



类型转换

4.2 ▾

Swift 编程语言 / 类型转换

这是一篇社区协同翻译的文章，你可以点击右边区块信息里的『改进』按钮向译者提交改进建议。

类型转换 是一种检查实例类型的方法，同时也能够将该实例转为其类继承关系中其他的父类或子类。

Swift 中的类型转换是通过 `is` 和 `as` 运算符实现的。这两个运算符提供了一种简单而直观的方法来检查值的类型，或将值转换为其他类型。

你还可以使用类型转换来检查类型是否符合协议，如 [检查协议一致性](#) 中所述。

定义一个类结构作为类型转换示例

你可以使用类型转换连同类和子类的层次结构来检查特定类的实例类型，并将该实例强制转换为同一层次结构中的另一个类。下面的三个代码片段定义了一种类的层次结构，以及一个包含了这些类的实例的数组，用于类型转换的示例。

第一个片段定义了一个名为 `MediaItem` 的新的基类。此类为数字媒体库中显示的任何项目提供基本功能。具体来说，它声明了一个 `String` 类型的 `name` 属性，以及 `init(name)` 初始化方法。（假设所有媒体项目，包括所有电影和歌曲，都有一个名字。）

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

下一个片段定义了两个 `MediaItem` 的子类。第一个子类 `Movie` 封装了有关电影或影片的额外信息。它在基类 `MediaItem` 的顶部添加了一个 `director` 属性，以及相

Infinity

翻译进度

9

分块数量

3

参与人数



jihongboo 翻译于 5个月

0

重译

由 Aufree 审阅



jihongboo 翻译于 5个月

0

重译

由 Summer 审阅



应的初始化方法。 第二个子类 `Song` 在基类之上添加了 `artist` 属性和初始化方法：

```
class Movie: MediaItem {
    var director: String

    init(name: String, director: String) {
        self.director = director
        super.init(name: name)
    }
}

class Song: MediaItem {
    var artist: String

    init(name: String, artist: String) {
        self.artist = artist
        super.init(name: name)
    }
}
```

最后一段代码创建了一个名为 `library` 的常量数组，其中包含了两个 `Movie` 实例和三个 `Song` 实例。 `library` 数组的类型是在其初始化时，根据数组中所包含的内容来推断的。Swift 的类型检查器能够推断出 `Movie` 和 `Song` 有一个共同的父类 `MediaItem`，因此它推断出 `library` 数组的类型为 `[MediaItem]`：

```
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")
]
// 「library」的类型被推断为 [MediaItem]
```

虽然实际存储在 `library` 中的项目仍然是 `Movie` 和 `Song` 的实例。但是当你迭代此数组时，你将得到 `MediaItem` 类型的实例，而不是 `Movie` 或 `Song`。为了把它们作为原类型使用，你需要 检查 它们的类型，或者将它们 强转 为其他类型，如下所述。

