

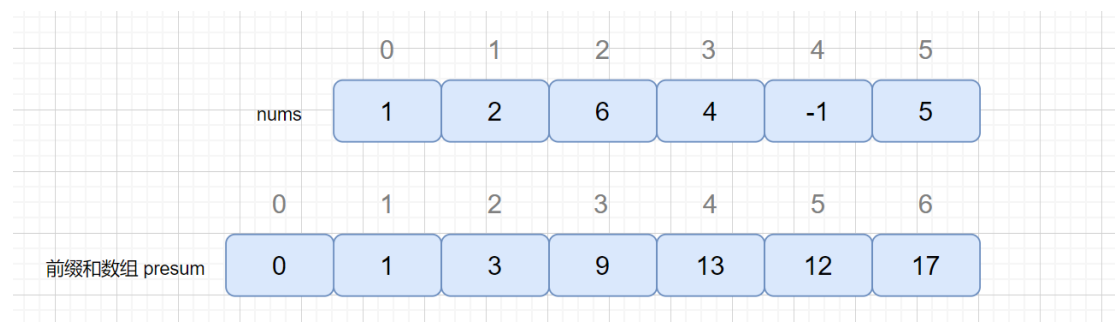
涉及到子数组、求和问题，我们一定要想到前缀和，因为前缀和可以有效地节省求和时间

今天我们来谈一下刷题时经常用到的前缀和思想，前缀和思想和滑动窗口会经常用在求子数组和子串问题上，当我们遇到此类问题时，则应该需要想到此类解题方式

**区分是用前缀和还是用滑动窗口的关键是：元素是否有负数。有负数则使用前缀和，没负数则使用滑动窗口**

下面我们先了解一下什么是前缀和。

前缀和其实我们很早之前就了解过的，我们求数列的和时， $S_n = a_1 + a_2 + a_3 + \dots + a_n$ ；此时  $S_n$  就是数列的前  $n$  项和。例  $S_5 = a_1 + a_2 + a_3 + a_4 + a_5$ ； $S_2 = a_1 + a_2$ 。所以我们完全可以通过  $S_5 - S_2$  得到  $a_3 + a_4 + a_5$  的值，这个过程就和我们做题用到的前缀和思想类似。我们的前缀和数组里保存的就是前  $n$  项的和。见下图

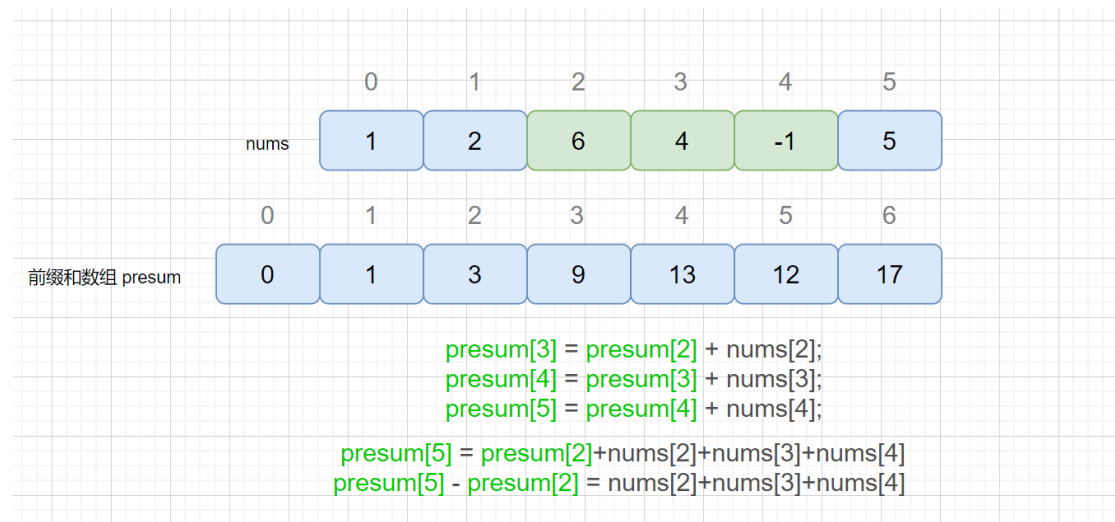


相当于前缀和数组多定义一个位置，这样剩下的位置表示原数组前  $n$  位的和

我们通过前缀和数组保存前  $n$  位的和， $presum[1]$ 保存的就是  $nums$  数组中前 1 位的和，也就是  $presum[1] = nums[0]$ ， $presum[2] = nums[0] + nums[1] = presum[1] + nums[1]$ 。依次类推，所以我们通过前缀和数组可以轻松得到每个区间的和。

例如我们需要获取  $nums[2]$  到  $nums[4]$  这个区间的和，我们则完全根据  $presum$  数组得到，是不是有点和我们之前说的字符串匹配算法中 BM,KMP 中的  $next$  数组和  $suffix$  数组作用类似。那么我们怎么根据  $presum$  数组获取  $nums[2]$  到  $nums[4]$  区间的和呢？

见下图



也就是  $\text{presum}[5] - \text{presum}[2] = \text{nums}[2] + \text{nums}[3] + \text{nums}[4]$

获取  $\text{nums}[2]$  到  $\text{nums}[4]$  区间的和就相当于以 1 为起始 index 的数组中第三个数到第五个数之间的和。也就是前五个数的和( $\text{presum}[5]$ )减去前两个数的和( $\text{presum}[2]$ )。也就是用  $\text{presum}[5] - \text{presum}[2]$

这个前缀和数组  $\text{presum}$  的含义也很好理解,  $\text{presum}[i]$  就是  $\text{nums}[0..i-1]$  的和。那么

如果我们想求  $\text{nums}[i..j]$  的和, 只需要一步操作  $\text{presum}[j+1] - \text{presum}[i]$  即可, 而不需要重新去遍历数组了。

### 构造前缀和数组:

```
for (int i = 0; i < nums.length; i++) {
    presum[i+1] = nums[i] + presum[i];
}
```

好啦, 我们已经了解了前缀和的解题思想了, 我们可以通过上面这段代码得到我们的前缀和数组, 非常简单