

# 常用的忽略事件操作符

RxSwift - step by step

[← 返回视频列表](#)**预计阅读时间: 12分钟**[< PREVIOUS](#)[NEXT >](#)

在对RxSwift的基本概念有了一个比较全面的认识之后，在进一步开发App之前，我们要先积累更多RxSwift operators相关的知识。理解它们并不困难，你不必完全记住它们，但是至少要在心里留有印象。因为，对特定的事件序列使用正确的operator，是编写语义正确的Rx代码的重要基础。

在这一节中，我们先来看如何忽略特定的事件。从忽略全部事件到自定义指定事件，RxSwift提供了多种operators。为了演示operators的用法，我们用SPM新建了一个RxSwift项目，并在Sources目录添加了两个文件：

```
→ IgnoreOpDemo tree -L 2
.
├── IgnoreOpDemo.xcodeproj
│   ├── GeneratedModuleMap
│   ├── RxBlocking_Info.plist
│   ├── RxCocoaRuntime_Info.plist
│   ├── RxCocoa_Info.plist
│   ├── RxSwift_Info.plist
│   ├── project.pbxproj
│   ├── project.xcworkspace
│   ├── xcshareddata
│   └── xcuserdata
├── Package.pins
├── Package.swift
├── Sources
│   ├── helper.swift
│   └── main.swift
└── Tests
```

其中，*helper.swift*中只定义了一个函数：

```
func example(_ description: String,
             action: (Void) -> Void) {

    print("===== \((description) =====")
    action()
}
```

它的作用有两个，一来，可以在任何测试代码执行前，打印一个提示方便我们观察结果；二来，`action`可以提供一个独立的scope，方便我们利用`DisposeBag`回收资源。

而我们所有的演示代码，都会编写在`main.swift`里。

## Ignore elements

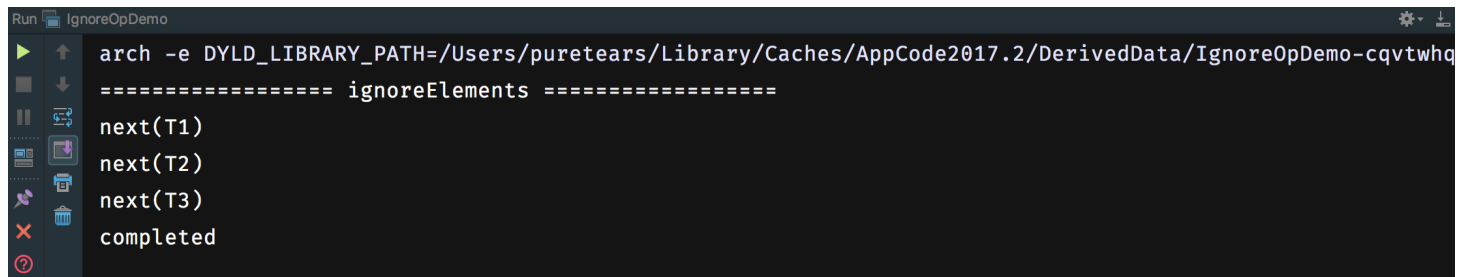
第一个要介绍的operator是`ignoreElements`，它会忽略序列中所有的`.next`事件。我们用一个`PublishSubject<String>`来演示它的用法：

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

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

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

这是一个标准的`PublishSubject`订阅，我们可以在控制台看到类似下面的结果：



```
Run IgnoreOpDemo
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvtwhq
===== ignoreElements =====
next(T1)
next(T2)
next(T3)
completed
```

