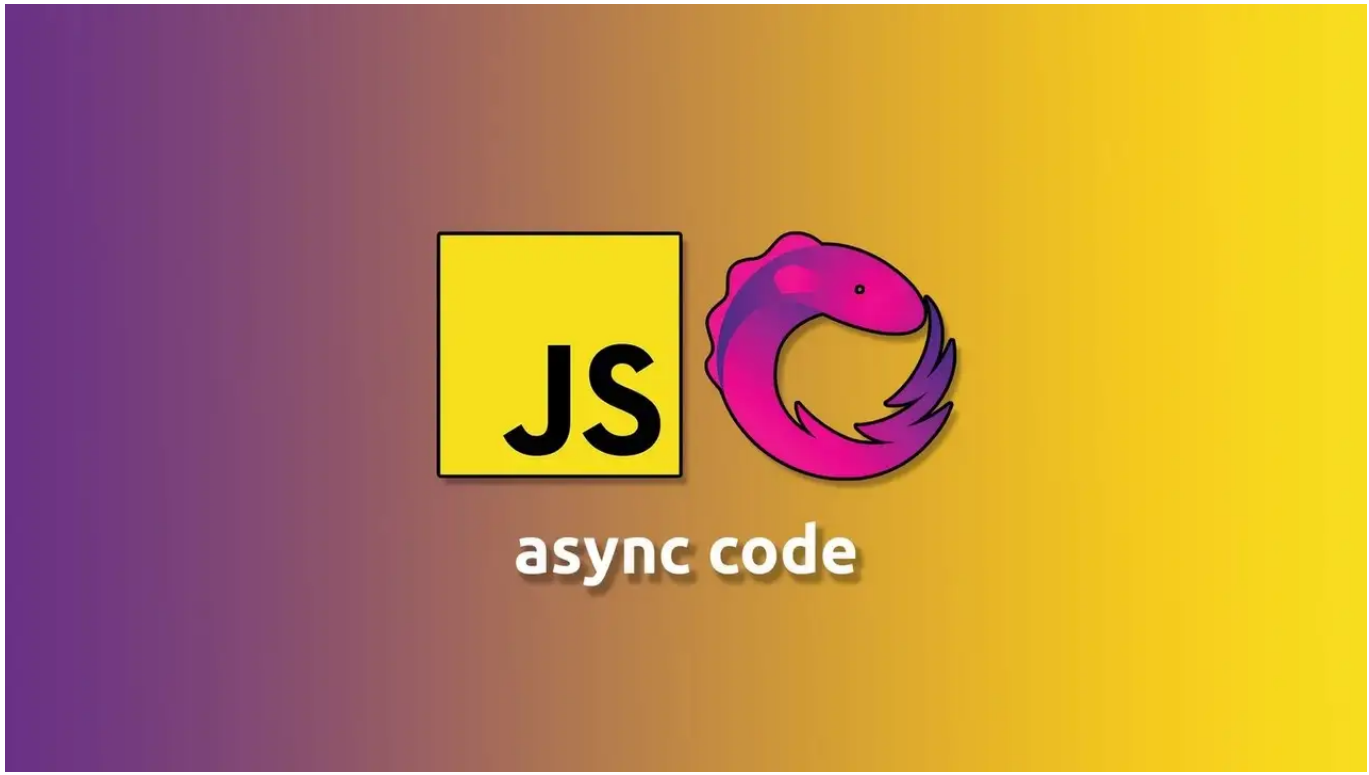
Ruchi Vora Following

Dec 9 · 4 min read · Listen

Javascript: Promise Vs Observable



Promise Vs Observable

In javascript, both promise and observable are used to handle asynchronous requests. While promise and observable have different purposes, it can be confusing when to use them. It is helpful to understand the differences between the two in order to make the best choice.

My purpose in this blog is to compare the observables and promises that will help you to make an informed decision. Understanding javascript promises and observables are essential before going further. You can refer to my previous blogs.....

