



可选链

4.2

Swift 编程语言 / 可选链

这是一篇社区协同翻译的文章，你可以点击右边区块信息里的『改进』按钮向译者提交改进建议。

可选链是在当前可能为 `nil` 的可选值上查询和调用属性、方法和下标的过程。如果可选值有值，则属性、方法或下标调用成功；如果可选值为 `nil`，则属性、方法或下标调用返回 `nil`。多个查询可以链接在一起，如果链中的任何一个节点为 `nil`，整个链会返回失败。

注意

Swift 中的可选链类似于 Objective-C 中向 `nil` 传递消息，不过适用于任何类型，并且可以检查其成功与否。

可选链作为强制展开的代替品

如果可选项不为 `nil`，可以在可选值后面加上问号（`?`），从而指定可选链。这非常类似于在可选值之后放置感叹号（`!`）来强制展开它的值。主要的区别是，可选链在可选项为 `nil` 时只会调用失败，而强制展开在可选项为 `nil` 时会触发运行时错误。

为了反映可选链可以对 `nil` 值进行调用这一事实，可选链调用的结果总是一个可选值，即使正在查询的属性、方法或下标返回一个不可选值。可以使用这个可选的返回值来检查可选链调用是否成功（返回的可选项包含一个值），或者由于链中的 `nil` 值而没有成功（返回的可选值是 `nil`）。

具体来说，可选链调用的结果与预期返回值的类型相同，但包装在一个可选值中。一个平时返回 `Int` 的属性，在可选链中访问将返回 `Int?`。

接下来的几个代码片段演示了可选链与强制解包的区别，并且能够检查是否成功。

首先，定义两个类 `Person` 和 `Residence`：

Infinity

翻译进度

13
分块数量

9
参与人数



hisoka 翻译于 5个月前

0

重译

由 chdzq 审阅



VonZen 翻译于 5个月前

0

重译

由 Aufree 审阅



```
class Person {  
    var residence: Residence?  
}  
  
class Residence {  
    var numberOfRooms = 1  
}
```

`Residence` 实例有一个 `Int` 类型的属性 `numberOfRooms`，其默认值为 `1`。`Person` 实例有一个可选属性 `residence`，其类型为 `Residence?`。

如果你新建一个 `Person` 实例，由于 `residence` 是可选的，所以会被初始化为 `nil`。如下代码所示，`john` 拥有一个 `residence` 属性，其值为 `nil`：

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

如果你尝试访问此人 `residence` 的 `numberOfRooms` 属性，通过在 `residence` 后面放置一个感叹号来强制展开其值，则会触发运行时错误，因为没有值来解包：

```
let roomCount = john.residence!.numberOfRooms  
// 这里会触发运行时错误
```

上述代码在 `john.residence` 有非 `nil` 值并且将 `roomCount` 设置为包含适当房间数的 `Int` 值时会运行成功。但是如上所述，此代码在 `residence` 为 `nil` 时始终会触发运行时错误。

可选链提供了另一种访问 `numberOfRooms` 值的方法。要使用可选链，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.")  
}  
  
// 打印 "Unable to retrieve the number of rooms."
```

这样就告诉 Swift 在可选的 `residence` 属性上「链接」，如果存在 `residence` 则检索 `numberOfRooms` 的值。

因为访问 `numberOfRooms` 的尝试有可能失败，所以可选链尝试返回类型为 `Int?`，或者说「可选 `Int`」。如上例所示，当 `residence` 为 `nil`，这个可选的



hisoka 翻译于 5个月前

👍 0

重译

由 Aufree 审阅



hisoka 翻译于 5个月前

👍 0

重译

由 Aufree 审阅

`Int` 也将是 `nil`，这反映了无法访问 `numberOfRooms` 这一事实。通过可选绑定访问该可选的 `Int` 值来展开整数并且将非可选值赋值给 `roomCount`。

请注意，即使 `numberOfRooms` 是非可选的 `Int` 也是如此。这意味着通过可选链访问 `numberOfRooms` 将始终返回 `Int?` 而不是 `Int`。

你可以把一个 `Residence` 实例赋值给 `john.residence`，以使它不再是 `nil`：

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

`john.residence` 现在拥有了一个真正的 `Residence` 实例，而不是 `nil`。这时用和之前相同的可选链来访问 `numberOfRooms`，则会返回包含 `numberOfRooms` 默认值 `1` 的一个 `Int?`。

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

// 打印 "John's residence has 1 room(s)."
```

