构造、复制、析构语意学

一种所谓的 Plain OI'Data 声明形式：

:::C++ struct Point { float x,y,z; };

概念上来讲，对于一段这样的 C++代码,编译器会为之合成一个默认构造函数、 复制构造函数、析构函数、赋值操作符。然而实际上编译器会分析这段代码， 并给 `Point` 贴上 Plain OI'Data 标签。编译器在此后对于 `Point` 的处理与在 C 中完全一样,也就是说上述的函数都不会被合成。可见概念上应当由编译器 合成的函数，并不一定会合成，编译器只有在必要的时候才会合成它们。由此 一来，原本在观念上应该调用这些函数的地方实质上不会调用，而是用其它的 方法来完成上面的功能，比方复制控制会用 bitwise copy。

对象构造语意学

无继承情况下的对象构造：略。

单继承体系下的对象构造

对于简单定义的一个对象 `T object;`,很明显它的默认构造函数会被调用（被 编译器合成的或用户提供的）。但是一个构造函数究竟做了什么，就显得比较 复杂了——编译器给了它很多的隐藏代码。编译器一般会做如下扩充操作 [1]：

1. 调用所有虚基类的构造函数，从左到右，从最深到最浅：
    1. 如果该类被列于成员初始化列表中，任何明确明确指定的参数，都应 该被传递过来。若没有列入成员初始化列表中，虚基类的一个默认构 造函数被调用（有的话）。
    2. 此外，要保证虚基类的偏移量在执行期可存取，对于使用 vbptr 来实现 虚基类的编译器来说，满足这点要求就是对 vbptr 的初始化。
    3. 然而，只有在类对象代表着"most-derived class"时，这些构造函数才 可能会被调用。一些支持这个行为的代码会被放进去 [2]（直观点说 就是，虚基类的构造由最外层类控制)。
2. 调用所有基类构造函数，依声明顺序：

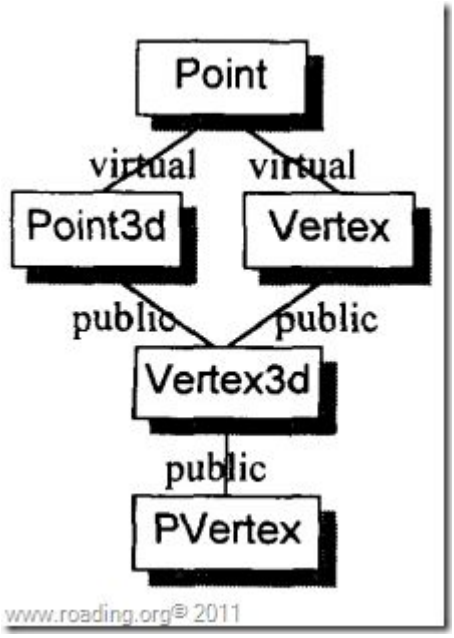
    1. 如果该基类被列入了成员初始化队列，那么所有明确指定的参数，应该 被传递过来。
    2. 没有列入的话，那么调用其默认构造函数，如果有的话。
    3. 如果该基类是第二顺位或之后的基类，this 指针必须被调整。

1. 正确初始化 vptr,如果有的话。

2. 调用没有出现在初始化成员列表中的 member object 的默认构造函数，如果 有的话。
3. 记录在成员初始化队列中的数据成员初始化操作以声明的顺序被放进构造函 数中。

虚拟继承下的构造抑制

有如下继承体系：



根据 c++ 语法，Point 的初始化应有 most-derived class 来施行。也就是说当 Vertex3d 为 most-derived class 的时候，应当由它的构造函数来调用 Point 的构 造函数初始化 Point 子对象，Vertex3d 的子对象的构造函数对于 Point 的调用则 应当抑制。如果没有抑制会怎么样?当我们定义 `Vertex3d cv;` 时，Vertex3d 的 构造函数中调用 Point 的构造函数、而随之调用它的子对象，Point3d 和 Vertex 的 构造函数中也调用了 Point 的构造函数。先不说，对于同一个子对象进行三次初 始化是否有效率，更重要的是，这将不可避免的带来错误。由 Vertex3d 指定的子 对象 Point 的值，会被覆盖掉。

编译器通常使用一个条件变量来表示是否为 most-derived class,各构造函数根 据这个条件变量来决定是否调用虚基类的构造函数，因此通过控制这个条件变量， 就可以抑制非 most-derived class 调用虚基类的构造函数。当然也有其它的方法 来做同样的事。

对象复制语意学

设计一个类，并考虑到要以一个对象指定给另一个对象时，有三种选择：

- 什么都不做，采用编译器提供默认行为（bitwise copy 或者由编译器合成 一个）。

- 自己提供一个赋值运算符操作。
- 明确拒绝将一个对象指定给另一个对象。

对于第三点，只要将赋值操作符声明为 private，且不定义它就可以了。对于第 二点，只有在第一点的行为不安全或不正确，或你特别想往其中插入点东西的时 候。

以下四种情况 copy assignment operator(还是用它的英文名，感觉顺畅点)，不 具有 bitwise copy 语意，也就是说这些情况下，编译器要合成 copy assignment operator 而不能依靠 bitwise copy 来完成赋值操作，这四种情况与构造函数、拷贝构造函数的情况类似，原因可以参考它们的。四种情况如下：

- 类包含有定义了 copy assignment operator 的 class object 成员。
- 类的基类有 copy assignment operator。
- 类声明有任何虚函数的时候（问题同样会出现在由继承类对象向基类对象拷贝 的时候）。
- 当 class 继承体系中有虚基类时。

在虚拟继承情况下，copy assignment opertator 会遇到一个不可避免的问题， virtual base class subobject 的复制行为会发生多次，与前面说到的在虚拟继承 情况下虚基类被构造多次是一个意思，不同的是在这里不能抑制非 most-derived class 对 virtual base class 的赋值行为。

安全的做法是把虚基类的赋值放在最后，避免被覆盖。

对象析构语意学

只有在基类拥有析构函数，或者 object member 拥有析构函数的时候，编译器才为类 合成析构函数，否则都被视为不需要。

析构的顺序正好与构造相反：

- 本身的析构函数被执行。
- 以声明的相反顺序调用 member object 的析构函数，如果有的话。
- 重设 vptr 指向适当的基类的虚函数表，如果有的话。
- 以声明相反的顺序调用上一层的析构函数，如果有的话。
- 如果当前类是 most-derived class，那么以构造的相反顺序调用虚基类的析构函数。

下面的叙述顺序看似与原书的顺序不一样。实际顺序并没有被调整，很多个“在此之前”的叙述并不适合我，我喜欢很直白的方式，按顺序来。书中的方 式在于，从最浅显的步骤入手，然后告诉你，做这步之前，你还该做点什么。 ↩

这一点上我参考英文版后，感觉侯大大的翻译并没有完全表达 Lippman 的原意， 所以，我以对原文的理解写下这点。Lippman 的原文为：

> *These constructors, however, may be invoked if, and only if, the class object represents the "most-derived class." Some mechanism supporting this must be put into place.*

侯捷的译文为：

> *如果 class object 是最底层（most-derived）的 class,其 constructors 可能被调用； 某些用以支持这个行为的机制必须被放进来。*

我认为，Lippman 在这一句上要说的是，虚基类的构造函数只能由 most-derived class 调用，而为了支持这一机制，需要插入一些代码来抑制非 most-derived class 对虚基类 构造函数的调用。同时说一点，5.4 的标题个人以为应该译为"对象的效率"而非"对象的 功能"——原标题为：Object Efficency。 ↩