RxSwift UI交互 - II

Back to series (/series/reactive-programming-in-swift)

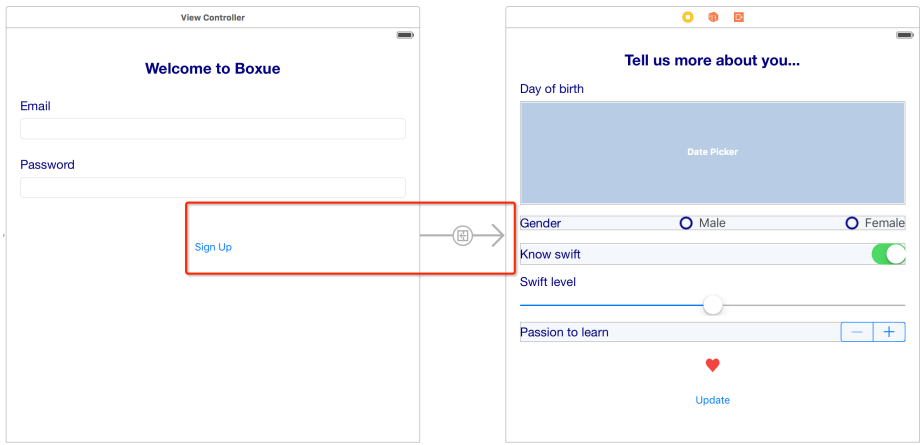
字号
字号
默认主题
金色主题
暗色主题

我们继续上个视频中的例子，使用RxSwift (https://github.com/ReactiveX/RxSwift)来处理常用的用户交互事件。大家可以在这里下载项目的初始模板 (https://github.com/Boxue/episode-samples/tree/master/RxSwift/ReactiveLogin-II/ReactiveLogin-II-Starter)。