Javascript: Promises

In order to understand promises in javascript, it is crucial to understand the callback functions and the challenges...

medium.com

Javascript(Rxjs): Observables

Observable is the term which is often discussed when it comes to asynchronous programming. In addition to its...

medium.com

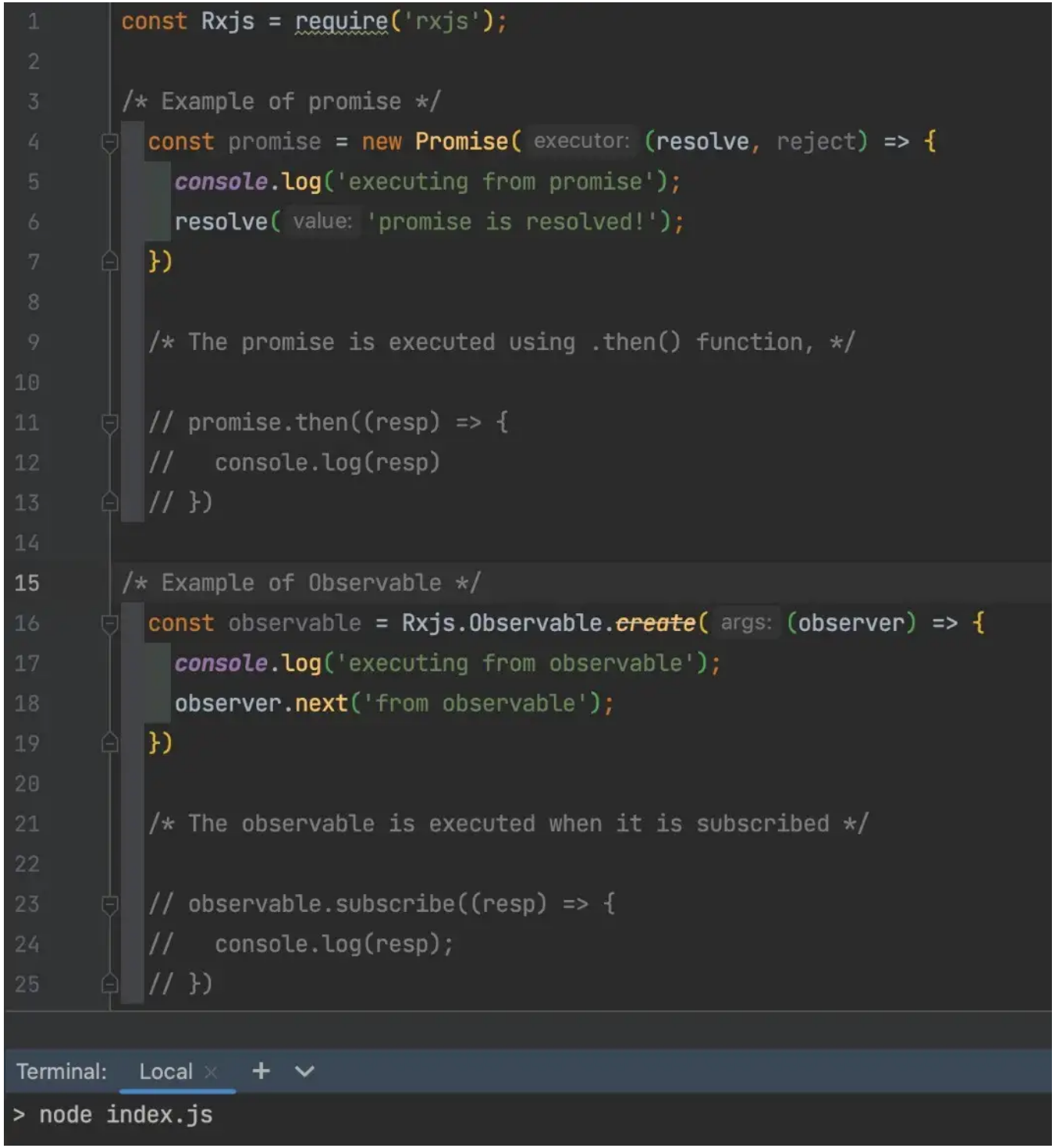
The various differences between promise and observable are:

1. Eager Vs lazy execution

The promises are executed eagerly and observables are executed lazily.

eagerly executed: Promises are executed immediately when the constructor is called.

lazily executed: Observables are executed only when they are subscribed.


const Rxjs = require('rxjs');
/* Example of promise */
const promise = new Promise(executor: (resolve, reject) => {
console.log('executing from promise');
resolve(value: 'promise is resolved!');
})
/* The promise is executed using .then() function, */
// promise.then((resp) => {
// console.log(resp)
// })
/* Example of Observable */
const observable = Rxjs.Observable.create(args: (observer) => {
console.log('executing from observable');
observer.next('from observable');
})
/* The observable is executed when it is subscribed */
// observable.subscribe((resp) => {
// console.log(resp);
// })
Terminal: Local
> node index.js

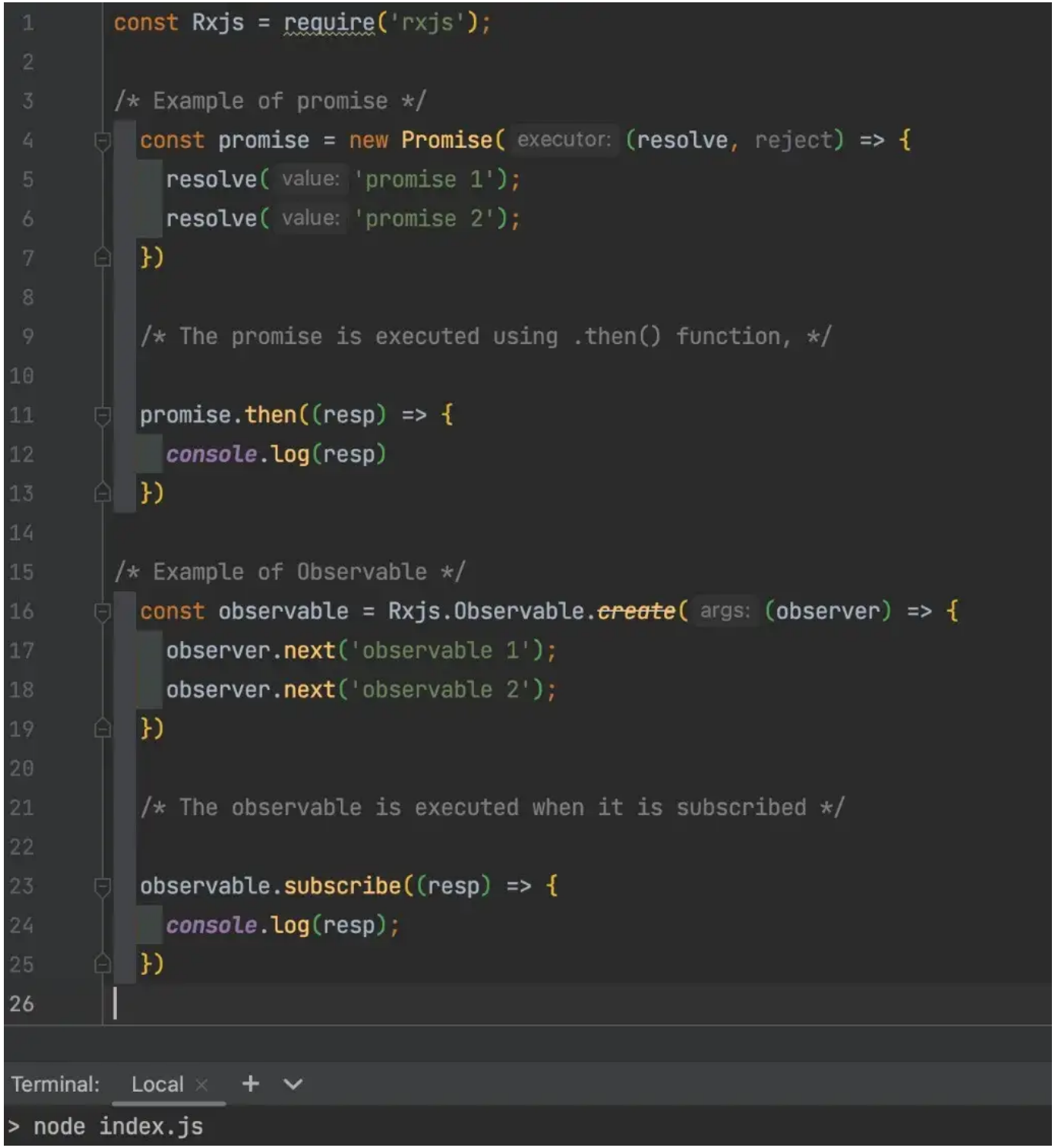
```
executing from promise
```

The code snippet above shows an example of a promise and an observable. So in order for the code inside the promise to execute, you need to call the .then() method. **The code inside the promise is executed even though the .then() method is commented, and therefore the statement `executing from promise` is printed in the console. This is called eager execution.**

On the other hand, for the code inside the observable to execute, you must subscribe to it. **Since the code that subscribes to the observable is commented, the code inside the observable is not executed and the statement `executing from observable` is not printed. This is called lazy execution.**
