RAII，资源获取即初始化。（构造器和析构器就是这个原理来设计的。为了保证任何情况下，有构造就有析构，构造的对象都被销毁。）

智能指针头文件 <memory>

对于编译器来说，智能指针实际上是一个栈对象，并非指针类型，在栈对象生命期即将结束时，智能指针通过析构函数释放有它管理的堆内存。所有智能指针都重载了“operator->”操作符，直接返回对象的引用，用以操作对象。访问智能指针原来的方法则使用“.”操作符。

boost::shared\_ptr 也可以很方便的使用。关键的一点，boost::shared\_ptr 内部维护了一个引用计数，由此可以支持复制、参数传递等。boost::shared\_ptr 提供了一个函数 use\_count() ，此函数返回 boost::shared\_ptr 内部的引用计数。

boost::weak\_ptr 属于 boost 库，定义在 namespace boost 中，包含头文件 #include<boost/smart\_ptr.hpp> 便可以使用。

在讲 boost::weak\_ptr 之前，让我们先回顾一下前面讲解的内容。似乎 boost::scoped\_ptr、boost::shared\_ptr 这两个智能指针就可以解决所有单个对象内存的管理了，这儿还多出一个 boost::weak\_ptr，是否还有某些情况我们没纳入考虑呢？

回答：有。首先 boost::weak\_ptr 是专门为 boost::shared\_ptr 而准备的。有时候，我们只关心能否使用对象，并不关心内部的引用计数。boost::weak\_ptr 是 boost::shared\_ptr 的观察者（Observer）对象，观察者意味着 boost::weak\_ptr 只对 boost::shared\_ptr 进行引用，而不改变其引用计数，当被观察的 boost::shared\_ptr 失效后，相应的 boost::weak\_ptr 也相应失效。

auto\_ptr<A> p(new A)

（p的行为像一个指针，效果和一般指针一样。相当于在堆上申请内存，但是可以自己控制生命周期，离开作用域时自动销毁空间。）

我们都知道程序中除了静态内存和栈内存，每个程序还拥有一个内存池。

**静态内存：**保存局部static对象、类static数据成员、以及定义在任何函数之外的变量；

**栈内存：**用来保存定义在函数内的非static对象；

**内存池：**称为内存空间(free store)或堆(heap)，程序用堆来储存动态分配的对象，即在程序运行时分配的对象。

shared\_ptr类，shared的pointer，智能指针的一种，这里shared\_ptr 允许多个指针指向同一个对象。