结构体和类

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

实力IT教育 www.520it.com

结构体

- 在 Swift 标准库中，绝大多数的公开类型都是结构体，而枚举和类只占很小一部分
  - 比如Bool、Int、Double、 String、Array、Dictionary等常见类型都是结构体

```
① struct Date {
②     var year: Int
③     var month: Int
④     var day: Int
⑤ }
⑥ var date = Date(year: 2019, month: 6, day: 23)
```

- 所有的结构体都有一个编译器自动生成的初始化器（initializer，初始化方法、构造器、构造方法）
  - 在第⑥行调用的，可以传入所有成员值，用以初始化所有成员（存储属性，Stored Property）

小码哥教育 SEEMYGO

小码哥教育 SEEMYGO

结构体的初始化器

- 编译器会根据情况，可能会为结构体生成多个初始化器，宗旨是：保证所有成员都有初始值

```
struct Point {
    var x: Int
    var y: Int
}
var p1 = Point(x: 10, y: 10)
var p2 = Point(y: 10)     2 Missing argument for parameter 'x' in call
var p3 = Point(x: 10)     2 Missing argument for parameter 'y' in call
var p4 = Point()          2 Missing argument for parameter 'x' in call
```

```
struct Point {
    var x: Int = 0
    var y: Int
}
var p1 = Point(x: 10, y: 10)
var p2 = Point(y: 10)
var p3 = Point(x: 10)
var p4 = Point()
```

```
struct Point {
    var x: Int
    var y: Int = 0
}
var p1 = Point(x: 10, y: 10)
var p2 = Point(y: 10)
var p3 = Point(x: 10)
var p4 = Point()
```

```
struct Point {
    var x: Int = 0
    var y: Int = 0
}
var p1 = Point(x: 10, y: 10)
var p2 = Point(y: 10)
var p3 = Point(x: 10)
var p4 = Point()
```

小码哥教育 SEEMYGO

思考：下面代码能编译通过么？

```
struct Point {
    var x: Int?
    var y: Int?
}
var p1 = Point(x: 10, y: 10)
var p2 = Point(y: 10)
var p3 = Point(x: 10)
var p4 = Point()
```

- 可选项都有个默认值nil
- 因此可以编译通过

小码哥教育@M了个J

小码哥教育 SEEMYGO

自定义初始化器

- 一旦在定义结构体时自定义了初始化器，编译器就不会再帮它自动生成其他初始化器

```
struct Point {
    var x: Int = 0
    var y: Int = 0
    init(x: Int, y: Int) {
        self.x = x
        self.y = y
    }
}
var p1 = Point(x: 10, y: 10)
var p2 = Point(y: 10)
var p3 = Point(x: 10)
var p4 = Point()
```

小码哥教育@M了个J

窥探初始化器的本质

■ 以下2段代码完全等效

```
struct Point {
    var x: Int = 0
    var y: Int = 0
}
var p = Point()
```

```
struct Point {
    var x: Int
    var y: Int
    init() {
        x = 0
        y = 0
    }
}
var p = Point()
```

小码哥教育 SEEMYGO

结构体内存结构

```swift
struct Point {
    var x: Int = 0
    var y: Int = 0
    var origin: Bool = false
}
print(MemoryLayout<Point>.size) // 17
print(MemoryLayout<Point>.stride) // 24
print(MemoryLayout<Point>.alignment) // 8
```

小码哥教育 SEEMYGO

类

- 类的定义和结构体类似，但编译器并没有为类自动生成可以传入成员值的初始化器

```swift
class Point {
    var x: Int = 0
    var y: Int = 0
}
let p1 = Point()
let p2 = Point(x: 10, y: 20)
let p3 = Point(x: 10)
let p4 = Point(y: 20)
```

```swift
struct Point {
    var x: Int = 0
    var y: Int = 0
}
let p1 = Point()
let p2 = Point(x: 10, y: 20)
let p3 = Point(x: 10)
let p4 = Point(y: 20)
```

```swift
class Point {          // Class 'Point' has no initializers
    var x: Int
    var y: Int
}
let p1 = Point()       // 'Point' cannot be constructed because it has no accessible initializers
```

类的初始化器

- 如果类的所有成员都在定义的时候指定了初始值，编译器会为类生成无参的初始化器
  - 成员的初始化是在这个初始化器中完成的

```
class Point {
    var x: Int = 10
    var y: Int = 20
}
let p1 = Point()
```

```
class Point {
    var x: Int
    var y: Int
    init() {
        x = 10
        y = 20
    }
}
let p1 = Point()
```

