

## 80 How to make your STRINGS faster

当然处理字符串时，很多人没有意识到字符串对程序性能的影响，而程序中还经常有很多字符串操作。我们能做一件非常简单的事情，能让我们的字符串以及字符串的操作更快速。

### 1. std::string 的内存分配

`std::string` 的主要问题之一，可能就是字符串格式化以及字符串操作，因为它们要分配内存。堆上分配内存是不可能避免的，但应该尽量避免，因为它会降低程序的速度，而 `std::string` 和它的很多函数都喜欢分配。

```
C++

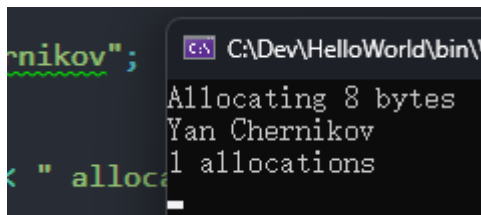
static uint32_t s_AllocCount = 0;

// 可以通过重载new操作符来实现跟踪内存分配
void* operator new(size_t size)
{
    s_AllocCount++;
    std::cout << "Allocating " << size << " bytes\n";
    return malloc(size);
}

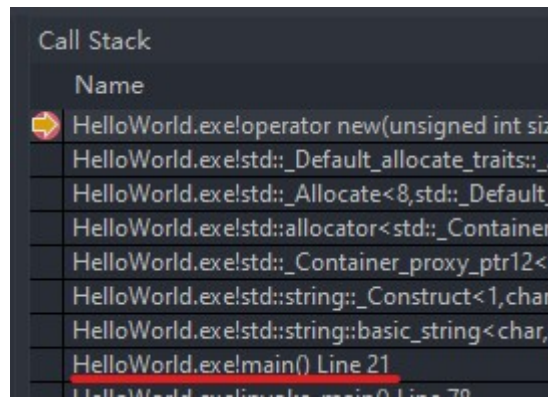
void PrintName(const std::string& name)
{
    std::cout << name << std::endl;
}

int main()
{
    std::string name = "Yan Chernikov";
    PrintName(name);

    std::cout << s_AllocCount << " allocations" << std::endl;
    std::cin.get();
}
```



在分配内存的位置设断点，可以看到发生在创建字符串的这一行，是这个时候导致了堆分配。



如果我们去掉这个字符串，只是复制粘贴const char数据到PrintName函数中呢，这是否分配了内存？

```
//std::string name = "Yan Chernikov";  
PrintName("Yan Chernikov");
```

结果显然是也会分配，因为虽然函数接受的是const引用，但是它仍然需要为我们构造一个std::string，构造仍然需要分配内存。

现在稍微复杂一点：

```
std::string name = "Yan Chernikov";  
  
std::string firstName = name.substr(0, 3);  
  
std::string lastName = name.substr(4, 9);  
PrintName(firstName);
```

```
std::string name = "Yan Chernikov";  
  
std::string firstName = name.substr(_off: 0, _Count: 3);  
  
std::string lastName = name.substr(_off: 4, _Count: 9);  
PrintName(firstName);
```

```
C:\Dev\HelloWorld\bin\W  
Allocating 8 bytes  
Allocating 8 bytes  
Allocating 8 bytes  
Yan  
3allocations
```

这样简单的操作都会造成三次分配，可以想象这样的事情会在你的程序中经常发生，如果你有一些实时运行的程序，比如游戏，如果你每帧都在做这种事情，它会堆积起来，损害你的帧速率。

怎么能让这一切变得更好呢？

有一种很简单的方法，可以修改现有代码，将分配减少到一个。在本节最后会删除所有分配，也就是会有0个分配。

首先来看这段代码，你想要做什么，往往这样就知道优化的方向了。这里你真正需要的是这个字符串的视图，而不需要复制数据创建这么多新字符串。

这就是string\_view发挥作用的地方了：

## 2. std::string\_view

`std::string_view`是C++17中的一个新类，它本质上只是一个指向现有内存的指针，换句话说，就是一个`const char`指针，指向它人拥有的现有的字符串，再加上一个`size`（大小）。

它的工作方式类似于创建一个窗口，进入现有内存的小视图，而不是分配一个新的字符串，这样就没有内存分配，按值传递字符串视图是非常轻量级的，因为它只是一个指针和一个大小。

