知乎

c++虚函数表

无我
没什么特点

关注

48 人赞同了该文章 ›

起

一、概念

1、C++ Virtual Table (虚函数表) 是C++ 实现多态✦的方式。

2、每一个具有虚函数（使用virtual关键字定义的函数）的类的**具体实现**，都有一个指向虚函数的指针表。

3、指向虚函数表的**指针**是**作为数据成员存在于所有对象中**。当调用虚函数时，查找对象的虚函数表指向正确的派生类函数。 **虚函数表是由虚函数指针组成的数组**。

4、静态绑定 & 动态绑定。

1. 对于类的普通成员函数使用的是静态绑定，发生在**编译期**。
2. 对于类的虚函数发生在运行期是动态绑定，发生在**运行期**。
3. 虽然虚函数的调用是在运行期才确定，但是**虚函数表的创建是在编译阶段就完成构建**。

静态绑定 vs 动态绑定

二、简单例子

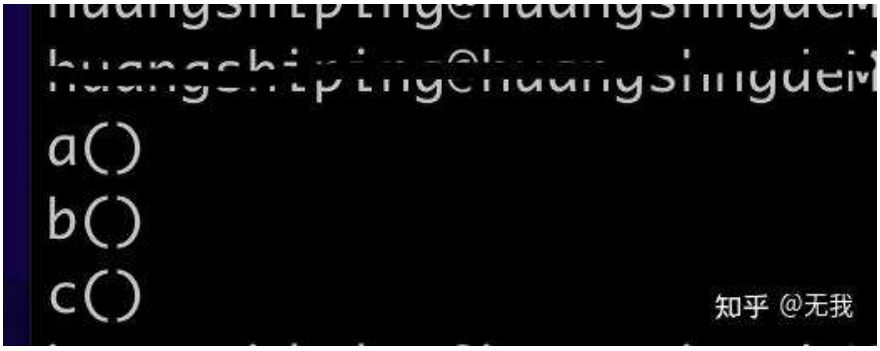
64位 & 32位 的寻址指针的大小分别是 int64_t & int

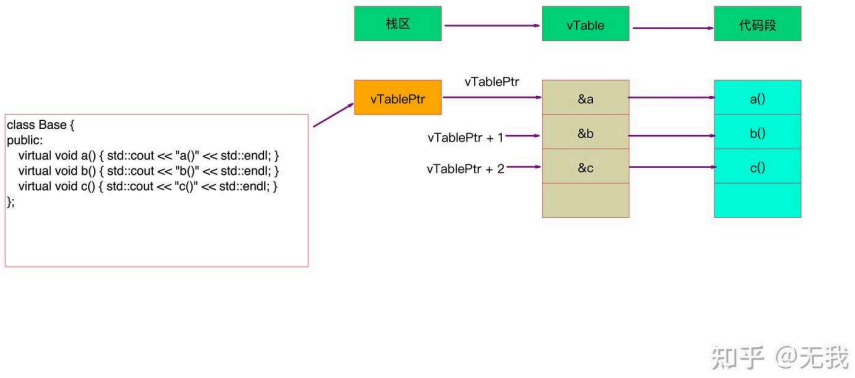
```
/**
 *64位机器测试例子
 */
#include <iostream>
class Base {
public:
        virtual void a() { std::cout << " a()" << std::endl; }
        virtual void b() { std::cout << " b()" << std::endl; }
        virtual void c() { std::cout << " c()" << std::endl; }
};

int main()
{
        Base base;
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&base)) + 0))   )
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&base)) + 1))   )
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&base)) + 2))   )
        return 0;
}
```

▲ 赞同 48 ▼ 4 条评论 分享 喜欢 收藏 申请转载 ···

```
 */
class Base {
public:
        virtual void a() { std::cout << "a()" << std::endl; }
        virtual void b() { std::cout << "b()" << std::endl; }
        virtual void c() { std::cout << "c()" << std::endl; }
};

int main()
{
        Base base;
        (   ((void(*())*((int*)(*((int*)&base)) + 0))   )     ();
        (   ((void(*())*((int*)(*((int*)&base)) + 1))   )     ();
        (   ((void(*())*((int*)(*((int*)&base)) + 2))   )     ();
        return 0;
}
```