对初始项目的改动

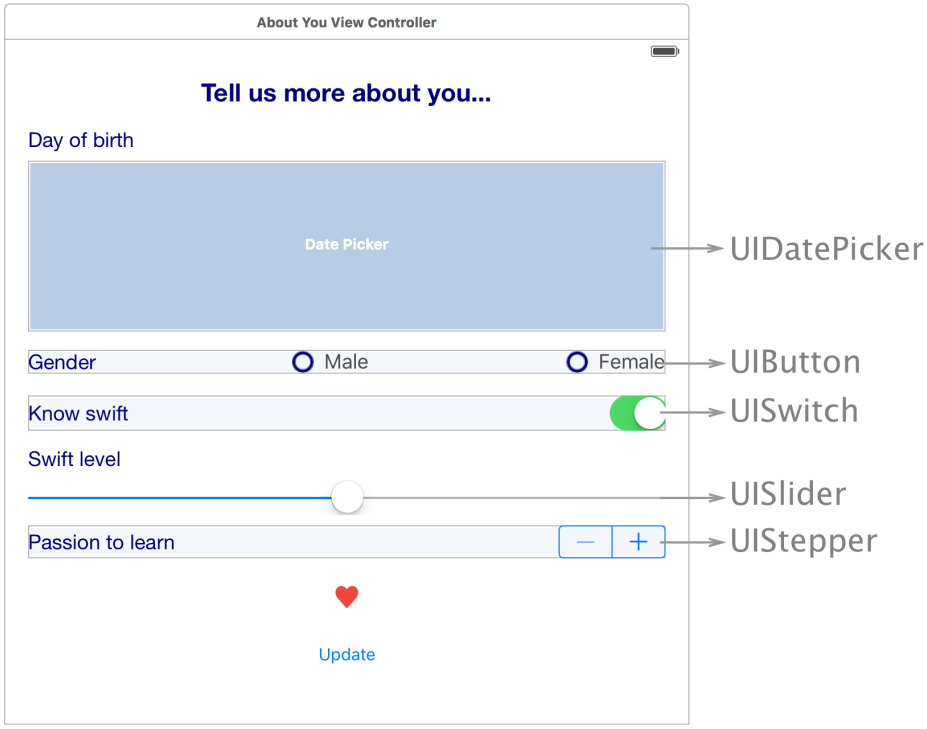
为了演示RxSwift的用法，我们对上一个视频用到的项目，做了以下改动：

首先，给Sign Up添加了一个Segue，点击后，会切换到一个用户提交各种信息的UI，我们所有要演示的交互都在这个新的UI上进行；



其次，在这个新的UI里：

- `UIDatePicker` 用于设置生日、当输入的生日小于当天时，我们会在这个picker外围显示一个绿框；
- Male和Female是两个按钮，它加载了两个 `UIImage` 用于模拟二选一的效果。只有这两个内容正确之后，我们才启用底部的update按钮，否则就禁用它；
- “Know swift”是一个 `UISwitch` ，表示用户是否了解Swift；
- 下面的 `UISlider` 则表示用户对Swift的熟悉程度；
- 接下来是一个 `UIStepper` ，用于设置对Swift的兴趣，当点击加号时，红心会变大；反之则变小；
- 最后，submit按钮只有在所有UI控件都有正确值的时候才启用，否则是禁用状态；



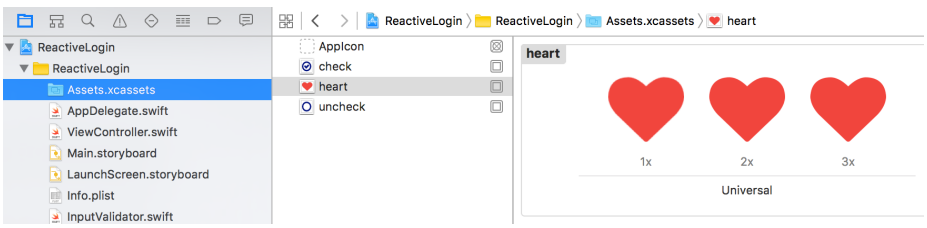
第三，我们为这个新的UI定义了一个 AboutYouViewController ，它里面有我们需要的IBOutlet以及 DisposeBag ；

```
class AboutYouViewController: UIViewController {

    @IBOutlet weak var birthday: UIDatePicker!
    @IBOutlet weak var male: UIButton!
    @IBOutlet weak var female: UIButton!
    @IBOutlet weak var knowSwift: UISwitch!
    @IBOutlet weak var swiftLevel: UISlider!
    @IBOutlet weak var passionToLearn: UIStepper!
    @IBOutlet weak var heartHeight: NSLayoutConstraint!
    @IBOutlet weak var update: UIButton!

    var bag: DisposeBag! = DisposeBag()

    // Omit for simplicity…
}
```

第四，给 InputValidator 添加了一个新的方法，用于验证 UIDatePicker 输入的日期是否小于当天；

```
class func isValidDate(date: NSDate) -> Bool {
    let calendar = NSCalendar.currentCalendar()
    let compare = calendar.compareDate(date,
            toDate: NSDate(),
            toUnitGranularity: .Day)

    return compare == .OrderedAscending
}
```

最后，我们在 InputValidator 还添加了一些需要用到的图片资源；



这就是我们对接下来内容的准备工作，了解清楚之后，我们就可以开工了。

校验UIDatePicker

首先来处理 UIDatePicker，给它添加边框的代码和 UITextField 是类似的。

RxSwift (https://github.com/ReactiveX/RxSwift)给 UIDatePicker 添加了一个扩展叫做 rx_date，我们可以直接把这个 Observable<NSDate> 映射成一个 Observable<Bool> 表示输入的生日是否合法。

在 viewDidLoad 方法里，添加下面的代码：

```
let birthdayObservable = self.birthday.rx_date.map {
    InputValidator.isValidDate($0)
}
```

