Storyboardを使わずに
コードだけでアプリを作る本
(Swift 5.0)

@takoikatakotako 著

2019/4/14 TeamSommelier 発行

まえがき

「Storyboardを使わずにコードだけでアプリを作る本」をお手に取っていただきありがとうございます。本書では、Storyboardを使わずにSwiftのコードだけでiOSアプリを作る方法、Tips、サンプルコードを紹介します。本書を作成するにあたって検証に使用したソースコードはGitHubにて公開しています。章によっては本書よりもサンプルコードの方がわかりやすいかもしれません。ご確認ください。

https://github.com/O-Junpei/TechnicalBookStore5-Sample

対象

本書は、コードを使ったUIレイアウトに興味のある方、Storyboardに苦しめられた経験のある方、Storyboardを使いたくない方を対象としています。

環境

本書を作成するにあたって著者が検証した環境は以下の通りとなります。以下の環境、バージョン以外では正常に動作しない可能性があります。ご了承ください。

- Xcode 10.2
- macOS Mojave 10.14.3

本著の内容はXcodeのバージョンに特に依存するため、不具合などがありましたらXcodeのメジャーバージョンを上記のものと揃えてみてください。Xcodeの過去バージョンはMore Downloads for Apple Developers(https://developer.apple.com/download/more/)からダウンロードすることができます。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、村書を用いた開発、制作、運用は必ずご自身の責任と判断によって行なってください。これらの情報による開発、制作、運用の結果について、著者はいかなる責任も負いません。

目次

第1章

1.1 Storyboardとは

Storyboardとは、Xcodeでプロジェクト作成時に自動生成されるMain.storyboardというファイルのことです。Storyboardを使うと、アプリ画面のUIパーツのレイアウトや画面遷移を視覚的に作成することができます。使う際にプログラミングの知識は不要ですので、ドラッグ&ドロップなどの操作で簡単に使うことができます。また、AutoLayoutを使用することで、UIパーツを画面の大きさや向きの違いに応じて動的に変化する、汎用的な画面レイアウトを作ることができます。

2.2 Storyboard(AutoLayout)の苦しみ

一見すると便利なStoryboard & AutoLayoutですが、以下の苦しみも存在します。

コンフリクトが辛い

Storyboardは開くだけでファイル内容が変更されるため、異なるブランチで同じStoryboardを触るとほぼ確実にコンフリクトします。しかもStoryboardのコンフリクトはとても厄介で、コンフリクトした.storyboardファイルはXcodeで開くことができません。.storyboardファイルをテキストエディタで開き、XMLで書かれた.storyboardファイルを修正することになります。

後からの仕様変更が辛い

複雑なUIをStoryboardで実装するとします。StoryboardにUIパーツを追加し、AutoLayoutの設定も終わり、複数のサイズの画面での動作も確認しました。そんな時に、「ボタンを一つ追加して欲しい」との要望を受けました。せっかく設定したAutoLayoutを削除して、ボタンを置いて、またAutoLayoutを設定して。。。穴を掘って、埋める作業です。
(.xibやStackViewを使う事でこの辛さはある程度軽減します。)

ブラックボックスで辛い(デバッグがしにくい)

AutoLayoutはオープンソースではなく、ブラックボックスとなっているため、何か困ったことが発生したら追いかけることができません。AutoLayoutの設定が良くなかったのか、それともAutoLayout自体のバグなのか、仕様なのかがわかりません。

これらのように、Storyboardには数多くの辛さがあります。
Storyboardの辛さをさらに知りたい人は「auto layout sucks」で検索してみてください。

1.3 コードでUIを書くことのメリットとデメリット

Storyboardを使わずにUIパーツをコードだけで実装する方法が存在します。Storyboard(AutoLayout)の辛さからは逃れることができますが、もちろんデメリットも存在します。

メリット

- Storyboard(AutoLayout)の辛さから解放される
- コンフリクトに強い(したとしても修正が容易)
- 別のプロジェクトからの再利用性が高まる(コピペがしやすくなる)
- 複雑なレイアウトに対応しやすくなる

