**12 动态内存**

1，动态内存与智能指针

静态内存：static对象和类外变量。使用前分配，程序结束时销毁。

栈内存：函数内的非static对象。由编译器自动创建和销毁。

堆：动态分配的对象。代码显示地销毁。

2，new和delete

动态内存的管理通过一对运算符来完成：new和delete。

new和delete的问题：1，忘记delete内存；2，使用已经释放掉的内存；3，同一块内存释放两次

3，智能指针

两种智能：shared\_ptr允许多个指针指向同一个对象；unique\_ptr则独占所指向的对象。weak\_ptr指向shared\_ptr所管理的对象。

**shared\_ptr**：

最安全的分配和使用动态内存方法：

shared\_ptr<int> p3 = make\_shared<int>(42);

使用new创建智能指针：

shared\_ptr<int> p2(new int(1024))

shared\_ptr通过引用计数实现。计数器变为0，通过析构函数自动释放管理的对象。

创建shared\_ptr时，可以传递一个指向删除器函数的参数：shared\_ptr<connection> p(&c, end\_connection)

**unique\_ptr：**

unique\_ptr不支持拷贝和赋值，但可以通过release或reset将指针所有权从一个unique\_ptr转移给另一个unique。

向unique\_ptr传递删除器

unique\_ptr<connection, decltype(end\_connection)\*> p(&c,end\_connection)

**weak\_ptr：**

weak\_ptr指向由一个shared\_ptr管理的对象。

将一个weak\_ptr绑定到一个shared\_ptr不会改变它的引用计数。

一旦最后一个指向对象的shared\_ptr被销毁，对象就会被释放，即使有weak\_ptr指向对象。

4，allocator

allocator类将内存分配和对象构造分离开来。

allocator<string> alloc;

auto const p = alloc.allocate(n);//分配n个未初始化的string

用完对象，必须对每个构造的元素调用destory来销毁它们。释放内存通过deallocate完成。