函数

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

实力IT教育 www.520it.com

小码哥教育@M了个J

函数的定义

```swift
func pi() -> Double {
    return 3.14
}
func sum(v1:  Int, v2: Int) -> Int {
    return v1 + v2
}
sum(v1: 10, v2: 20)
```

- 形参默认是let，也只能是let

- 无返回值

```swift
func sayHello() -> Void {
    print("Hello")
}
```

```swift
func sayHello() -> () {
    print("Hello")
}
```

```swift
func sayHello() {
    print("Hello")
}
```

小码哥教育 SEEMYGO

隐式返回（Implicit Return）

- 如果整个函数体是一个单一表达式，那么函数会隐式返回这个表达式

```
func sum(v1: Int, v2: Int) -> Int {
    v1 + v2
}
sum(v1: 10, v2: 20) // 30
```

小码哥教育@MJ了^

返回元组：实现多返回值

```swift
func calculate(v1:  Int, v2: Int) -> (sum: Int, difference: Int, average: Int) {
    let sum = v1 + v2
    return (sum, v1 - v2, sum >> 1)
}
let result = calculate(v1: 20, v2: 10)
result.sum // 30
result.difference // 10
result.average // 15
```

小码哥教育 SEEMYGO

函数的文档注释

```swift
/// 求和【概述】
///
/// 将2个整数相加【更详细的描述】
///
/// - Parameter v1: 第1个整数
/// - Parameter v2: 第2个整数
/// - Returns: 2个整数的和
///
/// - Note:传入2个整数即可【批注】
///
func sum(v1: Int, v2: Int) -> Int {
    v1 + v2
}
```

Summary

求和【概述】

Declaration

```swift
func sum(v1: Int, v2: Int) -> Int
```

Discussion

将2个整数相加【更详细的描述】

Note

传入2个整数即可【批注】

Parameters

v1 第1个整数

v2 第2个整数

Returns

2个整数的和

Declared In

03-函数.xcplaygroundpage

```swift
95 func sum(v1: Int, v2: Int) -> Int {
96     v1 + v2
97 }
```

- 参考：https://swift.org/documentation/api-design-guidelines/

小码哥教育 SEEMYGO

参数标签（Argument Label）

■ 可以修改参数标签

```
func goToWork(at time: String) {
    print("this time is \(time)")
}
goToWork(at: "08:00")
// this time is 08:00
```

■ 可以使用下划线 _ 省略参数标签

```
func sum(_ v1:  Int, _ v2: Int) -> Int {
    v1 + v2
}
sum(10, 20)
```

小码哥教育@M了个J

默认参数值（Default Parameter Value）

- 参数可以有默认值

```
func check(name: String = "nobody", age: Int, job: String = "none") {
    print("name=\(name), age=\(age), job=\(job)")
}
check(name: "Jack", age: 20, job: "Doctor") // name=Jack, age=20, job=Doctor
check(name: "Rose", age: 18) // name=Rose, age=18, job=none
check(age: 10, job: "Batman") // name=nobody, age=10, job=Batman
check(age: 15) // name=nobody, age=15, job=none
```

- C++的默认参数值有个限制：必须从右往左设置。由于Swift拥有参数标签，因此并没有此类限制
- 但是在省略参数标签时，需要特别注意，避免出错

```
// 这里的middle不可以省略参数标签
func test(_ first: Int = 10, middle: Int, _ last: Int = 30) { }
test(middle: 20)
```

可变参数（Variadic Parameter）

```swift
func sum(_ numbers: Int...) -> Int {
    var total = 0
    for number in numbers {
        total += number
    }
    return total
}
sum(10, 20, 30, 40) // 100
```

- 一个函数**最多只能有1个**可变参数
- 紧跟在可变参数后面的参数不能省略参数标签

```swift
// 参数string不能省略标签
func test(_ numbers: Int..., string: String, _ other: String) { }
test(10, 20, 30, string: "Jack", "Rose")
```

小码哥教育 SEEMYGO

Swift自带的print函数

```swift
/// - Parameters:
///   - items: Zero or more items to print.
///   - separator: A string to print between each item. The default is a single space (`" "`).
///   - terminator: The string to print after all items have been printed. The
///     default is a newline (`"\n"`).
public func print(_ items: Any..., separator: String = " ", terminator: String = "\n")
```

```swift
print(1, 2, 3, 4, 5) // 1 2 3 4 5
```

```swift
print(1, 2, 3, 4, 5, separator: "_") // 1_2_3_4_5
```

```swift
print("My name is Jake.", terminator: "")
print("My age is 18.")
// My name is Jake.My age is 18.
```

