插入排序（Insertion sort）

Back to series (/series/algorithms-in-swift3)

字号
字号
默认主题
金色主题
暗色主题

插入排序是最基础的排序算法之一。它最核心的思想，由以下几条构成。当我们要对一个值为 `[1, 5, 6]` 的数组从大到小排列时：

1. 把序列的第一个元素想象成一个“子序列” `[1]`，它是已经排序的；
2. 按照既定的排序规则，把由序列的前两个元素构成的“子序列”排序： `[5, 1]`；
3. 之后，读入6，在之前已经排序好的“子序列”中，**从右向左逐个**和新读入的元素进行比对。如果满足排序规则，就交换已排序数组中的元素和待排序的元素：

```
[5, 1, 6]
    ^ 6 > 1 == true
[5, 1, 6]
    <--->
     swap
[5, 6, 1]
```

简单来说，就是不断通过比对，移动待排序元素的位置。直到待排序元素和之前已排序“子序列”全部元素都比对完之后：

```
[5, 6, 1]
 ^ 6 > 5 == true
[5, 6, 1]
 <--->
  swap
[6, 5, 1]
```

新形成的序列就已经是排序好的了。（当然，这里也有一个潜台词，就是**如果和子序列中第一个元素比对之后不需要移动，则新添加进来的元素就应该直接添加到子序列末尾**）；

-

反复3的操作，当读完所有待排序的元素之后，整个序列就排序完成了；

> 在理解插入排序的时候，要时刻记住一件事情：**元素的操作永远只发生在相邻的两个元素之间**。当我们在头脑中执行插入排序时，偶尔会忘记这条，会想着是否存在着跨元素交换的情况，然后就把自己搞晕了。

实现

如何使用?

在实现之前，我们要先考虑下开发者会如何使用这个算法，例如这样：

```
let a: Array<Int> = [1, 5, 6]
insertionSortOf(a)
```

或者，我们允许用户指定一个排序方法

```
let a: Array<Int> = [1, 5, 6]
insertionSortOf(a, byCriteria: >) // [6, 5, 1]
```

然后，我们还应该允许对包含任何“可比较”元素的Array进行排序。于是， `insertionSort` 的声明可以是下面这样的。

如何按Swift 3的方式声明

```
typealias CRITERIA<T> = (T, T) -> Bool

func insertionSortOf<T: Comparable>(
        _ coll: Array<T>,
        byCriteria: CRITERIA<T> = { $0 < $1 }) -> Array<T>
```

在这个声明里，有以下和Swift 3相关的说明：

首先，我们使用了SE-0048 (https://github.com/apple/swift-evolution/blob/master/proposals/0048-generic-typealias.md)中的新特性，**允许在 `typealias` 中使用泛型参数**；

其次，在方法的命名上，我们参考了SE-0023 API设计指南 (https://github.com/Boxue/swift-api-design-guidelines/blob/master/SE-0023%20swift-api-guidelines.md)中的要求：

“如果方法中第一个参数和方法名一起形成了一个语法正确的短语，去掉第一个参数的label，并且把参数label放到方法名中”

因此，我们把“表示要排序的集合”使用的介词“of”，从第一个参数名，放到了函数名中。

第三，在Swift 3里，根据SE-0046 (https://github.com/apple/swift-evolution/blob/master/proposals/0046-first-label.md)中的提议，**函数的第一个参数不再默认省略label**，它将和其他参数一样拥有默认的label行为。因此，如果我们要省略label，必须在参数名前强制使用 _ 。因此，在声明里，我们需要强制省略第一个参数的label。

第四，根据SE-0023 API设计指南 (https://github.com/Boxue/swift-api-design-guidelines/blob/master/SE-0023%20swift-api-guidelines.md)中的要求：

- **要让方法调用时，形成语法正确的英文短语**：因此，我们让第二个表示自定义比较规则的参数名为 `byCriteria`；
- **要为方法中的closure参数设置label**：因此，我们没有去掉第二个closure参数的label；
- **当方法的参数在绝大多数时候使用相同值时，应为它指定默认值**：因此，我们让 `byCriteria` 的默认行为是按升序排列；

实现insertionSort

按照一开始我们在算法思路中的描述，在 `insertionSort` 中添加下面的代码：

首先，只有一个元素的数组是无需排序的，我们直接返回就好：

```
func insertionSortOf<T: Comparable>(
        _ coll: Array<T>,
        byCriteria: CRITERIA<T> = { $0 < $1 }) -> Array<T> {

    // 1. An array with a single element is ordered
    guard coll.count > 1 else {
        return coll
    }
}
```

其次，复制一份参数数组，用于在函数内部进行排序：

```
func insertionSortOf<T: Comparable>(
        _ coll: Array<T>,
        byCriteria: CRITERIA<T> = { $0 < $1 }) -> Array<T> {

    //: #### 1. An array with a single element is ordered
    guard coll.count > 1 else {
        return coll
    }

    var result = coll
}
```

