

本博客rss订阅地址: <http://feed.cnblogs.com/blog/u/147990/rss>

JustDoIT

博客园 首页 新随笔 联系 订阅 管理

随笔- 209 文章- 2 评论- 128

c++ 智能指针用法详解

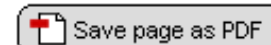
本文介绍c++里面的四个智能指针: auto_ptr, shared_ptr, weak_ptr, unique_ptr 其中后三个是c++11支持, 并且第一个已经被c++11弃用。

为什么要使用智能指针: 我们知道c++的内存管理是让很多人头疼的事, 当我们写一个new语句时, 一般就会立即把delete语句直接也写了, 但是我们不能避免程序还未执行到delete时就跳转了或者在函数中没有执行到最后的delete语句就返回了, 如果我们不在每一个可能跳转或者返回的语句前释放资源, 就会造成内存泄露。使用智能指针可以很大程度上的避免这个问题, 因为智能指针就是一个类, 当超出了类的作用域是, 类会自动调用析构函数, 析构函数会自动释放资源。下面我们逐个介绍。

auto_ptr ([官方文档](#))

```
1 class Test
2 {
3 public:
4     Test(string s)
5     {
6         str = s;
7         cout<<"Test creat\n";
```

公告



访问量: 132,339

[公益页面-寻找遗失儿童](#)

昵称: tenos

园龄: 1年9个月

粉丝: 93

关注: 7

[+加关注](#)

2015年2月						
<	日	一	二	三	四	五
	25	26	27	28	29	30
	1	2	3	4	5	6
	8	9	10	11	12	13
	15	16	17	18	19	20

```
8      }
9      ~Test()
10     {
11         cout<<"Test delete:"<<str<<endl;
12     }
13     string& getStr()
14     {
15         return str;
16     }
17     void setStr(string s)
18     {
19         str = s;
20     }
21     void print()
22     {
23         cout<<str<<endl;
24     }
25 private:
26     string str;
27 };
28
29
30 int main()
31 {
32     auto_ptr<Test> ptest(new Test("123"));
33     ptest->setStr("hello ");
34     ptest->print();
35     ptest.get()->print();
36     ptest->getStr() += "world !";
37     (*ptest).print();
38     ptest.reset(new Test("123"));
39     ptest->print();
40     return 0;
```

22 23 24 25 26 27 28

1 2 3 4 5 6 7

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](#)[更多链接](#)

我的标签

[leetcode \(119\)](#) [c++ \(17\)](#)[dp \(8\)](#) [pat \(5\)](#) [机器学习 \(4\)](#)[zoi \(4\)](#) [oj \(3\)](#) [stl \(3\)](#) [yacc \(2\)](#)[背包九讲 \(2\)](#) [更多](#)

随笔分类⁽³¹⁷⁾

[c/c++ 基础\(13\)](#)[c/c++进阶\(1\)](#)[HDOJ\(1\)](#)[IT基础\(11\)](#)[LeetCode\(121\)](#)[linux编程](#)

```
41 }
```

运行结果如下

```
Test creat
hello
hello
hello world !
Test creat
Test delete:hello world !
123
Test delete:123
```

如上面的代码：智能指针可以像类的原始指针一样访问类的public成员，成员函数get()返回一个原始的指针，成员函数reset()重新绑定指向的对象，而原来的对象则会被释放。注意我们访问auto_ptr的成员函数时用的是“.”，访问指向对象的成员时用的是“->”。我们也可用声明一个空智能指针auto_ptr<Test>ptest();

当我们对智能指针进行赋值时，如ptest2 = ptest，ptest2会接管ptest原来的内存管理权，ptest会变为空指针，如果ptest2原来不为空，则它会释放原来的资源，基于这个原因，应该避免把auto_ptr放到容器中，因为算法对容器操作时，很难避免STL内部对容器实现了赋值传递操作，这样会使容器中很多元素被置为NULL。判断一个智能指针是否为空不能使用if(ptest == NULL)，应该使用if(ptest.get() == NULL)，如下代码

[本文地址](#)

```
1  int main()
2  {
3      auto_ptr<Test> ptest(new Test("123"));
4      auto_ptr<Test> ptest2(new Test("456"));
5      ptest2 = ptest;
6      ptest2->print();
7      if(ptest.get() == NULL)cout<<"ptest = NULL\n";
8      return 0;
9  }
```

machine learning(4)

OJ(85)

Python(1)

STL(3)

windows编程(8)

