RxJs Subscription

Before learning about RxJS Subscription, let's see what is RxJS subscribe operator.

What is RxJS Subscribe Operator?

The [RxJS](https://www.javatpoint.com/rxjs) Subscribe operator is used as an adhesive agent or glue that connects an observer to an Observable. An observer must be first subscribed to see the items being emitted by an Observable or to receive an error or completed notifications from the Observable.

We can implement the Subscribe operator by using the following three methods:

**1. onNext() method**

An Observable calls the onNext() method whenever the Observable emits an item. This method takes as a parameter the item emitted by the Observable.

**2. onError() method**

An Observable calls the onError() method to specify that it has failed to generate the expected data or has encountered some other errors. When this method is called, it stops the Observable, and it will not make further calls to onNext or onCompleted. This method takes its parameter to indicate the error's type, which sometimes an object like an Exception or Throwable, other times a simple string, depending on the implementation.

**3. onCompleted() method**

An Observable calls the onCompleted() method when it has to called onNext for the last final time and has not encountered any errors.

Cold Observable vs. Hot Observable

An Observable is known as a **"cold" Observable** if it does not start to emit items until an observer has subscribed to it. On the other hand,

An Observable is known as a **"hot" Observable** if it starts emitting items at any time, and a subscriber starts observing the sequence of emitted items at some point after its commencement, missing out on any items emitted previously to the time of the subscription.

What is RxJS Subscription?

An RxJS Subscription is an object used to represent a disposable resource, usually the execution of an Observable. When we create an observable, we have to subscribe to it to execute the observable.

A Subscription has one important method, called the unsubscribe() method, that takes no argument and is used just to dispose of/ release resources or cancel Observable executions of the resource held by the subscription.

Note: Subscription was called "Disposable" in the previous old versions of RxJS.

**Syntax:**

1. **const** subscription = observable.subscribe(x => console.log(x));

Example of RxJS Subscription

Let's see some examples to understand the concept of RxJS subscription and see how to subscribe to an observable.

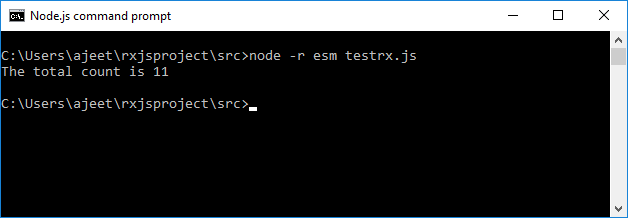
Here, we are using count() operator:

**Example1**

1. **import** { of } from 'rxjs';
2. **import** { count } from 'rxjs/operators';
3. let all\_nums = of(1, 3, 5, 7, 11, 13, 15, 17, 23, 27, 29);
4. let final\_val = all\_nums.pipe(count());
5. final\_val.subscribe(x => console.log("The total count is "+x));

**Output:**

The total count is 11



As we know that the RxJS subscription also has an important method called unsubscribe(). The unsubscribe() method is used to remove all the resources used for that observable i.e.; the observable will get canceled.

Let's see another example using the unsubscribe() method.

**Syntax:**

1. variable\_name.unsubscribe();

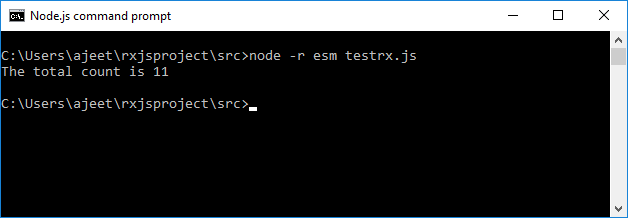
**Example 2**

Here, we are using the same above example with unsunscribe() method.

1. **import** { of } from 'rxjs';
2. **import** { count } from 'rxjs/operators';
3. let all\_nums = of(1, 3, 5, 7, 11, 13, 15, 17, 23, 27, 29);
4. let final\_val = all\_nums.pipe(count());
5. let test = final\_val.subscribe(x => console.log("The total count is "+x));
6. test.unsubscribe();

**Output:**

The total count is 11



Here, the subscription is stored in the variable named 'test' so we have used the test.unsubscribe() apply unsubscribe() method.

Nested Subscription

We can put together multiple subscriptions in a way that if we call to an unsubscribe() of one Subscription, it may unsubscribe multiple Subscriptions. We can do this by "adding" one subscription into another. See the following example:

**Example 3**

1. **import** { interval } from 'rxjs';
2. **const** observable1 = interval(400);
3. **const** observable2 = interval(300);
4. **const** subscription = observable1.subscribe(x => console.log('This is first: ' + x));
5. **const** childSubscription = observable2.subscribe(x => console.log('This is second: ' + x))subscription.add(childSubscription);
6. setTimeout(() => {
7. // this will unsubscribe BOTH subscription and childSubscription
8. subscription.unsubscribe();
9. }, 1000);

**Output:**

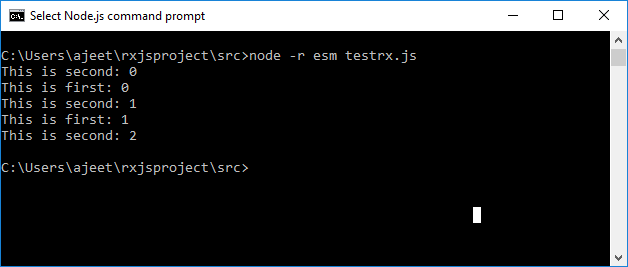
This is second: 0

This is first: 0

This is second: 1

This is first: 1

This is second: 2



Subscriptions also have a remove(otherSubscription) method, which can be used to undo the addition of a child Subscription.