SWIFT
Closure

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Closure?

- 클로저는 사용자의 코드 안에서 전달되어 사용할 수 있는 로직을 가진 중괄호“{}”로 구분된 코드의 블럭이며, 일급 객체의 역할을 할 수 있다.

- 일급 객체는 전달 인자로 보낼 수 있고, 변수/상수 등으로 저장하거나 전달할 수 있으며, 함수의 반환 값이 될 수도 있다.

- 클로저는 참조 타입(Reference Types)이다.

- 함수는 클로저의 한 형태로, 이름이 있는 클로저이다.

Closure vs Function

Closure

1) 이름이 없다.
2) func 키워드가 존재하지 않는다.
3) in 키워드가 존재한다.

Function

1) 이름이 있다.
2) func 키워드가 존재한다.
3) in 키워드가 존재하지 않는다.

First-Citizen

일급 객체(First Citizen)라는 말은 함수형 프로그래밍에서 자주 접할 수 있는 단어로서 아래의 3가지 조건을 충족하면 일급 객체라고 할 수 있습니다.

1. 변수나 데이터로 할당할 수 있다.
2. 객체의 인자로서 넘길 수 있다.
3. 객체의 반환값으로서 리턴할 수 있다.

Swift에서 함수는 일급 객체입니다.
함수는 인자, 반환값, 상수나 변수로 할당할 수 있기 때문입니다.

Closure Expressions

기본 클로저 표현 방식은 아래와 같습니다.

```
{ (인자들) -> 반환타입 in
  로직 구현
}
```

사용 예시)

아래와 같이 함수로 따로 정의된 형태가 아닌 인자로 들어가 있는 형태의 클로저를 Inline Closure라고 부른다.

```
let reverse = names.sorted(by:
{ (s1: String, s2: String) -> Bool in
   return s1 > s2
}
)
```

Closure Shorthand

기본 클로저 표현 방식에서 아래와 같이 다양한 방식으로의 축약을 통하여 클로저 표현을 간소화할 수 있습니다.

타입 생략

```
let reverse = names.sorted(by: { s1, s2 in return s1 > s2 })
```

반환타입 생략

```
let reverse = names.sorted(by: { s1, s2 in s1 > s2 })
```

인자이름 생략

```
let reversed = names.sorted(by: { $0 > $1 })
```

연산자 메소드

```
let reversed = names.sorted(by: > )
```

Closures are Reference Types

기본적으로 클로저는 Reference Type입니다.

```
var a:Int = 5
var b:Int = 6

var closure =  { print(a, b) }

closure() // 5 6이 찍힘

a = 0 // a의 값을 0으로 변경
b = 0 // b의 값을 0으로 변경

closure() // 0 0이 찍힘
```

closure 내부의 a, b가 외부의 a, b 값을 참조하고 있기 때문에 값의 변화가 생길 수 있다.

이러한 Reference Type의 특성으로인해 원치않는 값의 변화가 생길 수 있습니다.

Reference Types 예시

```
var index:Int = 0
var closureArr:[()->()] = []

for _ in 1...5
{
   closureArr.append( {print(index)} )
   index += 1
}

for i in 0...4
{
   closureArr[i]()
}

// closureArr의 값은 5, 5, 5, 5, 5로 출력됨
// index 값을 closure 함수가 참조하고 있어서 최종 index 값인 5로만 closure 에서는 인식된다.
```

Trailing Closure

인자로 클로저를 넣기가 길경우 후행 클로저를 사용하여 함수의 뒤에 표현할 수 있는 기법이다.

```
let reversed = names.sorted() { $0 > $1 }
```

함수의 마지막 인자가 클로저이고, 후행 클로저를 사용하면 괄호“( )”를 생략할 수 있다.

```
let reversed = names.sorted { $0 > $1 }
let reversed = names.sorted { (s1: String, s2: String) ->
Bool in return s1 > s2 }
```

Escaping Closure

클로저가 함수의 인자로 전달되지만 함수 밖에서 실행되는 것(함수가 반환된 후 실행되는 것)을 Escape한다고 하며, 이러한 경우 매개변수의 타입 앞에 @escaping이라는 키워드를 명시해야한다.

다음과 같은 경우에 자주 사용된다.

- 비동기로 실행되는 경우
- completionHandler(완료에 따른 처리)로 사용되는 클로저의 경우

일반 지역변수가 함수 밖에서 살아있는 것은 전역변수를 함수에 가져와서 값을 주는 것과 다름이 없지만, 클로저의 Escaping은 하나의 함수가 마무리된 상태에서만 다른 함수가 실행되도록 함수를 작성할 수 있다는 점에서 유리하다.

즉, 이를 활용해서 함수 사이에 실행 순서를 정할 수 있다.

Escaping Closure

사용 예시)

```
var completionHandlers: [() -> Void] = []
func
someFunctionWithEscapingClosure(completionHandler:
@escaping () -> Void)
{
    completionHandlers.append(completionHandler)
}
```

위 함수에서 인자로 전달된 completionHandler는 someFunctionWithEscapingClosure 함수가 끝나고 나중에 처리된다.

만약 함수가 끝나고 실행되는 클로저에 @escaping키워드를 붙이지 않으면 컴파일시 오류가 발생한다.

