

常用的获取事件操作符

RxSwift - step by step

[← 返回视频列表](#)

预计阅读时间: 8分钟

[< PREVIOUS](#)[NEXT >](#)

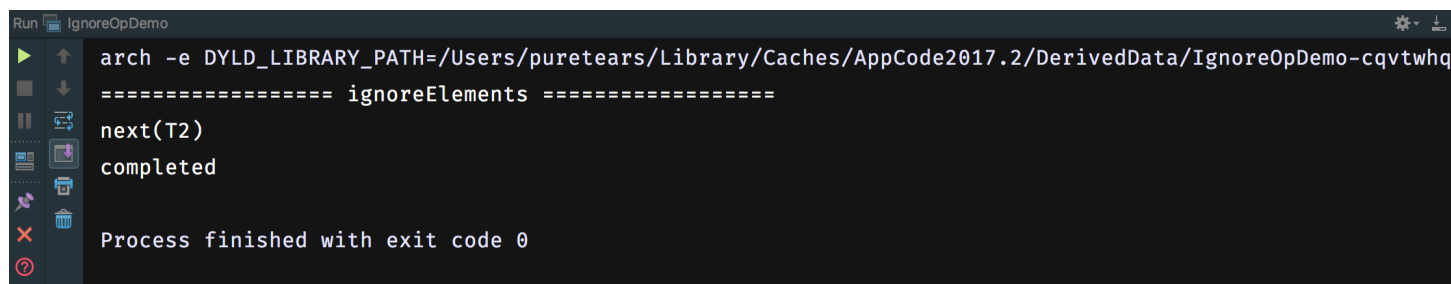
在这一节，我们来看如何选择序列中的特定事件。

elementAt

首先，是选择序列中的第n个事件。还是基于上一节的例子，如果我们要订阅第二个任务，就可以这样：

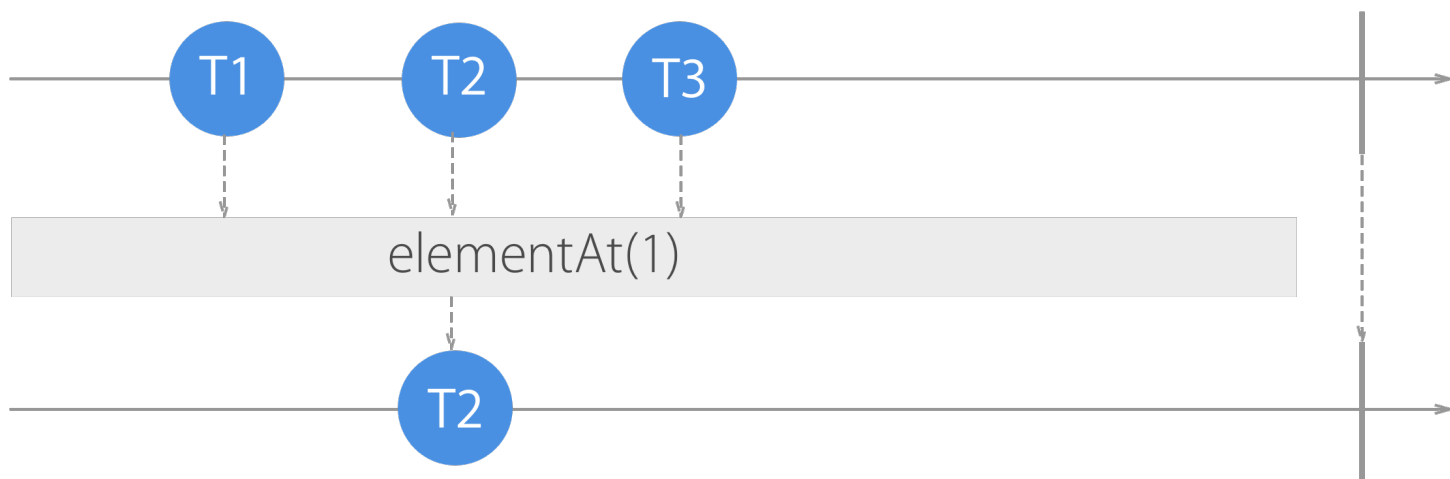
```
tasks.elementAt(1)
    .subscribe {
        print($0)
    }
    .addDisposableTo(bag)
```

