为代码的执行做个决定

几乎所有语言都有的条件判断和循环

(https://www.boxueio.com/series/make-a-decision/ebook/8)

使用高级样式匹配方式

(https://www.boxueio.com/series/make-a-decision/ebook/136)

使用简单的样式匹配

Back to series (/series/make-a-decision)

除了我们在上一节中提到的基本循环和分支控制之外，现实环境中，我们需要的判断条件可能比那些演示的例子复杂的多。为此，Swift从函数式编程中借鉴了一些样式匹配的方式，帮助我们构建表意丰富又易于维护的代码。在这一节中，我们就来看一些用在分支和循环控制语句中的基本样式匹配方式。

匹配值的方式

为了演示各种样式匹配的方式，我们先定义一个tuple，表示平面直角坐标系中的原点：

```
let origin = (x: 0, y: 0)
```

当我们要判断某个点是否是原点的时候，最原始的方式，是这样的：

```
let pt1 = (x: 0, y: 0)

if pt1.x == origin.x && pt1.y == origin.y {
    print("@Origin")
}
```

当然，这样判断xy坐标是否相等并不让人满意，写起来非常麻烦。实际上，我们还可以这样：

```
if case (0, 0) = pt1 {
    print("@Origin")
}
```

我们可以用 `case 匹配的值 = 要检查的对象` 的方式，对要检查的对象进行判断。在我们的例子里，判断的就是pt1是否等于原点。

除了用在 `if` 中匹配值，我们当然也可以在 `switch` 的 `case` 分支里，匹配特定形式的值：

```
switch pt1 {
case (0, 0):
    print("@origin")
case (_, 0):
    print("on x axis")
case (0, _):
    print("on y axis")
case (-1...1, -1...1):
    print("inside 2x2 square")
default:
    break
}
```

在上面这个例子里，除了用 `case (0, 0)` 表示匹配原点值之外，还可以用 `(_, 0)` 和 `(0, _)` 表示忽略掉 _ 的部分，仅对tuple中某一部分的值进行匹配，或者，在tuple的每一个成员位置，使用range operator匹配值的某个范围。

除了把 `case` 用于条件分支语句，我们还可以用于循环语句，用于进一步控制循环条件，例如：

```
let array1 = [1, 1, 2, 2, 2]

for case 2 in array1 {
    print("found two") // Three times
}
```

在上面这个例子里，当遇到数组中值为2的元素时，我们向控制台打印了一行话，因此， `print` 一共会打印3次。

把匹配的内容绑定到变量

除了在 case 中使用各种形式的具体值之外，我们还可以把匹配到的内容直接绑定到变量上，这样我们就可以在相应的处理代码中直接使用它们，例如：

```
switch pt1 {
case (let x, 0):
    print("(\(x), 0) is on x axis")
case (0, let y):
    print("(0, \(y)) is on y axis")
default:
    break
}
```

在上面这个例子里，我们把之前 _ 的部分换成了 let x 和 let y，这样，同样是匹配在坐标轴上的点，这次，我们就可以在对应的 case 中，直接访问匹配到的值了。我们管这样的形式，叫做value binding。

除了直接绑定变量自身的值之外，我们还可以用类似的形式绑定 enum 中的关联值。例如，我们先定义一个表示方向的 enum

```
enum Direction {
    case north, south, east, west(abbr: String)
}

let west = Direction.west(abbr: "W")
```

为了演示，我们给 .west 添加个了一个associated value，表示方向的缩写。然后，我们既可以像这样来判断 enum 值自身：

```
if case .west = west {
    print(west) // west("W")
}
```

此时， print 打印的就是 enum case 的值。我们也可以这样来直接绑定 west 的associated value：

```
if case .west(let direction) = west {
    print(direction) // W
}
```

此时， print 打印出来的值，就直接是字符“W”了。当然， case 这样的用法，在 switch 的分支中，也是完全可以的。

自动提取optional的值

除了绑定 enum 的associated value之外，我们还可以使用 case 来自动提取optional类型的非空值：

```
let skills: [String?] =
    ["Swift", nil, "PHP", "JavaScirpt", nil]

for case let skill? in skills {
    print(skill) // Swift PHP JavaScript
}
```

在我们的例子里， skills 包含了5个元素，其中两个是 nil，当我们用 case let skill? 这样的形式来绑定optional值的时候，Swift就会自动提取每一个非 nil 的元素，因此， print 会输出“Swift PHP JavaScript”。

自动绑定类型转换的结果

最后一类基本的样式匹配规则是自动绑定类型转换的结果。首先，我们创建一个 [Any]：

```
let someValues: [Any] = [1, 1.0, "One"]
```

当我们遍历 someValues，并且要根据不同类型的数组元素分别做一些操作的时候，可以这样：

```
for value in someValues {
    switch value {
    case let v as Int:
        print("Integer \(v)")
    case let v as Double:
        print("Double \(v)")
    case let v as String:
        print("String \(v)")
    default:
        print("Invalid value")
    }
}
// Integer 1
// Double 1.0
// String One
```

在上面的例子中，我们使用了 `case let Variable as Type` 的方式，把类型转换成功的结果，绑定在了变量 V 上。这样，我们就可以在对应的 `case` 里，访问到转换成功的值了。

或者，如果你仅仅想判断类型，而不需要知道具体内容的话，还可以使用更简单的 `is` 操作符：

```
for value in someValues {
    switch value {
    case is Int:
        print("Integer")
    // omit for simplicity...
}
```

这样，当数组元素类型为 `Int` 的时候，我们就会向控制台打印“Integer”。

What's next?

以上，就是Swift中，在分支判断和循环语句中，通过各种方式绑定值以简化代码的用法。简单来说，我们可以绑定值，绑定associated value，绑定optional的unwrapping结果，绑定类型转换后的对象。这看似很强大了，但故事还不仅于此，在下一节中，我们将进一步和大家分享更高级的样式匹配方法。

⏮ 几乎所有语言都有的条件判断和循环

(https://www.boxueio.com/series/make-a-decision/ebook/8)

使用高级样式匹配方式 ⏭

(https://www.boxueio.com/series/make-a-decision/ebook/136)

泊学相关

关于泊学 ›

加入泊学 ›

泊学用户隐私以及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)

联系泊学

Email: 10[AT]boxue.io (mailto:10@boxue.io)
QQ: 2085489246

友情链接 SwiftV (http://www.swiftv.cn) | Seay信息安全博客 (http://www.cnseay.com) | Swift.gg (http://swift.gg/) | Laravist (http://laravist.com/) | SegmentFault (https://segmentfault.com) | 靛青K的博客 (http://blog.dianqk.org/)