

陈皓专栏【空谷幽兰，心如皓月】

芝兰生于深谷，不以无人而不芳；君子修道立德，不为困穷而改节。

目录视图

摘要视图

RSS 订阅

我的BLOG

陈皓专栏（技术）(RSS)

酷壳（编程和技术）(RSS)

个人资料



haoel

访问：4844610次

积分：26756

等级：BLOG 7

排名：第195名

原创：120篇 转载：6篇

译文：15篇 评论：5484条

文章搜索

文章分类

技术趋势 (13)

抄袭事件 (7)

编程工具 (19)

编程语言 (58)

职业心情 (23)

软件开发 (28)

项目管理 (9)

文章存档

2011年04月 (1)

2011年02月 (3)

2010年09月 (1)

2010年08月 (2)

2010年07月 (5)

展开

阅读排行

【活动】Python创意编程活动开始啦!!! CSDN日报20170427 ——《如何在没有实际项目经验的情况下找到工作》 深入浅出，带你学习 Unity

深入浅出

C++ 虚函数表解析

标签：c++ fun class 编译器 语言 iostream

2007-12-18 22:07 298069人阅读 评论(438) 收藏 举报

本文章已收录于： C++知识库

分类：编程语言 (57)

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

C++ 虚函数表解析

陈皓

http://blog.csdn.net/haoel

前言

C++中的虚函数的作用主要是实现了多态的机制。关于多态，简而言之就是用父类型别的指针指向其子类的实例，然后通过父类的指针调用实际子类的成员函数。这种技术可以让父类的指针有“多种形态”，这是一种泛型技术。所谓泛型技术，说白了就是试图使用不变的代码来实现可变的算法。比如：模板技术，RTTI技术，虚函数技术，要么是试图做到在编译时决议，要么试图做到运行时决议。

关于虚函数的使用方法，我在这里不做过多的阐述。大家可以看看相关的C++的书籍。在这篇文章中，我只想从虚函数的实现机制上面为大家一个清晰的剖析。

当然，相同的文章在网上也出现过一些了，但我总感觉这些文章不是很容易阅读，大段大段的代码，没有图片，没有详细的说明，没有比较，没有举一反三。不利于学习和阅读，所以这是我想写下这篇文章的原因。也希望大家多给我提意见。

言归正传，让我们一起进入虚函数的世界。

虚函数表

对C++ 了解的人都应该知道虚函数（Virtual Function）是通过一张虚函数表（Virtual Table）来实现的。简称为V-Table。在这个表中，主是要一个类的虚函数的地址表，这张表解决了继承、覆盖的问

http://blog.csdn.net/haoel/article/details/1948051/

1/18

用GDB调试程序（一）	(435328)
C++ 虚函数表解析	(297984)
跟我一起写 Makefile（一）	(269193)
其实Unix很简单	(128222)
C++ 对象的内存布局（上）	(119140)
再谈“我是怎么招聘程序员”的	(102942)
用GDB调试程序（二）	(90471)
一些重要的算法	(85147)
Java NIO类库Selector机	(72934)
简单实用的Code Review	(72032)

评论排行	
C++ 虚函数表解析	(438)
清华大学出版社“抄袭事件”	(336)
再谈“我是怎么招聘程序员”的	(284)
“清华大学出版社抄袭事件”	(236)
我是怎么招聘程序员的	(235)
6个变态的C语言写的Hello World	(207)
恐怖的C++语言	(200)
优秀程序员的十个习惯	(167)
Java构造时成员初始化的顺序	(147)
其实Unix很简单	(137)

推荐文章	
* CSDN日报20170426 —— 《四无年轻人如何逆袭》	
* 抓取网易云音乐歌曲热门评论生成词云	
* Android NDK开发之从环境搭建到Demo级十步流	
* 个人的中小型项目前端架构浅谈	
* 基于卷积神经网络(CNN)的中文垃圾邮件检测	
* 四无年轻人如何逆袭	

