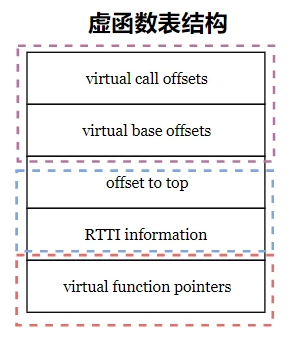
**总结**

虚表结构大体如上图，正常的虚表结构中都含有后三项，当有继承情况才会有前两个表项。



**offset\_to\_top**：对象在对象布局中与对象顶部地址的偏移量。

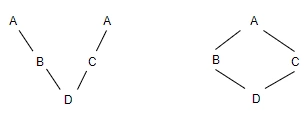
**RTTI指针**：指向存储运行时类型信息(type\_info)的地址，用于运行时类型识别，用于typeid和dynamic\_cast。

**vbase\_offset**：对象在对象布局中与指向虚基类虚函数表的指针地址的偏移量。

**vcall\_offset**：父类引用或指针指向子类对象，调用被子类重写的方法时，用于对虚函数执行指针地址调整，方便成功调用被重写的方法。

**thunk**：表示上面虚函数表中带有adjustment字段的函数调用需要先进行this指针调整，才可以调用到被子类重写的函数。

**为什么要虚继承**



非虚继承时，显然D会继承两次A，内容就会存储两份A的数据浪费空间，而且还有二义性，D调用A的方法时，由于有两个A，究竟调用哪个A的方法，编译器也不知道，就会报错，所以有了虚继承，解决了空间浪费以及二义性问题。在虚继承下，只有一个共享的基类子对象被继承，而无论该基类在派生层次中出现多少次。共享的基类子对象被称为虚基类。在虚继承下，基类子对象的复制及由此而引起的二义性都被消除了。

**为什么虚函数表中有两个析构函数**

虚函数表中有两个析构函数，一个标志为deleting，一个标志为complete，因为对象有两种构造方式，栈构造和堆构造，所以对应的实现上，对象也有两种析构方式，其中堆上对象的析构和栈上对象的析构不同之处在于，栈内存的析构不需要执行delete函数，会自动被回收。

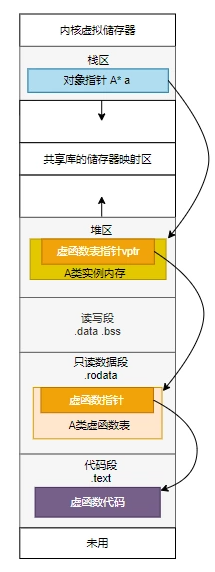
**为什么构造函数不能是虚函数**

构造函数就是为了在编译阶段确定对象的类型以及为对象分配空间，如果类中有虚函数，那就会在构造函数中初始化虚函数表，虚函数的执行却需要依赖虚函数表。如果构造函数是虚函数，那它就需要依赖虚函数表才可执行，而只有在构造函数中才会初始化虚函数表，鸡生蛋蛋生鸡的问题，很矛盾，所以构造函数不能是虚函数。

**为什么基类析构函数要是虚函数**

一般基类的析构函数都要设置成虚函数，因为如果不设置成虚函数，在析构的过程中只会调用到基类的析构函数而不会调用到子类的析构函数，可能会产生内存泄漏。

对象在Linux中的布局 A\* a = mew Derive();//A为Derive的基类



a作为对象指针存储在栈中，指向在堆中的类A的实例内存，其中实例内存布局中有虚函数表指针，指针指向的虚函数表存放在数据段中，虚函数表中的各个函数指针指向的函数在代码段中。