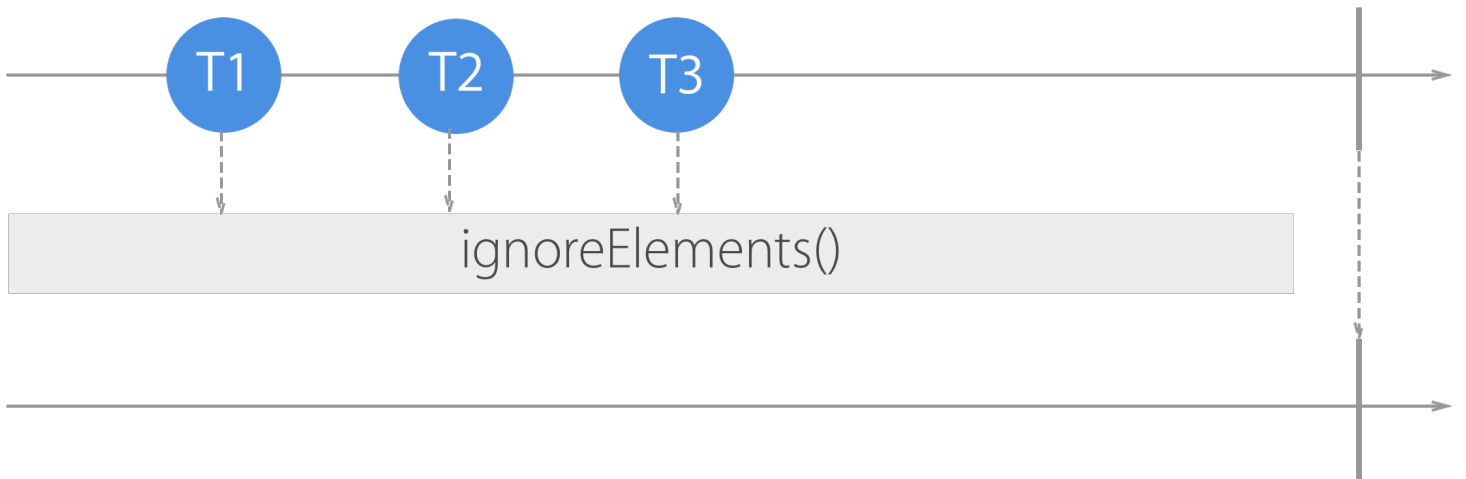
如果我们要忽略掉所有的`.next`事件，只接受`.completed`事件，可以把之前的订阅代码改成这样：

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

重新执行一下，就会看到下面的结果了：

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

用序列图把它画出来，就是这样的：



## skip

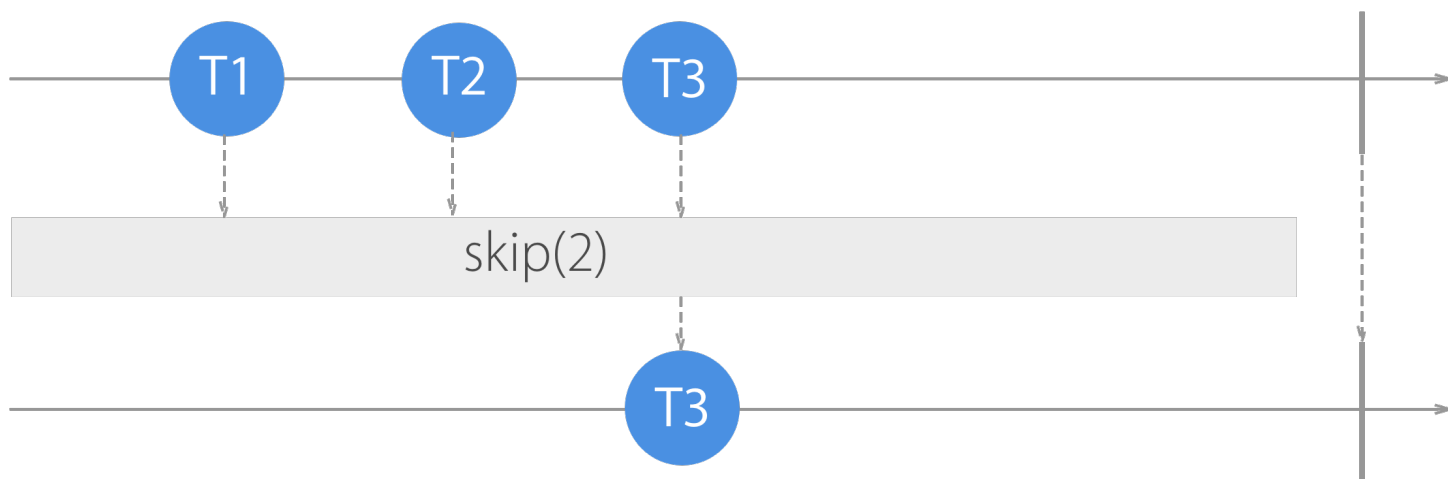
除了一次性忽略所有的`.next`之外，我们还可以选择忽略事件序列中特定个数的`.next`。例如，在我们的例子里，假设队列中前两个任务都是流水线上其它人完成的，而你只需要完成第三个任务，就可以这样：

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

重新执行一下，就能看到下面的结果了：

```
Run IgnoreOpDemo
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvtwhq
===== ignoreElements =====
next(T3)
completed
Process finished with exit code 0
```

