拷贝构造函数（copy constuctor）

拷贝构造函数（copy constuctor）

通常 C++初级程序员会认为当一个类为没有定义拷贝构造函数的时候，编译器会 为其合成一个，答案是否定的。编译器只有在必要的时候在合成拷贝构造函数。 那么编译器什么时候合成，什么时候不合成，合成的拷贝构造函数在不同情况下 分别如何工作呢？这是本文的重点。

拷贝构造函数的定义

有一个参数的类型是其类类型的构造函数是为拷贝构造函数。如下：

```
X::X( const X& x);Y::Y( const Y& y, int =0 );//可以是多参数形式，但其第二个即后继参数都有一个默认值
```

拷贝构造函数的应用

当一个类对象以另一个同类实体作为初值时，大部分情况下会调用拷贝构造函数。 一般是这三种具体情况：

- 显式地以一个类对象作为另一个类对象的初值，形如 `X xx=x;`
- 当类对象被作为参数交给函数时。
- 当函数返回一个类对象时。

后两种情形会产生一个临时对象。

编译器何时合成拷贝构造函数

并不是所有未定义有拷贝构造函数的类编译器都会为其合成拷贝构造函数，编译 器只有在必要的时候才会为其合成拷贝构造函数。所谓必要的时刻是指编译器在 普通手段无法完成解决“当一个类对象以另一个同类实体作为初值”时，才会合成 拷贝构造函数。也就是说，当常规武器能解决问题的时候，就没必要动用非常规 武器。

如果一个类没有定义拷贝构造函数，通常按照“成员逐一初始化(Default Memberwise Initialization)”的手法来解决“一个类对象以另一个同类实体作为 初值”——也就是说把内建或派生的数据成员从某一个对象拷贝到另一个对象身上， 如果数据成员是一个对象，则递归使用“成员逐一初始化(Default Memberwise Initialization)”的手法。

成员逐一初始化(Default Memberwise Initialization)具体的实现方式则是位 逐次拷贝（Bitwise copy semantics）[1]。也就是说在能使用这种常规方式 来解决"一个类对象以另一个同类实体作为初值"的时候，编译器是不需要合成拷 贝构造函数的。但有些时候常规武器不那么管用，我们就得祭出非常规武器了 ——拷贝构造函数。有以下几种情况之一，位逐次拷贝将不能胜任或者不适合来完 成"一个类对象以另一个同类实体作为初值"的工作。此时，如果类没有定义拷贝 构造函数，那么编译器将必须为类合成一个拷贝构造函数。

- 当类内含一个成员对象，而后者的类声明有一个拷贝构造函数时（不论是设 计者定义的还是编译器合成的）。
- 当类继承自一个声明有拷贝构造函数的类时（同样，不论这个拷贝构造函数 是被显示声明还是由编译器合成的）。
- 类中声明有虚函数。
- 当类的派生串链中包含有一个或多个虚基类。

对于前两种情况，不论是基类还是对象成员，既然后者声明有拷贝构造函数时， 就表明其类的设计者或者编译器希望以其声明的拷贝构造函数来完成"一个类对象 以另一个同类实体作为初值"的工作，而设计者或编译器这样做——声明拷贝构造函 数，总有它们的理由，而通常最直接的原因莫过于因为他们想要做一些额外的工 作或"位逐次拷贝"无法胜任。

对于有虚函数的类，如果两个对象的类型相同那么位逐次拷贝其实是可以胜任的。 但问题将出现在，如果基类由其继承类进行初始化时，此时若按照位逐次拷贝来 完成这个工作，那么基类的 vptr 将指向其继承类的虚函数表，这将导致无法预料 的后果——调用一个错误的虚函数实体是无法避免的，轻则带来程序崩溃，更糟糕 的问题可能是这个错误被隐藏了。所以对于有虚函数的类编译器将会明确的使被 初始化的对象的 vptr 指向正确的虚函数表。因此有虚函数的类没有声明拷贝构造 函数，编译将为之合成一个，来完成上述工作，以及初始化各数据成员，声明有 拷贝构造函数的话也会被插入完成上述工作的代码。

对于继承串链中有虚基类的情况，问题同样出现在继承类向基类提供初值的情况， 此时位逐次拷贝有可能破坏对象中虚基类子对象的位置。

Bitwise copy semantics 是 Default Memberwise Intializiation 的具体实现方式。也没有在任何地方对这两个名词进行对比或者更多的阐述，这 一度使我疑惑，上面只是我个人的理解，你有任何不同的见解，欢迎你与我讨论。 (2011/12/22 日 补) ↩