四种Subject的基本用法

RxSwift - step by step

← 返回视频列表 | 预计阅读时间: *20分钟* | ‹ PREVIOUS | NEXT ›

上节末尾，我们提到了**Subject**。既然它可以同时作为Observable和Observer，我们就直奔主题，从一个叫做`PublishSubject`的对象开始，感受下Subject的用法。

PublishSubject

顾名思义，`PublishSubject`就像个出版社，到处收集内容，此时它是一个Observer，然后发布给它的订阅者，此时，它是一个Observable。

首先，创建一个`PublishSubject`很简单，就像创建一个普通的类对象一样：

```
let subject = PublishSubject<String>()
```

其中`PublishSubject`的泛型参数，表示它可以订阅到的，以及可以发布的事件类型。

其次，当我们把`subject`当作Observer的时候，可以使用`onNext`方法给它发送事件：

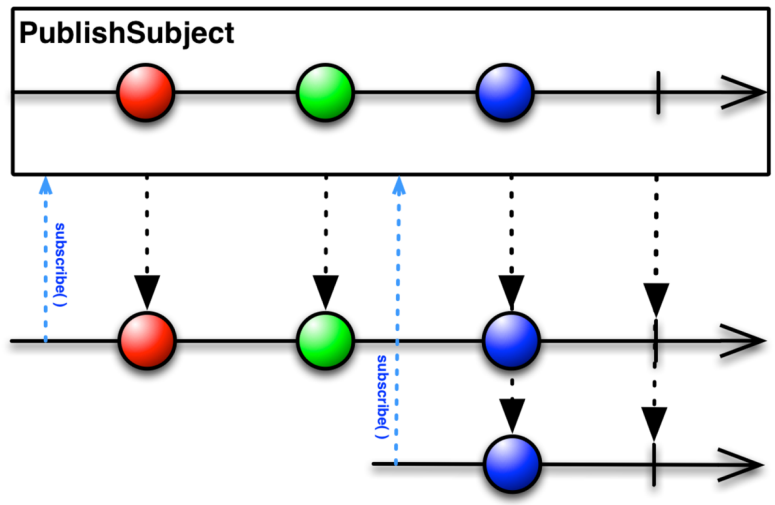
```
subject.onNext("Episode1 updated")
```

第三，当我们把`subject`当作Observable的时候，订阅它的代码和订阅普通的Observable完全一样：

```
let sub1 = subject.subscribe(onNext: {
    print("Sub1 - what happened: \($0)")
})
```

但是执行一下就会发现，控制台上不会显示任何订阅消息，也就是说`sub1`没有订阅到任何内容。这是因为`PublishSubject`执行的是“会员制”，它只会把最新的消息通知给消息发生之前的订阅者。用序列图表示出来，就是这样的：

PublishSubject



可以看到，在红灯之前订阅，就可以订阅到红、绿、蓝全部事件，如果在蓝灯之前订阅，就只能订阅到蓝色事件了。于是，为了订阅到`subject`的事件，我们得把订阅的代码，放到通知`subject`前面：

```
let sub1 = subject.subscribe(onNext: {
    print("Sub1 - what happened: \($0)")
})

subject.onNext("Episode1 updated")
```

重新执行下，就能看到*Sub1 - what happened: Episode1 updated*的通知了。然后，再来观察下面代码的执行结果：

```
sub1.dispose()

let sub2 = subject.subscribe(onNext: {
    print("Sub2 - what happened: \($0)")
})

subject.onNext("Episode2 updated")
subject.onNext("Episode3 updated")

sub2.dispose()
```