运行结果：





虚函数指针&虚函数表

从上边的例子结果我们能对虚函数表有一个直观的认知。

三、单继承下的虚函数表

子类继承自Base. 未覆盖父类的虚函数。

```
class SubClass : public Base {
public:
        virtual void d() { std::cout << " d()" << std::endl; }
};
```

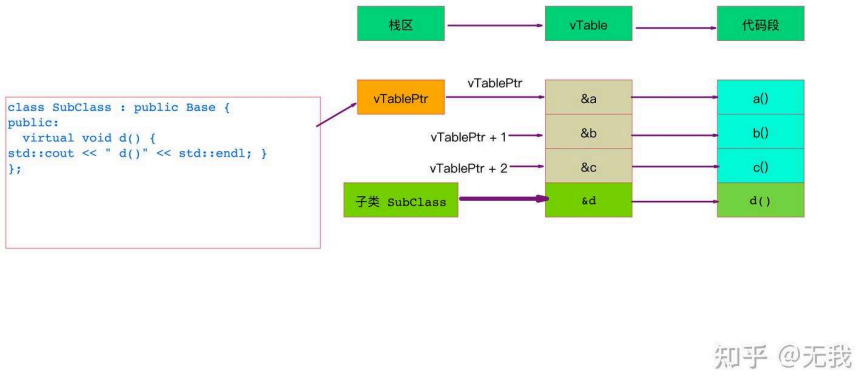
测试修改为：

```
        SubClass sub;
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&sub)) + 0))   )     (
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&sub)) + 1))   )     (
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&sub)) + 2))   )     (
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&sub)) + 3))   )     (
        return 0;
}
```

运行结果为：

结论：

1. 虚函数按照其**声明顺序**放于表中。
2. **父类的虚函数在子类的虚函数前面。**



子类虚函数未覆盖父类

子类继承自Base,覆盖父类虚函数

修改子类为：

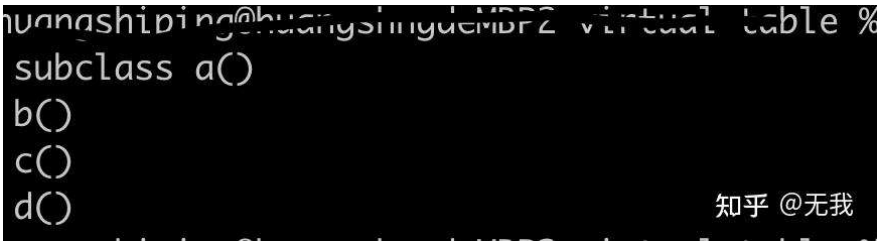
```
class SubClass : public Base {
public:
        virtual void a() { std::cout << " subclass a()" << std::endl; }
        virtual void d() { std::cout << " d()" << std::endl; }
};
```

测试用例保持：

```
int main()
{
        SubClass sub;
        (   ((void(*)())*((std::int64_t*)(*((std::int64_t*)&sub)) + 0))   )     (
        (   ((void(*)())*((std::int64_t*)(*((std
        (       ((void(*)())*((std::int64_t*)(*(
```

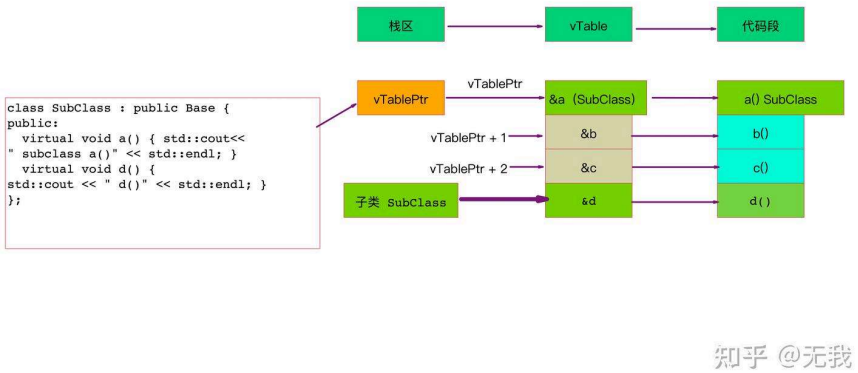
```
}
```

运行结果为：



结论：

1. **虚表中派生类覆盖的虚函数的地址被放在了基类相应的函数原来的位置** （SubClass的a()函数替换了Base的a()）
2. 派生类没有覆盖的虚函数延用父类的。



