**POCO C++库学习和分析 -- 内存管理 (二)**

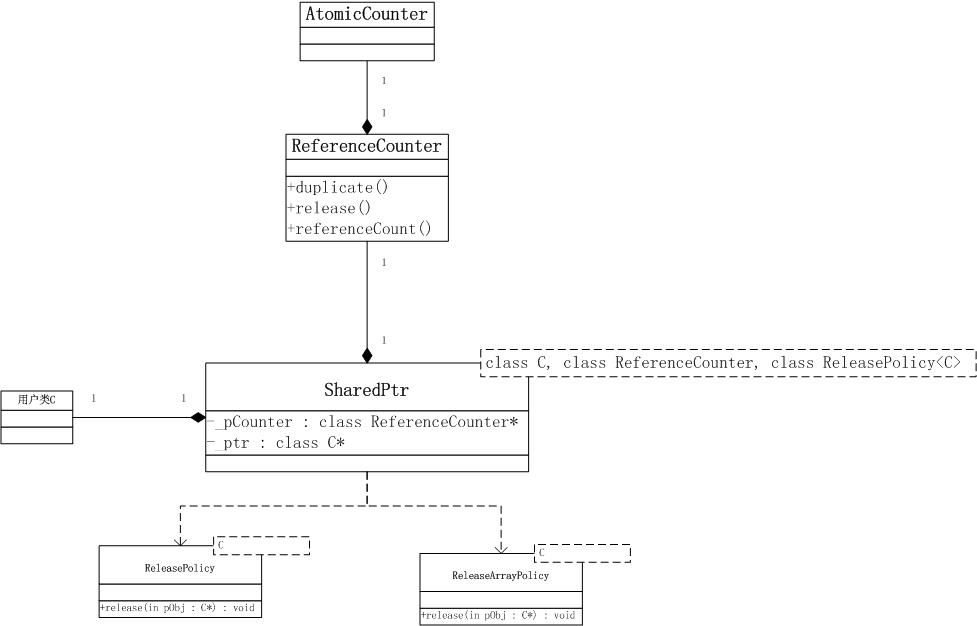
**3. SharedPtr**

        SharedPtr是Poco库中基于引用计数实现的另外一种智能指针。同AutoPtr相比，Poco::SharedPtr主要用于为没有实现引用计数功能的类（换句话说，也就是该类本身不是引用计数对象）提供引用计数服务，实现动态地址的自动回收。

        可以这么说，Poco::AutoPtr是使用继承关系来实现的智能指针，而Poco::SharedPtr是聚合方法实现的智能指针。

**3.1 SharedPtr的类图**

         首先来看一下SharedPtr的类图：



        从类图中可以看到SharedPtr是对引用计数和原生指针封装。其中有成员指针\_ptr，指向任意类型的C；同时还存在一个引用计数对象的指针\_pCounter，指向任意一个实现了引用计数的类。当然在Poco库中提供了ReferenceCount的默认实现，类ReferenceCounter。

        比较类ReferenceCounter和AutoPtr中依赖的类RefCountedObject，可以发现其实现相同，本质上就是一个东西。Poco库中之所以把两者分开，我想是为了明确的表示类与类之间的关系。ReferenceCounter用于组合，而RefCountedObject用于继承。

        SharedPtr在实现模板的时候，还预留了RP参数，这是一个释放策略，用于调整SharedPtr在释放数组和单个对象之间不同策略的转换。

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. **template** <**class** C, **class** RC = ReferenceCounter, **class** RP = ReleasePolicy<C> >
2. **class** SharedPtr
3. {
4. // ...
5. }

        其中C为对象原生指针，RC为SharedPtr管理的引用计数对象，RP为内存释放策略。

**3.2 SharedPtr操作符和值语义**

        1. Poco::SharedPtr同样支持关系操作符==, !=, <, <=, >, >=；

        2. 当Poco::SharedPtr中原生指针为空时，使用解引用操作符“\*”或者"->"，Poco::SharedPtr会抛出一个NullPointerException 异常。

        3. Poco::SharedPtr同样支持全值语义，包括默认构造函数，拷贝构造函数，赋值函数并且同样可以用于各类容器(如std::vector 和 std::map)

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. SharedPtr& operator = (C\* ptr)
2. {
3. **return** assign(ptr);
4. }
6. SharedPtr& assign(C\* ptr)
7. {
8. **if** (get() != ptr)
9. {
10. RC\* pTmp = **new** RC;
11. release();
12. \_pCounter = pTmp;
13. \_ptr = ptr;
14. }
15. **return** \***this**;
16. }
18. **void** release()
19. {
20. poco\_assert\_dbg (\_pCounter);
21. **int** i = \_pCounter->release();
22. **if** (i == 0)
23. {
24. RP::release(\_ptr);
25. \_ptr = 0;
27. **delete** \_pCounter;
28. \_pCounter = 0;
29. }
30. }

