标准库中string的底层实现方式

我们都知道， std::string的一些基本功能和用法了，但它底层到底是如何实现的呢? 其实在std::string的历史中，出现过几种不同的方式。下面我们来一一揭晓。

我们可以从一个简单的问题来探索，一个std::string对象占据的内存空间有多大，即sizeof(std::string)的值为多大？如果我们在不同的编译器（VC++, GNU, Clang++）上去测试，可能会发现其值并不相同；即使是GNU，不同的版本，获取的值也是不同的。

虽然历史上的实现有多种，但基本上有三种方式：

- Eager Copy(深拷贝)
- COW（Copy-On-Write 写时复制）
- SSO(Short String Optimization-短字符串优化)

每种实现，std::string都包含了下面的信息：

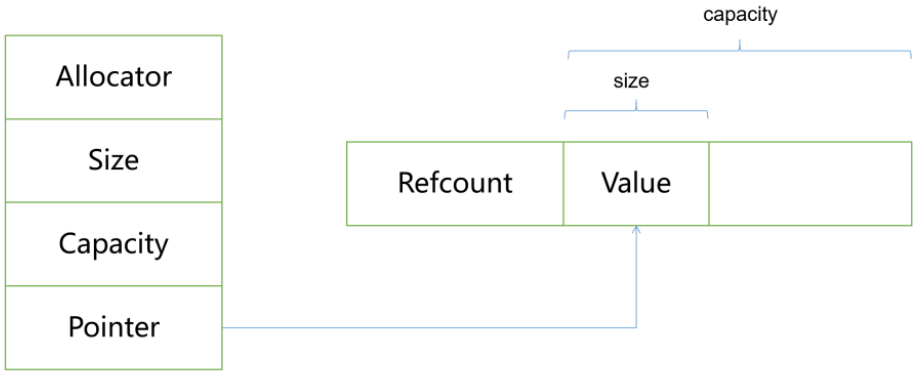
- 字符串的大小
- 能够容纳的字符数量
- 字符串内容本身

1、Eager Copy

最简单的就是深拷贝了。无论什么情况，都是采用拷贝字符串内容的方式解决，这也是我们之前已经实现过的方式。这种实现方式，在需要对字符串进行频繁复制而又并不改变字符串内容时，效率比较低下。所以需要对其实现进行优化，之后便出现了下面的COW的实现方式。

2、COW(Copy-On-Write)

当两个std::string发生复制构造或者赋值时，不会复制字符串内容，而是增加一个引用计数，然后字符串指针进行浅拷贝，其执行效率为O(1)。只有当需要修改其中一个字符串内容时，才执行真正的复制。其实现的示意图，有下面两种形式：

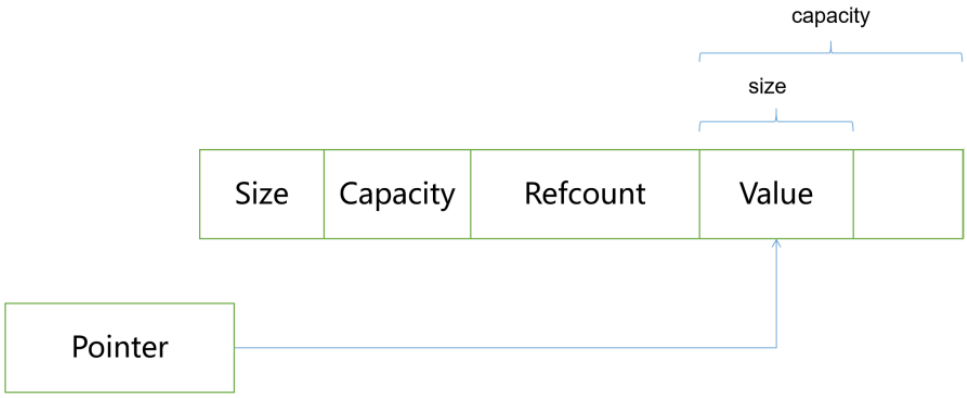


实现1

std::string的数据成员就是:

```
class string
{
private:
    Allocator _allocator;
    size_t size;
    size_t capacity;
    char * pointer;
};
```

第二种形式为：



实现2

std::string的数据成员就只有一个了：

```
class string
{
private:
    char * _pointer;
};
```

为了实现的简单，在GNU4.8.4的中，采用的是实现2的形式。从上面的实现，我们看到引用计数并没有与std::string的数据成员放在一起，为什么呢？大家可以思考一下。