子类的虚函数覆盖了父类的虚函数

四、多继承下的虚函数表

无虚函数的覆盖

```
class Base {
public:
        virtual void vFunc1() { std::cout << "  Base::vFunc1" << std::endl; }
        virtual void vFunc2() { std::cout << "  Base::vFunc2" << std::endl; }
        virtual void vFunc3() { std::cout << "  Base::vFunc3" << std::endl; }
};

class Base2 {
public:
        virtual void vBase2Func1() { std::cout << "  Base2::vFunc1" << std::endl;
        virtual void vBase2Func2() { std::cout << "  Base2::vFunc2" << std::endl;
};

class Base3 {
public:
        virtual void vBase3Func1() { std::cout << "  Base3::vFunc1" << std::endl;
        virtual void vBase3Func2() { std::cout << "  Base3::vFunc2" << std::endl;
};



class SubClass : public Base , public Base2, pub
```

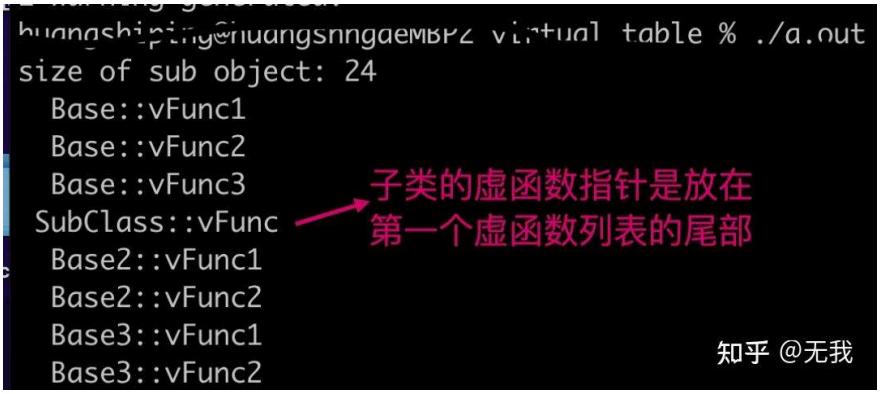
知乎

```
int main()
{
  SubClass sub;
  printf("size of sub object: %d \n", sizeof(sub));
  typedef void(*FUNCTION)();
  /**
    *Base有三个函数 vFunc1 & vFunc2 & vFunc3
    *Base2有两个函数 vFunc1 & vFunc2
    *Base3有两个函数 vFunc1 & vFunc2
    *从继承顺序知道虚函数表顺序是 vTablePtr1--->vTablePtr2--->vTablePtr3
    */

  /**
    *对象地址的前三个地址内容分别是指向vTable1的指针vTablePtr1 & 指向vTable2的指针vTabl
    * & 指向vTable3的指针vTablePtr3
    */

 /**
   *虚函数表一 Base vTablePtr1 (地址)
   */
  std::int64_t *subAddress = (std::int64_t*)&sub;
  std::int64_t *vTablePtr1 = (std::int64_t*)*(subAddress);

 /**
  *虚函数表二 Base2 vTablePtr2（地址）
  */
  std::int64_t *subAddress2 = (std::int64_t*)&sub + 1;
  std::int64_t *vTablePtr2 = (std::int64_t*)*(subAddress2);

 /**
   *虚函数表三 Base3 vTablePtr3 (地址)
   */

  std::int64_t *subAddress3 = (std::int64_t*)&sub + 2;
  std::int64_t *vTablePtr3 = (std::int64_t*)*(subAddress3);


  /**
   *定义测试函数
   */

  //Base
  std::int64_t *pBaseFunc1 = (std::int64_t *)*(vTablePtr1 + 0);
  std::int64_t *pBaseFunc2 = (std::int64_t *)*(vTablePtr1 + 1);
  std::int64_t *pBaseFunc3 = (std::int64_t *)*(vTablePtr1 + 2);

  //Base2
  std::int64_t *pBase2Func1 = (std::int64_t *)*(vTablePtr2 + 0);
  std::int64_t *pBase2Func2 = (std::int64_t *)*(vTablePtr2 + 1);

  //Base3
  std::int64_t *pBase3Func1 = (std::int64_t *)*(vTablePtr3 + 0);
  std::int64_t *pBase3Func2 = (std::int64_t *)*(vTablePtr3 + 1);

  /**
    *调用测试函数
    */

  //Base
  (FUNCTION(pBaseFunc1))();
  (FUNCTION(pBaseFunc2))();
  (FUNCTION(pBaseFunc3))();

  //Base2
```

