

第14篇:C++的string-两手抓的内存分配



铁甲万能狗

自由开发者，专攻C++/Python后端开发(简书平台同号)

12 人赞同了该文章

本文假定你对C/C++的string语法已经有基本的了解。

- 如果你对C++的string的内部实现原理不了解的话,请先阅读这一篇 [《对\[C/C++\]指针与字符串的总结》](#), 本篇是对前一篇的内容的延伸。
- 如果对字符串字面量不了解的话, 请先阅读 [《C/C++中的字符字面量》](#)。

C++的string对象实质上就是一个容器,其内部有一个c_str方法能够返回一个指向的实质存储字符串副本的数据成员。即通过string::c_str()配合printf函数可以获取的字符串副本的内存地址。

栈中的string的内存分配

首先,我们来看看如下代码的**关于string对象内部的栈中内存分配**,不少C++读物强力建议在C++开发中使用标准库的string对象,而非C版本的char*指针和char[]数组。但没有详细告诉读者为什么? string对象底层都做了些什么,因此理解string内部实现原理,对于你后续使用string类实现各种字符串操作的算法非常有必要, 以下代码, 是前一篇文章代码的深入的**演示版本**

首先我们在全局作用域重载了operator new和operator delete的函数原型,内部分别用C版本的malloc和free函数,目的在于:显式展示给读者,你在使用string过程中,它已经在底层自动完成了所有的内存分配和内存释放。实际开发过程不建议这样重载operator new和operator delete。

```
1  #include <cassert>
2  #include <string>
3  #include <iostream>
4  #include <string_view>
5
6  void* operator new(std::size_t count){
7      std::cout<<"分配了堆内存"<<count<<"字节."<<std::endl;
8      return malloc(count);
9  }
10
11 void operator delete(void* p){
12     std::cout<<"释放堆内存:"<<p<<std::endl;
13     free(p);
14 }
```

知乎 @铁甲万能狗

show_str()函数是用于打印传入参数string对象**str内部的字符串**的地址和函数内部的局部变量的string对象tmp的内部字符串的地址。

```
17 void show_str(const std::string &str){
18     std::cout<<std::endl;
19     std::cout<<"show_str()临时变量tmp初始化"<<std::endl;
20     std::string tmp=str;
21
22     printf("str副本的地址:%p\n",str.c_str());
23     printf("tmp副本的地址:%p\n",tmp.c_str());
24 }
```

知乎 @铁甲万能狗

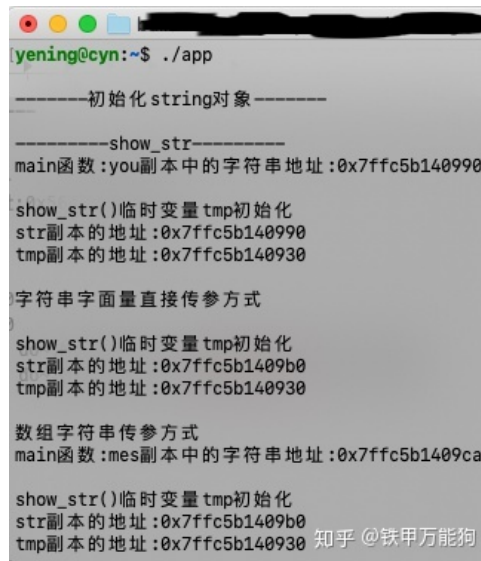
下面是调用函数

```
36
37     std::string you="Hello,World!!";
38     std::cout<<std::endl;
39
40     std::cout<<"-----show_str-----"<<std::endl;
41     printf("main函数:you副本中的字符串地址:%p\n",you.c_str());
42     show_str(you);
43
44     std::cout<<std::endl;
45     std::cout<<"字符串字面量直接传参方式"<<std::endl;
46     show_str("Hello,World!!");
47
48     std::cout<<std::endl;
49     std::cout<<"数组字符串传参方式"<<std::endl;
50     const char mes[]="Hello,World!!";
51     printf("main函数:mes副本中的字符串地址:%p\n",&mes[0]);
52     show_str(mes);
53
54     std::cout<<std::endl;
55
56     return 0;
57 }
```

知乎 @铁甲万能狗

输出结果:

首先继续进行下文之前,需要说明的是Linux下的x86_64版本的GCC/G++编译器默认情况下(编译时没有附带 -O 优化选项),仍然按照x86平台的过程调用约定组织程序栈,下文编译时使用的是默认设置。



```
yening@cyn:~$ ./app
-----初始化string对象-----
-----show_str-----
main函数:you副本中的字符串地址:0x7ffc5b140990
show_str()临时变量tmp初始化
str副本的地址:0x7ffc5b140990
tmp副本的地址:0x7ffc5b140930
字符串字面量直接传参方式
show_str()临时变量tmp初始化
str副本的地址:0x7ffc5b1409b0
tmp副本的地址:0x7ffc5b140930
数组字符串传参方式
main函数:mes副本中的字符串地址:0x7ffc5b1409ca
show_str()临时变量tmp初始化
str副本的地址:0x7ffc5b1409b0
tmp副本的地址:0x7ffc5b140930
```

知乎 @铁甲万能狗

从上面程序输出看来,在每次调用show_str()函数输出的内存地址看来,string对象内部持有字符串副本的内存分配都发生在程序栈帧中,有一些有趣的分析。

- main函数我们知道string对象内部持有字符串副本的地址是"0x7ffc5b140990",输出的参数地址跟main函数中的变量you是一致的,因为我们show_str()的参数类型是const string&即使用了引用传参,我们这里避免了字符串的拷贝。
- 每次string类型的局部变量赋值操作,string对象内部自动执行字符串拷贝,从每次打印的tmp程序地址可以得知。

匿名字符串字面量

我们第二次调用show_str()函数时,你们是否思考过如下两个问题。

1. 0x7ffc5b1409b0从那里冒出来的,为何跟main函数的you不是一致的?
2. 我们又没有定义新的string类型的局部变量,0x7ffc5b1409b0这个地址为什么后面会出现了两次?

类型的局部变量,只是直接传入一个字符串字面量。关键就是在这里,当我们直接向show_str传入一个字符串字面量之前,C++编译器会隐式创建一个临时变量,我们假设变量的名称是任意的x。隐式的临时变量它的内部字符串副本的地址自然就指向0x7ffc5b1409b0这个地址,我们第二次调用show_str的代码,即如下代码所示

```
int main(void){
    std::string you="Hello,World!!";
    show_str(you);
    .....

    //show_str("Hello,World!!")会等价于如下代码
    std::string& x="Hello,World!!";//隐式创建
    show_str(x);
    ....
}
```

接下来回答第二个问题就非常简单,由于C++已经隐式地定义了

```
std::string& x="Hello,World!!";
```

那么后续调用任意的被调用函数的传参类型只要是const string&,那么传入同一个匿名的字符串字面量。自然打印的都是同一个隐式局部变量的内部字符串副本的地址。

另外比较蹊跷的是tmp每次调用show_str输出的地址是相同的,因为我们这里陆续调用了相同show_str函数,那么show_str栈帧结构基本上一样的,如果你调用不同尺寸的函数,输出结果就会不一样。

堆中的string的内存分配

这次,我稍微做一下改动,现在我们在main中传入一个比之前**更长**的尺寸为33字节的字符串字面量,如下图

```
27 int main(void){
28     std::cout<<std::endl;
29     std::cout<<"-----初始化string对象-----"<<std::endl;
30     std::string you="How do you do~,My name is peter!";
31     std::cout<<std::endl;
32
33
34     std::cout<<"-----show_str-----"<<std::endl;
35     printf("main函数:你副本中的字符串地址:%p\n",you.c_str());
36     show_str(you);
37
38     std::cout<<std::endl;
39     std::cout<<"字符串字面量直接传参方式"<<std::endl;
40     show_str("How do you do~,My name is peter!");
41
42     std::cout<<"数组字符串传参方式"<<std::endl;
43     const char mes[]="How do you do~,My name is peter!";
44     show_str(mes);
45
46     std::cout<<std::endl;
47
48     return 0;
49 }
```

知乎 @铁甲万能狗

对应的输出

```

-----show_str-----
main函数:you副本中的字符串地址:0x55c9a62c9280

show_str()临时变量tmp初始化
分配了堆内存33字节.
str副本的地址:0x55c9a62c9280
tmp副本的地址:0x55c9a62c92b0
释放堆内存:0x55c9a62c92b0

字符串字面量直接传参方式
分配了堆内存33字节.

show_str()临时变量tmp初始化
分配了堆内存33字节.
str副本的地址:0x55c9a62c92b0
tmp副本的地址:0x55c9a62c92e0
释放堆内存:0x55c9a62c92e0
释放堆内存:0x55c9a62c92b0
数组字符串传参方式
分配了堆内存33字节.

show_str()临时变量tmp初始化
分配了堆内存33字节.
str副本的地址:0x55c9a62c92b0
tmp副本的地址:0x55c9a62c92e0
释放堆内存:0x55c9a62c92e0
释放堆内存:0x55c9a62c92b0
释放堆内存:0x55c9a62c9280
    
```