これらが主なメリットです。Storyboardを使用しないため、Storyboard(AutoLayout)の辛さから解放されます。.storyboardファイルを編集しないため、同じ画面を触ってもコンフリクトしづらくなり、もし起きたとしてもコードベースであるため修正が容易です。その他に、別のプロジェクトからの再利用しやすくなり、アニメーションや入り組んだUIパーツの実装など、複雑なレイアウトへ対応しやすくなります。

デメリット

- 視覚的に分かりにくくなる
- デザイナーさんとの共同作業が難しくなる
- 学習コスト
- 書き方によっては重くなる
- Appleの公式見解的にはNG

これらが主なデメリットです。Storyboardはデザインしたものをそのまま確認することができますが、UIパーツをコードで実装するとデザインを変更するたびにビルドを行う必要があります。Storyboardを使用しないため、デザイナーさんとの共同作業(Storyboardのプロパティを設定してもらうなど)が難しくなります。その他にも学習コストや、書き方によっては重くなるなどのデメリットがあります。そしてコードでUIパーツを実装することはAppleの公式見解的にNGです。

> You should override this method only if the autoresizing and constraint-based behaviors of the subviews do not offer the behavior you want.
AutoLayoutがうまくいかない時のみ、コードで実装して良いよ(意訳)
https://developer.apple.com/documentation/uikit/uiview/1622482-layoutsubviews

しかしこの見解はほぼ形骸化しており、AutoLayoutを使わないため審査が通らないなんてことはありません。Storyboardを使わずにコードのみで実装された有名アプリも数多くリリースされています。

結論として、StoryboardとコードでのUI実装はメリット、デメリットが共にあり、チームのスキルセットやiOSエンジニアの好みを考慮して決めるのが良いと思います。この本を通じて、AutoLayout以外にもUIを作る方法があるんだよ、ということを伝えられたら幸いです。

第2章 環境構築

2.1 Xcodeのダウンロードとインストール

Xcodeの最新版はAppStoreからダウンロードすることが出来ます。
(https://itunes.apple.com/jp/app/xcode/id497799835?mt=12)

過去バージョンを使いたい場合は
More Downloads for Apple Developers(https://developer.apple.com/download/more/)
からダウンロードすることができます。

2.2 プロジェクトの作成

Xcodeのインストールが終わったらプロジェクトを作成します。Xcodeを立ち上げ、[Create a new Xcode project] → [Single View App] →
[Product Name, Organization Name, Organization Identifierを入力] → [Create]でプロジェクトが作成されます。

Product Name、Organization Name、Organization Identifierは適当な値で問題ないですが、Organization Identifierは他のアプリと重複しない値、ドメインの逆順でつけることが推奨されています。(ex, com.google.gmail)

作成されたプロジェクトを開き、シミュレーターが選択されていることを確認した後に、[Product] -> [Run]で実行されます。シミュレーターが起動し、白い画面が表示されれば環境構築は完了です。

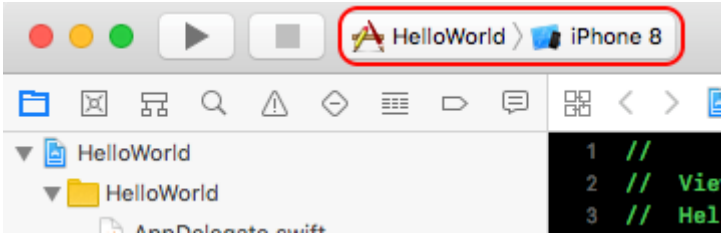

図2.2.1 シミュレーターを選択

2.3 HelloWorld

環境構築が完了しましたらHelloWorldをしてみましょう。作成したプロジェクトの中にあるViewController.swiftを開き、viewdidloadメソッドの中に以下のコードを入力します。[Product] → [Run]でアプリを実行するとHello Wordldと表示されます。

```
override func viewDidLoad() {
    super.viewDidLoad()
    // 背景色を白色に
    view.backgroundColor = UIColor.white
    // インスタンス生成
    let label: UILabel = UILabel()
    // サイズの指定、画面と同じ大きさにする
    label.frame = view.bounds
    // テキストの設定
    label.text = "Hello World"
    // 中央揃え
    label.textAlignment = NSTextAlignment.center
    // Viewに追加
    view.addSubview(label)
}
```

