Subjects In Angular

3 Comments / March 9, 2023 / 6 minutes of reading

← Unsubscribe from Angular | Angular Tutorial | Replaysubject, behaviorsubject & Asyncsubject →

In this tutorial. let us learn RxJs Subject in Angular. The Subjects are special observable which acts as both observer & observable. They allow us to emit new values to the observable stream using the `next` method. All the subscribers, who subscribe to the subject will receive the same instance of the subject & hence the same values. We will also show the difference between observable & subject. If you are new to observable then refer to our tutorial on what is observable in angular.

Table of Contents

What are Subjects ?

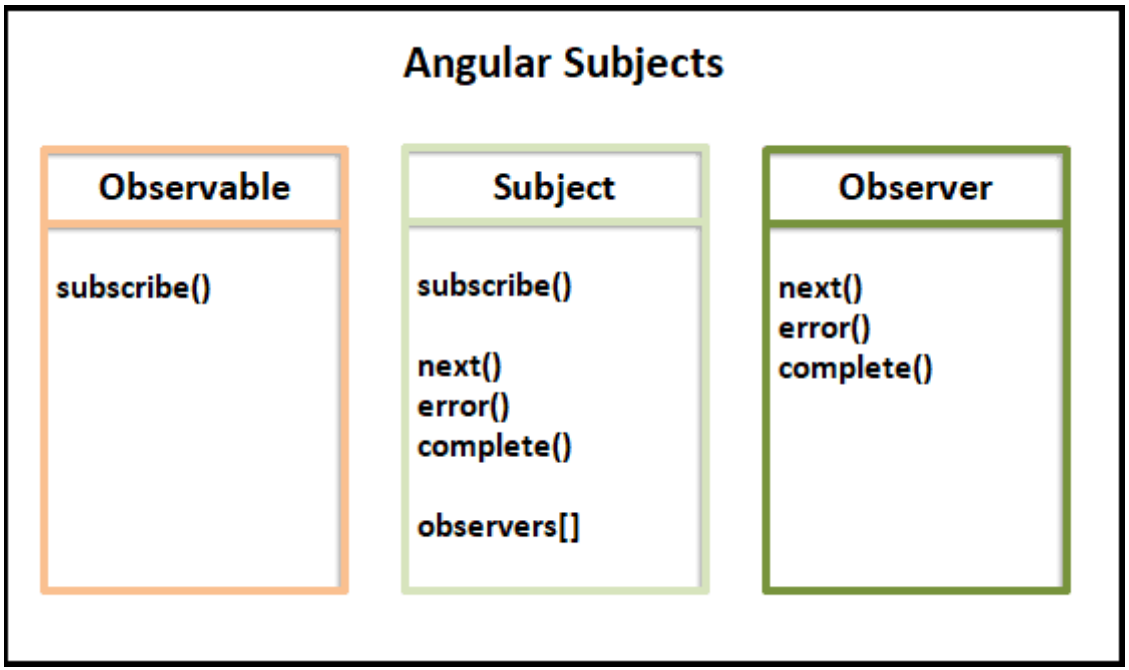
ıbject is a special type of Observable that allows values to be multicasted to many Observers. The subjects are also observers because they can subscribe to another observable and get value from it, which it will multicast to all of its

subscribers.

Basically, a subject can act as both observable & an observer.

How does Subjects work

Subjects implement both `subscribe` method and `next`, `error` & `complete` methods. It also maintains a collection of `observers[]`



An Observer can subscribe to the Subject and receive value from it. Subject adds them to its collection observers. Whenever t' re is a value in the stream it notifies all of its Observers.

The Subject also implements the `next`, `error` & `complete` methods. Hence it can subscribe to another observable and receive values from it.

