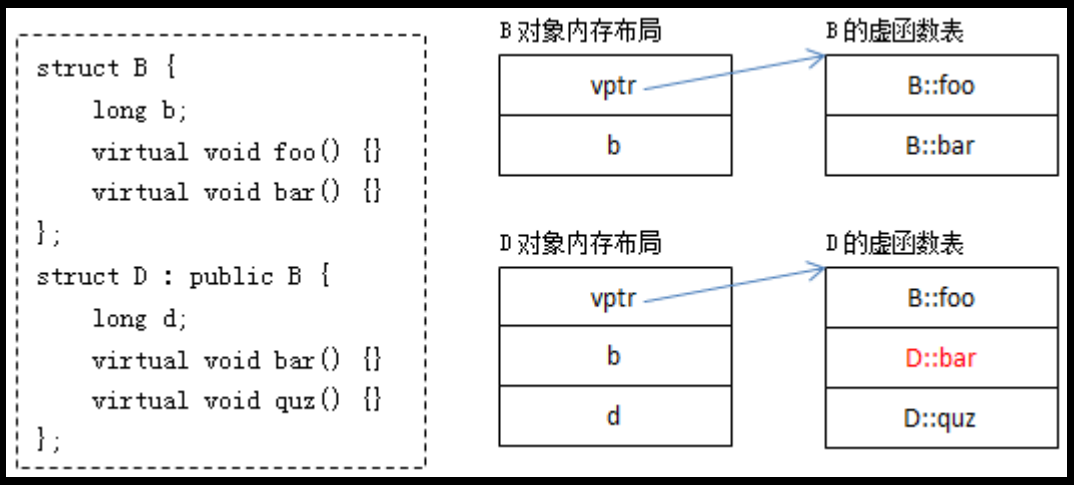
C++原理深入

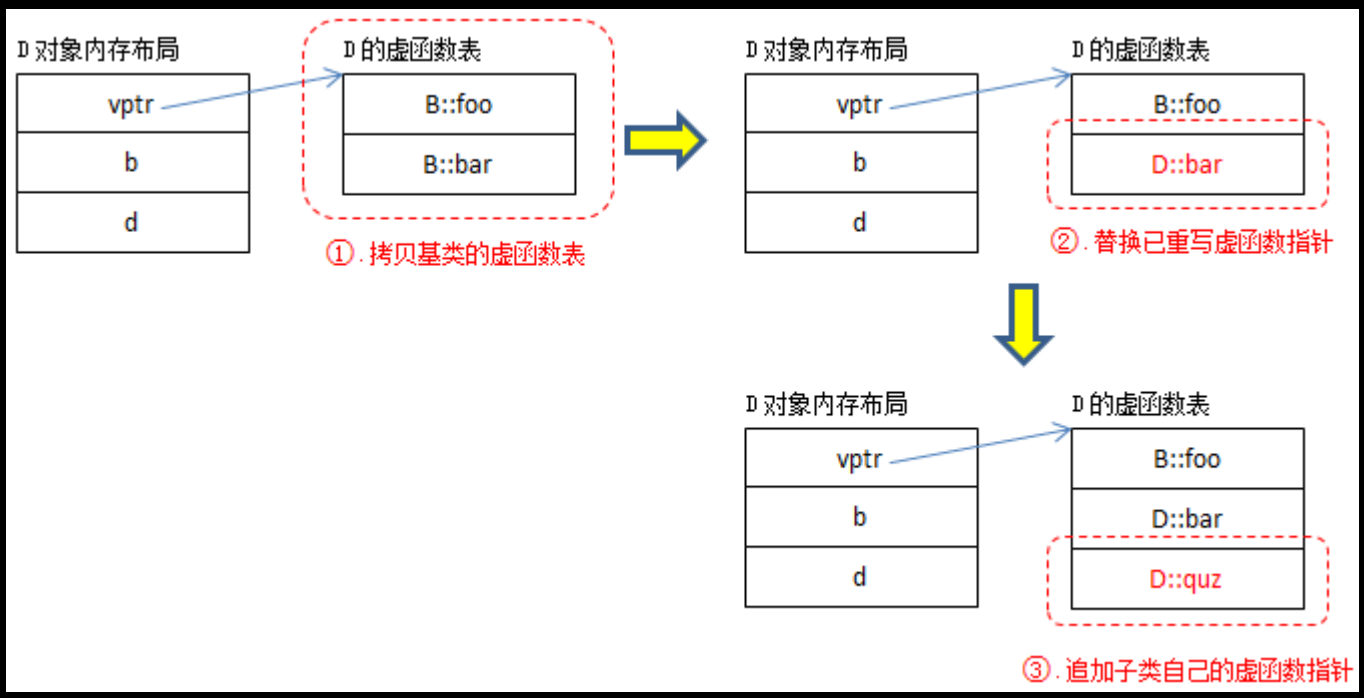
虚函数原理

每一个含有虚函数（无论是其本身的，还是继承而来的）的类都至少有一个与之对应的虚函数表，其中存放着该类所有的虚函数对应的函数指针。例：



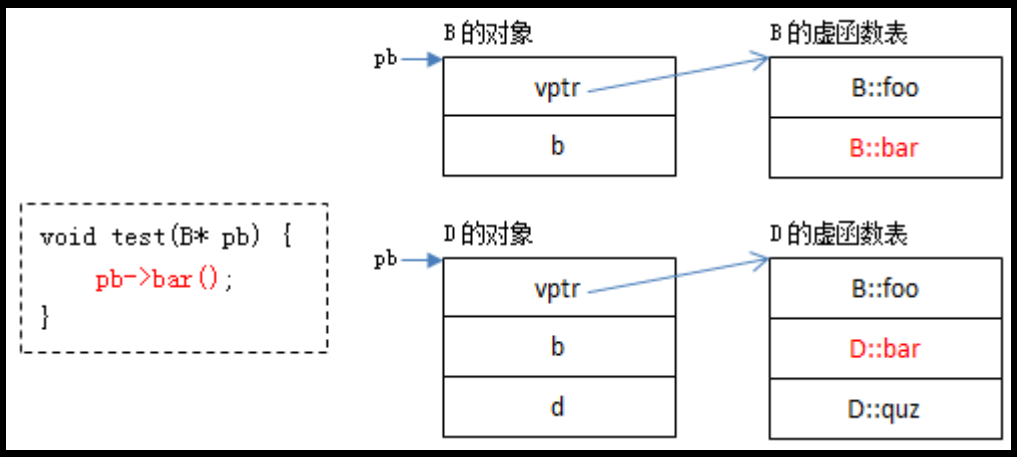
B 的虚函数表中存放着 B::foo 和 B::bar 两个函数指针。
D 的虚函数表中存放的既有继承自 B 的虚函数 B::foo，又有重写（override）了基类虚函数 B::bar 的 D::bar，还有新增的虚函数 D::quz。

从编译器的角度来说，B 的虚函数表很好构造，D 的虚函数表构造过程相对复杂。下面给出了构造 D 的虚函数表的一种方式



虚函数替换过程发生在编译时。

虚函数调用过程



编译器只知道 pb 是 B*类型的指针，并不知道它指向的具体对象类型 ：pb 可能指向的是 B 的对象，也可能指向的是 D 的对象。

但对于“pb->bar()”，编译时能够确定的是：此处 operator->的另一个参数是 B::bar（因为 pb 是 B*类型的，编译器认为 bar 是 B::bar），**而 B::bar 和 D::bar 在各自虚函数表中的偏移位置是相等的**。

无论 pb 指向哪种类型的对象，只要能够确定被调函数在虚函数中的偏移值，待运行时，能够确定具体类型，并能找到相应 vptr 了，就能找出真正应该调用的函数。

虚函数指针中的 ptr 部分为虚函数表中的偏移值（以字节为单位）加 1。

当程序执行到“pb->bar()”时，已经能够判断 pb 指向的具体类型了：

1. 如果 pb 指向 B 的对象，可以获取到 B 对象的 vptr，加上偏移值 8（(char*)vptr + 8），可以找到 B::bar。
2. 如果 pb 指向 D 的对象，可以获取到 D 对象的 vptr，加上偏移值 8（(char*)vptr + 8），可以找到 D::bar。

实现多态的关键点：

重写的方法在虚函数表中的偏移是一样的，给出偏移，从具体的虚函数表中找到对应的方法，则可以产生具体的调用。