編集したViewControllerはUIViewControllerクラスのサブクラスで、[Single View App]でプロジェクトを作成すると自動で作成されます。初めに表示される画面に何を表示するか制御を行うクラスになります。

UIViewControllerクラスは名前の通り表示されるViewを管理、操作(表示、非表示、配置、アニメーションなど)をするクラスです。受け取ったデータに合わせてTextやViewを表示・管理する役目を持ちます。

UILabelはテキストの表示に使われるクラスで、3.1章で詳しく使い方を解説します。

図2.3.1 Hello World

2.4 Storyboardを削除する

不要となったMain.storyboardファイルを削除します。(LaunchScreen.storyboardファイルはスプラッシュ画面の実装に必要です。)

次にinfo.plistを選択し、その中の[Main storyboard file base name]の項を削除します。最後にAppDelegate.swiftを開き、

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?)
```

メソッドを以下のように修正します。

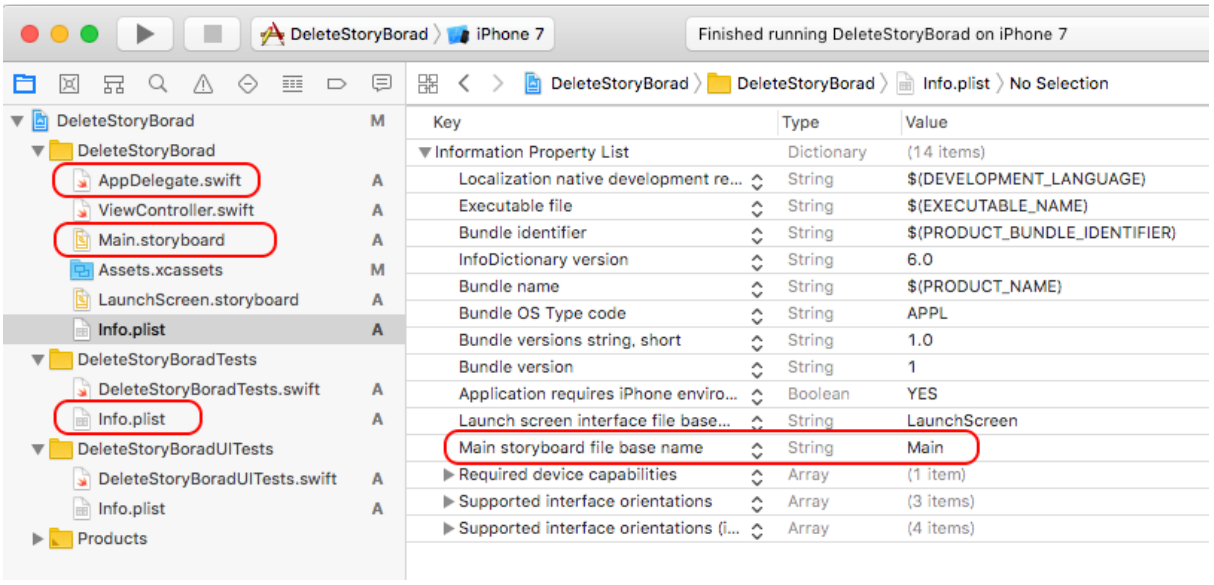

図2.4.1 Main.storyboardファイルの削除とinfo.plistの編集

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        let window = UIWindow(frame: UIScreen.main.bounds)
        window.makeKeyAndVisible()
        let viewController:UIViewController = ViewController()
        window.rootViewController = viewController
        self.window = window
        return true
    }
```

この状態でアプリをビルドし、2.3章と同じように「Hello World」と表示されればMain.storyboardファイルの削除は完了です。

第3章

本章ではコードでUIパーツを実装するときのTipsやサンプルコードをまとめました。

3.1 UILabel

UILabelは読み取り専用テキストの表示に使用されるUIパーツです。UILabelの実装でよく使うメソッドを紹介します。

```
func sizeToFit()
```

