Swift 方法 Swift 继承

Swift 下标脚本

下标脚本 可以定义在类（Class）、结构体（structure）和枚举（enumeration）这些目标中，可以认为是访问对象、集合或序列的快捷方式，不需要再调用实例的特定的赋值和访问方法。

举例来说，用下标脚本访问一个数组(Array)实例中的元素可以这样写 someArray[index] ，访问字典(Dictionary)实例中的元素可以这样写 someDictionary[key]。

对于同一个目标可以定义多个下标脚本，通过索引值类型的不同来进行重载，而且索引值的个数可以是多个。

下标脚本语法及应用

语法

下标脚本允许你通过在实例后面的方括号中传入一个或者多个的索引值来对实例进行访问和赋值。

语法类似于实例方法和计算型属性的混合。

与定义实例方法类似，定义下标脚本使用subscript关键字，显式声明入参（一个或多个）和返回类型。

与实例方法不同的是下标脚本可以设定为读写或只读。这种方式又有点像计算型属性的getter和setter：

```
subscript(index: Int) -> Int {
    get {
        // 用于下标脚本值的声明
    }
    set(newValue) {
        // 执行赋值操作
    }
}
```

实例 1

```
import Cocoa

struct subexample {
    let decrementer: Int
    subscript(index: Int) -> Int {
        return decrementer / index
    }
}
let division = subexample(decrementer: 100)

print("100 除以 9 等于 \(division[9])")
print("100 除以 2 等于 \(division[2])")
print("100 除以 3 等于 \(division[3])")
```

```
print("100 除以 5 等于 \(division[5])")
print("100 除以 7 等于 \(division[7])")
```

以上程序执行输出结果为：

```
100 除以 9 等于 11
100 除以 2 等于 50
100 除以 3 等于 33
100 除以 5 等于 20
100 除以 7 等于 14
```

在上例中，通过 subexample 结构体创建了一个除法运算的实例。数值 100 作为结构体构造函数传入参数初始化实例成员 decrementer。

你可以通过下标脚本来得到结果，比如 division[2] 即为 100 除以 2。

实例 2

```
import Cocoa

class daysofaweek {
    private var days = ["Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "saturday"]
    subscript(index: Int) -> String {
        get {
            return days[index]   // 声明下标脚本的值
        }
        set(newValue) {
            self.days[index] = newValue   // 执行赋值操作
        }
    }
}
var p = daysofaweek()

print(p[0])
print(p[1])
print(p[2])
print(p[3])
```

以上程序执行输出结果为：

```
Sunday
Monday
Tuesday
Wednesday
```

用法

根据使用场景不同下标脚本也具有不同的含义。

通常下标脚本是用来访问集合（collection），列表（list）或序列（sequence）中元素的快捷方式。

你可以在你自己特定的类或结构体中自由的实现下标脚本来提供合适的功能。

例如，Swift 的字典（Dictionary）实现了通过下标脚本对其实例中存放的值进行存取操作。在下标脚本中使用和字典索引相同类型的值，并且把一个字典值类型的值赋值给这个下标脚来为字典设值：

```
import Cocoa

var numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
numberOfLegs["bird"] = 2

print(numberOfLegs)
```

以上程序执行输出结果为：

```
["ant": 6, "bird": 2, "cat": 4, "spider": 8]
```

上例定义一个名为numberOfLegs的变量并用一个字典字面量初始化出了包含三对键值的字典实例。numberOfLegs的字典存放值类型推断为Dictionary。字典实例创建完成之后通过下标脚本的方式将整型值2赋值到字典实例的索引为bird的位置中。

下标脚本选项

下标脚本允许任意数量的入参索引，并且每个入参类型也没有限制。

下标脚本的返回值也可以是任何类型。

下标脚本可以使用变量参数和可变参数。

一个类或结构体可以根据自身需要提供多个下标脚本实现，在定义下标脚本时通过传入参数的类型进行区分，使用下标脚本时会自动匹配合适的下标脚本实现运行，这就是**下标脚本的重载**。

```
import Cocoa

struct Matrix {
    let rows: Int, columns: Int
    var print: [Double]
    init(rows: Int, columns: Int) {
        self.rows = rows
        self.columns = columns
        print = Array(repeating: 0.0, count: rows * columns)
    }
    subscript(row: Int, column: Int) -> Double {
        get {
            return print[(row * columns) + column]
        }
```

```
        set {
            print[(row * columns) + column] = newValue
        }
    }
}
// 创建了一个新的 3 行 3 列的Matrix实例
var mat = Matrix(rows: 3, columns: 3)

// 通过下标脚本设置值
mat[0,0] = 1.0
mat[0,1] = 2.0
mat[1,0] = 3.0
mat[1,1] = 5.0

// 通过下标脚本获取值
print("\(mat[0,0])")
print("\(mat[0,1])")
print("\(mat[1,0])")
print("\(mat[1,1])")
```

以上程序执行输出结果为：

```
1.0
2.0
3.0
5.0
```

Matrix 结构体提供了一个两个传入参数的构造方法，两个参数分别是rows和columns，创建了一个足够容纳rows * columns个数的Double类型数组。为了存储，将数组的大小和数组每个元素初始值0.0。

你可以通过传入合适的row和column的数量来构造一个新的Matrix实例。

← Swift 方法 Swift 继承 →