[**C++ 中类的内存布局**](http://www.cnblogs.com/jiaochen/p/5524335.html)

在许多笔试面试中都会涉及到sizeof 运算符的求值问题。

这类问题主要分四类：

1. 基本数据类型，如int，bool，fload，long，long，int \* 等，这一类比较简单，但要注意x86和x64情况下的指针大小
2. 枚举 enum。这个类型网络上有说是1-4个byte，根据最大值决定的；也有说是sizeof(int)。我这边个人使用 visual studio 2015 获得的结果是4个byte
3. struct 和 union 组合类型。union 是取其中一个最大成员的size作为其size；struct 则要考虑对齐填充因素
4. class 类型，class 就稍微复杂点，不仅仅要考虑对齐填充因素，还要考虑继承，虚继承，虚函数等因素。

下文主要讲述class 的内存布局，稍带介绍一下struct 的size。

**struct 的内存布局：**

struct 的内存对齐和填充概念学过C 的都应该知道一点。其实只要记住一个概念和三个原则就可以了：

一个概念：

**自然对齐**：如果一个变量的内存地址正好位于它长度的整数倍，就被称做自然对齐。

　　如果不自然对齐，会带来CPU存取数据时的性能损失。（PS：具体应该与CPU通过总线读写内存数据的细节相关，具体没有细究）

三个原则：

1. struct 的起始地址需要能够被其成员中最宽的**基本数据类型**整除；
2. struct 的size 也必须能够被其成员中最宽的**基本数据类型**整除；
3. struct 中每个成员地址相对于struct 的起始地址的offset，必须是自然对齐的。

**Class 的内存布局：**

在学习C++ 的class 的内存布局前，先介绍下文会被用到的Visual studio 中的编译选项"/d1reportAllClassLayout" 和 "/d1reportSingleClassLayout[ClassName]"。

这两个编译选项分别会输出当前编译单元中所以class 的内存布局和指定class 的内存布局。对于学习class 的内存布局很方便。

关于一个class 的定义，在定义过程中涉及到的有：

　　成员数据（静态，非静态）和成员函数（静态，非静态，virtual）。

所有的成员函数都不会占用对象的存储空间，无论是静态，非静态还是虚函数。

而对于成员数据来说，只有非静态的数据才会占用对象的存储空间。

这个很好理解，静态成员数据和成员函数是属于class 的，而非属于具体的对象，所以只要维护一份内存就可以了，无需每个对象都拷贝一份。

但是影响对象的大小的因素并不仅仅与看到的成员变量有关：

**非静态成员变量，虚函数表指针(\_vftprt)，虚基类表指针(\_vbtptr)，上文的内存对齐**

* **空类**

|  |  |
| --- | --- |
| 1 | class CEmpty{}; |

　　对于空类，许多人想当然的认为大小应该是0。这是错误的，如果是正确的话，这个类可以被实例化成一个对象，且这个对象不占任何存储空间，且可以有很多不占任何空间的对象，而且这个不占空间的对象还可以有指针，这样就很奇怪了。

　　所以正常编译器会给空类分配1个byte 的空间用于标示。

　　sizeof(CEmpty) = 1

* **普通类**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class CBase {  public:      int m\_ia;      static int s\_ib;  private:      void f();      void g();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4 | class CBase size(4):      +---   0  | m\_ia      +--- |

　　只有m\_ia 成员，size 为4个byte。因为静态数据成员和成员函数不占有对象空间。

* **有虚函数的类**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class CBase {  public:      int m\_ia;  private:      void f();      void g();      virtual void h();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class CBase size(8):      +---   0  | {vfptr}   4  | m\_ia      +---    CBase::$vftable@:      | &CBase\_meta      |  0   0  | &CBase::h |

　　可以看到该类的起始地址是放了一个"vfptr"，这个指针用来指向该类的虚函数表。

* **单一继承的类(无虚函数)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class CBase {  public:      int m\_ia;  private:      void f();      void g();  };    class CChild :public CBase {  public:      int m\_iChild;  }; |

　　类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class CChild    size(8):      +---      | +--- (base class CBase)   0  | | m\_ia      | +---   4  | m\_iChild      +--- |

