问题描述

我刚从 Java 转到使用 C++ 进行面向对象开发，我发现一个很让我非常困惑的问题：C++ 中经常出现使用对象指针，而不是直接使用对象本身的代码，比如下面这个例子：

```
Object *myObject = new Object;
```

而不是使用：

```
Object myObject;
```

要不就是调用对象的方法（比如 testFunc()）时不使用这种方式：

```
myObject.testFunc();
```

而是得写成这样：

```
myObject->testFunc();
```

我不明白代码为什么要写成这种形式，我能想到的是指针方式是直接访问内存，这么写代码可以提高代码效率以及执行速度，是这样的么？

最佳回复来自 Joseph Mansfield

非常不幸，你在代码中遇到这么多的动态内存分配，但这个只能说明有现在有太多不合格的 C++ 程序员。

这么说吧，你的两个问题本质上是同个问题。第一个问题是，应该何时使用动态分配（使用 new 方法）？第二问题是，什么时候该使用指针？

最先要牢记的重点是，你应该根据实际需求选择合适的方法。 一般来说，使用定义对象的方式比起使用手工动态分配（或new指针）的方式会更加合理以及安全。

动态分配

你的提问中，所列出的两种分配对象方式的主要区别在于对象的生存期。通过 Object myObject 方式定义对象，对象的生存期是在其作用域内自维护（automatic storage），这个意味着程序离开对象的作用域之后，对象将被自动销毁。当通过 new Object() 方式分配对象时，对象的生存期是动态的，这个意味着若不显式地 detete 对象，对象将一直存在。你应该只在必要的时候使用动态分配对象。换句话说，只要有可能，你应该首选定义可自维护的对象。

这里是两个常见需要动态分配对象的情况：

1. 分配不限制作用域的对象，对象存储在其特定的内存中，而不是在内存中存储对象的拷贝。如果对象是可以拷贝/移动的，一般情况下你应该选择使用定义对象的方式。
2. 定义的对象会消耗大量内存，这时可能会耗尽栈空间。如果我们永远不需要考虑这个问题那该多好（实际大部分情况下，我们真不需要考虑），因为这个本身已经超出 C++ 语言的范畴，但不幸的是，在我们实际的开发过程中却不得不去处理这个问题。

当你确实需要动态分配对象时，应该将对象封装在一个智能指针（smart pointer）或其他提供RAII机制的类型中（类似标准的 container）。智能指针提供动态对象的所有权语义（ownership），具体可以看一下std::unique_ptr 和 std::shared_ptr 这两个例子。如果你使用得当，基本上可以避免自己管理内存（具参见 Rule of Zero）。

指针

当然，不使用动态分配而采取原始指针（raw pointer）的用法也很常见，但是大多数情况下动态分配可以取代指针，因此一般情况应该首选动态分配的方法，除非你遇到不得不用指针的情况。

1. 使用引用语义（reference semantics）的情况。有时你可能需要通过传递对象的指针（不管对象是如何分配的）以便你可以在函数中去访问/修改这个对象的数据（而不是它的一份拷贝），但是在大多数情况下，你应该优先考虑使用引用方式，而不是指针，因为引用就是被设计出来实现这个需求的。注意，采用这种方式，对象生存期依旧在其作用域内自维护。当然，如果通过传递对象拷贝可以满足要求的情况下是不需要使用引用语义。

2. 使用多态的情况。通过传递对象的指针或引用调用多态函数（根据入参类型不同，会调用不同处理函数）。如果你的设计就是可以传递指针或传递引用，显然，应该优先考虑使用传递引用的方式。

3. 对于入参对象可选的情况，常见的通过传递空指针表示忽略入参。如果只有一个参数的情况，应该优先考虑使用缺省参数或是对函数进行重载。要不然，你应该优先考虑使用一种可封装此行为的类型，比如 boost::optional （或者std::optional，已经在 C++ 14 草案 n3797 14 中发布 ）。

4. 通过解耦编译类型依赖减少编译时间的情况。使用指针的一个好处在于可以用于前向声名（forward declaration）指向特定类型（如果使用对象类型，则需要定义对象），这种方式可以减少参与编译的文件，从而显著地提高编译效率，具体可以看 Pimpl idiom 用法。

5. 与C库或C风格的库交互的情况。此时只能够使用指针，这种情况下，你要确保的是指针使用只限定在必要的代码段中。指针可以通过智能指针的转换得到，比如使用智能指针的get成员函数。如果C库操作分配的内存需要你在代码中维护并显式地释放时，可以将指针封装在智能指针中，通过实现 deleter 从而可以有效的地释放对象。