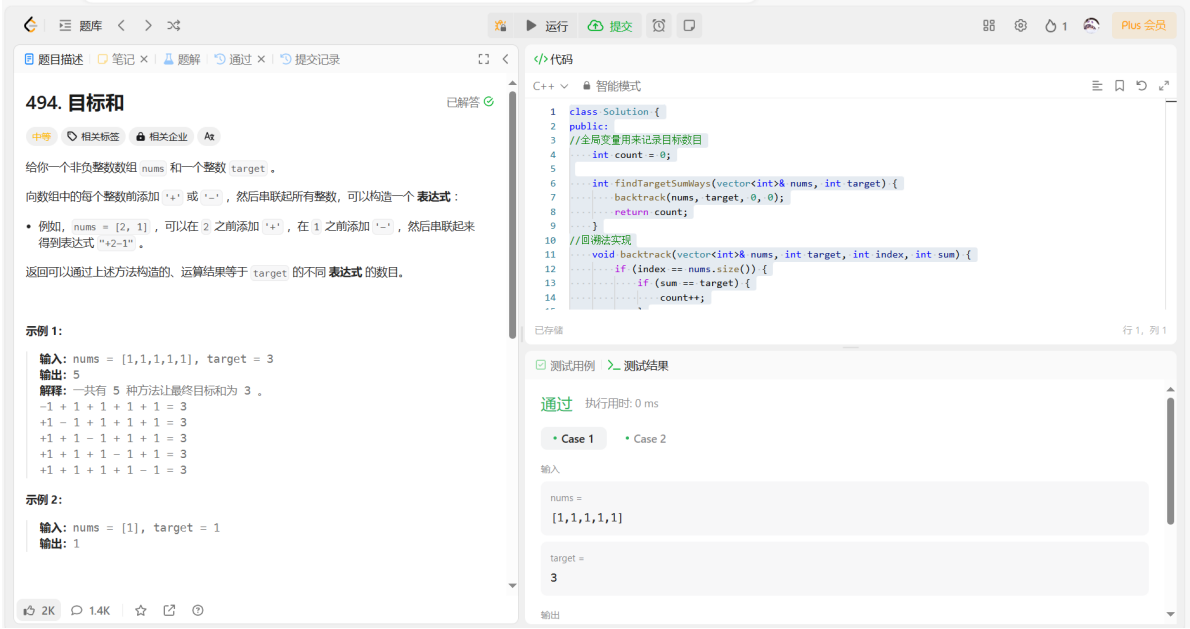
题目一：目标和



解题思路一：

回溯法，对于每一个解，一定是有构成解的成员的个数是等于数组大小的，那么我们不断回溯，回溯的终止条件就是index=nums.size()，对于满足要求的，判断是否构成解，能则++，对于不符号终止条件的，我们回溯时更新索引并分别选择对下一个元素+|-,以达到符号不同

```
class Solution {
public:
//全局变量用来记录目标数目
    int count = 0;

    int findTargetSumWays(vector<int>& nums, int target) {
        backtrack(nums, target, 0, 0);
        return count;
    }
//回溯法实现
    void backtrack(vector<int>& nums, int target, int index, int sum) {
        if (index == nums.size()) {
            if (sum == target) {
                count++;
            }
        } else {
            //递归处理不同的符号和
            backtrack(nums, target, index + 1, sum + nums[index]);
            backtrack(nums, target, index + 1, sum - nums[index]);
        }
    }
};
```

解题思路二：

动态规划：设所有正数元素之和是p，所有负数元素之和的绝对值是q,那么有p+q=s,p-q=target,那么p=(s-target)/2,q=(s+target)/2,对target的正负分别讨论，得到最小背包数目是(s-abs(target))/2是01背包的最优解，对于s,如果该值<0说明一定没有解，背包的空间是0，如果不能被2整除也没有解，因为解的个数不能是小数个，有解时，解的数目一定是大于等于1的，因此有f[0]=1，只要当前的值的和小于背包容量，那么说明还有解，f[c]+=f[c-x]

```
#include<iostream>
#include<vector>
#include<unordered_map>
#include<stack>
#include<numeric>
using namespace std;

class Solution {
public:
    int findTargetSumWays(vector<int>& nums, int target) {
        int sums = 0;
        for (int k : nums)
        {
            sums += k;
        }
        int s = sums - abs(target);
        //%2=0时说明是奇数，奇数/2有余数，说明没有解
        if (s < 0 || s % 2) {
            return 0;
        }
        //有解的情况那么解的数目一定是>=1的
        int m = s / 2; // 背包容量
//把解都给初始化
        vector<int> f(m + 1);
        f[0] = 1;
        for (int x : nums) {
            for (int c = m; c >= x; c--) {
                f[c] += f[c - x];
            }
        }
        return f[m];
    }
};

int main()
{
    vector<int> test = { 2,2,1,1,2 };
    Solution sol;
    int result=sol.findTargetSumWays(test, 2);
    cout << result;
    return 0;
}
```