Swift 中級編

Nishiyama Yusei (@yuseinishiyama)

Swift LT at Yahoo! JAPAN

2014/6/25

自己紹介

name:Nishiyama Yusei

twitter:@yuseinishiyama

github:yuseinishiyama

blog:http://yuseinishiyama.com

- iOSアプリケーション開発者
- Unityも少々
- 最近はOpenCV、機械学習の勉強をしているが数学がアレなのでアレ
- WWDC2014関連は、MetalをはじめとしたGPU周りを追っていきたい

ヤバい

Swift情報出尽くしてる感がある

※まとめていただきありがとうございます。
大変重宝しております。

とりあえずドキュメント

全部読んでニッチな情報探そう...

“Using Swift With Cocoa and Objective-C”

“Swift Programming Language”

読んで気づいた

私はObjective-Cが大好きです

第一部

Objective-Cを愛する人のためのSwift

巷の声

- Objective-Cは潰しが効かない。
- Objective-Cのシンタックスは糞。
- SwiftでObjective-Cから解放された。
- []←これが生理的に無理。
- (()(()))←これは大好きだけどね！

一方、私は…

```
Ltmp9:
        .cfi_def_cfa_register %rbp
        subq    $32, %rsp
        movl    $0, %eax
        movl    $10, %edx
        movl    $0, -4(%rbp)
        movl    %edi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -24(%rbp), %rsi
        movq    L_OBJC_SELECTOR_REFERENCES_(%rip), %rcx
        movq    %rsi, %rdi
        movq    %rcx, %rsi
        movl    %eax, -28(%rbp)         ## 4-byte Spill
        callq   _objc_msgSend
        movl    -28(%rbp), %eax         ## 4-byte Reload
        addq    $32, %rsp
        popq    %rbp
        ret
        .cfi_endproc
```

◀objc_msgSendの勇姿と
CPUの歓喜(左図)

IMP

豊富な実行時情報

消えないObjective-Cへの愛

動的束縛

NEXTSTEP

メッセージ送信

Swiftに

Objective-Cの面影を探す毎日...

超絶いとおしいid型

```
typedef struct objc_object {
    class isa;
} *id;

// かわいい...
```

“Swift imports id as AnyObject.”

「Using Swift with Cocoa and Objective-C」

より

いた！！

使ってみる

```
// 適当な関数を宣言して...
func someFunc() { return }

// id型相当のオブジェクトを宣言して...
var anyObj : AnyObject

// コンパイルは通るはず...
anyObj.someFunc()
```

'AnyObject' does not have a member named 'someFunc()'

コンパイルできない...

こんなのid型じゃない！！

ドキュメントのサンプルコード

```
var myObject: AnyObject
myObject = NSDate()
myObject.characterAtIndex(5)
// crash, myObject does't respond to that method”

// ドキュメントでは実行時エラーになると書いてある。
```

もう少し調べてみる

- You can also call any Objective-C method and access any property without casting to a more specific class type. This includes Objective-C compatible methods marked with the @objc attribute.

- @objc属性がついた、Objective-C互換のメソッドでないといけないらしい。

さらに...

- When you define a Swift class that inherits from NSObject or any other Objective-C class, the class is automatically compatible with Objective-C.

- どうやらObjective-Cのクラスを継承したSwiftのクラスは自動的にObjective-C互換になるらしい。

- さっきの例で呼び出していたcharacterAtIndexはObjective-Cのメソッドだから動的に呼び出せた。

気を取り直して

```
// Objective-C互換のクラス。
@objc class SomeClass {
    func someFunc() { return }
}

// AnyObjectにInt型の値を代入。
var anyObj : AnyObject = 1

// 実装されていないメソッドの呼び出し...
anyObj.someFunc()
```

-[__NSCFNumber someFunc]: unrecognized selector sent to instance 0x137

unrecognized selector

この声は…

id型だ！！！！

Selectorも探す

```
let mySEL: Selector = “someFunc"

// 注意: Playground上だとクラッシュする。

// ※コマンドラインからのREPLだと大丈夫。
```

