Swift5 vs Objective C

Bill Kim(김정훈) | ibillkim@gmail.com

목차

개요

본 문서는 Swift5의 주요 특징 및 기본 문법에 대해서

Objective C와 비교를 통하여

간략하게 소개해주는 문서입니다.

크게 개념적 부분과 문법적인 부분에 대해서 두 언어간의

차이점을 살펴봅니다.

문서 후반부에는 Swift 만의 고유한 문법적인 기능에 대해서

소개해드립니다.

개념적 차이

개념적 차이

Swift :

멀티 프로그래밍 패러다임 지원(객체 지향, 함수형, 프로토콜 지향 언어)

Objective C :

C 방식의 스몰토크 스타일의 메시지 구문형 객체 지향 언어

개념적 차이

Swift :

별도의 헤더 파일이 필요없음

Objective C :

헤더(선언) 및 본문(구현) 파일 존재

개념적 차이

Swift :

컴파일러 버전 업에 따른 API 변동이 발생

Objective C :

컴파일 버전 업에 따른 API 변동 없음

개념적 차이

Swift :

Java와 유사한 형태의 간결한 코딩 방식

Objective C :

C 스타일의 [] 타입의 코딩 방식

개념적 차이

Swift :

변수 선언이 명확한 타입은 없어도 됨. 단 초기화에서는 명확하게 이러어저야 함. nil 타입 불가

Objective C :

변수형이 초기에 명확하게 선언하여야 함.

개념적 차이

Swift :

null 포인트 에러 및 할당값 오류를 컴파일 시에서 명확히 에러로 잡아줌

Objective C :

컴파일 시에는 null 포인터 에러 및 구문 오류를 잡아주지는 않음.

개념적 차이

Swift :

타입 세이브(Type-Safe) 언어 방식, 상수 및 변수 타입을 컴파일 시에 바로 체크하여 준다.

Objective C :

컴파일 시에는 변수 타입에 대한 체크는 하지 않아 사용자는 자유로우나 런타임 시에 에러는 사용자의 책임이 따른다.

개념적 차이

Swift :

ARC가 C 네이티브 코드에도 적용 가능하다

Objective C :

C 네이티브 코드에서는 별도로 ARC를 관리하여야 한다.

개념적 차이

Swift :

컴파일 및 앱 실행 속도가 빠른 대신에 앱 사이즈가 더 큼

Objective C :

컴파일도 기존 C보다 느리지만 앱 사이즈는 Swift 대비 작음

문법적 차이

문법적 차이 : 확장자

Swift :

.swift

Objective C :

.h, .m

문법적 차이 : 파일 포함

Swift :

import

Objective C :

#import

문법적 차이 : 스타일

Swift :

모든 구문 뒤에 세미콜론 없음

Objective C :

모든 구문 사이에 [ ] 문자로 감쌈, 구문 끝에는 세미콜론 표시

문법적 차이 : 동적 타입

Swift :

Any : 모든 변수 및 객체에 지정 가능

AnyObject : 객체 타입에 지정 가능

Objective C :

Id : 모든 변수 및 객체에 지정 가능

문법적 차이 : 자료형

Swift :

Bool : true, false 값을 명확하기 선언하여야 함, 0, 1을 사용할 수 없음

Int : +, -, 0을 포함한 64비트 정수형 타입

UInt : 양수 정수형 타입, Int 타입 값을 넣을 수 없음

Float : 32비트 부동 소수형 타입, 정수값 입력 가능

Double : 64비트 부동 소수형 타입, 정수값 입력 가능

Character : 한글자의 문자를 가지는 문자형 타입

String : 여러 문자를 가지는 문자형 타입

문법적 차이 : 자료형

Objective C :

기본적으로 C, C++ 형태의 자료형 사용 가능

BOOL, bool : true, false 값을 위한 자료형 0, 1을 사용할 수 있음

NSInteger : +, -, 0을 포함한 64비트 정수형 타입

NSUInteger : 양수 정수형 타입, Int 타입 값을 넣을 수 없음

CGFloat : 32비트 부동 소수형 타입, 정수값 입력 가능

double : 64비트 부동 소수형 타입, 정수값 입력 가능, Objective C 전용 자료형은 없음

char : 한글자의 문자를 가지는 문자형 타입, Objective C 전용 자료형은 없음

NSString : 여러 문자를 가지는 문자형 타입

문법적 차이 : 문자형

Swift :

String str = “abcd”

Objective C :

NSString = @“abcd”

문법적 차이 : 상수, 변수

Swift :

let, var

Objective C :

const, 기본 자료형 사용

문법적 차이 : 컬렉션

Swift :

Array : 순서가 있는 리스트 컬렉션, 배열과 비슷, 중복 허용

Dictionary : Key, Value로 이루어진 컬렉션, (해쉬)맵과 비슷, 키는 중복 허용 안함, 값은 중복 허용

Set : 순서가 없고 컬렉션 안의 멤버들이 각각 유일한 컬렉션, 중복 허용 안함, 집합 내의 빠르게 각 원소값을 확인할 때 유용

문법적 차이 : 컬렉션

Objective C :