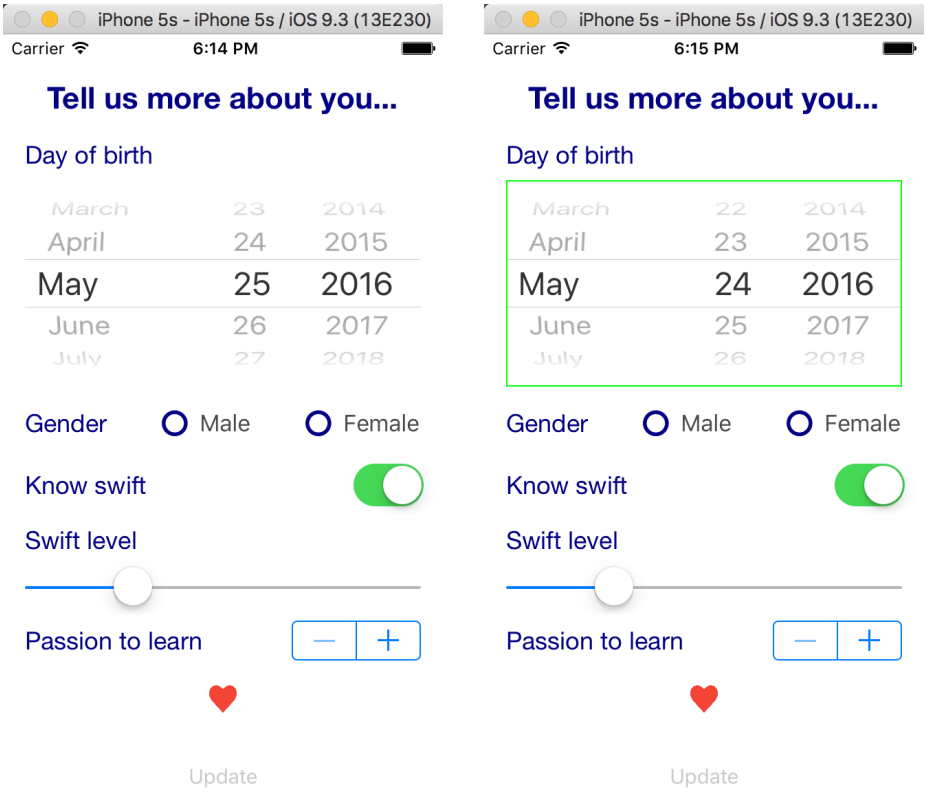
然后，把得到的结果进一步 map 成 UIColor，并且订阅它：

```
birthdayObservable.map {
    $0 ? UIColor.greenColor() : UIColor.clearColor()
}.subscribeNext {
    self.birthday.layer.borderColor = $0.CGColor
}.addDisposableTo(self.bag)
```

最后，别忘记设置 borderWidth 属性：

```
self.birthday.layer.borderWidth = 1
```

完成后， Command + R 编译执行，就可以看到结果了，只有在设置当天以前的日期时，UIDatePicker 才会有绿色的边框。



理解Rx编程中的Subject

接下来，我们来处理选择性别的按钮。由于默认情况下，没有任何一个性别被选中，因此，实际上我们要处理的逻辑有两个：

- 用户选择了一个性别，表示按钮被点击了；
- 用户具体选择的是哪个性别，我们要根据这个选择加载正确的 UIImage；

我们来逐步实现它。

首先，添加一个 enum，表示用户选择的性别：

```
enum Gender {
    case notSelected
    case male
    case female
}
```

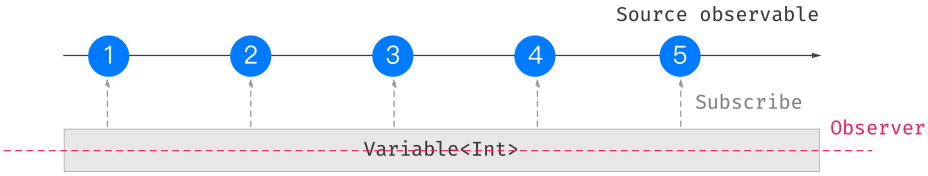
其次，我们需要一个 observer ，用来订阅按钮的点击事件，这样，我们就知道按钮被点击了，并以此作为启用udpate按钮的依据之一（这跟我们上个视频中用到的例子是相同的）。

但是，我们还需要这个 observer 是一个 Observable ，因为我们订阅它，并根据用户点击的按钮设置按钮图片，怎么做呢?

