I took the challenge to explain RxJS to developers in a simplistic way

RxJS is JavaScript library for transforming, composing and querying asynchronous streams of data. RxJS can be used both in the browser or in the server-side using Node.js.

I took the challenge to explain RxJS to developers in a simplistic way. The hardest part of the learning RxJS is “Thinking in Reactively”.

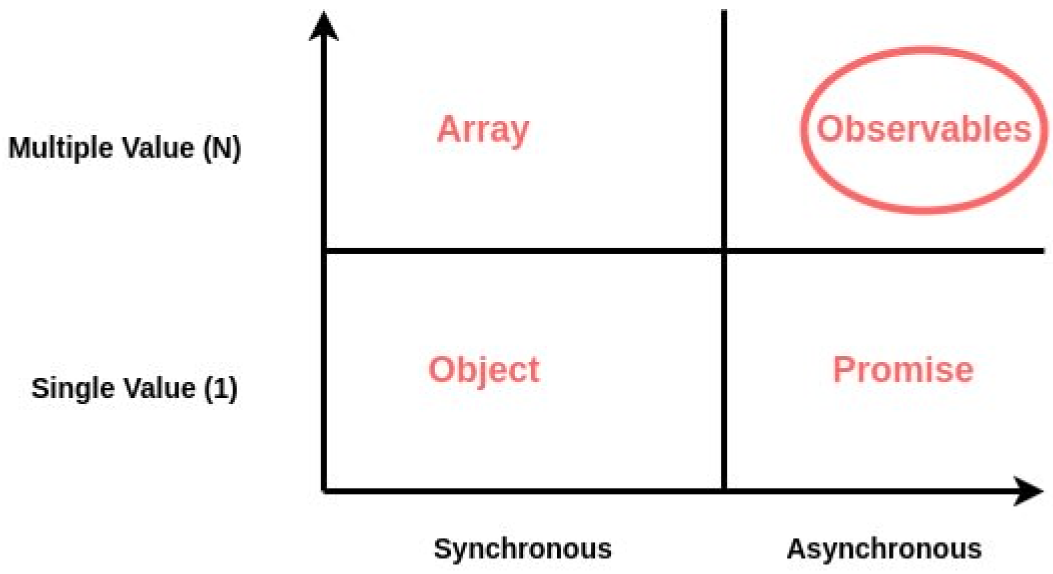
Think of RxJS as “LoDash” for handling asynchronous events.

If you want to learn RxJS, start with one of these classes

- Getting Started with Reactive Programming Using RxJS – Scott Allen
- Getting Started with RxJS – Brice Wilson
- Play by Play: Angular 2/RxJS/HTTP and RESTful Services – John Papa and Dan Wahlin

Reactive Programming in JavaScript with RxJS

Reactive programming is a programming paradigm for writing code, mainly concerned with asynchronous data streams.