UILabelのインスタンスに対して sizeToFit メソッドを使うことで、UILabelのインスタンスのサイズが文字の長さに合わせてリサイズされます。例えば、横幅の長さが固定で、高さが可変のレイアウトを実装したい時に便利です。

また、sizeToFit でリサイズを行う前にリサイズ後のサイズを知りたい時は sizeThatFits メソッドを使う前ことで事前に調整されたサイズを取得することができます。具体的な使い方はサンプルコードで説明します。

```
func sizeThatFits(_ size: CGSize) -> CGSize
```

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = UIColor.white
        // x = 100, y = 100 の位置に
        // width 150, height可変のUILabelを設置
        // インスタンス生成
        let label: UILabel = UILabel()
        // テキストの設定
        label.text = "恥の多い生涯を送って来ました。自分には。。。"
        // 行数の設定、0にすることで表示可能最大行数が無限になる
        label.numberOfLines = 0
        // labelの横幅を設定、高さは0とする
        let size = CGSize(width: 150, height: 0)
        // 文字列の幅に調節したサイズを取得
        let fitedSize = label.sizeThatFits(size)
```

```
        // labelを文字列の幅に調節
        label.sizeToFit()
        // labelの大きさを設定する
        label.frame = CGRect(
            x: 100, y: 100,
            width: fitedSize.width,
            height: fitedSize.height)
        // 背景色を薄い灰色にする
        label.backgroundColor = UIColor.lightGray
        // Viewに追加
        view.addSubview(label)
    }
}
```

図3.1.1 x 100, y 100の位置に横幅150のUILabelを設置

3.2 CustomUIView

オリジナルのUIパーツを作成する方法を紹介します。本章では画像の下にラベルがあるサムネイルのようなUIパーツを作成します。

表示する画像を登録した後に、UIViewクラスを継承したCustomViewクラスを作成し、override init メソッドの中でメンバ変数であるimageViewとlabelの初期化を行います。

次に layoutSubviews メソッド内でCustomView自身を基準としたサイズを取得し、メンバ変数のimageViewとlabelの大きさを設定する処理を実装します。

```
import UIKit
class CustomView: UIView {
    // メンバ変数
    var imageView: UIImageView!
    var label: UILabel!
    // イニシャライザのoverrideに必要
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }
    // イニシャライザ
    override init(frame: CGRect) {
        super.init(frame: frame)
        // 黒い枠線
        layer.borderWidth = 2.0
        layer.borderColor = UIColor.black.cgColor
        // ImageViewをCustomViewに追加
        imageView = UIImageView()
        imageView.image = UIImage(named: "swift")
        imageView.contentMode = UIView.ContentMode.scaleAspectFit
        addSubview(imageView)
        // LabelをCustomViewに追加
        label = UILabel()
        label.text = "Swift"
        label.textAlignment = .center
        addSubview(label)
    }
    // CustomViewが表示される前に呼ばれる
    override func layoutSubviews() {
        // CustomView の横幅と高さを取得
        let viewWidth: CGFloat = frame.width
        let viewHeight: CGFloat = frame.height
        let labelHeight: CGFloat = 20
        // viewの高さからlabelの高さを引い高さがimageViewの高さ
        let imageViewHeight: CGFloat = viewHeight - labelHeight
```

```
        // imageViewの大きさを設定、
        imageView.frame = CGRect(
            x: 0, y: 0,
            width: viewWidth,
            height: imageViewHeight)
        // labelの大きさを設定
        label.frame = CGRect(
            x: 0, y: imageViewHeight,
            width: viewWidth,
            height: labelHeight)
    }
}
```

最後に作成したCustomView クラスのインスタンスをViewController内で使用することで、オリジナルのUIパーツとして使用することができます。

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = UIColor.white
        // CustomViewのインスタンスを生成
        let customView = CustomView()
        // x 50, y 50 に width 100, height 120
        customView.frame = CGRect(x: 50, y: 50, width: 100, height: 120)
        view.addSubview(customView)
    }
}
```

