

☰ Reactive Programming in Swift

⏪ 理解Disposable & DisposeBag

RxSwift UI交互 - II ⏩

(<https://www.boxueio.com/series/reactive-programming-in-swift/ebook/76>)

(<https://www.boxueio.com/series/reactive-programming-in-swift/ebook/78>)

RxSwift UI交互 - I

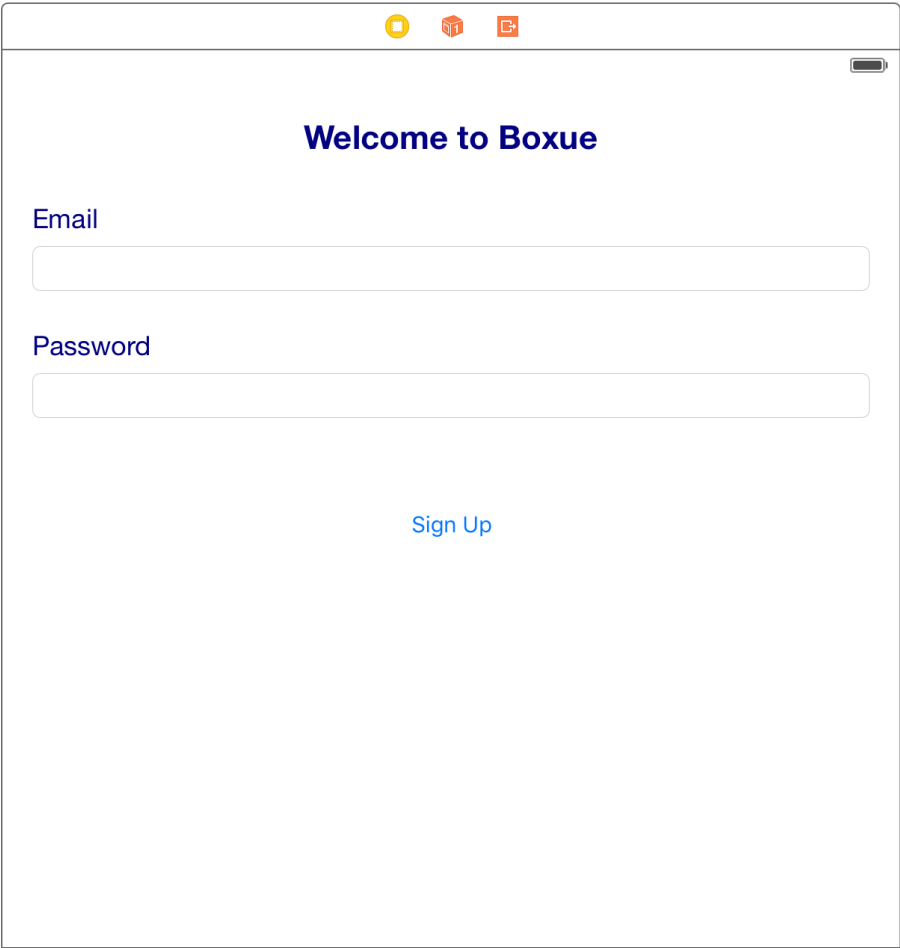
[⏪ Back to series \(/series/reactive-programming-in-swift\)](#)

了解了RxSwift (<https://github.com/ReactiveX/RxSwift>)的基本概念和用法之后，我们通过一系列视频向大家介绍如何用RxSwift (<https://github.com/ReactiveX/RxSwift>)处理UI交互。在这个例子里，我们实现一个简单的登录UI，对比传统的delegate方式，我们将看到RxSwift (<https://github.com/ReactiveX/RxSwift>)在处理异步事件时的简洁和便利。

准备工作

下载项目初始模板。(<https://github.com/Boxue/episode-samples/tree/master/RxSwift/ReactiveLogin-I/ReactiveLoginStarter>)

首先，我们创建了一个Single View Application，并且安装好了RxSwift (<https://github.com/ReactiveX/RxSwift>)。在Main.storyboard里，我们添加两个 UITextField 用于输入邮箱和密码，以及一个 UIButton 表示注册。



⊕ 字号

● 字号

🖌️ 默认主题

🖌️ 金色主题

🖌️ 暗色主题

并且，我们在ViewController里，添加了对应的IBOutlet，以及一个用于回收 Disposable 对象的 DisposeBag：

```
class ViewController: UIViewController {

    @IBOutlet weak var email: UITextField!
    @IBOutlet weak var password: UITextField!
    @IBOutlet weak var register: UIButton!

    var bag: DisposeBag! = DisposeBag()

}
```

除此之外，我们还添加了一个辅助类 InputValidator，它有两个类方法：

- isValidEmail(email: String) 用于验证email是否是一个合法的电子邮件：

```
class func isValidEmail(email: String) -> Bool {
    let re = try? NSRegularExpression(
        pattern: "^\\S+@\\S+\\.\\S+$",
        options: .CaseInsensitive)

    if let re = re {
        let range = NSMakeRange(0,
            email.lengthOfBytesUsingEncoding(
                NSUTF8StringEncoding))

        let result = re.matchesInString(email,
            options: .ReportProgress,
            range: range)

        return result.count > 0
    }

    return false
}
```

- isValidPassword(password: String) 用于验证密码的长度是否大于等于8；

```
class func isValidPassword(
    password: String) -> Bool {
    return password.characters.count >= 8
}
```

