可选链接

在本页

*可选链*是一个查询和调用当前可选的属性，方法和下标的过程nil。如果可选包含一个值，则属性，方法或下标调用会成功; 如果可选nil，则返回属性，方法或下标调用nil。多个查询可以链接在一起，并且链条中的任何链接都会优雅地失败nil。

> 注意
>
> Swift中的可选链接类似于nilObjective-C中的消息传递，但是可以适用于任何类型，并且可以检查成功或失败。

可选链接作为强制解包的替代方案

您可以通过在?想要调用属性，方法或下标的可选值（如果可选为非）后面放置问号（）来指定可选链接nil。这与!在可选值后放置感叹号（）以强制展开其值相似。主要的区别在于，可选链接在可选项时失败nil，而强制解包会在可选项时触发运行时错误nil。

为了反映可以对某个nil值调用可选链接的事实，即使正在查询的属性，方法或脚标返回非可选值，可选链接调用的结果也始终为可选值。您可以使用此可选返回值来检查可选链接调用是否成功（返回的可选值包含值），或者由于nil链中的值（返回的可选值是nil）而不成功。

具体来说，可选的链接调用的结果与预期的返回值的类型相同，但包含在可选项中。通常返回的属性在通过可选链接访问时Int将返回一个属性Int?。

接下来的几个代码片段演示了可选链接与强制解包的不同之处，并使您能够检查成功。

首先，定义两个类Person并Residence定义：

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

class Residence {
    var numberOfRooms = 1
}
```

Residence实例有一个Int名为的属性numberOfRooms，默认值为1。Person实例具有residence类型的可选属性Residence?。

如果你创建一个新的Person实例，它的residence属性默认是初始化的nil，因为它是可选的。在下面的代码中，john有一个residence属性值nil：

```
let john = Person()
```

如果您试图访问numberOfRooms此人的属性residence，通过在后面放置一个感叹号residence来强制展开其值，则会触发运行时错误，因为没有residence要展开的值：

```
let roomCount = john.residence!.numberOfRooms
// this triggers a runtime error
```

上面的代码在john.residence具有非nil值时会成功，并将设置roomCount为Int包含适当房间数的值。然而，当该代码始终触发运行时错误residence是nil，如上述所示。

可选链接提供了另一种访问值的方法numberOfRooms。要使用可选链接，请使用问号代替感叹号：

```
if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
```

```
// Prints "Unable to retrieve the number of rooms."
```

这告诉Swift在可选residence属性上“链接”并检索numberOfRoomsif residenceexists 的值。

由于访问尝试nu
nil，如上面的例子中，这个可选的Int也将是nil，反映事实，这是不可能的访问numberOfRooms。可选的Int是通过可选的结合访问以解开的整数并分配非可选值的roomCount变量。

请注意，即使numberOfRooms是非自选的，情况也是如此Int。通过可选链查询的事实意味着调用numberOfRooms将始终返回一个Int?而不是一个Int。

您可以将Residence实例分配给john.residence它，以便它不再具有nil值：

```
john.residence = Residence()
```

john.residence现在包含一个实际的Residence实例，而不是nil。如果您尝试使用numberOfRooms与之前相同的可选链接进行访问，则它将返回一个Int?包含默认numberOfRooms值的1：

```
if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
// Prints "John's residence has 1 room(s)."
```

为可选链定义模型类

您可以使用可选的链接调用属性，方法和深度超过一级的下标。这使您可以深入查看相关类型的复杂模型中的子属性，并检查是否可以访问这些子属性上的属性，方法和下标。

以下代码片段定义了四个模型类，以供后续几个示例使用，其中包括多级可选链的示例。这些类在所述展开Person和Residence通过添加模型从上方Room和Address类，具有相关联的属性，方法，和下标。

这个Person类的定义和以前一样：

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

这个Residence班比以前更复杂。这一次，Residence类定义了一个名为的变量属性rooms，它用一个空类型的数组初始化[Room]：

```
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("The number of rooms is \(numberOfRooms)")
    }
    var address: Address?
}
```

由于此版本Residence存储Room实例数组，因此其numberOfRooms属性将作为计算属性实现，而不是存储的属性。计算numberOfRooms属性只是返回数组中count属性的值rooms。

作为访问其rooms数组的快捷方式，此版本Residence提供了一个读写下标，可在rooms数组中请求的索引处访问该房间。

这个版本Residence也提供了一个叫做的方法printNumberOfRooms，它只是打印房间的房间数量。

最后，Residence定义了一个可选属性，名称为address，类型为Address?。Address下面定义了此属性的类型。

在Room用于类rooms阵列是简单的类与一种属性调用name，以及一个初始值设定到该属性设置为一个合适的房间名称：

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

这个模型中的最后一个类被调用Address。这个类有三个可选的类型属性String?。前两个特性，buildingName和buildingNumber，的替代方式，以识别特定建筑物作为地址的一部分。第三个属性，street用于为该地址命名街道：