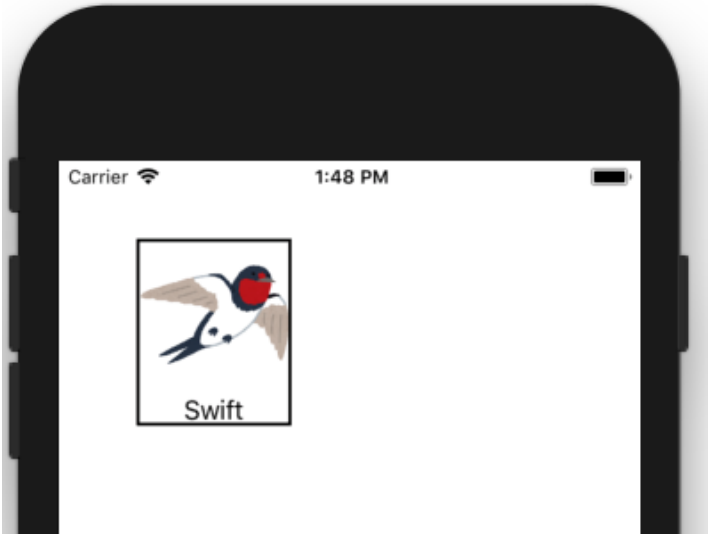

図3.2.1 オリジナルのUIパーツ

3.3 UINavigationController

　UINavigationControllerは階層的な画面遷移を管理するクラスです。UINavigationControllerをコードで実装するには、AppDelegateクラスでUINavigationControllerインスタンスをrootViewControllerとなるように設定を行うことで実装可能です。

　まず遷移元であるViewControllerクラスと、遷移先であるSecondViewControllerクラスを実装します。ViewControllerクラスには押すとSecondViewControllerへ遷移するボタンを付けてあります。

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "FirstView"
        view.backgroundColor = UIColor.white
        let viewWidth = view.frame.width
        let viewHeight = view.frame.height
        // 押されると画面遷移するボタン
        let transitionBtn = UIButton()
        transitionBtn.frame.size =
            CGSize(width: viewWidth * 0.6, height: viewHeight * 0.2)
        transitionBtn.center = view.center
        transitionBtn.backgroundColor = UIColor.blue
        transitionBtn.setTitle("Go SecondVC",
            for: UIControl.State.normal)
        transitionBtn.addTarget(self,
            action: #selector(goSevondVC(sender:)), for: .touchUpInside)
        view.addSubview(transitionBtn)
    }
    // ボタンが押されたら呼ばれるメソッド
    @objc func goSevondVC(sender: UIButton) {
        // 画面遷移
        let secondVC = SecondViewController()
        navigationController?.pushViewController(secondVC,
            animated: true)
    }
}
```

```
import UIKit

class SecondViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "SecondView"
        view.backgroundColor = UIColor.white
    }
}
```

次にAppDelegateでViewControllerのインスタンスを作成し、それを引数にUINavigationControllerのインスタンスを作成し、アプリのrootViewControllerに設定します。

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
    var navigationController: UINavigationController?
    func application(_
        application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        let window = UIWindow(frame: UIScreen.main.bounds)
        window.makeKeyAndVisible()
        let viewController = ViewController()
        // navigationControllerをrootViewControllerに設定する
        navigationController =
            UINavigationController(rootViewController: viewController)
        window.rootViewController = navigationController
        self.window = window
        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {}
    func applicationDidEnterBackground(_ application: UIApplication) {}
    func applicationWillEnterForeground(_ application: UIApplication) {}
    func applicationDidBecomeActive(_ application: UIApplication) {}
    func applicationWillTerminate(_ application: UIApplication) {}
}
```