为可选链定义模型类

你可以使用可选链来调用超过一级深度的属性、方法和下标。这使你可以深入查看相互关联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。

下面的代码片段定义了四个模型类，用于后续几个示例，包括多级可选链的示例。这些类是从上面的 `Person` 和 `Residence` 扩展而来，添加了 `Room` 和 `Address` 类以及相关的属性、方法和下标。

`Person` 类的定义和之前一样：

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

`Residence` 类比之前复杂一点。这里定义一个变量属性 `rooms`，并初始化成类型为 `[Room]` 的空数组。

```
class Residence {
    var rooms = [Room]()
}
```



hisoka 翻译于 5个月前

👍 0 重译

由 Aufree 审阅

```

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?
}

```

由于这个版本的 `Residence` 储存了一个 `Room` 的数组示例，所以

`numberOfRooms` 属性变成了计算属性，而不是存储属性。计算出的

`numberOfRooms` 属性返回 `rooms` 数组的 `count` 属性的值。

这个版本的 `Residence` 提供了一个可读写的下标作为访问 `rooms` 数组的快捷方式，可以访问 `rooms` 数组对应索引值下的房间。

`Residence` 同时提供了一个 `printNumberOfRooms` 方法来打印房间数量。

最后，`Residence` 定义了一个 `Address?` 类型的可选属性

`address`，`Address` 类会在随后定义。

`rooms` 数组中的 `Room` 类只有一个 `name` 属性，以及一个初始化构造函数来将该属性设置为合适的房间名：

```

class Room {
    let name: String

    init(name: String) { self.name = name }
}

```

最后是 `Address` 类。这个类有3个 `String?` 类型的可选属性。前两个属性

`buildingName` 和 `buildingNumber` 是该地址建筑物的标识。第三个属性

`street` 是该地址的街道名：

```

class Address {
    var buildingName: String?

```



hisoka 翻译于 5个月前

👍 0

重译

由 Aufree 审阅

```
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
    }
}
```

`Address` 类同时提供了一个 `buildingIdentifier()` 方法，其返回值类型为 `String?`。这个方法检查地址的属性，如果 `buildingNumber` 和 `street` 都有值则把它们连接起来并返回，如果 `buildingName` 有值则返回其值，否则返回 `nil`。

通过可选链访问属性

就像这篇文档演示的一样，[可选链作为一种强制解包的可选方式](#)，你可以用可选链访问可选类型值得属性，并检查是否成功访问。

用上面定义类来创建一个新的 `Person` 实例，并跟前面一样去访问该实例的 `numberOfRooms` 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.")
}
// 打印 "Unable to retrieve the number of rooms."
```

由于 `john.residence` 的值是 `nil`，所以此次可选链的调用仍然会失败。

你也可以试着用可选链去设置属性的值：

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



ImAPIs 翻译于 5个月前

👍 0

重译

由 Aufree 审阅

```
john.residence?.address = someAddress
```

在该例中，因为 `john.residence` 当前的值为 `nil`，所以试图设置

`john.residence` 的 `address` 属性的值将会失败。

赋值是可选链的一部分，也就意味着 `=` 操作符的右操作数不会被计算。而之所以前面的例子不容易看出来 `someAddress` 的值从未被计算，那是因为访问常量没有副作用。接下来看下面的代码片段，它同样是赋值，但是是用函数创建 `Address` 的实例。“函数被调用”在函数返回值之前就打印出来了，这样你可以清楚的看到是否 `=` 右操作数被计算了。

