14 CHAPTER 選項串連

選項串連 (optional chaining)是用以查詢和呼叫屬性、方法和索引，在選項的情況有可能是 nil。若在選項的情況下包含一值，則屬性、方法或索引的呼叫將會是成功的，若選項是 nil，則屬性、方法或索引將回傳 nil。多個串連可以串連在一起，但有任何一串連為 nil 將會使整個選項串連宣告失敗。

14.1 選項串連可當做強迫解開的方法

您可以在想要呼叫屬性、方法或索引，在選項的後面加上問號(?) 來指定選項串連，看看選項是否不是 nil。這和將驚嘆號(!)置於選項後很相似，它用以解開其值。我們還是以範例來加以說明，如下所示：

範例程

```
01 // optional chaining
02 class Student {
03     var dorm: Dormitory?
04 }
05
06 class Dormitory {
07     var numberOfRooms = 2
08 }
09
10 let peter = Student()
11 let rooms = peter.dorm?.numberOfRooms
12 print("Dormitory has \(rooms) rooms")
```

輸出結果

```
Dormitory has nil rooms
```

上述程式表示 Student 類別有一變數 dorm，其型態為選項型態 Dormitory? 類別。而 Dormitory 類別有一變數 numberOfRooms，其初始值為 2。

注意，此時的 dorm 是 Dormitory? 的選擇型態變數，預設值是 nil，如輸出結果所示。

若將上述的

```
let rooms = peter.dorm?.numberOfRooms
```

改為

```
let rooms = peter.dorm!.numberOfRooms
```

將會產生以下的錯誤訊息：

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

主要的原因是無物件時不可使用！強迫印出。那應如何修正呢？我們需要在

```
let peter = Student()
```

加上

```
peter.dorm = Dormitory()
```

就可以了，其輸出結果如下：

```
Dormitory has 2 rooms
```

完整的程式如下所示：

範例程式

```
01 // optional chaining
02 class Student {
03     var dorm: Dormitory?
04 }
05
06 class Dormitory {
07     var numberOfRooms = 2
08 }
09
10 let peter = Student()
11 peter.dorm = Dormitory()
12 let rooms = peter.dorm!.numberOfRooms
13 print("Dormitory has \(rooms) rooms")
```


若將上一程式的

```
let rooms = peter.dorm!.numberOfRooms
```

改為

```
let rooms = peter.dorm?.numberOfRooms
```

則輸出結果如下：

```
Dormintory has Optional(2) rooms
```

差異在於多了 Optional 和小括號的字眼而已。

14.2 經由選項串連呼叫屬性、方法

為了解釋選項串連起見，我們將程式再加以擴充。如下所示：

```
class Student {
    var dorm: Dormitory?
}

class Dormitory {
    var numberOfRooms: Int

    func printNumberOfRooms() {
        print("The number of rooms is \(numberOfRooms)")
    }
    init(numberOfRooms: Int) {
        self.numberOfRooms = numberOfRooms
    }
    var location: Location?
}
```

```
class Location {
    var dormitoryName: String?
    var street: String?
}
```

類別之間的關係圖如下所示：

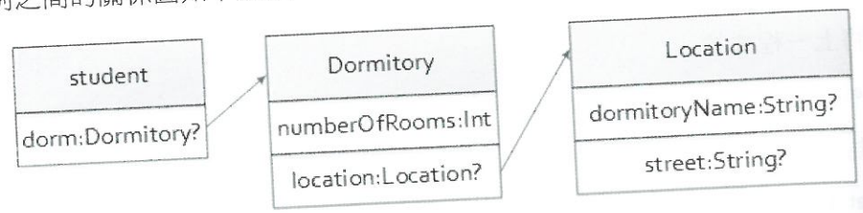

圖 14-1

讀者可配合此圖形幫助您了解，如何經由選項串連呼叫屬性或方法，以及多向的串連。以下的程式皆會用到上述的片段程式。

14.2.1 經由選項串連呼叫屬性

如何經由選項串連呼叫屬性，在本章第一個範例已大略看過了，此處以另一方式說明。定義一 peter 為 Student 類別變數，然後判斷其宿舍有多少房間，如下所示：

範例程式

```
01  class Student {
02      var dorm: Dormitory?
03  }
04
05  class Dormitory {
06      var numberOfRooms: Int
07
08      func printNumberOfRooms() {
09          print("The number of rooms is \(numberOfRooms)")
10      }
11      init(numberOfRooms: Int) {
12          self.numberOfRooms = numberOfRooms
13      }
14      var location: Location?
```



```
15  }
16
17  class Location {
18      var dormitoryName: String?
19      var street: String?
20  }
21
22  let peter = Student()
23  if let roomNumber = peter.dorm?.numberOfRooms {
24      print("Peter's dormitory has \(roomNumber) rooms")
25  } else {
26      print("Unabel to retrieve the number of rooms")
27  }
```

輸出結果

```
Unabel to retrieve the number of rooms
```

因為選項型態的預設值為 nil，所以判斷式為假，此時執行 else 所對應的敘述。主要的原因是無法得知 numberOfRooms 的屬性。若將上述的程式在 if 判斷式前加上

```
peter.dorm = Dormitory(numberOfRooms: 10)
```

如下所示：

```
22  let peter = Student()
23  peter.dorm = Dormitory(numberOfRooms: 10)
24  if let roomNumber = peter.dorm?.numberOfRooms {
25      print("Peter's dormitory has \(roomNumber) rooms")
26  } else {
27      print("Unabel to retrieve the number of rooms")
28  }
```