ZJU-PAT(5)

ZOJ(4)

编程之美(3)

编译原理(2)

代码优化(1)

概率问题(1)

计算几何学(2)

经验分享(6)

庞果英雄会(2)

设计模式(1)

树(1)

数据库(1)

数学(2)

算法与数据结构(33)

图论(2)

网络编程(3)

字符串

随笔档案(209)

2014年12月 (2)

2014年10月 (4)

2014年9月 (3)

2014年7月 (1)

2014年6月 (22)

2014年5月 (20)

2014年4月 (22)

2014年3月 (7)

```
Test creat
Test creat
Test delete:456
123
ptest = NULL
Test delete:123
```

还有一个值得我们注意的成员函数是release，这个函数只是把智能指针赋值为空，但是它原来指向的内存并没有被释放，相当于它只是释放了对资源的所有权，从下面的代码执行结果可以看出，析构函数没有被调用。

```
1 int main()
2 {
3     auto_ptr<Test> ptest(new Test("123"));
4     ptest.release();
5     return 0;
6 }
```

```
Test creat
```

那么当我们想要在中途释放资源，而不是等到智能指针被析构时才释放，我们可以使用ptest.reset(); 语句。

unique_ptr (官方文档)

unique_ptr,是用于取代c++98的auto_ptr的产物,在c++98的时候还没有移动语义(move semantics)的支持,因此对于auto_ptr的控制权转移的实现没有核心元素的支持,但是还是实现了auto_ptr的移动语义,这样带来的一些问题是拷贝构造函数和复制操作重载函数不够完美,具体体现就是把auto_ptr作为函数参数,传进去的时候控制权转移,转移到函数参数,当函数返回的时候并没有一个控制权移交的过程,所以过了函数调用则原先的auto_ptr已经失效了.在c++11当中有了移动语义,使用move()把unique_ptr传入函数,这样你就知道原先的unique_ptr已经失效了.移动语义本身就说明了这样的问题,比较坑爹的是标准描述是说对于

2014年2月 (1)
2013年12月 (25)
2013年11月 (53)
2013年10月 (10)
2013年8月 (2)
2013年7月 (4)
2013年6月 (6)
2013年4月 (27)

积分与排名

积分 - 115999

排名 - 1274

最新评论

1. Re:LeetCode:Word Search
@tenos谢谢楼主~~我自己再看书理解下。 ...

--cindy_niu

2. Re:LeetCode:Word Search
@cindy_niu引用楼主，您好。
我尝试了设置一个标志数组：
vector> used(row, vector(col, false));但是leetcode却超时了。不知道您能给我解释下么？建议你参.....

--tenos

3. Re:编程之美 1.1 让cpu占...
楼主，话说1.2跑哪里去了

--Lthis

4. Re:LeetCode:Word Search
楼主，您好。我尝试了设置一个标志数组：vector> used(row, vector(col, false));但是leetcode却超时了。不知道您能给我解释下么？ ...

--cindy_niu

move之后使用原来的内容是未定义行为,并非抛出异常,所以还是要靠人肉遵守游戏规则.再一个,auto_ptr不支持传入deleter,所以只能支持单对象(delete object),而unique_ptr对数组类型有偏特化重载,并且还做了相应的优化,比如用[]访问相应元素等.

unique_ptr 是一个独享所有权的智能指针,它提供了严格意义上的所有权,包括:

- 1、拥有它指向的对象
- 2、无法进行复制构造,无法进行复制赋值操作.即无法使两个unique_ptr指向同一个对象.但是可以进行移动构造和移动赋值操作
- 3、保存指向某个对象的指针,当它本身被删除释放的时候,会使用给定的删除器释放它指向的对象

unique_ptr 可以实现如下功能:

- 1、为动态申请的内存提供异常安全
- 2、讲动态申请的内存所有权传递给某函数
- 3、从某个函数返回动态申请内存的所有权
- 4、在容器中保存指针
- 5、auto_ptr 应该具有的功能

```
1  unique_ptr<Test> fun()
2  {
3      return unique_ptr<Test>(new Test("789"));
4  }
5  int main()
6  {
7      unique_ptr<Test> ptest(new Test("123"));
8      unique_ptr<Test> ptest2(new Test("456"));
9      ptest->print();
10     ptest2 = std::move(ptest);//不能直接ptest2 = ptest
11     if(ptest == NULL)cout<<"ptest = NULL\n";
12     Test* p = ptest2.release();
```

5. Re:LeetCode:Longest Pa...

好文章

--Sawyer Ford

阅读排行榜

1. LeetCode:Max Points on ...
2. LeetCode 解题报告索引(...
3. LeetCode:Sort List(3681)
4. LeetCode:Candy(3607)
5. lib 和 dll 的区别、生成以...