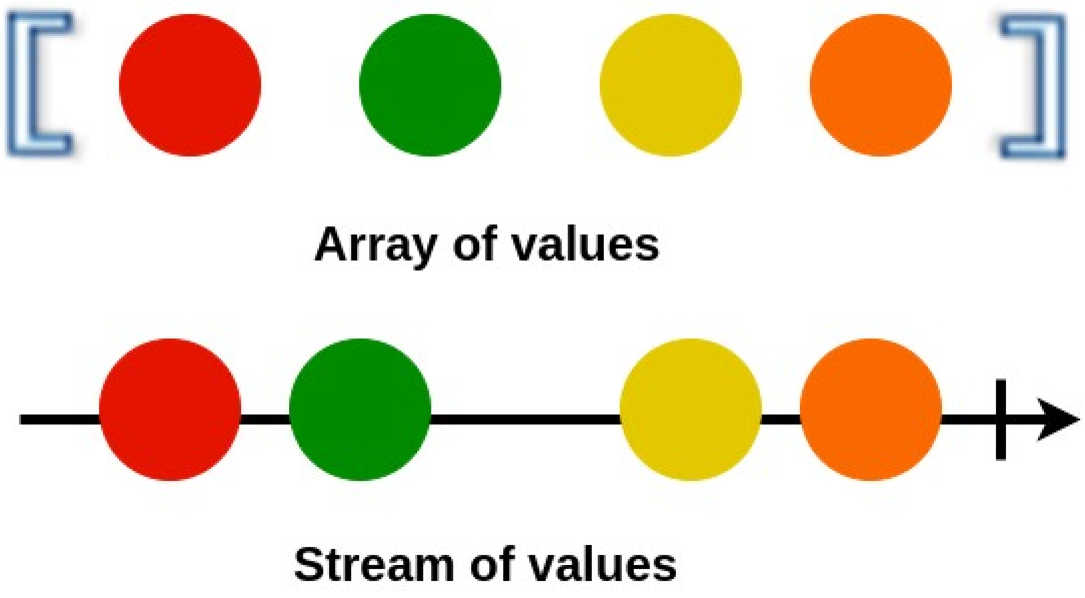


Source: dev.to

Just a different way of building software applications which will “react” to changes that happen instead of the typical way of writing software where we explicitly write code (aka “imperative” programming) to handle those changes.

Stream

A stream is a sequence of ongoing events ordered in time. It can be anything like user inputs, button clicks or data structures. You can listen to a stream and react to it accordingly. You can use functions to combine, filter or map streams.



Source: dev.to

Stream emit three things during its timeline, a value, an error, and complete signal. We have to catch this asynchronous event and execute functions accordingly.

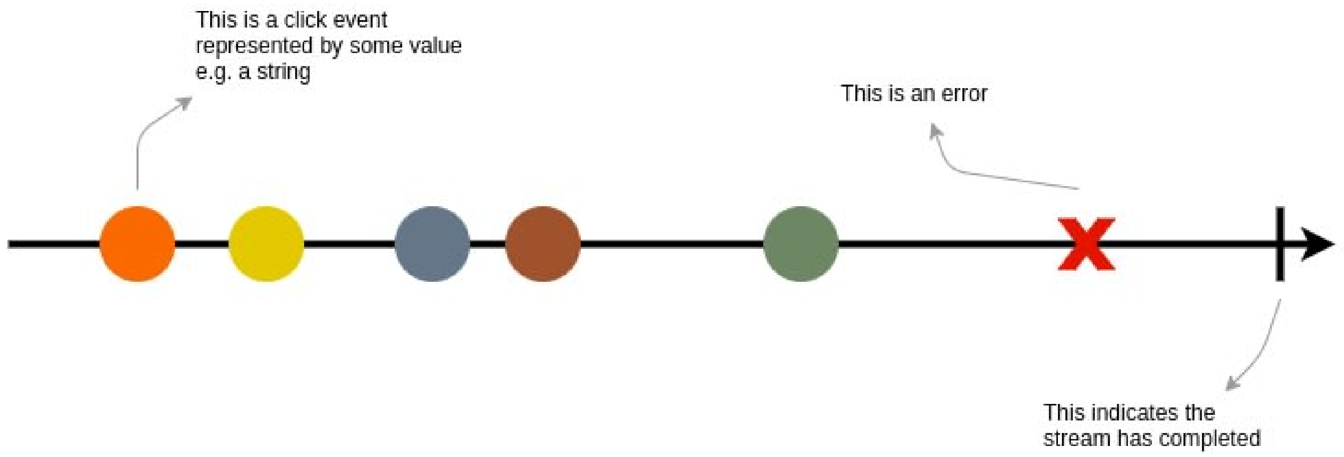
Both promise and observables are built to solve problems around async (to avoid "callback hell").

Types of async operations in modern web applications

- DOM Events- (mouse events, touch events, keyboard events, form events etc)
- Animations – (CSS Transitions and Animations, requestAnimationFrame etc)
- AJAX
- WebSockets
- SSE – Server-Sent Events
- Alternative inputs (voice, joystick etc)

If you're still confused, don't worry, this normally doesn't make much sense at this point. Let’s dive in step by step.