此時的輸出結果如下：

```
Peter's dormitory has 10 rooms
```

很清楚的得知，peter 的 dorm 屬性一定要指定 Dormitory 類別的實例給它，因為初始值為 10, 所以輸出結果是 10 。

14.2.2 經由選項串連呼叫方法

看懂了如何從選項串連呼叫屬性後，那如何從選項呼叫方法就易懂了。我們想辦法呼叫 printNumberOfRooms 方法。此方法是印出宿舍有多方房間 numberOfRooms 的屬性。

範例程式

```
01  class Student {
02      var dorm: Dormitory?
03  }
04
05  class Dormitory {
06      var numberOfRooms: Int
07
08      func printNumberOfRooms() {
09          print("The number of rooms is \(numberOfRooms)")
10      }
11      init(numberOfRooms: Int) {
12          self.numberOfRooms = numberOfRooms
13      }
14      var location: Location?
15  }
16
17  class Location {
18      var dormitoryName: String?
19      var street: String?
20  }
21
22  // call method
23  let peter = Student()
24  peter.dorm = Dormitory(numberOfRooms: 10)
25  peter.dorm!.printNumberOfRooms()
```

輸出結果

```
The number of rooms is 10
```


程式中最後一個敘述

```
peter.dorm!.printNumberOfRooms()
```

是經由選項型態串連呼叫 printNumberOfRooms 方法。

14.3 多重的串連

接下來，我們將 Location 類別定義一實例 peterLocation。然後指定 Location 類別實例 peterLocation 的屬性。最後將其指定給 peter.dorm?.location。

範例程式

```
01  class Student {
02      var dorm: Dormitory?
03  }
04
05  class Dormitory {
06      var numberOfRooms: Int
07
08      func printNumberOfRooms() {
09          print("The number of rooms is \(numberOfRooms)")
10      }
11      init(numberOfRooms: Int) {
12          self.numberOfRooms = numberOfRooms
13      }
14      var location: Location?
15  }
16
17  class Location {
18      var dormitoryName: String?
19      var street: String?
20  }
21
22  // multiple chain
23  let peter = Student()
24  peter.dorm = Dormitory(numberOfRooms: 10)
25  let peterLocation = Location()
26  peterLocation.dormitoryName = "BigHouse Building"
27  peterLocation.street = "Hsingchang 777"
```

```
peter.dorm?.location = peterLocation

print(peter.dorm?.location?.dormitoryName)
print(peter.dorm?.location?.street)
```

輸出結果

```
Optional(BigHouse Building)
Optional(Hsingchang 777)
```

其中 peter.dorm?.location?.dormitoryName 與 peter.dorm?.location?.street 皆為所謂的多重串連。有幾個點代表它有幾層的意思。

自我練習題

1. 以下的程式皆有些許的 bugs，可否大家一起來練功。

(a)

```
// optional chaining
class Student {
    var dorm: Dormitory
}

class Dormitory {
    var numberOfRooms = 2
}

let peter = Student()
let rooms = peter.dorm!.numberOfRooms
print("Dormitory has \(rooms) rooms")
```

(b)

```
class Student {
    var dorm: Dormitory?
}

class Dormitory {
    var numberOfRooms: Int

    func printNumberOfRooms() {
        print("The number of rooms is \(numberOfRooms)")
    }
    init(numberOfRooms: Int){
        self.numberOfRooms = numberOfRooms
    }
    var location: Location?
}

class Location {
    var dormitoryName: String?
    var street: String?
}

let peter = Student()
peter.dorm = Dormitory()
if let roomNumber = peter.dorm?.numberOfRooms {
```

```
    print("Peter's dormitory has \(roomNumber) rooms")
} else {
    print("Unabel to retrieve the number of rooms")
}
```

(c)

```
class Student {
    var dorm: Dormitory?
}

class Dormitory {
    var numberOfRooms: Int

    func printNumberOfRooms() {
        print("The number of rooms is \(numberOfRooms)")
    }
    init(numberOfRooms: Int){
        self.numberOfRooms = numberOfRooms
    }
    var location: Location?
}

class Location {
    var dormitoryName: String?
    var street: String?
}

// multiple chain
let peter = Student()
let peterLocation = Location()
peterLocation.dormitoryName = "BigHouse Building"
peterLocation.street = "Hsingchang 777"
peter.dorm?.location = peterLocation

print(peter.dorm?.location?.dormitoryName)
print(peter.dorm?.location?.street)
```


15 CHAPTER 型態轉換與延展

型態轉換 (type casting) 用於檢查是否為某一實例的型態，或是在類別的架構做為轉型用。在 Swift 中利用 is 和 as 運算子來完成轉換的事項。

延展 (extension) 是將現存在的類別、結構或列舉加入一些新的功能。這包括可以延展在原本的程式碼無法存取的型態功能。以下我們將一一的討論此兩項主題。

15.1 檢查型態

我們以範例來說明，以下有三個類別，分別為 University、Teacher 以及 Student。而 Teacher 與 Student 類別皆繼承 University 類別。程式如下所示：

範例程式

```
01  class University {
02      var name: String
03      init(name: String) {
04          self.name = name
05      }
06  }
07
08  class Teacher: University {
09      var status: String
10      init(name: String, status: String) {
11          self.status = status
```