So, Observables allow us to prevent unnecessary execution of code

2. Single Vs multiple values

Once a promise has been fulfilled, it will emit a single value. In contrast, an observable can emit multiple values, and if subscribed, the subscriber can receive all of the values emitted by the observer.


const Rxjs = require('rxjs');
/* Example of promise */
const promise = new Promise(executor: (resolve, reject) => {
resolve(value: 'promise 1');
resolve(value: 'promise 2');
})
/* The promise is executed using .then() function, */
promise.then((resp) => {
console.log(resp)
})
/* Example of Observable */
const observable = Rxjs.Observable.create(args: (observer) => {
observer.next('observable 1');
observer.next('observable 2');
})
/* The observable is executed when it is subscribed */
observable.subscribe((resp) => {
console.log(resp);
})
Terminal: Local
> node index.js

```
observable 1
observable 2
promise 1
```

Where even though the resolve() is called two times, but **in the output only the first value emitted by resolve() i.e `promise 1` is printed.**
Whereas in the above code snippet, **the observable is emitting two values and both of them are printed** i.e `obervable 1 and observable 2` as the observable is subscribed.
So, observable allow us to emit multiple values

3. Observable subscriptions are cancellable

The promise gets started as soon as the constructor is called and once it is started, it can not be stopped. **Whereas observables can emit multiple values and you can stop listening to those emitted values by unsubscribing them.**

