响应式编程

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

小码哥教育 SEEMYGO

实力IT教育 www.520it.com

码拉松

小码哥教育 SEEMYGO

响应式编程

- 响应式编程（Reactive Programming，简称RP）
  - 也是一种编程范式，于1997年提出，可以简化异步编程，提供更优雅的数据绑定
  - 一般与函数式融合在一起，所以也会叫做：函数响应式编程（Functional Reactive Programming，简称FRP）

- 比较著名的、成熟的响应式框架
  - ReactiveCocoa
    - ✓ 简称RAC，有Objective-C、Swift版本
    - ✓ 官网： http://reactivecocoa.io/
    - ✓ github：https://github.com/ReactiveCocoa

  - ReactiveX
    - ✓ 简称Rx，有众多编程语言的版本，比如RxJava、RxKotlin、RxJS、RxCpp、RxPHP、RxGo、RxSwift等等
    - ✓ 官网： http://reactivex.io/
    - ✓ github： https://github.com/ReactiveX

RxSwift

- RxSwift（ReactiveX for Swift），ReactiveX的Swift版本
- 源码：https://github.com/ReactiveX/RxSwift
- 中文文档： https://beeth0ven.github.io/RxSwift-Chinese-Documentation/

- RxSwift的github上已经有详细的安装教程，这里只演示CocoaPods方式的安装

① Podfile

```
use_frameworks!

target 'target_name' do
  pod 'RxSwift', '~> 5'
  pod 'RxCocoa', '~> 5'
end
```

② 命令行

- pod repo update
- pod install

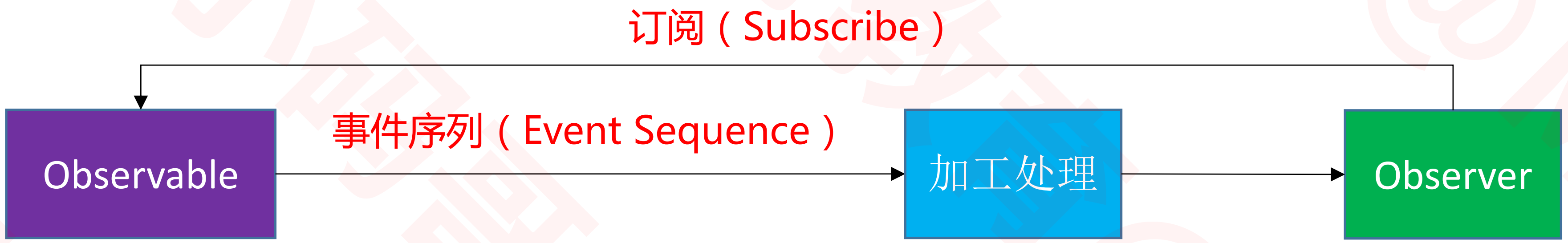
③ 导入模块

```
import RxSwift
import RxCocoa
```

- 模块说明
- RxSwift：Rx标准API的Swift实现，不包括任何iOS相关的内容
- RxCocoa：基于RxSwift，给iOS UI控件扩展了很多Rx特性

RxSwift的核心角色

- Observable：负责发送事件（Event）
- Observer：负责订阅Observable，监听Observable发送的事件（Event）



```
public enum Event<Element> {
    /// Next element is produced.
    case next(Element)

    /// Sequence terminated with an error.
    case error(Swift.Error)

    /// Sequence completed successfully.
    case completed
}
```

- Event有3种
- next：携带具体数据
- error：携带错误信息，表明Observable终止，不会再发出事件
- completed：表明Observable终止，不会再发出事件

小码哥教育 SEEMYGO

创建、订阅Observable1

```
var observable = Observable<Int>.create { observer in
    observer.onNext(1)
    observer.onCompleted()
    return Disposables.create()
}
// 等价于
observable = Observable.just(1)
observable = Observable.of(1)
observable = Observable.from([1])
```

```
var observable = Observable<Int>.create { observer in
    observer.onNext(1)
    observer.onNext(2)
    observer.onNext(3)
    observer.onCompleted()
    return Disposables.create()
}
// 等价于
observable = Observable.of(1, 2, 3)
observable = Observable.from([1, 2, 3])
```

```
observable.subscribe { event in
    print(event)
}.dispose()
```

```
observable.subscribe(onNext: {
    print("next", $0)
}, onError: {
    print("error", $0)
}, onCompleted: {
    print("completed")
}, onDisposed: {
    print("dispose")
}).dispose()
```