至此，所有的准备工作就结束了，接下来，我们来处理用户交互。

让输入框内容合法时变成绿色

先来处理Email的输入。

之前我们也提到过，RxSwift给 UITextField 添加了一个扩展 rx_text，表示输入事件序列，而事件的值，是每一次输入后，UITextField 中的字符串。因此，我们可以先使用 map 把字符串变成一个 Bool，表示当前 UITextField 中的值是否是一个合法的电子邮件。

在 viewDidLoad 方法里，添加下面的代码：

```
let emailObservable =
    self.email.rx_text.map {
        (input: String) -> Bool in
        return InputValidator.isValidEmail(input)
    }
```

然后，我们希望当内容为合法的Email时，给 UITextField 添加一个绿色的边框，因此，我们还要进一步把 Observable<Bool> 变成一个 Observable<UIColor>：

```
emailObservable.map { (valid: Bool) -> UIColor in
    let color = valid ?
        UIColor.greenColor() : UIColor.clearColor()

    return color
}
```

这样，我们就可以使用 subscribeNext 订阅这个事件了：

```
emailObservable.map { (valid: Bool) -> UIColor in
    let color = valid ?
        UIColor.greenColor() : UIColor.clearColor()

    return color
}.subscribeNext({
    self.email.layer.borderColor = $0.CGColor
}).addDisposableTo(self.bag)
```

这反而是最简单的一步，我们直接把Next的associated value赋值给self.email.layer.borderColor 属性就可以了。

最后，为了能看到这个绿色的边框，我们在 viewDidLoad 开始要设置一下边框的宽度：

```
self.email.layer.borderWidth = 1
```

这样，按 Command + R 编译执行，当我们输入一个完整的email后，就可以看到Email输入框变为绿色了：



Welcome to Boxue

Email

Password

Sign Up

接下来，我们可以用同样的方式处理password输入框，过程就不细说了，只是贴上代码：

```
let passwordObservable =
    self.password.rx_text.map {
        (input: String) -> Bool in
            return InputValidator.isValidPassword(input)
    }

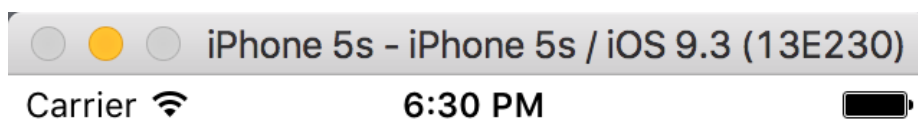
passwordObservable.map {
    (valid: Bool) -> UIColor in
        let color = valid ?
            UIColor.greenColor() : UIColor.clearColor()

        return color
    }.subscribeNext({
        self.password.layer.borderColor = $0.CGColor
    }).addDisposableTo(self.bag)
```

最后，不要忘记在 viewDidLoad 开始，也设置password输入框边框的宽度：

```
self.password.layer.borderWidth = 1
```

然后，Command + R 编译执行，当我们在密码框中输入长度大于等于8的密码后，password输入框就变成绿色了：



Welcome to Boxue

Email

Password

Sign Up

这就是 UITextField 在 RxSwift 中的用法，简单来说，就是利用输入的字符串，把 `rx_text` 变换成我们需要的事件逻辑，然后订阅对应的事件进行操作就可以了。

接下来，我们要实现另外一个效果：我们希望只有当 Email 和 Password 中的输入都合法时，才启用 Sign Up 按钮，否则禁用它，该怎么做呢？

禁用和启用UIButton

在我们的例子里，emailObservable 和 passwordObservable 是两个独立的事件序列。如果我们要表达“它们的输入都合法”这样的语义，也就是说，**这两个事件序列中最新的事件值都是合法的**。

为此，RxSwift (<https://github.com/ReactiveX/RxSwift>)提供了一个专门的operator，叫做 combineLatest，它用于将多个事件序列中最新的事件进行合并。我们可以在这里 (<http://reactivex.io/documentation/operators/combineatest.html>)找到 combineLatest 的详细定义。

至此，我们就有思路了。只要将 emailObservable 和 passwordObservable 中最新的事件进行合并，如果它们都是 true，就启用Sign Up按钮，否则就禁用。

有了思路之后，就可以开工了。继续在 viewDidLoad 里，添加下面的代码：

```
Observable.combineLatest(
    emailObservable, passwordObservable) {
    (validEmail: Bool, validPassword: Bool) -> [Bool] in
    return [validEmail, validPassword]
}
```

combineLatest 的前两个参数表示要合并的事件序列，第三个参数是一个closure，表示合并的方法，在我们的例子里，我们把emailObservable和passwordObservable中的两个最新事件的Bool，变成了一个 Bool 数组。

接下来，我们使用 map 把合并的结果变成一个单一的 Observable<Bool>：

```
Observable.combineLatest(
    emailObservable, passwordObservable) {
    (validEmail: Bool, validPassword: Bool) -> [Bool] in
    return [validEmail, validPassword]
}
.map { (input: [Bool]) -> Bool in
    let validValues = input.reduce(true,
        combine: { $0 && $1 })

    return validValues
}
```

这样，我们就得到了一个 Observable<Bool>，我们订阅它，然后设置按钮的状态就可以了：

```
Observable.combineLatest(
    emailObservable, passwordObservable) {
    (validEmail: Bool, validPassword: Bool) -> [Bool] in
    return [validEmail, validPassword]
}
.map { (input: [Bool]) -> Bool in
    let validValues = input.reduce(true,
        combine: { $0 && $1 })

    return validValues
}
.subscribeNext { (isEnabled: Bool) in
    self.register.enabled = isEnabled
}.addDisposableTo(self.bag)
```

然后，按Command + R编译执行，就可以看到只有当Email和Password都输入正确后，Sign Up按钮才会被启用的效果了。