高级运算符

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

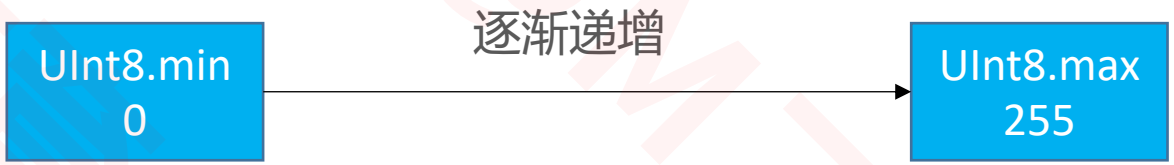
码拉松

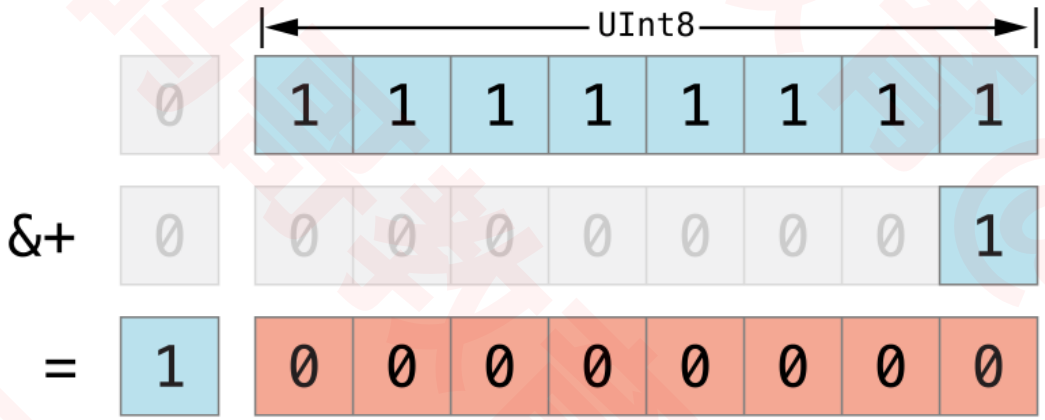
溢出运算符（Overflow Operator）

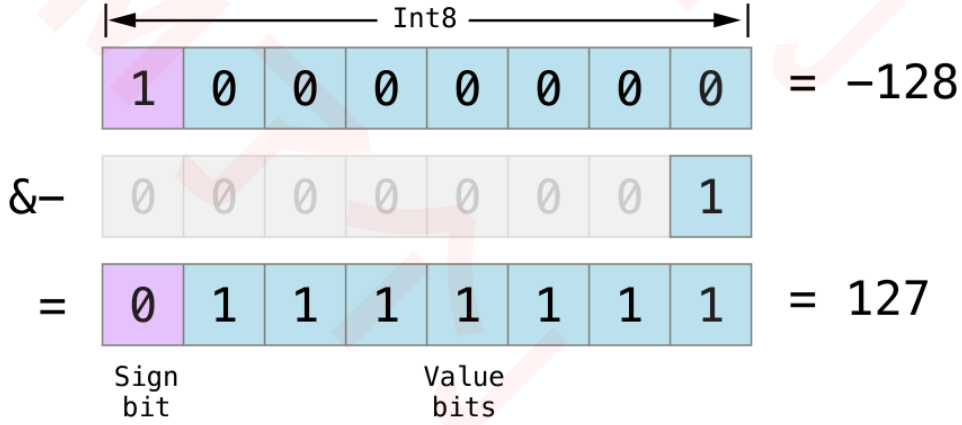
- Swift的算数运算符出现溢出时会抛出运行时错误
- Swift有溢出运算符（&+、&-、&*），用来支持溢出运算

```
var min = UInt8.min
print(min &- 1) // 255, Int8.max

var max = UInt8.max
print(max &+ 1) // 0, Int8.min
print(max &* 2) // 254, 等价于 max &+ max
```







运算符重载（Operator Overload）

- 类、结构体、枚举可以为现有的运算符提供自定义的实现，这个操作叫做：运算符重载

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

func + (p1: Point, p2: Point) -> Point {
    Point(x: p1.x + p2.x, y: p1.y + p2.y)
}

let p = Point(x: 10, y: 20) + Point(x: 11, y: 22)
print(p) // Point(x: 21, y: 42)
```

```
struct Point {
    var x: Int, y: Int
    static func + (p1: Point, p2: Point) -> Point {
        Point(x: p1.x + p2.x, y: p1.y + p2.y)
    }
}
```

小码哥教育 SEEMYGO

运算符重载

