

Todo II - 如何通过Subject传递数据

RxSwift - step by step

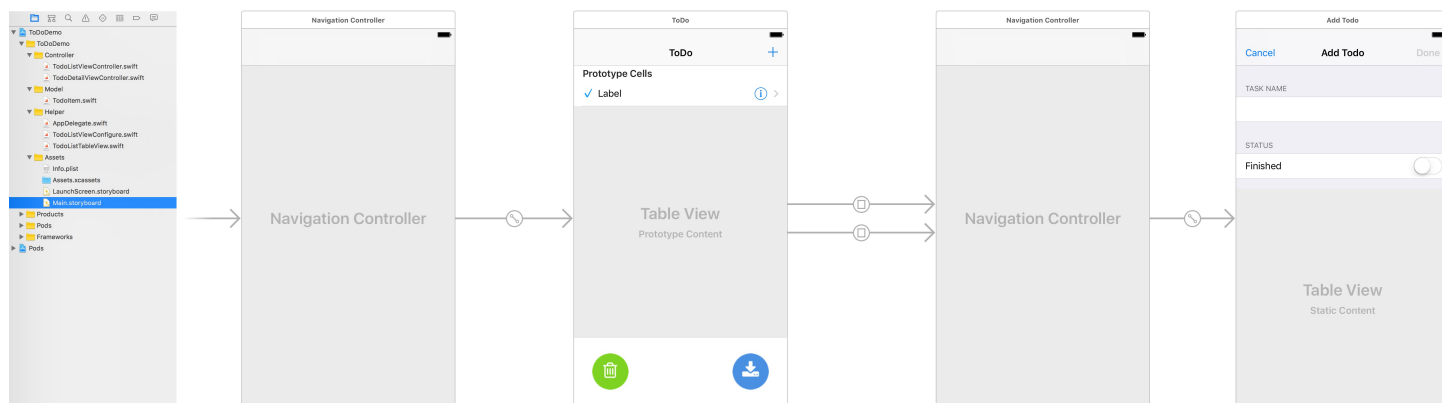
← [返回视频列表](#)

预计阅读时间: 35分钟

< [PREVIOUS](#)