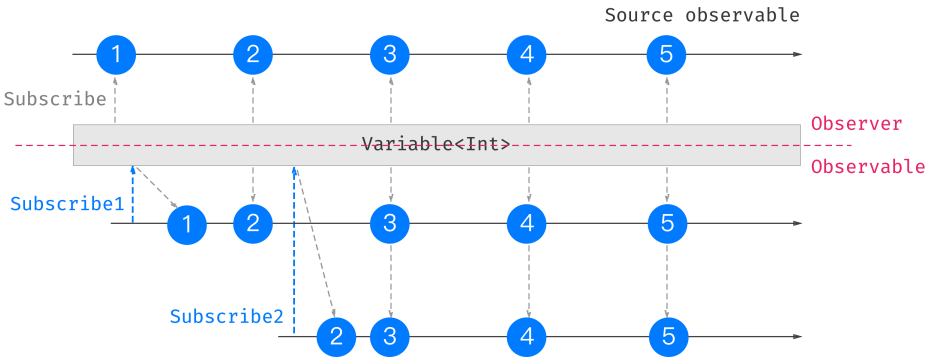
在Reactive编程里，有一个概念叫做Subject，它是一类对象的统称。这类对象既可以做 Observer ，又可以做 Observable 。大家可以在Reactive.io找到它的详细定义(http://reactivex.io/documentation/subject.html)。

在RxSwift (https://github.com/ReactiveX/RxSwift)里，我们使用其中一个叫做 Variable 的Subject，简单用图来表示，它是这样的：

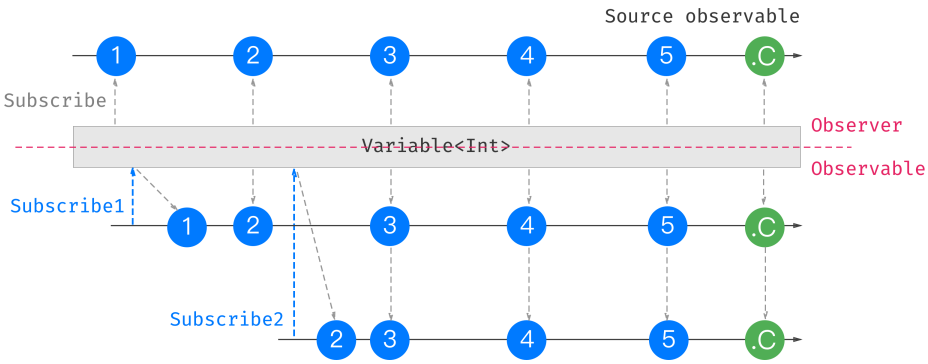
- Variable 作为Observer，它可以订阅一个Observable，我们管这个Observable叫做**source observable**；



- Variable 作为Observable，它还可以被其他的Observer订阅，每当有新订阅的时候，它就会发送最近一次发生的事件以及以后陆续会发生的事件；



- 而当**source observable**发生 .Complete 或 .Error 时， Variable 会向observer转发对应的事件，并自动被回收；



介绍了理论之后，我们来看代码。先定义一个 Variable<Gender> ：

```
let genderSelection = Variable<Gender>(.notSelected)
```

让它先去订阅按钮的点击事件：

```
self.male.rx_tap.map {
    return Gender.male
}
.bindTo(genderSelection)
.addDisposableTo(self.bag)

self.female.rx_tap.map {
    return Gender.female
}
.bindTo(genderSelection)
.addDisposableTo(self.bag)
```

这里有两点需要说明：

1. 我们使用 map 方法把点击事件变成了一个值为Gender enum 事件；
2. 使用 bindTo 订阅了 map 后的事件，在这里， bindTo 和 subscribe 是等价的，只是当我们想表达“把一个值绑定给 Variable 这样的语义时”， bindTo 比 subscribe 的表意更明确一些；

