

Tipos de Subjects

O Subject possui algumas subclasses que possuem alguns comportamentos diferentes. São elas o **BehaviorSubject**, **ReplaySubject** e o **AsyncSubject**

ReplaySubject

O ReplaySubject, diferentemente do que vimos antes com o Subject normal, ele nos permite acessar dados que já foram enviados antes de darmos o subscribe, ou seja, mesmo que façamos um subscribe “atrasado”, os dados serão retornados a partir do primeiro. Para vermos funcionar, vamos utilizar o exemplo anterior, mas, desta vez, com o ReplaySubject

```
import { Observable, Observer, ReplaySubject } from 'rxjs'

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

  let i = 1

  setInterval(() => {
    if (i == 5) {
      observer.complete()
    } else {
      observer.next(i++)
    }
  }, 1000)
})

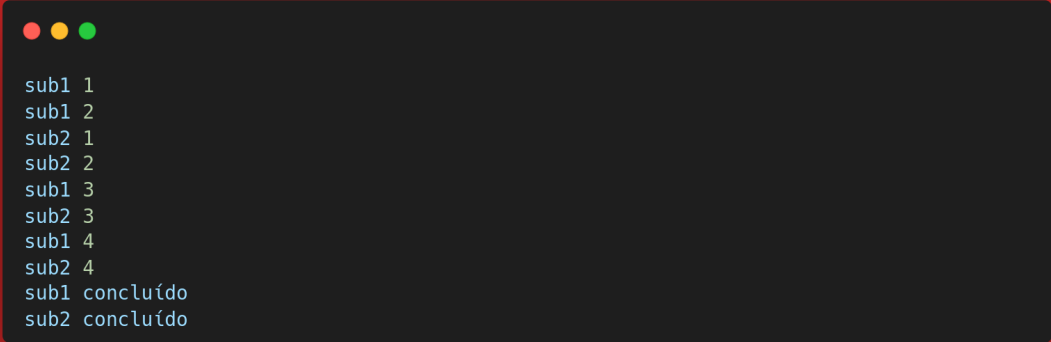
let subject$: ReplaySubject<number> = new ReplaySubject()

obs$.subscribe(subject$)

subject$.subscribe(
  data => {
    console.log('sub1', data)
  },
  error => {
    console.log(error)
  },
  () => {
    console.log('sub1 concluído')
  }
)

setTimeout(() => {
  subject$.subscribe(
    data => {
      console.log('sub2', data)
    },
    error => {
      console.log(error)
    },
    () => {
      console.log('sub2 concluído')
    }
  )
}, 3000)
```

Repare como será a execução desse código




```
sub1 1
sub1 2
sub2 1
sub2 2
sub1 3
sub2 3
sub1 4
sub2 4
sub1 concluído
sub2 concluído
```

Repare que, mesmo com o segundo subscribe tendo sido feito segundos depois, ele ainda conseguiu recuperar os valores antigos, mesmo sendo um Subject, pelo fato de estarmos usando o ReplaySubject dessa vez. Isso é bom para quando você quer trabalhar com uma única fonte de dados, mas sempre vai precisar recuperar os dados informados anteriormente.

AsyncSubject

Este Subject retorna somente o **último** dado enviado pelo subject. Isso é interessante quando você realmente precisa do último dado informado e outros não interessam para você naquele momento, sendo assim, você pode descartá-los e ir atrás somente do último diretamente. Vamos Alterar nosso Subject para o AsyncSubject e ver qual seria o resultado.



```
sub1 4
sub2 4
sub1 concluído
sub2 concluído
```

Repare que todos os outros dados retornados foram ignorados em todos os subscribes e somente o último foi utilizado

BehaviorSubject

O Behavior Subject, diferentemente dos outros, te obriga a passar no construtor um valor inicial. Esse valor inicial será retornado para você enquanto nenhum outro dado da própria fonte de dados foi enviado para você. Isso é eficiente quando você precisa ter uma resposta inicial enquanto você não

```

import { BehaviorSubject, Observable, Observer } from 'rxjs'

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

  let i = 1

  setInterval(() => {
    if (i == 5) {
      observer.complete()
    } else {
      observer.next(i++)
    }
  }, 1000)
})

let subject$: BehaviorSubject<number> = new BehaviorSubject(10)

obs$.subscribe(subject$)

subject$.subscribe(
  data => {
    console.log('sub1', data)
  },
  error => {
    console.log(error)
  },
  () => {
    console.log('sub1 concluído')
  }
)

```

Repare que passei um valor inicial para o BehaviorSubject. Esse dado será retornado enquanto os outros não forem enviados. Reparem bem na execução

```

sub1 10
sub1 1
sub1 2
sub1 3
sub1 4
sub1 concluído

```

Veja que o valor **10**, valor inicial do BehaviorSubject, foi enviado antes dos outros.