[NEXT](#) >

基于上个Todo的例子，在这段视频里，我们完成添加和编辑Todo任务的功能。在开始之前，先了解下基于上段视频完成的例子，我们做了哪些主要修改：

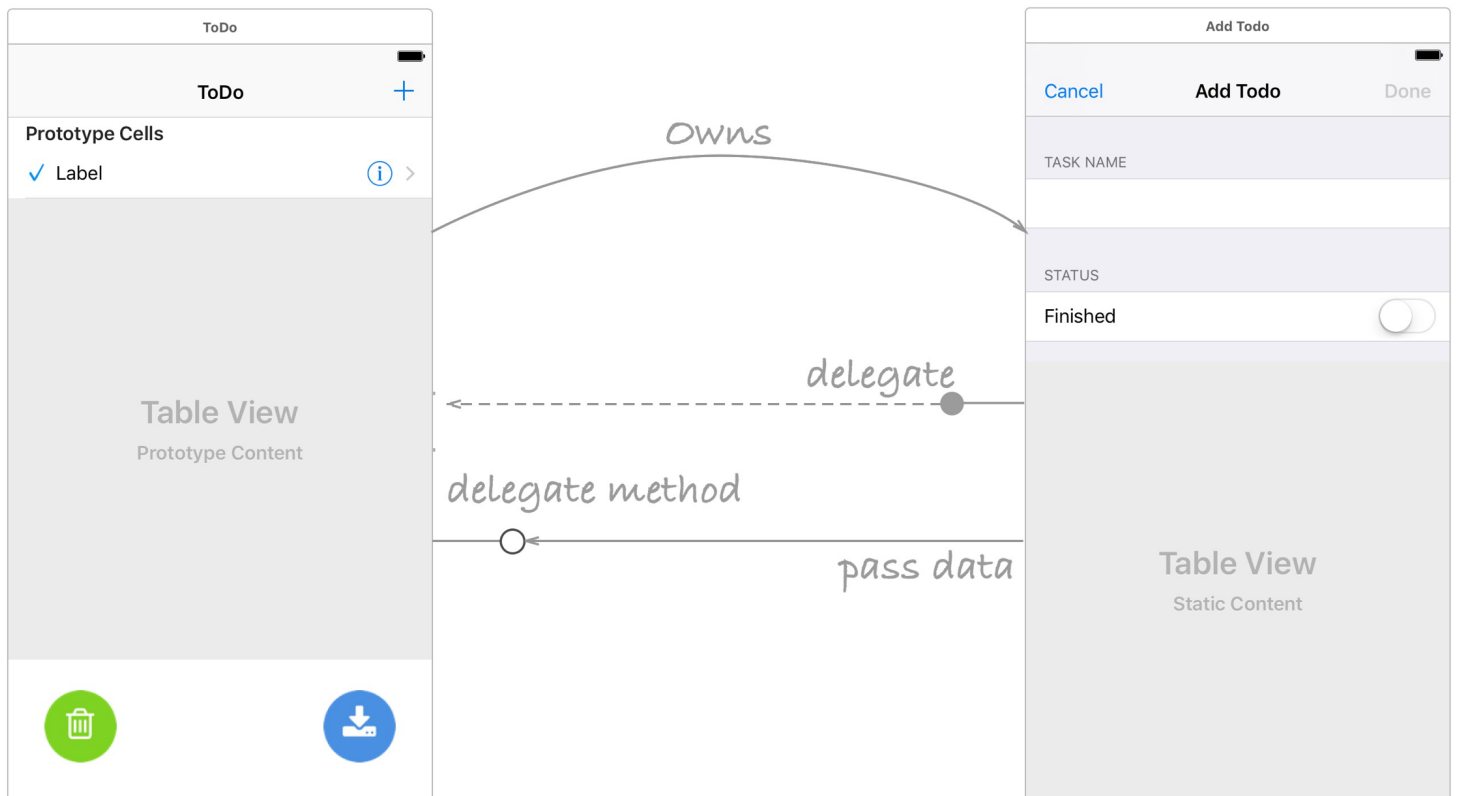


- 修改了之前添加按钮的代码，让它打开一个创建Todo的View；
- 给table cell的accessory添加了segue，让它打开一个编辑当前Todo的View；
- 新建了一个`ToDoDetailViewController`，处理添加和编辑Todo的逻辑；
- 在`ToDoListViewController`中，根据segue的目标修改了新打开View的标题；

大家可以在[这里](#)下载项目的起始模板。

新建Todo

接下来，我们就动手实现添加一个新的Todo。这个事情唯一的要点，就是如何把在`ToDoDetailViewController`中创建的Todo内容，传递给`ToDoListViewController`。在使用RxSwift之前，Cocoa的套路是这样的：



1. 定义一个`protocol`，并让这个`protocol`类型的对象成为`TodoDetailViewController`的`delegate`；
2. 让`TodoListViewController`实现这个`protocol`中的方法，并设置成`TodoDetailViewController`的`delegate`对象；
3. `TodoDetailViewController`通过`delegate`方法发送数据；

于是，当我们在Controller中发送数据的时候，方法一直是“不对称”的，可以通过给属性赋值把数据“发出去”，但是却要通过`protocol`“传回来”。

借助RxSwift，我们可以更方便和统一地在Controllers之间发送数据。简单来说，让发送数据的一方包含一个Observable对象，让接收方直接订阅就好了。



创建PublishSubject

按照这个思路，我们先在`TodoDetailViewController`中，添加下面的代码：

```
class TodoDetailViewController: UITableViewController {
    fileprivate let todoSubject = PublishSubject<TodoItem>()
    var todo: Observable<TodoItem> {
        return todoSubject.asObservable()
    }
    // ...
}
```

在继续之前，思考两个问题：

1. 为什么这里我们使用了一个Subject对象呢？
2. 为什么这个Subject是一个PublishSubject呢？

对于问题一，是因为在`TodoDetailViewController`内部，我们需要一个Observer，它要订阅到`UITextField`和`UISwitch`的值；但同时，我们也需要它是一个Observable，可以让`TodoListViewController`订阅到之后更新Todo列表的显示。

