



Deferring Observable code until subscription in RxJava

23 JULY 2015 on rxjava

I've grown fond of RxJava's `defer()` as a tool for ensuring `Observable` code runs when subscribed (rather than when created). I've written about `defer()` before but I'd like to go into more detail here.

Suppose you've got this data class:

```
public class SomeType {  
    private String value;  
  
    public void setValue(String value) {  
        this.value = value;  
    }  
  
    public Observable<String> valueObservable() {  
        return Observable.just(value);  
    }  
}
```

What do you think will be printed when I run this code?

```
SomeType instance = new SomeType();  
Observable<String> value = instance.valueObservable();  
instance.setValue("Some Value");  
value.subscribe(System.out::println);
```

If you guessed "Some Value", you're wrong. It actually prints out "null", since `value` had yet to be initialized when `Observable.just()` is called.

`just()`, `from()`, and other `Observable` creation tools store the value of data when created, not when subscribed. In this case that's not the desired behavior – I want `valueObservable()` to represent the current value whenever requested.

Do It Yourself

One solution is to use `Observable.create()` since it allows you to control the sequence precisely for each subscriber:

```
public Observable<String> valueObservable() {  
    return Observable.create(subscriber -> {  
        subscriber.onNext(value);  
        subscriber.onCompleted();  
    });  
}
```

Now `valueObservable()` will emit the current value when subscribed. It's roughly equivalent to what `Observable.just()` does, except it retrieves `value` on subscription (not creation).

The only issue here is that I have been wary of writing custom operators since reading Dávid Karnok's [excellent series of](http://blog.danlew.net/2015/07/23/deferring-observable-code-until-subscription-in-rxjava/)

[articles on operators](#). My main takeaway from the series is that operators are tricky to write correctly. Just take a look at [this post](#), which shows how `Observable.just()` will need to change in order to support backpressure and unsubscription.

While the above code works now, how do I know it will always work for future versions of RxJava? And how do I know I've safely covered all my bases, like backpressure and unsubscription? I'm not an expert on operator development. As such, I've tried avoiding custom operators unless necessary.

The Simple Way

Here's an alternative that uses no custom operators:

```
public Observable<String> valueObservable() {  
    return Observable.defer(() -> Observable.just(value));  
}
```

All I did was wrap the original code with `defer()`, but now the behavior is what I want. None of the code inside of `defer()` is executed until subscription. We only call `Observable.just()` when someone requests the data.

I prefer this solution for a couple reasons:

1. It's simpler than `Observable.create()` – no need to call `onCompleted()`.
2. It uses built-in operators, so they're (probably) implemented correctly.

The only downside to `defer()` is that it creates a new `Observable` each time you get a subscriber. `create()` can use the same function for each subscriber, so it's more efficient. As always, measure performance and optimize if necessary.

Going Deeper

The code above is simple for pedagogical reasons, but realistically we could have replaced all of it with a `BehaviorSubject`. Let's take a look at something more complex.

Suppose we want a method which writes data to disk, then returns that data. This is one way to do it with `defer()`:

```
public Observable<SomeType> createSomeType(String value)
{
    return Observable.defer(() -> {
        SomeType someType = new SomeType();
        someType.setValue(value);

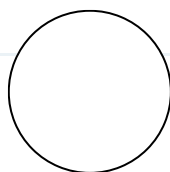
        try {
            db.writeToDisk(someType);
        }
        catch (IOException e) {
            return Observable.error(e);
        }

        return Observable.just(someType);
    });
}
```

This sample is more complex; it writes data to the disk and calls `onError` if there are issues. The basic idea remains the same, though: we don't want any of the code to execute until subscription.

There are multiple ways of attacking the above problems. `defer()` is just one of them, but I've found it handy.

Dan Lew



Read [more posts](#) by this author.


Share this post



📍 *Minneapolis* 🔗 <http://blog.danlew.net/about/>

20 Comments

Dan Lew Codes

 Login ▾ Recommend 15 Share

Sort by Best ▾



Join the discussion...

**sebaslogen** • a year ago

There is a newer alternative to transform a method call into an observable triggered on subscription.

Instead of `defer()+just()+myMethod()` you can use `fromCallable()+myMethod()`

2 ^ | ▾ • Reply • Share >

**Toan Tran** → sebaslogen • a year ago

Thanks for this. I found it was added from rxjava 1.015, which is better than the bare call to create new Observable and shorter in handling error.

^ | ▾ • Reply • Share >

**Nathan** • 2 years ago

Thanks Dan, I found this article helpful when deciding on when to use defer vs a subject. <http://davesexton.com/blog/...>

1 ^ | ▾ • Reply • Share >

**Nitzan Dori** • 10 months ago

Hi, I want to defer a call until a certain event occurs, let's say there is a network request `Observable<x> getX()`, but it can only happen if the system is in a "connected" state, so I need to subscribe to `getX` but without actually making the request until I get the "connected" event.

