【前端面试】将数组旋转K步，寻求最优解！

前言

将数组旋转K步这算是一道非常经典的面试题了，题目不算太难，我相信绝大多数小伙伴都有实现思路。针对算法题，我们不仅仅实现它就好了，我们更重要的是学会思路，更要有时间复杂度和空间复杂度的概念，今天这篇文章再讲解题目的同时，还希望大家能够有自己的见解。

1.实现目标

这道题目比较简单，我们先来看看题目需求。

题目描述：

假如有一个数组[1,2,3,4,5,6,7]，我们需要将它旋转K步，K是一个数字。

输入输出分析：

案例一

> 输入：[1,2,3,4,5,6,7] K=3
>
> 输出：[5,6,7,1,2,3,4]

案例二

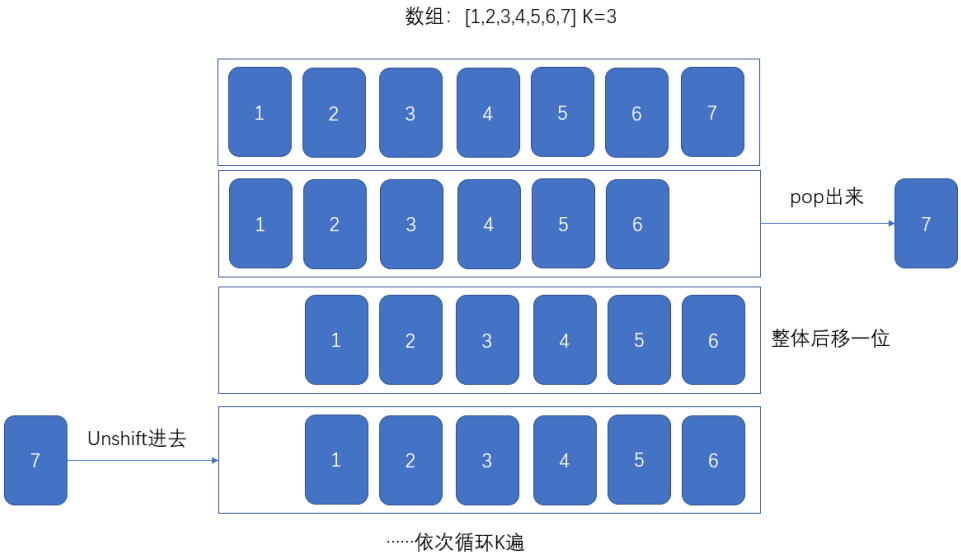
> 输入：[1,2,3,4,5,6,7] K=4
>
> 输出：[4,5,6,7,1,2,3]

总体来说，题目不难，我们需要实现一个方法，这个方法返回一个新的数组，方法接收一个原始数组和K两个参数。

接下来我们就来实现一下。

2.利用pop和unshift

利用pop和unshift是大多数小伙伴都能想到的方法，pop是取出数组最后一个元素，unshift是在数组最前面插入一个元素，我们画一张图，就能够更好理解这种实现思路了，如下图：



上了上面的图大家应该一下就能理解了，原理非常的简单，接下来编写代码即可。

代码如下：

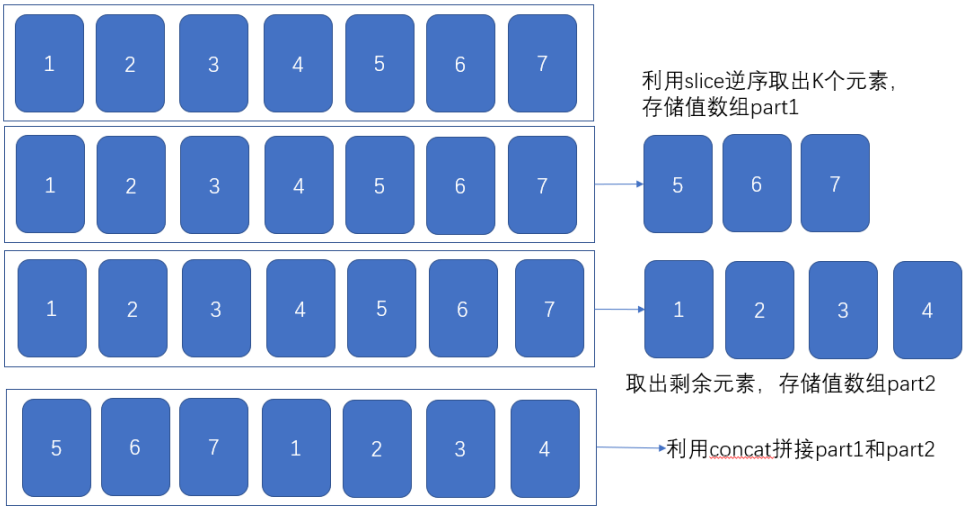
```
let array1 = [1, 2, 3, 4, 5, 6, 7];
function revolve1(arr, k) {
  const length = arr.length;
  if (!length || !k) {
    return arr;
  }
  // 取k的绝对值
  const step = Math.abs(k);

  for (let index = 0; index < step; index++) {
    const item = arr.pop(); // 取出最后一个
    if (item !== null) {
      arr.unshift(item); //插到最前面
    }
  }
  return arr;
}
// 测试
console.log(revolve1(array1, 3)); // [5, 6, 7, 1, 2, 3, 4]
```