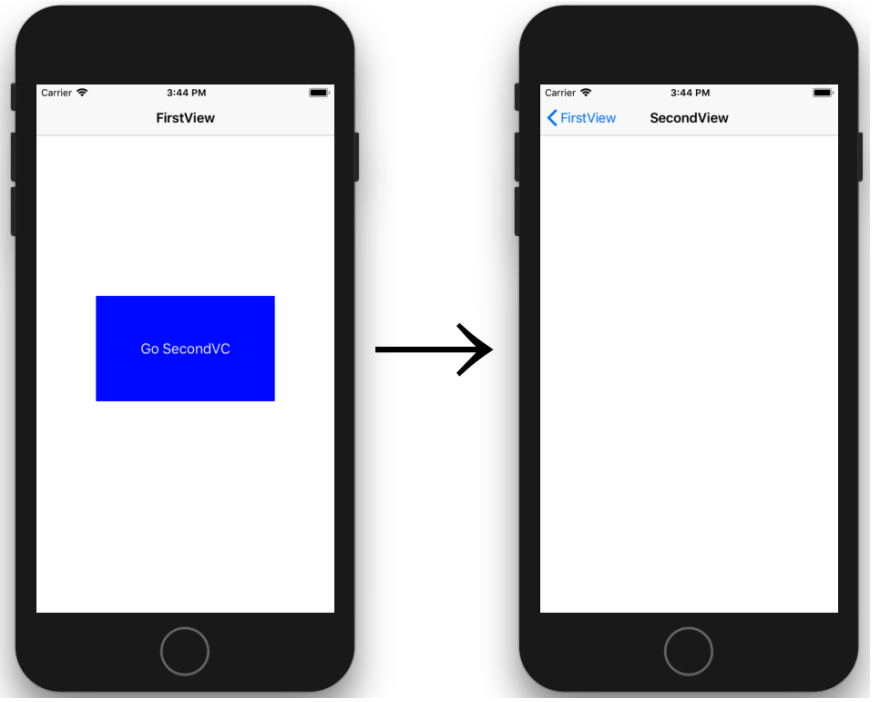

図3.3.1 NavigationControllerによる画面遷移

3.4 UINavigationController & UITabBarController

Twitterの公式アプリのように、UINavigationControllerとUITabBarControllerはセットで使われることも多いです。この場合もUITabBarControllerインスタンスをrootViewControllerとすることで実現できます。

```
import UIKit
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
    func application(_
        application: UIApplication,
        didFinishLaunchingWithOptions
        launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        // MARK: - TabBarController
        // ページを格納する配列
        var viewControllers: [UIViewController] = []

        // 1つ目のViewController
        let firstVC = FirstVC()
        // TabBarのアイコンに設定する画像は30×30の透過画像
        firstVC.tabBarItem = UITabBarItem(
            title: "First", image: UIImage(named: "swift-tab"), tag: 1)
        // NavigationControllerのrootにfirstVCを設定
        let firstNavigationController = UINavigationController(
            rootViewController: firstVC)
        viewControllers.append(firstNavigationController)

        // 2つ目のViewController
        let secondVC = SecondVC()
        secondVC.tabBarItem = UITabBarItem(
            tabBarSystemItem: UITabBarItem.SystemItem.downloads, tag: 2)
        let secondNavigationController = UINavigationController(
            rootViewController: secondVC)
        viewControllers.append(secondNavigationController)

        // 3つ目のViewController
        let thirdVC = ThirdVC()
        thirdVC.tabBarItem = UITabBarItem(
            tabBarSystemItem: UITabBarItem.SystemItem.history, tag: 3)
        let thirdNavigationController = UINavigationController(
            rootViewController: thirdVC)
        viewControllers.append(thirdNavigationController)
```

```
        // TabBarControllerにViewControllerの配列を設定
        let tabBarController = UITabBarController()
        tabBarController.setViewControllers(
            viewControllers, animated: false)

        let window = UIWindow(frame: UIScreen.main.bounds)
        window.makeKeyAndVisible()
        window.rootViewController = tabBarController
        self.window = window
        return true
    }
    func applicationWillResignActive(_ application: UIApplication) {}
    func applicationDidEnterBackground(_ application: UIApplication) {}
    func applicationWillEnterForeground(_ application: UIApplication) {}
    func applicationDidBecomeActive(_ application: UIApplication) {}
    func applicationWillTerminate(_ application: UIApplication) {}
}
```