NSArray, NSMutableArray : 순서가 있는 리스트 컬렉션, 값 설정 이후 수정 불가

NSMutableArray : NSArray와 비슷하지만 값 설정 이후 수정 가능

NSDictionary, NSMutableDictionary : Key, Value로 이루어진 컬렉션, 키는 중복 허용 안함, 값은 중복 허용, NSArray와 마찬가지로 NSMutable- 타입도 지원

NSSet, NSMutableSet : 순서가 없고 컬렉션 안의 멤버들이 각각 유일한 컬렉션, 중복 허용 안함, 집합 내의 빠르게 각 원소값을 확인할 때 유용

문법적 차이 : 조건문

Swift :

If : () 안에 조건문을 표현할 수도 있고 안해도 무방, && 연산자 대신에 , 로 나열 가능, 반드시 { } 로 해당 조건문을 감싸야 한다.

guard : if와 비슷하게 조건문이지만 else 구문이 반드시 필요하며 참일 경우 실행 코드 블럭이 없다

Switch : 기존 switch 문과 유사하며 case, default 형식으로 조건 검사, case 문에서 조건(case let(x, y)) 및 범위(case (-3...3, 0))로 분기 가능

문법적 차이

Objective C :

If : () 안에 조건문 표기, 한 줄일 경우 { } 생략 가능

Switch : 기존 C의 switch 문과 유사하며 case, default 형식으로 조건 검사

문법적 차이 : 반복문

Swift :

for 루프 상수 in 루프 구간(조건) {

[코드]

}

루프 순회 표현은 0..2 형식으로 표현 가능

Swift 5에서는 기존에 C 스타일의 형식은 사용 불가

while 루프 조건 {

[코드]

}

문법적 차이 : 반복문

Objective C :

기존 C 스타일 for문 사용 가능, 콜렉션 타입의 경우 아래의 방식 사용 가능

for(루프 상수 : 콜렉션 객체) {

[코드]

}

while(루프 조건) {

[코드]

}

문법적 차이 : 함수

Swift :

```
public func name(name:String) -> String(반환값)
{
    return name
}
```

Objective C :

```
-(NSString *)name:(NSString *)name
{
    return  name;
}
```

문법적 차이 : 구조체

구조체의 경우 Swift와 Objective C에서의 기본 개념은 차이가 없다.

여러 변수를 담을 수 있는 컨테이너, 프로퍼티와 메서드 사용 가능, 초기화 가능, 확장(extension) 가능, 프로토콜 사용 가능,

subscript(특정 member elements에 간단하게 접근할 수 있는 문법) 사용이 가능,

상속 불가

문법적 차이 : 클래스

클래스의 경우도 Swift와 Objective C에서의 기본 개념은 차이가 없다.

클래스는 구조체와 거의 비슷한 하나의 객체를 표현하는 단위,

구조체는 값 타입, 클래스는 참조 타입, 구조체는 상속 불가, 타입캐스팅은 클래스의 인스턴스에만 허용, 디이니셜라이저는 클래스의 인스턴스만 활용 가능,

참조 획수 계산은 클래스의 인스턴스에만 적용

문법적 차이 : Protocol

Protocol : Interface 클래스와 비슷, 프로퍼티({get set}), 멤버, 메서드등을 정의함 함, 구현은 할 수 없음, 프로토콜 간의 단일 및 다중 상속 가능

Objective C도 Protocol이 존재하면 비슷한 방식으로 사용 가능

Protocol 사용예

Swift 사용예 :

```
protocol Human{
var name:String
}

Public class Man : Human {
var name:String
var age:Int
}
```

문법적 차이 : Extension

Extension : 기존 클래스를 확장하는 기능, 새로운 이니셜라이저 제공 가능, 서브스크립트 정의, 인스턴스 및 타입 메소드 정의 가능, 새로운 중첩 타입 정의 및 사용 가능

Swift나 Objective C에서의 Extension은 개념적으로 크게 차이가 없음

문법적 차이 : enum

Swift :

```
enum Number {
case one
case two
case three
}
```

문법적 차이 : enum

Objective C :

C 스타일과 동일

```
enum  Number {
one = 1,
two = 2,
three = 3
};
```

Swift 고유 문법

Swift : Property

저장 프로퍼티 : 인스턴스의 변수 또는 상수를 의미

지연(lazy) 프로퍼티 : 호출이 될 경우에 값을 초기화한다. 메모리나 성능 상에서 효율적일 수 있다

연산(Computed) 프로퍼티 : 특정 연산을 수행한 결과의 값을 의미한다. 실제 값을 저장하는게 아니라 특정 상태에 따른 값을 연산하는 프로퍼티이다. getter, setter 역할도 수행한다.

