自判断链接（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。

**自判断链接可替代强制拆包**

通过在想调用的属性、方法、或子脚本的自判断值（optional value）（非空）后面放一个问号，可以定义一个自判断链接。这一点很像在自判断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自判断值为空时自判断链接即刻失败，然而一般的强制拆包将会引发运行时错误。

为了反映自判断链接可以调用空（nil），不论你调用的属性、方法、子脚本等返回的值是不是自判断值，它的返回结果都是一个自判断值。你可以利用这个返回值来检测你的自判断链接是否调用成功，有返回值即成功，返回nil则失败。

调用自判断链接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自判断值，当自判断链接调用成功时，一个应该返回Int的属性将会返回Int？。

下面几段代码将解释自判断链接和强制拆包的不同。

首先定义两个类Person和Residence。

1. **class** Person {
2. **var** residence: Residence?
3. }
5. **class** Residence {
6. **var** numberOfRooms = 1
7. }

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：

1. let john = Person()

如果你想使用声明符！强制拆包获得这个人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拆包的residence值。

1. let roomCount = john.residence!.numberOfRooms
2. // this triggers a runtime error”
3. //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。

自判断链接提供了一种另一种获得numberOfRooms的方法。利用自判断链接，使用问号来代替原来！的位置：

1. **if** let roomCount = john.residence?.numberOfRooms {
2. println("John's residence has \(roomCount) room(s).")
3. } **else** {
4. println("Unable to retrieve the number of rooms.")
5. }
6. // 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，自判断链接会返回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。只要是通过自判断链接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：

1. john.residence = Residence()
2. 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自判断链接来获得numberOfRoooms，它将返回一个包含默认值1的Int？：
4. **if** let roomCount = john.residence?.numberOfRooms {
5. println("John's residence has \(roomCount) room(s).")
6. } **else** {
7. println("Unable to retrieve the number of rooms.")
8. }
9. // 打印 "John's residence has 1 room(s)"。

**为自判断链接定义模型类**

你可以使用自判断链接来多层调用属性，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，其中包括多层自判断链接。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。

1. **class** Person {
2. **var** residence: Residence?
3. }

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：

1. **class** Residence {
2. **var** rooms = Room[]()
3. **var** numberOfRooms: Int {
4. **return** rooms.count
5. }
6. subscript(i: Int) -> Room {
7. **return** rooms[i]
8. }
9. func printNumberOfRooms() {
10. println("The number of rooms is \(numberOfRooms)")
11. }
12. **var** address: Address?
13. }

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。

1. **class** Room {
2. let name: String
3. init(name: String) { self.name = name }
4. }

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：

1. **class** Address {
2. **var** buildingName: String?
3. **var** buildingNumber: String?
4. **var** street: String?
5. func buildingIdentifier() -> String? {
6. **if** buildingName {
7. **return** buildingName
8. } **else** **if** buildingNumber {
9. **return** buildingNumber
10. } **else** {
11. **return** nil
12. }
13. }
14. }

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。

**通过自判断链接调用属性**

正如上面“ 自判断链接可替代强制拆包”中所述，你可以利用自判断链接的自判断值获取属性，并且检查属性是否获取成功。然而，你不能使用自判断链接为属性赋值。

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：

1. let john = Person()
2. **if** let roomCount = john.residence?.numberOfRooms {
3. println("John's residence has \(roomCount) room(s).")
4. } **else** {
5. println("Unable to retrieve the number of rooms.")
6. }
7. // 打印 "Unable to retrieve the number of rooms。

由于john.residence是空，所以这个自判断链接和之前一样失败了，但是没有运行时错误。

**通过自判断链接调用方法**

你可以使用自判断链接的来调用自判断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，你依然可以使用自判断链接来达成这一目的。

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

1. func printNumberOfRooms(){
2. println(“The number of rooms is \(numberOfRooms)”)
3. }

