報道各位

2018 年 12 月 27 日
株式会社インプレスR&D
https://nextpublishing.jp/

一歩先に進みたい iOS アプリ開発者のための RxSwift 解説書！

『比較して学ぶ RxSwift 入門』発行

技術書典シリーズ、12月の新刊

インプレスグループで電子出版事業を手がける株式会社インプレス R&D は、『比較して学ぶ RxSwift 入門』(著者:高橋 凌)を発行いたします。

『比較して学ぶRxSwift入門』
https://nextpublishing.jp/isbn/9784844398790

著者:高橋 凌
小売希望価格:電子書籍版 1600 円(税別)／印刷書籍版 1800 円(税別)
電子書籍版フォーマット:EPUB3／Kindle Format8
印刷書籍版仕様:B5 判／カラー／本文 86 ページ
ISBN:978-4-8443-9879-0
発行:インプレス R&D

＜＜発行主旨・内容紹介＞＞

【一歩先に進みたい iOS アプリ開発者必見！】

本書は、あるテーマに沿って Delegate、CallBack、KVO、RxSwift/RxCocoa、それぞれのパターンでコードを実装しそれぞれの実装を比較して RxSwift の書き方について学びます。

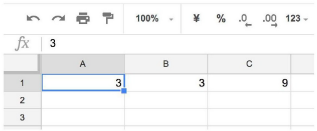
RxSwift の動向をキャッチアップし、一歩先に進みたい iOS アプリ開発初心者のための一冊です。

〈本書の対象読者〉

・Swift による iOS アプリの開発経験が少しだけある(3 ヶ月から 1 年未満)
・RxSwift ライブラリーを使った開発をしたことがない、またはほんの少しだけある
(本書は、次世代出版メソッド「NextPublishing」を使用し、出版されています。)

RxSwift の特徴を丁寧に解説

図 1.2: A1 セルを変更



A1 の値の変更に合わせて、C1 が自動で再計算されました。概念的には、「値と振る舞いの関係を定義する」と考えると理解しやすいかも知れません。

1.6 RxSwiftの特徴

RxSwift の主な特徴として「値の変化が検知しやすい」「非同期処理を簡潔に書ける」が挙げられます。

この特徴は UI の変更の検知（タップや文字入力）や通信処理等の際、RxSwift を用いることで delegate や callback を用いたコードよりもスッキリと見やすく書けるようになります。

その他のメリットとしては次のものが挙げられます。

・時間経過に関する処理をシンプルに書ける
・コード全体が一貫する
・まとまった流れが見やすい
・差分がわかりやすい
・処理スレッドを変えやすい
・callback を減らせる
　—インデントの浅いコードにできる

デメリットとしては、主に「学習コストが高い」「デバッグしにくい」が挙げられます。

プロジェクトメンバーのほとんどがRxSwift の扱いにあまり長けていない状況で、とりあえずこれを導入すれば開発速度が早くなるんでしょ？といった考え方で安易に導入すると、逆に開発速度が落ちる可能性があります。

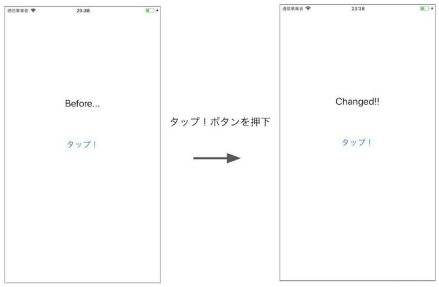
その他のデメリットとして、簡単な処理で使うと長くなりがちという点もあります。プロジェクトによってRxSwift の有用性が変わるので、そのプロジェクトの特性とRxSwift のメリット・デメリットを照らし合わせた上で検討しましょう。

1.7 RxSwiftは何が解決できる？

わかりやすいのは「アプリのライフサイクルと、UI の delegate や IBAction などの処理を定義している部分が離れている」という問題の解決です。実際にコードを書いて見てみましょう。