最新评论	
免费电子书列表 海渔夫: 谢谢分享	
C++ 虚函数表解析 pkxpp: @lzhui1987: 没去试验过，我的理解是：类对象里面的指针指的并不是虚表的起始地址，而是第一个函...	
跟我一起写 Makefile（一） nacl: 就说一个，换行符是反斜线 \ 而不是 / 推荐 www.gnu.org/software/make/manu...	
跟我一起写 Makefile（五） 趁着d年轻: %d: %c @set -e; rm -f \$@; / ...	
用GDB调试程序（一） 谗阿星: 感谢楼主啊，一口气看完了你写的整个系列，作为初学者，你的文章为我节省了大把的时间！谢谢分享！！！	
跟我一起写 Makefile（一） candy73046: @snikeguo: 这个QBS是什么？	
C++ 虚函数表解析 HymanLiuTS: 有一个问题，刚刚看<>，它在第9页所说每个类所	

题，保证其容真实反应实际的函数。这样，在有虚函数的类的实例中这个表被分配在了这个实例的内存中，所以，当我们用父类的指针来操作一个子类的时候，这张虚函数表就显得尤为重要了，它就像一个地图一样，指明了实际所应该调用的函数。

这里我们着重看一下这张虚函数表。C++的编译器应该是保证虚函数表的指针存在于对象实例中最前面的位置（这是为了保证取到虚函数表的有最高的性能——如果有多层继承或是多重继承的情况下）。这意味着我们通过对象实例的地址得到这张虚函数表，然后就可以遍历其中函数指针，并调用相应的函数。

听我扯了那么多，我可以感觉出来你现在可能比以前更加晕头转向了。没关系，下面就是实际的例子，相信聪明的你一看就明白了。

假设我们有这样的一个类：

```
class Base {
public:
    virtual void f() { cout << "Base::f" << endl; }
    virtual void g() { cout << "Base::g" << endl; }
    virtual void h() { cout << "Base::h" << endl; }
};
```

按照上面的说法，我们可以通过Base的实例来得到虚函数表。下面是实际例程：

```
typedef void(*Fun)(void);

Base b;

Fun pFun = NULL;

cout << "虚函数表地址: " << (int*)(&b) << endl;
cout << "虚函数表 — 第一个函数地址: " << (int*)(int*)(&b) << endl;

// Invoke the first virtual function
pFun = (Fun)*((int*)(int*)(&b));
pFun();
```

实际运行结果如下：(Windows XP+VS2003, Linux 2.6.22 + GCC 4.1.3)

```
虚函数表地址: 0012FED4
虚函数表 — 第一个函数地址: 0044F148
Base::f
```

通过这个示例，我们可以看到，我们可以通过强行把&b转成int*，取得虚函数表的地址，然后，再次取址就可以得到第一个虚函数的地址了，也就是Base::f()，这在上面的程序中得到了验证（把int*强制转成了函数指针）。通过这个示例，我们就可以知道如果要调用Base::g()和Base::h()，其代码如下：

```
(Fun)*((int*)(int*)(&b)+0); // Base::f()
(Fun)*((int*)(int*)(&b)+1); // Base::g()
(Fun)*((int*)(int*)(&b)+2); // Base::h()
```

关联的type-info object会放到虚函数表的...

再谈“我是怎么招聘程序员的”
请叫我大湿胸: 实实在在的一个感受就是，这篇文章写了这么久了，但现在看到，仍然感触颇大，感觉分析的十分透彻，和其他分...

再谈“我是怎么招聘程序员的”
请叫我大湿胸: LZ从多个维度观察了程序员的评判标准，深入换位思考重新定义了应聘者与面试官的关系，其眼光之独到、思想...

用GDB调试程序（一）
hacker_2017:
https://gitlore.com/subject/15
100个gdb小技巧

这个时候你应该懂了吧。什么？还是有点晕。也是，这样的代码看着太乱了。没问题，让我画个图解释一下。如下所示：

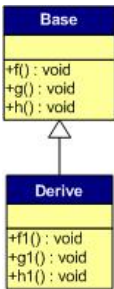


注意：在上面这个图中，我在虚函数表的最后多加了一个结点，这是虚函数表的结束结点，就像字符串的结束符“/0”一样，其标志了虚函数表的结束。这个结束标志的值在不同的编译器下是不同的。在WinXP+VS2003下，这个值是NULL。而在Ubuntu 7.10 + Linux 2.6.22 + GCC 4.1.3下，这个值是如果1，表示还有下一个虚函数表，如果值是0，表示是最后一个虚函数表。

