Todo VI - 更好的处理授权提示

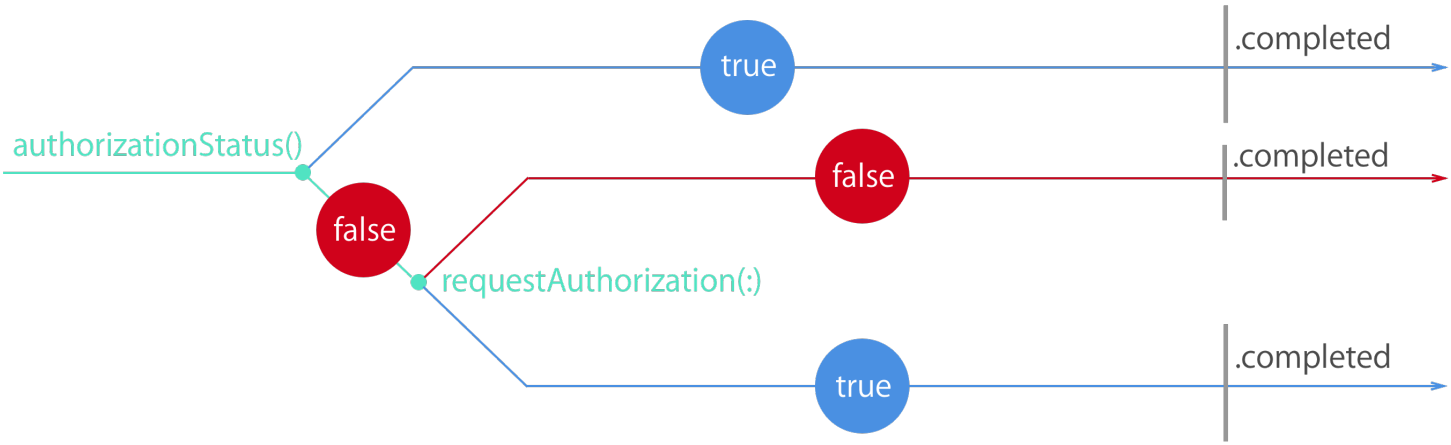
RxSwift - step by step

← 返回视频列表 | 预计阅读时间: *18分钟* | ‹ PREVIOUS | NEXT ›

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：



- 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 {
                    // Notify the subscriber of the authorization failure
                    // event explicitly. We may omit this event by default.
                    observer.onNext(false)
                    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，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)
```

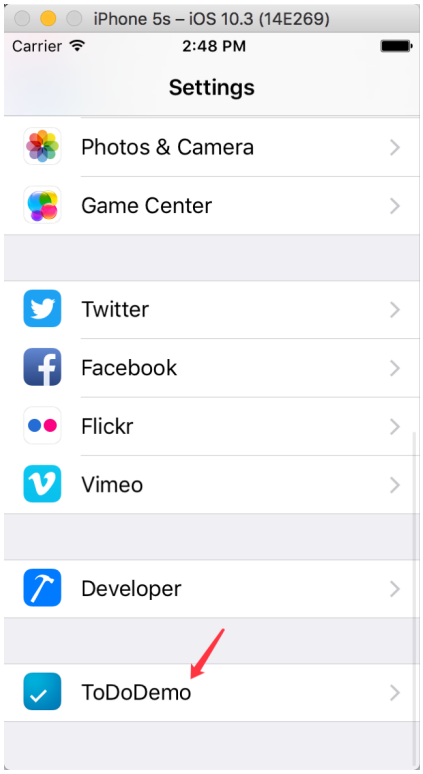
其中，observeOn(MainScheduler.instance)就表示了在主线程中执行订阅代码，这样的逻辑，结果和之前是一样的，只是它看起来，更RxSwift一些。现在，在模拟器里把TodoDemo删掉，重新编译、安装和执行，在iOS提示我们授权访问图片库的时候选择OK，就可以看到图片立即刷新的效果了。

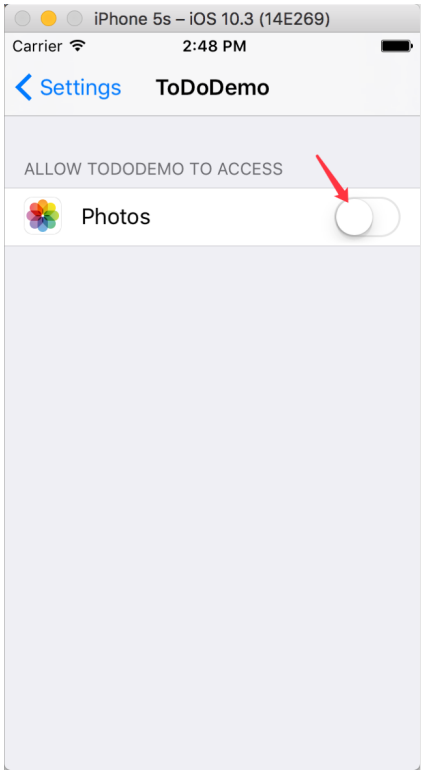
订阅失败事件

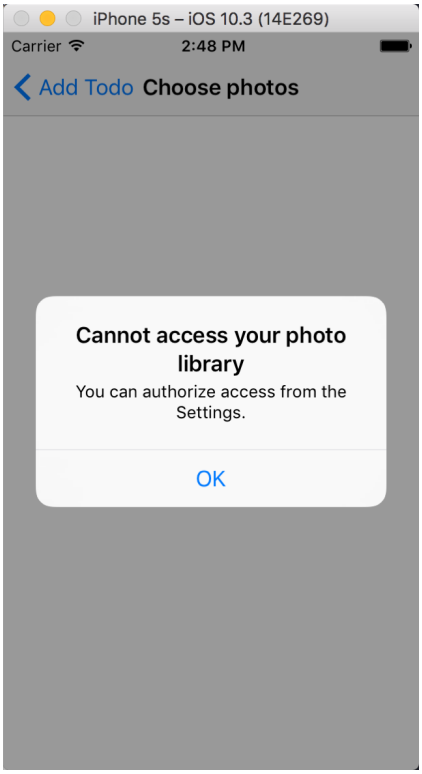
接下来，我们处理拒绝授权的情况。这种情况相比成功简单一些，因为它对应的事件序列只有一种情况：.next(false)，.next(false)，.completed。因此，我们**只要对事件序列中所有元素去重之后，订阅最后一个.next事件，如果是false**，就可以确定是用户拒绝授权了。因此，在订阅成功授权的代码后面，继续添加下面的代码：

```
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不会有任何困难。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*。

‹ Prev: Todo V - 理解重复订阅Observable的行为

Todo VI - 更好的处理授权提示

Next: 了解常用的transform operators ›

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