将之前的`substr`重写：

```
C++

void PrintName(std::string_view name) // 这里就不用引用传递了，类型改为string_view
{
    std::cout << name << std::endl;
}

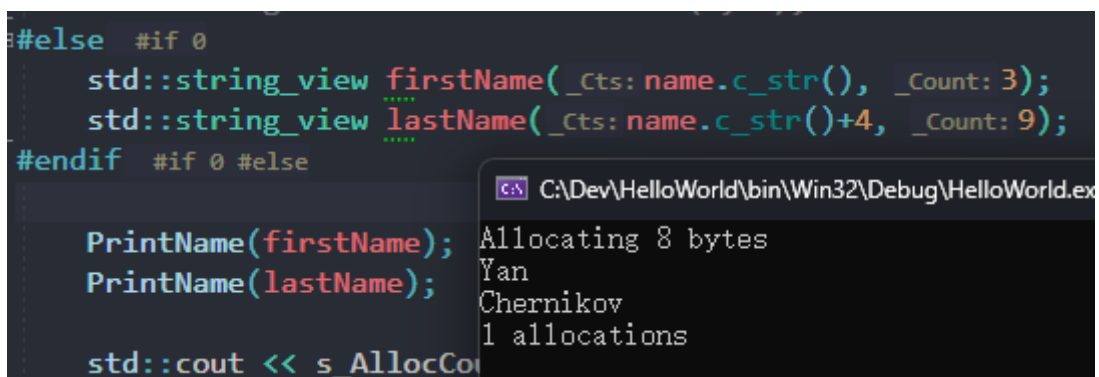
int main()
{
    std::string name = "Yan Chernikov";

    #if 0
        std::string firstName = name.substr(0, 3);

        std::string lastName = name.substr(4, 9);
    #else
        std::string_view firstName(name.c_str(), 3);
        std::string_view lastName(name.c_str()+4, 9);
    #endif

    PrintName(firstName);

    std::cout << s_AllocCount << " allocations" << std::endl;
    std::cin.get();
}
```



```
#else #if 0
    std::string_view firstName(_Cts: name.c_str(), _Count: 3);
    std::string_view lastName(_Cts: name.c_str()+4, _Count: 9);
#endif #if 0 #else

PrintName(firstName);
PrintName(lastName);

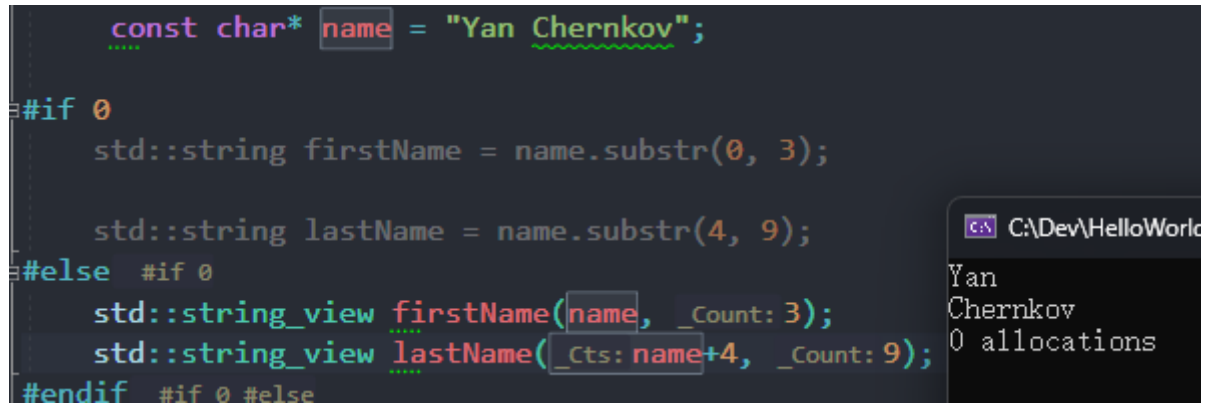
std::cout << s_AllocCo
```

C:\Dev\HelloWorld\bin\Win32\Debug\HelloWorld.exe

Allocating 8 bytes  
Yan  
Chernikov  
1 allocations

现在只有一次分配了，但其实是完全可以去掉所有的分配的，通过完全不使用 `std::string` 来实现：

```
const char* name = "Yan Chernkov";  
#if 0  
    std::string firstName = name.substr(0, 3);  
  
    std::string lastName = name.substr(4, 9);  
#else #if 0  
    std::string_view firstName(name, _Count: 3);  
    std::string_view lastName(_Cts: name+4, _Count: 9);  
#endif #if 0 #else
```



由于这里就是个静态字符串，没必要非得用std::string来实现，所以可以这么做。

如果字符串是来自某个文件或以某种方式生成的，在这种情况下，让它是一个std::string可能是更现实的情况