枚举

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

枚举的基本用法

```
enum Direction {
    case north
    case south
    case east
    case west
}
```

```
enum Direction {
    case north, south, east, west
}
```

```
var dir = Direction.west
dir = Direction.east
dir = .north
print(dir) // north
```

```
switch dir {
case .north:
    print("north")
case .south:
    print("south")
case .east:
    print("east")
case .west:
    print("west")
}
```

关联值（Associated Values）

- 有时将枚举的成员值跟其他类型的值关联存储在一起，会非常有用

```
enum Score {
    case points(Int)
    case grade(Character)
}
```

```
var score = Score.points(96)
score = .grade("A")
```

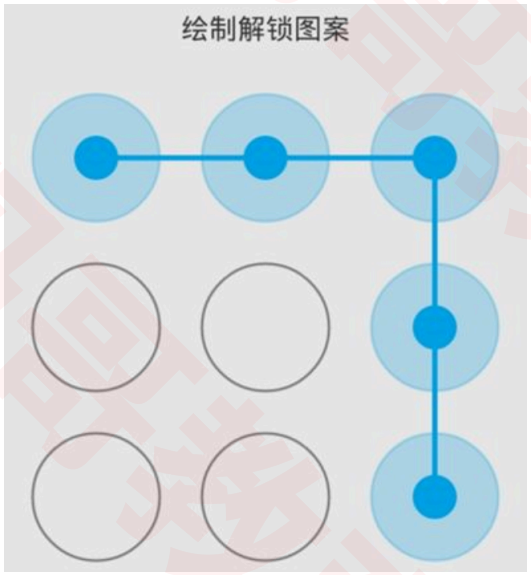
```
switch score {
case let .points(i):
    print(i, "points")
case let .grade(i):
    print("grade", i)
} // grade A
```

```
enum Date {
    case digit(year: Int, month: Int, day: Int)
    case string(String)
}
var date = Date.digit(year: 2011, month: 9, day: 10)
date = .string("2011-09-10")
switch date {
case .digit(let year, let month, let day):
    print(year, month, day)
case let .string(value):
    print(value)
}
```

- 必要时let也可以改为var

小码哥教育 SEEMYGO

关联值举例




```
enum Password {
    case number(Int, Int, Int, Int)
    case gesture(String)
}
```

```
var pwd = Password.number(3, 5, 7, 8)
pwd = .gesture("12369")
```

```
switch pwd {
case let .number(n1, n2, n3, n4):
    print("number is ", n1, n2, n3, n4)
case let .gesture(str):
    print("gesture is", str)
}
```

原始值（Raw Values）

- 枚举成员可以使用相同类型的默认值预先对应，这个默认值叫做：原始值

```
enum PokerSuit : Character {
    case spade = "♠"
    case heart = "❤"
    case diamond = "♦"
    case club = "♣"
}
```

```
var suit = PokerSuit.spade
print(suit) // spade
print(suit.rawValue) // ♠
print(PokerSuit.club.rawValue) // ♣
```

```
enum Grade : String {
    case perfect = "A"
    case great = "B"
    case good = "C"
    case bad = "D"
}
print(Grade.perfect.rawValue) // A
print(Grade.great.rawValue) // B
print(Grade.good.rawValue) // C
print(Grade.bad.rawValue) // D
```

- 注意：原始值不占用枚举变量的内存

隐式原始值（Implicitly Assigned Raw Values）

- 如果枚举的原始值类型是Int、String，Swift会自动分配原始值

```
enum Direction : String {
    case north = "north"
    case south = "south"
    case east = "east"
    case west = "west"
}
```

```
// 等价于
enum Direction : String {
    case north, south, east, west
}
print(Direction.north) // north
print(Direction.north.rawValue) // north
```

```
enum Season : Int {
    case spring, summer, autumn, winter
}
print(Season.spring.rawValue) // 0
print(Season.summer.rawValue) // 1
print(Season.autumn.rawValue) // 2
print(Season.winter.rawValue) // 3
```

```
enum Season : Int {
    case spring = 1, summer, autumn = 4, winter
}
print(Season.spring.rawValue) // 1
print(Season.summer.rawValue) // 2
print(Season.autumn.rawValue) // 4
print(Season.winter.rawValue) // 5
```