其中，`skip`表示从事件序列中的第一个元素开始，忽略其参数指定个数的事件，用序列图表示，就是这样的：



## skipWhile / skipUntil

除了可以忽略指定个数的事件外，我们还可以通过一个closure自定义忽略的条件，这个operator叫做`skipWhile`。但它和我们想象中有些不同的是，它不会“遍历”事件序列上的所有事件，而是当遇到第一个不满足条件的事件之后，就不再忽略任何事件了。

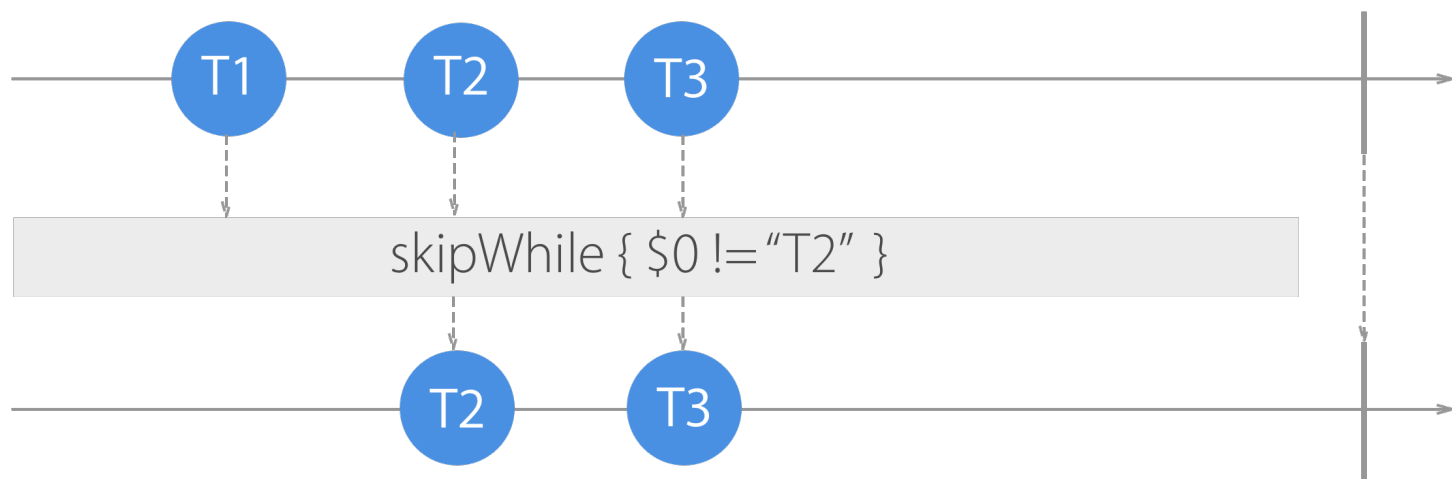
例如，为了忽略名称不等于T2事件，我们编写了下面的代码：

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

但执行一下，却会看到这样的结果：

```
Run IgnoreOpDemo
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvtwhq
===== ignoreElements =====
next(T2)
next(T3)
completed
Process finished with exit code 0
```

可以看到，`skipWhile`只忽略了T1，而没有忽略T3。而这就是While的含义，它只忽略到第一个不满足条件的事件，然后，就完成任务了。用序列图表示，就是这样的：



