Swift 类型转换 Swift 协议

Swift 扩展

扩展就是向一个已有的类、结构体或枚举类型添加新功能。

扩展可以对一个类型添加新的功能，但是不能重写已有的功能。

Swift 中的扩展可以：

- 添加计算型属性和计算型静态属性
- 定义实例方法和类型方法
- 提供新的构造器
- 定义下标
- 定义和使用新的嵌套类型
- 使一个已有类型符合某个协议

语法

扩展声明使用关键字 **extension**：

```
extension SomeType {
    // 加到SomeType的新功能写到这里
}
```

一个扩展可以扩展一个已有类型，使其能够适配一个或多个协议，语法格式如下：

```
extension SomeType: SomeProtocol, AnotherProctocol {
    // 协议实现写到这里
}
```

计算型属性

扩展可以向已有类型添加计算型实例属性和计算型类型属性。

实例

下面的例子向 Int 类型添加了 5 个计算型实例属性并扩展其功能：

```
extension Int {
   var add: Int {return self + 100 }
   var sub: Int { return self - 10 }
   var mul: Int { return self * 10 }
   var div: Int { return self / 5 }
}
```

```
let addition = 3.add
print("加法运算后的值: \(addition)")

let subtraction = 120.sub
print("减法运算后的值: \(subtraction)")

let multiplication = 39.mul
print("乘法运算后的值: \(multiplication)")

let division = 55.div
print("除法运算后的值: \(division)")

let mix = 30.add + 34.sub
print("混合运算结果: \(mix)")
```

以上程序执行输出结果为：

```
加法运算后的值: 103
减法运算后的值: 110
乘法运算后的值: 390
除法运算后的值: 11
混合运算结果: 154
```

构造器

扩展可以向已有类型添加新的构造器。

这可以让你扩展其它类型，将你自己的定制类型作为构造器参数，或者提供该类型的原始实现中没有包含的额外初始化选项。

扩展可以向类中添加新的便利构造器 init()，但是它们不能向类中添加新的指定构造器或析构函数 deinit() 。

```
struct sum {
    var num1 = 100, num2 = 200
}

struct diff {
    var no1 = 200, no2 = 100
}

struct mult {
    var a = sum()
    var b = diff()
}


extension mult {
    init(x: sum, y: diff) {
```

```
        _ = x.num1 + x.num2
        _ = y.no1 + y.no2
    }
}


let a = sum(num1: 100, num2: 200)
let b = diff(no1: 200, no2: 100)

let getMult = mult(x: a, y: b)
print("getMult sum\(getMult.a.num1, getMult.a.num2)")
print("getMult diff\(getMult.b.no1, getMult.b.no2)")
```

以上程序执行输出结果为：

```
getMult sum(100, 200)
getMult diff(200, 100)
```

方法

扩展可以向已有类型添加新的实例方法和类型方法。

下面的例子向Int类型添加一个名为 topics 的新实例方法：

```
extension Int {
   func topics(summation: () -> ()) {
      for _ in 0..<self {
         summation()
      }
   }
}

4.topics({
   print("扩展模块内")
})

3.topics({
   print("内型转换模块内")
})
```

以上程序执行输出结果为：

```
扩展模块内
扩展模块内
扩展模块内
扩展模块内
```

```
内型转换模块内
内型转换模块内
内型转换模块内
```

这个topics方法使用了一个() -> ()类型的单参数，表明函数没有参数而且没有返回值。

定义该扩展之后，你就可以对任意整数调用 topics 方法,实现的功能则是多次执行某任务：

可变实例方法

通过扩展添加的实例方法也可以修改该实例本身。

结构体和枚举类型中修改self或其属性的方法必须将该实例方法标注为mutating，正如来自原始实现的修改方法一样。

实例

下面的例子向 Swift 的 Double 类型添加了一个新的名为 square 的修改方法，来实现一个原始值的平方计算：

```
extension Double {
   mutating func square() {
      let pi = 3.1415
      self = pi * self * self
   }
}

var Trial1 = 3.3
Trial1.square()
print("圆的面积为: \(Trial1)")


var Trial2 = 5.8
Trial2.square()
print("圆的面积为: \(Trial2)")


var Trial3 = 120.3
Trial3.square()
print("圆的面积为: \(Trial3)")
```

以上程序执行输出结果为：

```
圆的面积为: 34.210935
圆的面积为: 105.68006
圆的面积为: 45464.070735
```

下标

扩展可以向一个已有类型添加新下标。

实例

以下例子向 Swift 内建类型Int添加了一个整型下标。该下标[n]返回十进制数字

```
extension Int {
   subscript(var multtable: Int) -> Int {
      var no1 = 1
      while multtable > 0 {
         no1 *= 10
         --multtable
      }
      return (self / no1) % 10
   }
}

print(12[0])
print(7869[1])
print(786543[2])
```

以上程序执行输出结果为：

```
2
6
5
```

嵌套类型

扩展可以向已有的类、结构体和枚举添加新的嵌套类型：

```
extension Int {
   enum calc
   {
      case add
      case sub
      case mult
      case div
      case anything
   }

   var print: calc {
      switch self
      {
         case 0:
            return .add
         case 1:
            return .sub
```

```
         case 2:
            return .mult
         case 3:
            return .div
         default:
            return .anything
       }
   }
}

func result(numb: [Int]) {
   for i in numb {
      switch i.print {
         case .add:
            print(" 10 ")
          case .sub:
            print(" 20 ")
         case .mult:
         print(" 30 ")
         case .div:
         print(" 40 ")
         default:
         print(" 50 ")

      }
   }
}

result([0, 1, 2, 3, 4, 7])
```

以上程序执行输出结果为：

```
 10
 20
 30
 40
 50
 50
```

← Swift 类型转换 Swift 协议 →

点我分享笔记