执行一下，就可以看到结果了：



```
Run IgnoreOpDemo
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvtwhq
==== ignoreElements =====
next(T2)
completed
Process finished with exit code 0
```

要注意的是，`elementAt`的参数和数组的索引一样，第一个任务的索引是0，而不是1。它的序列图，是这样的：

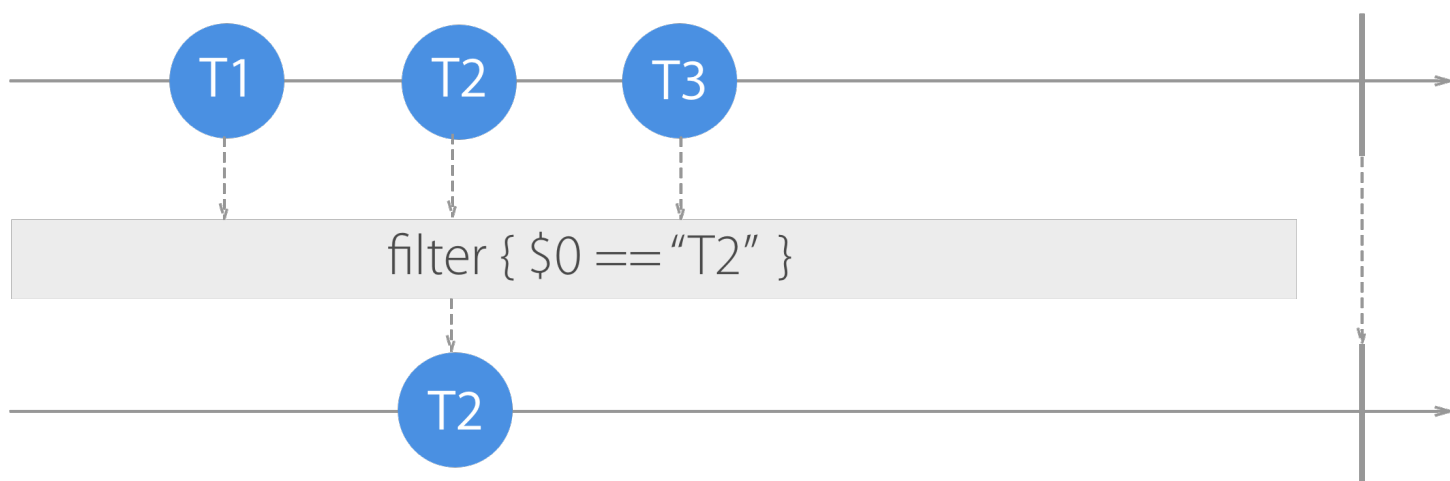


filter

除了用事件的索引来选择之外，我们也可以用 `closure` 设置选择事件的标准，这就是 `filter` 的作用，它会选择序列中所有满足条件的元素。例如，我们订阅值是 `T2` 的事件：

```
tasks.filter { $0 == "T2" }  
    .subscribe {  
        print($0)  
    }  
    .addDisposableTo(bag)
```

执行一下，就会发现，结果和刚才是一样的。这说明，`filter` 同时过滤掉了 `T1` 和 `T3`，它会检查序列中的每一个事件。它的序列图，是这样的：



take

除了选择订阅单一事件之外，我们也可以选择一次性订阅多个事件，例如，选择序列中的前两个事件：

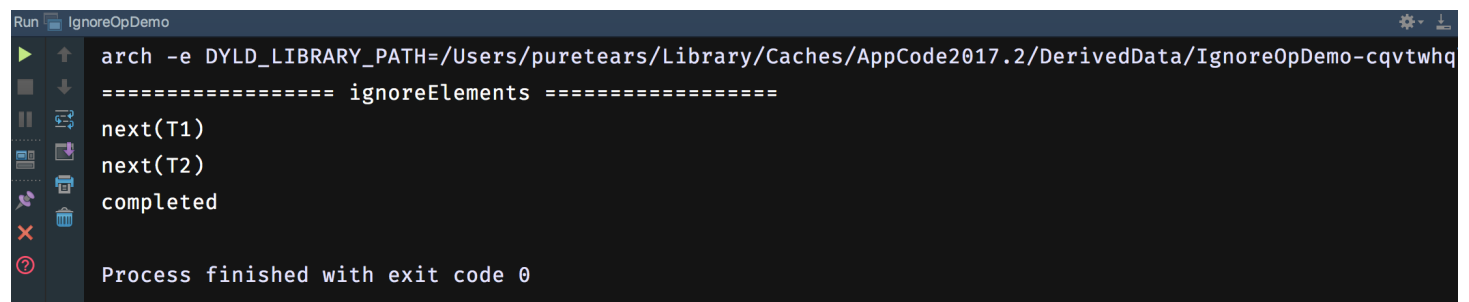
```
tasks.take(2)
    .subscribe {
        print($0)
    }
    .addDisposableTo(bag)
```