知乎 @铁甲万能狗

这次string对象的内存分配已经发生变化,show_str()函数中的他们的内部数据成员分别指向各自堆中分配的内存块,的字符副本分别存储这些堆中的内存块。如上图输出都分别调用了void* operator new(size_t)的重载版本。

到这里你就应该要思考两个问题

- 为什么在处理 “Hello,Word!!” 只在栈中进行内存分配?
- 为什么在处理 “Hello,My name is peter!!” 这样的字符串,就会在堆中进行内存分配?

没错,答案就是字符串字面量的长度决定的。这个我在前一编《对[C/C++]指针与字符串的总结》已经提到过,但当时我没有指出,触发string对象内部的新操作的准确阈值是多少。请看如下表

编译器名称	初次堆内存分配条件
MS Visual C++	大于15字节
GCC	大于15字节
Clang	大于23字节

string对象内部约定:

- 只要传入的字符串字面量小于上表的阈值,string内部实现在栈中分配内存,有个很骚的名字**小型字符串优化**(Small String Optimisation)。
- 只要大于上述C++编译器指定阈值,string对象内部会隐式执行new操作在堆中根据指定的字符串尺寸分配**初次内存**。
- 如果后续任何字符串的push_back操作,string会根据 “double方案” 的内存分配方式对堆内存执行扩容操作,见前文《对[C/C++]指针与字符串的总结》。
- 还有根据RAII的约定,C++编译器会对string对象在其调用函数的生命周期结束之时自动执行垃圾回收。(见上图的输出)。

建议:到这里,如果还没搞懂如下代码背后的内存含义的话,建议还是去补补栈和堆内存管理的知识,再去深入了解string对象。这样会让你少走很多弯路。

```
string s=new string(...)
```

和



}

我们从内存地址的角度，分析了string对象在栈中和堆中的内存分配细节。从这篇文章你应该知道，在C++中掌握内存分析方法是多么地重要，本篇用到了以前我所写随笔的程序栈和堆内存管理的知识。

扩展阅读，如果关注我的读者应该了解我写软文的套路是一环扣一环的，可能在说string的话题，然后有跳到程序栈，这就是所谓的知识碎片整理。

- [《第2篇:C/C++ 内存布局与程序栈》](#)
- [《第3篇：戏说程序栈-call指令和ret指令》](#)
- [《第4篇：戏说程序栈-栈帧》](#)
- [《第5篇-戏说程序栈-寄存器和函数状态》](#)
- [《第6篇-戏说程序栈 x86_64过程调用》](#)

后记

了解string对象的行为之后,接下来我们如何考虑使用什么方法来避免字符串频繁的拷贝,有些经验的“老油条”应该都领略过了const string&这类参数类型声明并不能从根本上解决问题（上例子的程序输出已经隐藏地说明了这一点）。于是C++17就有了string_view这个标准库的扩展，这个扩展极大地解决了string拷贝的空间成本和时间成本问题。我们后续文章会继续新的话题。

发布于 2020-08-17 03:01

C++ string

▲ 赞同 12 ▼ ● 添加评论 🔗 分享 ❤ 喜欢 ★ 收藏 📄 申请转载 ...

文章被以下专栏收录



C/C++内存管理

侧重C/C++内存模型，反编译分析

推荐阅读

C++ string类（学习笔记：第6章 23）

string类[1]使用字符串类string表示字符串（string类是C++标准库中一个封装起来的字符串）string实际上是对字符数组操作的封装（string类是C++标准库

品颜完月

发表于开发积累



C++:为何不建议用string作为函数参数

黄裕玲

发表于C语言程序...

C++11 的 to_string

看到用 Java 的朋友 “string” 可以用 toStr 我这学 C++ 的顿觉惆怅我大 C++ 没有这么好法，直到昨天我才在网原来 C++11 中已经有

可乐船长2...

发表

还没有评论