　　即派生类中拷贝了一份基类中的成员数据，所以size 为8个byte。

* **单一继承的类（含有虚函数）**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class CBase {  public:      int m\_ia;  public:      virtual ~CBase();      virtual void f();      virtual void g();  };    class CChild :public CBase {  public:      int m\_iChild;  public:      virtual ~CChild();      virtual void g();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class CChild    size(12):      +---      | +--- (base class CBase)   0  | | {vfptr}   4  | | m\_ia      | +---   8  | m\_iChild      +---    CChild::$vftable@:      | &CChild\_meta      |  0   0  | &CChild::{dtor}   1  | &CBase::f   2  | &CChild::g |

　　可以看到派生类中只有一个"vfptr"，但是虚函数表中的函数却不同于基类中的函数，没有重写的虚函数沿用基类中的虚函数，而被重写的虚函数则更新为派生类中的虚函数。

* **多重继承的类（基类都含有虚函数）**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | class CBase1 {  public:      int m\_i1;  public:      virtual ~CBase1();      virtual void f1();      virtual void g1();  };    class CBase2 {  public:      int m\_i2;  public:      virtual ~CBase2();      virtual void f2();      virtual void g2();  };    class CChild :public CBase1, public CBase2 {  public:      int m\_iChild;  public:      virtual ~CChild();      virtual void f1();      virtual void g2();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | class CChild    size(20):      +---      | +--- (base class CBase1)   0  | | {vfptr}   4  | | m\_i1      | +---      | +--- (base class CBase2)   8  | | {vfptr}  12  | | m\_i2      | +---  16  | m\_iChild      +---    CChild::$vftable@CBase1@:      | &CChild\_meta      |  0   0  | &CChild::{dtor}   1  | &CChild::f1   2  | &CBase1::g1    CChild::$vftable@CBase2@:      | -8   0  | &thunk: this-=8; goto CChild::{dtor}   1  | &CBase2::f2   2  | &CChild::g2 |

　　CChild 分别从CBase1 和 CBase 中继承一个vfptr.

* **菱形结构继承的类（非虚继承）**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | class CBase {  public:      int m\_iBase;  public:      virtual ~CBase();      virtual void f0();      virtual void g0();      virtual void h0();  };    class CChild1:public CBase {  public:      int m\_iChild1;  public:      virtual ~CChild1();      virtual void f0();      virtual void h1();  };    class CChild2:public CBase {  public:      int m\_iChild2;  public:      ~CChild2();      void g0();      void h1();  };    class CGrandChild :public CChild1, public CChild2 {  public:      int m\_iGrandChild;  public:      virtual ~CGrandChild();      virtual void h0();      virtual void h1();      virtual void h2();      virtual void f0();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | class CGrandChild   size(28):      +---      | +--- (base class CChild1)      | | +--- (base class CBase)   0  | | | {vfptr}   4  | | | m\_iBase      | | +---   8  | | m\_iChild1      | +---      | +--- (base class CChild2)      | | +--- (base class CBase)  12  | | | {vfptr}  16  | | | m\_iBase      | | +---  20  | | m\_iChild2      | +---  24  | m\_iGrandChild      +---    CGrandChild::$vftable@CChild1@:      | &CGrandChild\_meta      |  0   0  | &CGrandChild::{dtor}   1  | &CGrandChild::f0   2  | &CBase::g0   3  | &CGrandChild::h0   4  | &CGrandChild::h1   5  | &CGrandChild::h2    CGrandChild::$vftable@CChild2@:      | -12   0  | &thunk: this-=12; goto CGrandChild::{dtor}   1  | &thunk: this-=12; goto CGrandChild::f0   2  | &CChild2::g0   3  | &thunk: this-=12; goto CGrandChild::h0 |

　　这种继承是有风险的，即通过CGrandChild 去访问m\_iBase 时，容易造成二义性，需要使用"pGrandChild->CChild::m\_iBase" 这种方法去访问。