- 上面2段代码是完全等效的

小码哥教育 SEEMYGO

结构体与类的本质区别

■ 结构体是值类型（枚举也是值类型），类是引用类型（指针类型）

```
class Size {
    var width = 1
    var height = 2
}
```

```
struct Point {
    var x = 3
    var y = 4
}
```

```
func test() {
    var size = Size()
    var point = Point()
}
```

栈空间

内存地址	内存数据		说明
0x10000	3	point	point.x
0x10008	4		point.y
0x10010	0x90000	size	Size对象的内存地址

堆空间

内存地址	内存数据		说明
0x90000	0xe41a8	Size对象	指向类型信息
0x90008	0x20002		引用计数
0x90010	1		size.width
0x90018	2		size.height

■ 上图都是针对64bit环境

小码哥教育 SEEMYGO

值类型

- 值类型赋值给var、let或者给函数传参，是直接将所有内容拷贝一份
  - 类似于对文件进行copy、paste操作，产生了全新的文件副本。属于深拷贝（deep copy）

```
struct Point {
    var x: Int
    var y: Int
}
```

```
func test() {
    var p1 = Point(x: 10, y: 20)
    var p2 = p1
}
```

栈空间

内存地址	内存数据		说明
0x10000	10	p2	p2.x
0x10008	20		p2.y
0x10010	10	p1	p1.x
0x10018	20		p1.y

```
p2.x = 11
p2.y = 22
// 请问p1.x和p1.y是多少？
```

值类型的赋值操作

```swift
var s1 = "Jack"
var s2 = s1
s2.append("_Rose")
print(s1) // Jack
print(s2) // Jack_Rose
```

```swift
var a1 = [1, 2, 3]
var a2 = a1
a2.append(4)
a1[0] = 2
print(a1) // [2, 2, 3]
print(a2) // [1, 2, 3, 4]
```

```swift
var d1 = ["max" : 10, "min" : 2]
var d2 = d1
d1["other"] = 7
d2["max"] = 12
print(d1) // ["other": 7, "max": 10, "min": 2]
print(d2) // ["max": 12, "min": 2]
```

- 在Swift标准库中，为了提升性能，String、Array、Dictionary、Set采取了Copy On Write的技术
  - 比如仅当有“写”操作时，才会真正执行拷贝操作
  - 对于标准库值类型的赋值操作，Swift 能确保最佳性能，所有没必要为了保证最佳性能来避免赋值
- 建议：不需要修改的，尽量定义成let

小码哥教育 SEEMYGO

值类型的赋值操作

```
struct Point {
    var x: Int
    var y: Int
}
var p1 = Point(x: 10, y: 20)
p1 = Point(x: 11, y: 22)
```

栈空间

内存地址	内存数据		说明
0x10010	10	p1	p1.x
0x10018	20		p1.y

栈空间

内存地址	内存数据		说明
0x10010	11	p1	p1.x
0x10018	22		p1.y

小码哥教育@M了个J

引用类型

- 引用赋值给var、let或者给函数传参，是将内存地址拷贝一份
  - 类似于制作一个文件的替身（快捷方式、链接），指向的是同一个文件。属于浅拷贝（shallow copy）

```
class Size {
    var width: Int
    var height: Int
    init(width: Int, height: Int) {
        self.width = width
        self.height = height
    }
}
```

```
func test() {
    var s1 = Size(width: 10, height: 20)
    var s2 = s1
}
```

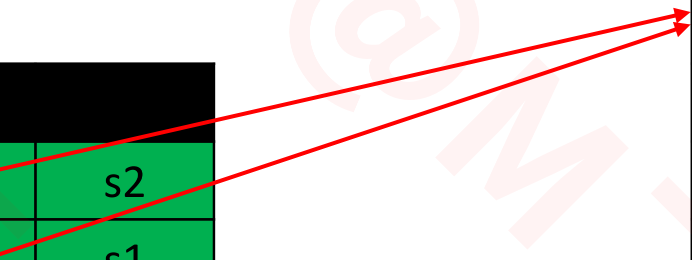
```
s2.width = 11
s2.height = 22
// 请问s1.width和s1.height是多少？
```