下面，我将分别说明“无覆盖”和“有覆盖”时的虚函数表的样子。没有覆盖父类的虚函数是毫无意义的。我之所以要讲述没有覆盖的情况，主要目的是为了给一个对比。在比较之下，我们可以更加清楚地知道其内部的具体实现。

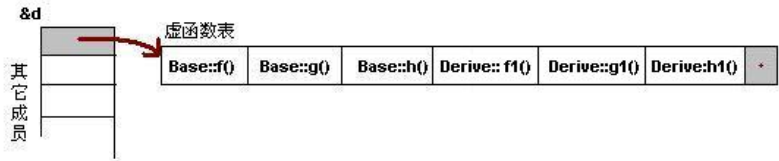
一般继承（无虚函数覆盖）

下面，再让我们来看看继承时的虚函数表是什么样的。假设有如下所示的一个继承关系：



请注意，在这个继承关系中，子类没有重载任何父类的函数。那么，在派生类的实例中，其虚函数表如下所示：

对于实例：Derive d; 的虚函数表如下：



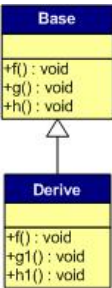
我们可以看到下面几点：

- 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。

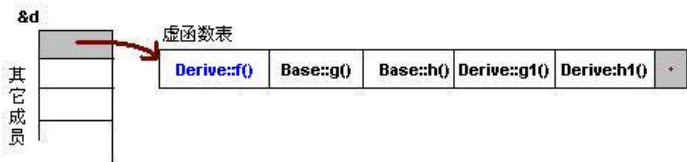
我相信聪明的你一定可以参考前面的那个程序，来编写一段程序来验证。

一般继承（有虚函数覆盖）

覆盖父类的虚函数是很显然的事情，不然，虚函数就变得毫无意义。下面，我们来看一下，如果子类中有虚函数重载了父类的虚函数，会是一个什么样子？假设，我们有下面这样的一个继承关系。



为了让大家看到被继承过后的效果，在这个类的设计中，我只覆盖了父类的一个函数：f()。那么，对于派生类的实例，其虚函数表会是下面的一个样子：



- 我们从表中可以看到下面几点，
- 1) 覆盖的f()函数被放到了虚表中原来父类虚函数的位置。
 - 2) 没有被覆盖的函数依旧。

这样，我们就可以看到对于下面这样的程序，

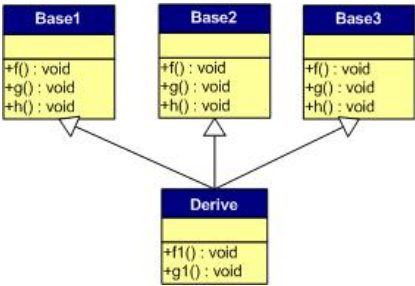
```
Base *b = new Derive();

b->f();
```

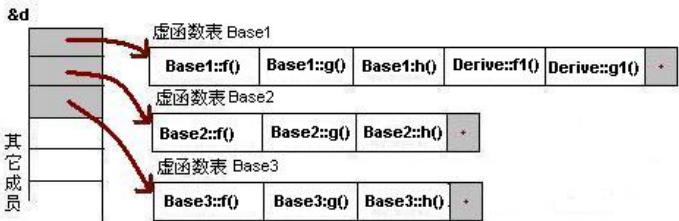
由b所指的内存中的虚函数表的f()的位置已经被Derive::f()函数地址所取代，于是在实际调用发生时，是Derive::f()被调用了。这就实现了多态。

多重继承（无虚函数覆盖）

下面，再让我们来看看多重继承中的情况，假设有下面这样一个类的继承关系。注意：子类并没有覆盖父类的函数。



对于子类实例中的虚函数表，是下面这个样子：



我们可以看到：

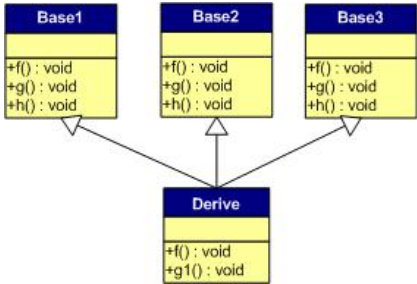
- 1) 每个父类都有自己的虚表。
- 2) 子类的成员函数被放到了第一个父类的表中。（所谓的第一个父类是按照声明顺序来判断的）