知乎

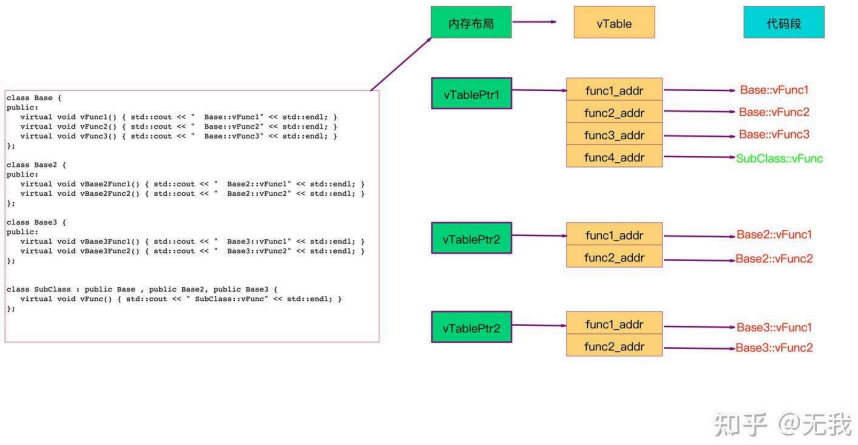
```
    //Base3
    (FUNCTION(pBase3Func1))();
    (FUNCTION(pBase3Func2))();
    return 0;
}
```

测试结果如下：



结论：

1. **每个基类都有自己的虚函数表**
2. **派生类的虚函数地址存依照声明顺序放在第一个基类的虚表最后**（这点和单继承无虚函数覆盖相同）



多继承无覆盖

有虚函数的覆盖

SubClass的代码修改为如下：

```
class SubClass : public Base , public Base2, public Base3 {
        /**
         *覆盖父类Base的vFunc1
         */
        virtual void vFunc1() { std::cout << " SubClass::vFunc1" << std::endl; }
        virtual void vFunc() { std::cout << " Sul
        /**
```

```
        virtual void vBase2Func1() { std::cout << "  SubClass::vFunc1" << std::en
};
```

运行结果如下：

结论：

1. 虚表中派生类覆盖的虚函数的地址被放在了基类相应的函数原来的位置。（相对于单继承，多继承依然是根据对应重写的函数来覆盖）
2. 派生类没有覆盖的虚函数延用基类的。

菱形继承

测试如下：

```
class Base {
public:
        virtual void vBaseFunc1() { std::cout << "  Base::vFunc1" << std::endl; }
        virtual void vBaseFunc2() { std::cout << "  Base::vFunc2" << std::endl; }
        virtual void vBaseFunc3() { std::cout << "  Base::vFunc3" << std::endl; }
};

class SubClass1 : public Base {
public:
        /**
         *继承自Base的vBaseFunc1
         */
        virtual void vBaseFunc1() { std::cout << "  SubClass1::Base::vFunc1" << s
        virtual void vSubClass1Func() { std::cout << "  SubClass1::vFunc" << std:
};

class SubClass2 : public Base {
public:
        /**
         *继承自Base的vBaseFunc2
         */
        virtual void vBaseFunc2() { std::cout << "  SubClass2::Base::vFunc2" << s
        virtual void vSubClass2Func() { std::cout << "  SubClass2::vFunc" << std:
};


class SubSubClass : public SubClass1, public SubClass2 {
public:
        /**
         *继承自SubClass1& SubClass2 的vBaseFunc1 :所以两个虚函数表都应该被修改
         */
        virtual void vBaseFunc1() { std::cout <<
```

