

吾尝终日而思矣，不如须臾之所学也。吾尝跂而望矣，不如登高之博见也。……君子生非异也，善假于物也。

## 陈硕的 Blog

### 为什么多线程读写 shared\_ptr 要加锁?

陈硕 (giantchen\_AT\_gmail\_DOT\_com)

2012-01-28

我在《Linux 多线程服务端编程：使用 muduo C++ 网络库》第 1.9 节“再论 shared\_ptr 的线程安全”中写道：

(shared\_ptr) 的引用计数本身是安全且无锁的，但对象的读写则不是，因为 shared\_ptr 有两个数据成员，读写操作不能原子化。根据文档

([http://www.boost.org/doc/libs/release/libs/smart\\_ptr/shared\\_ptr.htm#ThreadSafety](http://www.boost.org/doc/libs/release/libs/smart_ptr/shared_ptr.htm#ThreadSafety))，shared\_ptr 的线程安全级别和内建类型、标准库容器、std::string 一样，即：

- 一个 shared\_ptr 对象实体可被多个线程同时读取（文档例1）；
- 两个 shared\_ptr 对象实体可以被两个线程同时写入（例2），“析构”算写操作；
- 如果要从多个线程读写同一个 shared\_ptr 对象，那么需要加锁（例3~5）。

请注意，以上是 shared\_ptr 对象本身的线程安全级别，不是它管理的对象的线程安全级别。


后文 (p.18) 则介绍如何高效地加锁解锁。本文则具体分析一下为什么“因为 shared\_ptr 有两个数据成员，读写操作不能原子化”使得多线程读写同一个 shared\_ptr 对象需要加锁。这个在我看来显而易见的结论似乎也有人抱有疑问，那将导致灾难性的后果，值得我写这篇文章。本文以 boost::shared\_ptr 为例，与 std::shared\_ptr 可能略有区别。

## shared\_ptr 的数据结构

shared\_ptr 是引用计数型 (reference counting) 智能指针，几乎所有的实现都采用在堆 (heap) 上放个计数值 (count) 的办法（除此之外理论上还有用循环链表的办法，不过没有实例）。具体来说，shared\_ptr<Foo> 包含两个成员，一个是指向 Foo 的指针 ptr，另一个是 ref\_count 指针（其类型不一定是原始指针，有可能是 class 类型，但不影响这里的讨论），指向堆上的 ref\_count 对象。ref\_count 对象有多个成员，具体的数据结构如图 1 所示，其中 deleter 和 allocator 是可选的。

2013年1月						
日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

### 导航

博客园  
首页  
新随笔  
联系  
订阅   
管理

### 统计

随笔 - 73  
文章 - 0  
评论 - 413  
引用 - 0

### 公告

本人博客的文章均为原创作品，除非另有声明。个人转载或引用时请保留本人的署名及博客网址，商业转载请事先联系，我的 gmail 用户名是 giantchen。

昵称： 陈硕  
园龄： 10年9个月  
粉丝： 1321  
关注： 0  
[+加关注](#)

### 搜索

### 常用链接

我的随笔  
我的评论  
我的参与  
最新评论  
我的标签

### 随笔分类 (67)

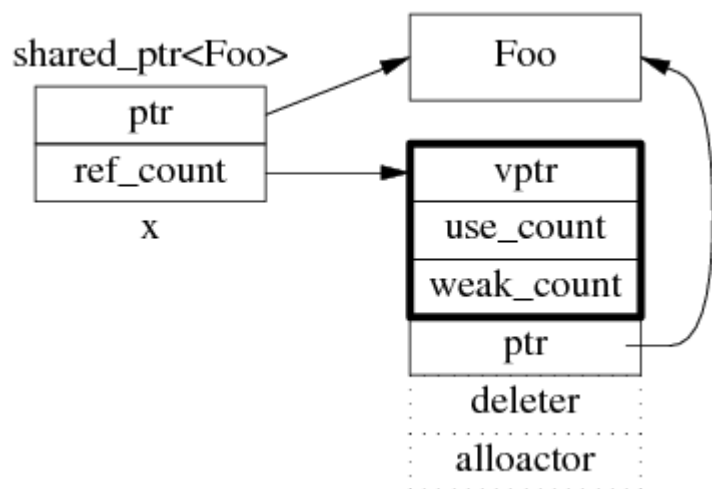
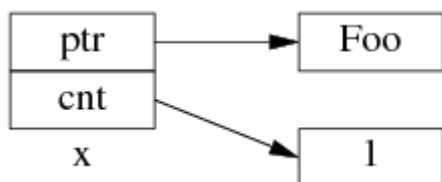