타입(Type) 프로퍼티 : 인스턴스가 아닌 타입 자체에 속하게 되는 프로퍼티, 인스턴스의 생성 여부와 상관없이 타입 프로퍼티의 값은 하나이다. Class 타입, static 타입, 저장 타입 프로퍼티 등이 있다.

프로퍼티 감시자(Property Observers)

프로퍼티의 값이 변할 때 변화에 따른 특정 액션을 수행한다. 저장 프로포티에 적용 가능하며 지연 프로퍼티에는 적용불가하다. willSet, didSet 등의 액션이 있다.

Swift : 접근제한자

private : 클래스 까지 허용

fileprivate : 클래스 및 파일 범위까지 허용(Swift 3.0 이상)

Internal : 같은 모듈(프로젝트)까지 허용

public : 모듈 외부까지 허용

open : 모듈 외부까지 허용, 접근 및 수정 가능(Swift 3.0 이상)

Swift : Optional

상수 및 변수 선언 끝에 아래의 구문자를 붙여서 구분할 수 있다. Swift 언어

? : 값이 있을 수도 없을 수도 있다고 알려주는 구문자

! : 값이 무조건 있다고 알려주는 구문자

사용 이유 : nil의 가능성을 명시적으로 표현 가능, 전달받은 값이 옵셔널이 아니라면 nil 체크를 하지 않아도 되므로 안심하고 사용 가능(안전하고 효율적인 코딩 유지)

Swift : if-let

if-let은 옵셔널 타입의 값을 안전하게 추출하기 위해서 사용

if-let을 통해 참조 타입 변수를 추출해도 참조는 유지(같은 주소값 사용)된다.

if-let 사용예

```
Var value:String? = “String Value”

Print(value) // “String Value”

If let checkValue = value {

print(checkValue) // “String Value”

}

If let value = value {

Value = “New Value” // Error

}
```

Swift : Type-Casting

타입 확인 : is를 사용하여 타입을 확인

업 캐스팅 : as를 사용하여 부모 클래스로 캐스팅

다운 캐스팅 : as?(조건), as!(강제) 등을 사용하여 자식 클래스로 캐스팅

Type-Casting 사용예

```
class Human {
var name;
}
class Man {
}
class Woman {
}
var human = Human()
var man = Man()
var woman = Woman()
```

Type-Casting 사용예

```
var bill = human is Human // OK
var Joyce = human is Man // Fail

Var Paul:Human = Main() as Human
Var jinx:Any = Human() // as 생략 가능

var jorge = Paul as? Human // OK
var bob = man as?  Human // nil

var mike as! paul as! Human // OK
var Jenny as! woman as! Woman // Error
```

Swift : Guard

guard를 사용하여 값을 전달 시 잘못된 값이면 특정 실행 구문을 빠르게 종료 가능, 항상 else와 매칭하여 사용,

else 블럭에는 반드시 종료 지시자(return, break 등)가 있어야 한다.

Swift : Generic

C++ 에서의 탬플릿과 같은 개념

Array 및 Dictionary 또한 Generic 콜렉션

타입 제약(Type Constraint) 설정 가능 : 특정한 타입만 사용할 수 있도록 제한을 걸 수 있음

Swift : Generic

```
func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}

func swapTwoValues<T : SomeClass>(_ a: inout T, _ b:
inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

References

References

[1] Swift 문법 기초 : https://velog.io/@wimes/2019-11-01-2111-작성됨-3zk2g3ey9k

[2] [Swift] 기본 문법 : https://velog.io/@max9106/Swift-기본-문법-1

[3] Swift(프로그래밍 언어) : https://namu.wiki/w/Swift(프로그래밍%20언어)

[4] Swift Github : https://github.com/apple/swift

[5] Swift 정보 링크 : https://medium.com/@jang.wangsu/swift-swift-처음-시작하는-개발자-분들을-위한-한국어-동영상-강의-정보를-모아놓은-링크-151dae076b14

[6] Swift란 어떤 언어인가? : https://post.naver.com/viewer/postView.nhn?volumeNo=6165588&memberNo=34635212

[7] swift.org : https://swift.org

[8] Swift는 어떤 개발언어인가? : https://brunch.co.kr/@myoungjun/24

[9] What's new in Swift 5.1 : https://kka7.tistory.com/354

[10] Swift 문법정리 : http://tech.inswave.com/2018/04/02/Swift/

References

[11] Swift 접근제어 : https://baked-corn.tistory.com/80

[12] Swift UI란 무엇인가 : https://g-y-e-o-m.tistory.com/149

[13] Swift5에서는 뭐가 바뀌었을까? : https://seorenn.tistory.com/9

[14] (Swift) 시퀀스와 관련된 Swift 표준 함수들 : https://sooooprmx.com/archives/7059

[15] Swift 코스 : http://www.w3big.com/ko/swift/default.html

Thank you!

Enjoy with Swift!