```
func createAddress() -> Address {
    print("函数被调用。")

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

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

因为什么都没有打印，所以你可以区分出来函数 `createAddress()` 未被调用。

通过可选链调用方法

你可以使用可选链来调用一个可选值的方法，以及检查调用是否成功。即使那个方法没有返回值你依然可以这样做。

`Residence` 的 `printNumberOfRooms()` 方法打印当前 `numberOfRooms` 的值。

代码如下：

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

这个方法没有指定返回类型。但是，没有返回值的函数和方法会有一个隐式返回类型

`Void`，可在此查看 [无返回值函数](#)。这意味着会返回一个 `()`，或者说一个空的元组。

如果你用可选链调用一个可选值的方法，这个方法的返回类型会是 `Void?`，而不是

`Void`。因为可选链永远返回可选值。这使得你可以使用 `if` 表达式来检查是否可以调用 `printNumberOfRooms()` 方法，即使这个方法本身没有定义返回值。将



hisoka 翻译于 5个月前

👍 0

重译

由 RecherJ 审阅

`printNumberOfRooms` 的返回值与 `nil` 进行比较来查看方法是否调用成功：

```
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.")
}

// 打印 "It was not possible to print the number of rooms."
```

通过可选链尝试给一个属性赋值也是同样。这个例子 [通过可选链访问属性](#) 尝试给

`john.residence` 设置一个 `address` 值，即便这时 `residence` 属性是个

`nil`。任何通过可选链给属性赋值的尝试都会返回一个 `Void?` 类型的值。这样你可

以和 `nil` 比较来检查赋值是否成功：

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

// 打印 "It was not possible to set the address."
```

通过可选链访问下标

你可以使用可选链尝试从可选值的下标中检索和设置值，并检查该下标调用是否成功。

注意

当你通过可选链访问一个可选值的下标时，在下标中括号 *前面* 放置问号，而不是后面。

可选链的问号永远是直接跟在可选表达式后面。

下面的示例尝试使用 `Residence` 类中定义的下标检索 `john.residence` 属性的

`rooms` 数组中第一个房间的名称。由于 `john.residence` 当前是 `nil`，下标访

问失败：

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



hisoka 翻译于 5个月前

👍 0

重译

由 Aufree 审阅

```
// 打印 "Unable to retrieve the first room name."
```

这里的问号直接跟在 `john.residence` 后面，下标中括号的前面。因为是在 `john.residence` 这个可选值上尝试取值。

类似的，你可以尝试通过可选链下标来设置一个新的值：

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

在这里赋值的尝试都会失败，因为 `residence` 当前是 `nil`。

如果你创建一个 `Residence` 实例并赋值给 `john.residence`，并且 `rooms` 数组中有一个或更多 `Room` 实例，你可以通过可选链来真正地用 `Residence` 下标访问到 `rooms` 数组里的实际内容：

```
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.")
}

// 打印 "The first room name is Living Room."
```

访问可选类型的下标

如果下标返回一个可选类型的值，例如 Swift 中 `Dictionary` 的键下标，在下标的右括号后面放置一个问号来链接其可选的返回值：

```
var testScores = ["Dave": [86, 82, 84], "Bev": [79, 94, 81]]
testScores["Dave"]?[0] = 91
testScores["Bev"]?[0] += 1
testScores["Brian"]?[0] = 72

