

扩展

@M了个J

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>

码拉松



实力IT教育 www.520it.com

扩展 (Extension)

- Swift中的扩展，有点类似于OC中的分类 (Category)
- 扩展可以为枚举、结构体、类、协议添加新功能
 - 可以添加方法、计算属性、下标、（便捷）初始化器、嵌套类型、协议等等
- 扩展不能办到的事情
 - 不能覆盖原有的功能
 - 不能添加存储属性，不能向已有的属性添加属性观察器
 - 不能添加父类
 - 不能添加指定初始化器，不能添加反初始化器
 - ...

计算属性、下标、方法、嵌套类型

```
extension Double {  
    var km: Double { self * 1_000.0 }  
    var m: Double { self }  
    var dm: Double { self / 100.0 }  
    var cm: Double { self / 10.0 }  
    var mm: Double { self / 1_000.0 }  
}
```

```
extension Array {  
    subscript(nullable idx: Int) -> Element? {  
        if (startIndex..<endIndex).contains(idx) {  
            return self[idx]  
        }  
        return nil  
    }  
}
```

```
extension Int {  
    func repetitions(task: () -> Void) {  
        for _ in 0..  
self { task() }  
    }  
    mutating func square() -> Int {  
        self = self * self  
        return self  
    }  
    enum Kind { case negative, zero, positive }  
    var kind: Kind {  
        switch self {  
            case 0: return .zero  
            case let x where x > 0: return .positive  
            default: return .negative  
        }  
    }  
    subscript(digitIndex: Int) -> Int {  
        var decimalBase = 1  
        for _ in 0..  
digitIndex { decimalBase *= 10 }  
        return (self / decimalBase) % 10  
    }  
}
```

协议、初始化器

```
class Person {  
    var age: Int  
    var name: String  
    init(age: Int, name: String) {  
        self.age = age  
        self.name = name  
    }  
}  
  
extension Person : Equatable {  
    static fun == (left: Person, right: Person) -> Bool {  
        left.age == right.age && left.name == right.name  
    }  
    convenience init() {  
        self.init(age: 0, name: "")  
    }  
}
```

```
struct Point {  
    var x: Int = 0  
    var y: Int = 0  
}  
  
extension Point {  
    init(_ point: Point) {  
        self.init(x: point.x, y: point.y)  
    }  
}  
  
var p1 = Point()  
var p2 = Point(x: 10)  
var p3 = Point(y: 20)  
var p4 = Point(x: 10, y: 20)  
var p5 = Point(p4)
```

- 如果希望自定义初始化器的同时，编译器也能够生成默认初始化器
- 可以在扩展中编写自定义初始化器

- 如果一个类型已经实现了协议的所有要求，但是还没有声明它遵守了这个协议
- 可以通过扩展来让它遵守这个协议

```
protocol TestProtocol {  
    func test()  
}  
  
class TestClass {  
    func test() {  
        print("test")  
    }  
}  
  
extension TestClass : TestProtocol { }
```

- 编写一个函数，判断一个整数是否为奇数？

```
func isOdd<T: BinaryInteger>(_ i: T) -> Bool {  
    i % 2 != 0  
}
```

```
extension BinaryInteger {  
    func isOdd() -> Bool { self % 2 != 0 }  
}
```

- 扩展可以给协议提供默认实现，也间接实现『可选协议』的效果
- 扩展可以给协议扩充『协议中从未声明过的方法』

```
protocol TestProtocol {  
    func test1()  
}  
  
extension TestProtocol {  
    func test1() {  
        print("TestProtocol test1")  
    }  
    func test2() {  
        print("TestProtocol test2")  
    }  
}
```

```
class TestClass : TestProtocol { }  
var cls = TestClass()  
cls.test1() // TestProtocol test1  
cls.test2() // TestProtocol test2  
var cls2: TestProtocol = TestClass()  
cls2.test1() // TestProtocol test1  
cls2.test2() // TestProtocol test2
```

```
class TestClass : TestProtocol {  
    func test1() { print("TestClass test1") }  
    func test2() { print("TestClass test2") }  
}  
var cls = TestClass()  
cls.test1() // TestClass test1  
cls.test2() // TestClass test2  
var cls2: TestProtocol = TestClass()  
cls2.test1() // TestClass test1  
cls2.test2() // TestProtocol test2
```

```
class Stack<E> {  
    var elements = [E]()  
    func push(_ element: E) {  
        elements.append(element)  
    }  
    func pop() -> E { elements.removeLast() }  
    func size() -> Int { elements.count }  
}  
// 扩展中依然可以使用原类型中的泛型类型  
extension Stack {  
    func top() -> E { elements.last! }  
}  
// 符合条件才扩展  
extension Stack : Equatable where E : Equatable {  
    static func == (left: Stack, right: Stack) -> Bool {  
        left.elements == right.elements  
    }  
}
```