UIButton と UILabel が画面に配置されていて、ボタンをタップすると文字列が変更される、という仕様のアプリを題材として作ります。

図 1.3: サンプル画面



まずは従来の IBAction を使った方法で作ってみましょう。

リスト 1.3: IBAction を用いたコード

```
1: class SimpleTapViewController: UIViewController {
2:
3:   @IBOutlet weak var messageLabel: UILabel!
4:
5:   @IBAction func tapLoginButton(_ sender: Any) {
6:       messageLabel.text = "Tap Login Button!"
7:   }
8:
9: }
```

通常の書き方では、ひとつのボタンに対してひとつの関数を定義します。図に表してみましょう。

RxSwift のコードの書き方や仕組みを紹介

第3章　RxSwiftの基本的な書き方

本章では、RxSwift の基本的な書き方や仕組みについて解説していきます。RxSwift を支える全ての仕組みを解説することは本書のテーマから逸れてしまうので、良く使われるところを抜粋して解説します。

3.1 メソッドチェーンのように直感的に書ける

RxSwift／RxCocoa は、メソッドチェーンのように直感的にコードを書くことができます。メソッドチェーンとは、その名前のとおりメソッドを実行し、その結果に対してさらにメソッドを実行するような書き方を指します。jQuery を扱った経験がある人なら、なんとなく分かるのではないでしょうか？

たとえば、次のように書くことができます

リスト 3.1: loginButton のイベント購読

```
1: loginButton.rx.tap
2:     .subscribe(onNext: { [weak self] in
3:         /* 処理 */
4:     })
5:     .disposed(by: disposeBag)
```

それぞれの戻り値の型をコメントで表現してみます。

リスト 3.2: loginButton のイベント購読

```
1: loginButton
2:     .rx     // Reactive<UIButton>
3:     .tap    // ControlEvent<Void>
4:     .subscribe(onNext: { /* 処理 */ }) // Disposable
5:     .disposed(by: disposeBag) // Void
```