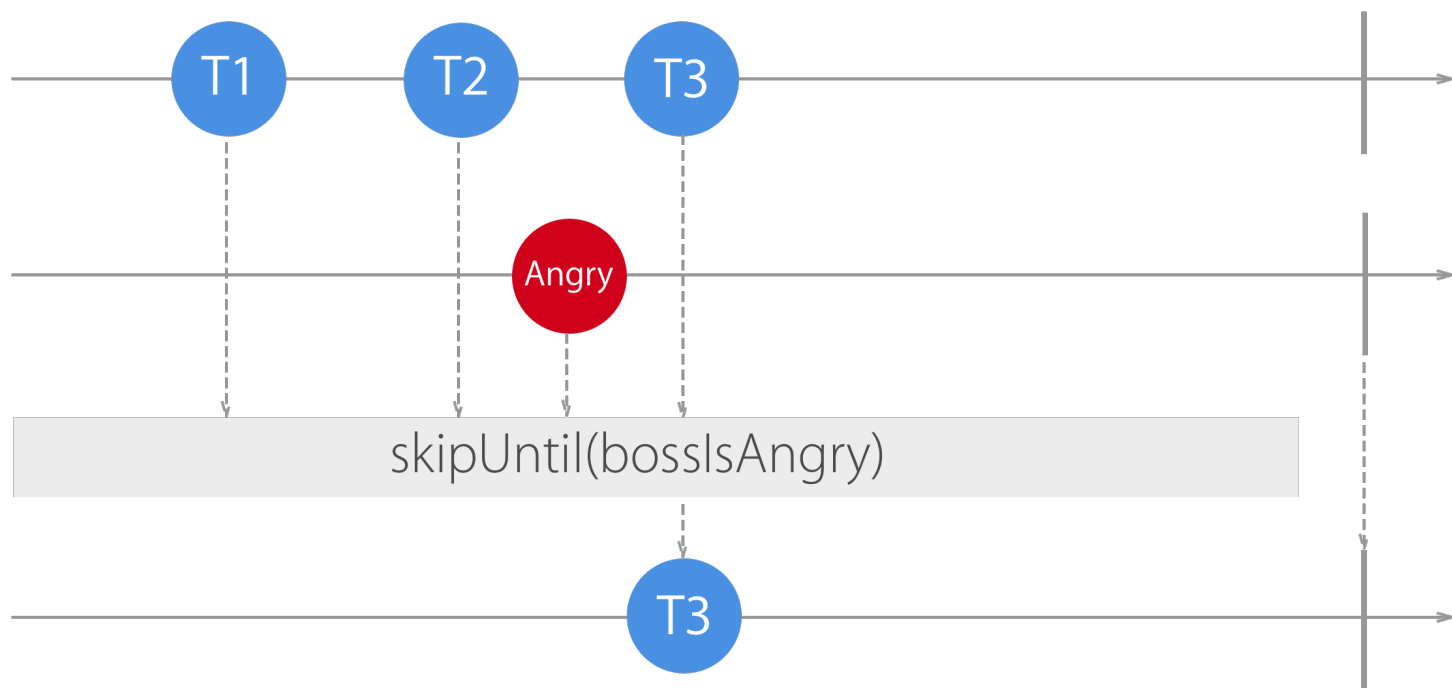
另外一个和`skipWhile`类似的operator是`skipUntil`，它不用一个closure指定忽略的条件，而是使用另外一个事件序列中的事件。例如，我们先把代码改成这样：

```
let tasks = PublishSubject<String>()
let bossIsAngry = PublishSubject<Void>()
let bag = DisposeBag()

tasks.skipUntil(bossIsAngry)
    .subscribe {
        print($0)
    }
    .addDisposableTo(bag)

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

执行一下就会看到，我们不会订阅到任何事件。这就是`skipUntil`的效果，它会一直忽略`tasks`中的事件，直到`bossIsAngry`中发生事件为止。把它用序列图表示出来，是这样的：



为了观察到这个效果，我们在T2和T3之间添加下面的代码：

