智能指针与动态内存

一、总述：

---》智能指针：通过使用包装指针的类，自动的分配和释放动态内存（构造和析构时）

---》新标准shared\_ptr：允许多个指针指向同一个对象，共享动态内存和引用计数

---》新标准unique\_ptr：独占一个给定对象，不支持普通的拷贝后赋值（函数返回值可以移动拷贝）

---》新标准weak\_ptr：可以和shared\_ptr“弱”共享对象（不改变引用计数）

总结

（1）建议：

---》坚持只使用智能指针，管理动态内存

---》智能指针不要和普通指针混用

（2）注意：智能指针陷阱

---》不要使用相同的内置指针初始化(或reset)多个智能指针

---》不delete get()返回的指针

---》不使用get()初始化reset另一个智能指针（情况与第一条类似）

---》如果使用了get()返回的指针，记住当最后一个对应的智能指针销毁后，该指针也无效了

---》如果使用智能指针管理的不是动态内存，记住传递一个删除器（见p416）

==================================================

二、shared\_ptr类

1、shared\_ptr和unique\_ptr都支持的操作

---》shared\_ptr<T> sp 默认初始化为nullptr

---》unique\_ptr<T> up 默认初始化为nullptr

---》p 用作判断条件，若p指向一个对象，则为true

---》p.get() 返回p中保存的new指针

---》swap()/p.swap() 交换

2、shared\_ptr独有的操作

（1）初始化

---》make\_shared<T>(10) 返回一个shared\_ptr

（2）拷贝构造

---》shared\_ptr<T> p3(p1) p1的引用计数++

（3）拷贝赋值

---》p3 = p2; p3的引用计数--，p2的引用计数++

（4）引用计数

---》p.use\_count() 返回共享对象的引用计数

---》p.unique() 若p.use\_count()=1，返回true

3、引用计数的增减

（1）何时递增

---》当shared\_ptr发生拷贝时，如：拷贝构造、赋值、函数参数、函数返回值等

（2）何时递减

---》当给shared\_ptr赋一个新值（如：nullptr）

---》shared\_ptr离开作用域

（3）shared\_ptr引用计数实现方式

---》每当使用内置指针初始化(或reset)一个shared\_ptr对象，该对象就会开辟内存，管理引用计数器

---》当shared\_ptr对象发生拷贝构造、赋值时，会指向同一块管理引用计数的内存，并执行引用计数增减

---》当shared\_ptr对象发生（拷贝、赋值、析构）时，会修改相应的引用计数

-------------------------------------------

4、共享容器中元素的方法（了解）

---》用类包装含有shared\_ptr指针包装的容器类型（见p404）

==================================================

三、new和delete直接管理内存

1、new操作符

（1）new操作符会调用类类型的构造函数

（2）动态分配的const对象

---》const int \*pci = new const int(1024);

（3）new内存失败（见p409）

---》分配失败，抛出std::bad\_alloc int \*p1 = new int;

---》不抛异常，new返回一个空指针 int \*p2 = new (nothrow) int;

2、delete操作符

（1）直接管理动态内存常见的3个问题

---》忘记delete内存

---》使用已经delete的对象： 指针置空，可以解决

---》同一块内存释放两次

（2）讨论一种情况：当有2个指针指向相同的动态内存时

---》如果只delete一个指针并置空，另一个指针的查找（并置空）可能很困难

---》如果delete两个指针，会发生重析构错误

（3）使用智能指针的好处

---》注意：使用智能指针可以避免以上所有问题

---》坚持只使用智能指针，管理动态内存

==================================================

四、shared\_ptr和new结合使用

1、使用new指针初始化，shared\_ptr构造函数

---》因为，接受指针参数的智能指针构造函数是explicit的，所以不能发生隐式转换

---》可能发生隐式转换的情况有两种：new指针初始化shared\_ptr、函数返回shared\_ptr指针

---》不建议使用new指针初始化shared\_ptr，应该用make\_shared<T>初始化

2、shared\_ptr构造函数和reset()

---》构造函数参数可以是：new指针、unique\_ptr、shared\_ptr

---》reset()：原指向引用计数减一，新指向引用计数加一，参数可以为空

注意：关于可调用对象p412，未解决

3、智能指针不要和普通指针混用（见p413）

---》当一个智能指针与普通指针绑定时，我们就将内存的管理责任交给了这个智能指针

---》此时，我们就不应该在使用内置指针来访问智能指针指向的内存了，因为不知道对象何时被销毁

4、使用get返回的指针的代码不能delete此指针

---》情况：有时，我们需要向不能使用智能指针的代码传递一个智能指针

---》原理同上

5、有时，我们需要对共享对象深拷贝

---》见p414，结合使用unique和reset

==================================================

五、智能指针和异常

（1）使用智能指针，当程序抛出异常仍会释放资源，见p415

（2）使用智能指针管理非动态内存

---》传递一个删除器（函数指针），会执行指定的善后操作，代替delete（见p416）

==================================================

六、unique\_ptr类

1、任何时刻，只能有一个unique\_ptr指向一个给定对象

2、unique\_ptr不支持普通拷贝或赋值

---》例外：作为函数返回值时，会发生移动拷贝

3、unique\_ptr独有的操作

---》reset()和release()

---》见p418

4、unique\_ptr同样可以管理非动态内存，记住传递一个删除器

==================================================

七、weak\_ptr类

1、weak\_ptr可以“弱”共享shared\_ptr

---》初始化和析构weak\_ptr，不会改变所指向shared\_ptr的引用计数

2、weak\_ptr的操作

---》reset()/use\_count()

---》expired() 若p.use\_count()=1，返回true

---》lock() 若expired为true，返回一个空shared\_ptr，否则返回一个指向共享对象的shared\_ptr

3、注意

---》由于是弱共享，创建weak\_ptr不会改变shared\_ptr的引用计数，因此weak\_ptr指向的对象可能被释放掉

---》由于对象可能不存在，所以不能使用weak\_ptr直接访问对象，而必须调用lock检查共享对象是否还在

==================================================

八、动态数组

1、建议使用容易代替动态数组，所以这问题不用深究了

2、跳过（p425~p435）

3、包含内容：

---》unique\_ptr和shared\_ptr管理动态数组的方法

---》allocator类的使用方法

---》例子：使用标准库的文本查询程序