翻译：Jasonbroker　校对：numbbbbb

可选链 Optional Chaining

本页包含内容：

可选链（Optional Chaining）是一种可以请求和调用属性、方法及子脚本的过程，它的自判断性体现于请求或调用的目标当前可能为空（`nil`）。如果自判断的目标有值，那么调用就会成功；相反，如果选择的目标为空（`nil`），则这种调用将返回空（`nil`）。多次请求或调用可以被链接在一起形成一个链，如果任何一个节点为空（`nil`）将导致整个链失效。

注意： Swift 的自判断链和 Objective-C 中的消息为空有些相像，但是 Swift 可以使用在任意类型中，并且失败与否可以被检测到。

可选链可替代强制解析

通过在想调用的属性、方法、或子脚本的可选值（`optional value`）（非空）后面放一个问号，可以定义一个可选链。这一点很像在可选值后面放一个声明符号来强制拆得其封包内的值。他们的主要的区别在于当可选值为空时可选链即刻失败，然而一般的强制解析将会引发运行时错误。

为了反映可选链可以调用空（`nil`），不论你调用的属性、方法、子脚本等返回的值是不是可选值，它的返回结果都是一个可选值。你可以利用这个返回值来检测你的可选链是否调用成功，有返回值即成功，返回nil则失

败。

调用可选链的返回结果与原本的返回结果具有相同的类型，但是原本的返回结果被包装成了一个可选值，当可选链调用成功时，一个应该返回Int的属性将会返回Int?。

下面几段代码将解释可选链和强制解析的不同。

首先定义两个类Person和Residence。

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

class Residence {
    var numberOfRooms = 1
}
```

Residence具有一个Int类型的numberOfRooms，其值为 1。Person具有一个自判断residence属性，它的类型是Residence? 。

如果你创建一个新的Person实例，它的residence属性由于是被定义为自判断型的，此属性将默认初始化为空：

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

如果你想使用感叹号（!）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。

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

当john.residence不是nil时，会运行通过，且会将roomCount 设置为一个int类型的合理值。然而，如上所述，当residence为空时，这个代码将会导致运行时错误。

可选链提供了一种另一种获得numberOfRooms的方法。利用可选链，使用问号来代替原来!的位置：

```
if let roomCount = john.residence?.numberOfRooms {
```

```
    println("John's residence has \(roomCount)
room(s).")
} else {
    println("Unable to retrieve the number of
rooms.")
}
// 打印 "Unable to retrieve the number of rooms.
```

这告诉 Swift 来链接自判断`residence?`属性，如果`residence`存在则取回`numberOfRooms`的值。

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“自判断`Int`”。当`residence`是空的时候（上例），选择`Int`将会为空，因此会出先无法访问`numberOfRooms`的情况。

要注意的是，即使numberOfRooms是非自判断`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`。

你可以自己定义一个`Residence`实例给`john.residence`，这样它就不再为空了：

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

`john.residence` 现在有了实际存在的实例而不是nil了。如果你想使用和前面一样的可选链来获得`numberOfRoooms`，它将返回一个包含默认值 1 的`Int?`：

```
if let roomCount = john.residence?.numberOfRooms {
    println("John's residence has \(roomCount)
room(s).")
} else {
    println("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多层调用属性，方法，和子脚本。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。

后面的代码定义了四个将在后面使用的模型类，其中包括多层可选链。这些类是由上面的`Person`和`Residence`模型通过添加一个`Room`和一个`Address`类拓展来。

`Person`类定义与之前相同。

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

`Residence`类比之前复杂些。这次，它定义了一个变量`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() {
        println("The number of rooms is \
(numberOfRooms)")
    }
    var address: Address?
}
```

因为`Residence`存储了一个`Room`实例的数组，它的`numberOfRooms`属性值不是一个固定的存储值，而是通过计算而来的。`numberOfRooms`属性值是由返回`rooms`数组的`count`属性值得到的。

为了能快速访问`rooms`数组，`Residence`定义了一个只读的子脚本，通过

插入数组的元素角标就可以成功调用。如果该角标存在，子脚本则将该元素返回。

Residence中也提供了一个printNumberOfRooms的方法，即简单的打印房间个数。

最后，Residence定义了一个自判断属性叫address（address?）。Address类的属性将在后面定义。 用于rooms数组的Room类是一个很简单的类，它只有一个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 }
}
```

这个模型中的最终类叫做Address。它有三个自判断属性他们额类型是String?。前面两个自判断属性buildingName和 buildingNumber作为地址的一部分，是定义某个建筑物的两种方式。第三个属性street，用于命名地址的街道名：

```
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if buildingName {
            return buildingName
        } else if buildingNumber {
            return buildingNumber
        } else {
            return nil
        }
    }
}
```

