Swift 自动引用计数（ARC） Swift 类型转换

# Swift 可选链

可选链（Optional Chaining）是一种可以请求和调用属性、方法和子脚本的过程，用于请求或调用的目标可能为nil。

可选链返回两个值：

- 如果目标有值，调用就会成功，返回该值
- 如果目标为nil，调用将返回nil

多次请求或调用可以被链接成一个链，如果任意一个节点为nil将导致整条链失效。

## 可选链可替代强制解析

通过在属性、方法、或下标脚本的可选值后面放一个问号(?)，即可定义一个可选链。

| 可选链 '?' | 感叹号（!）强制展开方法，属性，下标脚本可选链 |
| --- | --- |
| ? 放置于可选值后来调用方法，属性，下标脚本 | ! 放置于可选值后来调用方法，属性，下标脚本来强制展开值 |
| 当可选为 nil 输出比较友好的错误信息 | 当可选为 nil 时强制展开执行错误 |

### 使用感叹号(!)可选链实例

```
class Person {
    var residence: Residence?
}

class Residence {
    var numberOfRooms = 1
}

let john = Person()

//将导致运行时错误
let roomCount = john.residence!.numberOfRooms
```

以上程序执行输出结果为：

```
fatal error: unexpectedly found nil while unwrapping an Optional value
```

想使用感叹号（!）强制解析获得这个人residence属性numberOfRooms属性值，将会引发运行时错误，因为这时没有可以供解析的residence值。

### 使用问号(?)可选链实例

```
class Person {
    var residence: Residence?
}

class Residence {
    var numberOfRooms = 1
}

let john = Person()

// 链接可选residence?属性，如果residence存在则取回numberOfRooms的值
if let roomCount = john.residence?.numberOfRooms {
    print("John 的房间号为 \(roomCount)。")
} else {
    print("不能查看房间号")
}
```

以上程序执行输出结果为：

```
不能查看房间号
```

因为这种尝试获得numberOfRooms的操作有可能失败，可选链会返回Int?类型值，或者称作"可选Int"。当residence是空的时候（上例），选择Int将会为空，因此会出现无法访问numberOfRooms的情况。
要注意的是，即使numberOfRooms是非可选Int（Int?）时这一点也成立。只要是通过可选链的请求就意味着最后numberOfRooms总是返回一个Int?而不是Int。

---

# 为可选链定义模型类

你可以使用可选链来多层调用属性，方法，和下标脚本。这让你可以利用它们之间的复杂模型来获取更底层的属性，并检查是否可以成功获取此类底层属性。

## 实例

定义了四个模型类，其中包括多层可选链：

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
```

```
        return rooms[i]
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}
```

## 通过可选链调用方法

你可以使用可选链的来调用可选值的方法并检查方法调用是否成功。即使这个方法没有返回值，你依然可以使用可选链来达成这一目的。

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        return rooms[i]
```

```
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}

let john = Person()


if ((john.residence?.printNumberOfRooms()) != nil) {
    print("输出房间号")
} else {
    print("无法输出房间号")
}
```

以上程序执行输出结果为：

```
无法输出房间号
```

使用if语句来检查是否能成功调用printNumberOfRooms方法：如果方法通过可选链调用成功，printNumberOfRooms的隐式返回值将会是Void，如果没有成功，将返回nil。

---

# 使用可选链调用下标脚本

你可以使用可选链来尝试从下标脚本获取值并检查下标脚本的调用是否成功，然而，你不能通过可选链来设置下标脚本。

## 实例1

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        return rooms[i]
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}

let john = Person()
if let firstRoomName = john.residence?[0].name {
    print("第一个房间名 \(firstRoomName).")
```

```
} else {
    print("无法检索到房间")
}
```

以上程序执行输出结果为：

```
无法检索到房间
```

在下标脚本调用中可选链的问号直接跟在 circname.print 的后面，在下标脚本括号的前面，因为circname.print是可选链试图获得的可选值。

## 实例2

实例中创建一个Residence实例给john.residence，且在他的rooms数组中有一个或多个Room实例，那么你可以使用可选链通过Residence下标脚本来获取在rooms数组中的实例了：

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        return rooms[i]
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
```

```
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}

let john = Person()
let johnsHouse = Residence()
johnsHouse.rooms.append(Room(name: "客厅"))
johnsHouse.rooms.append(Room(name: "厨房"))
john.residence = johnsHouse

let johnsAddress = Address()
johnsAddress.buildingName = "The Larches"
johnsAddress.street = "Laurel Street"
john.residence!.address = johnsAddress

if let johnsStreet = john.residence?.address?.street {
    print("John 所在的街道是 \(johnsStreet)。")
} else {
    print("无法检索到地址。 ")
}
```

