|  |  |
| --- | --- |
| 1. shared\_ptr和new结合使用 2. 不要混合使用普通指针和智能指针 3. 也不要使用get初始化另一个智能指针或为智能指针赋值 4. 其他shared\_ptr操作 | Shared\_ptr<int> p1(new int(42));  Shared\_ptr<int> clone(int p){  Return shared\_ptr<int>(new int(p));  }  Void process(shared\_ptr<int> ptr)  Shared\_ptr<int> p(new int(42));  Process(p);  Int I = \*p;  Int \*x(new int(1024);  Process(shared\_ptr<int>(x));//合法，但内存会被释放  Shared\_ptr<int>p(new int(42));  Int \*q = p.get();  {  Shared\_ptr<int>(q);  }  Int foo = \*p;//未定义  p.reset(new int(1024)); |
| 1. shared\_ptr和new结合使用 2. 若不初始化一个智能指针，就被初始化为一个空指针 3. 我们还可以用new返回的指针来初始化智能指针 4. 智能指针的构造函数是explicit ，就是说，必须用直接初始化，不能用赋值初始化 =   Share\_ptr<int> p1 = new int(1024);//no   1. 初始化智能指针的普通指针必须指向动态内存，智能指针用delete删除它，但是可以关联指向其它类型的资源的指针上，需提供自己delete的操作 2. 不要混合使用普通指针和智能指针 3. 协调对象的析构，但仅限于自身的拷贝，避免无意中将同一块内存绑定到多个独立创建的shared\_ptr上。 4. 用new不能协调，用make\_shared可以协调 5. 例子，给process传递一个临时shared\_ptr指向内置指针   Int \*x(new int(1024);  Process(shared\_ptr<int>(x));//合法，但x内存会被释放  表达式是将这个临时对象传给了形参才结束，创建时为1，传递给形参计数为2，表示式结束销毁为1，形参结束为0，所以会销毁指针x的内存   1. X会变成空悬指针，未定义状态。 2. 把内置指针交给智能指针操作，就不要使用内置指针访问内存的好，无法知道什么时候被销毁 3. 也不要使用get初始化另一个智能指针或为智能指针赋值 4. Shared\_ptr的Get函数返回一个内置指针，指向智能指针管理的对象。是为了只能只用内置指针不能使用智能指针而设计的。Get返回的指针不能用delete此指针 5. 若将get返回的指针给另一个shard\_ptr，这两个shared\_ptr并没有关联，于是若另一个shared\_ptr被销毁那么原shared\_ptr指向的内存也被销毁，变为空悬指针，产生未定义行为。而且原指针被销毁，同一块内存delete两次。   Note:使用get得到的内置指针最好不要delete，不要用get初始化另一个智能指针或者为另一个智能指针赋值   1. 其他shared\_ptr操作   p7.reset(new int(19));   1. 用reset来将一个新的指针赋予一个shared\_ptr，经常与unique一起使用 2. Reset将会销毁原来p7的内存并且指向新内存，若没有指向新内存，让p置位空 3. Unique判断自己是否唯一用户，是的话可以随意改变自身的值 | |



