

# Operadores Para Tratamento de Erro

Como vimos, um dos três casos que nossos observables podem ter é o caso de erro. Para tratarmos desses erros, o RXJS possui operadores para isso. Vamos ver alguns dos principais operadores para tratamento de erros.

## CatchError

O `CatchError` é um operador que captura o erro e te permite tratá-lo como achar melhor. Mas, ao concluir, você deve retornar um observable qualquer. Vamos ver como pode funcionar. Vamos fazer um observable que retorna números de 1 a 4. Quando for igual a 5, o observable retornará erro e iremos tratar esse erro com o `CatchError`

```
import { catchError, Observable, Observer, of } from 'rxjs'

let obs$: Observable<number> = new Observable((observer: Observer<number>) => {

  let i = 1

  setInterval(() => {
    if (i == 5) {
      observer.error('Houve um erro :(')
    } else {
      observer.next(i++)
    }
  }, 1000)
})

obs$.pipe(
  catchError((error) => {
    return of(error)
  })
).subscribe(
  (data) => {
    console.log(data)
  },
  (error) => {
    console.log(error)
  },
  () => {
    console.log('OK')
  }
)
```

Aqui, quando houver o erro e ele for para o `CatchError`, ele irá retornar um observable do RXJS chamado `of()`. Ele retorna os dados que você passar para ele. Nesse caso, vamos retornar o dado em string que é enviado no método `error`, executado no observable.

Repare que, quando o `catchError` for executado, o observable não vai parar de ser executado, pois quando retornamos um observable no `catchError`, ele faz com que o

subscribe caia na função de sucesso. Ou seja, da maneira que fizemos, ele continuará executando. Mas podemos evitar isso utilizando um operador especial chamado **throwError**. Ele realmente lançará um erro e fará com que o subscribe pare de ser executado.

```
1 import { catchError, Observable, Observer, throwError } from 'rxjs'
2
3 let obs$: Observable<number> = new Observable((observer: Observer<number>) => {
4
5     let i = 1
6
7     setInterval(() => {
8         if (i === 5) {
9             observer.error('Houve um erro :(')
10        } else {
11            observer.next(i++)
12        }
13    }, 1000)
14 })
15
16 obs$.pipe(
17     catchError((error) => {
18         return throwError(error)
19     })
20 ).subscribe(
21     (data) => {
22         console.log(data)
23     },
24     (error) => {
25         console.log(error)
26     },
27     () => {
28         console.log('OK')
29     }
30 )
31
```

Agora sim, esse método cairá na função de erro e deixará de ser executado.

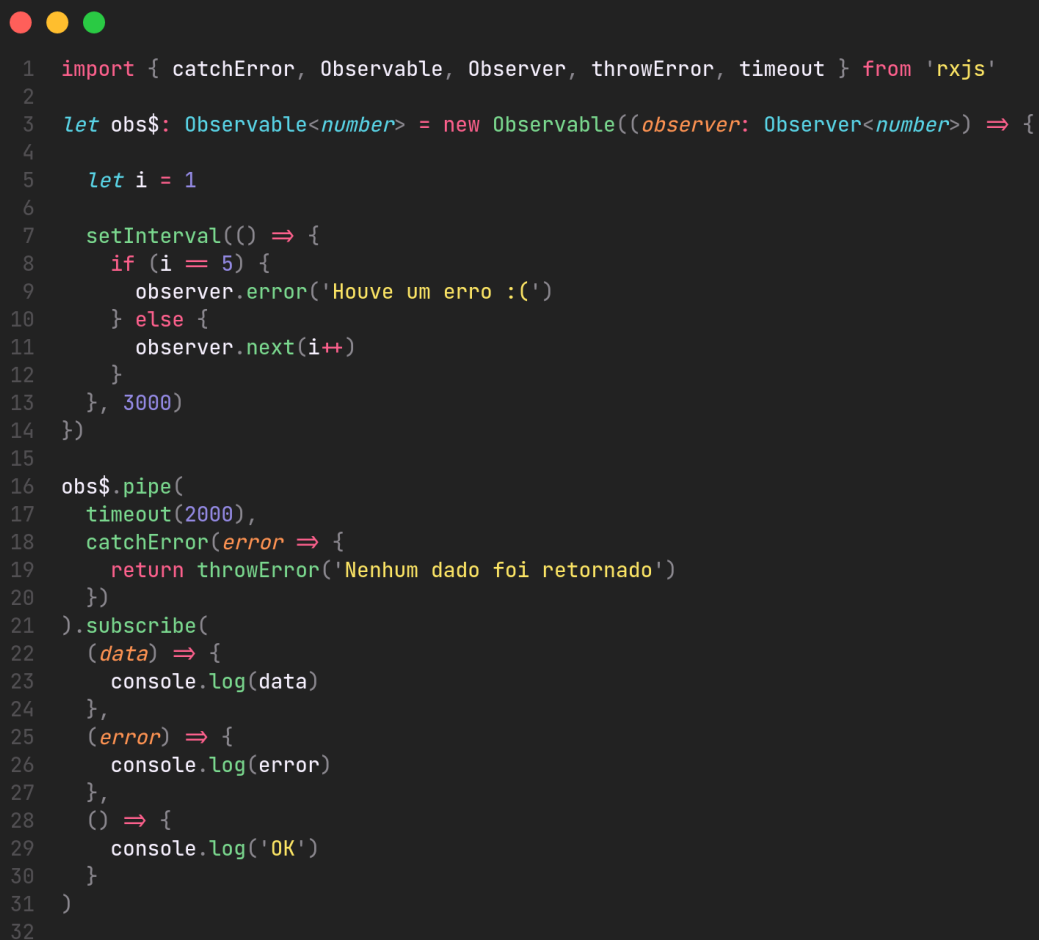
## Retry

O Retry é um operador e vai tentar executar um observable uma quantidade X de vezes que você informar. Ele vai sempre re-executar o observable toda vez. Se ele tentar todas as vezes e não funcionar, ele vai cair na função de erro.

Ele vai tentar re-executar esse observable mais 3 vezes. Se ele não conseguir, ele cairá na função de erro.

## Timeout

O timeout faz com que o subscribe espere um tempo X para que o observable retorne os dados. Se, durante esse tempo, nenhum dado foi retornado, ele lançará um erro. E você precisa tratar esse erro com um catchError, por exemplo.



```

1  import { catchError, Observable, Observer, throwError, timeout } from 'rxjs'
2
3  let obs$: Observable<number> = new Observable((observer: Observer<number>) => {
4
5      let i = 1
6
7      setInterval(() => {
8          if (i == 5) {
9              observer.error('Houve um erro :(')
10         } else {
11             observer.next(i++)
12         }
13     }, 3000)
14 })
15
16 obs$.pipe(
17     timeout(2000),
18     catchError(error => {
19         return throwError('Nenhum dado foi retornado')
20     })
21 ).subscribe(
22     (data) => {
23         console.log(data)
24     },
25     (error) => {
26         console.log(error)
27     },
28     () => {
29         console.log('OK')
30     }
31 )
32

```

O subscribe esperará 2 segundos para que algum dado seja retornado. Se nenhum for retornado, o catchError será acionado e essa mensagem irá aparecer.