这样，当不同的按钮被点击时， genderSelection 就有不同的值了。接下来，我们要让 genderSelection 是一个observable，并根据它的值来为按钮设置图片：

```
genderSelection.asObservable().subscribeNext({
    switch $0 {
    case .male:
        self.male.setImage(UIImage(named: "check"),
            forState: .Normal)
        self.female.setImage(UIImage(named: "uncheck"),
            forState: .Normal)
    case .female:
        self.male.setImage(UIImage(named: "uncheck"),
            forState: .Normal)
        self.female.setImage(UIImage(named: "check"),
            forState: .Normal)
    default:
        break;
    }
}).addDisposableTo(self.bag)
```

在上面的代码里，我们使用 genderSelection.asObservable() 把一个 Variable 明确转换成了observable。这样，我们就能使用 subscribeNext 来订阅它了，在上面的代码里 $0 的值是我们之前定义的 Gender enum ，我们只要根据用户点击的选择，为按钮设置正确的图片就好了。

最后，还有一个工作。当用户设置了生日和性别之后，UI上所有控件的值就都有合法值了，我们添加启用Submit按钮的代码，这和我们在上个视频中的代码是类似的。

我们先把 genderSelection 变成一个 Observable<Bool>：

```
let genderBtnObservable = genderSelection.asObservable().map {
    return $0 != .notSelected ? true : false
}
```

然后，使用 combineLatest 方法，把 birthdayObservable 和 genderBtnObservable 合并起来，再变成一个 Observable<Bool>：

```
Observable.combineLatest(birthdayObservable, genderBtnObservable) {
    return [$0, $1]
}.map {
    $0.reduce(true, combine: { $0 && $1 })
}
```

然后，订阅这个合并的结果，把它和update按钮的 rx_tap ”绑定”起来：

```
Observable.combineLatest(birthdayObservable, genderBtnObservable) {
    return [$0, $1]
}.map {
    $0.reduce(true, combine: { $0 && $1 })
}
.bindTo(self.update.rx_enabled)
.addDisposableTo(self.bag)
```

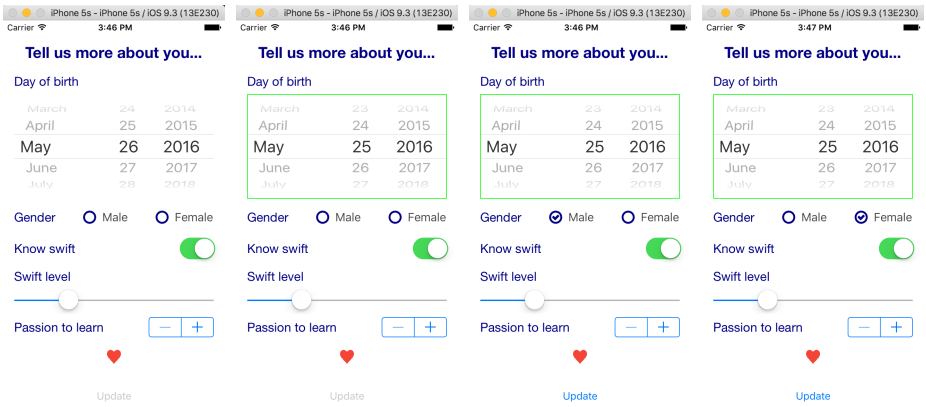
这里有两点要说明：

1. 我们再一次使用了 bindTo 代替了 subscribe 用于表达“绑定”的语义；
2. rx_enabled 是RxSwift (https://github.com/ReactiveX/RxSwift)给 UIButton 添加的另外一个扩展，表示按钮是否启用；

至此，我们就实现了两个功能：

1. 模拟了二选一的按钮效果；
2. 当UI上所有控件都有合法值时，启用update按钮；

按 Command + R 编译执行，我们就能看到对应的效果了：



Next?

这就是我们这段视频的全部内容，其中最重要的，就是要理解 Variable subject的用法。在下一段视频中，我们将完成这个UI。包括：

- ControlProperty 的双向绑定；
- 使用RxSwift (https://github.com/ReactiveX/RxSwift)实现 UIStepper 交互；

RxSwift UI交互 - I
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/77)

RxSwift UI交互 - III
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/79)