递归枚举（Recursive Enumeration）

```
indirect enum ArithExpr {
    case number(Int)
    case sum(ArithExpr, ArithExpr)
    case difference(ArithExpr, ArithExpr)
}
```

```
enum ArithExpr {
    case number(Int)
    indirect case sum(ArithExpr, ArithExpr)
    indirect case difference(ArithExpr, ArithExpr)
}
```

```
let five = ArithExpr.number(5)
let four = ArithExpr.number(4)
let two = ArithExpr.number(2)
let sum = ArithExpr.sum(five, four)
let difference = ArithExpr.difference(sum, two)
```

```
func calculate(_ expr: ArithExpr) -> Int {
    switch expr {
    case let .number(value):
        return value
    case let .sum(left, right):
        return calculate(left) + calculate(right)
    case let .difference(left, right):
        return calculate(left) - calculate(right)
    }
}

calculate(difference)
```

MemoryLayout

- 可以使用MemoryLayout获取数据类型占用的内存大小

```
enum Password {
    case number(Int, Int, Int, Int)
    case other
}
```

```
MemoryLayout<Password>.stride // 40, 分配占用的空间大小
MemoryLayout<Password>.size // 33, 实际用到的空间大小
MemoryLayout<Password>.alignment // 8, 对齐参数
```

```
var pwd = Password.number(9, 8, 6, 4)
pwd = .other
MemoryLayout.stride(ofValue: pwd) // 40
MemoryLayout.size(ofValue: pwd) // 33
MemoryLayout.alignment(ofValue: pwd) // 8
```

思考下面枚举变量的内存布局

```
enum TestEnum {
    case test1, test2, test3
}
var t = TestEnum.test1
t = .test2
t = .test3
```

```
enum TestEnum : Int {
    case test1 = 1, test2 = 2, test3 = 3
}
var t = TestEnum.test1
t = .test2
t = .test3
```

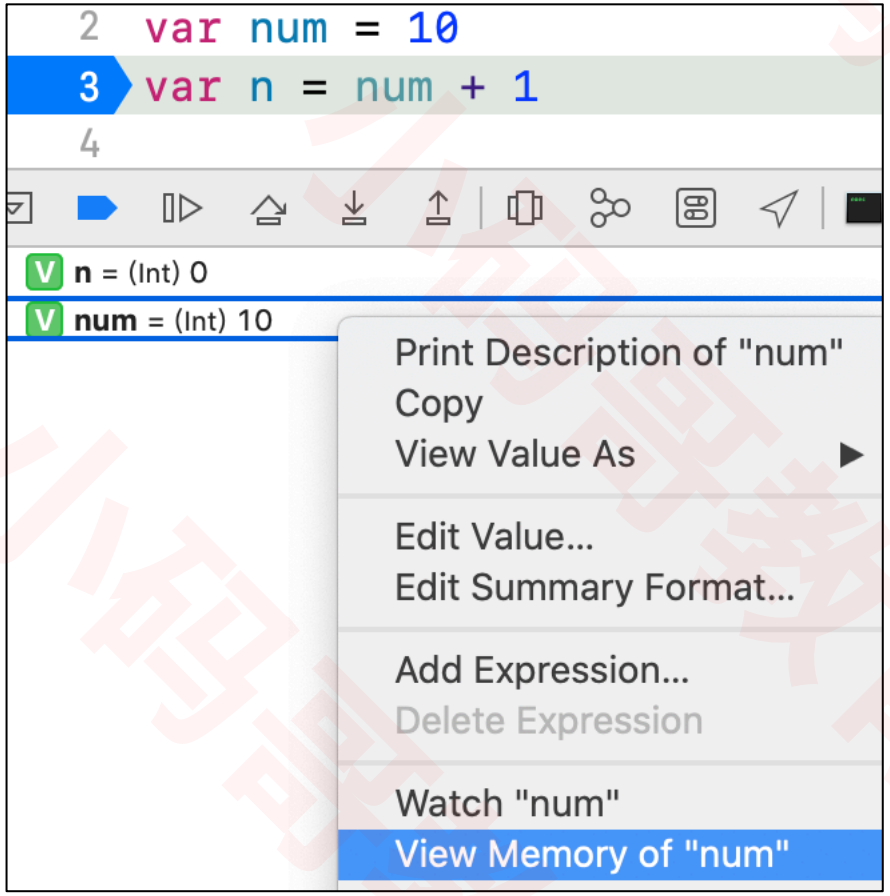
```
enum TestEnum {
    case test
}
var t = TestEnum.test
```

