# 两数之和

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]`

## 方法一 暴力求解

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
       vector<int>res;
       for(int i=0;i<nums.size();i++)
       {
           for(int j=i+1;j<nums.size();j++)
           {
               if(nums[i]+nums[j]==target)
               {
                   res.push_back(i);
                   res.push_back(j);
                   break;
               }
           }
       }
       return res;
    }
};
```

方法二 哈希表

哈希表使用无序哈希表存储，通过遍历 nums 中每一个 找 target-nums[i]是否在 hash 表中，如果存在，返回元素的迭代器，通过迭代器的 second 属性获取值，没有找到返回 unordered_map.end

创建一个哈希表，对于每一个 x，我们首先查询哈希表中是否存在 target-x，然后将 x 插入到哈希表中，即可保证不会让 x 和自己匹配

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> hash;
        vector<int> res;
        for(int i=0;i<nums.size();i++)
        {
            auto it=hash.find(target-nums[i]);//查找 key 值（target-nums[i]）的元素，找到返回元素的迭代器，通过迭代器的 second 属性获取值，没有找到返回 unordered_map.end
            if(it!=hash.end())//找到
            {
                res.push_back(it->second);
                res.push_back(i);
            }
            hash[nums[i]]=i;//没有找到  则第 i 个数作为新的键值对 添加
        }
        return res;
    }
};
```