类型检查

使用 类型检查运算符（`is`）来检查实例是否属于某个特定子类型。如果实例属于该子



jihongboo 翻译于 5个月

👍 0 重译

由 Summer 审阅

类型，则类型检查运算符将返回 `true` ，否则，将返回 `false` 。

下面的例子定义了两个变量，`movieCount` 和 `songCount` ，它们分别计算 `library` 数组中 `Movie` 和 `Song` 实例的数量：

```
var movieCount = 0
var songCount = 0

for item in library {
  if item is Movie {
    movieCount += 1
  } else if item is Song {
    songCount += 1
  }
}

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

此示例遍历了 `library` 数组中的所有项。在每次跳转时，`for` - `in` 循环将 `item` 常量设置为数组中的下一个 `MediaItem` 。

如果当前的 `MediaItem` 是一个 `Movie` 实例，那么 `item is Movie` 将返回 `true` ，如果不是，则返回 `false` 。类似地，`item is Song` 检查该项是否是 `Song` 的实例。在 `for` - `in` 循环结束时，`movieCount` 和 `songCount` 的值就是每种类型所包含的 `MediaItem` 实例的个数。

强制转型

实际上某个类型的常量或变量可能本来就是某个子类的实例。当确认是这种情况情况时，你可以尝试使用 类型强制转换运算符 （`as?` 或 `as!` ）将该常量或变量 强制转换 成子类型。

由于强制转换可能会失败，因该类型转换运算符有两种不同的形式。条件形式 `as?` 会返回你尝试强制转换的类型的可选值。强制形式 `as!` 则会尝试强制转换，并同时将结果强制解包。

当你不确定强制转换是否成功时，请使用类型转换运算符的条件形式（`as?`）。这种形式的运算符将始终返回一个可选值，如果无法进行强制转换，该值将为 `nil` 。这使得你可以检查强制转换是否成功。

仅当你确定强制转换会始终成功时，才使用类型转换运算符的强制形式（`as!`）。如果你尝试强制转换为不正确的类型，此形式的运算符将触发运行时错误。



jihongboo 翻译于 5个月

👍 0 重译

由 Summer 审阅



jihongboo 翻译于 5个月

👍 0 重译

由 Summer 审阅

下面的示例遍历了 `library` 中的每个 `MediaItem`，并为每个元素打印适当的描述。要做到这一点，它需要把每个元素当作真正的 `Movie` 或 `Song` 来访问，而不仅仅是作为 `MediaItem`。这是必要的，以便它能够访问 `Movie` 或 `Song` 的 `director` 或 `artist` 属性用于描述。

在这个例子中，数组中的每个元素可能是 `Movie`，也可能是 `Song`。你事先并不知道每个元素的真实类型，因此建议使用类型转换运算符的条件形式（`as?`）来检查每次循环时的强制转换：

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

该示例首先尝试将当前的 `item` 强制转换为 `Movie`。因为 `item` 是一个 `MediaItem` 的实例，它 *可能* 是一个 `Movie`，也可能是一个 `Song`，甚至只是一个基础 `MediaItem`。由于这种不确定性，类型转换运算符 `as?` 在尝试强制转换子类型时会返回 *可选* 值。`item as? Movie` 的结果是 `Movie?` 类型，即「可选 `Movie`」类型。

将数组中的 `Song` 实例强制转换为 `Movie` 时会失败。为了解决这个问题，上面的例子使用了可选绑定来检查可选的 `Movie` 是否包含一个值（也就是说，判断强转是否成功。）这个可选绑定是「`if let movie = item as? Movie`」，可以被理解为：

「尝试将 `item` 当作 `Movie` 来访问。如果成功，则将存储在可选 `Movie` 中的值赋给一个名为 `movie` 的新的临时常量。」

如果强制转换成功，那么 `movie` 的属性将被用于打印该 `Movie` 实例的描述，包括其 `director` 的名称。类似的原理也被用于检查 `Song` 实例，并在库中找到 `Song` 时打印适当的描述（包括 `artist` 名称）。

注意



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅

转换实际上不会变更实例或修改其值。原本的实例保持不变；我们仅仅把它看作是它类型的实例，对其进行简单地处理和访问。

对Any和AnyObject做类型转换

Swift 提供了两种特殊的类型来处理非特定类型：

- `Any` 可以表示任何类型的实例，包括函数类型。
- `AnyObject` 可以表示任何类类型的实例。

只有在明确需要 `Any` 或 `AnyObject` 所提供的行为和功能时才使用他们。最好在你的代码中明确需要使用的类型。

下面是使用 `Any` 来处理不同类型混合的示例，包括函数类型和非类类型。该示例创建了一个名为 `things` 的数组，它可以存储类型为 `Any` 的值：

```
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.append({ (name: String) -> String in "Hello, \(name)" })
```

`things` 数组包含了两个 `Int` 值，两个 `Double` 值，一个 `String` 值，一个类型为 `(Double, Double)` 的元组，一个名为 `Ghostbusters` 的 movie，以及一个包含 `String` 参数并返回另一个 `String` 值的闭包表达式。

如果要查看一个常量或变量的具体类型，但只知道它的类型是 `Any` 或 `AnyObject`，可以在 `switch` 分句中使用 `is` 或 `as` 模式。下面的示例遍历了 `things` 数组中的元素，并使用 `switch` 语句查询每个元素的类型。这几个 `switch` 分句将每个元素所匹配到的值绑定到指定类型的常量中，以使其值可以打印：

```
for thing in things {
    switch thing {
    case 0 as Int:
        print("zero as an Int")
    case 0 as Double:
```



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅


```

    case 0 as Double:
        print("zero as a Double")
    case let someInt as Int:
        print("an integer value of \(someInt)")
    case let someDouble as Double where someDouble > 0:
        print("a positive double value of \(someDouble)")
    case is Double:
        print("some other double value that I don't want to print")
    case let someString as String:
        print("a string value of \"\(someString)\"")
    case let (x, y) as (Double, Double):
        print("an (x, y) point at \(x), \(y)")
    case let movie as Movie:
        print("a movie called \(movie.name), dir. \(movie.director)")
    case let stringConverter as (String) -> String:
        print(stringConverter("Michael"))
    default:
        print("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
// Hello, Michael

```

注意

`Any` 类型表示任何类型的值，包括可选类型。如果程序需要一个类型为 `Any` 的值，而你却使用了可选类型，Swift 会向你发出警告。如果你确实需要将可选值作为 `Any` 使用，可以使用 `as` 操作符将可选类型显式地转换为 `Any` 类型，如下所示。

```

let optionalNumber: Int? = 3
things.append(optionalNumber)           // 警告
things.append(optionalNumber as Any)    // 没有警告

```

本文中的所有译文仅用于学习和交流目的，转载请务必注明文章译者、出处、和本文链接

我们的翻译工作遵照 [CC 协议](#)，如果我们的工作有侵犯到您的权益，请及时联系我们。

← 上一篇

下一篇 →

👍 点赞



👤 参与译者：3



更多职位





💬 讨论数量：0

✎ 发起讨论

☐ 只看当前版本讨论

暂无话题~


兄弟社区

-  Laravel China
-  PythonCaff.com
-  GolangCaff.com
-  VuejsCaff.com

资源推荐

资源推荐

其他信息

-  软件外包
-  商务合作
-  联系站长