Address类还提供了一个buildingIdentifier的方法，它的返回值类型为String?。这个方法检查buildingName和buildingNumber的属性，如果buildingName有值则将其返回，或者如果buildingNumber有值则

将其返回，再或如果没有一个属性有值，返回空。

通过可选链调用属性

正如上面“ 可选链可替代强制解析”中所述，你可以利用可选链的可选值获取属性，并且检查属性是否获取成功。然而，你不能使用可选链为属性赋值。

使用上述定义的类来创建一个人实例，并再次尝试后去它的`numberOfRooms`属性：

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

由于`john.residence`是空，所以这个可选链和之前一样失败了，但是没有运行时错误。

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。

`Residence`的`printNumberOfRooms`方法会打印`numberOfRooms`的当前值。方法如下：

```
func printNumberOfRooms(){
    println(“The number of rooms is \
```

```
(numberOfRooms)”)
}
```
这个方法没有返回值。但是，没有返回值类型的函数和方法有一个隐式的返回值类型`Void`（参见Function Without Return Values）。

如果你利用可选链调用此方法，这个方法的返回值类型将是`Void?`，而不是`Void`，因为当通过可选链调用方法时返回值总是可选类型（optional type）。，即使是这个方法本是没有定义返回值，你也可以使用`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`：

```
if john.residence?.printNumberOfRooms() {
    println("It was possible to print the number of
rooms.")
} else {
    println("It was not possible to print the number
of rooms.")
}
// 打印 "It was not possible to print the number of
rooms."。
```

使用可选链调用子脚本

你可以使用可选链来尝试从子脚本获取值并检查子脚本的调用是否成功，然而，你不能通过可选链来设置子代码。

注意： 当你使用可选链来获取子脚本的时候，你应该将问号放在子脚本括号的前面而不是后面。可选链的问号一般直接跟在自判断表达语句的后面。

下面这个例子用在`Residence`类中定义的子脚本来获取`john.residence`数组中第一个房间的名字。因为`john.residence`现在是`nil`，子脚本的调用失败了。

```
if let firstRoomName = john.residence?[0].name {
    println("The first room name is \
```

```
(firstRoomName).")
} else {
    println("Unable to retrieve the first room
name.")
}
// 打印 "Unable to retrieve the first room name."。
```

在子代码调用中可选链的问号直接跟在`john.residence`的后面，在子脚本括号的前面，因为`john.residence`是可选链试图获得的可选值。

如果你创建一个`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`子脚本来获取在`rooms`数组中的实例了：

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

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

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子脚本。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。 也就是说：

如果你试图获得的类型不是可选类型，由于使用了可选链它将变成可选类型。 如果你试图获得的类型已经是可选类型，由于可选链它也不会提高

自判断性。

因此：

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?`。

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，他们两者都是可选类型：

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

`john.residence`的值现在包含一个`Residence`实例，然而`john.residence.address`现在是`nil`，因此`john.residence?.address?.street`调用失败。

从上面的例子发现，你试图获得`street`属性值。这个属性的类型是`String?`。因此尽管在可选类型属性前使用了两层可选链，`john.residence?.address?.street`的返回值类型也是`String?`。

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。

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

if let johnsStreet = john.residence?.address?.street
{
```

```
    println("John's street name is \(johnsStreet).")
} else {
    println("Unable to retrieve the address.")
}
// 打印 "John's street name is Laurel Street."。
```

值得注意的是，“!”符的在定义address实例时的使用（john.residence.address）。john.residence属性是一个可选类型，因此你需要在它获取address属性之前使用!解析以获得它的实际值。

链接自判断返回值的方法

前面的例子解释了如何通过可选链来获得可选类型属性值。你也可以通过调用返回可选类型值的方法并按需链接方法的返回值。

下面的例子通过可选链调用了Address类中的buildingIdentifier 方法。这个方法的返回值类型是String?。如上所述，这个方法在可选链调用后最终的返回值类型依然是String?：

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

如果你还想进一步对方法返回值执行可选链，将可选链问号符放在方法括号的后面：

```
if let upper =
john.residence?.address?.buildingIdentifier()?.upperc
aseString {
    println("John's uppercase building identifier is
\(upper).")
}
// 打印 "John's uppercase building identifier is THE
```

`LARCHES."`。

注意： 在上面的例子中，你将可选链问号符放在括号后面是因为你想要链接的可选值是`buildingIdentifier`方法的返回值，不是`buildingIdentifier`方法本身。