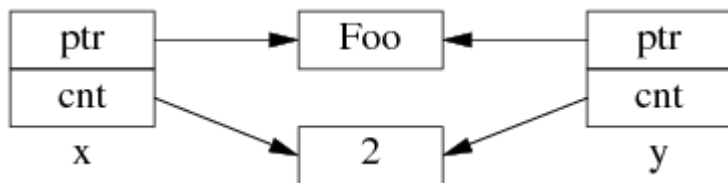
图 1: shared\_ptr 的数据结构。

为了简化并突出重点，后文只画出 use\_count 的值：



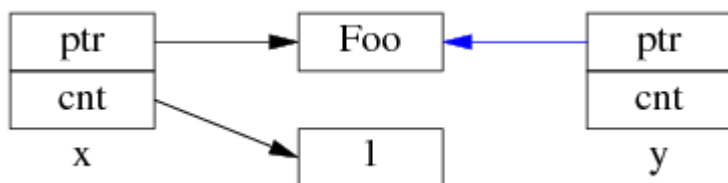
以上是 shared\_ptr<Foo> x(new Foo); 对应的内存数据结构。

如果再执行 shared\_ptr<Foo> y = x; 那么对应的数据结构如下。

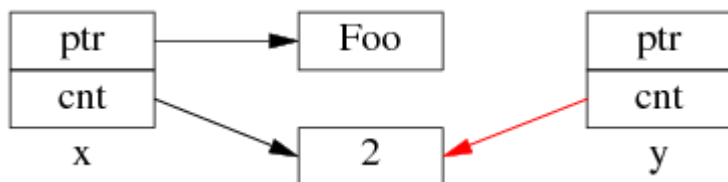


但是 y=x 涉及两个成员的复制，这两步拷贝不会同时（原子）发生。

中间步骤 1，复制 ptr 指针：



中间步骤 2，复制 ref\_count 指针，导致引用计数加 1：



步骤1和步骤2的先后顺序跟实现相关（因此步骤 2 里没有画出 y.ptr 的指向），我见过的都是先1后2。

C++ 工程实践(19)  
muduo(27)  
多线程(12)  
分布式系统(9)

#### 随笔档案 (73)

2014年12月(1)  
2014年5月(1)  
2013年11月(1)  
2013年10月(2)  
2013年9月(1)  
2013年8月(3)  
2013年7月(1)  
2013年2月(1)  
2013年1月(4)  
2012年12月(1)  
2012年9月(1)  
2012年7月(2)  
2012年6月(1)  
2012年4月(2)  
2012年3月(1)  
2011年8月(2)  
2011年7月(2)  
2011年6月(3)  
2011年5月(6)  
2011年4月(7)  
2011年3月(5)  
2011年2月(9)  
2010年10月(1)  
2010年9月(4)  
2010年8月(3)  
2010年5月(1)  
2010年4月(1)  
2010年3月(2)  
2010年2月(4)

#### 最新评论

1. Re:“过家家”版的移动离线计费系统实现

记得有个设计模式可以解决鸳鸯咖啡，好像是装饰器模式，《Head First 设计模式》里有。意思是不要用继承应该用组合，比如一个饮料类，里边有成员咖啡、茶等。这样就可以防止类爆炸。...

