特別解説

基般

プログラミング言語 Swift の紹介

沼田哲史（大阪電気通信大学 総合情報学部 デジタルゲーム学科）

新しいプログラミング言語 Swift

本稿では，Apple 社が 2014 年 6 月 2 日に行った WWDC（World Wide Developers Conference）において発表した，iOS と OS X のアプリケーション開発用の新しい言語である Swift について解説したいと思います．Swift は Apple 社が時代に合わせて投入してきた新しい言語であり，コーディング時のミスが少なく高い拡張性を持つように工夫され，今後のプログラミング言語の基準の 1 つとして注目すべき言語です（**図-1**）．

図-1　Swift 言語のロゴ（Apple 社の公式ホームページより）

Swift が登場する背景

2008 年に iPhone SDK が公開されて以降，iOS というプラットフォームに参入する企業や個人の数は飛躍的な速度で伸び，iOS と OS X の 2 つの環境でしか使われない Objective-C というプログラミング言語の利用者数は C++ や C# といった有名な言語を凌ぐほどになりました．そして 2014 年の 6 月に，Apple 社から新しいプログラミング言語として Swift が紹介されました．Swift は Objective-C よりもおよそ 1.4 ～ 1.7 倍高速に動作し，よりプログラミングしやすい環境として紹介されています．

今回 Apple 社から発表された Swift はかなり突然に登場したようにも思われますが，その背後で Apple 社による開発環境の基盤拡充が着実に進められていたことは見逃せません．Apple 社が iBooks 上で公開している「The Swift Programming Language」によれば，Swift の開発には数年かかっています．2011 年 10 月に登場した iOS 5 で新しく採用されたメモリ管理機構である Automatic Reference Counting（ARC）は，Objective-C のプログラムにおけるメモリ管理に起因するバグを大幅に低減させました．ARC と同時に，並列処理の記述のためのブロック記法が導入されたこともあって，コード自体もかなり簡潔に書けるようになっています．さらに 2012 年 6 月に追加されたコレクション・リテラルやモジュールの導入によって，Objective-C はモダンな言語技術に支えられて効率的に動作し，プログラミング自体もモダンな記述が可能となる基盤が構築されたのです．そうした基盤の上に，さらにプログラミングしやすい言語として登場したのが Swift というわけです．

Swift は Objective-C と同様に，Xcode に組み込まれる形で提供されます．Xcode は現在 Apple 社のオンラインストア App Store 上で無償提供されていますので，Swift をサポートしたバージョンは，この秋に登場するとされている iOS 8 の登場に合わせて，アップデートの形で提供されると思われます．今後は新しいプロジェクトを作成するときに，Swift と Objective-C のどちらをメインで使用するかを選択して，プログラミングを始めることになります．

Swift は今後の Apple 社のデバイス向けにアプリケーションを作成する際に使われる標準言語の 1 つとして，新しい環境で新たなアプリケーション制作を考える若い人たちにとって修得必須の言語となる可能性があります．

Swift の言語仕様

WWDC の基調講演において，Swift は「Objective-C without the C」として紹介されました．すなわち，多くのフレームワークを資産として持っている Objective-C の良いところを残しながら，C 言語のレガシーな部分を取り除いた言語として登場したのです．

まず簡単な Swift のコードを見てみましょう（**図-2**）．

Swift では，let を使って定数の宣言を，var を使っ

```
let π = 3.14159
for var θ = 0.0; θ < 2*π; θ+=0.2*π {
    println("角度は\(θ)ラジアン")
}
```

図 -2　角度を少しずつ変えながら 360 度まで表示していくコードの例

```
var str = "test.png"
str = 3   // ←コンパイルエラーが出る
```

図 -3　推測される型と操作が一致しないコード記述

```
let π : Double = 3.14159
var θ : Double = 2*π
```

図 -4　明示的に型を指定した変数と定数の宣言

て変数の宣言を行います．C 言語とは違い，明示的な型の宣言が不要で，行末のセミコロンも不要だということに気づくでしょう．また，変数や定数の名前に「π」や「θ」といった多バイト文字が使えているのも Swift の特徴の 1 つです．Apple 社のデモでは犬や猫の絵文字まで使われていますが，Swift では UTF-8 の Unicode に言語仕様が対応しており，π (0xcf80) や θ (0xceb8) といったギリシャ文字を使うこともできるのです（それが良いかどうかという議論はさておき，ですが）．println() は標準で用意されている標準出力への文字出力のための関数ですが，文字列の中にバックスラッシュとそれに続く丸括弧を書き，その中に変数名や数値式を書くことで，そこに変数の値や算出された数値が代入される形で文字の書き出しが行われます．

Swift では，代入される値に応じて，変数の型が推測されます．たとえば**図 -3** のように，最初に文字列を代入した変数に対して，数値を代入しようとすると，コンパイルエラーが出ます．