上段代码其实非常的简单，就一个for循环，然后执行pop和unshift操作。

3.利用concat

使用concat也比较简单，我们也画张图一起来看看，如下图：

数组：[1,2,3,4,5,6,7] K=3



上面的步骤就比较简单了，基本上散步就可以完成了，首先就是将原数组拆分为了两端：part1和part2，然后再拼接两端数组即可，但是大家需要注意，上图中很明显原数组没有改变。

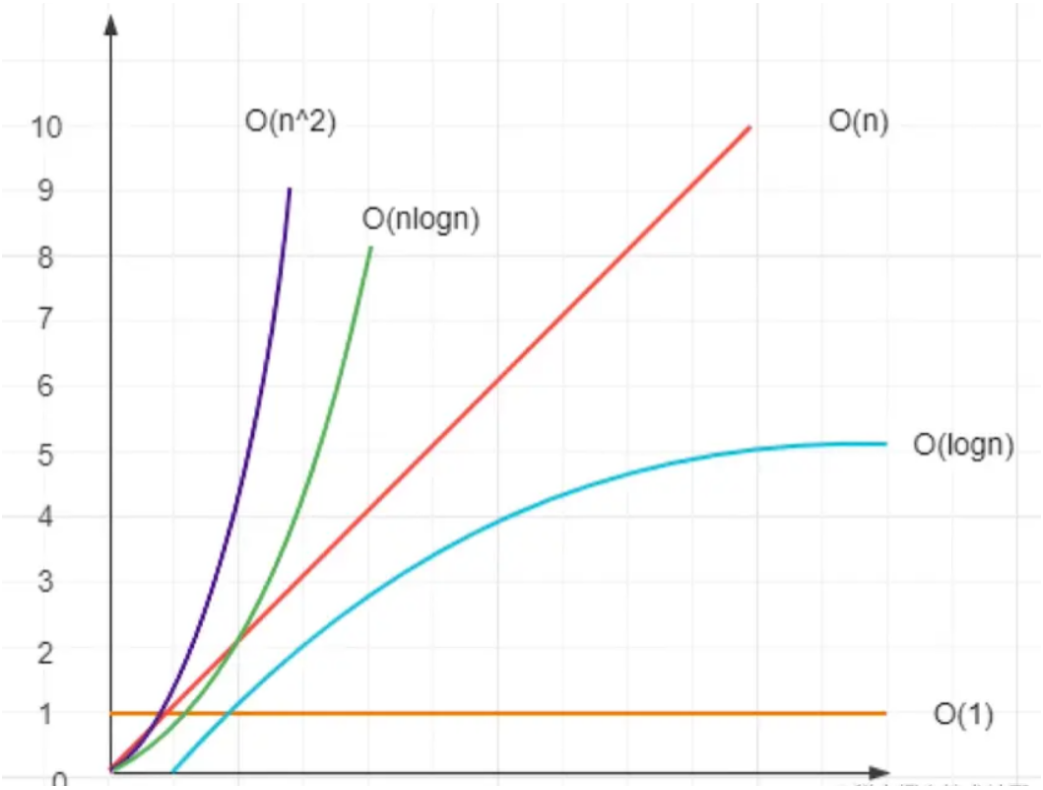
代码如下：

```
let array2 = [1, 2, 3, 4, 5, 6, 7];
function revolve2(arr, k) {
  const length = arr.length;
  if (!length || !k) {
    return arr;
  }
  // 取k的绝对值
  const step = Math.abs(k);

  const part1 = arr.slice(-step); // 从末尾取出几个元素，并组成新数组
  const part2 = arr.slice(0, length-step); // 取出剩余元素组成新数组
  const part3 = part1.concat(part2);
  return part3;
}
// 测试
console.log(revolve2(array2, 4)); // [4, 5, 6, 7, 1, 2, 3]
```