```
static func + (p1: Point, p2: Point) -> Point {
    Point(x: p1.x + p2.x, y: p1.y + p2.y)
}
static func - (p1: Point, p2: Point) -> Point {
    Point(x: p1.x - p2.x, y: p1.y - p2.y)
}
static prefix func - (p: Point) -> Point {
    Point(x: -p.x, y: -p.y)
}
static func += (p1: inout Point, p2: Point) {
    p1 = p1 + p2
}
```

```
static prefix func ++ (p: inout Point) -> Point {
    p += Point(x: 1, y: 1)
    return p
}
static postfix func ++ (p: inout Point) -> Point {
    let tmp = p
    p += Point(x: 1, y: 1)
    return tmp
}
static func == (p1: Point, p2: Point) -> Bool {
    (p1.x == p2.x) && (p1.y == p2.y)
}
```

Equatable

- 要想得知2个实例是否等价，一般做法是遵守 Equatable 协议，重载 == 运算符
  - 与此同时，等价于重载了 != 运算符

```
struct Point : Equatable {
    var x: Int, y: Int
}
var p1 = Point(x: 10, y: 20)
var p2 = Point(x: 11, y: 22)
print(p1 == p2) // false
print(p1 != p2) // true
```

- Swift为以下类型提供默认的 Equatable 实现
  - 没有关联类型的枚举
  - 只拥有遵守 Equatable 协议关联类型的枚举
  - 只拥有遵守 Equatable 协议关联类型的结构体

- 引用类型比较存储的地址值是否相等（是否引用着同一个对象），使用恒等运算符 === 、!==

Comparable

```
// score大的比较大，若score相等，age小的比较大
struct Student : Comparable {
    var age: Int
    var score: Int
    init(score: Int, age: Int) {
        self.score = score
        self.age = age
    }
    static func < (lhs: Student, rhs: Student) -> Bool {
        (lhs.score < rhs.score)
            || (lhs.score == rhs.score && lhs.age > rhs.age)
    }
    static func > (lhs: Student, rhs: Student) -> Bool {
        (lhs.score > rhs.score)
            || (lhs.score == rhs.score && lhs.age < rhs.age)
    }
    static func <= (lhs: Student, rhs: Student) -> Bool {
        !(lhs > rhs)
    }
    static func >= (lhs: Student, rhs: Student) -> Bool {
        !(lhs < rhs)
    }
}
```

- 要想比较2个实例的大小，一般做法是遵守 Comparable 协议，重载相应的运算符

```
var stu1 = Student(score: 100, age: 20)
var stu2 = Student(score: 98, age: 18)
var stu3 = Student(score: 100, age: 20)
print(stu1 > stu2) // true
print(stu1 >= stu2) // true
print(stu1 >= stu3) // true
print(stu1 <= stu3) // true
print(stu2 < stu1) // true
print(stu2 <= stu1) // true
```

自定义运算符（Custom Operator）

- 可以自定义新的运算符：在全局作用域使用operator进行声明

```
prefix operator 前缀运算符
postfix operator 后缀运算符
infix operator 中缀运算符 : 优先级组
```

```
precedencegroup 优先级组 {
    associativity: 结合性(left\right\none)
    higherThan: 比谁的优先级高
    lowerThan: 比谁的优先级低
    assignment: true代表在可选链操作中拥有跟赋值运算符一样的优先级
}
```

```
prefix operator +++
infix operator +- : PlusMinusPrecedence
precedencegroup PlusMinusPrecedence {
    associativity: none
    higherThan: AdditionPrecedence
    lowerThan: MultiplicationPrecedence
    assignment: true
}
```

- Apple文档参考：
  - https://developer.apple.com/documentation/swift/swift_standard_library/operator_declarations
  - https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#ID380

小码哥教育 SEEMYGO

自定义运算符

```
struct Point {
    var x: Int, y: Int
    static prefix func +++ (point: inout Point) -> Point {
        point = Point(x: point.x + point.x, y: point.y + point.y)
        return point
    }
    static func +- (left: Point, right: Point) -> Point {
        return Point(x: left.x + right.x, y: left.y - right.y)
    }
    static func +- (left: Point?, right: Point) -> Point {
        print("+-")
        return Point(x: left?.x ?? 0 + right.x, y: left?.y ?? 0 - right.y)
    }
}
```

```
struct Person {
    var point: Point
}
var person: Person? = nil
person?.point +- Point(x: 10, y: 20)
```