Creating a Subject in Angular

The following code shows how to create a subject in Angular.

app.component.ts

```
1
2  import { Component, VERSION } from "@angular/core";
3  import { Subject } from "rxjs";
4
5  @Component({
6    selector: "my-app",
7    templateUrl: "./app.component.html",
8    styleUrls: ["./app.component.css"]
9  })
10 export class AppComponent {
11
12   subject$ = new Subject();
13
14   ngOnInit() {
15
16     this.subject$.subscribe(val => {
17       console.log(val);
18     });
19
20     this.subject$.next("1");
       this.subject$.next("2");
22     this.subject$.complete();
```

```
  }
}

```

Stackblitz

The code that creates a subject.

```

subject$ = new Subject();

```

We subscribe to it just like any other observable.

```

this.subject$.subscribe(val => {
  console.log(val);
});

```

The subjects can emit values. You can use the `next` method to emit the value to its subscribers. Call the `complete` & `error` method to raise complete & error notifications.

```

this.subject$.next("1");
this.subject$.next("2");
this.subject$.complete();

```

```
5 //this.subject$.error("error")
6
```

That’s it.

Subject is an Observable

The subject is observable. It must have the ability to emit a stream of values

The previous example shows, we can use the `next` method to emit values into the stream.

```
1
2 this.subject$.next("1");
3 this.subject$.next("2");
4
```

Subject is hot Observable

Observables are classified into two groups. Cold & Hot

Cold observable

- cold observable does not activate the producer until there is a subscriber. This is usually the case when the observable itself produces the data.

```
1
2  import { Component, VERSION } from "@angular/core";
3  import { Subject, of } from "rxjs";
4
5  import { Component, VERSION } from "@angular/core";
6  import { Observable, of } from "rxjs";
7
8  @Component({
9    selector: "my-app",
10   templateUrl: "./app.component.html",
11   styleUrls: ["./app.component.css"]
12 })
13 export class AppComponent {
14   obs1$ = of(1, 2, 3, 4);
15
16   obs$ = new Observable(observer => {
17     console.log("Observable starts");
18     observer.next("1");
19     observer.next("2");
20     observer.next("3");
21     observer.next("4");
22     observer.next("5");
23   });
24
25   ngOnInit() {
26     this.obs$.subscribe(val => {
27       console.log(val);
28     });
29   }
30 }
31
32
```

kblitz

The Producer is one that produces the data. In the above example above it is part of the observable itself. We cannot use that to emit data ourselves.

Even in the following code, the producer is part of the observable.

```
1
2 obs1$ = of(1, 2, 3, 4);
3
```

The producer produces the value only when a subscriber subscribes to it.

t observable

The hot observable does not wait for a subscriber to emit the data. It can start emitting the values right away. The happens when the producer is outside the observable.

Consider the following example. We have created an observable using `subject`. In the ngOnInit, we emit the values & close the Subject. That is without any subscribers.

Since there were no subscribers, no one receives the data. But that did not stop our subject from emitting data.

```
1
2  import { Component, VERSION } from "@angular/core";
3  import { Subject } from "rxjs";
4
5  @Component({
6    selector: "my-app",
7    templateUrl: "./app.component.html",
8    styleUrls: ["./app.component.css"]
9  })
10 export class AppComponent {
11
12   subject$ = new Subject();
13
14   ngOnInit() {
15     this.subject$.next("1");
16     this.subject$.next("2");
17     this.subject$.complete();
18   }
19 }
```

Stackblitz

Now, consider the following example. Here the subjects the values 1 & 2 are lost because the subscription happens after they emit values.

```
1
2  import { Component, VERSION } from "@angular/core";
3  import { Subject } from "rxjs";
4
5  @Component({
6    selector: "my-app",
7    templateUrl: "./app.component.html",
8    styleUrls: ["./app.component.css"]
9  })
10 export class AppComponent {
11   subject$ = new Subject();
12
13   ngOnInit() {
14     this.subject$.next("1");
15     this.subject$.next("2");
16
17     this.subject$.subscribe(val => {
18       console.log(val);
19     });
20
21     this.subject$.next("3");
22     this.subject$.complete();
23   }
24 }
```