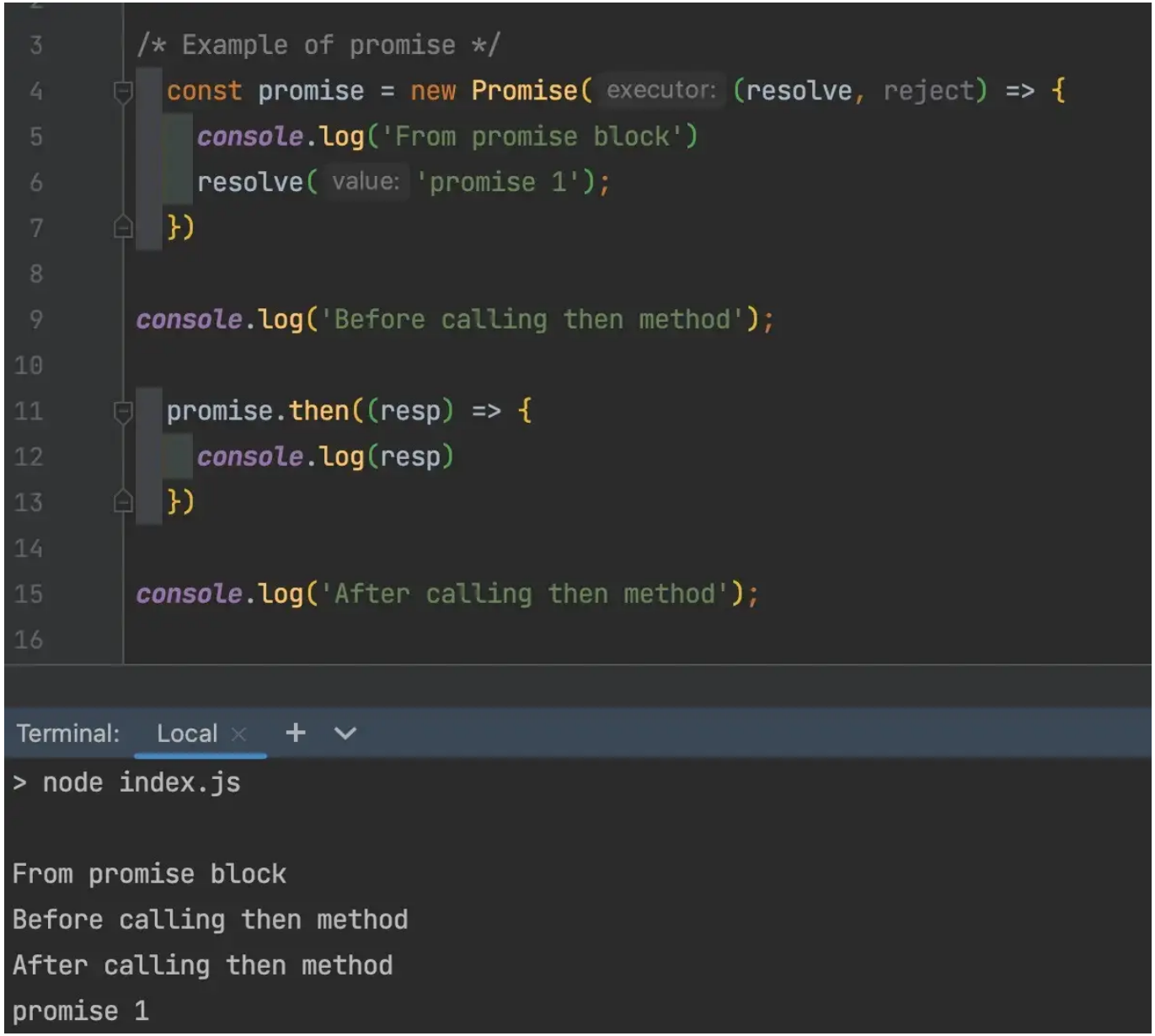
`observer.unsubscribe()` is used to unsubscribe an observable.

4. Asynchronous Vs Synchronous

Promises are always asynchronous. So even if the promise is resolved immediately, the task is put onto the microtask queue and will be executed