```
tasks.onNext("T1");
tasks.onNext("T2");
bossIsAngry.onNext();
tasks.onNext("T3");
```

重新执行一下，就可以看到订阅T3的结果了。

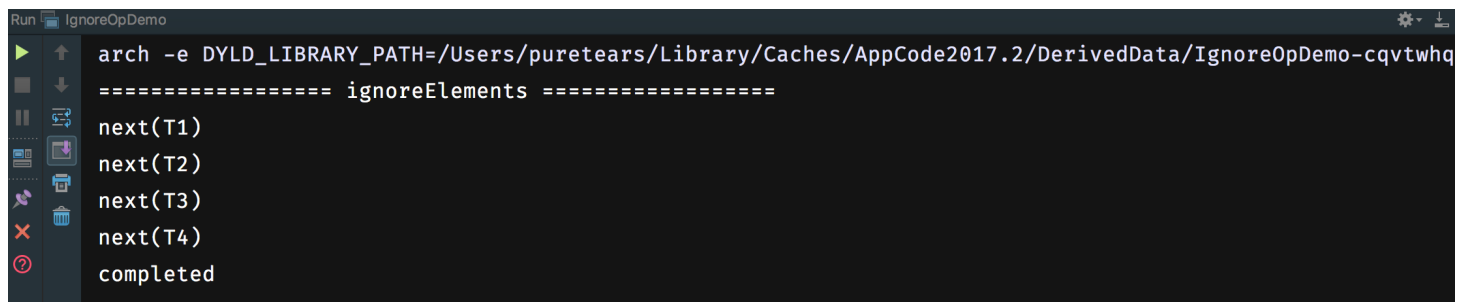
```
Run IgnoreOpDemo
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvthq
===== ignoreElements =====
next(T3)
completed
Process finished with exit code 0
```

## distinctUntilChanged

最后一个要介绍的，是通过`distinctUntilChanged`忽略序列中连续重复的事件。例如下面这个例子：

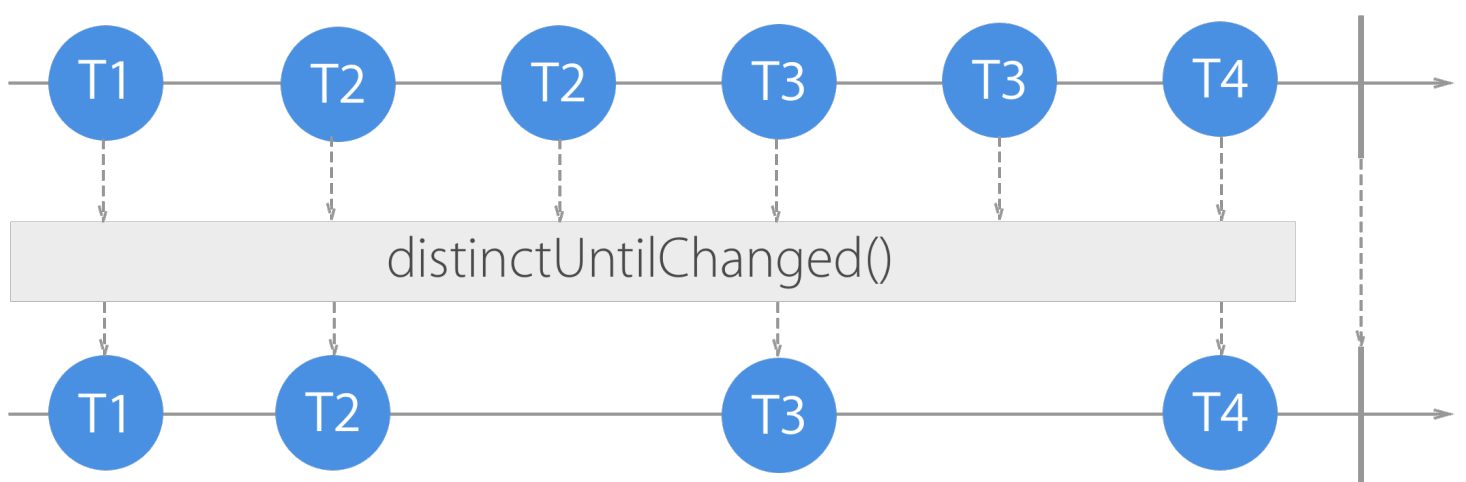
```
example("ignoreElements") {  
    let tasks = PublishSubject<String>()  
    let bag = DisposeBag()  
  
    tasks.distinctUntilChanged()  
        .subscribe {  
            print($0)  
        }  
        .addDisposableTo(bag)  
  
    tasks.onNext("T1")  
    tasks.onNext("T2")  
    tasks.onNext("T2")  
    tasks.onNext("T3")  
    tasks.onNext("T3")  
    tasks.onNext("T4")  
    tasks.onCompleted()  
}
```

由于T2和T3都属于连续重复的事件，因此它们各自的第二次出现都会被忽略，我们只能订阅到下面这样的结果：



```
Run IgnoreOpDemo  
arch -e DYLD_LIBRARY_PATH=/Users/puretears/Library/Caches/AppCode2017.2/DerivedData/IgnoreOpDemo-cqvtwhq  
===== ignoreElements =====  
next(T1)  
next(T2)  
next(T3)  
next(T4)  
completed
```

它的序列图是这样的：



但是，如果把T2放到两个T3中间，此时就没有任何连续重复的事件了，我们会订阅到所有任务。

## What's next?

以上，就是和忽略事件相关的operators，简单来说，就是从忽略全部（`ignoreElements`）、到忽略指定个数（`skip(n)`）、再到忽略指定条件的事件（`skipWhile`和`skipUntil`），最后是忽略连续重复的事件（`distinctUntilChanged`）。理解了它们之后，下一节，我们来看如何获取特定的事件。

[◀ Prev: Todo III - 自定义Observable统一用户交互处理](#)

[☰ 常用的忽略事件操作符](#)

[Next: 常用的获取事件操作符 >](#)

#### 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