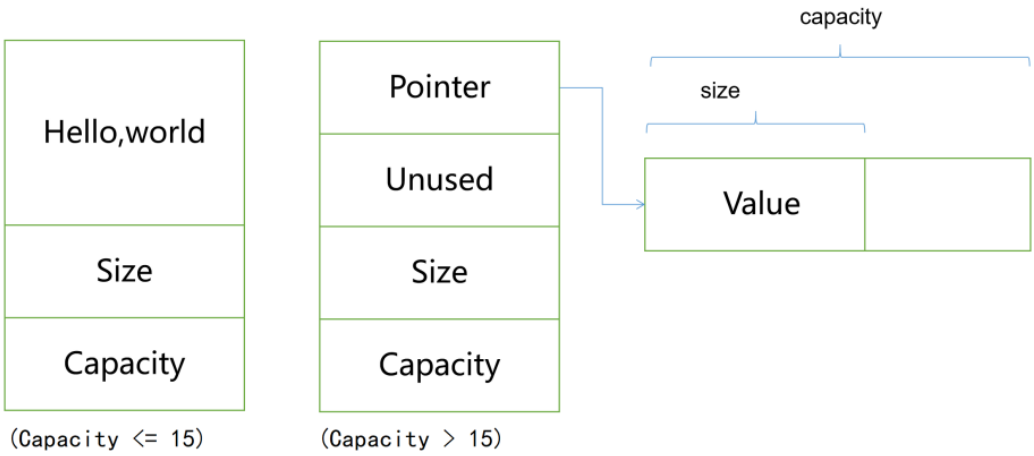
当执行复制构造或赋值时，引用计数加1，std::string对象共享字符串内容；当std::string对象销毁时，并不直接释放字符串所在的空间，而是先将引用计数减1，直到引用计数为0时，则真正释放字符串内容所在的空间。根据这个思路，大家可以自己动手实现一下。

大家再思考一下，既然涉及到了引用计数，那么在多线程环境下，涉及到修改引用计数的操作，是否是线程安全的呢？为了解决这个问题，GNU4.8.4的实现中，采用了原子操作。

3、SSO（Short String Optimization）

目前，在VC++、GNU5.x.x以上、Clang++上，std::string实现均采用了SSO的实现。

通常来说，一个程序里用到的字符串大部分都很短小，而在64位机器上，一个char*指针就占用了8个字节，所以SSO就出现了，其核心思想是：发生拷贝时要复制一个指针，对小字符串来说，为啥不直接复制整个字符串呢，说不定还没有复制一个指针的代价大。其实现示意图如下：



实现3

std::string的数据成员就是：

```
class string
{
    union Buffer
    {
        char * _pointer;
        char _local[16];
    };

    Buffer _buffer;
    size_t _size;
    size_t _capacity;
};
```

当字符串的长度小于等于15个字节时，buffer直接存放整个字符串；当字符串大于15个字节时，buffer存放的就是一个指针，指向堆空间的区域。这样做的好处是，当字符串较小时，直接拷贝字符串，放在string内部，不用获取堆空间，开销小。

4、最佳策略

以上三种方式，都不能解决所有可能遇到的字符串的情况，各有所长，又各有缺陷。综合考虑所有情况之后，facebook开源的folly库中，实现了一个fbstring, 它根据字符串的不同长度使用不同的拷贝策略，最终每个fbstring对象占据的空间大小都是24字节。

1. 很短的（0~22）字符串用SSO，23字节表示字符串（包括'\0'）,1字节表示长度
2. 中等长度的（23~255）字符串用eager copy，8字节字符串指针，8字节size，8字节capacity.
3. 很长的(大于255)字符串用COW, 8字节指针（字符串和引用计数），8字节size，8字节capacity.

5、线程安全性

两个线程同时对同一个字符串进行操作的话, 是不可能线程安全的, 出于性能考虑, C++并没有为string实现线程安全, 毕竟不是所有程序都要用到多线程。

但是两个线程同时对独立的两个string操作时, 必须是安全的. COW技术实现这一点是通过原子的对引用计数进行+1或-1操作。

CPU的**原子操作**虽然比mutex锁好多了，但是仍然会带来性能损失, 原因如下:

- 阻止了CPU的乱性执行.
- 两个CPU对同一个地址进行原子操作, 会导致cache失效, 从而重新从内存中读数据.
- 系统通常会lock住比目标地址更大的一片区域，影响逻辑上不相关的地址访问

这也是在多核时代，各大编译器厂商都选择了SS0实现的原因吧。