输入输出参数（In-Out Parameter）

- 可以用inout定义一个输入输出参数：可以在函数内部修改外部实参的值

```
func swapValues(_ v1: inout Int, _ v2: inout Int) {
    let tmp = v1
    v1 = v2
    v2 = tmp
}
var num1 = 10
var num2 = 20
swapValues(&num1, &num2)
```

```
func swapValues(_ v1: inout Int, _ v2: inout Int) {
    (v1, v2) = (v2, v1)
}
```

- 可变参数不能标记为inout
- inout参数不能有默认值
- inout参数只能传入可以被多次赋值的
- inout参数的本质是地址传递（引用传递）

小码哥教育 SEEMYGO

函数重载（Function Overload）

■ 规则
- 函数名相同
- 参数个数不同 || 参数类型不同 || 参数标签不同

```swift
func sum(v1: Int, v2: Int) -> Int {
    v1 + v2
}
```

```swift
func sum(v1: Int, v2: Int, v3: Int) -> Int {
    v1 + v2 + v3
} // 参数个数不同
```

```swift
func sum(v1: Int, v2: Double) -> Double {
    Double(v1) + v2
} // 参数类型不同
func sum(v1: Double, v2: Int) -> Double {
    v1 + Double(v2)
} // 参数类型不同
```

```swift
func sum(_ v1: Int, _ v2: Int) -> Int {
    v1 + v2
} // 参数标签不同
func sum(a: Int, b: Int) -> Int {
    a + b
} // 参数标签不同
```

```swift
sum(v1: 10, v2: 20) // 30
sum(v1: 10, v2: 20, v3: 30) // 60
sum(v1: 10, v2: 20.0) // 30.0
sum(v1: 10.0, v2: 20) // 30.0
sum(10, 20) // 30
sum(a: 10, b: 20) // 30
```

函数重载注意点

- 返回值类型与函数重载无关

```
func sum(v1: Int, v2: Int) -> Int { v1 + v2 }
func sum(v1: Int, v2: Int) { }
sum(v1: 10, v2: 20)        Ambiguous use of 'sum(v1:v2:)'
```

- 默认参数值和函数重载一起使用产生二义性时，编译器并不会报错（在C++中会报错）

```
func sum(v1: Int, v2: Int) -> Int {
    v1 + v2
}
func sum(v1:  Int, v2: Int, v3: Int = 10) -> Int {
    v1 + v2 + v3
}
// 会调用sum(v1: Int, v2: Int)
sum(v1: 10, v2: 20)
```

小码哥教育 SEEMYGO

函数重载注意点

- 可变参数、省略参数标签、函数重载一起使用产生二义性时，编译器有可能会报错

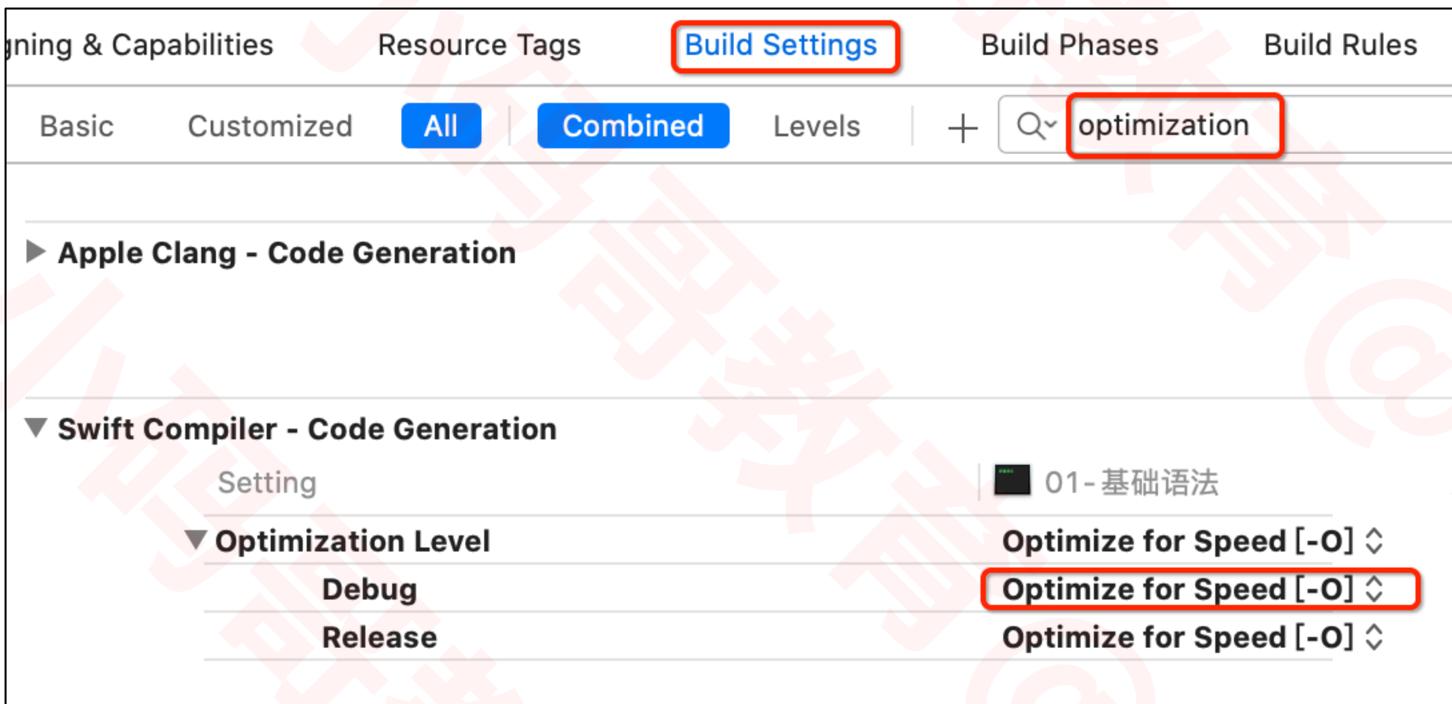
```
func sum(v1: Int, v2: Int) -> Int {
    v1 + v2
}
func sum(_ v1: Int, _ v2: Int) -> Int {
    v1 + v2
}
func sum(_ numbers: Int...) -> Int {
    var total = 0
    for number in numbers {
        total += number
    }
    return total
}
// error: ambiguous use of 'sum'
sum(10, 20)
```