// 「Dave」 数组现在是 [91, 82, 84]，「Bev」 数组现在是 [80, 94, 81]
```

上述示例定义了一个 `testScores` 字典，它包含两个键值对，将 `String` 类型的键映射到 `Int` 值的数组。该例中使用可选链将 `Dave` 数组中的第一个元素置为 `91`；将 `Bev` 数组的第一个元素加 `1`；尝试向 `Brian` 的第一个元素设一个



hisoka 翻译于 5个月前

👍 0

重译

由 Aufree 审阅

值。前两个调用成功，因为 `testScores` 字典包含 `Dave` 和 `Bev` 两个键。第三个调用失败，因为 `testScores` 不包含 `Brian` 键。

多级链表关联

你可以关联多层次的可选链，以深入到模型中更深层次的属性、方法和下标。但是，多个可选链表级别不增加返回值的可选级别。

换个说法：

- 如果要检索的类型不是可选的，通过可选链，它将成为可选的。
- 如果您要检索的类型已经是可选的，那么它将保持原状。

因此：

- 如果你试图通过可选链检索一个 `Int` 值，那么不管用了多少链，总是返回一个 `Int?`。
- 同样，你试图通过可选链检索一个 `Int?` 值，无论使用多少链，也总是返回一个 `Int?`。

下面的例子尝试访问 `John` 中的 `Residence` 属性中的 `Address` 属性中的 `Street` 属性。这里使用了两层可选链式调用，`Residence` 以及 `Address` 都是可选值：

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

// 打印 "Unable to retrieve the address."
```

`john.residence` 现在包含一个有效的 `Residence` 实例。然而，`john.residence.address` 的值当前为 `nil`。因此，调用 `john.residence?.address?.street` 会失败。

需要注意的是，上面的例子中，`Street` 的属性为 `String?`。 `john.residence?.address?.street` 的返回值也依然是 `String?`，即使已经使用了两层可选链式调用。

如果为 `john.residence.address` 赋值一个 `Address` 实例，并且为 `Address` 中的 `Street` 属性设置一个有效值，我们就能过通过可选链式调用来访



liuzhanjing 翻译于 5个月前

0

重译

由 RecherJ 审阅



Kevin 翻译于 5个月前

0

重译

由 Summer 审阅

问 `Street` 属性:

```
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.")
}

// 打印 "John's street name is Laurel Street."
```

在上面的例子中，因为 `john.residence` 包含一个有效的 `Address` 实例，所以对 `john.residence` 的 `Address` 属性赋值将会成功。

在方法的可选返回值上进行可选链式调用

上面的例子展示了如何在一个可选值上通过可选链式调用来获取它的属性值。我们还可以在一个可选值上通过可选链式调用来调用方法，并且可以根据需要继续在方法的可选返回值上进行可选链式调用。

在下面的例子中，通过可选链式调用来调用 `Address` 的 `buildingIdentifier()` 方法。这个方法返回 `String?` 类型的值。如上所述，通过可选链式调用来调用该方法，最终的返回值依旧会是 `String?` 类型：

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

// 打印 "John's building identifier is The Larches."
```

如果要在该方法的返回值上进行可选链式调用，在方法的圆括号后面加上问号即可：

```
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\".")
    }
}
```



Kevin 翻译于 5个月前

👍 0 重译

由 Summer 审阅

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

注意

在上面的例子中，在方法的圆括号后面加上问号是因为你要在

`buildingIdentifier()` 方法的可选返回值上进行可选链式调用，而不是
`buildingIdentifier()` 方法本身。

本文中的所有译文仅用于学习和交流目的，转载请务必注明文章译者、出处、和本文链接

我们的翻译工作遵照 [CC 协议](#)，如果我们的工作有侵犯到您的权益，请及时联系我们。

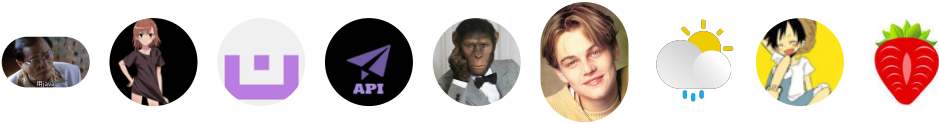
← 上一篇

下一篇 →

👍 点赞



👤 参与译者：9



更多职位





💬 讨论数量：0

✎ 发起讨论

☐ 只看当前版本讨论

暂无话题~




兄弟社区

-  [Laravel China](#)
-  [PythonCaff.com](#)
-  [GolangCaff.com](#)
-  [VuejsCaff.com](#)

资源推荐

资源推荐

其他信息

-  [软件外包](#)
-  [商务合作](#)
-  [联系站长](#)