RxJS Working with Scheduler

What is an RxJS Scheduler?

An RxJS Scheduler is a way to control the timing strategy used to execute tasks in RxJS apps or reactive applications. A scheduler comes in action to control when a subscription starts and when notifications are delivered.

According to the official documentation of RxJS application website, a Scheduler is defined as "A Scheduler is an entity that facilitates you to define in what execution context, an Observable will deliver notifications to its Observer."

An [RxJS](https://www.javatpoint.com/rxjs) Scheduler consists of the following three components:

1. **Data structure:** An RXJS Scheduler is a data structure. It knows how to store and queue tasks according to their priority or other criteria.
2. **Execution context:** An RxJS Scheduler is an execution context. It is used to denote where and when the task is executed (For example: immediately, or in another callback mechanism such as setTimeout or process.nextTick, or the animation frame).
3. **Virtual clock:** An RxJS Scheduler consists of a virtual clock that provides a notion of "time" by a getter method now() on the Scheduler. Tasks being scheduled on a particular scheduler will adhere only to the time denoted by that clock.

How to Define RxJS Scheduler?

We know that an RxJS Scheduler controls the execution of the program when the subscription has to start. We can define a basic scheduler in the following way.

**Syntax:**

1. **import** { of, asyncScheduler } from "rxjs";
2. **import** { observeOn } from "rxjs/operators";
3. of(1, 2, 3)
4. .pipe(observeOn(asyncScheduler))
5. .subscribe(val => console.log(val));

Let's see synchronous Observable and asynchronous Observable both and notice the difference by comparing the output of both:

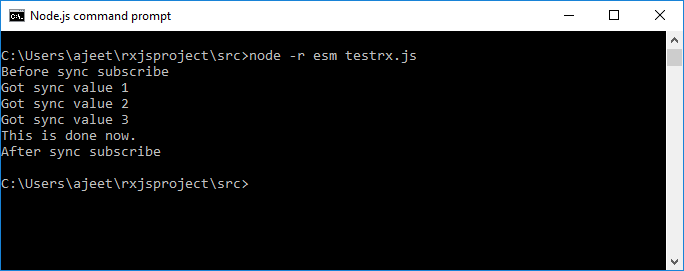
See the synchronous example

**Example 1**

1. **import** { of, asyncScheduler } from "rxjs";
2. **import** { observeOn } from "rxjs/operators";
3. **const** observable = of(1, 2, 3);
4. console.log("Before sync subscribe");
5. observable.subscribe({
6. next(x) {
7. console.log(`Got sync value ${x}`);
8. },
9. error(err) {
10. console.error(`Something wrong occurred: ${err}`);
11. },
12. complete() {
13. console.log("This is done now.");
14. }
15. });
16. console.log("After sync subscribe");

**Output:**

After the execution of the above program, you will see the following result:



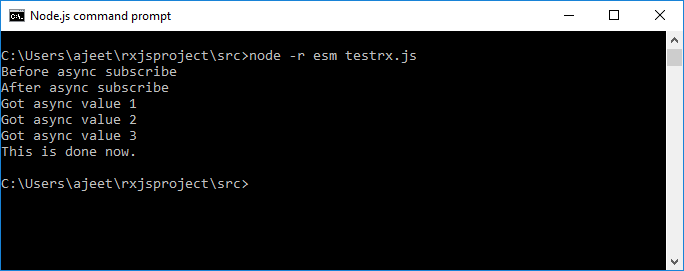
See the asynchronous example (By using Scheduler)

**Example 2**

1. **import** { of, asyncScheduler } from "rxjs";
2. **import** { observeOn } from "rxjs/operators";
3. **const** observable = of(1, 2, 3);
4. console.log("Before async subscribe");
5. observable.pipe(observeOn(asyncScheduler)).subscribe({
6. next(x) {
7. console.log(`Got async value ${x}`);
8. },
9. error(err) {
10. console.error(`Gomething wrong occurred: ${err}`);
11. },
12. complete() {
13. console.log("This is done now.");
14. }
15. });
16. console.log("After async subscribe");