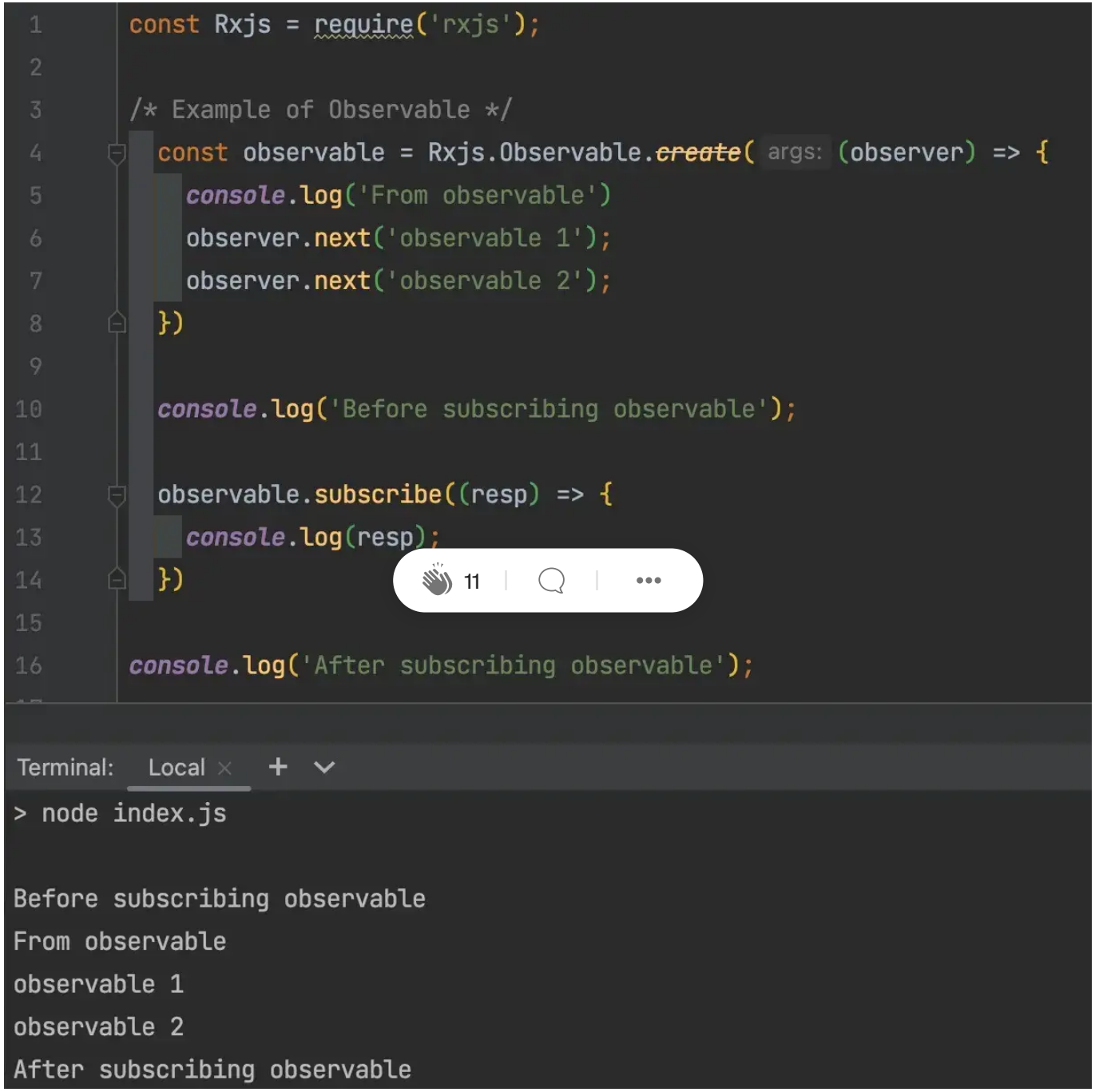
only when the call stack is empty.

Whereas **observables can be synchronous or asynchronous depending on whether it is emitting the values synchronously or asynchronously.**



