1 两数之和

Label：Hash

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，数组中同一个元素不能使用两遍。

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]
```

- 双重遍历

```java
// 暴力法
class Solution {
    public int[] twoSum(int[] nums, int target) {
        for (int i = 0; i < nums.length - 1; i++) { // -1 是因为没有必要遍历最后一个
            for (int j = i + 1; j < nums.length; j++) {
                if (nums[i] + nums[j] == target)
                    return new int[]{i,j};  // 创建数组
            }
        }
        return null;
    }
}
// 排序之后再进行遍历，相当于双重指针的做法，虽然不用遍历全部的
```

- 两次遍历HashMap

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
         Map<Integer, Integer> map = new HashMap<>(); // 这里必须显示定义出Map K V的类型，防止后面使用-时重复进行强转

         // 先插入
         for (int i=0;i < nums.length;i++) {
             map.put(nums[i],i);
         }
         // 再遍历查找
         for (int i=0; i < nums.length; i++) {
             int diff = target - nums[i];
             if (map.containsKey(diff) && map.get(diff) != i) {  // 防止出现自己加自己的情况
                 return new int[] {i, map.get(diff)};
             }
         }
         return null;
    }
}
```

- 一次遍历

```
// 边插入，边遍历查找
class Solution {
    public int[] twoSum(int[] nums, int target) {

        Map<Integer,Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length ; i++) {
            int diff = target - nums[i];
            if (map.containsKey(diff)) {  // 因为插入在判断之后，所以不用防止自己加自己
的情况
                return new int[] {map.get(diff), i};
            }
            map.put(nums[i], i);
        }
        throw new IllegalArgumentException("no solution");
    }
}
```