

## Todo VI - 更好的处理授权提示

⌕ Back to series (/series/rxswift-101)

在这一节，我们修复第一次打开图片选择界面由于授权导致的白屏问题。大家可以在这里下载 ()项目的初始模板，唯一的改动，就是在Flash.swift里，我们把 flash 变成了 UIViewController 的 extension。因为稍后，我们要在不同的view controller中使用它。

把用户的授权结果变成一个Observable

之前我们说过，iOS的用户授权动作是异步的。因此，为了能在用户完成授权操作之后继续更新UI，我们得先把授权的结果封装成一个Observable。实际上，这个Observable只可能是下面三种情况：

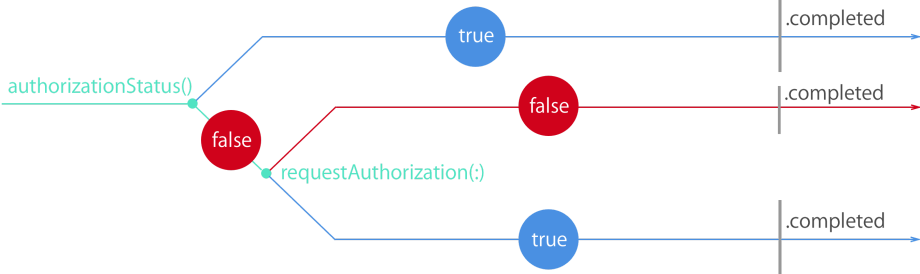
⊕ 字号

● 字号

🖌 默认主题

🖌 金色主题

🖌 暗色主题



- 如果用户已授权，事件序列就是：.next(true)，.completed()；
- 如果用户未授权，序列的第一个事件就一定是 .next(false)。然后，如果用户拒绝授权，序列中的事件就是：.next(false) 和 .completed。否则，就是 .next(true) 和 .completed；

有了这个思路之后，我们在新添加的PHPhotoLibrary+Rx.swift中添加下面的代码：

```
extension PHPhotoLibrary {
    static var isAuthorized: Observable<Bool> {
        return Observable.create { observer in
            DispatchQueue.main.async {
                if authorizationStatus() == .authorized {
                    observer.onNext(true)
                    observer.onCompleted()
                } else {
                    requestAuthorization {
                        observer.onNext($0 == .authorized)
                        observer.onCompleted()
                    }
                }
            }
        }
        return Disposables.create()
    }
}
```

其中，用到了 PHPhotoLibrary 的两个API：

- authorizationStatus 获取当前授权状态；
- requestAuthorization 申请用户授权；

至于为什么要把通知 observer 的代码放在 DispatchQueue.main.async 里，是为了避免在自定义的事件序列中影响其它Observable的订阅，甚至是把整个UI卡住。理解了这些之后，上面的代码就很好理解了，就是之前那个事件流的代码表示。接下来，我们只要订阅这个Observable就好了。

### 订阅用户的授权结果

订阅的部分，应该写在 `PhotoCollectionViewController.viewDidLoad` 方法里。先别着急，这个过程要比我们想象的复杂一点，我们不能直接订阅 `isAuthorized` 的 `onNext` 并处理 `true/false` 的情况，因为单一的事件值并不能反映真实的授权情况。按照之前分析的：

- 授权成功的序列可能是：`.next(true)`，`.completed` 或 `.next(false)`，`.next(true)`，`.completed`；
- 授权失败的序列则是：`.next(false)`，`.next(false)`，`.completed`；

因此，我们需要把 `isAuthorized` 这个事件序列处理一下，分别处理授权成功和失败的情况。

## 订阅成功事件

首先来订阅授权成功事件，我们只要忽略掉事件序列中所有的 **false**，并读到第一个 **true**，就可以认为授权成功了。使用“过滤型”operator可以轻松完成这个任务：

```
// In PhotoCollectionViewController
override func viewDidLoad() {
    super.viewDidLoad()
    setCellSpace()

    let isAuthorized = PHPhotoLibrary.isAuthorized

    isAuthorized
        .skipWhile { $0 == false }
        .take(1)
        .subscribe(onNext: {
            [weak self] _ in
            // Reload the photo collection view
        })
        .addDisposableTo(bag)
}
```

可以看到，上面的代码里，我们使用了 `skipWhile` 和 `take` 模拟了忽略所有 **false** 并读取第一个 **true** 这个动作。然后，在 `.next` 事件的订阅里，我们直接更新UI就好了：

```
// Inside subscribe(onNext: )
// Reload the photo collection view
if let `self` = self {
    self.photos = PhotoCollectionViewController.loadPhotos()

    DispatchQueue.main.async {
        self.collectionView?.reloadData()
    }
}
```

这次，为什么又使用了 `DispatchQueue.main.async` 呢？因为，当我们调用 `requestAuthorization` 请求用户授权时，在这个API的说明中 (<https://developer.apple.com/reference/photos/phphotolibrary/1620736-requestauthorization>)，我们可以找到这样一段话：

Photos may call your handler block on an arbitrary serial queue. If your handler needs to interact with UI elements, dispatch such work to the main queue.

