【前端面试】实现数组求和！不能用循环、...

前言

如果你没有可以的去刷过题或者学习算法，那么我相信很多人的心理过程都是这样一个变化：数组求和？这太简单了！...不对，不能用循环，不能用内置函数？那我咋办啊！

很多小伙伴刚开始以为这道题很简单，但是仔细一看却摸不着头脑。其实这道题说难不难，说简单也不简单，主要是看你对算法的灵敏度如何了。

1.题目描述

虽然通过文章标题，我们大概能够知道这是一道数组求和的题目，但是至于数组是否固定、数组是否全是数据等等我们还需要细化一下，让这道题变得更严谨起来。

题目描述：

> 给定任意长度的整数数组 arr，例如[1,2,3,4,5,6]，实现一个求和函数 sum，在不利用循环、标准库函数的情况下实现数组的求和。

输入：

> sum([1,2,3])

输出：

> 6

到这里我们基本上明白题目的意思了，就是给定一个任意长度的整数数组，在某些限制条件下实现求和。

2.到底考察什么？

我们平时实现数组求和直接循环一遍就好了，实在不行借助一些内置函数也是能够实现的，但是如果我们想抛开这两项，我们还能如何操作数组呢？

很简单：**我们可以直接操作数组下标！**

但是问题又接踵而来了，数组长度是不固定，我们怎么知道数组下标何时没有呢？

这个时候我们又得换一个思路了，既可以使用数组下标，又不用担心长度的问题，那该用什么呢？

到这里我们不妨考虑一下**递归**！递归是一个比较基础但是又必要要会的算法知识，递归的实现有一点循环的意思来里面，但是它又不是循环，而且还有终止条件，也解决了我们数组长度步固定的问题。

所以本题考察的重点就是：**递归！**

如果还不会递归的建议去好好学习一下，因为这是算法考察中避不开的一道坎！

递归的典型案例：

我们在学习递归算法的时候，通常都是拿一道非常经典的案例来讲解：斐波那契数列。那么什么是斐波那契数列呢，我们看下面题目：

> 已知：1、1、2、3、5、8、13、21.....
>
> 由上可知：f(1) = 1、f(2) = 1、f(3) = 2 ......f(n) = f(n-1) + f(n-2)

上面其实就是一道数学题，当然上面的数列就是斐波那契数列，我们可以使用递归的方式求得第 n 个数的值。

代码如下：

```
function fib(n){
    if(n==1 || n==2){
        return 1;
    }
    return fib(n-1) + fib(n-2);
}
```

3.解题思路

上一节我们给出了递归解决斐波那契数列的代码，大家应该能发现，代码的本质上就是一个求和操作，而我们这里不正是想要实现数组求和吗?

我们假设数组为：[1,2,3,4,5,6]

实现数组求和无非就是数组第一位累加到最后一位，如下：

```
1 + 2 + 3 + 4 + 5 + 6
```

而我们的斐波那契数列实质上就是前面数字的累加求和，和我们的数组求和不约而合。

假如我们有一个函数 f(n)，n 代表从数组坐标哪一位开始向后累加求和，那么可以设想出下列公式：

> f(5) = arr[5] // 6
>
> f(4) = f(5) + arr[4] // 6 + 5
>
> f(3) = f(4) + arr[3] // 6 + 5 + 4
>
> ...

> fn(n) = f(n+1) + arr[n]

公式我们已经推导出来了，f(n)返回的就是从数组坐标 n 开始向后的所有元素的累加和。

实现思路：

利用公式 fn(n) = f(n+1) + arr[n]实现数组求和，传入 n 为 0，即代表数组所有元素的和。

4.代码实现

既然解题思路有了，那么我们就可以实现它了，首先来实现 f(n)函数。

代码如下：

```
<script>
  let arr = [1, 2, 3, 4, 5, 6];
  function f(n) {
    return n >= arr.length ? 0 : f(n + 1) + arr[n];
  }
  console.log(f(0)); // 21
  console.log(f(1)); // 20
  console.log(f(2)); // 18
</script>
```

上段代码我们直接把之前总结的公式带入函数了，当然多加了一个判断，如果传入的 n 大于等于数组长度了，说明到头了，无需再递归函数了，所以直接返回 0。

虽然已经实现了数组的求和，但是我们题目要求是实现 sum 函数，该函数接收的是一个数组，所以我们还得封装一下。

代码如下：

```
<script>
  let arr = [1, 2, 3, 4, 5, 6];
  function sum(arr) {
    function f(n) {
      return n >= arr.length ? 0 : f(n + 1) + arr[n];
    }
    return f(0);
  }
  console.log(sum(arr)); // 21
</script>
```

到这里我们这道题目就算解决了，没有借助循环和内置函数，实现了数组的求和。

总结

递归是我们算法中很重要的一种思想，大家一定要理解它，今天这道题目其实不算难，但是如果不知道递归，估计还是无从下手吧！

那么，小伙伴们是否还有其它方法实现呢？欢迎留言！