--01hack

2. Re:C++ 工程实践(7):  
iostream 的用途与局限

写的很好！这个说一下：#include <stdio.h> int main() { const int

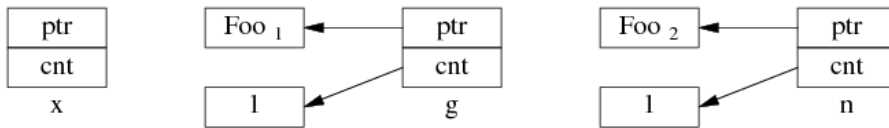
既然  $y=x$  有两个步骤, 如果没有 mutex 保护, 那么在多线程里就有 race condition.

## 多线程无保护读写 shared\_ptr 可能出现的 race condition

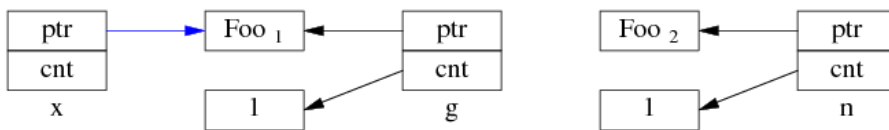
考虑一个简单的场景, 有 3 个 `shared_ptr<Foo>` 对象 `x`、`g`、`n`:

- `shared_ptr<Foo> g(new Foo);` // 线程之间共享的 `shared_ptr`
- `shared_ptr<Foo> x;` // 线程 A 的局部变量
- `shared_ptr<Foo> n(new Foo);` // 线程 B 的局部变量

一开始, 各安其事。

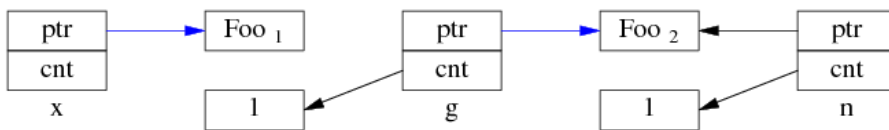


线程 A 执行 `x = g;` (即 read `g`), 以下完成了步骤 1, 还没来得及执行步骤 2。这时切换到了 B 线程。

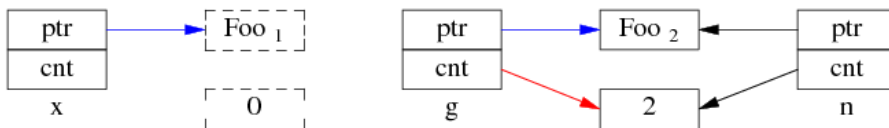


同时线程 B 执行 `g = n;` (即 write `g`), 两个步骤一起完成了。

先是步骤 1:

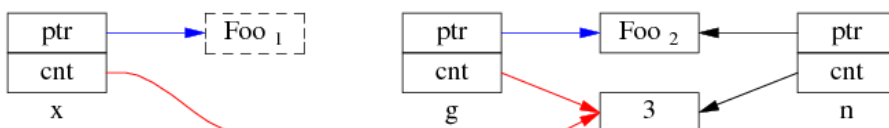


再是步骤 2:



这是 `Foo1` 对象已经销毁, `x.ptr` 成了空悬指针!

最后回到线程 A, 完成步骤 2:



`max = 80; char name[max];`  
`char fmt[10]; sprintf...`

--xxeray

3. Re:C++ 工程实践(8): 值语义

页面字体与背景很舒服

--设计与艺术

4. Re:从《C++ Primer 第四版》入手学习 C++

陈大师没有继续在博客园上更新博客了啊。

--IclodQ

5. Re:多线程服务器的常用编程模型厉害了

--素描描青眉

6. Re:谈一谈网络编程学习经验 (06-08更新)

楼主已经达到独孤求败的境界了。

--孤火

7. Re:Muduo 网络编程示例之八: 用 Timing wheel 踢掉空闲连接

有点意思, 不错不错~

--oyld

8. Re:一种自动反射消息类型的 Google Protobuf 网络传输方案

@ zjx20引用其实跟山寨做法没有本质区别, 最后免不了一个switch做各种处理。除非再为每种类型bind一个 handle function而且有个问题是 如果包的类型比较多 而处理又比较频繁这个s...