変数や定数の宣言時に，変数名の後ろにコロンを書いて型名を書くことで，Int や Double といった型名を明示的に指定することもできます（**図 -4**）．

また，変数や定数の宣言時に値を代入しない場合には，代入された値に基づく型の推測ができませんので，宣言時に明示的な型の指定が必要となります（**図 -5**）．

オプショナルな値

Objective-C（やその元となる C 言語）でバグが起きやすかった点の 1 つに，ポインタ変数に対していつでも無効な値として NULL（nil）が代入できるという点が挙げられます．これを解決するために，Swift

```
var str   // ←コンパイルエラーが出る
str = "test.png"
```

図 -5　変数宣言時に型推論か型指定かのどちらかは必要

```
var str = "Hello"
str = nil    // ←コンパイルエラーが出る
```

図 -6　無効な値の代入は原則禁止されている

```
var str:String? = "Hello"
str = nil
```

図 -7　nil の代入が許されたオプショナルな変数

```
var str: String? = "Hello"
str = nil
if str {
   println("str is valid.")
}
if !str {
   println("str is invalid.")
}
```

図 -8　オプショナルな変数に対する有効・無効のチェック

では nil というキーワードで表される無効な値の代入が，デフォルトで禁止されています（**図 -6**）．

Swift では，nil を代入して「無効な値が入っている状態」を表すことのできる変数は，宣言時に型指定をし，その型の後ろにクエスチョンマークを付けることで，「オプショナルな値」として，nil がセットできるようになります（**図 -7**）．

オプショナルな値に対しては，if 文の条件に直接変数名だけを書くことで，値が有効かどうかをチェックすることができます．またエクスクラメーションマーク（!）を変数名の頭に付けると，無効かどうかのチェックになります．この変数名だけを条件として書くという記法は，変数がオプショナルな値として宣言されていない場合にはコンパイルエラーとなります（**図 -8**）．

関数の宣言と呼び出し

Swift では関数の宣言は**図 -9** のように記述します．「func」というキーワードの後に関数名を書き，その後に丸括弧を付けて引数のリストを書きます．関数の引数にはコロン（:）を付けて後ろに型名を書き，型を明示的に指定しなくてはいけません．丸括弧の後ろには，「->」とマイナス記号と不等号をつなげて書いて矢印を表し，その後ろにリターン値の型を書きます．リターン値がない場合はこのリターン値の型記述は不要です．

```
func add(a:Int, b:Int) -> Int {
   return a + b
}
println(add(4, 5))  // 9が出力される
```

図 -9　2 数を足し合わせる関数

```
func add 敬称(name:String, title:String="さん")
-> String {
    return name+title
}

println(add 敬称("鈴木"))  // "鈴木さん"と出力
println(add 敬称("田中", title:"君"))  // "田中君"
と出力
```

図 -10　関数の第 2 引数にデフォルト値を指定した例

```
// Objective-C
UIButton *button = self.myButton;
[button setTitle:@"Stop" forState:UIControlState
Normal];

// Swift
let button = myButton
button.setTitle("Stop", forState:.Normal)
```

図 -11　Swift と Objective-C のメソッド呼び出しの比較

関数の引数には，C++ のようにデフォルト値を指定できます．**図 -10** の例では，第 2 引数のデフォルト値を指定していますので，第 2 引数を省略して 1 つの引数だけで関数を呼び出すことができます．明示的に第 2 引数を書いた場合には，その値でデフォルト値が上書きされます．

Objective-C に似たメソッド呼び出し

Objective-C の豊富な資産を利用するためでもあるでしょうが，Swift のメソッド呼び出しにおいては，Objective-C と同様の名前付き引数を使用します．ただし，Swift のメソッド呼び出しの書き方は，角括弧を使う Objective-C の独特な書き方ではなく，ドットと丸括弧を使う Java や C# に近いものです．第 1 引数の名前は省略でき，定数の名前も一部省略して書くことができます．**図 -11** に同じ動作を行うコードを Objective-C と Swift で比較して書いてみましたが，Swift の方がすっきりしたコードになっていることが分かるでしょう．

クラス定義

Swift におけるクラス定義は，C++ の書き方に似ています．「class」キーワードの後ろにクラス名を書いて，その後ろに波括弧を付けて，その中に変数やメソッドを書いていきます．C++ のコンストラクタに相当するイニシャライザと呼ばれる関数は，「func」キーワードを付けずに「init」という名前で書きます．クラスのインスタンス変数には，「デフォルトで 0 になる」といったルールはありません．必ず変数の宣言時か，イニシャライザの中か，あるいはその両方で明示的に値を指定する必要があります．

```
class Enemy {
   var x:Double = 0.0

   init() {}
   init(x:Double) {
       self.x = x
   }

   func move() {
       println("x=\(x)")
       x += 2.5
   }
}

class EnemyEx: Enemy {
    override func move() {
        println("x=\(x)")
        x += 50.0
    }
}

var e1 = Enemy()
e1.move()  // "x=0.0"と出力
e1.move()  // "x=2.5"と出力

var e2 = EnemyEx(x:100.0)
e2.move()  // "x=100.0"と出力
e2.move()  // "x=150.0"と出力
```