4.复杂度分析

做算法我们最重要的不是实现功能，而是要注意效率，所以我们很有必要分析代码的时间复杂度和空间复杂度，所以我们分析一下上面两端代码的时间复杂度和空间复杂度，给大家提供思路，以后在做算法题也不会抓瞎了。

我们先看一张关于算法复杂度的坐标图，方便后续理解，如下图：



从上图可以简单看出，O(1)是算法复杂度最低的，O(n^2)算法复杂度是较高的。

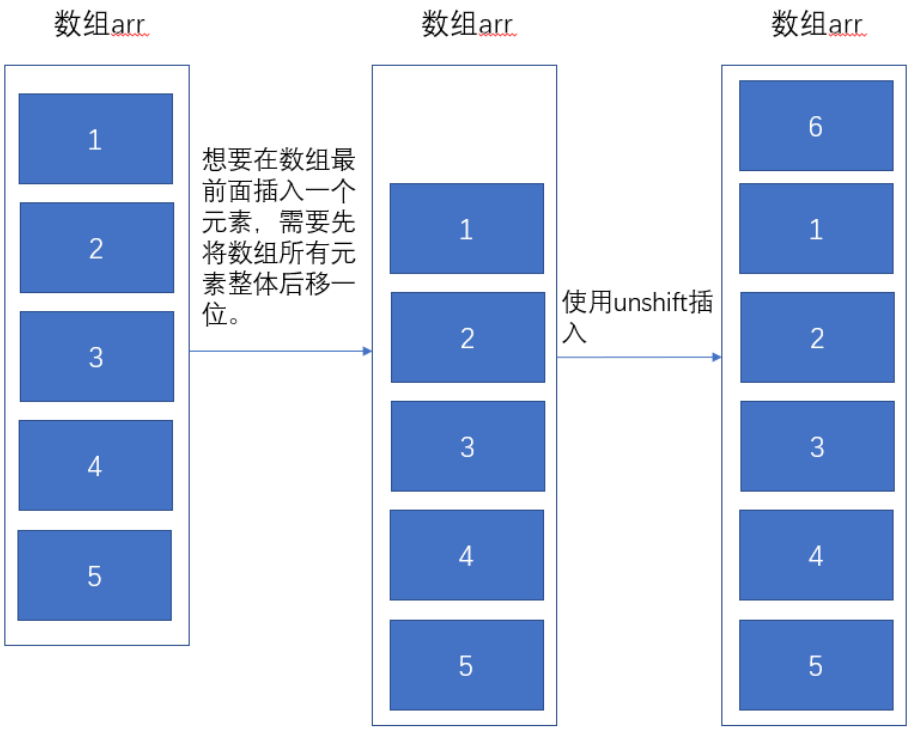
4.1 pop和unshift

时间复杂度：O(n^2)

空间复杂度：O(1)

解释：

为什么时间复杂度是O(n^2)呢？有些小伙伴可以以为代码中只有一个for循环，所以时间复杂度应该是O(n)，但是大家忽略了一个API，那就是unshift。这个API的时间复杂度也是O(n)，给大家看一张图大家就理解了。



上图就展示了使用unshift往数组最前面插入一个元素的整个过程，很明显，unshift改变了元素组，而且插入一个元素，所有的数组元素都得往后挪一个，因为数组是一个有序结构，所以unshift的时间复杂度是O(n)，结合for循环时间复杂度自然而然变为了O(n^2)。