这样做就是为了解决不同的父类类型的指针指向同一个子类实例，而能够调用到实际的函数。

多重继承（有虚函数覆盖）

下面我们再来看看，如果发生虚函数覆盖的情况。

下图中，我们在子类中覆盖了父类的f()函数。



下面是对于子类实例中的虚函数表的图：



我们可以看见，三个父类虚函数表中的f()的位置被替换成了子类的函数指针。这样，我们就可以任一静态类型的父类来指向子类，并调用子类的f()了。如：

```
Derive d;  
Base1 *b1 = &d;  
Base2 *b2 = &d;  
Base3 *b3 = &d;  
  
b1->f(); //Derive::f()  
b2->f(); //Derive::f()  
b3->f(); //Derive::f()  
  
b1->g(); //Base1::g()  
b2->g(); //Base2::g()  
b3->g(); //Base3::g()
```

安全性

每次写C++的文章，总免不了要批判一下C++。这篇文章也不例外。通过上面的讲述，相信我们对虚函数表有一个比较细致的了解了。水可载舟，亦可覆舟。下面，让我们来看看我们可以用虚函数表来干点什么坏事吧。

一、通过父类型的指针访问子类自己的虚函数

我们知道，子类没有重载父类的虚函数是一件毫无意义的事情。因为多态也是要基于函数重载的。虽然上面的图中我们可以看到Base1的虚表中有Derive的虚函数，但我们根本不可能使用下面的语句来调用子类的自有虚函数：

```
Base1 *b1 = new Derive();  
b1->f1(); //编译出错
```

任何妄图使用父类指针想调用子类中的未覆盖父类的成员函数的行为都会被编译器视为非法，所以，这样的程序根本无法编译通过。但在运行时，我们可以通过指针的方式访问虚函数表来达到违反C++语义的行为。（关于这方面的尝试，通过阅读后面附录的代码，相信你可以做到这一点）

二、访问non-public的虚函数

另外，如果父类的虚函数是private或是protected的，但这些非public的虚函数同样会存在于虚函数表中，所以，我们同样可以使用访问虚函数表的方式来访问这些non-public的虚函数，这是很容易做到的。

如：

```
class Base {  
    private:  
        virtual void f() { cout << "Base::f" << endl; }  
  
};  
  
class Derive : public Base{  
  
};  
  
typedef void(*Fun)(void);  
  
void main() {  
    Derive d;  
    Fun pFun = (Fun)*((int*)(int*)&d)+0;  
    pFun();  
}
```

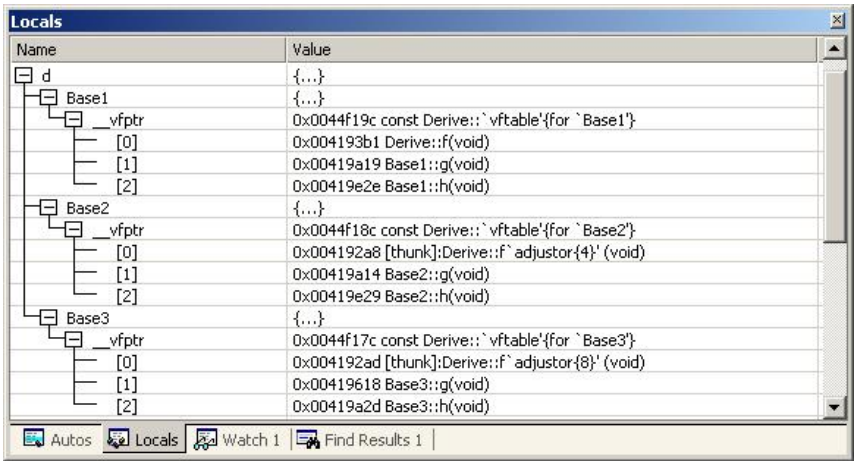
结束语

C++这门语言是一门Magic的语言，对于程序员来说，我们似乎永远摸不清楚这门语言背着我们在干了什么。需要熟悉这门语言，我们就必需要了解C++里面的那些东西，需要去了解C++中那些危险的东西。不然，这是一种搬起石头砸自己脚的编程语言。