I've managed to do it using `flatMap` and wrapping the original `getX()` so I will return an observable that emits the states and once it gets "connected" it will `flatMap` and return `getX()`.

I can post my transformer if you like.

I wonder if there is a better approach?

Tnx

^ | ▾ • Reply • Share >

**Dan Lew** Mod → Nitzan Dori • 10 months ago

That's reasonable. Alternatively, you could look into using the ``switch()`` operator, which can switch to the network request whenever the network is available.

^ | ▾ • Reply • Share >

**Siavash** • a year ago

I want to change some data in ui when one element changes. Can I use defer for this purpose? I found this answer on stackoverflow that use create:

<http://stackoverflow.com/a/...>

now i cant understand how I can do this with defer! because I pass subject to another class and i want every time something changed emit data to subscribers.

^ | v • Reply • Share >



Dan Lew Mod → Siavash • a year ago

You can't do that with defer(). Defer is only good for cold observables (that don't update on their own, but instead emit items when subscribed to).

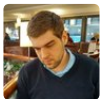
^ | v • Reply • Share >



Siavash → Dan Lew • a year ago

I see a video about rxjava common mistakes; as far as i remember you are the lecturer of this video. in this video you(he) said it's better to do not create observable but with that answer on stackoverflow i have to create it. is there any other way for this?

^ | v • Reply • Share >



preslavrachev • a year ago

Hi, Dan, a great post as always! I've been trying to make a connection between it, and your previous post about loading data from different sources. For my remote source, I am using Retrofit which returns an Observable of some type.

What I am trying to wrap my head around, is if the remote API method will fire off immediately. I've looked into the Retrofit code, and i don't think that Observable responses are deferred by default. This means that if i want to concatenate my remote Observable response, together with a local cache Observable, I would have to wrap the remote Observable using `Observable.defer(...)`, am I right. This way, network calls won't fire until the local cache Observable "gives up", and the subscriber hits the remote one. Is my logic correct?

^ | v • Reply • Share >



Dan Lew Mod → preslavrachev • a year ago

Retrofit doesn't make the request until it gets subscribe, so using ``concat(cache, retrofit)`` would work correctly. I actually wrote a solution using this idea here: <http://blog.danlew.net/2015...>

What you should look into is the idea of hot vs. cold observables. It gives one a way to reason about the flow of data based on how the source emits. I think this is the best article on it: <http://davesexton.com/blog/...>

^ | v • Reply • Share >



Chris Arriola • a year ago

Thanks for all the RxJava articles you've been writing so far, they've been super useful!

I noticed all your examples use lambda expressions instead of an anonymous class. Is that just for brevity and in code you actually use anonymous classes? Or do you use something like `retrolambda` in your project? I'm a little wary of `retrolambda` and was wondering what your thoughts are on that.

^ | v • Reply • Share ›



Dan Lew **Mod** ➔ Chris Arriola • a year ago

We use Retrolambda, it hasn't given us any problems for a long time.

Here, I generally like using it for brevity's sake.

^ | v • Reply • Share ›



Thomas Nield • 2 years ago

Interesting... so is this a much better solution to make imperative code work with reactive code without the hassle of subjects.

^ | v • Reply • Share ›



Dan Lew **Mod** ➔ Thomas Nield • 2 years ago

Typically yes.

^ | v • Reply • Share ›



Vasili Chyrvon • 2 years ago

Thanks, Dan! I just started to learn the Rx and your posts really help.

^ | v • Reply • Share ›



Alexey • 2 years ago

Don't you think that such usage of 'defer' leads to confusion?

Because you neither get observable with current value and you do not get observable with value+value changes.

I think it's better to use either `Observable.just(someType.getValue())` if you want to get immediate value or create subject (+ may be 'startWith' operator) if you want to observe value + its changes.

^ | v • Reply • Share ›



artem_zin ➔ Alexey • 2 years ago

Defer has clear purpose — prevent calculation of `someType.getValue()` until subscription. In your example with `Observable.just()` it will be calculated in thread that called `Observable.just()`

^ | v • Reply • Share ›



Alexey ➔ artem_zin • 2 years ago

Yes, but in this example value is not calculated, this is where confusion comes from.

^ | v • Reply • Share ›

READ THIS NEXT

How to upgrade to RxAndroid 1.0

A number of people have asked me recently, "What the hell happened to RxAndroid?" The fact of the matter...

YOU MIGHT ENJOY

Loading data from multiple sources with RxJava

Suppose I have some Data that I query from the network. I could simply hit the network each time...

Dan Lew Codes © 2017

Proudly published with **Ghost**