The Swift Programming Language in Chinese

https://github.com/letsswift

http://letsswift.com/

June 15, 2014

目录

1 Swift 中文教程（十八）类型检查

类型检查是一种检查类实例的方式，并且或者也是让实例作为它的父类或者子类的一种方式。

类型检查在 Swift 中使用 is 和 as 操作符实现。这两个操作符提供了一种简单达意的方式去检查值的类型或者转换它的类型。

你也可以用来检查一个类是否实现了某个协议，就像在 Protocols Checking for Protocol Conformance 部分讲述的一样。

1.1 定义一个类层次作为例子

你可以将它用在类和子类的层次结构上，检查特定类实例的类型并且转换这个类实例的类型成为这个层次结构中的其他类型。这下面的三个代码段定义了一个类层次和一个包含了几个这些类实例的数组，作为类型检查的例子。

第一个代码片段定义了一个新的基础类 MediaItem。这个类为任何出现在数字媒体库的媒体项提供基础功能。特别的，它声明了一个 String 类型的 name 属性，和一个 init name 初始化器。(它假定所有的媒体项都有个名称。)

```
class MediaItem {
    var name: String
    init(name: String) {
        self.name = name
    }
}
```

下一个代码段定义了 MediaItem 的两个子类。第一个子类 Movie，在父类 (或者说基类) 的基础上增加了一个 director（导演）属性，和相应的初始化器。第二个类在父类的基础上增加了一个 artist（艺术家）属性，和相应的初始化器:

```
class Song: MediaItem {
    var artist: String
    init(name: String, artist: String) {
        self.artist = artist
        super.init(name: name)
    }
}
```

最后一个代码段创建了一个数组常量 library ，包含两个 Movie 实例和三个 Song 实例。library 的类型是在它被初始化时根据它数组中所包含的内容推断来的。Swift 的类型检测器能够演绎出 Movie 和 Song 有共同的父类 MediaItem，所以它推断出 MediaItem[] 类作为 library 的类型。

```
let library = [
    Movie(name: "Casablanca", director: "Michael Curtiz"),
    Song(name: "Blue Suede Shoes", artist: "Elvis Presley"),
    Movie(name: "Citizen Kane", director: "Orson Welles"),
    Song(name: "The One And Only", artist: "Chesney Hawkes"),
    Song(name: "Never Gonna Give You Up", artist: "Rick Astley")
]
// the type of "library" is inferred to be MediaItem[]
```

在幕后 library 里存储的媒体项依然是 Movie 和 Song 类型的，但是，若你迭代它，取出的实例会是 MediaItem 类型的，而不是 Movie 和 Song 类型的。为了让它们作为它们本来的类型工作，你需要检查它们的类型或者向下转换它们的类型到其它类型，就像下面描述的一样。

1.2 检查类型

用类型检查操作符 (is) 来检查一个实例是否属于特定子类型。类型检查操作符返回 true 若实例属于那个子类型，若不属于返回 false 。

下面的例子定义了两个变量，movieCount 和 songCount，用来计算数组 library 中 Movie 和 Song 类型的实例数量。

```
var movieCount = 0
var songCount = 0

for item in library {
    if item is Movie {
        ++movieCount
    } else if item is Song {
        ++songCount
    }
}
```

```
println("Media library contains \(movieCount) movies and \(songCount) songs")
// prints "Media library contains 2 movies and 3 songs"
```

示例迭代了数组 library 中的所有项。每一次，for-in 循环设置 item 为数组中的下一个 MediaItem。

若当前 MediaItem 是一个 Movie 类型的实例，item is Movie 返回 true，相反返回 false。同样的，item is Song 检查 item 是否为 Song 类型的实例。在循环结束后，movieCount 和 songCount 的值就是被找到属于各自的类型的实例数量。

1.3 向下转型（Downcasting）

某类型的一个常量或变量可能在幕后实际上属于一个子类。你可以相信，上面就是这种情况。你可以尝试向下转到它的子类型，用类型检查操作符 (as)

因为向下转型可能会失败，类型检查操作符带有两种不同形式。可选形式 (optional form) as? 返回一个你试图下转成的类型的可选值 (optional value)。强制形式 as 把试图向下转型和强制解包（force-unwraps）结果作为一个混合动作。

当你不确定下转可以成功时，用类型检查的可选形式 (as?)。可选形式的类型检查总是返回一个可选值（optional value），并且若下转是不可能的，可选值将是 nil 。这使你能够检查下转是否成功。

只有你可以确定下转一定会成功时，才使用强制形式。当你试图下转为一个不正确的类型时，强制形式的类型检查会触发一个运行时错误。

下面的例子，迭代了 library 里的每一个 MediaItem ，并打印出适当的描述。要这样做，item 需要真正作为 Movie 或 Song 的类型来使用。不仅仅是作为 MediaItem。为了能够使用 Movie 或 Song 的 director 或 artist 属性，这是必要的。

在这个示例中，数组中的每一个 item 可能是 Movie 或 Song。事前你不知道每个 item 的真实类型，所以这里使用可选形式的类型检查（as?）去检查循环里的每次下转。

```
for item in library {
    if let movie = item as? Movie {
        println("Movie: '\(movie.name)', dir. \(movie.director)")
    } else if let song = item as? Song {
        println("Song: '\(song.name)', by \(song.artist)")
    }
```

```
}

// Movie: 'Casablanca', dir. Michael Curtiz
// Song: 'Blue Suede Shoes', by Elvis Presley
// Movie: 'Citizen Kane', dir. Orson Welles
// Song: 'The One And Only', by Chesney Hawkes
// Song: 'Never Gonna Give You Up', by Rick Astley
```

