Observable vs Subject and asObservable

Asked 1 year, 6 months ago Active 1 year, 6 months ago Viewed 4k times

10

5

I am learning RxJs, I am seeking confirmation or correction on my assumption.

I am trying to make a *public read only* observable in a service that I can use `.next()` on in various places in my service class. I am wondering if this is the proper way to do it:

```
private myObservable = new Subject<T>();
public myObservable$: Observable<T> = this.myObservable.asObservable();
```

- The user can subscribe to `myObservable$`
- I can use `myObservable.next(...);`

It works perfectly but I am experience enough to know that I may just be being an unwitting idiot (RxJS is huge). Is this correct pattern and correct object for said use case?

javascript angular typescript rxjs angular-services

edited Feb 22 '18 at 20:59
martin
52.6k 14 111 153

asked Feb 22 '18 at 19:20
JsFlipper
497 1 4 16

2 Yes, this is a common and correct pattern. But in many cases, it is not needed. You only really need a Subject if you need to send notifications to components that are *not* binding. If you are binding, Angular's change detection can take care of handling notifications. I have an example with Subject/BehaviorSubject AND matching code with the same functionality without it here: github.com/DeborahK/MovieHunter-communication (MH-Take4 and MH-Take5) – DeborahK Feb 22 '18 at 22:35

1 @DeborahK Based on some of the comments on some of the replies in this thread I am very curious what your thoughts are on the use of private in TypeScript. Should one not write code using private because it is in fact not private, should one write code in typescript as if they are writing C# and simple ignore the fact that the transpile treats both private and public the same? – JsFlipper Feb 23 '18 at 0:44

2 The entire point of TypeScript is to give us coding assistance/productivity by giving us types, interfaces, and accessibility key words (such as `private`). If we weren't going to use these things ... there would be no point in using TypeScript. The fact that all of this is transpiled out doesn't matter. It all becomes 1's and 0's at some point. :-) – DeborahK Feb 23 '18 at 1:42

1 You are correct. TypeScript provides higher *development time* abstractions ... it does nothing really for runtime. – DeborahK Feb 23 '18 at 19:15

1 @DeborahK the purpose of typescript, as stated by its designers, is to provide a statically verifiable formalization of JavaScript's implicit type system. Tangentially, while programmers using a variety of frameworks, wish to pretend that it describes a Java like type system, this mistake is by far most commonly observable among the Angular community. There's long been unwillingness to learn JavaScript among many developers, and thinking that TypeScript offers an *alternative* to JavaScript is serious wrong. With your background in JS, and role as an educator, you're no doubt aware of this. – Aluan Haddad Feb 23 '18 at 19:36

2 Answers

5

What you're doing is correct. There's however still a little shorter notation. Since Subject is already an Observable (it inherits the Observable class) you can leave the type checking to TypeScript:

```
private myObservable = new Subject<T>();
public myObservable$: Observable<T> = this.myObservable;
```

Any consumer of your service can subscribe to `myObservable$` but won't be able to call `myObservable$.next()` because TypeScript won't let you do that (Observable class doesn't have any `next()` method).

This is actually the recommended way of doing it and RxJS internally never uses `asObservable` anyway. For more detailed discussion see:

- https://github.com/ReactiveX/rxjs/pull/2408
- https://github.com/ReactiveX/rxjs/issues/2391

See a very similar question: Should rxjs subjects be public in the class?

answered Feb 22 '18 at 20:58

martin
52.6k 14 111 153

2 Correct second line is `myObservable$ = myObservable.asObservable();` which doesn't have a `next` method at all and is also of type of `Observable<T>` – Aluan Haddad Feb 22 '18 at 21:08

1 The `Observable<T>` type is enforced by TypeScript. The only way you could avoid it is using `(myObservable$ as any).next()` which you typically never do. Read the two GitHub links why RxJS internally always uses this way. – martin Feb 22 '18 at 21:10

then why do they provide `asObservable` . This is a weak approach. – Aluan Haddad Feb 22 '18 at 21:12

1 All right it's a bit harsh but it's a false sense of security. The thing is that people tend to forget that type assertions are not casts because they look like

casts. Anyway you've got the RX Js people backing you up although one of them disagreed pretty strongly. – Aluan Haddad Feb 22 '18 at 23:09

1 If you are really worried about safety, I would also recommend that you use a getter to receive the `Observable`. Otherwise someone could overwrite it with his own `Subject` and start pushing unwanted data. – Markai Feb 23 '18 at 13:20

3

In project we are using this kind of Observables, this is giving you proper encapsulation to your private observable, but you still can call `next()` using some public method.

```
private sourceName = new Subject<T>();
name = this.sourceProductName.asObservable();

sendName(item: T) {
  this.sourceName.next(item);
}
```

edited Feb 22 '18 at 20:08

answered Feb 22 '18 at 19:56

Kraken
1,089 4 19

1 You used BehaviorSubject, which I do not need in my use case. and added a public method thereby negating the encapsulation of broadcasting. – JsFlipper Feb 22 '18 at 20:03

1 I doesn't matter you can replace it with just `Subject`, i am using this to sometimes retrieve value of this Observable in some period of time with `getValue()` It will fit your needs. – Kraken Feb 22 '18 at 20:08

Sorry i edited and left initial value there, `Subject` don't need that. – Kraken Feb 22 '18 at 20:12

You literally copied my code and then broke it by adding a public method accessor to the private variable to remove the encapsulation. I think you may have misunderstood the question. If you read your code it can be replaced with `name = new Subject<T>();` well, you did protect everything except `next()`, based on that I think you just misread my post. I think what this is coming down to is: my assumption in my OP is was correct. But thank you for taking a poke at it for me. – JsFlipper Feb 22 '18 at 20:26

@EddieMarinaro there is no such thing as privacy via `private` so this is hardly broken. I agree its kind of pointless to simply wrap that method. The point of `asObservable` is to hide the subject but it's a different sort of hiding – Aluan Haddad Feb 22 '18 at 21:03