<https://rxjs-dev.firebaseapp.com/guide/overview>

# Observable

Observable are like promises for multiple values.

Observables are lazy Push collections of multiple values. They fill the missing spot in the following table:

|  | **Single** | **Multiple** |
| --- | --- | --- |
| **Pull** | [Function](https://developer.mozilla.org/en-US/docs/Glossary/Function) | [Iterator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols) |
| **Push** | [Promise](https://developer.mozilla.org/en-US/docs/Mozilla/JavaScript_code_modules/Promise.jsm/Promise) | [Observable](https://rxjs-dev.firebaseapp.com/class/es6/Observable.js~Observable.html) |

Pull: Consumer decides when to receive the data.

Push: Producer decides when to send the data.

Don’t confuse push with asynchronous.

* Observable are not asynchronous by default. Observables are able to deliver values either synchronously or asynchronously.
* func.call() means "*give me one value synchronously*"
* observable.subscribe() means "*give me any amount of values, either synchronously or asynchronously*"
* If error callback is not found it throws exception

Unlike observer pattern, Observables does not keep track of subscribers. Each .subscribe() call start new execution of Observable.

In library:

 Observable.prototype.subscribe = function (observerOrNext, error, complete) {

If we pass next, error, complete functions it will be converted to Observer () instance. Else observer will be used. In that case error and complete in method param will be ignored.

Subscribe implementation looks something like this

Subscribe(observerOrNext, error, complete) {

toSubscriber(observerOrNext, error, complete);

//logic

}

Function toSubscriber(observerOrNext, error, complete) {

If(observerOrNext instanceof Observer) {

Return observerOrNext;

} else {

Return new Observer(observerOrNext, error, complete)

}

}

<http://reactivex.io/rxjs/class/es6/MiscJSDoc.js~ObserverDoc.html>

## Observable lifecycle

Creating observable: new Observable(function subscriber());

Subscribing/executing to observable: observable.subscribe(nextFunctionOrObserver)

# Operator

There are 2 types

1. Pipeable operator
2. Creation operator

## Pipeable opearator

Operator are pure function.

Operator takes an observable and return another. Let obs2 = op()(obs1);

Here op() – return an operator

Above syntax is similar to obs2 = obs1.pipe(op())

Const obs2 = op1()(op2()(obs1)) is same as obs2 = obs1.pipe(op1(), op2())

## Creation operator

Creates observable.

E.x. obs = of(1,2,3)

# Subject

Observable – Unicast

Subject – multicast (Like observer pattern)

It keeps list of registered subscribers and multicast to all subscribers.

It has next, error, complete; hence can also be used as observer.

## Multicast

Operator to convert unicast Observable to multicast.