補足:
文字列互換にするプロトコル

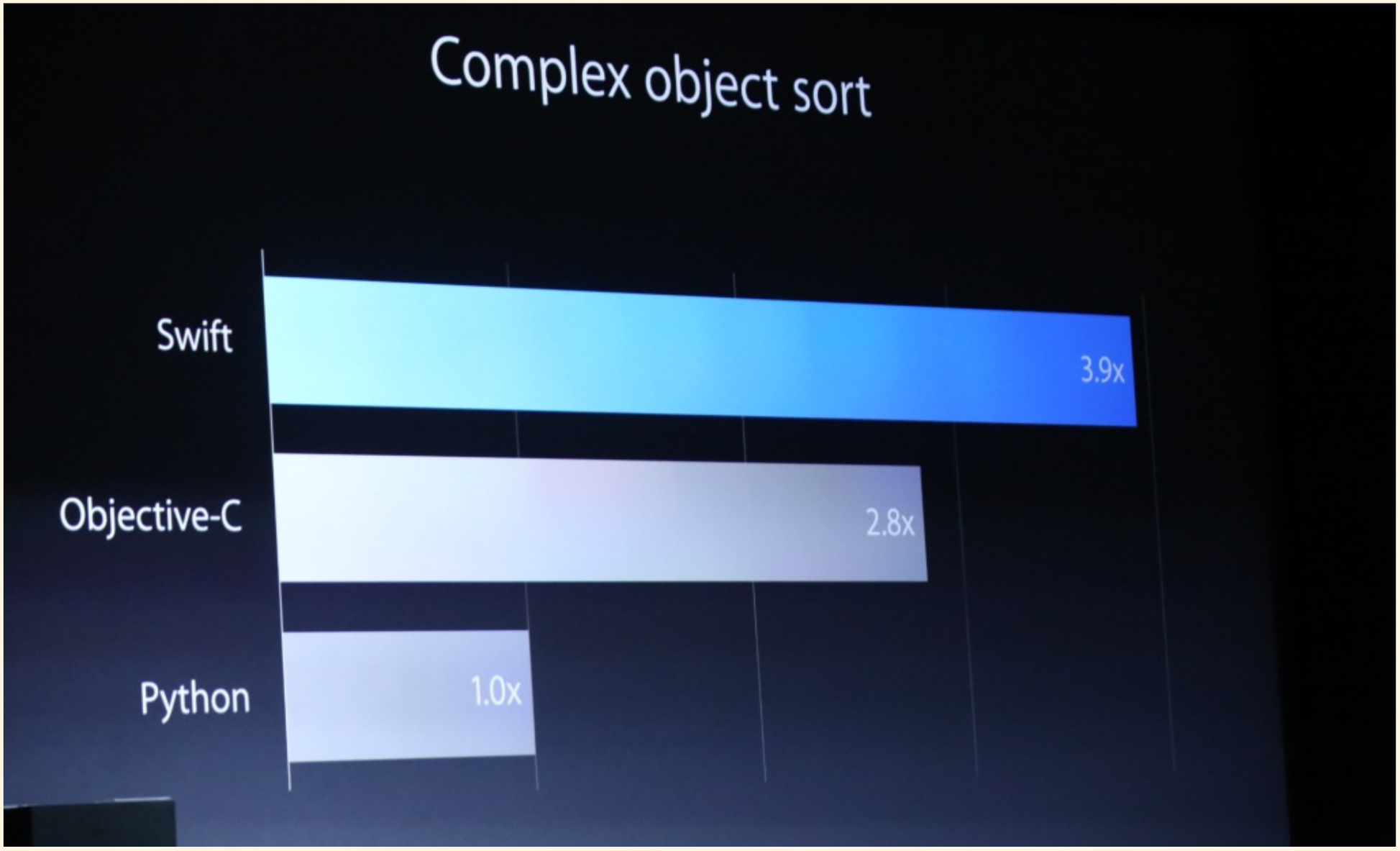
```
// Selectorは以下のプロトコルに適応しているので、
// 文字列から初期化できる。
protocol StringLiteralConvertible :
ExtendedGraphemeClusterLiteralConvertible {
    typealias StringLiteralType
    class func convertFromStringLiteral(value:
StringLiteralType) -> Self
}

protocol ExtendedGraphemeClusterLiteralConvertible {
    typealias ExtendedGraphemeClusterLiteralType
    class func
convertFromExtendedGraphemeClusterLiteral(value:
ExtendedGraphemeClusterLiteralType) -> Self
}
```