Escaping Closure

사용 예시)

```
var completionHandlers: [() -> Void] = []

func someFunctionWithEscapingClosure(completionHandler: @escaping () -> Void) {
  completionHandlers.append(completionHandler)
}
```

```
func someFunctionWithNonescapingClosure(closure: () -> Void) {
  closure()    // 함수 안에서 끝나는 클로저
}

class SomeClass {
  var x = 10
  func doSomething() {
    someFunctionWithEscapingClosure { self.x = 100 } // 명시적으로 self를 적어줘야 한다.
    someFunctionWithNonescapingClosure { x = 200 }
  }
}

let instance = SomeClass()
instance.doSomething()
print(instance.x) // 200
completionHandlers.first?()
print(instance.x) // 100
```

AutoClosure

자동 클로저는 인자 값이 없으며 특정 표현을 감싸서 다른 함수에 전달 인자로 사용할 수 있는 클로저를 말한다.

즉 함수의 인자로 전달되는 표현을 알아서 클로저로 변환하는 클로저를 자동 클로저라고 합니다.

자동 클로저는 클로저가 호출되기 전까지 내부의 코드를 동작하지 않습니다.

따라 연산을 지연시킬 수 있다는 장점이 있습니다.

AutoClosure

사용 예시)

```
var customersInLine = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

print(customersInLine.count)
// Prints "5" let

let customerProvider = {
   customersInLine.remove(at: 0)
}
print(customersInLine.count)
// Prints "5"

print("Now serving \(customerProvider())!")
// Prints "Now serving Chris!"

print(customersInLine.count) // Prints "4"

// customersInLine is ["Alex", "Ewa", "Barry", "Daniella"]
func serve(customer customerProvider: @autoclosure () -> String) {
    print("Now serving \(customerProvider())!")
}

// comtomerProvider 매개변수의 타입이 @autoclosure 속성으로 되어있기 때문에 그 인자는 자동으로 클로저로 변환된다. 즉 중괄호를 생략할 수 있다.

serve(customer: customersInLine.remove(at: 0))
// Prints "Now serving Alex!"
```

AutoClosure

사용 예시) AutoClosure + Escaping Closure

```swift
var customersInLine = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

// customersInLine is ["Barry", "Daniella"]
var customerProviders: [() -> String] = []        //  클로저를 저장하는 배열을 선언

func collectCustomerProviders(_ customerProvider: @autoclosure @escaping () -> String) {
    customerProviders.append(customerProvider)
} // 클로저를 인자로 받아 그 클로저를 customerProviders 배열에 추가하는 함수를 선언

collectCustomerProviders(customersInLine.remove(at: 0)) // 클로저를 customerProviders 배열에 추가
collectCustomerProviders(customersInLine.remove(at: 0))

print("Collected \(customerProviders.count) closures.")
// Prints "Collected 2 closures."
// 2개의 클로저가 추가 됨

for customerProvider in customerProviders {
   print("Now serving \(customerProvider())!”) // 클로저를 실행하면 배열의 0번째 원소를 제거하며 그 값을 출력
   // Prints "Now serving Chris!"
   // Prints "Now serving Alex!"
}

// Prints "Now serving Barry!"
// Prints "Now serving Daniella!"
```

Capturing Values

클로저는 특정 문맥의 상수나 변수의 값을 캡쳐할 수 있다.

즉 원본 값이 사라져도 클로저의 body 안 에서 그 값을 활용할 수 있다.

함수와 클로저를 상수나 변수에 할당할 때 실제로는 상수와 변수에 해당 함수나 클로저의 참조가 할당된다.

Capturing Values

만약 한 클로저를 두 상수나 변수에 할당하면 그 두 상수나 변수는 같은 클로저를 참조하고 있게되는 것이다.

만약 클로저를 어떤 클래스 인스턴스의 프로퍼티로 할당하고 그 클로저가 그 인스턴스를 캡쳐하면 강한 순환 참조에 빠지게 된다.

즉. 인스턴스의 사용이 끝나도 메모리를 해제하지 못하는 것이다. 그래서 swift는 이 문제를 다루기 위해 캡쳐 리스트를 사용한다.

Capturing Values

사용 예시)

```
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
          runningTotal += amount
          return runningTotal
    }
    return incrementer
}
```

```
let plusTen = makeIncrementer(forIncrement: 10)
let plusSeven = makeIncrementer(forIncrement: 7)

// 함수가 각기 실행되어도 실제로는 변수 runnigTotal과 amount가 캡쳐되서
// 그 변수를 공유하기 때문에 누적된 결과를 가진다.
let plusedTen = plusTen()
print(plusedTen) // 10

let plusedTen2 = plusTen()
print(plusedTen2) // 20

// 다른 클로저이기 때문에 고유의 저장소에 runningTotal과 amount를 캡쳐해서 사용한다.
let plusedSeven = plusSeven()
print(plusedSeven) // 7

let plusedSeven2 = plusSeven()
print(plusedSeven2) // 14
```

References

[1] 클로저 : https://yagom.github.io/swift_basic/contents/12_closure/

[2] 오늘의 Swift 상식 (Closure) : https://medium.com/@jgj455/오늘의-swift-상식-closure-aa401f76b7ce

[3] [Swift] Closure [02] : https://baked-corn.tistory.com/25

[4] Closure 축약하기 : https://programmers.co.kr/learn/courses/4/lessons/560

[5] Swift 클로저(Closure) : https://nightohl.tistory.com/entry/Swift-클로저Closure

References

[6] Swift - 클로저 & 고차함수 : https://medium.com/@ggaa96/swift-study-1-클로저-고차함수-abe199f22ad8

[7] Closures : https://docs.swift.org/swift-book/LanguageGuide/Closures.html

[8] [Swift문법] 함수(Function) - 일급 객체로서의 함수 : https://yzzzzun.tistory.com/13

[9] Swift - 클로저 : https://penguin-story.tistory.com/38

[10] [Swift] Closure [01] : https://baked-corn.tistory.com/11

References

[11] [Swift 3] 클로저 (Closures) : https://beankhan.tistory.com/158

Thank you!