Stackblitz

Every Subject is an Observer

The observer needs to implement the next, error & Complete callbacks (all optional) to become an Observer

```
1
2  import { Component, VERSION } from "@angular/core";
3  import { Observable, Subject } from "rxjs";
4
5  @Component({
6    selector: "my-app",
7    templateUrl: "./app.component.html",
8    styleUrls: ["./app.component.css"]
9  })
10 export class AppComponent {
11   subject$ = new Subject();
12
13   observable = new Observable(observer => {
14     observer.next("first");
15     observer.next("second");
16     observer.error("error");
17   });
18
19   ngOnInit() {
20     this.subject$.subscribe(val => {
21       console.log(val);
22     });
23
     this.observable.subscribe(this.subject$);
25   }
```

```
26 }
27
```

Stackblitz

The following example creates a `Subject` & and an `Observable`

We subscribe to the `Subject` in the `ngOnInit` method.

We also subscribe to the observable, passing the subject. Since the subject implements the next method, it receives the values from the observable and emits them to the subscribers.

The Subject here acts as a proxy between the observable & subscriber.

```
1
2 this.observable.subscribe(this.subject$);
3
```

Subjects are Multicast

Another important distinction between observable & subject is that subjects are multicast.

More than one subscriber can subscribe to a subject. They will share the same instance of the observable. This means that all of them receive the same event when the subject emits it.

Multiple observers of an observable, on the other hand, will receive a separate instance of the observable.

Multicast vs Unicast

The Subjects are multicast.

Consider the following example, where we have an observable and subject defined. The observable generates a random number and emits it.

```
1
2  import { Component, VERSION } from "@angular/core";
3  import { from, Observable, of, Subject } from "rxjs";
4
5  @Component({
6    selector: "my-app",
7    templateUrl: "./app.component.html",
8    styleUrls: ["./app.component.css"]
9  })
10 export class AppComponent {
11   observable$ = new Observable<number>(subscriber => {
12     subscriber.next(Math.floor(Math.random() * 200) + 1);
13   });
14
15   subject$ = new Subject();
16
17   ngOnInit() {
18     this.observable$.subscribe(val => {
         console.log("Obs1 :" + val);
       });
21
```

```
    this.observable$.subscribe(val => {
      console.log("Obs2 :" + val);
    });

    this.subject$.subscribe(val => {
      console.log("Sub1 " + val);
    });
    this.subject$.subscribe(val => {
      console.log("Sub2 " + val);
    });

    this.subject$.next(Math.floor(Math.random() * 200) + 1);
  }
}

```

Stackblitz

We have two subscribers of the observable. Both these subscribers will get different values. This is because observable on subscription creates a separate instance of the producer. Hence each one will get a different random number

```

    this.observable$.subscribe(val => {
      console.log("Obs1 :" + val);
    });

    this.observable$.subscribe(val => {
      console.log("Obs2 :" + val);
    });

```

Next, we create two subscribers to the subject. The subject emits the value using the random number. Here both subscribets gets the same value.

```
1
2   this.subject$.subscribe(val => {
3     console.log("Sub1 " + val);
4   });
5   this.subject$.subscribe(val => {
6     console.log("Sub2 " + val);
7   });
8
9   this.subject$.next(Math.floor(Math.random() * 200) + 1);
10
```

Subjects maintain a list of subscribers

Whenever a subscriber subscribes to a subject, it will add it to an array of subscribers. This way Subject keeps track of its subscribers and emits the event to all of them.

There are other types of subjects

What we have explained to you is a simple subject. But there are few other types of subjects. They are

1. ReplaySubject
2. BehaviorSubject

3. AsyncSubject

We will talk about them in the next tutorial.

← Unsubscribe from Angular | Angular Tutorial | Replaysubject, behaviorsubject & Asyncsubject →

3 thoughts on “Subjects in Angular”

ANONYMOUS

OCTOBER 13, 2022 AT 1:01 PM

a

Reply

R

SEPTEMBER 15, 2022 AT 4:32 PM

how to test the Subject in a component ?

Reply

STEFANO

JULY 3, 2022 AT 2:37 PM

Precise explaining. Good job.

Reply

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Website

Post Comment »

This site uses Akismet to reduce spam. Learn how your comment data is processed.