Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.
```

这儿有问题dp[i][j] = Math.max(dp[i-1][j], dp[i-1][j-nums[i]] + nums[i])

```
    * 这个题目就是一个0-1背包问题，首先对数组元素求和，因为要分为两个数组，且两个数组之和
    * 相等，因此若为奇数，直接返回false，若为偶数，则建立一个f[n][sum+1]的数组，sum为和的一半
    * dp[i][j]表示前i个放入（不一定都放进去）产品放入一个和为j的背包可以获得的最大价值。
    * 这里的每个产品的价值就是nums[i]的数值大小
    */
    public boolean canPartition(int[] nums) {
        int sum = 0;
        for (int i : nums)
            sum += i;
        if(sum % 2 == 1) {
            return false;
        }else {
            sum /= 2;
            int n = nums.length;
            int [][] dp = new int [n][sum + 1];
            //先初始化，表示容量为nums[0]到sum时，放入第i个商品，最大价值
            for(int i = nums[0]; i <= sum; i ++) {
                dp[0][i] = nums[0];
            }
            for(int i = 1; i < n; i ++) {
                for(int j = nums[i]; j <= sum; j ++) {
                    dp[i][j] = Math.max(dp[i-1][j], dp[i-1][j-nums[i]] + nums[j]);
                }
            }
            if(dp[n-1][sum] == sum) {
                return true;
            }else {
                return false;
            }
        }
```