--Visual C++

9. Re:分布式系统部署、监控与进程管理的几重境界

我们公司现在正好需要这么一个东西, 并且打算着手开发。这篇文章很给力啊, 让思路清晰了好多。

--孤帆

10. Re:从《C++ Primer 第四版》入手学习 C++

顶

--Giser-阿飞

### 阅读排行榜

1. 一种自动反射消息类型的 Google Protobuf 网络传输方案(37619)
2. 关于 TCP 并发连接的几个思考题与试验(27746)
3. C++ 工程实践(7): iostream 的用途与局限(22208)
4. 多线程服务器的常用编程模型(19847)

多线程无保护地读写 g, 造成了“x 是空悬指针”的后果。这正是多线程读写同一个 shared\_ptr 必须加锁的原因。

当然, race condition 远不止这一种, 其他线程交织 (interweaving) 有可能会造成其他错误。

思考, 假如 shared\_ptr 的 operator= 实现是先复制 ref\_count (步骤 2) 再复制 ptr (步骤 1), 会有哪些 race condition?

## 杂项

### shared\_ptr 作为 unordered\_map 的 key

如果把 boost::shared\_ptr 放到 unordered\_set 中, 或者用于 unordered\_map 的 key, 那么要小心 hash table 退化为链表。  
<http://stackoverflow.com/questions/6404765/c-shared-ptr-as-unordered-sets-key/12122314#12122314>

直到 Boost 1.47.0 发布之前, unordered\_set<std::shared\_ptr<T>> 虽然可以编译通过, 但是其 hash\_value 是 shared\_ptr 隐式转换为 bool 的结果。也就是说, 如果不自定义 hash 函数, 那么 unordered\_{set/map} 会退化为链表。<https://svn.boost.org/trac/boost/ticket/5216>

Boost 1.51 在 boost/functional/hash/extensions.hpp 中增加了有关重载, 现在只要包含这个头文件就能安全高效地使用 unordered\_set<std::shared\_ptr> 了。

这也是 muduo 的 examples/idleconnection 示例要自己定义 hash\_value(const boost::shared\_ptr<T>& x) 函数的原因 (书第 7.10.2 节, p.255)。因为 Debian 6 Squeeze、Ubuntu 10.04 LTS 里的 boost 版本都有这个 bug。

### 为什么图 1 中的 ref\_count 也有指向 Foo 的指针?

shared\_ptr<Foo> sp(new Foo) 在构造 sp 的时候捕获了 Foo 的析构行为。实际上 shared\_ptr.ptr 和 ref\_count.ptr 可以是不同的类型 (只要它们之间存在隐式转换), 这是 shared\_ptr 的一大功能。分 3 点来说:

**1. 无需虚析构;** 假设 Bar 是 Foo 的基类, 但是 Bar 和 Foo 都没有虚析构。

shared\_ptr<Foo> sp1(new Foo); // ref\_count.ptr 的类型是 Foo\*

5. 从《C++ Primer 第四版》入手学习 C++(19648)
6. 分布式系统的工程化开发方法(19124)
7. 谈一谈网络编程学习经验 (06-08 更新) (17365)
8. 当析构函数遇到多线程 —— C++ 多线程安全的对象回调(14192)
9. 学之者生, 用之者死——ACE历史与简评(12193)
10. C++ 工程实践(8): 值语义(11378)

#### 评论排行榜

1. 计算机图书赠送(40)
2. 从《C++ Primer 第四版》入手学习 C++(22)
3. 关于 TCP 并发连接的几个思考题与试验(20)
4. 学之者生, 用之者死——ACE历史与简评(20)
5. 谈一谈网络编程学习经验 (06-08 更新) (17)
6. 一种自动反射消息类型的 Google Protobuf 网络传输方案(17)
7. 发布一个基于 Reactor 模式的 C++ 网络库(17)
8. 并发编程的 15 条建议(译)(16)
9. 多线程服务器的常用编程模型(14)
10. 《Linux多线程服务端编程: 使用muduo C++网络库》上市半年重印两次, 总印数达到了9000册(12)

