

Subjects

Cold Observables VS Hot Observables

Existem dois tipos de observables: os **hot observables** e os **cold observables**. A principal diferença é que um cold observable cria um produtor de dados para cada assinante, enquanto um hot observable cria primeiro um produtor de dados, e cada assinante obtém os dados de um mesmo produtor, a partir do momento da assinatura.

Vamos comparar assistir a um filme na Netflix com ir ao cinema. Pense em você como um observador. Qualquer pessoa que decida assistir a Missão: Impossível na Netflix obterá o filme inteiro, independentemente de quando apertar o botão de reprodução. A Netflix cria um novo produtor para transmitir um filme só para você. Este é um cold observable.

Se você vai ao cinema e a hora do show é às 16h, o produtor é criado às 16h e o streaming começa. Se algumas pessoas (assinantes) chegam atrasadas ao programa, perdem o início do filme e só podem assisti-lo a partir do momento da chegada. Este é um hot observable.

Um cold observable começa a produzir dados quando algum código invoca uma função subscribe nele. Por exemplo, seu aplicativo pode declarar um observable fornecendo uma URL no servidor para obter determinados produtos. A solicitação será feita somente no momento da assinatura. Se outro script fizer a mesma solicitação ao servidor, ele obterá o mesmo conjunto de dados.

Um hot observable produz dados mesmo se nenhum assinante estiver interessado nos dados. Por exemplo, um acelerômetro em seu smartphone produz dados sobre a posição do seu dispositivo, mesmo que nenhum aplicativo assine esses dados. Um servidor pode produzir os preços de estoque mais recentes, mesmo que nenhum usuário esteja interessado neste estoque.

Criando um Cold Observable

Aquele primeiro observable que criamos anteriormente seria um exemplo de cold observable. Mas vamos fazer um que consigamos ver de uma maneira mais fácil essa questão.

Vamos criar um observable que retornará números de 1 a 4 a cada 1 segundo. Quando fizermos isso, vamos dar dois subscribes nesse observable. Um de maneira imediata e outro depois de 2 segundos.

```
import { 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)
})

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

setTimeout(() => {
  obs$.subscribe(
    (data) => {
      console.log('sub2', data)
    },
    (error) => {
      console.log(error)
    },
    () => {
      console.log('sub2 concluído')
    }
  )
}, 2000)
```

Se você executar esse código, você verá este resultado.

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

Note que o sub1 foi executado antes e o sub2 depois dele, mas, mesmo assim, o sub2 recebeu os mesmos dados do sub1. Justamente pelo motivo de, a cada subscribe nesse observable, é criada uma nova fonte de dados para cada subscribe. Assim, esse observable se caracteriza como um **cold observable**.

Criando um Hot Observable

Existem diversas maneiras de se criar um Hot Observable. A melhor delas seria utilizar um **Subject**. O Subject é uma subclasse dos Observables. Eles podem tanto ser um Observable quanto um Observer. Ou seja, ele pode tanto observar quanto ser observado.

Para criar um Subject, é exatamente da mesma forma que criamos um Observable

```
let subject$: Subject<number> = new Subject()
```

Mas, nesse caso, não precisamos passar uma função como parâmetro como fizemos com o Observable. O Subject já é tanto um Observable quanto um Observer. Repare que, a partir dele, podemos acessar os métodos **subscribe**, **next**, **complete**, **error** e vários outros, devido ele possuir os comportamentos tanto de um Observable quanto de um Observer.

Agora, para fazermos um hot observable, vamos fazer um subscribe naquele nosso outro observable e passar o nosso subject como parâmetro.

```
import { Observable, Observer, Subject } 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$: Subject<number> = new Subject()

obs$.subscribe(subject$)
```

Agora, vamos fazer o subscribe a partir do subject e vamos fazer o mesmo esquema de antes: dois subscribes, onde o segundo vai demorar mais tempo para ser feito

```
import { Observable, Observer, Subject } 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$: Subject<number> = new Subject()

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)
```

Vamos ver o resultado da execução dessa aplicação

A terminal window with a dark gray background and a red title bar. The title bar has three colored buttons (red, yellow, green) on the left. The terminal displays a sequence of log messages. The first four lines are 'sub1 1', 'sub1 2', 'sub1 3', and 'sub1 4'. The next two lines are 'sub2 3' and 'sub2 4'. The final two lines are 'sub1 concluído' and 'sub2 concluído', both in a light purple color. The text is in a monospaced font.

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

Agora repare que, mesmo com o segundo subscribe ter acontecido segundos depois do primeiro, ele apresentou os mesmos dados do primeiro e não começou a partir do **1**, pois ele perdeu os outros dados devido ter chegado atrasado. Isso seria o conceito do hot observable atuando. Como ele não deu o subscribe no momento certo, os dados anteriores foram perdidos e apenas os novos foram recuperados.