Advanced RxSwift – Day 4

Younghwan Kim

RxSwift Basics

- Day 1 – Observable, Operator (Filter, Transform, Combine)
- Day 2 – Subject (flatMap, flatMapFirst, flatMapLatest)
- Day 3 – Two VCs communications with Subject, RxCocoa (Button)
- Day 4 – Sequential, Merged Observable Calls
- Day 5 – RxCocoa, UI Binding (Button, TextField, Label, TableView)

Advanced RxSwift

- Day 1 – Protocol-Oriented Programming, Protocol Extension, Associatetype
- Day 2 – Network Call, Generic Enum
- Day 3 – Binding Track Activity (show / hide ‘Loading’)
- **Day 4 – Adding a reactive extension to Custom UI Element,**
- **2 Way Binding, Advanced TableView – RxDataSources**

- Day 5 – Schedulers (observeOn, subscribeOn),

  Unit Test (RxTest, RxBlocking)

Binding

```
OneObservable
    .bind(to: TwoObservable)
    .disposed(by disposeBag)
```

Binding

```
cell.textValue.asObservable()
        .bind(to: self.userInputLabel.rx.text)
        .disposed(by: cell.disposeBag)
```

```
cell.textValue.asObservable()
         .subscribe(onNext: { input in
               self.userInputLabel.text = input
          })
          .disposed(by: cell.disposeBag)
```

```
cell.textValue.asDriver()
        .drive(self.userInputLabel.rx.text)
        .disposed(by: cell.disposeBag)
```

```
cell.textValue.asDriver()
         .drive(onNext: { input in
              self.userInputLabel.text = input
           })
           .disposed(by: cell.disposeBag)
```

2 Way Binding

Custom Implementation

https://github.com/ReactiveX/RxSwift/blob/master/RxExample/RxExample/Operators.swift

```
func <-> <T>(property: ControlProperty<T>, variable: BehaviorRelay<T>) -> Disposable {

    let bindToUIDisposable = variable.asObservable()
        .bind(to: property)
    let bindToVariable = property
        .subscribe(onNext: { n in
            variable.accept(n)
        }, onCompleted:  {
            bindToUIDisposable.dispose()
        })

    return Disposables.create(bindToUIDisposable, bindToVariable)
}
```

2 Way Binding

Custom Implementation

```
@IBOutlet weak var textField: UITextField!
let textValue = BehaviorRelay(value: "")
var disposeBag = DisposeBag()


override func awakeFromNib() {
   super.awakeFromNib()
   // Initialization code

   let textDisposable = textField.rx.textInput <-> textValue
   textDisposable.disposed(by: self.disposeBag)
}
```

Adding a reactive extension to Custom UI Element

UILabel

```
myObservable
.map { "new value is \($0)" }
.bind(to: myLabel.rx.text )
.disposed(by: bag)
```

```
 extension Reactive where Base: UILabel {

    /// Bindable sink for `text` property.
    public var text: Binder<String?> {
        return Binder(self.base) { label, text in
            label.text = text
        }
    }
}
```

Adding a reactive extension to Custom UI Element

SwiftSpinner

```
Observable<Int>.timer(0.0, period: 0.15, scheduler: MainScheduler.instance)
    .bind(to: SwiftSpinner.sharedInstance.rx.progress )
    .disposed(by: bag)
```

```
extension Reactive where Base: SwiftSpinner {
    public var progress: Binder<Int> {
        return Binder(self.base) { spinner, progress in
            let progress = max(0, min(progress, 100))
            SwiftSpinner.show(progress: Double(progress)/100.0, title: "\(progress)% completed")
        }
    }
}
```

RxDataSources

Using **RxDataSources** requires more work to learn its idioms, but offers more powerful, advanced features. Instead of a simple array of data, it requires you to provide contents using objects which conform to the **SectionModelType** protocol.

Each section itself contains the actual objects. For sections with multiple object types, use the enum technique shown above to differentiate the types.

https://github.com/RxSwiftCommunity/RxDataSources

Lab