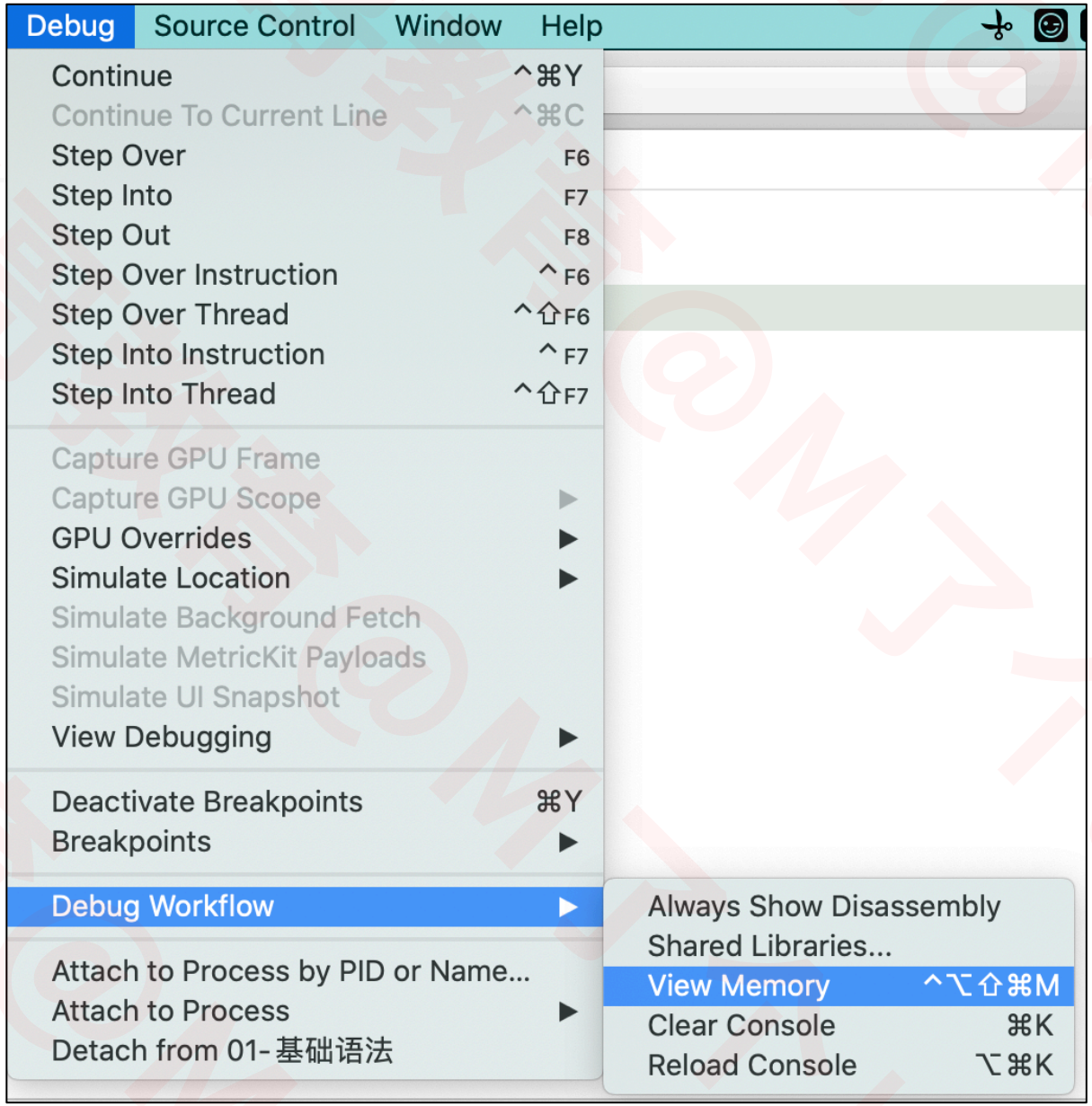
```
enum TestEnum {
    case test(Int)
}
var t = TestEnum.test(10)
```

```
enum TestEnum {
    case test1(Int, Int, Int)
    case test2(Int, Int)
    case test3(Int)
    case test4(Bool)
    case test5
}
var e = TestEnum.test1(1, 2, 3)
e = .test2(4, 5)
e = .test3(6)
e = .test4(true)
e = .test5
```

- 它们的switch语句底层又是如何实现的？

窥探内存





- 窥探内存细节的小工具：https://github.com/CoderMJLee/Mems

小码哥教育 SEEMYGO

进一步观察下面枚举的内存布局

```
enum TestEnum {
    case test0
    case test1
    case test2
    case test4(Int)
    case test5(Int, Int)
    case test6(Int, Int, Int, Bool)
}
```

```
enum TestEnum {
    case test0
    case test1
    case test2
    case test4(Int)
    case test5(Int, Int)
    case test6(Int, Int, Bool, Int)
}
```

```
enum TestEnum {
    case test0
    case test1
    case test2
    case test4(Int)
    case test5(Int, Int)
    case test6(Int, Bool, Int)
}
```