那为什么空间复杂度为O(1)呢？因为我们整段代码中没有新增加数组，所有操作都是在原数组上完成的，没有造成额外的空间存储，所以空间复杂度为O(1)。

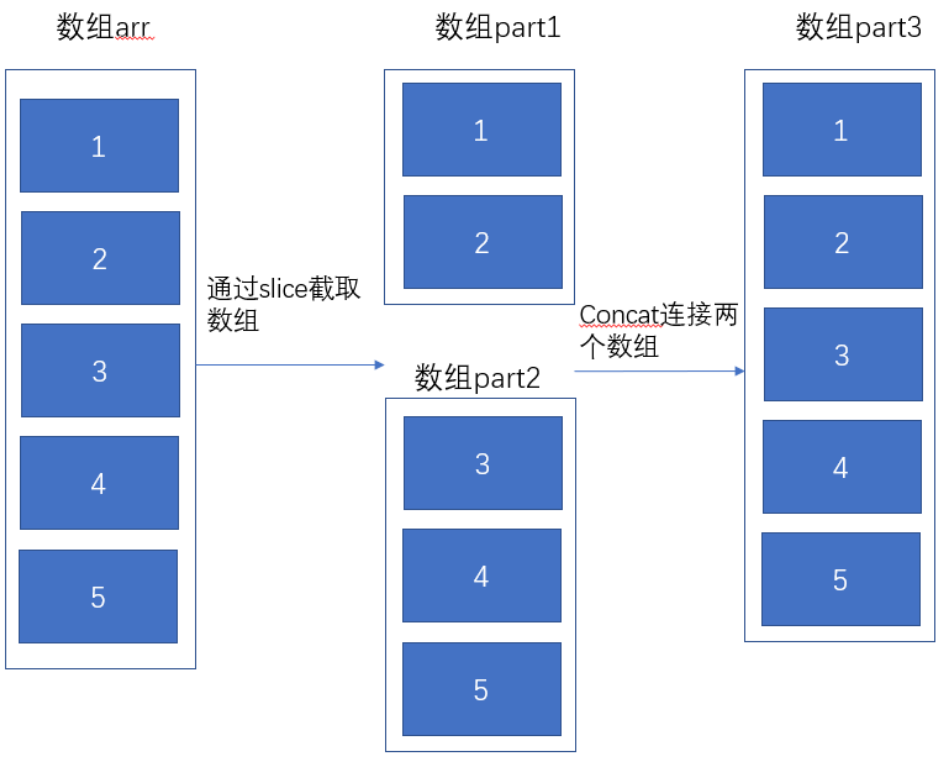
4.2 slice和concat

时间复杂度：O(1)

空间复杂度：O(n)

解释：

采用slice和concat的时间复杂度要低的多，为O(1)，我们先来看一张图。



从上图可以看到我们原数组arr自始至终都是没有变化的，而是新生成了两个新数组part1和part2，代码中也没有循环，所以时间复杂度为O(1)。

然而也因为新生成了两个新数组，使用的新的存储空间，所以空间复杂度是O(n)。

但是我们针对于前端而言，我们是**重时间轻空间**的，所以这种方法优于上一种。

5.性能对比

耳听为虚，眼见为实，我们真正做一个实验，来看看我们所谓的算法复杂度有没有体现出来。

代码如下：

```
const arr1 = []
for (let index = 0; index < 10 * 10000; index++) {
  arr1.push(index)
}
console.time('revolve1');
revolve1(arr1, 9  * 10000)
console.timeEnd('revolve1')

const arr2 = []
for (let index = 0; index < 10 * 10000; index++) {
  arr2.push(index)
}
console.time('revolve2');
revolve2(arr2, 9 * 10000)
console.timeEnd('revolve2')
```

上段代码中我们定义了两个非常大的数组，然后分别利用两个方法来实现K步旋转，最后看看它们的计算时间是多少。

输出结果：

```
revolve1: 1637.192138671875 ms
revolve2: 0.739990234375 ms
```

从上图可以看出，两个时间对比相差不是一点半点，相差了成百上千倍，当然这也和电脑性能相关。

不过在重时间轻空间的前提下，无疑是使用concat的方法是最优秀的！

总结

虽然这道面试题比较简单，但是可以带给我们很多启发，比如关于重时间轻空间的概念，原生JS API的一些特点等等。当然，还有小伙伴有其它办法，甚至有些小伙伴直接采用更改坐标的形式在做，不是说不可以，但是没必要！