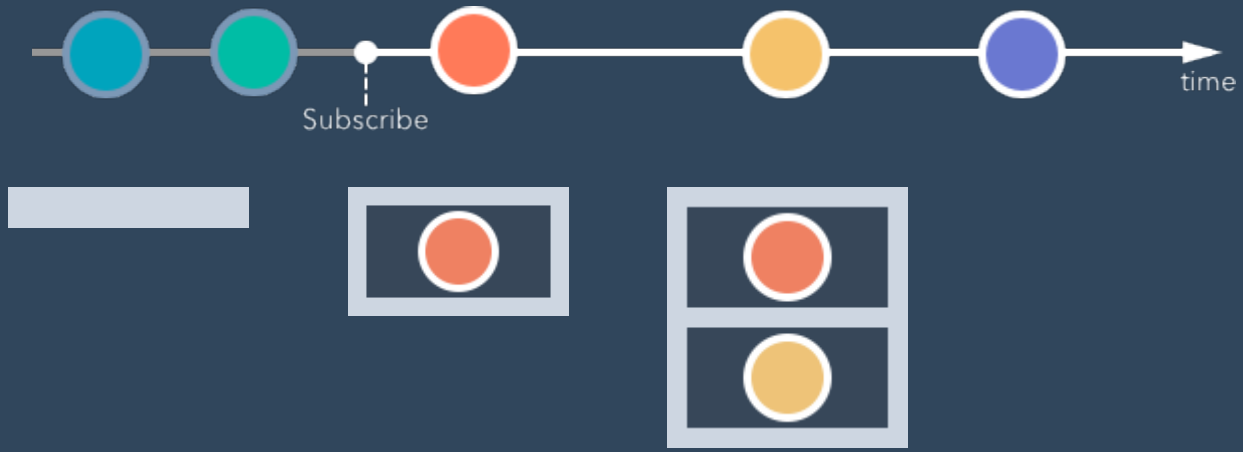
Reactive Programming - Subscribing

An Observer subscribes to an Observable.



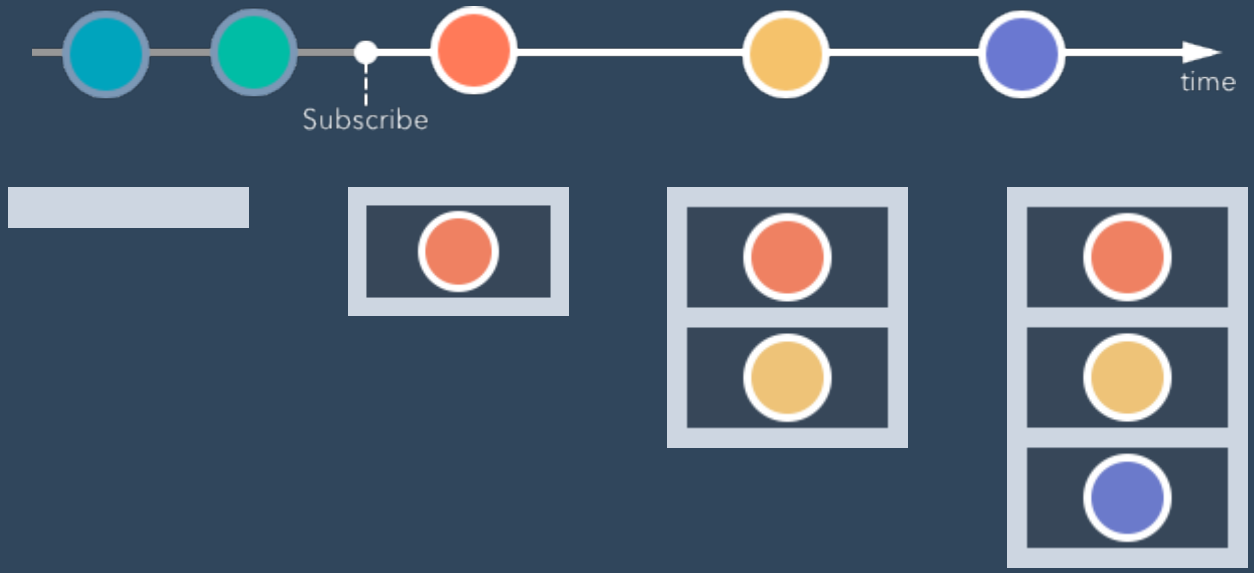
Reactive Programming - Subscribing

An Observer subscribes to an Observable.