以上程序执行输出结果为：

```
John 所在的街道是 Laurel Street。
```

# 通过可选链接调用来访问下标

通过可选链接调用，我们可以用下标来对可选值进行读取或写入，并且判断下标调用是否成功。

## 实例

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
```

```
    subscript(i: Int) -> Room {
        return rooms[i]
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}

let john = Person()

let johnsHouse = Residence()
johnsHouse.rooms.append(Room(name: "客厅"))
johnsHouse.rooms.append(Room(name: "厨房"))
john.residence = johnsHouse

if let firstRoomName = john.residence?[0].name {
    print("第一个房间名为\(firstRoomName)")
} else {
    print("无法检索到房间")
}
```

以上程序执行输出结果为：

```
第一个房间名为客厅
```

# 访问可选类型的下标

如果下标返回可空类型值，比如Swift中Dictionary的key下标。可以在下标的闭合括号后面放一个问号来链接下标的可空返回值：

```
var testScores = ["Dave": [86, 82, 84], "Bev": [79, 94, 81]]
testScores["Dave"]?[0] = 91
testScores["Bev"]?[0]++
testScores["Brian"]?[0] = 72
// the "Dave" array is now [91, 82, 84] and the "Bev" array is now [80, 94, 81]
```

上面的例子中定义了一个testScores数组，包含了两个键值对， 把String类型的key映射到一个整形数组。
这个例子用可选链接调用把"Dave"数组中第一个元素设为91，把"Bev"数组的第一个元素+1，然后尝试把"Brian"数组中的第一个元素设为72。
前两个调用是成功的，因为这两个key存在。但是key"Brian"在字典中不存在，所以第三个调用失败。

# 连接多层链接

你可以将多层可选链连接在一起，可以掘取模型内更下层的属性方法和下标脚本。然而多层可选链不能再添加比已经返回的可选值更多的层。
如果你试图通过可选链获得Int值，不论使用了多少层链接返回的总是Int?。 相似的，如果你试图通过可选链获得Int?值，不论使用了多少层链接返回的总是Int?。

## 实例1

下面的例子试图获取john的residence属性里的address的street属性。这里使用了两层可选链来联系residence和address属性，它们两者都是可选类型：

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        return rooms[i]
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
```

```
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}

let john = Person()

if let johnsStreet = john.residence?.address?.street {
    print("John 的地址为 \(johnsStreet).")
} else {
    print("不能检索地址")
}
```

以上程序执行输出结果为：

```
不能检索地址
```

## 实例2

如果你为Address设定一个实例来作为john.residence.address的值，并为address的street属性设定一个实际值，你可以通过多层可选链来得到这个属性值。

```
class Person {
   var residence: Residence?
}

class Residence {
```

```
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        get{
            return rooms[i]
        }
        set {
            rooms[i] = newValue
        }
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

class Room {
    let name: String
    init(name: String) { self.name = name }
}

class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
            return buildingNumber
        } else {
            return nil
        }
    }
}
let john = Person()
john.residence?[0] = Room(name: "浴室")

let johnsHouse = Residence()
johnsHouse.rooms.append(Room(name: "客厅"))
johnsHouse.rooms.append(Room(name: "厨房"))
john.residence = johnsHouse

if let firstRoomName = john.residence?[0].name {
    print("第一个房间是\(firstRoomName)")
} else {
```

```
    print("无法检索房间")
}
```

以上实例输出结果为：

```
第一个房间是客厅
```

---

# 对返回可选值的函数进行链接

我们还可以通过可选链接来调用返回可空值的方法，并且可以继续对可选值进行链接。

## 实例

```
class Person {
    var residence: Residence?
}

// 定义了一个变量 rooms，它被初始化为一个Room[]类型的空数组
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        return rooms[i]
    }
    func printNumberOfRooms() {
        print("房间号为 \(numberOfRooms)")
    }
    var address: Address?
}

// Room 定义一个name属性和一个设定room名的初始化器
class Room {
    let name: String
    init(name: String) { self.name = name }
}

// 模型中的最终类叫做Address
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if (buildingName != nil) {
            return buildingName
        } else if (buildingNumber != nil) {
```

```
            return buildingNumber
        } else {
            return nil
        }
    }
}

let john = Person()

if john.residence?.printNumberOfRooms() != nil {
    print("指定了房间号)")
}  else {
    print("未指定房间号")
}
```

以上程序执行输出结果为：

```
未指定房间号
```

Swift 自动引用计数（ARC）

Swift 类型转换

点我分享笔记