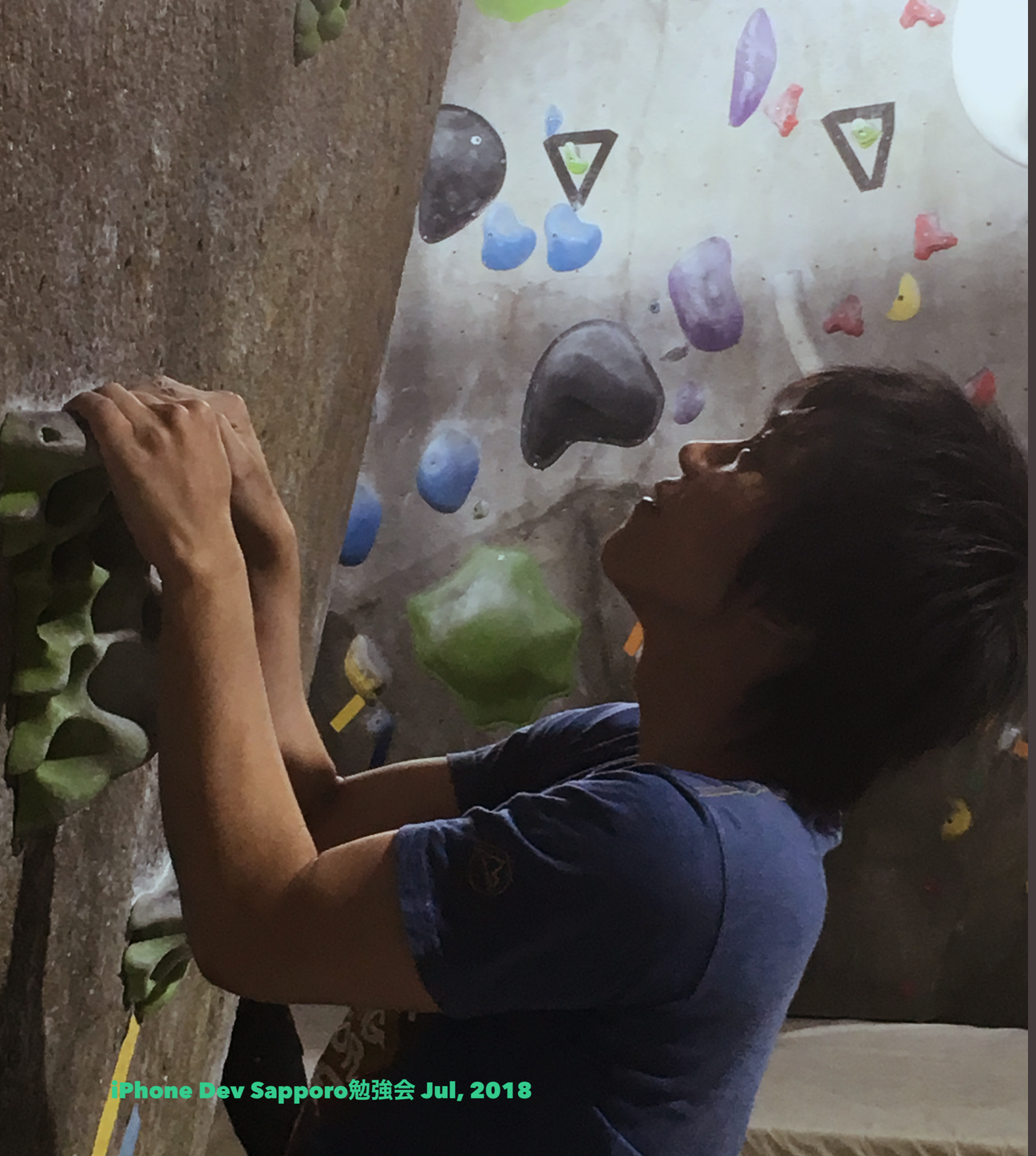
コード生成のススメ

iPhone Dev Sapporo勉強会 Jul, 2018



Takeshi Ihara

- Twitter: @nonchalant0303
- GitHub: Nonchalant

コード生成

- SwiftではRuntime Reflectionがあまり強力ではない
  - プログラムの実行過程でプログラム自身の構造を読み取ったり書き換えたりする技術
- その代わりSwiftではコード生成で解決することが増えてきた
  - ex. ボイラープレートの自動生成

DIKit

https://github.com/ishkawa/DIKit

DIKit provides interfaces to express dependency graph.
A code generator named dikitgen finds implementations of the interfaces, and **generate codes** which satisfies dependency graph.

