华为OD机试 - 符合要求的元组的个数（Java & JS & Python）

题目描述

给定一个整数数组 nums、一个数字k，一个整数目标值 target，请问nums中是否存在k个元素使得其相加结果为target，请输出所有符合条件且不重复的k元组的个数

数据范围

- 2 ≤ nums.length ≤ 200
- -10^9 ≤ nums[i] ≤ 10^9
- -10^9 ≤ target ≤ 10^9
- 2 ≤ k ≤ 100

输入描述

第一行是nums取值：2 7 11 15

第二行是k的取值：2

第三行是target取值：9

输出描述

输出第一行是符合要求的元组个数：1

补充说明：[2,7]满足，输出个数是1

用例

输入	-1 0 1 2 -1 -4 3 0
输出	2
说明	[-1,0,1]，[-1,-1,2]满足条件

输入	2 7 11 15 2 9
输出	1
说明	[2,7]符合条件

题目解析

本题其实就是要求K数之和。

本题的K数之和，和LintCode - 89 K数之和_伏城之外的博客-CSDN博客是存在区别的，

本题的要求的K元组是从整数数组中选取的，这里的整数数组，既可能包含正数，也可能包含负数，也可能包含0，另外最终求得的符合要求的K元组，还要进行去重。

因此，本题无法参考：LintCode - 89 K数之和_伏城之外的博客-CSDN博客

本题其实需要参考：

LeetCode - 15 三数之和_伏城之外的博客-CSDN博客

LeetCode - 18 四数之和_伏城之外的博客-CSDN博客

这两题。如果你对这两题不熟悉，请做本题前，先做完前面这两题，这两题是基础。否则下面代码会看不懂。

其中三数之和，是需要固定三元组中的最小的一个值，然后通过双指针找到剩余两个数。

其中四数之和，是需要固定四元组中的最小的两个值，然后通过双指针找到剩余两个数。

而K数之和，其实需要固定K元组中最小的K-2个值，然后通过双指针找到剩余两个数。

因此，下面代码实现中分为了两部分：

1. K-2重for循环完成 K元组中最小的K-2个值的确定
2. 通过双指针完成剩余两个值的确定

而实际上K的值是不确定的，因此第1部分的K-2重for循环需要通过递归完成。

具体请看下面代码实现。

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int[] nums = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
    int k = Integer.parseInt(sc.nextLine());
    int target = Integer.parseInt(sc.nextLine());

    System.out.println(getResult(nums, k, target));
  }

  public static int getResult(int[] nums, int k, int target) {
```

```
17      if (k > nums.length) return 0;
18      Arrays.sort(nums);
19      return kSum(nums, k, target, 0, 0, 0);
20    }
21
22    // k数之和
23    public static int kSum(int[] nums, int k, int target, int start, int count, long sum)
    {
24      if (k < 2) return count;
25
26      if (k == 2) {
27        return twoSum(nums, target, start, count, sum);
28      }
29
30      for (int i = start; i <= nums.length - k; i++) {
31        // 剪枝
32        if (nums[i] > 0 && sum + nums[i] > target) break;
33
34        // 去重
35        if (i > start && nums[i] == nums[i - 1]) continue;
36        count = kSum(nums, k - 1, target, i + 1, count, sum + nums[i]);
37      }
38
39      return count;
40    }
41
42    // 两数之和
43    public static int twoSum(int[] nums, int target, int start, int count, long preSum)
    {
44      int l = start;
45      int r = nums.length - 1;
46
47      while (l < r) {
48        long sum = preSum + nums[l] + nums[r];
49
50        if (target < sum) {
51          r--;
52        } else if (target > sum) {
53          l++;
54        } else {
```

```
55          count++;
56          // 去重
57          while (l + 1 < r && nums[l] == nums[l + 1]) l++;
58          // 去重
59          while (r - 1 > l && nums[r] == nums[r - 1]) r--;
60          l++;
61          r--;
62        }
63      }
64
65      return count;
66    }
67  }
```