loginButton のタップイベントを購読し、タップされたときに subscribe メソッドの引数である onNext のクロージャ内の処理を実行します。

最後にクラスが解放されたとき、自動的に購読が破棄されるように `disposed(by:)` メソッドを使っています。（仕組みは後述します）

まとめると、次の順で処理を定義しています。

1．ストリームの購読
2．ストリームにイベントが流れてきた時にどうするかを定義
3．クラスが破棄されると同時に購読を破棄させるように設定

3.2 Hello World

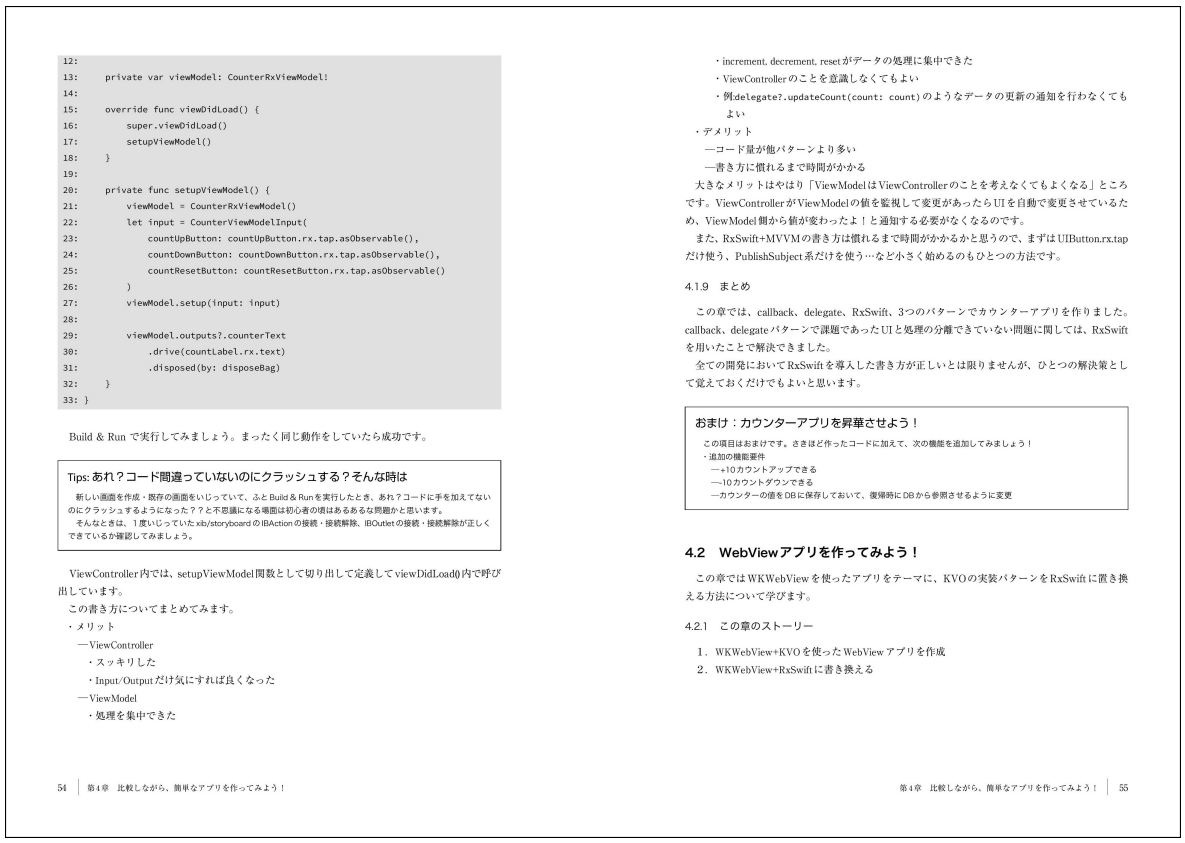
RxSwift での HelloWorld 的なものを書いてみます。

リスト 3.3: subject-example

```
1: import UIKit
2: import RxSwift
3: import RxCocoa
4:
5: class ViewController: UIViewController {
6:
7:     private let disposeBag = DisposeBag()
8:
9:     override func viewDidLoad() {
10:         super.viewDidLoad()
11:
12:         let helloWorldSubject = PublishSubject<String>()
13:
14:         helloWorldSubject
15:             .subscribe(onNext: { message in
16:                 print("onNext: \(message)")
17:             }, onCompleted: {
18:                 print("onCompleted")
19:             }, onDisposed: {
20:                 print("onDisposed")
21:             })
22:             .disposed(by: disposeBag)
23:
24:         helloWorldSubject.onNext("Hello World!")
25:         helloWorldSubject.onNext("Hello World!!")
26:         helloWorldSubject.onNext("Hello World!!!")
27:         helloWorldSubject.onCompleted()
28:     }
29: }
```

実行結果です。

コラムや Tips を交えながら、RxSwift でのコーディングを学習



```
12:
13:     private var viewModel: CounterRxViewModel!
14:
15:     override func viewDidLoad() {
16:         super.viewDidLoad()
17:         setupViewModel()
18:     }
19:
20:     private func setupViewModel() {
21:         viewModel = CounterRxViewModel()
22:         let input = CounterViewModelInput(
23:             countUpButton: countUpButton.rx.tap.asObservable(),
24:             countDownButton: countDownButton.rx.tap.asObservable(),
25:             countResetButton: countResetButton.rx.tap.asObservable()
26:         )
27:         viewModel.setup(input: input)
28:
29:         viewModel.outputs?.counterText
30:             .drive(countLabel.rx.text)
31:             .disposed(by: disposeBag)
32:     }
33: }
```

Build & Run で実行してみましょう。まったく同じ動作をしていたら成功です。

Tips: あれ？コード間違っていないのにクラッシュする？そんな時は

新しい画面を作成・既存の画面をいじっていて、ふと Build & Run を実行したとき、あれ？コードに手を加えてないのにクラッシュするようになった？？と不思議になる場面は初心者の頃はあるある な問題かと思います。

そんなときは、1度いじっていた xib/storyboard の IBAction の接続・接続解除、IBOutlet の接続・接続解除が正しくできているか確認してみましょう。

ViewController 内では、setupViewModel 関数として切り出して定義して viewDidLoad() 内で呼び出しています。

この書き方についてまとめてみます。

・メリット
　―ViewController
　　・スッキリした
　　・Input/Output だけ気にすれば良くなった
　―ViewModel
　　・処理を集中できた
　　・increment, decrement, reset がデータの処理に集中できた
　　・ViewController のことを意識しなくてもよい
　　・例 `delegate?.updateCount(count: count)` のようなデータの更新の通知を行わなくてもよい
・デメリット
　―コード量が他パターンより多い
　―書き方に慣れるまで時間がかかる