也就是说，我们传递给 `requestAuthorization` 的closure参数有可能并不在主线程中执行，一旦如此，我们订阅的授权结果的代码也就不会在主线程中执行。但是，由于我们在订阅中更新了UI，如果这个代码不在主线程中，App就会立即闪退了。因为，需要人为保证它执行在主线程里。

不过，实际上，我们并不经常在RxSwift的订阅代码里使用GCD，RxSwift提供了一个更简单的机制，叫做 **scheduler**，在后面的内容里，我们会专门提到它。现在可以先体验下它的用法，为了保证订阅代码一定执行在主线程，我们可以把之前的代码改成这样：

```
isAuthorized
    .skipWhile { $0 == false }
    .take(1)
    .observeOn(MainScheduler.instance)
    .subscribe(onNext: {
        [weak self] _ in
        // Reload the photo collection view
        if let `self` = self {
            self.photos = PhotoCollectionViewController.loadPhotos()
            self.collectionView?.reloadData()
        }
    })
    .addDisposableTo(bag)
```

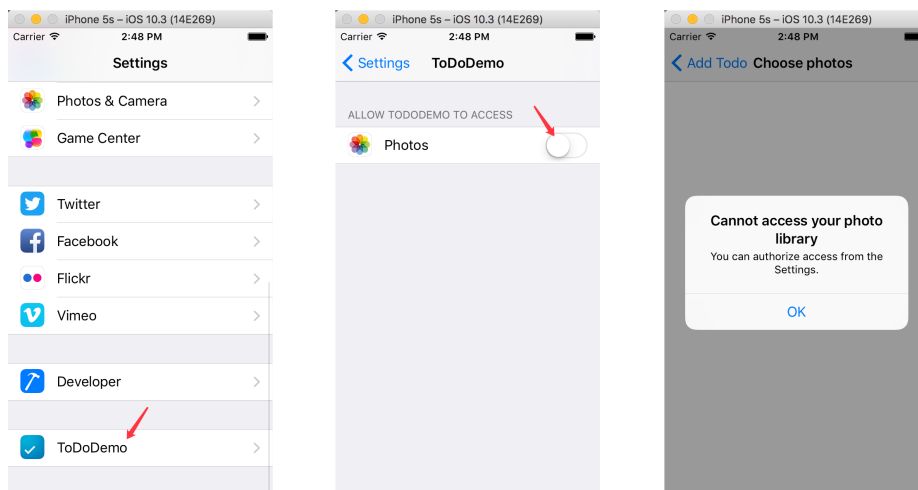
其中，`observeOn(MainScheduler.instance)` 就表示了在主线程中执行订阅代码，这样的逻辑，结果和之前是一样的，只是它看起来，更RxSwift一些。现在，在模拟器里把TodoDemo删掉，重新编译、安装和执行，在iOS提示我们授权访问图片库的时候选择OK，就可以看到图片立即刷新的效果了。

## 订阅失败事件

接下来，我们处理拒绝授权的情况。这种情况相比成功简单一些，因为它对应的事件序列只有一种情况：`.next(false)`，`.next(false)`，`.completed`。因此，我们只要对事件序列中所有元素去重之后，订阅最后一个`.next`事件，如果是`false`，就可以确定是用户拒绝授权了。因此，在订阅成功授权的代码后面，继续添加下面的代码：

```
isAuthorized
    .distinctUntilChanged()
    .takeLast(1)
    .filter { $0 == false }
    .subscribe(onNext: { [weak self] _ in
        self?.flash(title: "Cannot access your photo library",
            message: "You can authorize access from the Settings.",
            callback: { [weak self] _ in
                self?.navigationController?.popViewController(animated: true)
            })
    })
    .addDisposableTo(bag)
```

如果你还记得之前我们讲过的各种“过滤型”operator，理解上面的代码应该没有任何困难。我们用alert view给用户显示了一个错误提示，并在用户关闭提示时，自动退回到todo编辑界面。为了看到这个错误提示，我们在Settings里，把ToDoDemo的访问授权关掉就好了：



## What's next?

在这一节中，我们了解了一种典型的场景：如何用RxSwift改进需要用户授权操作的用户体验。实际上，任何一个需要授权的操作都可以采用类似的方式来处理。接下来，在进一步开发App之前，我们来看除了“过滤型”之外的另一大类operator，*transform operators*。



职场漂泊的你，每天多学一点。

从开发、测试到运维，让技术不再成为你成长的绊脚石。我们用打磨产品的精神去传播知识，把最新的移动开发技术，通过简单的图表，清晰的视频，简明的文字和切实可行的例子一一向你呈现。让学习不仅是一种需求，也是一种享受。

## 泊学动态

一个工作十年PM终创业的故事（二）(https://www.boxueio.com/after-the-full-upgrade-to-swift3)

Mar 4, 2017

人生中第一次创业的"10有" (https://www.boxueio.com/founder-chat)

Jan 9, 2016

猎云网采访报道泊学 (http://www.lieyunwang.com/archives/144329)

Dec 31, 2015

What most schools do not teach (https://www.boxueio.com/what-most-schools-do-not-teach)

Dec 21, 2015

一个工作十年PM终创业的故事（一）(https://www.boxueio.com/founder-story)

May 8, 2015

## 泊学相关

关于泊学



加入泊学



泊学用户隐私及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)

## 联系泊学

Email: 10[AT]boxue.io (mailto:10@boxue.io)

QQ: 2085489246

2017 © Boxue, All Rights Reserved. 京ICP备15057653号-1 (http://www.miibeian.gov.cn/) 京公网安备 11010802020752号 (http://www.beian.gov.cn/portal/registerSystemInfo?recordcode=11010802020752)

友情链接 SwiftV (http://www.swiftv.cn) | Seay信息安全博客 (http://www.cnseay.com) | Swift.gg (http://swift.gg/) | Laravist (http://laravist.com/) | SegmentFault (https://segmentfault.com) | 骛青K的博客 (http://blog.dianqk.org/)