C++ 对象模型

jask

09/10/2024

深度探索 C++ 对象模型（总结篇）

C++ 自己的对象模式

C++ 对象采用一个连续的内存进行储存，一个内存大致会被分为两块，一块是数据本身的储存，一块是 virtual table 的储存

对于数据本身的存储

- static member 和 static function 与 nonstatic function 一样，都单独的储存在属于他们的内存中。
- nonstatic member data 一般会和虚函数表放在一起，可能会在虚函数表的前面或者后面。

对于 virtual table 的存储

- virtual table 里面储存大量的指针，这些指针包括
  - virtual 析构函数的指针
  - virtual member
  - 每一个 class 所关联的 type-info 指针
  - 虚拟继承的关系
- 使用 virtual table 带来的好处与坏处
  - 好处：真正使得 virtual function 得以实现，一并的也有 virtual class（保证多次继承只有一个实例）
  - 坏处：增加了访问成本，导致额外的时间与空间负担
- 一个 object 里面可以含有多个 virtual table，这个情况就是在多重继承的情况下。

Data member

布局

- 对于 nonstatic data member，他们会按照声明的顺序，依次进行创建和排列
- 对于 no data member 的 class，编译器回往里面塞上一个 char，确保他占用一个字节的空间，在运行的时候，编译器也会为他合成出来对应的 constructor 和 destructor
- 对于 static 类型的数据，他只有一个实例，并且存在 data segment 中，如果去他的地址，得到的是对于数据类型的指针，而与其所在的对象毫无关联
- 对于 function member，只有他们在需要的时候会被编译器合成出来，并且放在内存中，一般是与对应的 class 在连续的内存中

存取（通过指针和通过对象进行存取的区别）

- 对于 static data member，无论是通过指针还是通过对象都是一样的，因为他是单独放在 data segment 中。
- 对于 nonstatic data member
  - 一般来说通过对象存取的本质是通过 this 指针完成
  - 在内存上的时候，想要对一个 nonstatic data member 进行存取操作的时候，编译器需要将 class object 的起始地址加上 data member 的偏移位置
  - 从时间上看，从对象的角度来说存取一个 nonstatic data member，与 C 存取一个 struct member 是没有区别的
  - 但是在与 virtual class，virtual function 来说，使用指针和使用 class，存取速度就会有重大的差别，指针一般就会更加慢一些，class 则不然，原因之一就是其使用了 virtual table，其次就是因为编译时期我们不知道他到底是那种类型，需要在执行期间确定，也就是我们说的动态绑定

继承

- 一般来讲，非 virtual 的继承相对于比 virtual 的继承存取时间上不会增加额外的负担

- 对于单一或多次继承，without virtual
  - 对于单一继承或者是多次继承，无论是继承几次，其访问时间上是不会有很多差异的，基本与 C struct 相同
  - 对于单一继承或者是多次继承，在空间上，随着继承次数的增多，空间的变化往往要比 class 内部所含的数据所占的总内存要大，原因是他有需要用来进行内存对齐，填补的 bytes
- 对于带有 virtual function 的继承
  - 空间上，由于导入了一个和 class 有关的 virtual table，用来存放他申明每一个 virtual function，这样会导致空间的增加，
  - 时间上，也是因为 table 间接转换导致的效率低下也是可见的
  - 他也带来的加上 constructor 和 destcuctor，可以帮助他们设定 vptr 和销毁 vptr
- 对于多重继承
  - 对于多重继承，其时间上的差异与空间上的差异很多，因为它是有多个 class 继承到单一 class 的一个非自然继承
  - 在内存上，对于一个多重派生对象，一般来讲将起始地址给定一个基本的 base class，其后面的 class 都要加上或者减去一个对应的数字，用于修改地址
  - 在时间上，多重继承的对象，使用指针和对象进行访问的时间与单一继承带有 virtual 访问的时间相似
- 对于虚拟继承
  - 与多重继承不同，虚拟继承要支持一种 share 类型的存储方式（也就是所谓菱形储存）
  - 布局的策略一般是先安排好 derived class 的不变部分，再安排其共享部分

效率

- 对于封装：对于封装如果把优化开关打开，封装就不会带来执行期间的效率成本
- 对于聚合：单一的聚合操作也是一样，在优化开关打开的情况下，聚合也不会带来执行期间的效率成本
- 对于继承：非 virtual 的继承效率要远远好处 virtual。

指针

- 取得一个 nonstatic data member（例如：&A:: x）的地址，将会得到他在 class 中的 offset，
- 取一个绑定于真正 class object 身上的 data member（例如 &A.x ）的地址，将会得到真正的地址
- 如果进行优化，那么在运行的时候，非 virtual 继承使用指针或者是对象的存取效率是相同的
- virtual 继承随着虚拟函数增多而变大，额外的间接性会降低优化能力