小码哥教育 SEEMYGO

内联函数（Inline Function）

■ 如果开启了编译器优化（Release模式默认会开启优化），编译器会自动将某些函数变成内联函数

□ 将函数调用展开成函数体



■ 哪些函数不会被自动内联？

□ 函数体比较长

□ 包含递归调用

□ 包含动态派发

□ ......

@inline

```swift
// 永远不会被内联（即使开启了编译器优化）
@inline(never) func test() {
    print("test")
}
```

```swift
// 开启编译器优化后，即使代码很长，也会被内联（递归调用函数、动态派发的函数除外）
@inline(__always) func test() {
    print("test")
}
```

- 在Release模式下，编译器已经开启优化，会自动决定哪些函数需要内联，因此没必要使用@inline

函数类型（Function Type）

- 每一个函数都是有类型的，函数类型由形式参数类型、返回值类型组成

```
func test() { } // () -> Void 或者 () -> ()
```

```
func sum(a: Int, b: Int) -> Int {
    a + b
} // (Int, Int) -> Int
```

```
// 定义变量
var fn: (Int, Int) -> Int = sum
fn(2, 3) // 5，调用时不需要参数标签
```

小码哥教育 SEEMYGO

小码哥教育 SEEMYGO

函数类型作为函数参数

```swift
func sum(v1: Int, v2: Int) -> Int {
    v1 + v2
}
func difference(v1: Int, v2: Int) -> Int {
    v1 - v2
}
func printResult(_ mathFn: (Int, Int) -> Int, _ a: Int, _ b: Int) {
    print("Result: \(mathFn(a, b))")
}
printResult(sum, 5, 2) // Result: 7
printResult(difference, 5, 2) // Result: 3
```

小码哥教育 SEEMYGO

函数类型作为函数返回值

```
func next(_ input: Int) -> Int {
    input + 1
}
func previous(_ input: Int) -> Int {
    input - 1
}
func forward(_ forward: Bool) -> (Int) -> Int {
    forward ? next : previous
}
forward(true)(3) // 4
forward(false)(3) // 2
```

- 返回值是函数类型的函数，叫做高阶函数（Higher-Order Function）

小码哥教育@M了个J

typealias

- typealias用来给类型起别名

```
typealias Byte = Int8
typealias Short = Int16
typealias Long = Int64
```

```
typealias Date = (year: Int, month: Int, day: Int)
func test(_ date: Date) {
    print(date.0)
    print(date.year)
}
test((2011, 9, 10))
```

```
typealias IntFn = (Int, Int) -> Int

func difference(v1: Int, v2: Int) -> Int {
    v1 - v2
}
let fn: IntFn = difference
fn(20, 10) // 10

func setFn(_ fn: IntFn) { }
setFn(difference)

func getFn() -> IntFn { difference }
```

- 按照Swift标准库的定义，Void就是空元组()

```
public typealias Void = ()
```

嵌套函数（Nested Function）

■ 将函数定义在函数内部

```swift
func forward(_ forward: Bool) -> (Int) -> Int {
    func next(_ input: Int) -> Int {
        input + 1
    }
    func previous(_ input: Int) -> Int {
        input - 1
    }
    return forward ? next : previous
}
forward(true)(3) // 4
forward(false)(3) // 2
```