takeWhile / takeWhileWithIndex

或者，我们也可以用一个closure来指定“只要条件为true就一直订阅下去”这样的概念。例如，只要任务不是T3就一直订阅下去：

```
tasks.takeWhile {
    $0 != "T3"
}
    .subscribe {
        print($0)
    }
    .addDisposableTo(bag)
```

我们会在控制台看到这样的结果：



```
Run IgnoreOpDemo
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvtwhq
===== ignoreElements =====
next(T1)
next(T2)
completed
Process finished with exit code 0
```

这看似很简单，但有时候，我们经常会对它的用法产生一些错觉。例如，错把订阅终止条件写在了`takeWhile`参数里，就像这样：

```
// This following is WRONG
tasks.takeWhile {
    $0 == "T3"
}
    .subscribe {
        print($0)
    }
    .addDisposableTo(bag)
```

上面的代码会得到什么结果呢？实际上只能订阅到`.completed`。因为，当匹配到第一个事件的时候，`"T1" == "T3"`是`false`，所以订阅就结束了。有时，我们不仅希望约束订阅到的事件，还需要获得事件在序列中的位置。为此，RxSwift提供了一个功能和`takeWhile`类似的operator，只是在它的closure里，可以同时访问到事件值和事件在队列中的索引，像这样：

```
tasks.takeWhileWithIndex { (value, index) in
    value != "T3" && index < 1
}
.subscribe {
    print($0)
}
.addDisposableTo(bag)
```

`takeWhileWithIndex`的closure有两个参数，第一个是事件的值，第二个是事件在序列中的索引。它的语义和`takeWhile`是完全一样的，需要注意的仍旧是，在closure里写的，是读取事件的条件，而不是终止读取的条件。

takeUntil

除了使用closure表示订阅条件之外，我们也可以依赖另外一个外部事件，表达“直到某件事件发生前，一直订阅”这样的语义。例如下面的例子：

```
example("ignoreElements") {
    let tasks = PublishSubject<String>()
    let bossHasGone = PublishSubject<Void>()
    let bag = DisposeBag()

    tasks.takeUntil(bossHasGone).subscribe {
        print($0)
    }
    .addDisposableTo(bag)

    tasks.onNext("T1")
    tasks.onNext("T2")
    tasks.onNext("T3")
    tasks.onCompleted()
}
```

我们新添加了一个`bossHasGone`，表示老板已经不在了的事件，尽管我们使用了`takeUntil` operator，但`bossHasGone`还没有任何事件发生，因此我们仍旧会订阅到所有事件。现在，在T2和T3之间，给`bossHasGone`发送个事件：

```
tasks.onNext("T1")
tasks.onNext("T2")

bossHasGone.onNext()

tasks.onNext("T3")
tasks.onCompleted()
```

重新执行一下，就会看到，我们只能订阅到T1和T2了。

What's next?

以上，就是和订阅特定事件相关的operators。包括了订阅特定位置事件（`elementAt`）、订阅指定个数事件（`take`）到订阅满足特定条件事件（`filter` / `takeWhile` / `takeUntil`）。它们和上一节中的所有忽略事件operators一起，在Rx世界里，统称为事件过滤类型的operators。在下一节，我们就通过一个真实的App，进一步了解这些operators的用法。

[< Prev: 常用的忽略事件操作符](#)[☰ 常用的获取事件操作符](#)[Next: Todo IV - 进一步理解Subject的实际应用 >](#)

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！

[立即订阅](#)