Function

function 的底层的调用方式

- 对于 nonstatic member function:
  - 效率上，他与 nonmember function，static member 具有相同的效率，原因是 nonstatic member function 会被转换为 nonmember function。(this 指针)
  - 名称上，class 内部的 value 和 function 会拥有独一无二的姓名，一般是 class 名字 + 变量名
- 对于 virtual member function
  - 对于一个指针去调用 virtual member function，将会转化成指针指向对于的 virtual table 解引用再连接对应的成员指针

    ```
    ptr->a()==(*ptr->vptr[1])(ptr);
    //其中，vptr 也有可能有多个，并且可能会被重新转化成一个新的，独一无二的姓名
    ```

  - 在多重继承之下，所有的指针问题都需要在执行期间进行操作
- 静态成员函数
  - static memeber function 的特性是没有 this 指针，因此他不能直接存取 nonstatic members，不能被声明为 const ,volatile,virtual，不需要经过 class object 才能调用（但是可以通过 class object 调用，这是允许的）
  - 由于它缺乏 this 指针，因此差不多等同于 nonmember function
- inline 函数
  - 毫无疑问，inline 函数能一定幅度提升函数性能，与之而来的付出代价是程序体积的增大
  - 一般来说，编译器通过计算 assignments，function call，virtual function calls 的操作次数的综合计算 inline 函数的复杂性

Pointer-to-member-Function

- 对于一个 nonmember function，static member function 取地址的话，所取即所得
- 对于 member function 取地址的话并不是真实的地址，而是对应的 offset（上面有写）
- 对于一个 virtual member function 取地址的话，得到的是 virtual table 的索引值。

Constructor ,Destructor,copy

default constructor 合成

- default constructor：对于 default constructor，只有其在需要的时候才会出现，并不是所有的时候都会合成出来一个 default constructor，甚至是 class 本身都没有指定的 constructor 的情况下
- 如果 class 内部含有 value 且没有对应的 default constructor，那么编译器会帮你合成一个，合成的时机是当真正要用到 class 的时候
- 如果 class 中声明，继承有 virtual class/function，那么也编译器也会合成出 default constructor，以便正确初始化 class object 的 vptr 以及 virtual table
- 编译器合成出来的 default constructor 不会显式设定 class 每一个 data member 的默认值，编译器合成出来的只是编译器需要的，与是否初始化毫无关系

调用

- 如果 class A 内含一个或者一个以上的 member class objects，那么 class A 的每一个 constructor 都调用对应的 default constructor，如果不想让 A 调用 default constructor，那么记得在初始化列表中加入对应的初始化
- 如果设计者提供多个 constructor，但是其中都没有 default constructor 的情况下，那么编译器会扩展每一个 constructors，将用以所必要的 default constructor 加进去
- 不要把纯虚函数用在声明 constructor 上

什么时候使用初始化队列

- 当你初始化一个 reference member 时候
- 当你初始化一个 const member 时候
- 当你调用一个 base class constructor，而他有参数的时候
- 当你调用一个 member class 的 constructor，而它拥有一组参数时
- 注意：list 中的项目顺序是由 class 中 member 声明顺序决定的，不是由初始化队列中的排列顺序决定的

copy constructor 合成

- 所谓的 default copy constructor，也就是 bitwise copy，这就是默认的拷贝构造
- 在 class 不展现 bitwise copy 的情况下，默认的 copy constructor 才会被编译器产生出来

调用

- 三种可能的调用动作：X xx=x，作为函数初值，return xx 1. 如果一个类没有拷贝构造函数，但该类含有一个类类型的成员变量，类类型含有拷贝构造。

2. 如果一个类 CTBSon 没有拷贝构造函数，但是有一个父类 CTB，父类 CTb 含有拷贝构造函数。

3. 如果一个类 CTBSOn 没有拷贝构造函数，但是该类定义了虚函数或者该类的父类定义了虚函数。

4. 没有构造函数，但是该类含有虚基类。

什么时候不展现 bitwise copy

- 当 class 内含有一个 member object 但是后者 class 声明有一个 copy constructor 时候。
- 当 class 继承自一个 base class 而后者存在一个 copy constructor 时
- 当 class 声明了一个或多个 virtual function 时
- 当 class 派生自一个继承串链，其中有一个或者多个 virtual base class

destructror

- destructor：如果 class 没有定义 destructor，那么只有在 class 内含有的 member object 拥有 destructor 的情况下，编译器才会自动合成出一个，否则 destructor 被视为不需要，也就不需要合成
- destructor 的调用顺序
  - destructor 的函数本体首先被执行
  - 如果 class 拥有 member class object 而后者拥有 destructor，那么他们会以其声明顺序相反顺序调用。
  - 如果 object 内含一个 vptr，现在被重新设定，指向适当的 base class 的 virtual table
  - 如果有任何直接上一层的 nonvirtual base classes 拥有 destructor，他们会以声明的顺序相反的顺序进行调用
  - 如果有任何 virtual base classes 拥有 destructor，且目前这个 class 是最尾端的 class 那么他们会以其原来的构造顺序相反的顺序被调用