#### 推荐排行榜

1. 谈一谈网络编程学习经验 (06-08 更新) (30)
2. 关于 TCP 并发连接的几个思考题与试验(18)
3. 从《C++ Primer 第四版》入手学习 C++(16)
4. 分布式系统的工程化开发方法(10)
5. 计算机图书赠送(10)
6. 关于 std::set/std::map 的几个为什么(9)
7. C++ 工程实践(7): iostream 的用途与局限(9)
8. 分布式系统部署、监控与进程管理的几重境界(9)

```
shared_ptr<Bar> sp2 = sp1; // 可以赋值, 自动向上转型 (up-cast)
```

```
sp1.reset(); // 这时 Foo 对象的引用计数降为 1
```

此后 sp2 仍然能安全地管理 Foo 对象的生命期, 并安全完整地释放 Foo, 因为其 ref\_count 记住了 Foo 的实际类型。

**2. shared\_ptr<void>** 可以指向并安全地管理 (析构或防止析构) 任何对象; muduo::net::Channel class 的 tie() 函数就使用了这一特性, 防止对象过早析构, 见书 7.15.3 节。

```
shared_ptr<Foo> sp1(new Foo); // ref_count.ptr 的类型是 Foo*
```

```
shared_ptr<void> sp2 = sp1; // 可以赋值, Foo* 向 void* 自动转型
```

```
sp1.reset(); // 这时 Foo 对象的引用计数降为 1
```

此后 sp2 仍然能安全地管理 Foo 对象的生命期, 并安全完整地释放 Foo, 不会出现 delete void\* 的情况, 因为 delete 的是 ref\_count.ptr, 不是 sp2.ptr。

**3. 多继承。**假设 Bar 是 Foo 的多个基类之一, 那么:

```
shared_ptr<Foo> sp1(new Foo);
```

```
shared_ptr<Bar> sp2 = sp1; // 这时 sp1.ptr 和 sp2.ptr 可能指向不同的地址, 因为 Bar subobject 在 Foo object 中的 offset 可能不为0。
```

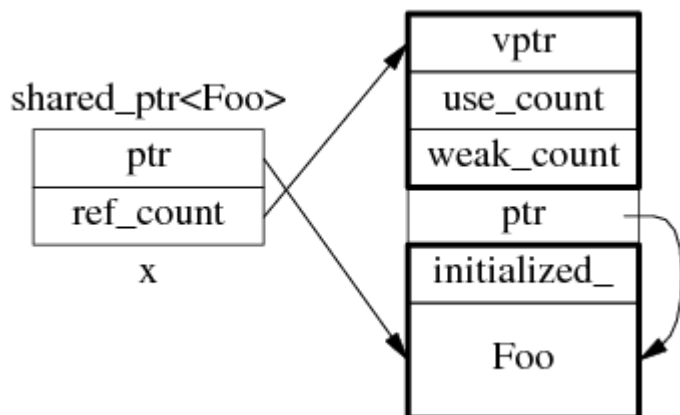
```
sp1.reset(); // 此时 Foo 对象的引用计数降为 1
```

但是 sp2 仍然能安全地管理 Foo 对象的生命期, 并安全完整地释放 Foo, 因为 delete 的不是 Bar\*, 而是原来的 Foo\*。换句话说, sp2.ptr 和 ref\_count.ptr 可能具有不同的值 (当然它们的类型也不同)。

## 为什么要尽量使用 make\_shared()?

为了节省一次内存分配, 原来 shared\_ptr<Foo> x(new Foo); 需要为 Foo 和 ref\_count 各分配一次内存, 现在用 make\_shared() 的话, 可以一次分配一块足够大的内存, 供 Foo 和 ref\_count 对象容身。数据结构是:





不过 Foo 的构造函数参数要传给 make\_shared(), 后者再传给 Foo::Foo(), 这只有在 C++11 里通过 perfect forwarding 才能完美解决。

(.完.)