Observable

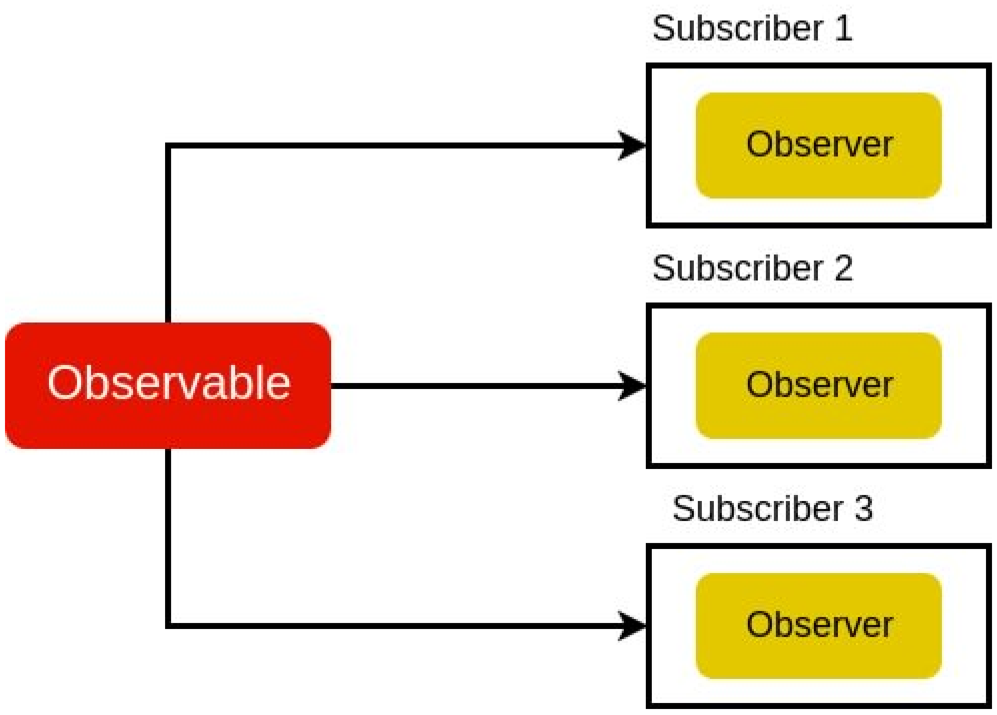

This is a click event represented by some value e.g. a string
This is an error
This indicates the stream has completed

Source: dev.to

An Observable is just a function, with a few special characteristics. It takes in an “observer” (an object with “next”, “error” and “complete” methods on it), and returns cancellation logic.
Observables provide support for passing messages between publishers and subscribers in your application.
Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values.
Observables are lazy. It doesn’t start producing data until you subscribe to it.
`subscribe()` returns a subscription, on which a consumer can be called`unsubscribe()` to cancel the subscription and tear down the producer.
RxJS offers a number of functions that can be used to create new observables. These functions can simplify the process of creating observables from things such as events, timers, promises, and so on.

For Example:

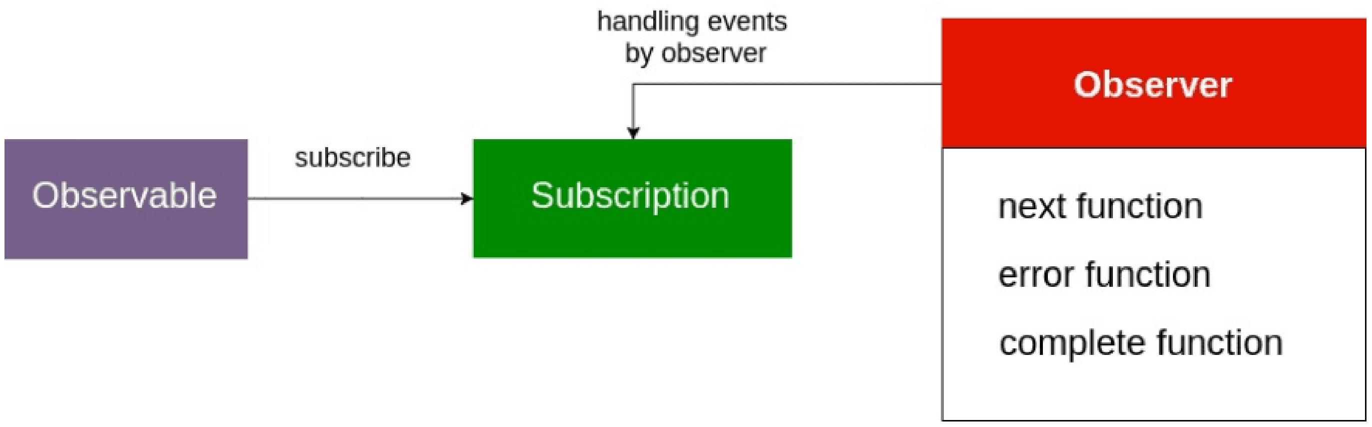
Subscription



Source: dev.to

An Observable instance begins publishing values only when someone subscribes to it. You subscribe by calling a `subscribe()` method the instance, passing an object `observer` for receiving the notifications. A Subscription has one important method, `unsubscribe()`that takes no argument and just disposes of the resource held by the subscription.

Observer

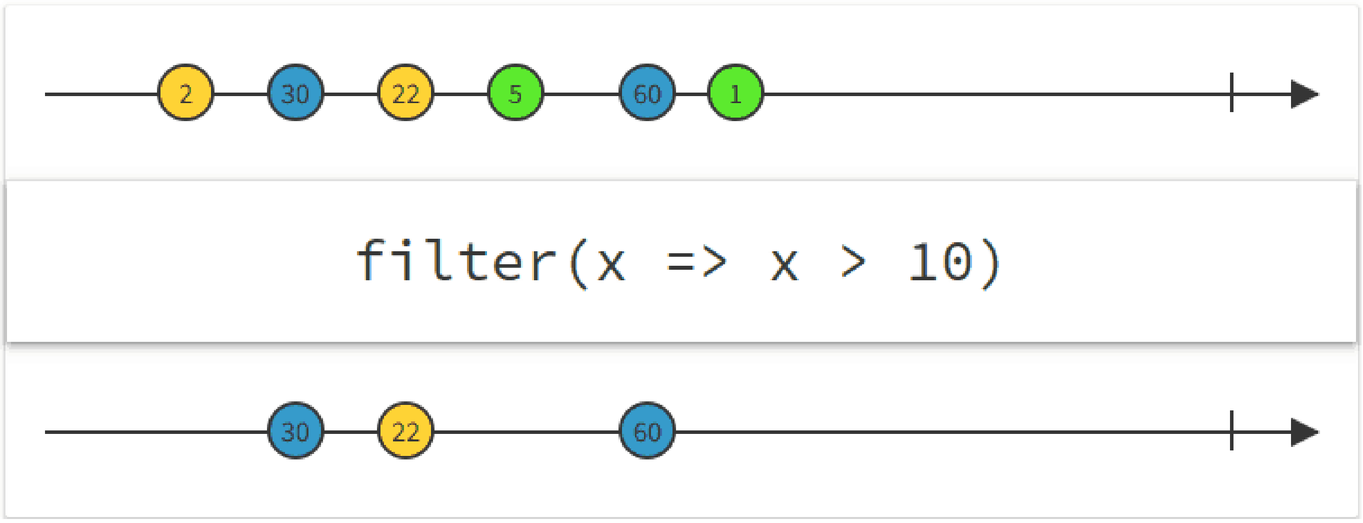


Source: dev.to

An `observer` is object literal with `next()`, `error()` and `complete()`functions. In the above example, the observer is the object literal we pass into our `.subscribe()` method.
When an Observable produces values, it then informs the observer, by calling `.next()` method when a new value was successfully captured and `.error()` when an error occurred.
When we subscribe to an Observable, it will keep passing values to an observer until the complete signal.

