# iOS エンジニアが Xamarin を始めた話

宇佐見 公輔

# About Me

- 宇佐見公輔（twitter : @usamik26）
- フェンリル株式会社 共同開発部
  - iOS アプリ開発
- プライベート
  - OS X アプリ開発 : CotEditor（テキストエディタ）

# Xamarin 歴

- 僕は半年ほど前から開始
- 社内には実務経験がある詳しい人が既に何人かいた
- 実務で **Xamarin + MvvmCross** を使用
  - **MvvmCross github** に **pull request** を送ったことも

# なぜ Xamarin をやろうと思ったか

# C# が使える？

- スマホアプリ開発が C# でできる
  - ・・・これは僕には無関係
    - iOS アプリ開発はできる
    - C# / .NET はあまり知らない

# クロスプラットフォーム開発

- iOS / Android アプリ開発が共通化できる
  - これは大きなメリット
    - 実務上、両方のアプリ開発が並行して行われることは多い
    - iOS アプリ開発者と Android アプリ開発者とで、アプリの仕様認識違いを防ぐのは案外たいへん

# 他のツールとの比較 (1)

- 他のツールでは、各プラットフォーム開発の良さを生かせないことが多い
  - **Xamarin** はネイティブ **SDK** を生かしている
  - さらに最新環境への対応も早い

# 他のツールとの比較 (2)

- 他のツールでは、プラットフォームにあわせた挙動をさせるのが案外難しい
  - iOS アプリが Android アプリっぽい動きをしたり、Android アプリが iOS アプリっぽい動きをしたりするのは不自然

# MVVM

- スマホアプリ開発が **MVVM** でできる
  - 個人的に決定打となったのはこの点
    - **iOS** アプリ開発では **ReactiveCocoa** を使うという選択があるが、まだプラクティスが確立されていない感じ
    - **C# / .NET** 文化圏でつちかわれてきた設計手法だから、**Xamarin** は相性がよい

# 結局、Xamarin を始めた理由は・・・

- MVVM で開発できる
  - ビューとロジックの分離
  - クロスプラットフォーム開発との親和性

# 改めて MVVM

# MVVM (Model - View - ViewModel)

- View はプラットフォーム依存
- View と ViewModel は Binding で連携
- ViewModel / Model がロジック部分、プラットフォーム非依存

# MVVM とプラットフォームの関係

- C# / .NET 文化圏でつちかわれてきた
- 当初は主に Windows プラットフォーム上のみで使われていた？
- しかし、ViewModel をプラットフォーム非依存にできるため、クロスプラットフォーム開発と相性がよい

# MVVM とスマホプラットフォーム

- View を各プラットフォームで開発する
- このため、各プラットフォームの世界観を大事にすることができる
  - iOS らしさ、Android らしさは大事
- クロスプラットフォーム開発でこそ MVVM の良さが生きるのでは、と勝手に思っている

# 実際に Xamarin.iOS を使ってみる

# まず最初に

- **C#** わからない
  - 触ったことはあったけど **C# 1.0 / 2.0** くらい
  - とりあえず基本的な文法をさらっと見て、あとは実務と並行で勉強

# C# に慣れてくると

- 言語が違うだけで根っこのフレームワークは同じ
  - フレームワーク関連はこれまでの知識がほぼそのまま通用する
- コードが書きやすいことに気づく
  - 個人的にはラムダ式とか LINQ とか好き

# よくやるミス

- Xamarin Studio 上で Objective-C のコードを書いてしまう
  - [UIImage ima くらいまで書いて、あれ補完されないなあ、とか
- メソッド名がわからなくなる
  - 命名規則が Objective-C 風と C# 風では違っている

# よくやるミスの逆パターン

- 慣れてくると逆パターンもやらかした
  - Xcode 上で `UIImage image =` とか書いていて、なんでビルドエラーなのかなあ、とか
    （アスタリスクを忘れる）

# ドキュメント？

- MonoDoc のドキュメントがちょっと情報不足？
  - 対応するメソッドの Xcode ドキュメントを参照したり
- Xamarin のサイトにはいろいろドキュメントがあった

# 便利クラス

- UITableViewSource
  - UITableViewDataSource と UITableViewDelegate の統合
  - こいつらはだいたいあわせて使うことが多いのでちょっと楽になる
- UICollectionViewSource もある

# Storyboard

- 名前入れるだけで勝手にプロパティ追加してくれるとか便利
- ダブルクリックでアクション生成できるとか便利
- Xcode にはこういうのなかったな・・・

# AutoLayout

- Constraint の追加の仕方がちょっと違う
  - 慣れると楽
- Constraint ダブルクリックで編集できる

# API 100% 対応

➔ AVFoundation とか使いたいんだけど・・・

➔ と思ったら、何にも考えなくても使えた

```
using MonoTouch.AVFoundation;

var session = new AVCaptureSession();
...
```

# MVVM の導入

# MvvmCross

- やり方はいろいろあるけど、担当プロジェクトでは **MvvmCross** を採用
- 社内に先駆者が多い
- オープンソースなのでなんなら修正する（実際した）

# これも最初に

- MVVM わからない
  - 実際にコード書いてみると最初は案外書けなくてとまどう
  - なんか手足縛られてる感がある

# MVVM に慣れてくると

- 逆になんでも **MVVM** にしたくなる
  - **Xamarin** なしで **iOS** アプリ？
    だったら **ReactiveCocoa** 使おうか？

# 便利クラス

- Binding を楽にしてくれる
  - MvxTableViewCell
  - MvxImageView
    - URL 渡したら勝手に画像取得してくれるとか

# MvvmCross プロジェクト構成

- Core プロジェクト : Model / ViewModel
- Touch プロジェクト : iOS View
- Droid プロジェクト : Android View

# よくやる（？）ミス

- MonoTouch のクラスが使えない、と思ったら ViewModel だった
  - ViewModel : Core プロジェクト
  - View : Touch プロジェクト

# よくやる（？）ミス (2)

- View にがりがり実装してしまう・・・
  - テンパってきたときにやらかす
  - MVVM の良さを台無しにする

# プラグイン

- プラットフォーム依存の機能を **ViewModel** から使いたい
  - インターフェース定義は **Core** プロジェクト
  - 実装は **Touch** プロジェクト

# プラグインのよしあし

- 正直わざわざプラグイン化するのは面倒くさい
  - だけどそうしないと View にがりがり書くことになる
- 一度書いてしまえばすっきりしたコードが書ける

# 懸念点

- 64ビット対応・・・？
  - Xamarin の Unified の仕組みに MvvmCross が対応してくれないとつらい
- Forms に移行したほうがいいの？
  - Forms で iOS / Android それぞれの世界観を大事にできるなら

# まとめ

# まとめ

- Xamarin で C# / MVVM で iOS アプリ作るの気持ちよくなってきた
- いま絶賛 Xamarin 案件忙しいので死なない程度にがんばる