分类: [C++](#) [工程实践](#)

好文要顶

关注我

收藏该文



陈硕

关注 - 0

粉丝 - 1321

6

0

+加关注

« 上一篇: [关于 std::set/std::map 的几个为什么](#)

» 下一篇: [用 LaTeX 排版编程技术书籍的一些个人经验](#)

posted on 2013-01-28 05:17 [陈硕](#) 阅读(8352) 评论(6) [编辑](#) [收藏](#)

## 评论

**#1楼 2013-01-28 10:31 烟影**

陈大哥的文章一向很清晰! 很好,学习了

[支持\(0\)](#) [反对\(0\)](#)

**#2楼 2013-01-28 12:08 法克给木**

因为引用计数值的维护不是线程安全的。我不知道为何要有这个问题? 难道不应该加锁吗? 还是为了阐述shared\_ptr内部实际上是没有加锁的?

[支持\(0\)](#) [反对\(0\)](#)

**#3楼 [楼主] 2013-01-28 12:51 陈硕**

@ 法克给木

shared\_ptr 引用计数值的维护是线程安全的。

因此多线程并发读同一个 shared\_ptr 就不必加锁。

例如 g 是共享的, x = g 和 y = g 可以并行执行 (读同一个 g), 无

须加锁。

只有一读一写，或者并发写才需要加锁。

支持(0) 反对(0)

#### #4楼 2013-01-28 13:38 法克给木

嗚，认真的看了一下内容，受教了。一直用的是chromium的scoped\_refptr，觉得非常贴切。之前也用过boost的shared\_ptr，但是不深入，也没意识到这些问题。chromium的scoped\_refptr设计中要求shared\_ptr的两个数据成员一体，也就是对被包装的类设计有要求，这种协议限制了准入门槛，却规避了使用过程中隐晦问题的出现，具体开发中我赞成后者。

支持(0) 反对(0)

#### #5楼 2013-01-28 22:50 边城浪

没注意到.还有这种问题.

使用的时候还是要注意一下.

尽量做到对 shared\_ptr对象一次赋值.

支持(0) 反对(0)

#### #6楼 2013-11-10 19:54 newzai

```
x = g;
```

```
g = n;
```

这个问题貌似和shared\_ptr关系不大吧。。

就算 g, x, n是基础类型 int,

发生在多线程的环境里面，不不能保证是正确的。。

```
int g = 1; //全局
```

```
int x; // A 线程局部变量
```

```
int n = 2; //B线程局部变量
```

2个线程同时执行 x=g;和 g =n; x得到的值是1，还是2呢？也是不确定的。。

所以，我觉得在这里shared\_ptr很冤枉。。

支持(1) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

**登录后才能发表评论，立即 [登录](#) 或 [注册](#)， [访问](#) 网站首页**

**博客园派送云上免费午餐，AWS注册立享12个月免费套餐**

**【推荐】News: 大型组态、工控、仿真、CADGIS 50万行VC++源码免费下载**

**【推荐】博客园 & 陌上花开HIMMR 给单身的程序员小哥哥助力脱单啦~**

**【推荐】网络安全攻防第1课：攻防靶场、Web渗透、漏洞利用**

**【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区**

【推荐】未知数的距离，毫秒间的传递，声网与你实时互动

【福利】AWS携手博客园为开发者送免费套餐与抵扣券

【推荐】阿里云折扣价格返场，错过再等一年

#### 相关博文：

- [为什么多线程读写shared\\_ptr要加锁?](#)
- [为什么多线程读写 shared\\_ptr 要加锁?](#)
- [shared\\_ptr智能指针源码剖析](#)
- [多线程读写shared\\_ptrshared\\_ptr要加锁分析! 学习笔记](#)
- [shared\\_ptr的线程安全性](#)
- » [更多推荐...](#)



#### 最新 IT 新闻：

- [在线少儿编程平台编程猫完成 13 亿元 D 轮融资](#)
- [飞书第一次发布会，尝试解答 B 端产品的几个难题](#)
- [在线教育红海，夸克如何破题“自主学习”?](#)
- [质量问题多，美知名杂志不再推荐特斯拉Model S](#)
- [绕过苹果！谷歌云游戏将以网页应用形式登陆iOS](#)
- » [更多新闻...](#)

---

Powered by:

[博客园](#)

Copyright © 2020 陈硕

Powered by .NET 5.0.0 on Kubernetes