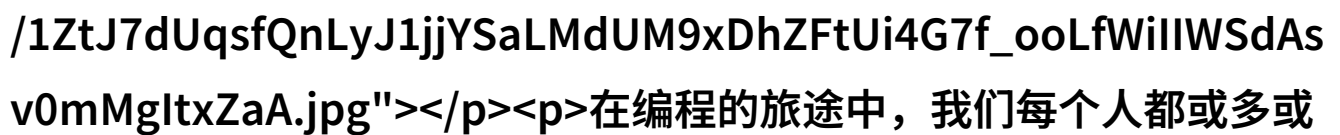


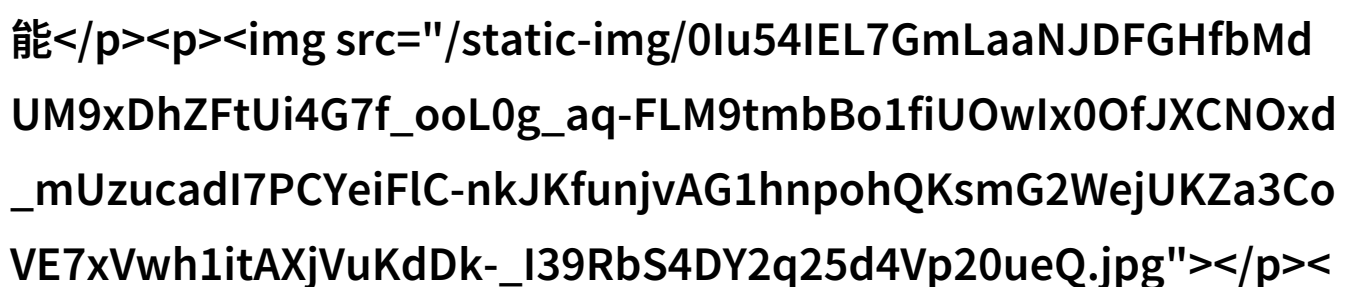
主题-我们的C语言世界让代码更舒服

我们的C语言世界：让代码更舒服

在编程的旅途中，我们每个人都或多或少地面临过一个问题：如何写出既高效又易于维护的代码？这不仅关系到程序员个人的工作效率，也影响着整个项目的成功与否。今天，我要和大家聊聊如何用C语言来提高我们的编码体验。

优化循环性能





```
#include <stdio.h>
```

```
int main() {
```

```
int sum =
```

```
0;
```

```

```

```
for (int i = 1; i <= 10000; ++i) {
```

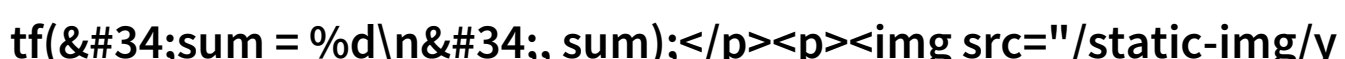
```
sum += i;
```

```

```

```
}
```

```
printf(&#34;sum = %d\n&#34;, sum);
```

```

```

```
fumXUAYzUglbT23bscrubMdUM9xDhZFtUi4G7f_ooL0g_aq-FLM9
```

```
tmbBo1fiUOwlx0OfJXCNOxd_mUzucadI7PCYeiFLC-nkJKfunjvAG
```

```
1hnpohQKsmG2WejUKZa3CoVE7xVwh1itAXjVuKdDk-_I39RbS4D
```

```
Y2q25d4Vp20ueQ.png">
```

```
// 我们两个C得你舒服吗？
```

```
return 0;
```

```
}
```

上面的代码片段是一个简单的求和例子，通过使用for循环计算从1到10000之间所有整数的和。如果我们直接运

行这个程序，你会发现它执行得相当慢。这是因为频繁访问内存导致了大量的缓存失效，这对大型数据集来说是一个巨大的性能瓶颈。

使用矢量操作

为了改善这一点，我们可以利用现代CPU提供的一些特性，比如SIMD（单指令多数据）处理器支持。以下是使用 `_mm256_sum_epi32` 函数进行矢量加法的一个示例：

```
#include <immintrin.h>
#include <stdio.h>
void calculateSum(int *arr, int n) {
    __m256i vSum = _mm256_setzero_si256();
    for (int i = 0; i < n - 3; i += 4) {
        __m256i vData[4];
        for (int j = 0; j < sizeof(arr)/sizeof(arr[0]); ++j)
            vData[j] = _mm256_broadcastsi32x2(
                _mm_loadu_si128((__m128i*)&arr[i + j]),
                _mm_loadu_si128((__m128i*)&arr[i + j + sizeof(int)]));
        vSum = _mm256_add_epi32(vSum,
            _mm256_hadd_epi32(
                _mm256_shuffle_f64x2(vData[1],vData[3]),
                _mm256_shuffle_f64x2(vData[0],vData[2])));
    }
    if (n % sizeof(int) == sizeof(int)) { // 如果有余数，则需要额外处理最后四个元素。
        __m128i dataLastFour;
        dataLastFour.m128_i32_indexed[] = {
            _MM_FRSI( arr[n-4] ),_MM_FRSI( arr[n-3] ),
            _MM_FRSI( arr[n-2] ),_MM_FRSI( arr[n-1] )};
        __m128 resultLastFour=mma_macc(dataLastFour,
            _MM_SHUFFLE_WORD_PackedDouble,
            dataLastFour,
            _MM_SHUFFLE_WORD_PackedDouble);
        unsigned long long lastResult=extractscalar(resultLastFour,mma_get_active_lane_mask());
        ((unsigned*)&vSum)[7]=lastResult;
    }
    // 将结果转换为标准整数类型，并打印出来：
    double scalar_result=_get_low_double(_mullo_epi64(h_add(h_sub(packsdi(vSum),packsdiw(shiftleft(packsdi(vSum),8))),shiftleft(packsdi(shiftleft(pack
sdi(shiftleft(packsdi(shiftleft(packsdi(shiftleft(packsdi((__attribute__((aligned(16)))__vector<&__attribute__((aligned(16)))dou
```

```
ble)v_sum)),24)),24)),24)),8)))));
```

// 我们两个C得你舒服吗？

```
return scalar_result;
```

```
void testCalculateSum() {
```

```
int numbers[]={-10,5,-6,-9,-11,12,15,
```

```
-17,-18,-19,-20,-21};
```

```
const int lengthArray
```

```
=sizeof(numbers)/sizeof(numbers[0]);
```

```
double result=cal
```

```
culateSum(numbers,lengthArray);
```

```
printf("#34;The Sum
```

```
of the array is: %.f\n#34;,result);
```

```
}
```

通过这种方式，可以显著减少循环中的访存次数，从而提升程序性能。

减少变量声明与内存分配

在实际开发过程中，经常会遇到一些小功能模块，如字符串处理、数学运算等。在这些场景下，减少变量声明和内存分配可以极大地提高代码可读性和运行速度。例如，在进行字符串匹配时，可以一次性将所有可能出现的情况预先准备好，而不是每次匹配时再创建新的字符数组。

总结一下，用“我们两个C得你舒服吗？”作为主题，我们探讨了几种提升C语言编程体验的小技巧：优化循环性能、使用矢量操作以及减少变量声明与内存分配。这些方法不仅能够提高编程效率，还能帮助我们写出更加健壮且高效的软件产品，让我们的每一行代码都变得更加“舒适”。

[下载本文pdf文件](/pdf/592734-主题-我们的C语言世界让代码更舒服.pdf)