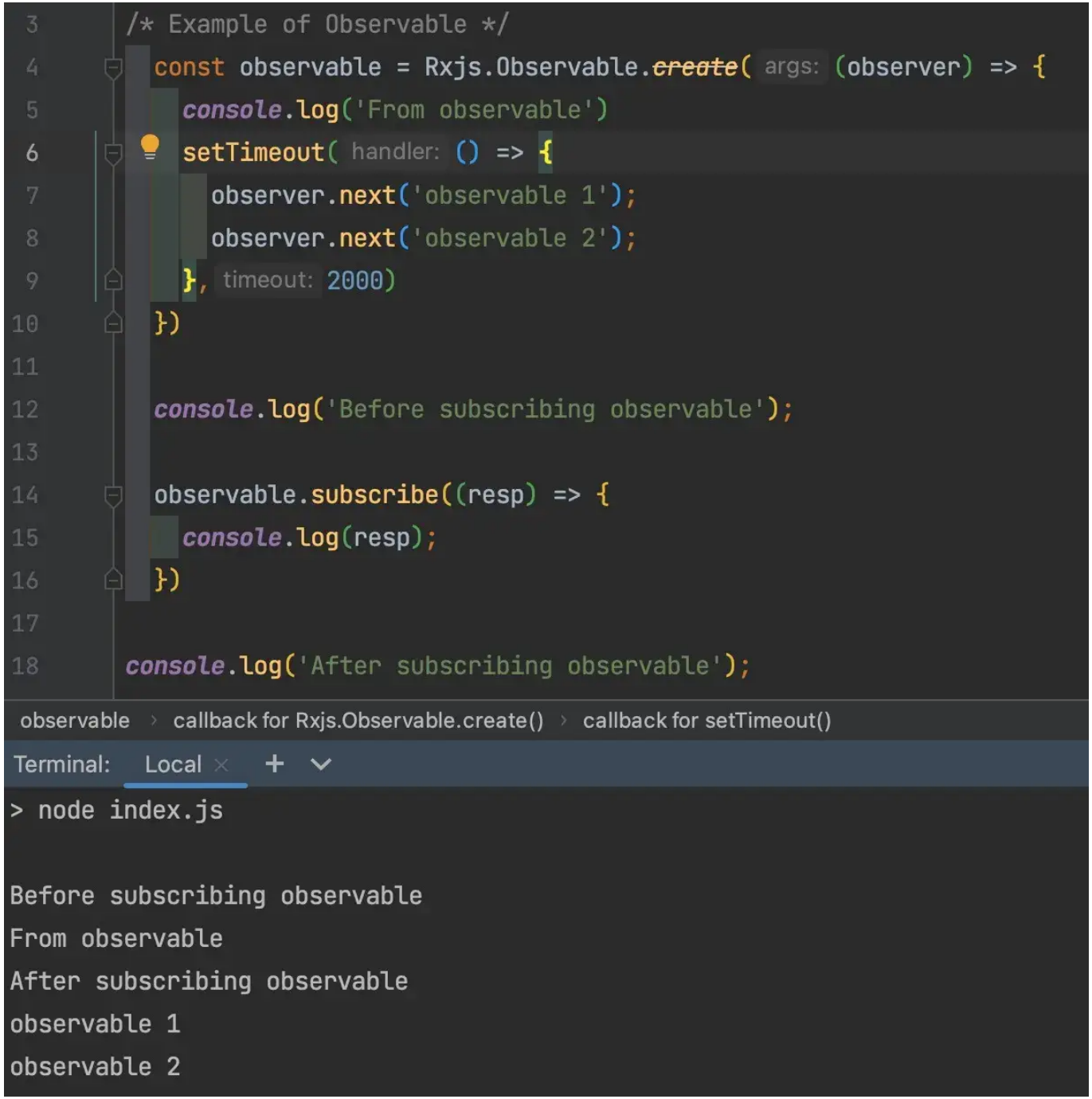
Asynchronous Promise

In the above code snippet, even though the promise is resolved immediately, the resolve('promise 1') is executed at the end when the call stack is empty. The resolve('promise 1') is handled asynchronously even though it is resolved immediately.



Synchronous Observables

The above code snippet consists of synchronous observables. Due to the synchronous execution of code inside the observable, the observable generates synchronous output i.e **all the console statements are printed in sequence.**



Asynchronous Observables

This code snippet contains an asynchronous observable. Due to the asynchronous code within the observable, the output is also asynchronous i.e not sequential. **Below is a detailed explanation of each line of the output.**

1. `Before subscribing observable` — It's a console statement on line no. 12
2. `From observable` — as the observable is subscribed on line no.14 which executes the observable and logs the statement on line no. 5. The next 2 statements i.e .next('observable 1') and .next('observable 2) are pushed to the microtask queue(handled asynchronously).
3. `After subscribing observable` — It's a console statement on line no.18 and as there are no more statements to execute, the call stack is empty
4. `observable 1 , observable 2` — as the call stack is empty, so the statements on the microtask queue are executed.

Also, Observables provide various other benefits, they are:

1. It provides operators to perform different actions on the data stream.
2. It provides functions/operators to convert arrays/strings into the data stream.
3. Using Subject, you can multicast a value or event to multiple observers (most useful in a component-based framework).

After reading this blog, I hope you will be in a better position to choose between promise and observable. To learn more about Rxjs, you can refer to

the documentation https://rxjs.dev/guide/overview.

Feel free to comment down below, if you have any doubts or suggestions.....

Programming Front End Development Java Script Beginner

Reactive Programming