标

类，结构和枚举可以定义 下标，它们是访问集合，列表或序列的成员元素的快捷方式。您可以使用下标以索引设置和检索值，而无需单独的设置和检索方法。例如，您可以访问在元素`Array`实例作为`someArray[index]`和元素的`Dictionary`实例作为`someDictionary[key]`。

您可以为单个类型定义多个下标，并根据传递给下标的索引值的类型选择适当的下标超载。下标不限于单个维度，您可以使用多个输入参数定义下标以适应您的自定义类型的需求。

下标语法

下标使您能够通过在实例名称后面的方括号中写入一个或多个值来查询某个类型的实例。它们的语法类似于实例方法语法和计算属性语法。您可以使用`subscript`关键字编写下标定义，并以与实例方法相同的方式指定一个或多个输入参数和返回类型。与实例方法不同，下标可以是可读写或只读的。这种行为由getter和setter以与计算属性相同的方式传递：

```
subscript(index: Int) -> Int {
    get {
        // return an appropriate subscript value here
    }
    set(newValue) {
        // perform a suitable setting action here
    }
}
```

类型`newValue`与下标的返回值相同。与计算属性一样，您可以选择不指定设置者的`(newValue)`参数。`newValue`如果您自己没有提供默认参数，则会向您的setter提供默认参数。

与只读计算属性一样，您可以通过删除`get`关键字及其大括号来简化只读脚本的声明：

```
subscript(index: Int) -> Int {
    // return an appropriate subscript value here
}
```

下面是一个只读脚本实现的例子，它定义了一个`TimesTable`表示一个*n*次表格整数的结构：

```
struct TimesTable {
    let multiplier: Int
    subscript(index: Int) -> Int {
        return multiplier * index
    }
}
let threeTimesTable = TimesTable(multiplier: 3)
print("six times three is \(threeTimesTable[6])")
// Prints "six times three is 18"
```

在这个例子中，`TimesTable`创建一个新的实例来表示三次表。这通过传递`3`结构的`initializer`值作为用于实例`multiplier`参数的值来指示。

您可以`threeTimesTable`通过调用其下标来查询实例，如调用中所示`threeTimesTable[6]`。这就要求在三次表，它返回的值的第六项`18`，或`3`时间`6`。

> 注意
>
> 一个*n*次表是基于一个固定的数学规则。设置`threeTimesTable[someIndex]`为新值并不合适，因此下标for `TimesTable`被定义为只读下标。

下标使用

“下标”的确切含义取决于其使用的上下文。下标通常用作访问集合，列表或序列中成员元素的快捷方式。您可以自由地以最合适

例如，Swift的`Dictionary`类型实现了一个下标来设置和检索存储在`Dictionary`实例中的值。您可以在字典中设置一个值，方法是在下标括号内提供字典键类型的键，并将字典值类型的值赋给下标：

```
var numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
numberOfLegs["bird"] = 2
```

上面的例子定义了一个被调用的变量`numberOfLegs`，并用一个包含三个键值对的字典文字初始化它。`numberOfLegs`字典的类型被推断为`[String: Int]`。创建字典后，本示例使用下标分配将字典的`String`键`"bird"`和`Int`值添加`2`到字典中。

有关`Dictionary`下标的更多信息，请参阅访问和修改字典。

> 注意
>
> Swift的`Dictionary`类型实现了其键值下标作为一个下标，它接受并返回一个*可选*类型。对于`numberOfLegs`上面的字典，键值下标采用并返回一个类型值`Int?`，或“可选的int”。该`Dictionary`类型使用可选的下标类型来模拟不是每个键都有值的事实，并且通过为键分配`nil`值来给出一种方法来删除键的值。

下标选项

下标可以接受任意数量的输入参数，并且这些输入参数可以是任何类型。下标还可以返回任何类型。下标可以使用可变参数，但不能使用输入参数或提供默认参数值。

类或结构可以提供尽可能多的下标实现，并根据使用下标时下标括号内包含的值或值的类型推断使用的下标。多重下标的定义称为 *下标重载*。

虽然下标最常见的是采用单个参数，但如果它适合您的类型，也可以使用多个参数定义下标。以下示例定义了一个`Matrix`表示`Double`值的二维矩阵的结构。该`Matrix`结构的标有两个整型参数：

```
struct Matrix {
    let rows: Int, columns: Int
    var grid: [Double]
    init(rows: Int, columns: Int) {
        self.rows = rows
        self.columns = columns
        grid = Array(repeating: 0.0, count: rows * columns)
    }
    func indexIsValid(row: Int, column: Int) -> Bool {
        return row >= 0 && row < rows && column >= 0 && column < columns
    }
    subscript(row: Int, column: Int) -> Double {
        get {
            assert(indexIsValid(row: row, column: column), "Index out of range")
            return grid[(row * columns) + column]
        }
        set {
            assert(indexIsValid(row: row, column: column), "Index out of range")
            grid[(row * columns) + column] = newValue
        }
    }
}
```

`Matrix`提供了一个初始化函数，它接受两个名为`rows`and的参数`columns`，并创建一个足够大的数组来存储`rows * columns`类型的值`Double`。矩阵中的每个位置都有一个初始值`0.0`。为了达到这个目的，数组的大小和初始单元格值`0.0`被传递给一个数组初始值设定器，该初始值设定器创建并初始化一个正确大小的新数组。该初始值设定项在使用默认值创建数组中有更详细的描述。

您可以`Matrix`通过将适当的行和列计数传递给它的初始值设定项构造一个新的实例：

```
var matrix = Matrix(rows: 2, columns: 2)
```

上面的例子创建了一个Matrix两行两列的新实例。grid这个Matrix实例的数组实际上是矩阵的扁平版本，从左上角到右下角读：

```
grid = [0.0, 0.0, 0.0, 0.0]
```

```
          column
            0      1
row  0  [ [0.0, 0.0,
     1    0.0, 0.0 ] ]
```

矩阵中的值可以通过将行和列值传递给下标来设置，并用逗号分隔：

```
matrix[0, 1] = 1.5
matrix[1, 0] = 3.2
```

这两个语句调用下标的设置器1.5在矩阵的右上角位置（其中rowis 0和columnis 1）以及3.2左下角位置（rowis 1和columnis 0）中设置值：

```
[ 0.0  1.5 ]
[ 3.2  0.0 ]
```

在本页

该Matrix标的getter和setter都包含一个断言，以检查标的row和column值是有效的。为了协助这些断言，Matrix包括一个称为的便捷方法indexIsValid(row:column:)，它检查所请求的row并且column在矩阵的范围内：

```
func indexIsValid(row: Int, column: Int) -> Bool {
    return row >= 0 && row < rows && column >= 0 && column < columns
}
```

如果尝试访问矩阵范围之外的下标，则触发断言：

```
let someValue = matrix[2, 2]
// this triggers an assert, because [2, 2] is outside of the matrix bounds
```