第三，我们从数组中第二个元素开始，通过逐个比对，来不断形成已排序好的子数组：

```
for x in 1 ..< coll.count {
    var y = x
    let key = result[y]

    print("Get: \(key)")

    // 2. If the key needs to swap in the previous ordered sub array
    while y > 0 && byCriteria(key, result[y - 1]) {
         print("----------------------------")
         print("Remove: \(result[y]) at pos: \(y)")
         print("Insert: \(key) at pos: \(y - 1)")
         print("----------------------------")

         // 3. Swap the value
         // The new Swift 3 API:
         // remove(at:) replaces removeAtIndex
         // You can also use swap(:) instead of remove and insert.
         result.remove(at: y)
         result.insert(key, at: y - 1)

         y -= 1
     }
}
```

最后，数组中所有的元素都遍历之后，整个数组就完成排序了，我们直接把排序后的数组返回：

```
func insertionSortOf<T: Comparable>(
        _ coll: Array<T>,
        byCriteria: CRITERIA<T> = { $0 < $1 }) -> Array<T> {

    guard coll.count > 1 else {
        return coll
    }

    var result = coll

    for x in 1 ..< coll.count {
        var y = x
        let key = result[y]

        print("Get: \(key)")

        // 2. If the key needs to swap in the previous ordered sub array
        while y > 0 && byCriteria(key, result[y - 1]) {
            print("----------------------------")
            print("Remove: \(result[y]) at pos: \(y)")
            print("Insert: \(key) at pos: \(y - 1)")
            print("----------------------------")

            // 3. Swap the value
            // Notice the new Swift 3 API: remove(at:) replaces removeAtIn
dex
            // You can also use swap(:) instead of remove and insert
            result.remove(at: y)
            result.insert(key, at: y - 1)

            y -= 1
        }
    }

    // 4. Return the sorted array
    return result
}
```

测试

用一开始我们设计的使用方法来测试 `insertionSort`：

```
let a: Array<Int> = [1, 5, 6]
insertionSortOf(a)
```

由于默认就是从小到大排序，并且，原始数组本身就是已经排序的，因此，我们可以在控制台看到下面的结果：

```
let a: Array<Int> = [1, 5, 6]                    [1, 5, 6]
insertionSortOf(a)                               [1, 5, 6]

Get: 5
Get: 6
```

如果我们传递一个自定义的比较规则，例如从大到小排序：

```
let a: Array<Int> = [1, 5, 6]
insertionSortOf(a, byCriteria: >)
```

就可以在控制台看到这样的结果：

```
let a: Array<Int> = [1, 5, 6]                    [1, 5, 6]
insertionSortOf(a, byCriteria: >)                [6, 5, 1]

Get: 5
----------------------------
Remove: 5 at pos: 1
Insert: 5 at pos: 0
----------------------------
Get: 6
----------------------------
Remove: 6 at pos: 2
Insert: 6 at pos: 1
----------------------------
----------------------------
Remove: 6 at pos: 1
Insert: 6 at pos: 0
----------------------------
```

数字5经历了一次交换，数字6经历了两次交换。

Have a try?

不用交换元素的插入排序方法

除了使用 `remove` & `insert` 或 `swap` 之外，还有一种插入排序的手段。用之前的 `[1, 5, 6]` 降序排列举例。假设算法执行到了读入数字6:

1. 记录读入的值：

   ```
   [5, 1, 6]
          ^ --> remember 6
   ```

2. 在新读入位置前已排序好的子数组里，不断用前一个数字覆盖后一个位置，为新读入的元素找到合适的位置：

   ```
   [5, 1, 1]
        --> shift 1 right
   [5, 5, 1]
    --> shift 5 right
   [6, 5, 1]
    ^ --> Copy 6 here
   ```

不同的实现方法之间的性能差异有多大呢?

- `insert` & `remove`；
- `swap`；
- 以及我们最后提到的移动元素；

当移动大量元素时，这些算法之间的差异有多大呢？自己试验一下吧，欢迎大家把实验的结果贴到泊学视频下面的Disqus论坛 ()里。 :-)

返回视频 选择排序（Selection sort）

职场漂泊的你，每天多学一点。

从开发、测试到运维，让技术不再成为你成长的绊脚石。我们用打磨产品的精神去传播知识，把最新的移动开发技术，通过简单的图表， 清晰的视频，简明的文字和切实可行的例子一一向你呈现。让学习不仅是一种需求，也是一种享受。

泊学动态

一个工作十年PM终创业的故事（二） (https://www.boxueio.com/after-the-full-upgrade-to-swift3)
Mar 4, 2017

人生中第一次创业的"10有" (https://www.boxueio.com/founder-chat)
Jan 9, 2016

猎云网采访报道泊学 (http://www.lieyunwang.com/archives/144329)
Dec 31, 2015

What most schools do not teach (https://www.boxueio.com/what-most-schools-do-not-teach)
Dec 21, 2015

一个工作十年PM终创业的故事（一） (https://www.boxueio.com/founder-story)
May 8, 2015

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/)