494 目标和

Label：深度优先遍历、动态规划

给你一个整数数组 nums 和一个整数 target 。
向数组中的每个整数前添加 '+' 或 '-' ，然后串联起所有整数，可以构造一个 表达式 ：

例如，nums = [2, 1] ，可以在 2 之前添加 '+' ，在 1 之前添加 '-' ，然后串联起来得到表达式 "+2-1" 。
返回可以通过上述方法构造的、运算结果等于 target 的不同 表达式 的数目。

```
1 <= nums.length <= 20
0 <= nums[i] <= 1000
0 <= sum(nums[i]) <= 1000
-1000 <= target <= 100
```

输入：nums = [1,1,1,1,1], target = 3
输出：5
解释：一共有 5 种方法让最终目标和为 3 。
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3

- 枚举 深度递归

```java
class Solution {
    int count = 0;
    public int findTargetSumWays(int[] nums, int target) {
        find(nums, target, 0);
        return count;
    }

    private void find(int[] nums, int re, int start) {
        if (start == nums.length) {  // 最后
            if (re == 0) {
                count++;
            }
        }else{
            find(nums, re - nums[start], start+1);
            find(nums, re + nums[start], start+1);
        }
    }
}
```

- 动态规划

```
public class Solution {
    public int findTargetSumWays(int[] nums, int target) {
        int[][] dp = new int[nums.length][2001];//dp[i][j] 表示用数组中的前 i 个元素，组成和为 j 的方案数
        // 考虑第 i 个数 nums[i]，它可以被添加 + 或 -，因此状态转移方程如下：
        // dp[i][j] = dp[i - 1][j - nums[i]] + dp[i - 1][j + nums[i]]
        // 也可以写成递推形式：
        //       dp[i][j + nums[i]] += dp[i - 1][j]
        //       dp[i][j - nums[i]] += dp[i - 1][j]
        //  所有数的和不超过 1000，那么 j 的最小值可以达到 -1000。在很多语言中，是不允许数组的下标为负数的，因此我们需要给 dp[i][j] 的第二维预先增加 1000
        //   dp[i][j + nums[i] + 1000] += dp[i - 1][j + 1000]
        //   dp[i][j - nums[i] + 1000] += dp[i - 1][j + 1000]

        dp[0][nums[0] + 1000] = 1;
        dp[0][-nums[0] + 1000] += 1;  // 注意 这里是 += 1
        for (int i = 1; i < nums.length; i++) {
            for (int sum = -1000; sum <= 1000; sum++) {
                if (dp[i - 1][sum + 1000] > 0) {
                    dp[i][sum + nums[i] + 1000] += dp[i - 1][sum + 1000];
                    dp[i][sum - nums[i] + 1000] += dp[i - 1][sum + 1000];
                }
            }
        }
        return dp[nums.length - 1][target + 1000];
    }
}
```

- 动态规划（todo）

```
//                          sum(P) - sum(N) = target
//      sum(P) + sum(N) + sum(P) - sum(N) = target + sum(P) + sum(N)
//                                2 * sum(P) = target + sum(nums)
//      只需求解合成sum(P)的方式有多少种

class Solution {
    public int findTargetSumWays(int[] nums, int target) {
        int sum = Arrays.stream(nums).sum();
        int sumP = (target + sum)/2;

        if (sum < target || (sum + target) % 2 == 1) return 0;

        int[] dp = new int[sumP + 1]; // 转为动态规划求和问题，能组合成sum(P)的方式有多少种
        dp[0] = 1;    // dp[i]表示在target为i时的解法数量
        for (int num : nums) {
            for (int i = sumP; i >= num; i--) {
                dp[i] += dp[i - num];
            }
        }
        return dp[sumP];
    }
}
```