```
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if let buildingNumber = buildingNumber, let street = street {
            return "\(buildingNumber) \(street)"
        } else if buildingName != nil {
            return buildingName
        } else {
            return nil
        }
    }
}
```

所述Address类还提供了一个名为方法buildingIdentifier()，其具有的返回类型String?。此方法检查地址的属性，buildingName如果它有值，则返回buildingNumber; street如果两者都有值，则将其连接; nil否则返回值。

通过可选链访问属性

正如在可选链中作为强制解包的替代方法所演示的那样，您可以使用可选链来访问可选值上的属性，并检查该属性访问是否成功。

使用上面定义的类创建一个新Person实例，并numberOfRooms像以前一样尝试访问它的属性：

```
let john = Person()
if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
// Prints "Unable to retrieve the number of rooms."
```

因为john.residence是nil，这个可选的链式调用失败的方式和以前一样。

您还可以尝试通过可选的链接来设置属性的值：

```
let someAddress = Address()
someAddress.buildingNumber = "29"
someAddress.street = "Acacia Road"
john.residence?.address = someAddress
```

在这个例子中，设置address属性的尝试john.residence会失败，因为john.residence当前是nil。

赋值是可选链接的一部分，这意味着=操作符右侧没有任何代码被评估。在前面的例子中，不容易发现它someAddress从来没有被评估过，因为访问一个常量没有任何副作用。下面的列表进行相同的分配，但它使用一个函数来创建地址。在返回一个值之前，该函数会打印“函数被调用”，从而可以查看是否=评估了操作符的右侧。

```
func createAddress() -> Address {
    print("Function was called.")

    let someAddress = Address()
    someAddress.buildingNumber = "29"
    someAddress.street = "Acacia Road"

    return someAddress
}
john.residence?.address = createAddress()
```

您可以知道该createAddress()函数未被调用，因为没有打印任何内容。

通过可选链接调用方法

您可以使用可选的链接来调用可选值的方法，并检查该方法调用是否成功。即使该方法未定义返回值，也可以执行此操作。

在printNumberOfRooms()对方法Residence类打印的当前值numberOfRooms。以下是该方法的外观：

```
func printNumberOfRooms() {
    print("The number of rooms is \(numberOfRooms)")
}
```

此方法不指定返回类型。但是，没有返回类型的函数和方法具有隐式返回类型Void，如函数无返回值中所述。这意味着它们返回一个值()，或者一个空元组。

如果使用可选链接调用此方法的可选值，则方法的返回类型将Void?不是Void，因为通过可选链接调用时，返回值始终为可选类型。这使您可以使用if语句来检查是否可以调用该printNumberOfRooms()方法，即使该方法本身没有定义返回值。比较来自printNumberOfRooms调用的返回值nil以查看方法调用是否成功：

```
if john.residence?.printNumberOfRooms() != nil {
    print("It was possible to print the number of rooms.")
} else {
    print("It was not possible to print the number of rooms.")
}
// Prints "It was not possible to print the number of rooms."
```

如果您尝试通过可选链接设置属性，情况也是如此。上面的示例通过可选链访问属性尝试为其设置address值john.residence，即使residence属性是nil。任何通过可选链设置属性的尝试都会返回一个类型值Void?，这使您可以比较nil以查看属性是否设置成功：

```
if (john.residence?.address = someAddress) != nil {
    print("It was possible to set the address.")
} else {
    print("It was not possible to set the address.")
}
// Prints "It was not possible to set the address."
```

通过可选链访问下标

您可以使用可选链来尝试从可选值上的下标中检索和设置值，并检查该下标调用是否成功。

> 注意

> 当通过可选链接访问可选值上的下标时，可将问号放在下标的括号*之前*，而不是之后。可选的链接问号总是紧跟在可选表达式的部分之后。

下面的示例尝试
john.residenceResidencejohn.residencenil

```
if let firstRoomName = john.residence?[0].name {
    print("The first room name is \(firstRoomName).")
} else {
    print("Unable to retrieve the first room name.")
}
// Prints "Unable to retrieve the first room name."
```

此下标调用中的可选链接问号紧接john.residence在下标括号之前，因为它john.residence是可选链接尝试的可选值。

同样，您可以尝试通过带有可选链接的下标来设置新值：

