1：什么是智能指针？

智能指针(smart pointer)是存储指向动态分配（堆）对象指针的类，用于生存期控制，能够确保自动正确的销毁动态分配的对象，>防止内存泄露（利用自动调用类的析构函数来释放内存）。它的一种通用实现技术是使用引用计数（除此之外还有资源独占，如(auto\_ptr）,只引用，不计数（weak\_ptr））。智能指针类将一个计数器与类指向的对象相关联，引用计数跟踪该类有多少个对象共享同

一指针。每次创建类的新对象时，初始化指针并将引用计数置为1；当对象作为另一对象的副本而创建时，拷贝构造函数拷贝指针并>增加与之相应的引用计数；对一个对象进行赋值时，赋值操作符减少左操作数所指对象的引用计数（如果引用计数为减至0，则删除>对象），并增加右操作数所指对象的引用计数；调用析构函数时，构造函数减少引用计数（如果引用计数减至0，则删除基础对象）>。

2：为什么要用智能指针以及智能指针的原理是什么？

原因：1）：手动malloc/new出来的资源，容易忘记free/delete；

2）：影响执行流的地方需要注意释放释放资源，容易导致资源泄漏（如free/delete在return之后）。

3）：中途抛出异常，无法释放资源。如：int\*p1=new int；int\*p2=new int [10000000]；delete p1;delele [ ] p2;因为p2 new的内存比较大，万一new失败，则导致p1永远无法释放。

原理：为了解决以（1）、（2）、（3）上问题，创建一份资源出来的时候，交给一个类对象去管理，当类对象声明周期结束时，自>动调用析构函数释放资源。除此之外，通过运算符重载（重载\*和重在-> 等），可以像指针一样使用。

3：分析下常见的智能指针有哪些？

（1）：auto\_ptr

1）它是C++标准库提供的类模板，auto\_ptr对象通过初始化指向由new创建的动态内存，它是这块内存的拥有者，一块内存>不能同时被分给两个拥有者（资源独占）。当auto\_ptr对象生命周期结束时，其析构函数会将auto\_ptr对象拥有的动态内存自动释放

。

2）auto\_ptr不能指向数组，因为auto\_ptr在析构的时候只是调用delete,而数组应该要调用delete[]。（但uniquearray管>理的是一段连续的空间，它也是防拷贝的，功能类似于vector。）

3）auto\_ptr不能作为容器对象，因为它不支持拷贝构造与赋值（出错了也不容易发现），STL容器中的元素经常要支持拷贝

，赋值等操作，在这过程中auto\_ptr会传递所有权，那么就会出错。

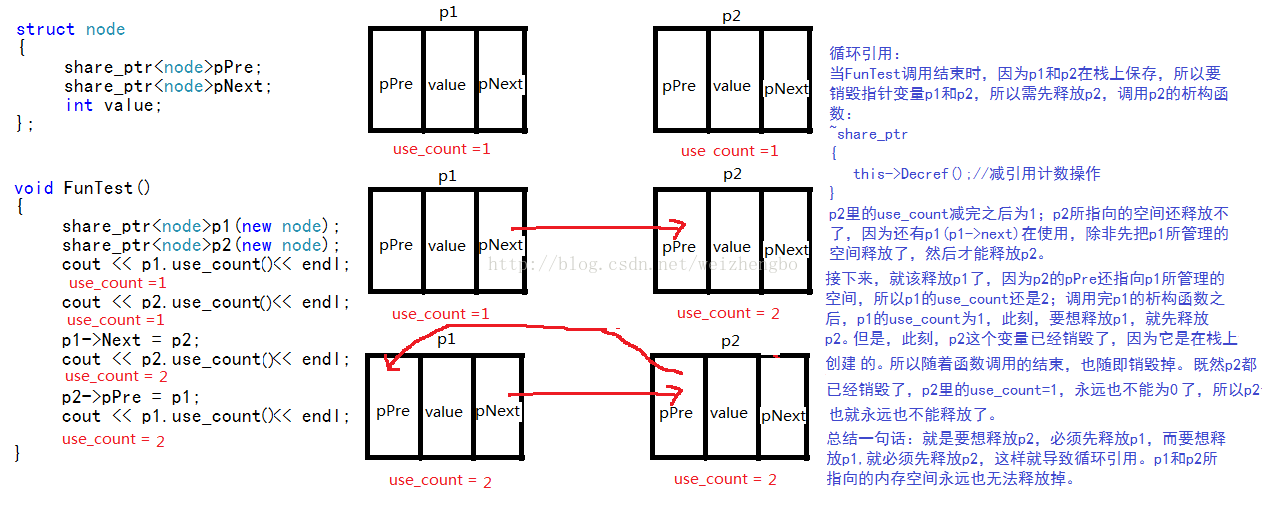
（2）：unique\_ptr

它是（ C++11引入的，前身是scoped\_ptr，scoped\_ptr是boost库里的），也不支持拷贝构造和赋值，但比auto\_ptr好，直>接赋值会编译出错（与auto\_ptr最大的不同就是类内私有的声明了拷贝构造函数和赋值运算符重载，是针对auto\_ptr的缺点而出现的

）。

（3）：shared\_ptr

C++11或boost的shared\_ptr，基于引用计数的智能指针。可随意赋值，直到内存的引用计数为0的时候这个内存会被释放。>环状的链式结构可能会形成内存泄露（循环引用）



为了解决类似这样的问题，C++11引入了weak\_ptr，来打破这种循环引用。

（4）：weak\_ptr

C++11或boost的weak\_ptr，弱引用。 引用计数有一个问题就是互相引用形成环，这样两个指针指向的内存都无法释放。需要手动打破循环引用或使用weak\_ptr。顾名思义，weak\_ptr是一个弱引用，它是为了配合shared\_ptr而引入的一种智能指针，它指向一个由shared\_ptr管理的对象而不影响所指对象的生命周期，也就是说，它只引用，不计数。如果一块内存被shared\_ptr和weak\_ptr同时引用，当所有shared\_ptr析构了之后，不管还有没有weak\_ptr引用该内存，内存也会被释放。所以weak\_ptr不保证它指向的内存一定是有效的，在使用之前需要检查weak\_ptr是否为空指针。weak\_ptr并没有重载operator->和operator \*操作符，因此不可直接通过weak\_ptr使用对象，典型的用法是调用其lock函数来获得shared\_ptr示例，进而访问原始对象。