RxJS Observables

In RxJS, an observable is a function that is used to create an observer and attach it to the source where values are expected from. For example, clicks, mouse events from a DOM element or an Http request, etc. are the example of observable.

In other words, you can say that observer is an object with callback functions, which is called when there is interaction to the Observable. For example, the source has interacted for an example, button click, [Http](https://www.javatpoint.com/computer-network-http) request, etc.

Observables can also be defined as lazy Push collections of multiple values. Let's see a simple example to understand how observables are used to push the values.

See the following example:

1. **import** { Observable } from 'rxjs';
2. **const** observable = **new** Observable(subscriber => {
3. subscriber.next(10);
4. subscriber.next(20);
5. subscriber.next(30);
6. setTimeout(() => {
7. subscriber.next(40);
8. subscriber.complete();
9. }, 1000);
10. });

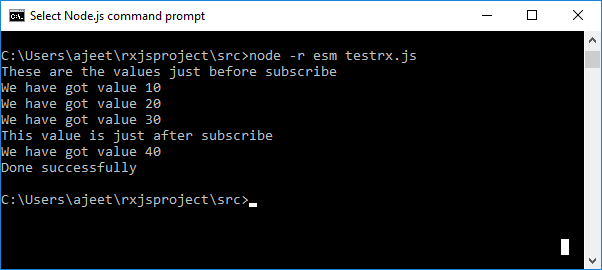
In the above example, there is an observable that pushes the values 10, 20, 30 immediately and synchronously when subscribed, but the value 40 will be pushed after one second since the subscribe method has called.

If you want to invoke the observable and see the above values, you have to subscribe to it. See the following example:

1. **import** { Observable } from 'rxjs';
2. **const** observable = **new** Observable(subscriber => {
3. subscriber.next(10);
4. subscriber.next(20);
5. subscriber.next(30);
6. setTimeout(() => {
7. subscriber.next(40);
8. subscriber.complete();
9. }, 1000);
10. });
12. console.log('These are the values just before subscribe');
13. observable.subscribe({
14. next(x) { console.log('We have got value ' + x); },
15. error(err) { console.error('something wrong occurred: ' + err); },
16. complete() { console.log('Done successfully'); }
17. });
18. console.log('This value is just after subscribe');

**Output:**

When we execute the above program, we shall the following result on the console:



Observables are generalizations of functions

We know that observables are functions that act as clicks, mouse events from a DOM element or an Http request, etc. but observables are not like EventEmitters, nor are they like Promises for multiple values. In some cases, observables may act like EventEmitters, namely when they are multicasted using [RxJS](https://www.javatpoint.com/rxjs) Subjects, but usually, they don't act like EventEmitters.

Observables are like functions with zero arguments, but generalize those to allow multiple values.

Let's see an example to understand this clearly.

**A simple example of a function:**

1. function foo() {
2. console.log('Hello World!');
3. **return** 123;
4. }
5. **const** x = foo.call(); // same as foo()
6. console.log(x);
7. **const** y = foo.call(); // same as foo()
8. console.log(y);

**Output:**

You will see the following output:

"Hello World!"

123

"Hello World!"

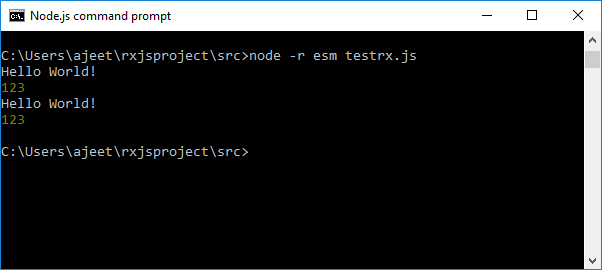
123

**Let's write the same example, but with Observables:**

1. **import** { Observable } from 'rxjs';
2. **const** foo = **new** Observable(subscriber => {
3. console.log('Hello World!');
4. subscriber.next(123);
5. });
6. foo.subscribe(x => {
7. console.log(x);
8. });
9. foo.subscribe(y => {
10. console.log(y);
11. });

**Output:**

You will see the same output as above:



You can see this because both functions and Observables are lazy computations. If you don't call the function, the console.log('Hello World!') won't happen. Also, with Observables, if you don't "call" it with subscribe, the console.log('Hello World!') won't happen.

Working of an Observable

There are three phases in an observable:

* Creating Observables
* Subscribing to Observables
* Executing Observables

Creating Observables

There are two ways to create observables:

* Using the Observable constructor method
* Using Observable create() method

**Using the Observable constructor method**

Let's create an observable using the observable constructor method and add a message, "This is my first Observable" using subscriber.next method available inside Observable.

**testrx.js file:**

1. **import** { Observable } from 'rxjs';
2. var observable = **new** Observable(
3. function subscribe(subscriber) {
4. subscriber.next("This is my first Observable")
5. }
6. );

You can also create Observable using, Observable.create() method as follows:

1. **import** { Observable } from 'rxjs';
2. var observer = Observable.create(
3. function subscribe(subscriber) {
4. subscriber.next("This is my first Observable")
5. }
6. );

Subscribing to Observables

Subscribing to an observable is like calling a function. It provides callbacks where the data will be delivered to.

You can subscribe to an observable by using the following syntax:

**Syntax:**

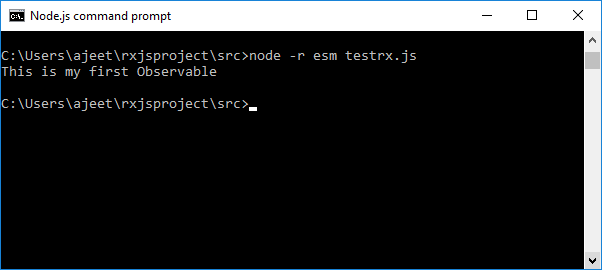
1. observable.subscribe(x => console.log(x));

See the above example with subscribe:

**testrx.js file:**

1. **import** { Observable } from 'rxjs';
2. var observer = **new** Observable(
3. function subscribe(subscriber) {
4. subscriber.next("This is my first Observable")
5. }
6. );
7. observer.subscribe(x => console.log(x));

**Output:**



Executing Observables

An observable is executed when it is subscribed. There are generally three methods in an observer that are notified:

**next():** This method is used to send values like a number, string, object etc.

**complete():** This method doesn't send any value. It indicates that the observable is completed.

**error():** This method is used to notify the error if any.

Let's see an example where we have created the observable with all three notifications and execute that example:

**testrx.js file:**

1. **import** { Observable } from 'rxjs';
2. var observer = **new** Observable(
3. function subscribe(subscriber) {
4. **try** {
5. subscriber.next("This is my first Observable");
6. subscriber.next("Testing Observable");
7. subscriber.complete();
8. } **catch**(e){
9. subscriber.error(e);
10. }
11. }
12. );
13. observer.subscribe(x => console.log(x), (e)=>console.log(e),
14. ()=>console.log("Observable is completed now."));

The error method is invoked only if there is an error. When you run the above code, you will see the following output in the console.

**Output:**

