53 最大子序和

Label：动态规划

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例：
    输入: [-2,1,-3,4,-1,2,1,-5,4],
    输出: 6
    解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

- 动态规划

如果前面的sum没发给后面的第i个元素带来正向的收益（前面的sum为负数就根本没必要再相加了），那么直接取i值，否则sum还需要再加一个负数，并且不需要在意他的序列是什么。

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int sum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            if (sum <= 0) {
                sum = nums[i]; // 不能带来正向收益，那加了以后肯定会比nums[i]小，所以直接
就去nums[i]
            }else {
                sum += nums[i];
            }

            maxSum = Math.max(sum, maxSum);  // 动态计算每一次迭代的sum，保存最大的sum
        }
        return maxSum;
    }
}
```

- 代码优化

```
class Solution {
    public int maxSubArray(int[] nums) {
        int max = nums[0];
        int now = nums[0];
        for (int i = 1; i < nums.length ; i++) {
            now = Math.max(nums[i], now + nums[i]);
            max = Math.max(max, now);
        }
        return max;
    }
}
```

- 暴力(O(n^3))

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];

        for (int i = 0; i < nums.length; i++) {
            for (int j = i; j < nums.length; j++) {
                int tempSum = 0;

                for (int k = i; k <= j; k++) {
                    tempSum += nums[k];
                }

                maxSum = Math.max(tempSum, maxSum);
            }
        }
        return maxSum;
    }
}
```

- 暴力改进(O(n^2))

```
class Solution {
    public int maxSubArray(int[] nums) {

        int maxSum = nums[0];

        for (int i = 0; i < nums.length; i++) {
            int tempSum = 0;

            for (int j = i; j < nums.length; j++) {
                tempSum += nums[j];

                if (tempSum > maxSum)  // 在构造的时候就比一遍
                    maxSum = tempSum;
            }
        }
        return maxSum;

    }
}
```