Cuckoo

https://github.com/Brightify/Cuckoo

Cuckoo has two parts. One is the runtime and the other one is an OS X command-line tool simply called CuckooGenerator.

Unfortunately Swift does not have a proper reflection, so we decided to use **a compile-time generator** to go through files you specify and generate supporting structs/classes that will be used by the runtime in your test target.

DIKit

https://github.com/ishkawa/DIKit

Then, integrate DIKit.framework to your project. There are some option to install DIKit.framework.

- **Manual**: Clone this repository and add `DIKit.xcodeproj` to your project.
- **Carthage**: Add a line `github "ishkawa/DIKIt"` to your Cartfile and run `carthage update`.

Optionally, insert shell script running `dikitgen` to early part of build phases.

```
if which dikitgen >/dev/null; then
  dikitgen ${SRCROOT}/YOUR_PROJECT > ${SRCROOT}/YOUR_PROJECT/AppResolver.generated.swift
else
  echo "warning: dikitgen not installed, download from https://github.com/ishkawa/DIKit"
fi
```

Cuckoo

https://github.com/Brightify/Cuckoo

And add the following `Run script` build phase to your test target's `Build Phases`:

```
# Define output file. Change "$PROJECT_DIR/${PROJECT_NAME}Tests" to your test's root source folder, if it's
OUTPUT_FILE="$PROJECT_DIR/${PROJECT_NAME}Tests/GeneratedMocks.swift"
echo "Generated Mocks File = $OUTPUT_FILE"

# Define input directory. Change "${PROJECT_DIR}/${PROJECT_NAME}" to your project's root source folder, if i
INPUT_DIR="${PROJECT_DIR}/${PROJECT_NAME}"
echo "Mocks Input Directory = $INPUT_DIR"

# Generate mock files, include as many input files as you'd like to create mocks for.
"${PODS_ROOT}/Cuckoo/run" generate --testable "$PROJECT_NAME" \
--output "${OUTPUT_FILE}" \
"$INPUT_DIR/FileName1.swift" \
"$INPUT_DIR/FileName2.swift" \
"$INPUT_DIR/FileName3.swift"
# ... and so forth, the last line should never end with a backslash

# After running once, locate `GeneratedMocks.swift` and drag it into your Xcode test target group.
```

Sourcery

https://github.com/krzysztofzablocki/Sourcery

Sourcery is **a code generator** for Swift language, built on top of Apple's own SourceKit. It extends the language abstractions to allow you to generate boilerplate code automatically.

Sourcery

```
protocol Animal {
    func bark()
}

struct Dog: Animal {} // コンパイルエラー

struct Apple {}
```

Stencil

ファイルテンプレート言語

```
{% for struct in types.structs where struct.implementing.Animal %}
extension {{ struct.name }} {
    func bark() {
        print("{{ struct.name }}")
    }
}
{% endfor %}
```

↓ Sourcery

```
extension Cat {
    func bark() {
        print("Cat")
    }
}
```

SourceKitten

https://github.com/jpsim/SourceKitten

- An adorable little framework and command line tool for interacting with SourceKit
- DIKit, Cuckoo, Sourceryで採用されている

SourceKitten

使ってみる

```
// Source.swift
struct Dog {}

// SourceKitten
import SourceKittenFramework

let file = File(path: "./Source.swift")!
let dic = try! Structure(file: file).dictionary
print(dic)
```

SourceKitten

`[String: SourceKitRepresentable]`

```
struct Dog {}
```

```
[
    "key.diagnostic_stage": "source.diagnostic.stage.swift.parse",
    "key.substructure": [
        [
            "key.bodylength": 0,
            "key.nameoffset": 7,
             "key.accessibility": "source.lang.swift.accessibility.internal",
             "key.length": 13,
             "key.name": "Dog",
             "key.kind": "source.lang.swift.decl.struct",
             "key.namelength": 3,
             "key.offset": 0,
             "key.bodyoffset": 12
        ]
    ],
    "key.offset": 0,
    "key.length": 13
]
```

FactoryProvider

https://github.com/Nonchalant/FactoryProvider

- SourceKitten + Stencilを使ってファクトリーを自動生成するライブラリを作ってみた！
- 詳細はどこかに資料あげます！ (iOSDC落ちてしまった...)

Demo

まとめ

- コード生成楽しい！がやりすぎると属人化するのでシンプルに扱うのが大事な気がした
- iOSDC or DevSapでまた会いましょう！