示例首先试图将 item 下转为 Movie。因为 item 是一个 MediaItem 类型的实例，它可能是一个 Movie；同样，它可能是一个 Song，或者仅仅是基类 MediaItem。因为不确定，as? 形式在试图下转时将返还一个可选值。item as Movie 的返回值是 Movie? 类型或 “optional Movie”。

当下转为 Movie 应用在两个 Song 实例时将会失败。为了处理这种情况，上面的例子使用了可选绑定（optional binding）来检查可选 Movie 真的包含一个值（这个是为了判断下转是否成功。）可选绑定是这样写的 “if let movie = item as? Movie”，可以这样解读：

“尝试将 item 转为 Movie 类型。若成功，设置一个新的临时常量 movie 来存储返回的可选 Movie”

若下转成功，然后 movie 的属性将用于打印一个 Movie 实例的描述，包括它的导演的名字 director。当 Song 被找到时，一个相近的原理被用来检测 Song 实例和打印它的描述。

> 注意：转换没有真的改变实例或它的值。潜在的根本的实例保持不变；只是简单地把它作为它被转换成的类来使用。

1.3.1 Any 和 AnyObject 的类型检查

Swift 为不确定类型提供了两种特殊类型别名：

- AnyObject 可以代表任何 class 类型的实例。
- Any 可以表示任何类型，除了方法类型（function types）。

> 注意：只有当你明确的需要它的行为和功能时才使用 Any 和 AnyObject。在你的代码里使用你期望的明确的类型总是更好的。

1.3.2 AnyObject 类型

当需要在工作中使用 Cocoa APIs，它一般接收一个 AnyObject[] 类型的数组，或者说 “一个任何对象类型的数组”。这是因为 Objective-C 没有明确的类型化数组。但是，你常常可以确定包含在仅从你知道的 API 信息提供的这样一个数组中的对象的类型。

在这些情况下，你可以使用强制形式的类型检查 (as) 来下转在数组中的每一项到比 AnyObject 更明确的类型，不需要可选解析（optional unwrapping)。

下面的示例定义了一个 AnyObject[] 类型的数组并填入三个 Movie 类型的实例:

```
let someObjects: AnyObject[] = [
   Movie(name: "2001: A Space Odyssey", director: "Stanley Kubrick"),
    Movie(name: "Moon", director: "Duncan Jones"),
    Movie(name: "Alien", director: "Ridley Scott")
]
```

因为知道这个数组只包含 Movie 实例，你可以直接用 (as) 下转并解包到不可选的 Movie 类型（ps: 其实就是我们常用的正常类型，这里是为了和可选类型相对比)。

```
for object in someObjects {
    let movie = object as Movie
    println("Movie: '\(movie.name)', dir. \(movie.director)")
}
// Movie: '2001: A Space Odyssey', dir. Stanley Kubrick
// Movie: 'Moon', dir. Duncan Jones
// Movie: 'Alien', dir. Ridley Scott
```

为了变为一个更短的形式，下转 someObjects 数组为 Movie[] 类型来代替下转每一项方式。

```
for movie in someObjects as Movie[] {
    println("Movie: '\(movie.name)', dir. \(movie.director)")
}
// Movie: '2001: A Space Odyssey', dir. Stanley Kubrick
// Movie: 'Moon', dir. Duncan Jones
// Movie: 'Alien', dir. Ridley Scott
```

1.4 Any 类型

这里有个示例，使用 Any 类型来和混合的不同类型一起工作，包括非 class 类型。它创建了一个可以存储 Any 类型的数组 things。

```
var things = Any[]()

things.append(0)
things.append(0.0)
things.append(42)
things.append(3.14159)
things.append("hello")
things.append((3.0, 5.0))
things.append(Movie(name: "Ghostbusters", director: "Ivan Reitman"))
```

things 数组包含两个 Int 值，2 个 Double 值，1 个 String 值，一个元组 (Double, Double) ，Ivan Reitman 导演的电影 “Ghostbusters”。

你可以在 switch cases 里用 is 和 as 操作符来发觉只知道是 Any 或 AnyObject 的常量或变量的类型。下面的示例迭代 things 数组中的每一项的并用 switch 语句查找每一项的类型。这几种 switch 语句的情形绑定它们匹配的值到一个规定类型的常量，让它们可以打印它们的值:

```
for thing in things {
    switch thing {
    case 0 as Int:
        println("zero as an Int")
    case 0 as Double:
        println("zero as a Double")
    case let someInt as Int:
        println("an integer value of \(someInt)")
    case let someDouble as Double where someDouble > 0:
        println("a positive double value of \(someDouble)")
    case is Double:
       println("some other double value that I don't want to print")
    case let someString as String:
        println("a string value of \"\(someString)\"")
    case let (x, y) as (Double, Double):
```

```
        println("an (x, y) point at \(x), \(y)")
    case let movie as Movie:
     println("a movie called '\(movie.name)', dir. \(movie.director)")
    default:
        println("something else")
    }
}

// zero as an Int
// zero as a Double
// an integer value of 42
// a positive double value of 3.14159
// a string value of "hello"
// an (x, y) point at 3.0, 5.0
// a movie called 'Ghostbusters', dir. Ivan Reitman
```

注意：在一个 switch 语句的 case 中使用强制形式的类型检查操作符 (as, 而不是 as?) 来检查和转换到一个明确的类型。在 switch case 语句的内容中这种检查总是安全的。

本文部分原文来自于 http://www.swiftguide.cn/翻译小组的译文，共同校对中。

参考文献