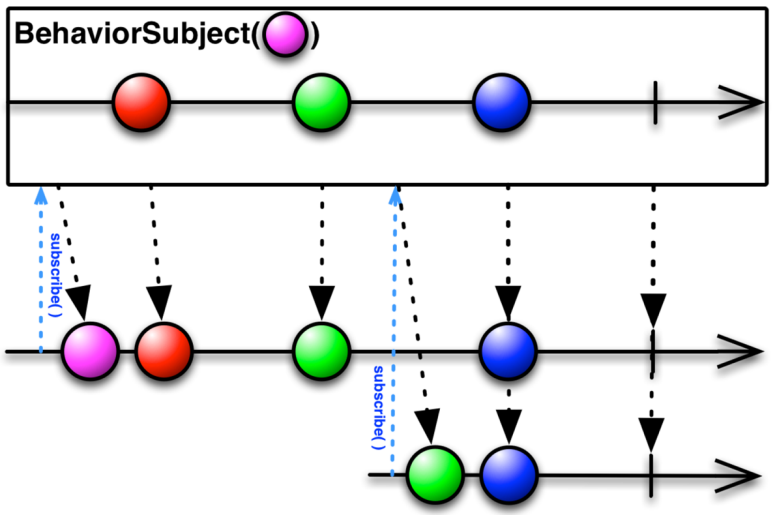
- 首先，在执行过`sub1.dispose()`之后，`sub1`就不会再接收来自`subject`的任何消息了；
- 其次，`subject`有了一个新的订阅者`sub2`；
- 第三，`subject`又捕获到了两条新的消息。按照刚才的说法，`sub2`不会接收到订阅之前的消息，因此，我们应该只能在控制台看到*Sub2 - what happened: Episode2 updated*和*Sub2 - what happened: Episode3 updated*这两条消息；
- 最后，`sub2`取消对`subject`的订阅；

重新执行一下，就能在控制台看到结果了。

BehaviorSubject

如果你希望Subject从“会员制”变成“试用制”，就需要使用BehaviorSubject。它和PublisherSubject唯一的区别，就是只要有人订阅，它就会向订阅者发送最新的一次事件作为“试用”。

BehaviorSubject



如图所示，BehaviorSubject带有一个紫灯作为默认消息，当红灯之前订阅时，就会收到紫色及以后的所有消息。而在绿灯之后订阅，就只会收到绿灯及以后的所有消息了。因此，当初始化一个BehaviorSubject对象的时候，要给它指定一个默认的推送消息：

```
let subject = BehaviorSubject<String>(
    value: "RxSwift step by step")
```

然后，当我们再执行先订阅，后发送消息的逻辑时：

```
let sub1 = subject.subscribe(onNext: {
    print("Sub1 - what happened: \($0)")
})

subject.onNext("Episode1 updated")
```

由于BehaviorSubject有了一个默认的事件，sub1订阅之后，就会陆续收到*RxSwift step by step*和*Sub1 - what happened: Episode1 updated*的消息了。此时，如果我们再添加一个新的订阅者：

```
let sub2 = subject.subscribe(onNext: {
    print("Sub2 - what happened: \($0)")
})
```

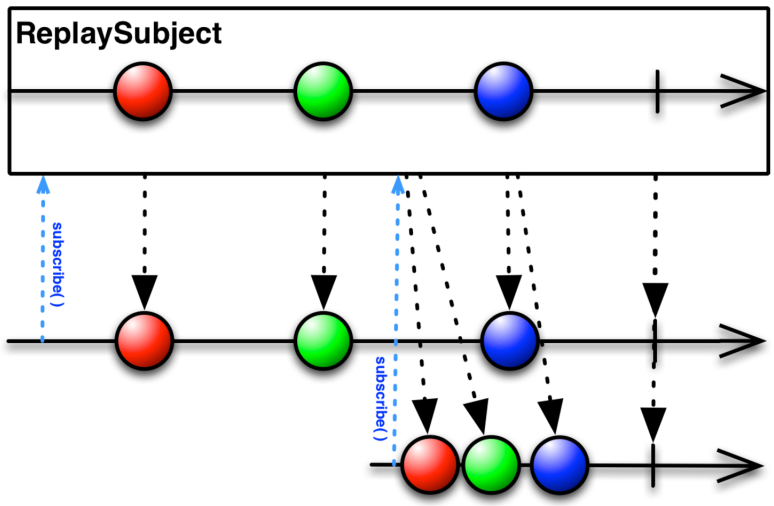
此时，sub2就只能订阅到*Sub2 - what happened: Episode1 updated*消息了。如果我们要让sub2在订阅的时候获取到过去所有的消息，就需要使用ReplaySubject。