Target-Actionパターンは健在

```
class UIResponder : NSObject {
   .
   .
   .
    func canPerformAction(action: Selector,
withSender sender: AnyObject!) -> Bool
    // Allows an action to be forwarded to another
target. By default checks -
canPerformAction:withSender: to either return self,
or go up the responder chain.
    func targetForAction(action: Selector,
withSender sender: AnyObject!) -> AnyObject!
   .
   .
   .
}
```

結論

- Swiftでも動的なコードがかける。
- とはいうものの、結局のところObjective-Cをかましているだけ。
- Objective-Cとの連携を考えない限り、積極的に使用するべきはなさそう。



そもそも、メソッドの動的呼び出しを辞めたから

高速化したに違いない...

(静的にすればLLVMによる最適化のメリットも受けれそう)

第二部

~~ネタ切れ~~

Swift小ネタ集

Special Literals

- __FILE__
  現在のファイル名

- __LINE__
  現在の行数

- __COLUMN__
  現在の列数

- __FUNCTION__
  実行中の関数名

REPLでやると

```
__FILE__     // => “<REPL>"

__FUNCTION__// => "$__lldb_expr"
```

カリー化と部分適用

“カリー化 (currying) とは、計算機科学分野の技法の一つ。複数の引数をとる関数を、引数が「もとの関数の最初の引数」で戻り値が「もとの関数の残りの引数を取り結果を返す関数」であるような関数にすること。”

http://ja.wikipedia.org/wiki/カリー化

Haskell、MLのようにデフォルトでカリー化されるわけではない

```
func addTwoNumbers(a: Int, b: Int) -> Int {
    return a + b
}

var partial = addTwoNumbers(4)

// コンパイルエラー
```

こう書けば部分適用できる！

```
func addTwoNumbers(a: Int)(b: Int) -> Int {
    return a + b
}

var partial = addTwoNumbers(4) // => (b: Int) -> Int

var reuslt = partial(b: 5)     // => 9
```

Preprocessor Directive

- in Swift you use a global constant instead.

  プリプロセッサではなく、グローバル定数を使う。

- Complex macros are used in C and Objective-C but have no counterpart in Swift.

  関数形式のマクロにあたるようなものは存在しない。

メソッドの命名規約

操作＋前置詞スタイルは踏襲される模様

```
class myCounter {
    func incrementBy(amout: Int, numberOfTimes: Int) {
        // increment
    }
}

let counter = myCounter()

// デフォルトでは、第一引数の外部引数名は省略される。
counter.incrementBy(1, numberOfTimes:1)
```

第二引数以降も
引数名を省略したい場合

```
class myCounter {
    // アンダースコアでオーバーライドしてしまう。
    func incrementBy(amout: Int, _ numberOfTimes: Int) {
        // increment
    }
}

let counter = myCounter()

counter.incrementBy(1, 1)
```

Overflowの扱い

- Swiftのデフォルトの挙動ではオーバーフローした場合エラーが発生する。

- オーバフローを許容する場合、
  アンパサンド(&)から始まる算術演算子を使用する必要がある。

- &+, &-, &*, &/, &% ←こいつら

呼び出し元に帰ってこない関数の宣言

```
@noreturn func tryMe() {
    exit(EXIT_FAILURE)
}

// もちろんexit自体も@noreturn
@noreturn func exit(_: CInt)
```

まとめ

型推論付きの静的型付け言語

Swift最高！！！(寝返る)

ありがとうございました