Example of an observer:

Operators

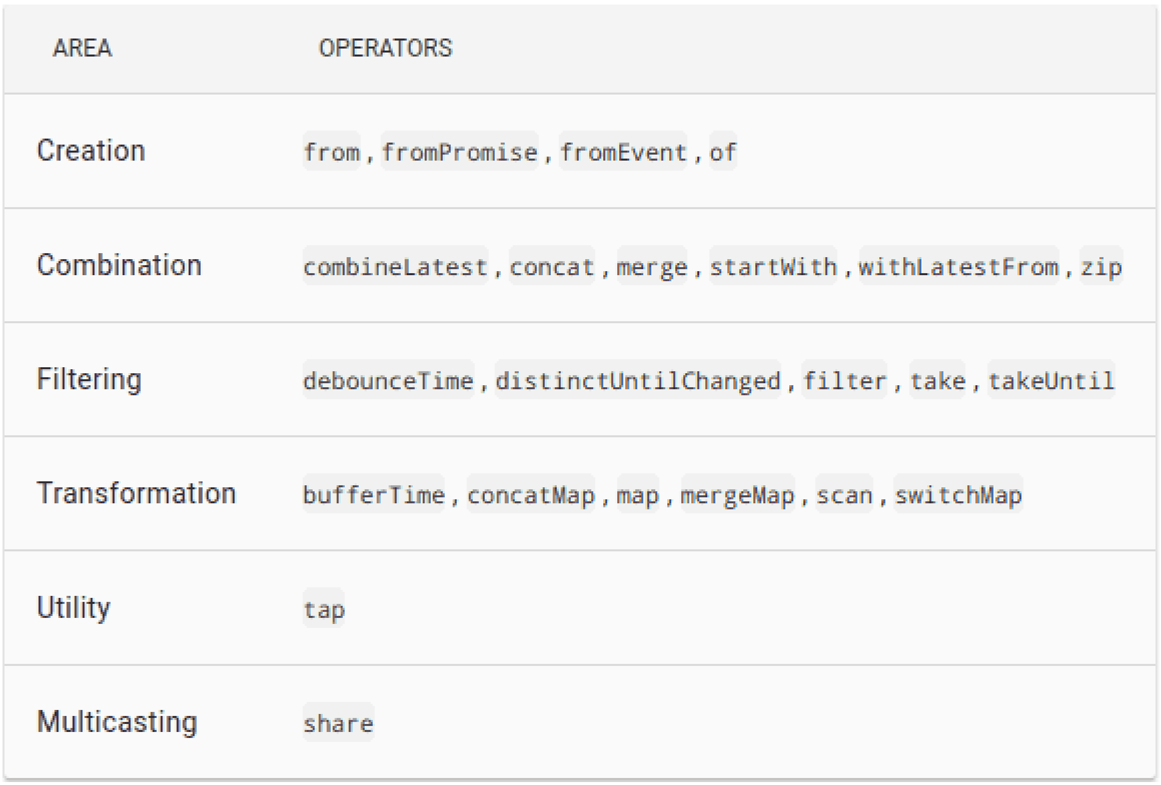


Source: dev.to

- Operators are functions that build on the Observables foundation to enable sophisticated manipulation of collections.
- An Operator is essentially a pure function which takes one Observable as input and generates another Observable as output.
- There are operators for different purposes, and they may be categorized as creation, transformation, filtering, combination, multicasting, error handling, utility etc.
- Operators pass each value from one operator to the next before proceeding to the next value in the set. This is different from array operators (map and filter) which will process the entire array at each step.

For Example:

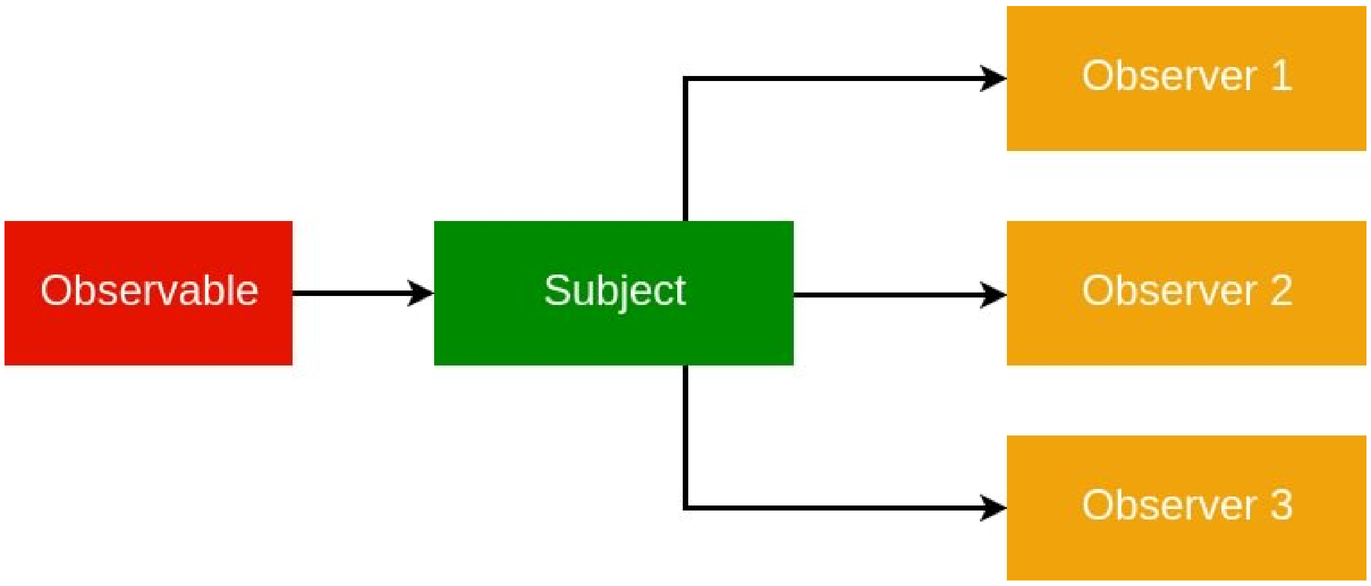
RxJS provides many operators, but only a handful are used frequently. For a list of operators and usage samples, visit the RxJS API Documentation.



AREA	OPERATORS
Creation	`from`, `fromPromise`, `fromEvent`, `of`
Combination	`combineLatest`, `concat`, `merge`, `startWith`, `withLatestFrom`, `zip`
Filtering	`debounceTime`, `distinctUntilChanged`, `filter`, `take`, `takeUntil`
Transformation	`bufferTime`, `concatMap`, `map`, `mergeMap`, `scan`, `switchMap`
Utility	`tap`
Multicasting	`share`

