


C++11智能指针（一）：shared_ptr介绍与实例

 c/c++ 同时被 2 个专栏收录 ▾

19 订阅 54 篇文章

订阅专栏

什么是std::shared_ptr<>？

std::shared_ptr<>是c++11中引入的一种智能指针，它足够聪明，如果指针不在任何地方使用，就会自动删除指针。这可以帮助我们彻底消除内存泄露和悬挂指针的问题。

shared_ptr和共享所有权

它遵循共享所有权的概念，即不同的shared_ptr对象可以与相同的指针相关联，并且在内部使用引用计数机制来实现。

每个shared_ptr对象内部指向两块内存区域

- 1)指向对象
- 2)指向用于引用计数的控制数据

共享所有权怎样在引用计数的帮助下工作

- 当一个新的shared_ptr对象与一个指针相关联时，在它的构造函数中，它将与这个指针相关的引用计数增加1.
 - 当任何shared_ptr对象超出作用域时，则在其析构函数中将相关指针的引用计数递减1.
- 当引用计数变为0时，意味着没有任何shared_ptr对象与这块内存关联，在这种情况下，它使用“删除”功能删除这块内存

创建shared_ptr对象

当我们将shared_ptr对象与原始指针绑定时，

```
std::shared_ptr<int> p1(new int());
```

它在堆上分配两块内存：

- 1)为int分配的内存
- 2)用于引用计数的内存，将用于管理与此内存相关的shared_ptr对象的技术，初始值为1。

检查shared_ptr对象的引用计数

```
p1.use_count();
```

std::make_shared<T>

如何分配指向shared_ptr的指针

```
1 //Compile Error
2 std::shared_ptr<int> p1 = new int(); //Compile error
```

因为采用参数的shared_ptr构造函数是Explicit，并且在上面的行中，我们正在隐含地调用它。创建shared_ptr对象的最好方法是使用 **std::make_shared**,

```
std::shared_ptr<int> p1 = std::make_shared<int>();
```

std::make_shared为引用计数所需的对象和数据结构做了一次内存分配，即新运算符只会被调用一次。

分离关联的原始指针

可以调用reset()方法使std::shared_ptr对象解除连接它的指针

无参数的reset()函数

```
p1.reset();
```

它将引用计数减1，如果引用计数变为0，则删除指针

有参数的reset()函数

```
p1.reset(new int(34));
```

在这种情况下，它将在内部指向新的指针，因此其引用计数将再次变为1。

使用nullptr重置

```
p1 = nullptr;
```

shared_ptr是一个伪指针

shared_ptr作为普通的指针，即我们可以在shared_ptr对象上使用*和->，也可以像其他shared_ptr对象那样进行比较;

完整的例子：

```
1 #include <iostream>
2 #include <memory> //使用shared_ptr需要include它
3
4 int main() {
5     //通过make_shared创建shared_ptr
6     std::shared_ptr<int> p1 = std::make_shared<int>();
7     *p1 = 78;
8     std::cout << "p1 = " << *p1 << std::endl;
9
10    //查看引用计数
11    std::cout << "p1 Reference count = " << p1.use_count() << std::endl;
12
13    //第二个shared_ptr也将在内部指向相同的指针
14    //这将会使引用计数变为2
15    std::shared_ptr<int> p2(p1);
16
17    //查看引用计数
18    std::cout << "p2 Reference count = " << p2.use_count() << std::endl;
19    std::cout << "p1 Reference count = " << p1.use_count() << std::endl;
20
21    //比较智能指针
22    if (p1 == p2) {
23        std::cout << "p1 and p2 are pointing to same pointer\n";
24    }
25
26    std::cout << "Reset p1" << std::endl;
27
28    //重置shared_ptr，在这种情况下，其内部不会指向内部的任何指针
29    //因此其引用计数将会变为0
30    p1.reset();
31    std::cout << "p1 Reference Count = " << p1.use_count() << std::endl;
32
33    //重置shared_ptr，在这种情况下，其内部将会指向一个新的指针
34    //因此其引用计数将会变为1
35    p1.reset(new int(11));
36    std::cout << "p1 Reference Count = " << p1.use_count() << std::endl;
37
38    //分配nullptr将取消关联指针并使其指向空值
39    p1 = nullptr;
40    std::cout << "p1 Reference Count = " << p1.use_count() << std::endl;
41
42    if (!p1) {
43        std::cout << "p1 is NULL" << std::endl;
44    }
45
46    return 0;
47 }
```

输出：

```
p1 = 78
p1 Reference count = 1
p2 Reference count = 2
p1 Reference count = 2
p1 and p2 are pointing to same pointer
Reset p1
p1 Reference Count = 0
p1 Reference Count = 1
p1 Reference Count = 0
p1 is NULL
```