堆空间

内存地址	内存数据		说明
0x90000	0xe41a8	Size对象	指向类型信息
0x90008	0x20002		引用计数
0x90010	10		width
0x90018	20		height

栈空间

内存地址	内存数据	
0x10000	0x90000	s2
0x10008	0x90000	s1



小码哥教育 SEEMYGO

对象的堆空间申请过程

- 在Swift中，创建类的实例对象，要向堆空间申请内存，大概流程如下
- Class.__allocating_init()
- libswiftCore.dylib：_swift_allocObject_
- libswiftCore.dylib：swift_slowAlloc
- libsystem_malloc.dylib：malloc
- 在Mac、iOS中的malloc函数分配的内存大小总是16的倍数
- 通过class_getInstanceSize可以得知：类的对象至少需要占用多少内存

```
class Point  {
    var x = 11
    var test = true
    var y = 22
}
var p = Point()
class_getInstanceSize(type(of: p)) // 40
class_getInstanceSize(Point.self) // 40
```

引用类型的赋值操作

```
class Size {
    var width: Int
    var height: Int
    init(width: Int, height: Int) {
        self.width = width
        self.height = height
    }
}
var s1 = Size(width: 10, height: 20)
s1 = Size(width: 11, height: 22)
```

堆空间

内存地址	内存数据		说明
0x90000	0xe41a8	Size对象	指向类型信息
0x90008	0x20002		引用计数
0x90010	10		width
0x90018	20		height

内存地址	内存数据		说明
0x80000	0xe41a8	Size对象	指向类型信息
0x80008	0x20002		引用计数
0x80010	11		width
0x80018	22		height

栈空间

内存地址	内存数据	
0x10000	0x90000	s1

内存地址	内存数据	
0x10000	0x80000	s1

值类型、引用类型的let

```swift
struct Point {
    var x: Int
    var y: Int
}
class Size {
    var width: Int
    var height: Int
    init(width: Int, height: Int) {
        self.width = width
        self.height = height
    }
}
```

```swift
let p = Point(x: 10, y: 20)
p = Point(x: 11, y: 22)
p.x = 33
p.y = 44

let s = Size(width: 10, height: 20)
s = Size(width: 11, height: 22)
s.width = 33
s.height = 44
```

```swift
let str = "Jack"
str.append("_Rose")

let arr = [1, 2, 3]
arr[0] = 11
arr.append(4)
```

小码哥教育 SEEMYGO

嵌套类型

```swift
struct Poker {
    enum Suit : Character {
        case spades = "♠", hearts = "♥", diamonds = "♦", clubs = "♣"
    }
    enum Rank : Int {
        case two = 2, three, four, five, six, seven, eight, nine, ten
        case jack, queen, king, ace
    }
}
```

```swift
print(Poker.Suit.hearts.rawValue)

var suit = Poker.Suit.spades
suit = .diamonds

var rank = Poker.Rank.five
rank = .king
```

小码哥教育@M了个J

小码哥教育 SEEMYGO

枚举、结构体、类都可以定义方法

- 一般把定义在枚举、结构体、类内部的函数，叫做方法

```
class Size {
    var width = 10
    var height = 10
    func show() {
        print("width=\(width), height=\(height)")
    }
}
let s = Size()
s.show() // width=10, height=10
```

```
struct Point {
    var x = 10
    var y = 10
    func show() {
        print("x=\(x), y=\(y)")
    }
}
let p = Point()
p.show() // x=10, y=10
```

```
enum PokerFace : Character {
    case spades = "♠", hearts = "♥", diamonds = "♦", clubs = "♣"
    func show() {
        print("face is \(rawValue)")
    }
}
let pf = PokerFace.hearts
pf.show() // face is ♥
```

- 方法占用对象的内存么？
  - 不占用
  - 方法的本质就是函数
  - 方法、函数都存放在代码段

作业

■ 思考以下结构体、类对象的内存结构是怎样的？

```
struct Point {
    var x: Int
    var b1: Bool
    var b2: Bool
    var y: Int
}
var p = Point(x: 10, b1: true, b2: true, y: 20)
```

```
class Size {
    var width: Int
    var b1: Bool
    var b2: Bool
    var height: Int
    init(width: Int, b1: Bool, b2: Bool, height: Int) {
        self.width = width
        self.b1 = b1
        self.b2 = b2
        self.height = height
    }
}
var s = Size(width: 10, b1: true, b2: true, height: 20)
```