```
john.residence?[0] = Room(name: "Bathroom")
```

此下标设置尝试也失败，因为residence目前nil。

如果创建实际Residence实例并将其分配给数组中的john.residence一个或多个Room实例rooms，则可以使用Residence下标rooms通过可选链访问数组中的实际项目：

```
let johnsHouse = Residence()
johnsHouse.rooms.append(Room(name: "Living Room"))
johnsHouse.rooms.append(Room(name: "Kitchen"))
john.residence = johnsHouse

if let firstRoomName = john.residence?[0].name {
    print("The first room name is \(firstRoomName).")
} else {
    print("Unable to retrieve the first room name.")
}
// Prints "The first room name is Living Room."
```

访问可选类型的下标

如果下标返回可选类型的值（例如Swift Dictionary类型的键下标），则在下标的闭括号*之后*放置一个问号以链接其可选返回值：

```
var testScores = ["Dave": [86, 82, 84], "Bev": [79, 94, 81]]
testScores["Dave"]?[0] = 91
testScores["Bev"]?[0] += 1
testScores["Brian"]?[0] = 72
// the "Dave" array is now [91, 82, 84] and the "Bev" array is now [80, 94, 81]
```

上面的例子定义了一个名为的字典testScores，其中包含两个将键映射String到Int值数组的键 - 值对。该示例使用可选的链接将"Dave"数组中的第一项设置为91; 通过增加"Bev"数组中的第一项1; 并尝试将数组中的第一项设置为一个键"Brian"。前两个调用成功，因为该testScores字典包含"Dave"和的键"Bev"。第三次调用失败，因为testScores字典中不包含关键字"Brian"。

链接多个级别的链接

您可以将多个可选链接级别链接在一起，以深入到模型中更深的属性，方法和下标。但是，多级可选链接不会为返回的值增加更多级别的可选性。

换一种方式：

- 如果您尝试检索的类型不是可选的，则由于可选的链接，它将变为可选。

- 如果您尝试检索的类型*已经*是可选的，则由于链接，它将不会变得*更加*可选。

因此：

- 如果尝试`]`
- 同样，如果您尝试`Int?`通过可选链接检索值，`Int?`则无论使用多少级别的链接，总会返回一个值。

下面的例子试图访问`street`该财产`address`的财产`residence`的性质`john`。这里有*两个*可选的链接级别来链接`residence`和`address`属性，它们都是可选的类型：

```
if let johnsStreet = john.residence?.address?.street {
    print("John's street name is \(johnsStreet).")
} else {
    print("Unable to retrieve the address.")
}
// Prints "Unable to retrieve the address."
```

`john.residence`当前 的值包含有效的`Residence`实例。但是，`john.residence.address`目前的价值`nil`。因此，呼叫`john.residence?.address?.street`失败。

请注意，在上面的示例中，您试图检索该`street`属性的值。这个属性的类型是`String?`。`john.residence?.address?.street`因此`String?`，即使除了属性的基础可选类型之外还应用了两个可选链接级别，返回值也是如此。

如果您将实际`Address`实例设置为值`john.residence.address`，并为该地址的`street`属性设置实际值，则可以`street`通过多级可选链访问该属性的值：

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

if let johnsStreet = john.residence?.address?.street {
    print("John's street name is \(johnsStreet).")
} else {
    print("Unable to retrieve the address.")
}
// Prints "John's street name is Laurel Street."
```

在这个例子中，设置`address`属性的尝试`john.residence`会成功，因为`john.residence`当前的值包含有效的`Address`实例。

链接可选返回值的方法

前面的示例演示如何通过可选链接检索可选类型的属性的值。您还可以使用可选链来调用返回可选类型值的方法，并根据需要链接该方法的返回值。

下面的例子通过可选链接调用`Address`类的`buildingIdentifier()`方法。这个方法返回一个类型的值`String?`。如上所述，在可选链之后调用此方法的最终返回类型也是`String?`：

```
if let buildingIdentifier = john.residence?.address?.buildingIdentifier() {
    print("John's building identifier is \(buildingIdentifier).")
}
// Prints "John's building identifier is The Larches."
```

如果你想在这个方法的返回值进行进一步的可选链接，将链接可选问号*后，*该方法的括号：

```
if let beginsWithThe =
    john.residence?.address?.buildingIdentifier()?.hasPrefix("The") {
    if beginsWithThe {
        print("John's building identifier begins with \"The\".")
    } else {
        print("John's building identifier does not begin with \"The\".")
```

```
    }
}
// Prints "John's building identifier begins with "The"."
```

> 注意
>
> 在上面的例子中，您将可选链接问号*后*的括号内，因为你要串联上可选的值是`buildingIdentifier()`方法的返回值，而不是`buildingIdentifier()`方法本身。