关于作者

无我
没什么特点

回答	文章	关注者
6	86	618

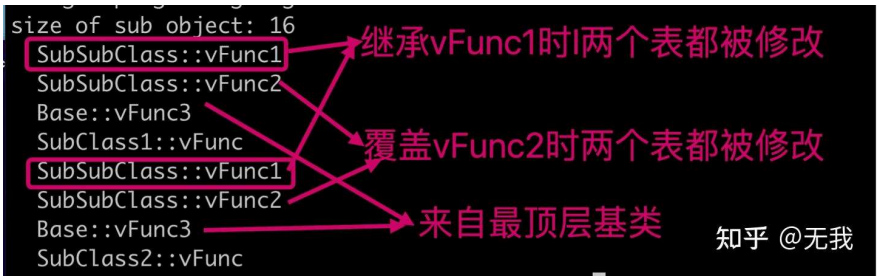
关注 发私信

知乎

```
         */
        virtual void vBaseFunc2() { std::cout << "  SubSubClass::vFunc2" << std::

        virtual void vSubSubClassFunc() { std::cout << "  SubSubClass::vFunc" << 


};


int main()
{
        SubSubClass subsub;


        printf("size of sub object: %d \n", sizeof(subsub));

        typedef void(*FUNCTION)();

         /**
          *虚函数表一 Base vTablePtr1 (地址)
          */
        std::int64_t *subAddress = (std::int64_t*)&subsub;
        std::int64_t *vTablePtr1 = (std::int64_t*)*(subAddress);


        /**
         *虚函数表二 Base2 vTablePtr2（地址）
         */
    std::int64_t *subAddress2 = (std::int64_t*)&subsub + 1;
        std::int64_t *vTablePtr2 = (std::int64_t*)*(subAddress2);


        /**
         *定义测试函数
         */

        //SubClass1
        std::int64_t *pBaseFunc1 = (std::int64_t *)*(vTablePtr1 + 0);
        std::int64_t *pBaseFunc2 = (std::int64_t *)*(vTablePtr1 + 1);
        std::int64_t *pBaseFunc3 = (std::int64_t *)*(vTablePtr1 + 2);
        std::int64_t *pBaseFunc4 = (std::int64_t *)*(vTablePtr1 + 3);

    //SubClass2
        std::int64_t *pBase2Func1 = (std::int64_t *)*(vTablePtr2 + 0);
        std::int64_t *pBase2Func2 = (std::int64_t *)*(vTablePtr2 + 1);
        std::int64_t *pBase2Func3 = (std::int64_t *)*(vTablePtr2 + 2);
        std::int64_t *pBase2Func4 = (std::int64_t *)*(vTablePtr2 + 3);



        /**
         *SubClass1继承自Base有三个虚函数&自身定义了一个虚函数
         */

        (FUNCTION(pBaseFunc1))();
        (FUNCTION(pBaseFunc2))();
        (FUNCTION(pBaseFunc3))();
        (FUNCTION(pBaseFunc4))();

    /**
     *SubClass2 继承自Base有三个虚函数&自身定义了一个虚函数
     */
        (FUNCTION(pBase2Func1))();
        (FUNCTION(pBase2Func2))();
        (FUNCTION(pBase2Func3))();
        (FUNCTION(pBase2Func4))();
```

```
}
```

输出结果：



原则

单继承

1. 虚函数表派生类覆盖的虚函数的地址被放在了基类虚函数表对应的函数原来的位置。（覆盖）
2. 派生类没有覆盖的虚函数就延用基类的。同时，虚函数按照其声明顺序放于表中，**父类的虚函数在子类的虚函数前面。**

多继承

1. 每个基类都有自己的虚函数表
2. 派生类的虚函数地址存依照声明顺序**放在第一个基类的虚表最后**.

参考：

[20] Inheritance -- virtual functions, C++ FAQ Lite

stackoverflow.com/quest...

18.6 - The virtual table

编辑于 2023-03-04 23:26 · 北京

C++ C / C++ Modern C++