        注意，在SharedPtr赋值操作符"="中的操作，对于原生指针\_ptr的操作策略是交换，而引用计数对象\_pCounter的策略是先new一个，再交换。

        4. 可以用SharedPtr::isNull()和SharedPtr::operator ! () 去检查内部的原生指针是否为空。

**3.3 SharedPtr和Cast类型转换**

        同普通指针类似，Poco::SharedPtr支持cast操作符。这在 template <class Other>SharedPtr<Other> cast() const中实现，其定义如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. **template** <**class** Other>
2. SharedPtr<Other, RC, RP> cast() **const**
3. /// Casts the SharedPtr via a dynamic cast to the given type.
4. /// Returns an SharedPtr containing NULL if the cast fails.
5. /// Example: (assume class Sub: public Super)
6. ///    SharedPtr<Super> super(new Sub());
7. ///    SharedPtr<Sub> sub = super.cast<Sub>();
8. ///    poco\_assert (sub.get());
9. {
10. Other\* pOther = **dynamic\_cast**<Other\*>(\_ptr);
11. **if** (pOther)
12. **return** SharedPtr<Other, RC, RP>(\_pCounter, pOther);
13. **return** SharedPtr<Other, RC, RP>();
14. }

        Poco::SharedPtr中类型转换总是安全的，在其内部实现时，使用了dynamic\_cast ，所以一个不合法的转换，会导致原生指针为空。

        Poco::SharedPtr中赋值操作符的兼容性通过构造函数和赋值操作符共同完成。

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. **template** <**class** Other, **class** OtherRP>
2. SharedPtr& operator = (**const** SharedPtr<Other, RC, OtherRP>& ptr)
3. {
4. **return** assign<Other>(ptr);
5. }
7. **template** <**class** Other, **class** OtherRP>
8. SharedPtr& assign(**const** SharedPtr<Other, RC, OtherRP>& ptr)
9. {
10. **if** (ptr.get() != \_ptr)
11. {
12. SharedPtr tmp(ptr);
13. swap(tmp);
14. }
15. **return** \***this**;
16. }
18. **template** <**class** Other, **class** OtherRP>
19. SharedPtr(**const** SharedPtr<Other, RC, OtherRP>& ptr): \_pCounter(ptr.\_pCounter), \_ptr(**const\_cast**<Other\*>(ptr.get()))
20. {
21. \_pCounter->duplicate();
22. }

         下面是关于操作符的一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. #include "Poco/SharedPtr.h"
2. **class** A
3. {
4. **public**:
5. **virtual** ~A()
6. {}
7. };
8. **class** B: **public** A
9. {
10. };
11. **class** C: **public** A
12. {
13. };
14. **int** main(**int** argc, **char**\*\* argv)
15. {
16. Poco::SharedPtr<A> pA;
17. Poco::SharedPtr<B> pB(**new** B);
18. pA = pB;                         // okay, pB is a subclass of pA
19. pA = **new** B;
20. // pB = pA;                     // will not compile
21. pB = pA.cast<B>();              // okay
22. Poco::SharedPtr<C> pC(**new** C);
23. pB = pC.cast<B>();              // pB is null
24. **return** 0;
25. }

**3.4 SharedPtr使用注意事项**

        从上面我们可以看到Poco::SharedPtr拥有Poco::AutoPtr类似的一些特征，如解引用，赋值操作符。但同Poco::AutoPtr不同的是，当使用赋值操作符“=”把一个SharedPtr赋给一个原生指针，然后再把这个原生指针赋予另个SharedPtr时是不允许的。这时候两个SharedPtr都会声称拥有对象的所有权，将导致程序crash。在AutoPtr中虽然不推荐如此做，但提供了一个解决方案，使用以下函数，并至"shared=true"。

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. AutoPtr::AutoPtr(C\* pObject, **bool** shared);
2. AutoPtr& AutoPtr::assign(C\* pObject, **bool** shared);

        对于Poco::SharedPtr来说，最好的方法是一旦用SharedPtr获取到对象所有权后，就不要再试图使用指向对象的原生指针。

       下面是SharedPtr的一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. #include "Poco/SharedPtr.h"
2. #include <string>
3. #include <iostream>
4. **using** Poco::SharedPtr;
5. **int** main(**int** argc, **char**\*\* argv)
6. {
7. std::string\* pString = **new** std::string("hello, world!");
8. Poco::SharedPtr<std::string> p1(pString);                  // rc == 1
9. Poco::SharedPtr<std::string> p2(p1);                       // rc == 2
10. p2 = 0;                                                    // rc == 1
11. // p2 = pString;                                           // BAD BAD BAD: multiple owners -> multiple delete
12. p2 = p1;                                                   // rc == 2
13. std::string::size\_type len = p1->length();                 // dereferencing with ->
14. std::cout << \*p1 << std::endl;                             // dereferencing with \*
15. **return** 0;
16. }
17. // rc == 0 -> deleted

**3.5 SharedPtr和数组**

       默认的SharedPtr删除策略是指删除对象。如果创建对象时使用数组，并把它委托给SharedPtr，必须使用对应数组删除策略。这时候SharedPtr的模板参数中ReleasePolicy应该使用类ReleaseArrayPolicy。

      下面是对应的另一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8636559)

1. **template** <**class** C>
2. **class** ArrayReleasePolicy
3. {
4. **public**:
5. **static** **void** release(C\* pObj)
6. /// Delete the object.
7. /// Note that pObj can be 0.
8. {
9. **delete** [] pObj;
10. }
11. };
12. **char**\* pStr = **new** **char**[100];
13. SharedPtr<**char**, Poco::ReferenceCounter, ArrayReleasePolicy> p(pStr);

**3.6 其他**

      同boost库比较的话，Poco中的SharedPtr同boost库中的shared\_ptr可以说是类似的,行为上相似，虽然实现不同。

（版权所有，转载时请注明作者和出处  <http://blog.csdn.net/arau_sh/article/details/8636559>）