ReplaySubject

ReplaySubject的行为和BehaviorSubject类似，都会给订阅者发送历史消息。不同地方有两点：

- ReplaySubject没有默认消息，订阅空的ReplaySubject不会收到任何消息；
- ReplaySubject自带一个缓冲区，当有订阅者订阅的时候，它会向订阅者发送缓冲区内的所有消息；

ReplaySubject



ReplaySubject缓冲区的大小，是在创建的时候确定的：

```
let subject = ReplaySubject<String>.create(bufferSize: 2)
```

这样，我们就创建了一个可以缓存两个消息的ReplaySubject。作为Observable，它此时是一个空的事件序列，订阅它，不会收到任何消息：

```
let sub1 = subject.subscribe(onNext: {
    print("Sub1 - what happened: \($0)")
})
```

然后，我们让subject接收3个事件，sub1就会收到三次事件订阅：

```
subject.onNext("Episode1 updated")
subject.onNext("Episode2 updated")
subject.onNext("Episode3 updated")

// Sub1 - what happened: Episode1 updated
// Sub1 - what happened: Episode2 updated
// Sub1 - what happened: Episode3 updated
```

这时，我们再给subject添加一个订阅者：

```
let sub2 = subject.subscribe(onNext: {
    print("Sub2 - what happened: \($0)")
})

// Sub2 - what happened: Episode2 updated
// Sub2 - what happened: Episode3 updated
```

由于subject缓冲区的大小是2，它会自动给sub2发送最新的两次历史事件。在控制台中执行一下，就可以看到注释中的结果了。

Variable

除了事件序列之外，在平时的编程中我们还经常需遇到一类场景，就是需要某个值是有“响应式”特性的，例如可以通过设置这个值来动态控制按钮是否禁用，是否显示某些内容等。为了方便这个操作，RxSwift还提供了一个特殊的subject，叫做Variable。

我们可以像定义一个普通变量一样定义一个Variable：

```
let stringVariable = Variable("Episode1")
```

当我们要订阅一个Variable对象的时候，要先明确使用asObservable()方法。而不像其他subject一样直接订阅：

```
let stringVariable = Variable("Episode1")

let sub1 = stringVariable
    .asObservable()
    .subscribe {
    print("sub1: \($0)")
}

// sub1: next(Episode1)
```

而当我们要给一个Variable设置新值的时候，要明确访问它的value属性，而不是使用onNext方法：

```
stringVariable.value = "Episode2"

// sub1: next(Episode2)
```

最后要说明的一点是，Variable只用来表达一个“响应式”值的语义，因此，它有以下两点性质：

- 绝不会发生.error事件；
- 无需手动给它发送.complete事件表示完成；

因此，下面的代码都会导致编译错误：

```
// !!! The following code CANNOT compile !!!
stringVariable.asObservable().onError(MyError.myError)
stringVariable.asObservable().onCompleted()
```

What's next?

以上，就是RxSwift中4种Subject的用法。至此，我们就一切准备就绪了，接下来，我们就在一个真实的App里，逐步了解如何用RxSwift实现一些之前常见的开发任务。

‹ Prev: 理解create和debug operator

四种Subject的基本用法

Next: Todo I - 通过一个真实的App体会Rx的基本概念 ›

关于我们

想循序渐进的跟上最新的技术趋势？想不为了学点东西到处搜索？想找个伙伴一起啃原版技术经典书？技术之外，还想了解高效的工作流技巧？甚至，工作之余，想找点儿东西放松心情？没问题，我们用4K开发视频，配以详尽的技术文档，以及精心准备的广播节目，让你渴望成长的技术需求，也是一种享受。

Email Address

10@boxue.io

客户服务

2085489246

相关链接

- 版权声明
- 用户隐私以及服务条款
- 京ICP备15057653号-1
- 京公网安备 11010802020752号

关注我们

在任何你常用的社交平台上关注我们，并告诉我们你的任何想法和建议！

邮件列表

订阅泊学邮件列表以了解泊学视频更新以及最新活动，我们不会向任何第三方公开你的邮箱！

Email address 立即订阅