而对于问题二，相信等我们完成Todo编辑之后，你自然就会明白了，我们暂且先不管它。

这里，我们还使用了一个小技巧，为了避免`todoSubject`意外从`TodoDetailViewController`外部接受`onNext`事件，我们把它定义成了`fileprivate`属性。对外，只提供了一个仅供订阅的Observable属性`todo`。

实现onNext

接下来，我们要在用户创建Todo的时候，给`todoSubject`发送`onNext`事件。首先，给`TodoDetailViewController`添加一个保存Todo内容的属性：

```
class TodoDetailViewController: UITableViewController {
    var todoItem: TodoItem!
    // ...
}
```

其次，在`viewWillAppear`中初始化它：

```
class TodoDetailViewController: UITableViewController {

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        todoName.becomeFirstResponder()

        todoItem = TodoItem()
    }

    // ...
}
```

最后，在`Done`按钮的事件处理方法里，通知`todoSubject`：

```
class TodoDetailViewController: UITableViewController {
    @IBAction func done() {
        todoItem.name = todoName.text!
        todoItem.isFinished = isFinished.isOn

        todoSubject.onNext(todoItem)
        dismiss(animated: true, completion: nil)
    }
}
```

至此，`TodoDetailViewController`这一侧的装修就完工了。我们可以到`TodoListViewController`去订阅了。

在另一个Controller中订阅

要在另外一个Controller中订阅`todo`，最核心的问题，就是如何得到`TodoDetailViewController`对象。然而，这对我们来说，并不是一个问题，通过Segue进行场景转换的时候，我们已经通过`topViewController`得到了。因此，在`TodoListViewController`中，添加下面的代码：

```
class TodoListViewController: UIViewController {
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        let naviController =
            segue.destination as! UINavigationController
        var todoDetailController =
            naviController.topViewController as! TodoDetailViewController

        if segue.identifier == "AddTodo" {
            todoDetailController.title = "Add Todo"

            todoDetailController.todo.subscribe(
                onNext: {
                    [weak self] newTodo in
                        self?.todoItems.value.append(newTodo)
                },
                onDisposed: {
                    print("Finish adding a new todo.")
                }
            ).addDisposableTo(bag)
        }
    }
}
```

其中，新增的代码，就是订阅`todo`的部分，我们从事件中订阅到要添加的内容，然后塞进`todoItems`，由于它也是响应式的，`UITableView`就能自动更新了。

资源被正常回收了么？

此时，尽管已经可以正常添加Todo了，但是如果你足够细心就可以发现，控制台并没有打印`Finish adding a new todo.`的提示。也就是说，在`dismiss`了`TodoDetailViewController`之后，`todoSubject`并没有释放，我们应该在某些地方导致了资源泄漏。

为了进一步确认这个问题，在Podfile中添加下面的内容：

```
post_install do |installer|
    installer.pods_project.targets.each do |target|
        if target.name == 'RxSwift'
            target.build_configurations.each do |config|
                if config.name == 'Debug'
                    config.build_settings['OTHER_SWIFT_FLAGS'] ||= ['-D', 'TRACE_RESOURCES']
                end
            end
        end
    end
end
```

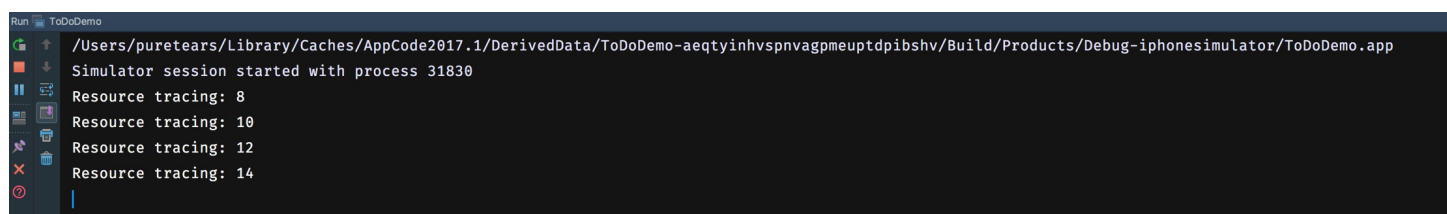
简单来说，就是找到项目中的RxSwift target，在它的Debug配置中，添加`-D TRACE_RESOURCES`编译参数，并在Terminal中重新执行`pod install`更新下RxSwift。然后，在`TodoDetailViewController`的`viewWillAppear`方法中，添加下面的代码：

