题目描述：

给定一个整数数组 `nums` 和一个目标值 `target`，请你在该数组中找出和为目标值的那 **两个** 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

示例:

> 给定 nums = [2, 7, 11, 15], target = 9
>
> 因为 nums[0] + nums[1] = 2 + 7 = 9 所以返回 [0, 1]

方法1，暴力解法

```java
public int[] twoSum(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[j] == target - nums[i]) {
                return new int[] { i, j };
            }
        }
    }
    throw new IllegalArgumentException("没有这两个数");
}
```

该种解法是最为直观简单的，对每个元素都将在数组中往后寻找其是否有满足`target`的值。

复杂度分析

- 时间复杂度：$n-1+n-2+\ldots\ldots 1=O(N^2)$。
- 空间复杂度：由于没有开辟额外的空间，因此其空间复杂度为$O(1)$。

方法2，使用HashMap

```java
    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            map.put(nums[i], i);
        }
        for (int i = 0; i < nums.length; i++) {
            int res = target - nums[i];

            /**
             * 顺序不能搞反，先确定map中含有这个元素，再去取值与i判断
             * 若先get，再containskey，当i=0，此时res=8，但是数组中没有这个元素，get就会
             * 空指针异常
             */
            if (map.containsKey(res) && map.get(res) != i) {
                return new int[]{i, map.get(res)};
            }
        }
        throw new IllegalArgumentException("没有这两个数");
    }
```

利用`HashMap`来存储每个数组元素所对应的下标，然后通过`HashMap`的常数级时间的查询效率来获取相加为target所需的值，相比上面的暴力解法，大大减少了所需时间。

还可以只使用一次for 循环

```java
    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int res = target - nums[i];
            if (map.containsKey(res)) {
                return new int[]{map.get(res), i};
            }
            map.put(nums[i], i);
        }
        throw new IllegalArgumentException("没有这两个数");
    }
```

复杂度分析

- 时间复杂度：由于`HashMap`每次查询的时间复杂度为$O(1)$，因此只考虑`for`循环所带来的影响，所以其时间复杂度为$O(N)$。
- 空间复杂度：使用了`HashMap`来对数组中的元素进行了额外的存储，因此其空间复杂度为O(N)。