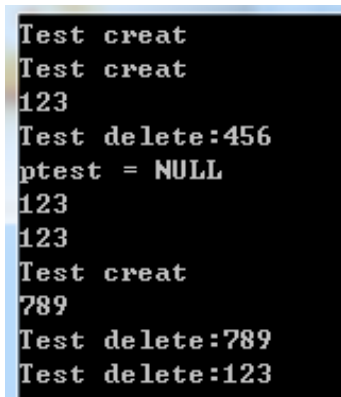
评论排行榜

1. 二维平面上判断点是否在...
2. 选择爱人的数学方法 (经...
3. 编程之美 1.1 让cpu占用...
4. C++ 内存对齐(8)
5. LeetCode:Max Points on ...

推荐排行榜

1. 均匀的生成圆和三角形内...
2. c++ 智能指针用法详解(4)
3. 链表排序 (冒泡、选择、...
4. 选择爱人的数学方法 (经...
5. C++ 内存对齐(3)

```
13     p->print();
14     ptest.reset(p);
15     ptest->print();
16     ptest2 = fun(); //这里可以用=, 因为使用了移动构造函数
17     ptest2->print();
18     return 0;
19 }
```



```
Test creat
Test creat
123
Test delete:456
ptest = NULL
123
123
Test creat
789
Test delete:789
Test delete:123
```

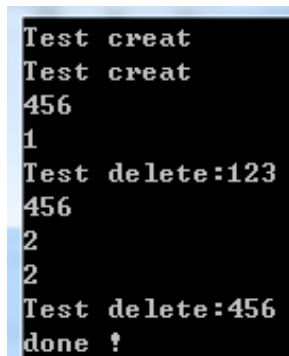
`unique_ptr` 和 `auto_ptr`用法很相似, 不过不能使用两个智能指针赋值操作, 应该使用`std::move`; 而且它可以直接用`if(ptest == NULL)`来判断是否空指针; `release`、`get`、`reset`等用法也和`auto_ptr`一致, 使用函数的返回值赋值时, 可以直接使用`=`, 这里使用c++11 的移动语义特性。另外注意的是当把它当做参数传递给函数时 (使用值传递, 应用传递时不用这样), 传实参时也要使用`std::move`, 比如 `foo(std::move(ptest))`。它还增加了一个成员函数`swap`用于交换两个智能指针的值

`share_ptr` (官方文档)

从名字`share`就可以看出了资源可以被多个指针共享, 它使用计数机制来表明资源被几个指针共享。可以通过成员函数`use_count()`来查看资源的所有者个数。出了可以通过`new`来构造, 还可以通过传入 `auto_ptr`, `unique_ptr`, `weak_ptr`来构造。当我们调用`release()`时, 当前指针会释放资源所有权, 计数减一。当计数等于0时, 资源会被释放。具体的成员函数解释可以参考 [here](#)

```
1  int main()
2  {
```

```
3     shared_ptr<Test> ptest(new Test("123"));
4     shared_ptr<Test> ptest2(new Test("456"));
5     cout<<ptest2->getStr()<<endl;
6     cout<<ptest2.use_count()<<endl;
7     ptest = ptest2;//"456"引用次数加1, "123"销毁
8     ptest->print();
9     cout<<ptest2.use_count()<<endl;//2
10    cout<<ptest.use_count()<<endl;//2
11    ptest.reset();
12    ptest2.reset();//此时"456"销毁
13    cout<<"done !\n";
14    return 0;
15 }
```



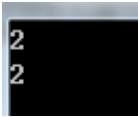
```
Test creat
Test creat
456
1
Test delete:123
456
2
2
Test delete:456
done !
```

weak_ptr(官方文档)

weak_ptr是用来解决shared_ptr相互引用时的死锁问题,如果说两个shared_ptr相互引用,那么这两个指针的引用计数永远不可能下降为0,资源永远不会释放。它是对对象的一种弱引用,不会增加对象的引用计数,和shared_ptr之间可以相互转化,shared_ptr可以直接赋值给它,它可以通过调用lock函数来获得shared_ptr。