Source: dev.to

Subject



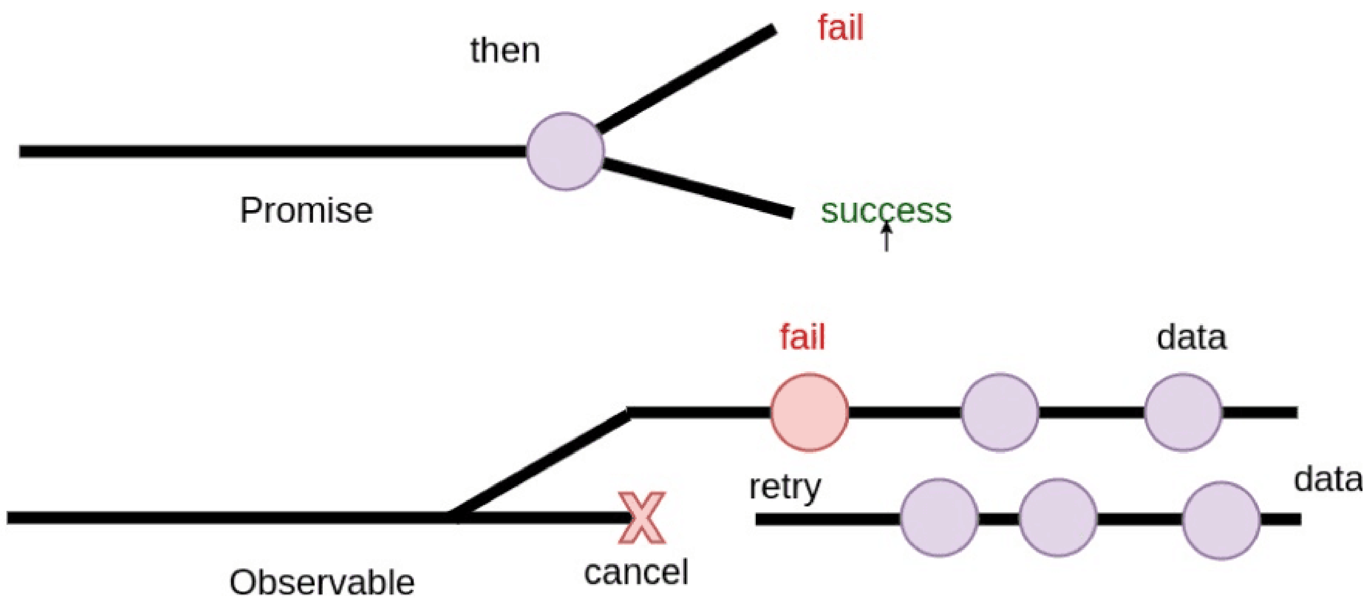
Source: dev.to

RxJS Subject is a special type of Observable that allows values to

be multicasted to many Observers. While plain Observables are unicast (each subscribed Observer owns an independent execution of the Observable), Subjects are multicast.
A subject in RxJS is a special hybrid that can act as both an Observable and an Observer at the same time.

In the example below, we have two Observers attached to a Subject, and we feed some values to the Subject:

Observable vs Promise



Source: dev.to

For better understanding, we’re going to compare and contrast the ES6 Promise API to the Observable library RxJS. We will see how similar Promises and Observables are as well as how they differ and why we would want to use Observables over promises in certain situations.

Single value vs multiple values

If you make a request through the promise and wait for a response. You can be sure that there won’t be multiple responses to the same request. You can create a Promise, which resolves with some value.
The promise is always resolved with the first value passed to the resolve function and ignores further calls to it.
On the contrary, Observables allow you to resolve multiple values until we call `observer.complete()` function.

Example of Promise and Observable:

Eager vs lazy

Promises are eager by design meaning that a promise will start doing whatever task you give it as soon as the promise constructor is invoked.
Observables are lazy. Observable constructor gets called only when someone actually subscribes to an Observable means nothing happens until you subscribe to it.

Examples:

Not cancellable vs cancellable

One of the first things new promise users often wonder about is how to cancel a promise. ES6 promises do not support cancellation yet. It is, the reality of the matter is cancellation is really an important scene in client-side programming.

- Use a third party library like a `bluebird` or `axios` they offer promise cancellation feature.
- Observable support cancellation of the asynchronous task by calling `unsubscribe()` method on Observable.
- When you subscribe to an Observable, you get back a Subscription, which represents the ongoing execution. Just call `unsubscribe()` to cancel the execution.

Example of cancellable observable

Practical Examples

Creating observables from values

Creating Observables from a Stream of values

Observable from DOM Events

Observable from Promise

Observable from Timer method

Observable from Interval

Map operator

Do Operator

Debounce and Throttle

Debounce – Wait X time, then give me the last value.

Throttle – Give me the first value, then wait X time.

bufferTime

Collects values from the past as an array, and emits those arrays periodically in time.

References

RxJS official website

The introduction to Reactive Programming you've been missing

LearnRxJS

What is RxJS?

RxJS Quick Start With 20 Practical Examples

Angular official website

RxJS: Observables, Observers and Operators Introduction

Promises vs Observables

Conclusion

The promise is the best fit for AJAX operations where Observables are extremely powerful for handling asynchronous tasks. Observables provide a bunch of operators for creating, transforming, filtering and multicasting asynchronous events. Sounds great, doesn’t it? ▯

Closing Note

Thanks for reading. I hope you like this article feel free to like, comment or share this article with your friends. For more depth understanding of RxJS checkout provided reference links.

This post was originally published on dev.to by @sagar on 12 Oct 2018 and reposted with permitted edits.