StatusBar、NavigationBar、TabBarの高さは以下のように取得することができます。

```
import UIKit
class FirstVC: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "First View"
        view.backgroundColor = .orange
        // Viewの横幅を取得
        let viewWidth = view.frame.size.width
        // Viewの高さを取得
        let viewHeight = view.frame.size.height
        // ステータスバーの高さを取得
        let statusBarHeight =
            UIApplication.shared.statusBarFrame.size.height
        // ナビゲーションバーの高さの取得
        var navigationBarHeight: CGFloat = 0
        if let navigationController = navigationController {
            navigationBarHeight
                = navigationController.navigationBar.frame.size.height
        }
        // タブバーの高さを取得
        var tabBarHeight: CGFloat = 0
        if let tabBarController = tabBarController {
            tabBarHeight = tabBarController.tabBar.frame.size.height
        }
```

```
        // 実際に使える画面の高さ
        let contentsViewHeight = viewHeight
            - (statusBarHeight + navigationBarHeight + tabBarHeight)

        let firstViewLabel = UILabel()
        // statusBarとnavigationBarの下にLabelを追加
        firstViewLabel.frame = CGRect(
            x: 0, y: statusBarHeight + navigationBarHeight,
            width: viewWidth,
            height: contentsViewHeight)
        firstViewLabel.textAlignment = NSTextAlignment.center
        firstViewLabel.font = UIFont.boldSystemFont(ofSize: 40)
        firstViewLabel.textColor = .black
        firstViewLabel.text = "First View"
        view.addSubview(firstViewLabel)
    }
}
```

図3.4.1 UINavigationControllerとUITabBarController

3.5 UITableView & UITableViewCell

CustomTableViewCellも3.2章のCustomViewと同様に実装することが可能です。

```
import UIKit
class CustomTableViewCell: UITableViewCell {
    var thumbnailImageView: UIImageView
    var label: UILabel
    override init(style:
            UITableViewCell.CellStyle, reuseIdentifier: String?) {
        thumbnailImageView = UIImageView()
        label = UILabel()
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        // imageView
        thumbnailImageView.backgroundColor = UIColor.lightGray
        thumbnailImageView.contentMode
            = UIView.ContentMode.scaleAspectFit
        contentView.addSubview(thumbnailImageView)

        // label
        label.font = UIFont.systemFont(ofSize: 24)
        contentView.addSubview(label)
    }

    required init(coder aDecoder: NSCoder) {
        fatalError("init(coder: ) has not been implemented")
    }

    override func layoutSubviews() {
        super.layoutSubviews()
        let cellWidth = frame.width
        let cellHeight = frame.height
        let margin: CGFloat = 8

        // thumbnailImageViewの縦、横幅を指定
        let imageViewSize: CGFloat = cellHeight - margin * 2
        // thumbnailImageViewの場所、大きさを指定
        thumbnailImageView.frame = CGRect(
            x: margin, y: margin,
            width: imageViewSize, height: imageViewSize)

        // labelの場所、大きさを背定
        label.frame = CGRect(
            x: cellHeight, y: 0,
```

```
            width: cellWidth - cellHeight, height: cellHeight)
    }
}
```

作成したCustomTableViewCellをTableViewで呼び出します。

```
import UIKit
class ViewController: UIViewController {
    var tableView: UITableView!
    override func viewDidLoad() {
        super.viewDidLoad()
        // テーブルビューの初期化
        tableView = UITableView()
        tableView.delegate = self
        tableView.dataSource = self
        // テーブルビューの大きさの指定
        tableView.frame = self.view.frame
        // カスタムセルをテーブルビューに登録する
        tableView.register(
            CustomTableViewCell.self,
            forCellReuseIdentifier:
                NSStringFromClass(CustomTableViewCell.self))
        // 高さを80に設定
        tableView.rowHeight = 80
        view.addSubview(tableView)
    }
}

// MARK: - TableViewのデリゲートッメソッド
extension ViewController: UITableViewDelegate, UITableViewDataSource {
    // テーブルビューのセルの数を設定する
    func tableView(_ tableView: UITableView,
        numberOfRowsInSection section: Int) -> Int {
        return 10
    }

    // テーブルビューのセルの中身を設定する
    func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        //myItems配列の中身をテキストにして登録した
        let cell
            = tableView.dequeueReusableCell(withIdentifier:
                    NSStringFromClass(CustomTableViewCell.self))
        as! CustomTableViewCell
        cell.thumbnailImageView.image = UIImage(named: "swift")
```

```
        cell.label.text = "セル番号:" + indexPath.row.description
        return cell
    }

    // テーブルビューのセルが押されたら呼ばれる
    func tableView(_ tableView: UITableView,
        didSelectRowAt indexPath: IndexPath) {
        print("\(indexPath.row)番のセルを選択しました！ ")
    }
}
```

