43 Object Lifetime in C++ (Stack_Scope Lifetimes)

C++的对象生存期（栈作用域生存期）
可复习 38 How to CREATE_INSTANTIATE OBJECTS in C++

1. 作用域

关于栈，你可以想成把一本书放到书架上，你在此作用域下（这本书内）声明的变量（书里写的东西），一旦作用域结束，书就被拿出来扔掉了。你在书里声明的每一个基于*stack*(栈)的变量，所有栈里创建的对象都消失了。

```C++
class Entity
{
private:
    int x;
};
```

这里有一个栈中初始化的变量，不是在堆上分配的。这个变量也在这个类的作用域里，这意味着当这个类消失时，变量也会消失。

```C++
class Entity
{
public:
    Entity()
    {
        std::cout << "Created Entity" << std::endl;
    }

    ~Entity()
    {
        std::cout << "Destroyed Entity" << std::endl;
    }

};

int main()
{
    {
        Entity e;    // 这样写的话，就不是将它创建在堆上，而是创建在栈上
    }                // 将调用默认构造函数

    std::cin.get();
}

// 输出
// Created Entity
// Destroyed Entity
```

改为堆上分配：

```C++
int main()
{
    {
        Entity* e = new Entity();
    }   // 跳过了这个作用域

    std::cin.get();
}

// 输出
// Created Entity
```

请记住，如果你在栈上创建一个变量，当它超出范围就会消失。

下面看另一个例子：

```C++
int* CreateArray()
{
    int array[50];
    return array;
}
```

合法吗？
不合法，因为我们没有在堆上分配它，因为并没有使用`new`。我们只在栈上声明它，当我们返回一个指向它的指针时，它会返回一个指向栈内存的指针，而这个栈内存会在我们离开作用域的时候清除。

所以如果你像下面这么写它是会失败的。

```C++
int main()
{
    int* a = CreateArray();
}
```

如果你想写一个这样的函数，你基本上有两个选择。
你可以让这个在堆上分配数组，从而确保了它的生存期会一直存在。

```C++
int* CreateArray()
{
    int* array = new int[50];
    return array;
}

int main()
{
    int* a = CreateArray();
}
```

或者你可以将这里创建的数据复制给一个在栈作用域之外存在的变量

```C++
void CreateArray(int* array)
{
    // fill our array
}

int main()
{
    int array[50];
    CreateArray(array);
}
```

这里我们只是传递一个指针，所以不会做分配这个操作。

因此在局部创建数组是一个典型的错误。

那我们有什么办法将这种栈上变量自动销毁变得有用呢?

2. 作用域指针

它基本上是一个类，是一个指针的包装器，在构造时调用分配指针，然后在析构时删除指针。所以我们可以自动化这个`new`和`delete`。

这就是一个基本的*scoped pointer*(作用域指针) （类）

```C++
class ScopedPtr
{
private:
    Entity* m_Ptr;
public:
    ScopedPtr(Entity* ptr)
        : m_Ptr((ptr))
    {
    }

    ~ScopedPtr()
    {
        delete m_Ptr;
    }
};

int main()
{
    {
        ScopedPtr e = new Entity();
    }
}


//输出
// Created Entity
// Destroyed Entity
```

这个ScopedPtr类是在栈上被分配的，意味着如果e被删除了，在析构函数中，会`delete`这个被包装的Entity指针。即使我们用`new`来做堆分配，也完成了销毁。

这就是一个*smart pointer*：*unique_ptr*做的最基本的事、

自动构造、自动析构，这种基于栈的变量，离开作用域后会被销毁，是非常有用的。

计时器

假设你想计算你在基准测试范围内的时间，你可以写一个timer类，在对象构造时启动timer，然后在打印结果时，停止timer，也就是计时器对象被销毁时。这样你只需要在函数开头写一行代码，整个作用域就会被计时。而且你永远不需要手动改调用计时器停止，因为一旦它超出了作用域，会自动调用并停止

mutex locking（互斥锁）

如果你想锁定一个函数，以便多个线程可以同时访问它而不会爆炸，你可以有一个自动作用域锁定，在函数的开头锁住它，在函数结束时解锁它。