**Output:**

After the execution of the above program, you will see the following result:



**Example explanation**

* You can see in the above examples, the **asyncScheduler** defers the execution of the Observable until after the synchronous code is run.
* The async Scheduler operates with a **setTimeout** or **setInterval**. Even if the delay is zero, it still runs on setTimeout or setInterval. It will ll run in the next event loop iteration.
* The **asyncScheduler** consists of a schedule() method that takes a delay argument, which refers to the quantity of time according to the Scheduler's own internal clock. The internal clock doesn't have anything to do with the actual clock time. Temporal operations are dictated by the Scheduler's clock, not the real clock.
* This can be very useful in testing since we can easily set a time for testing while the tasks are run asynchronously.

Type of RxJS Schedulers

RxJS provides different schedulers that can be created and returned by using static properties of the Scheduler object. Let's see the different Schedulers and also explain the differences between them.

An RxJS Scheduler can be categorized in the following Scheduler types:

|  |  |  |
| --- | --- | --- |
| **Index** | **Scheduler Type** | **Purpose of Scheduler Type** |
| 1. | null | By using the null scheduler type, the notifications are delivered synchronously and recursively. This is useful for constant time or tail-recursive operations. |
| 2. | queueScheduler | The queueScheduler type of Scheduler is used to schedule on a queue in the current event frame. It is very useful for iteration. |
| 3. | asapScheduler | The asapScheduler is a scheduler type used to schedule on the microtask queue, which is the same one used for promises. |
| 4. | asyncScheduler | The asyncScheduler type is used for time-based operations. The works of this scheduler type are scheduled with setInterval. |
| 5. | animationFrameScheduler | The animationFrameScheduler type of Scheduler is used to schedule tasks to run just before the next browser content repaint. It can be used to smooth browser animations. |

Using Schedulers

We have already used schedulers in our RxJS code without explicitly stating the type of schedulers that we used. We don't find the requirement of specifying the type of the Scheduler because all Observable operators that deal with concurrency have optional schedulers. If we do not provide the Scheduler, RxJS will automatically choose a default scheduler using the least concurrency principle.

**For example:**

* If we use operators that return an observable with a finite and small number of messages, RxJS uses no Scheduler that would be null or undefined.
* If we use operators that return a potentially large or infinite number of messages, RxJS will automatically use queue Scheduler.
* If we use operators that use timers, the RxJS will use async.

By default, RxJS uses the least concurrency scheduler. To introduce concurrency for performance purposes, we can pick a different scheduler accordingly. To specify a particular scheduler, we can use those operator methods that take a scheduler, e.g., from([10, 20, 30], asyncScheduler).

Static creation operators usually take a Scheduler as an argument. For instance, from(array, scheduler) lets you specify the Scheduler to use when delivering each notification converted from the array. It is usually the last argument to the operator.

Following is a list of static creation operators that takes a Scheduler argument:

* bindCallback
* bindNodeCallback
* combineLatest
* concat
* empty
* from
* fromPromise
* interval
* merge
* of
* range
* throw
* timer

Some operators also take a time related Scheduler arguments such as bufferTime, debounceTime, delay, auditTime, sampleTime, throttleTime, timeInterval, timeout, timeoutWith, windowTime etc.

Other instance operators like cache, combineLatest, concat, expand, merge, publishReplay, startWith also take a Scheduler argument.

With Schedulers, we can control when data are emitted by Observables. There're different kinds of Schedulers which RxJS will pick automatically according to the principle of creating the least concurrency to do the job. Schedulers run tasks according to their own virtual clock. It has nothing to do with the real world's clock. Many operators facilitate us to set a Scheduler to meet our needs.

In short, we can say that the Schedulers influence the timing on which tasks get executed. We change the default schedulers of some operators by passing in an extra scheduler argument.