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    todoName.becomeFirstResponder()

    todoItem = TodoItem()

    print("Resource tracing: \(RxSwift.Resources.total)")
}
```

这样，我们就能够在控制台看到当前RxSwift分配的资源计数。重新build并执行整个项目，然后多添加几个Todo，就会在控制台看到resource一直在增加：



为什么会这样呢？其实，看下订阅的代码就明白了：

```
todoDetailController.todo.subscribe(
    onNext: {
        [weak self] newTodo in
        self?.todoItems.value.append(newTodo)
    },
    onDisposed: {
        print("Finish adding a new todo.")
    }
).addDisposableTo(bag)
```

我们把`todo.subscribe`返回的订阅对象放在了`ToDoListViewController.bag`里，但只要App不退出，作为initial view controller的`ToDoListViewController`是不会被释放的，因此，它的`bag`里装的订阅对象只会越来越多。这显然不是我们想要的，怎么办呢？

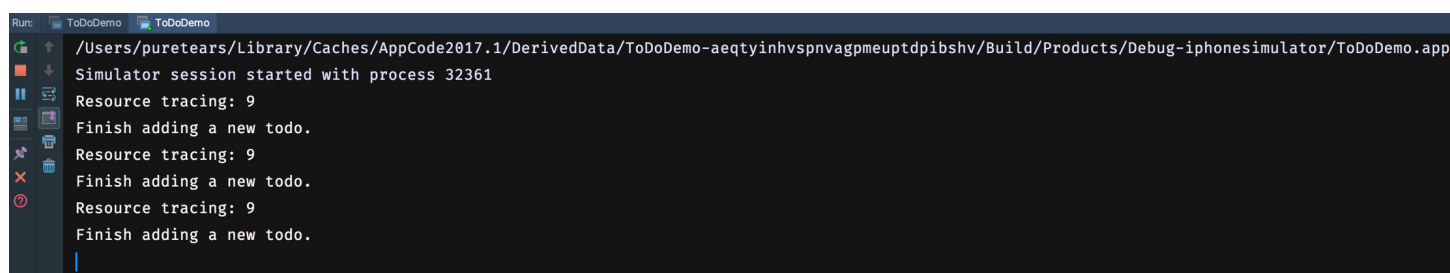
一个“头疼医头”的办法，就是把`todo.subscribe`返回的订阅对象放在`ToDoDetailViewController`的`bag`里。这样，当controller被`dismiss`的时候，`bag`里的订阅就会自动被取消，`todoSubject`占用的资源也就被回收了。为了验证这个想法，我们在`ToDoDetailViewController`中添加下面的代码：

```
class ToDoDetailViewController: UITableViewController {
    // ...
    var bag = DisposeBag()
}
```

然后，修改`ToDoListViewController`中的订阅代码：

```
todoDetailController.todo.subscribe(
    onNext: {
        [weak self] newTodo in
        self?.todoItems.value.append(newTodo)
    },
    onDisposed: {
        print("Finish adding a new todo.")
    }
).addDisposableTo(todoDetailController.bag)
```

重新编译执行，现在，多次打开添加Todo的界面，就会发现资源可以正常回收了：



但事情至此还没结束，可能你会觉得这样写代码感觉怪怪的，甚至有些危险。因为我们要依赖一个Controller (`ToDoDetailViewController`) 中的某个属性 (`bag`) 才能得以工作正常。而常规的开发经验通常告诉我们，如此密切的耦合关系通常是各种问题滋生的温床。这至多，只能算一个“非主流”的办法。

那么，更“主流”的办法是什么呢？

希望你还记得，对于一个Observable来说，除了所有订阅者都取消订阅会导致其被回收之外，Observable自然结束 (`onCompleted`) 或发生错误结束 (`onError`) 也会自动让所有订阅者取消订阅，并导致Observable占用的资源被回收。

因此，当`ToDoDetailViewController dismiss`之后，实际上我们也不会再使用它添加新的Todo了，这时，我们应该给`todoSubject`发送`onCompleted`事件，明确告知RxSwift，这个事件序列结束了：