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）。

如果你利用自判断链接调用此方法，这个方法的返回值类型将是Void？，而不是Void，因为当通过自判断链接调用方法时返回值总是自判断类型（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：如果方法通过自判断链接调用成功，printNumberOfRooms的隐式返回值将会是Void，如果没有成功，将返回nil：

1. **if** john.residence?.printNumberOfRooms() {
2. println("It was possible to print the number of rooms.")
3. } **else** {
4. println("It was not possible to print the number of rooms.")
5. }
6. // 打印 "It was not possible to print the number of rooms."。

**使用自判断链接调用子脚本**

你可以使用自判断链接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，你不能通过自判断链接来设置子代码。

注意： 当你使用自判断链接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。自判断链接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。

1. **if** let firstRoomName = john.residence?[0].name {
2. println("The first room name is \(firstRoomName).")
3. } **else** {
4. println("Unable to retrieve the first room name.")
5. }
6. // 打印 "Unable to retrieve the first room name."。

在子代码调用中自判断链接的问号直接跟在john.residence的后面，在子脚本括号的前面，因为john.residence是自判断链接试图获得的自判断值。

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，那么你可以使用自判断链接通过Residence子脚本来获取在rooms数组中的实例了：

1. let johnsHouse = Residence()
2. johnsHouse.rooms += Room(name: "Living Room")
3. johnsHouse.rooms += Room(name: "Kitchen")
4. john.residence = johnsHouse
6. **if** let firstRoomName = john.residence?[0].name {
7. println("The first room name is \(firstRoomName).")
8. } **else** {
9. println("Unable to retrieve the first room name.")
10. }
11. // 打印 "The first room name is Living Room."。

**连接多层链接**

你可以将多层自判断链接连接在一起，可以掘取模型内更下层的属性方法和子脚本。然而多层自判断链接不能再添加比已经返回的自判断值更多的层。 也就是说：

如果你试图获得的类型不是自判断类型，由于使用了自判断链接它将变成自判断类型。 如果你试图获得的类型已经是自判断类型，由于自判断链接它也不会提高自判断性。

因此，如果你试图通过自判断链接获得Int值，不论使用了多少层链接返回的总是Int？。 相似的，如果你试图通过自判断链接获得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。这里使用了两层自判断链接来联系residence和address属性，他们两者都是自判断类型：

1. **if** let johnsStreet = john.residence?.address?.street {
2. println("John's street name is \(johnsStreet).")
3. } **else** {
4. println("Unable to retrieve the address.")
5. }
6. // 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？。因此尽管在自判断类型属性前使用了两层自判断链接，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，你可以通过多层自判断链接来得到这个属性值。

1. let johnsAddress = Address()
2. johnsAddress.buildingName = "The Larches"
3. johnsAddress.street = "Laurel Street"
4. john.residence!.address = johnsAddress
6. **if** let johnsStreet = john.residence?.address?.street {
7. println("John's street name is \(johnsStreet).")
8. } **else** {
9. println("Unable to retrieve the address.")
10. }
11. // 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属性是一个自判断类型，因此你需要在它获取address属性之前使用！拆包以获得它的实际值。

**链接自判断返回值的方法**

前面的例子解释了如何通过自判断链接来获得自判断类型属性值。你也可以通过调用返回自判断类型值的方法并按需链接方法的返回值。

下面的例子通过自判断链接调用了Address类中的buildingIdentifier 方法。这个方法的返回值类型是String?。如上所述，这个方法在自判断链接调用后最终的返回值类型依然是String？：

1. **if** let buildingIdentifier = john.residence?.address?.buildingIdentifier() {
2. println("John's building identifier is \(buildingIdentifier).")
3. }
4. // 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自判断链接，将自判断链接问号符放在方法括号的后面：

1. **if** let upper = john.residence?.address?.buildingIdentifier()?.uppercaseString {
2. println("John's uppercase building identifier is \(upper).")
3. }
4. // 打印 "John's uppercase building identifier is THE LARCHES."。

注意： 在上面的例子中，你将自判断链接问号符放在括号后面是因为你想要链接的自判断值是buildingIdentifier方法的返回值，不是buildingIdentifier方法本身。