図 -12　クラス・サブクラスの宣言とその利用

また，サブクラスの宣言時には，クラス名の後ろにコロンを書いて，その後ろに親クラスの名前を書きます．メソッドのオーバライド時には，必ず「override」というキーワードをメソッド宣言の頭に付けなければいけません．このルールによって，意図しないオーバライドや，オーバライドしているつもりで基底クラスにメソッドがないといった事故を防ぐことができます．

図 -12 では，Double 型の X 座標を属性として持っている Enemy クラスとそのサブクラス EnemyEx を定義しています．この例のように，インスタンスの作成は，クラス名の後ろに丸括弧を書くことによって行います．丸括弧の中に引数を書くと，引数付きのイニシャライザが呼び出されます．インスタンス

```
func divide(a:Int, b:Int) -> (quot:Int,
remain:Int) {
    return (a/b, a%b)
}

let div = divide(7, 3)
println(div.quot)    // println(div.0)でも同じ
println(div.remain) // println(div.1)でも同じ
```

図 -13　複数の値をまとめて扱えるタプル

```
let array = [2, 9, 1, 3, 7]
println(array)  // [2, 9, 1, 3, 7]

sort(array)
println(array)  // [1, 2, 3, 7, 9]

sort(array, { $0 > $1 })
println(array)  // [9, 7, 3, 2, 1]
```

図 -14　クロージャによるソートの比較方法の指定

変数のデフォルト値が引数付きのイニシャライザによって上書きされている点，サブクラスのインスタンス作成時に親クラスで定義されているイニシャライザを呼び出している点などに注目してください．

Swift のその他の特徴

Swift のその他の特徴としては，関数のリターン値として複数の値をまとめて返すことのできるタプル（**図 -13**）や，Objective-C のブロックと同様にレキシカルなコード表現を行うためのクロージャ（**図 -14**）などがあります．

また switch 文が，文字列型が条件に指定できたり，より複雑な case の条件記述ができたり，break を書かなくてもフォールスルーしない（フォールスルーさせる場合には明示的に「fallthrough」と書く）など，とても強力になっています．**図 -15** の例では，URL 文字列を元に switch 文で条件分岐させて，「2 つの値に等しい場合」「拡張子が "png" である場合」「文字列が "http://" で始まる場合」の 3 つの分岐を同時に記述しています．switch 文では当てはまる可能性のあるすべての条件を網羅しなければいけませんので，基本的に default 文を必ず書かなければいけません．そのため，一部の条件の成立に対してチェックを忘れていた場合でも，そのことが何かしらの形で目に触れるようになっていることが期待できます．

Swift は，同じプロジェクトの中で C 言語や Objective-C と組み合わせて同時に使用することもできます．過去の資産を活用する上で，これも大きなことです．

```
let urlStr = "http://localhost/test.png"

switch urlStr {
case "about:blank", "http://-:-":
    println("Blank URL")
case let x where x.pathExtension == "png":
    println("PNG file")
case let x where x.hasPrefix("http://"):
    println("HTTP protocol")
default:
    println("No matching rule")
}
```

図 -15　さまざまな条件での switch 文の使用例

Swift をサポートする新しい Xcode にはまた，Playground という対話的なテスト環境が用意されます．Playground の上ではさまざまな種類の値の変化をビジュアルに確認でき，Swift をスクリプティング言語として簡単に使うことができます．Swift はインタープリタ型言語としての側面もコンパイラ型言語としての側面も持ちます．このことは，コードが書きやすくなるだけではなく，開発プロセス自体も今後大きく変化していくことを意味します．

今後の展望

ここまで見てきたように，2014 年になって新しく登場したプログラミング言語である Swift は，iOS アプリの開発にとってかなり強力な武器であると言えるでしょう．Swift の登場によって今後のアプリケーション開発がどのように変化していくのか，今からとても楽しみです．

なお，プログラミング言語 Swift についてのより詳しい情報は，Apple 社の公式 Web ページ https://developer.apple.com/swift/ を参照してください．言語の紹介のほか，Objective-C との連携方法についても情報が載っています．

（2014 年 6 月 13 日受付）

● 沼田 哲史（ぬまた さとし）　Twitter @sazameki

1978 年 1 月生まれ．2005 年大阪大学大学院情報科学研究科より博士号（情報科学）取得．同年より大阪電気通信大学総合情報学部デジタルゲーム学科講師．2009 年より日本英語発音協会監事．2001 年より Mac OS X を主体としたプログラミングを行うようになり，2007 年より iPhone アプリのプログラミングに取り組む．「レベルアップ Objective-C」など，iOS プログラミングの著書多数．現在は大学において，iPhone や iPad を用いたゲーム開発によるプログラミング教育を提案し実践している．趣味は弓道，ヴァイオリン，シンセサイザー演奏．