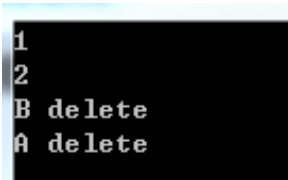
```
1     class B;
2     class A
3     {
```

```
4 public:
5     shared_ptr<B> pb_;
6     ~A()
7     {
8         cout<<"A delete\n";
9     }
10 };
11 class B
12 {
13 public:
14     shared_ptr<A> pa_;
15     ~B()
16     {
17         cout<<"B delete\n";
18     }
19 };
20
21 void fun()
22 {
23     shared_ptr<B> pb(new B());
24     shared_ptr<A> pa(new A());
25     pb->pa_ = pa;
26     pa->pb_ = pb;
27     cout<<pb.use_count()<<endl;
28     cout<<pa.use_count()<<endl;
29 }
30
31 int main()
32 {
33     fun();
34     return 0;
35 }
```

```
2
2
```

可以看到fun函数中pa，pb之间互相引用，两个资源的引用计数为2，当要跳出函数时，智能指针pa，pb析构时两个资源引用计数会减一，但是两者引用计数还是为1，导致跳出函数时资源没有被释放（A B的析构函数没有被调用），如果把其中一个改为weak_ptr就可以了，我们把类A里面的shared_ptr pb_；改为weak_ptr pb_；运行结果如下，这样的话，资源B的引用开始就只有1，当pb析构时，B的计数变为0，B得到释放，B释放的同时也会使A的计数减一，同时pa析构时使A的计数减一，那么A的计数为0，A得到释放。



```
1
2
B delete
A delete
```

注意的是我们不能通过weak_ptr直接访问对象的方法，比如B对象中有一个方法print(),我们不能这样访问，pa->pb_->print(); 英文pb_是一个weak_ptr，应该先把它转化为shared_ptr,如：shared_ptr p = pa->pb_.lock(); p->print();

参考资料

胡健: <http://www.cnblogs.com/hujian/archive/2012/12/10/2810776.html>

胡健: <http://www.cnblogs.com/hujian/archive/2012/12/10/2810754.html>

胡健: <http://www.cnblogs.com/hujian/archive/2012/12/10/2810785.html>

天方: <http://www.cnblogs.com/TianFang/archive/2008/09/20/1294590.html>

gaa_ra: http://blog.csdn.net/gaa_ra/article/details/7841204

cplusplus: <http://www.cplusplus.com/>

【版权声明】转载请注明出处: <http://www.cnblogs.com/TenosDolt/p/3456704.html>

分类: [c/c++ 基础](#)

标签: [c++智能指针](#), [智能指针](#)

绿色通道:

[好文要顶](#)

[关注我](#)

[收藏该文](#)

[与我联系](#)



tenos

关注 - 7

粉丝 - 93

[+加关注](#)

4

0

(请您对文章做出评价)

« 上一篇: [LeetCode:Maximal Rectangle](#)

» 下一篇: [LeetCode:Minimum Window Substring](#)

posted @ 2013-12-03 23:07 tenos 阅读(2462) 评论(2) 编辑 收藏

评论列表

#1楼

2013-12-04 20:21 Alexia(minmin)

好文章，虽然第一句话就出现了个大错别字，但还是收藏啦

支持(0) 反对(0)

#2楼

[楼主] 2013-12-05 20:04 tenos

@Alexia(minmin)

引用

好文章，虽然第一句话就出现了个大错别字，但还是收藏啦

谢谢提醒，已经改了

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【免费课程】案例：PDO—数据库抽象层

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

融云，免费为你的App加入IM功能——让你的App“聊”起来！！

【活动】百度开放云限量500台，抓紧时间申请啦！



最新IT新闻:

- 大众点评推年味地图产品 官方表示准确率9成
 - 那么多年过去了，为什么手机电池还是3000毫安？
 - 英国商务大臣：无人驾驶汽车会是一个大产业
 - BMW i5/i7：目标直指特斯拉Model S
 - 微软亚洲研究院计算机图像系统识别能力超人眼
- » [更多新闻...](#)



最新知识库文章:

- 使用2-3法则设计分布式数据访问层
- 数据清洗经验
- 设计中的变与不变
- 通俗解释「为什么数据库难以拓展」
- 手机淘宝高质量持续交付探索之路
- » 更多知识库文章...

Copyright ©2015 tenos

本博客rss订阅地址: <http://feed.cnblogs.com/blog/u/147990/rss>
公益页面-寻找遗失儿童