303. 区域和检索 - 数组不可变

算法 动态规划

> 题目：Range Sum Query - Immutable
> 语言: python3
> 英文版链接：https://leetcode.com/problems/range-sum-query-immutable/description/
> 中文版链接：https://leetcode-cn.com/problems/range-sum-query-immutable/

题目分析

首先拿到这一题，我想所有使用python的同学都会首先想到python的切片操作，用切片操作解决这个问题简直太简单了，如下：

```
class NumArray:

    def __init__(self, nums: List[int]):
        self.nums = nums

    def sumRange(self, i: int, j: int) -> int:
        return sum(self.nums[i:j+1])
```

那如果不用切片操作，按照正常思路，直接取索引，然后累加求和即可：

```
class NumArray:

    def __init__(self, nums: List[int]):
        self.nums = nums

    def sumRange(self, i: int, j: int) -> int:
        sum = 0
        for k in range(i, j+1):
            sum += self.nums[k]
        return sum
```

但提交了这个代码，你就会发现**超时了**，为什么会超时呢？按道理来说这是一个非常正常的操作。

但是仔细读题可以发现，是有问题的， 说明中提到“**会多次调用 sumRange 方法**”， 也就是说，对于一个数组来说，如果求解[2, 4]的累加，又求了[2,5]的累加，这样等于[2,4]的累加被重复计算了一遍，这就是问题的关键，也是动态规划和递归中常常要解决的，**重复子问题**。所以每次都逐个相加计算子区间的和不是理想的做法。

但是这里的问题是，给定我的区间是不可控的，长度不知道到底是多大，无法直接求解子问题，所以我们可以作一个转换:

$$sum[i,j] = sum[j+1] - sum[i]$$

其中sum[i]表示从0到i-1的累加。这样我们用一个list，把这些值缓存起来，再遇到的时候如果已经缓存过了，就直接取值，所以就没有大量的重复操作了，即可得到问题的解。

答案

我们首先来看看记忆化搜索的解决方式:
代码比较简单，就不多解释了。

```
class NumArray:

    def __init__(self, nums: List[int]):
        self.nums = nums
        self.sums = [-1] * (len(self.nums) + 1)

    def sumRange(self, i: int, j: int) -> int:
        if self.sums[i] == -1:
            self.sums[i] = self.sum_x(i)
        if self.sums[j + 1] == -1:
            self.sums[j + 1] = self.sum_x(j + 1)
        return self.sums[j + 1] - self.sums[i]
    def sum_x(self, x):
        sum = 0
        for i in range(x):
            sum += self.nums[i]
        return sum
```

再看**动态规划**的解决方式:

```
class NumArray:

    def __init__(self, nums: List[int]):
        self.nums = nums
        self.sums = [-1] * (len(self.nums) + 1)
        for i in range(1, len(self.nums)+1):
            self.sums[i] = self.sums[i - 1] + self.nums[i - 1]

    def sumRange(self, i: int, j: int) -> int:
        return self.sums[j + 1] - self.sums[i]
```

本题用动态规划进行解决是最优的方式!