小码哥教育 SEEMYGO

创建、订阅Observable2

```
let observable = Observable<Int>.timer(.seconds(3),
                                       period: .seconds(1),
                                       scheduler: MainScheduler.instance)
observable.map { "数值是\($0)" }
    .bind(to: label.rx.text)
    .disposed(by: bag)
```

小码哥教育 SEEMYGO

创建Observer

```
let observer = AnyObserver<Int>.init { event in
    switch event {
    case .next(let data):
        print(data)
    case .completed:
        print("completed")
    case .error(let error):
        print("error", error)
    }
}
Observable.just(1).subscribe(observer).dispose()
```

```
let binder = Binder<String>(label) { label, text in
    label.text = text
}
Observable.just(1).map { "数值是\($0)" }.subscribe(binder).dispose()
Observable.just(1).map { "数值是\($0)" }.bind(to: binder).dispose()
```

扩展Binder属性

```
extension Reactive where Base: UIView {
    var hidden: Binder<Bool> {
        Binder<Bool>(base) { view, value in
            view.isHidden = value
        }
    }
}
```

```
let observable = Observable<Int>.interval(.seconds(1),
                                          scheduler: MainScheduler.instance)
observable.map { $0 % 2 == 0 }.bind(to: button.rx.hidden).disposed(by: bag)
```

传统的状态监听

- 在开发中经常要对各种状态进行监听，传统的常见监听方案有
  - KVO
  - Target-Action
  - Notification
  - Delegate
  - Block Callback

- 传统方案经常会出现错综复杂的依赖关系、耦合性较高，还需要编写重复的非业务代码

小码哥教育 SEEMYGO

RxSwift的状态监听1

```swift
button.rx.tap.subscribe(onNext: {
    print("按钮被点击了1")
}).disposed(by: bag)
```

```swift
let data = Observable.just([
    Person(name: "Jack", age: 10),
    Person(name: "Rose", age: 20)
])
data.bind(to: tableView.rx.items(cellIdentifier: "cell")) { row, person, cell in
    cell.textLabel?.text = person.name
    cell.detailTextLabel?.text = "\(person.age)"
}.disposed(by: bag)

tableView.rx.modelSelected(Person.self)
    .subscribe(onNext: { person in
        print("点击了", person.name)
    }).disposed(by: bag)
```

小码哥教育 SEEMYGO

RxSwift的状态监听2

```swift
class Dog: NSObject {
    @objc dynamic var name: String?
}
dog.rx.observe(String.self, "name")
    .subscribe(onNext: { name in
        print("name is", name ?? "nil")
    }).disposed(by: bag)
dog.name = "larry"
dog.name = "wangwang"
```

```swift
NotificationCenter.default.rx
    .notification(UIApplication.didEnterBackgroundNotification)
    .subscribe(onNext: { notification in
        print("APP进入后台", notification)
    }).disposed(by: bag)
```

既是Observable，又是Observer

```
Observable.just(0.8).bind(to: slider.rx.value).dispose()
```

```
slider.rx.value.map {
    "当前数值是：\($0)"
}.bind(to: textField.rx.text).disposed(by: bag)
```

```
textField.rx.text
    .subscribe(onNext: { text in
        print("text is", text ?? "nil")
    }).disposed(by: bag)
```

- 诸如UISlider.rx.value、UTextField.rx.text这类属性值，既是Observable，又是Observer
  - 它们是RxCocoa.ControlProperty类型

Disposable

- 每当Observable被订阅时，都会返回一个Disposable实例，当调用Disposable的dispose，就相当于取消订阅
- 在不需要再接收事件时，建议取消订阅，释放资源。有3种常见方式取消订阅

```swift
// 立即取消订阅（一次性订阅）
observable.subscribe { event in
    print(event)
}.dispose()
```

```swift
// 当bag销毁（deinit）时，会自动调用Disposable实例的dispose
observable.subscribe { event in
    print(event)
}.disposed(by: bag)
```

```swift
// self销毁时（deinit）时，会自动调用Disposable实例的dispose
let _ = observable.takeUntil(self.rx.deallocated).subscribe { event in
    print(event)
}
```