Reactive Programming - Subscribing

An Observer subscribes to an Observable.



Observable ◂– Subscribe ◂– Observer

Kind of sounds like

```
button.onclick = function(event) {
  console.log('Click!');
};
```

So why use Reactive Programming

- You are able to create data streams of anything

So why use Reactive Programming

- You are able to create data streams of anything
- Great base of functions that can combine, create and filter these streams.

So why use Reactive Programming

- You are able to create data streams of anything
- Great base of functions that can combine, create and filter these streams.
  - Use a stream as an input to another.
  - Merge two streams and use as input to another.
  - Filter streams to only pluck interesting data.
  - Map values from one stream to another new one.

Think of RxJS as Lodash for events.

RxJS

Where does Observable fit in?

	Single	Multiple
Pull	Function	Iterator
Push	Promise	Observable

Where does Observable fit in?

Event	Iterable (Pull)	Observable (Push)
Retrieve data	next(val)	onNext(val)
Discover Error	throw(err)	onError(err)
Complete	return	onCompleted()

RxJS

```
button.onclick = function(event) {
  console.log(‘Clicked!’);
};
```

```
Rx.Observable.fromEvent(button, 'click')
  .subscribe(
    () => console.log('Clicked!')
  );
```

RxJS

```
let count = 0;

button.onclick = function(event) {
  console.log(‘Clicked!’, count);
};
```

```
Rx.Observable.fromEvent(button, 'click')
  .scan(count => count + 1, 0)
  .subscribe(
    (count) =>
      console.log('Clicked!', count)
  );
```

RxJS

```
let count = 0;
const rate = 1000;
let lastClick = Date.now() - rate;
button.onclick = function(event) {
  if (Date.now() - lastClick >= rate) {
    console.log('Clicked!', count)
    lastClick = Date.now();
  }
});
```

```
Rx.Observable.fromEvent(button, 'click')
  .throttleTime(1000)
  .scan(count => count + 1, 0)
  .subscribe(
    (count) =>
      console.log('Clicked!', count)
  );
```

RxJS

```
let count = 0;
const rate = 1000;
let lastClick = Date.now() - rate;
const handler = function(event) {
  if (Date.now() - lastClick >= rate) {
    console.log('Clicked!', count)
    lastClick = Date.now();
  }
});

button1.onclick = handler;
button2.onclick = handler;
```

```
Rx.Observable.merge(
  Rx.Observable.fromEvent(button1, 'click'),
  Rx.Observable.fromEvent(button2, 'click')
)
  .throttleTime(1000)
  .scan(count => count + 1, 0)
  .subscribe(
    (count) =>
      console.log('Clicked!', count)
  );
```

RxJS - Operators

Pure functions that enable a functional programming style for dealing with collections.

RxJS - Operators

```
Rx.Observable.merge(
  Rx.Observable.fromEvent(button1, 'click'),
  Rx.Observable.fromEvent(button2, 'click')
)
  .throttleTime(1000)
  .scan(count => count + 1, 0)
  .subscribe(
    (count) =>
      console.log('Clicked!', count)
  );
```

Operators

- Creating Observables
- Transforming Observables
- Filtering Observables
- Combining Observables
- Error Handling Operators - (catch)
- Observable Utility Operators - Assorted useful Operators for working with Observables (timeout)
- Conditional and Boolean Operators - (All, SkipUntil)
- Mathematical and Aggregate Operators - (average, reduce)
- Backpressure Operators - Pushing back
- Connectable Observable Operators - (replay)
- Operators to Convert Observables - (To)

Operators - Creating Observables

Create

Defer

From

Interval

Just

Range

Repeat

Start

Timer

Operators - Creating Observables

- Create
- Defer
- From
- Interval
- Just
- Range
- Repeat
- Start
- Timer

```
function f() {
  return Rx.Observable.from(arguments);
}

f(1, 2, 3).subscribe(
  function (x) { console.log('Next: ' + x); },
  function (err) { console.log('Error: ' + err); },
  function () { console.log('Completed'); }
);
```

```
Next: 1
Next: 2
Next: 3
Completed
```

Operators - Creating Observables

- Create
- Defer
- From
- Interval
- Just
- Range
- Repeat
- Start
- Timer

```
function f() {
  return Rx.Observable.just(arguments);
}

f(1, 2, 3).subscribe(
  function (x) { console.log('Next: ' + x); },
  function (err) { console.log('Error: ' + err); },
  function () { console.log('Completed'); }
);
```

```
Next: 1,2,3
Completed
```