```
class ToDoDetailViewController: UITableViewController {
    // ...
    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        todoSubject.onCompleted()
    }
}
```

于是，我们之前的订阅代码就可以进一步改成这样：

```
_ = todoDetailController.todo.subscribe(  
    onNext: {  
        [weak self] newTodo in  
        self?.todoItems.value.append(newTodo)  
    },  
    onDisposed: {  
        print("Finish adding a new todo.")  
    }  
)
```

至此，所有添加Todo的工作，就结束了。接下来，我们实现编辑的部分。

编辑Todo

编辑Todo，和新建Todo绝大部分工作都是一样的，只是相比新建，有两个关键问题要想清楚：

- 如何把要编辑的内容传递给`TodoDetailViewController`；
- 编辑后的内容如何传回来更新UI；

第一个问题，我们可以在segue中通过Identifier来确定如果是编辑操作，就读取当前table view中被选中的cell，然后根据cell的IndexPath，读取到Todo的内容，并传递给`TodoDetailViewController`。

有了这个思路之后，首先，我们来处理`prepare(for:sender:)`方法：

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    let naviController = segue.destination as! UINavigationController  
    var todoDetailController =  
        naviController.topViewController as! TodoDetailViewController  
  
    if segue.identifier == "AddTodo" {  
        // ...  
    }  
    else if segue.identifier == "EditTodo" {  
        // 1. The edit segue  
        todoDetailController.title = "Edit todo"  
  
        // 2. Get the selected cell index  
        if let indexPath = tableView.indexPath(  
            for: sender as! UITableViewCell) {  
  
            // 3. Pass the selected todo  
            todoDetailController.todoItem =  
                todoItems.value[indexPath.row]  
        }  
    }  
}
```

这样，我们就把用户选择编辑的Todo，传递给了`TodoDetailViewController.todoItem`。

其次，在`TodoDetailViewController`中，当`todoItem`不为`nil`时，我们要用它的内容初始化UI：


```
class TodoDetailViewController: UITableViewController {
    // ...

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        todoName.becomeFirstResponder()

        if let todoItem = todoItem {
            self.todoName.text = todoItem.name
            self.isFinished.isOn = todoItem.isFinished
        }
        else {
            todoItem = TodoItem()
        }

        print("Resource tracing: \(RxSwift.Resources.total)")
    }
}
```

第三，无论是新建还是编辑Todo，在最终提交操作的done方法里，我们都是给todoSubject发送一条onNext事件。接下来，只要回到TodoListViewController，在处理EditTodo的情况里，订阅这个事件更新UI就好了：

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // ...

    if segue.identifier == "AddTodo" {
        // ...
    }
    else if segue.identifier == "EditTodo" {
        todoDetailController.title = "Edit todo"

        if let indexPath =
            tableView.indexPath(for: sender as! UITableViewCell) {

            todoDetailController.todoItem =
                todoItems.value[indexPath.row]

            _ = todoDetailController.todo.subscribe(
                onNext: { [weak self] todo in
                    self?.todoItems.value[indexPath.row] = todo
                },
                onDisposed: {
                    print("Finish editing a todo.")
                }
            )
        }
    }
}
```

其中，最关键的，就是订阅到编辑过的Todo后，直接把它赋值给了当前正在编辑的Todo，由于`todoItems`是响应式的，因此整个`UITableView`就被自动更新了。

现在，可以回过头思考之前遗留的一个问题了，为什么在`TodoDetailViewController`中我们使用了`PublishSubject`，而不是其他的Subject呢？

这是因为其他的Subject会向事件的订阅者发送一个当前的默认值，当我们在segue中订阅事件的时候就会订阅到这个默认值。如果此时我们在新建Todo，那么就会同时创建出来两个Todo，一个是默认值，一个是用户自己添加的。这种行为，显然不是我们期望的。

What's next?

以上，就是这一节的内容，其中，最重要的内容有两点：

- 如何通过Subject在Controllers之间传递数据；
- 在Controllers之间传递数据的时候，如何正确的释放Subject资源；

在下一节，我们将对保存Todo列表的功能做一些修改，通过一个实际的例子，了解自定义Observable的应用场景。

[◀ Prev: Todo I - 通过一个真实的App体会Rx的基本概念](#)

[☰ Todo II - 如何通过Subject传递数据](#)

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

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