　　为了避免这种问题，C++ 中有一种机制是**虚继承**。

* **单一虚继承**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class CBase {  public:      int m\_iBase;  public:      virtual ~CBase();      virtual void f0();      virtual void g0();      virtual void h0();  };    class CChild1: virtual public CBase {  public:      int m\_iChild1;  public:      virtual ~CChild1();      virtual void f0();      virtual void h1();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | class CChild1   size(24):      +---   0  | {vfptr}   4  | {vbptr}   8  | m\_iChild1      +---  12  | (vtordisp for vbase CBase)      +--- (virtual base CBase)  16  | {vfptr}  20  | m\_iBase      +---    CChild1::$vftable@CChild1@:      | &CChild1\_meta      |  0   0  | &CChild1::h1    CChild1::$vbtable@:   0  | -4   1  | 12 (CChild1d(CChild1+4)CBase)    CChild1::$vftable@CBase@:      | -16   0  | &(vtordisp) CChild1::{dtor}   1  | &(vtordisp) CChild1::f0   2  | &CBase::g0   3  | &CBase::h0 |

　　从布局中看，发现多了一个vbptr 指针，则是一个指向基类的虚基类指针；在派生类和虚基类之间又多了“vtordisp for vbase CBase”，vtordisp 并不是每个虚继承的派生类都会生成的，关于这部分可以参考MSDN 中 vtordisp；在vtordisp 后面则是虚基类的一个拷贝。

* **多重继承的类（虚继承）**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | class CChild1 {  public:      int m\_iChild1;  public:      virtual ~CChild1();      virtual void f0();      virtual void h1();  };    class CChild2 {  public:      int m\_iChild2;  public:      ~CChild2();      void g0();      void h1();  };    class CGrandChild :public CChild1, public CChild2 {  public:      int m\_iGrandChild;  public:      virtual ~CGrandChild();      virtual void h0();      virtual void h1();      virtual void h2();      virtual void f0();  }; |

　　virtual public Child1, public CChild2

　　　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class CGrandChild   size(28):      +---   0  | {vfptr}      | +--- (base class CChild2)   4  | | m\_iChild2      | +---   8  | {vbptr}  12  | m\_iGrandChild      +---  16  | (vtordisp for vbase CChild1)      +--- (virtual base CChild1)  20  | {vfptr}  24  | m\_iChild1      +--- |

　　public Child1, virtual public CChild2

　　　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class CGrandChild   size(20):      +---      | +--- (base class CChild1)   0  | | {vfptr}   4  | | m\_iChild1      | +---   8  | {vbptr}  12  | m\_iGrandChild      +---      +--- (virtual base CChild2)  16  | m\_iChild2      +--- |

　　virtual public Child1, virtual public CChild2

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class CGrandChild   size(28):      +---   0  | {vfptr}   4  | {vbptr}   8  | m\_iGrandChild      +---  12  | (vtordisp for vbase CChild1)      +--- (virtual base CChild1)  16  | {vfptr}  20  | m\_iChild1      +---      +--- (virtual base CChild2)  24  | m\_iChild2      +--- |

　　通过上述虚继承的情况来看，可以看出有虚继承的派生类中，派生类和虚基类的数据是完全隔开的，先存放派生类自己的虚函数指针，虚基类指针和数据；然后有vtordisp 作为间隔；在存放虚基类的内容。

* **菱形结构继承的类（虚继承）**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | class CBase {  public:      int m\_iBase;  public:      virtual ~CBase();      virtual void f0();      virtual void g0();      virtual void h0();  };    class CChild1 : virtual public CBase {  public:      int m\_iChild1;  public:      virtual ~CChild1();      virtual void f0();      virtual void h1();  };    class CChild2 : virtual public CBase{  public:      int m\_iChild2;  public:      virtual ~CChild2();      virtual void g0();      virtual void h1();  };    class CGrandChild : public CChild1, public CChild2 {  public:      int m\_iGrandChild;  public:      virtual ~CGrandChild();      virtual void h0();      virtual void h1();      virtual void h2();      virtual void f0();  }; |

　　其类的布局如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class CGrandChild   size(40):      +---      | +--- (base class CChild1)   0  | | {vfptr}   4  | | {vbptr}   8  | | m\_iChild1      | +---      | +--- (base class CChild2)  12  | | {vfptr}  16  | | {vbptr}  20  | | m\_iChild2      | +---  24  | m\_iGrandChild      +---  28  | (vtordisp for vbase CBase)      +--- (virtual base CBase)  32  | {vfptr}  36  | m\_iBase      +--- |

　　有了上文的基础，这个派生类的机构就不难理解了。