Operators - Creating Observables

- Create
- Defer
- From
- Interval
- Just
- Range
- Repeat
- Start
- Timer

```
const promise = new Promise((resolve, reject) => {
  resolve([4, 8, 15, 16, 23, 42]);
});

Rx.Observable.fromPromise(promise)
 .flatMap(Rx.Observable.from)
 .subscribe(
   function (x) { console.log('Next: ' + x); },
   function (e) { console.log('Error: ' + e); },
   function ( ) { console.log('Completed'); }
);
```

```
Next: 4
Next: 8
Next: 15
Next: 16
Next: 23
Next: 42
Completed
```

Operators - Transforming Observables

Buffer

FlatMap

GroupBy

Map

Scan

Window

Operators - Transforming Observables

- Buffer
- FlatMap
- GroupBy
- Map
- Scan
- Window

```
const clicks = Rx.Observable.fromEvent(button, 'click');
const interval = Rx.Observable.interval(1000);
const buffered = interval.buffer(clicks);
buffered.subscribe(x => console.log(x));
```

```
[0, 1, 2, 3]
[]
[4, 5]
...
```

Operators - Filtering Observables

Debounce

Distinct

ElementAt

Filter

First

IgnoreElements

Last

Sample

Skip/SkipLast

Take/TakeLast

Operators - Combining Observables

And/Then/When

CombineLatest

Join

Merge

StartWith

Switch

Zip

Operators - Combining Observables

- And/Then/When
- CombineLatest
- Join
- Merge
- StartWith
- Switch
- Zip

```
Rx.Observable.merge(
  Rx.Observable.interval(150).timeInterval().take(2),
  Rx.Observable.interval(100).timeInterval().take(2),
).subscribe(
    function (x) { console.log('Next: ' + x); },
    function (e) { console.log('Error: ' + e); },
    function ( ) { console.log('Completed'); }
)
```

```
Next: 100
Next: 150
Next: 100
Next: 150
Completed
```

Operators - Error Handling

Catch

Retry

Operators - Error Handling

Catch

Retry

```
const rejectedPromise = () => new Promise((resolve, reject) =>
  reject('Rejected!')
);

example = Rx.Observable.fromPromise(rejectedPromise())
  .catch(error => Rx.Observable.of(`Bad Promise: ${error}`));

example.subscribe(val => console.log(val));
```

```
'Bad Promise: Rejected'
```

Operators - Utility Operators

Delay

Do

Materialize

ObserveOn

Serialize

Subscribe

SubscribeOn

TimeInterval

Timeout

Timestamp

Using

Operators - Conditional and Boolean

All

Amb

Contains

DefaultIfEmpty

SequenceEqual

SkipUntil

SkipWhile

TakeUntil

TakeWhile

Operators - Mathematical and Aggregate

Average

Concat

Count

Max

Min

Reduce

Sum

Operators - Mathematical and Aggregate

- Average
- Concat
- Count
- Max
- Min
- Reduce
- Sum

```
const source = Rx.Observable.range(1, 3)
    .reduce(function (acc, x) {
        return acc * x;
    }, 1)

source.subscribe(
    function (x) { console.log('Next: ' + x); },
    function (err) { console.log('Error: ' + err); },
    function () { console.log('Completed'); }
);
```

```
Next: 6
Completed
```

Cold vs Hot Observable

Cold

Hot

```
const coldObservable = Rx.Observable.from([1,2,3,4,5]);
const coldObservable2 = Rx.Observable.interval(1000);
```

An Observable that will emit the same sequence of items no matter when it is later subscribed to or how frequently those items are observed.

Cold vs Hot Observable

Cold

Hot

```
const hotObservable = Rx.Observable.fromEvent(button1, 'click');
```

A hot Observable begins generating items to emit immediately when it is created. Emits items at its own pace, and it is up to its observers to keep up.

Quick Observable - Slow Observer

Controlled

```
var source = Rx.Observable.range(0, 10).controlled();
source.subscribe(
    (x) => console.log('Next: ' + x)
);

source.request(2);
```

```
Next: 0
Next: 1
```

RxJS with React

Redux Observables

Redux Observable

https://github.com/redux-observable/redux-observable

Netflix JavaScript Talks - RxJS + Redux + React = Amazing!

https://www.youtube.com/watch?v=AslncyG8whg

Thank You

Thank You!

RxJS: http://reactivex.io/rxjs/

An API for asynchronous programming with observable streams

Lodash for events.