図3.5.1 CustomTableViewCell

3.6 iPhoneXのSafeAreaの取得

iPhoneXが登場したことで、iPhoneXのレイアウト(SafeAreaの取得)には気を使うようになりました。SafeAreaはviewDidLayoutSubviews メソッド内で、safeAreaInsets プロパティを参照することで取得することができます。

```
import UIKit
class ViewController: UIViewController {
    var topSafeAreaHeight: CGFloat = 0
    var bottomSafeAreaHeight: CGFloat = 0
    var topSafeAreaLabel: UILabel!
    var bottomSafeAreaLabel: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = UIColor.white
        // viewDidLoadの段階では高さは取得できない
        print("in viewDidLoad")
        print(topSafeAreaHeight) // 0
        print(bottomSafeAreaHeight) // 0

        // Top Safe Area 下のラベル
        topSafeAreaLabel = UILabel()
        topSafeAreaLabel.backgroundColor = UIColor.red
        topSafeAreaLabel.text = "↑Top Safe Area"
        topSafeAreaLabel.textAlignment = NSTextAlignment.center
        topSafeAreaLabel.textColor = UIColor.white
        topSafeAreaLabel.font = UIFont.boldSystemFont(ofSize: 20)
        view.addSubview(topSafeAreaLabel)

        // Bottom Safe Area 上のラベル
        bottomSafeAreaLabel = UILabel()
        bottomSafeAreaLabel.backgroundColor = UIColor.blue
        bottomSafeAreaLabel.text = "↓Bottom Safe Area"
        bottomSafeAreaLabel.textAlignment = NSTextAlignment.center
        bottomSafeAreaLabel.textColor = UIColor.white
        bottomSafeAreaLabel.font = UIFont.boldSystemFont(ofSize: 20)
        view.addSubview(bottomSafeAreaLabel)
    }

    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        if #available(iOS 11.0, *) {
            // viewDidLayoutSubviewsではSafeAreaの取得ができている
            topSafeAreaHeight = view.safeAreaInsets.top
            bottomSafeAreaHeight = view.safeAreaInsets.bottom
```

```
            print("in viewDidLayoutSubviews")
            print(topSafeAreaHeight) // iPhoneXなら44, その他は20.0
            print(bottomSafeAreaHeight) // iPhoneXなら34,  その他は0

            let width = view.frame.width
            let height = view.frame.height
            let labelHeight: CGFloat = 50
            topSafeAreaLabel.frame = CGRect(
                x: 0, y: topSafeAreaHeight,
                width: width, height: labelHeight)
            bottomSafeAreaLabel.frame = CGRect(
                x: 0, y: height - (labelHeight + bottomSafeAreaHeight),
                width: width, height: labelHeight)
        }
    }
}
```

図3.6.1 iPhoneXとiPhone8でのSafeAreaの取得

謝辞

本書を作成するにあたり、お世話になりました各ライブラリ、ツールの作者様、表紙イラストを描いてくれた友人、レビューをしてくれた友人、同僚、先輩の方々に感謝いたします。

あとがき

「Storyboardを使わずにコードだけでアプリを作る本」は実装のお役に立ちそうでしょうか。少しでもお役に立てば幸いです。

著者紹介

普段はSwiftを書いたりReactNativeを書いているエンジニアです。最近のマイブームはSwiftのWebフレームワークのVaporです。近いうちにVaporで何かを作ってリリースしたり、本体にコントリビュートできればなと思ってます。

StoryBoardを使わずにコードだけでアプリを作る本（Swift 5.0）

発行日: 2019/4/14
著者: かびごん小野 (@takoikatakotako)
Webページ: swiswiswift.com