大きなメリットはやはり「ViewModel は ViewController のことを考えなくてもよくなる」ところです。ViewController が ViewModel の値を監視して変更があったら UI を自動で変更させているため、ViewModel 側から値が変わったよ！と通知する必要がなくなるのです。

また、RxSwift+MVVM の書き方は慣れるまで時間がかかるかと思うので、まずは UIButton.rx.tap だけ使う、PublishSubject 系だけを使う…など小さく始めるのもひとつの方法です。

4.1.9　まとめ

この章では、callback、delegate、RxSwift、3つのパターンでカウンターアプリを作りました。callback、delegate パターンで課題であった UI と処理の分離できていない問題に関しては、RxSwift を用いたことで解決できました。

全ての開発において RxSwift を導入した書き方が正しいとは限りませんが、ひとつの解決策として覚えておくだけでもよいと思います。

おまけ：カウンターアプリを昇華させよう！

この項目はおまけです。さきほど作ったコードに加えて、次の機能を追加してみましょう！

・追加の機能要件
　―+10 カウントアップできる
　―-10 カウントダウンできる
　―カウンターの値を DB に保存しておいて、復帰時に DB から参照させるように変更

4.2　WebViewアプリを作ってみよう！

この章では WKWebView を使ったアプリをテーマに、KVO の実装パターンを RxSwift に置き換える方法について学びます。

4.2.1　この章のストーリー

1．WKWebView+KVO を使った WebView アプリを作成
2．WKWebView+RxSwift に書き換える

＜＜目次＞＞

＜＜著者紹介＞＞

髙橋 凌(たかはし りょう)

情報系専門学校を2017年に卒業、同年入社した受託開発会社を経て、2018年に株式会社トクバイに入社。以来、破壊的イノベーションで小売業界を変革するため iOS アプリエンジニアとして従事。とにかくアプリを作ることが好きで学生時代からWeb・モバイル問わず多種多様なアプリを作り、モバイルアプリでは複数のアプリをリリース。最近はサービス開発以外にも、技術同人誌の執筆や勉強会の主催を行うなど、これまでにやったことのなかった領域に手を伸ばし、視野を広げようと活動している。

＜＜販売ストア＞＞

電子書籍:

Amazon Kindle ストア、楽天 kobo イーブックストア、Apple Books、紀伊國屋書店 Kinoppy、Google Play Store、honto 電子書籍ストア、Sony Reader Store、BookLive!、BOOK☆WALKER

印刷書籍:

Amazon.co.jp、三省堂書店オンデマンド、honto ネットストア、楽天ブックス

※ 各ストアでの販売は準備が整いしだい開始されます。

※ 全国の一般書店からもご注文いただけます。

【株式会社インプレス R&D】 https://nextpublishing.jp/

株式会社インプレス R&D（本社：東京都千代田区、代表取締役社長：井芹昌信）は、デジタルファーストの次世代型電子出版プラットフォーム「NextPublishing」を運営する企業です。また自らも、NextPublishing を使った「インターネット白書」の出版など IT 関連メディア事業を展開しています。

※NextPublishing は、インプレス R&D が開発した電子出版プラットフォーム（またはメソッド）の名称です。電子書籍と印刷書籍の同時制作、プリント・オンデマンド（POD）による品切れ解消などの伝統的出版の課題を解決しています。これにより、伝統的出版では経済的に困難な多品種少部数の出版を可能にし、優秀な個人や組織が持つ多様な知の流通を目指しています。

【インプレスグループ】 https://www.impressholdings.com/

株式会社インプレスホールディングス（本社：東京都千代田区、代表取締役：唐島夏生、証券コード：東証1部 9479）を持株会社とするメディアグループ。「IT」「音楽」「デザイン」「山岳・自然」「旅・鉄道」「学術・理工学」を主要テーマに専門性の高いメデイア&サービスおよびソリューション事業を展開しています。さらに、コンテンツビジネスのプラットフォーム開発・運営も手がけています。

【お問い合わせ先】

株式会社インプレス R&D NextPublishing センター
TEL 03-6837-4820
電子メール: np-info@impress.co.jp