在文章束之前还是介绍一下自己吧。我从事软件研发有十个年头了，目前是软件开发技术主管，技术方面，主攻Unix/C/C++，比较喜欢网络上的技术，比如分布式计算，网格计算，P2P，Ajax等一切和互联网相关的东西。管理方面比较擅长于团队建设，技术趋势分析，项目管理。欢迎大家和我交流，我的MSN和Email是：haol@hotmail.com

附录一：VC中查看虚函数表

我们可以在VC的IDE环境中的Debug状态下展开类的实例就可以看到虚函数表了（并不是很完整的）



附录二：例程

下面是一个关于多重继承的虚函数表访问的例程：

```
#include <iostream>
using namespace std;

class Base1 {
public:
    virtual void f() { cout << "Base1::f" << endl; }
    virtual void g() { cout << "Base1::g" << endl; }
    virtual void h() { cout << "Base1::h" << endl; }
};

class Base2 {
public:
    virtual void f() { cout << "Base2::f" << endl; }
    virtual void g() { cout << "Base2::g" << endl; }
    virtual void h() { cout << "Base2::h" << endl; }
};

class Base3 {
public:
    virtual void f() { cout << "Base3::f" << endl; }
    virtual void g() { cout << "Base3::g" << endl; }
    virtual void h() { cout << "Base3::h" << endl; }
};

class Derive : public Base1, public Base2, public Base3 {
public:
    virtual void f() { cout << "Derive::f" << endl; }
    virtual void g1() { cout << "Derive::g1" << endl; }
};

typedef void(*Fun)(void);
```

```
int main()
{
    Fun pFun = NULL;

    Derive d;
    int** pVtab = (int**)&d;

    //Base1's vtable
    //pFun = (Fun)*((int*)*(int*)((int*)&d+0)+0);
    pFun = (Fun)pVtab[0][0];
    pFun();

    //pFun = (Fun)*((int*)*(int*)((int*)&d+0)+1);
    pFun = (Fun)pVtab[0][1];
    pFun();

    //pFun = (Fun)*((int*)*(int*)((int*)&d+0)+2);
    pFun = (Fun)pVtab[0][2];
    pFun();

    //Derive's vtable
    //pFun = (Fun)*((int*)*(int*)((int*)&d+0)+3);
    pFun = (Fun)pVtab[0][3];
    pFun();

    //The tail of the vtable
    pFun = (Fun)pVtab[0][4];
    cout<<pFun<<endl;

    //Base2's vtable
    //pFun = (Fun)*((int*)*(int*)((int*)&d+1)+0);
    pFun = (Fun)pVtab[1][0];
    pFun();

    //pFun = (Fun)*((int*)*(int*)((int*)&d+1)+1);
    pFun = (Fun)pVtab[1][1];
    pFun();

    pFun = (Fun)pVtab[1][2];
    pFun();

    //The tail of the vtable
    pFun = (Fun)pVtab[1][3];
    cout<<pFun<<endl;

    //Base3's vtable
    //pFun = (Fun)*((int*)*(int*)((int*)&d+1)+0);
    pFun = (Fun)pVtab[2][0];
    pFun();
```



```
//pFun = (Fun)*((int*)((int*)((int*)&d+1)+1);
pFun = (Fun)pVtab[2][1];
pFun();

pFun = (Fun)pVtab[2][2];
pFun();

//The tail of the vtable
pFun = (Fun)pVtab[2][3];
cout<<pFun<<endl;

return 0;
}
```

(转载时请注明作者和出处。未经许可，请勿用于商业用途)

更多文章请访问我的Blog: <http://blog.csdn.net/haol>

顶

53

踩

2

上一篇

GPLv3：大教堂和集市的新一轮对抗

下一篇

Java NIO类库Selector机制解析（上）

我的同类文章

编程语言（57）			
• 一些重要的算法	2010-07-22	• “21天教你学会C++”	2010-05-14
阅读 85147		阅读 50478	
• 程序命名的一些提示	2010-04-08	• 哥是玩程序的	2010-04-01
阅读 33047		阅读 63667	
• 程序语言性能比拼	2009-12-15	• 恐怖的C++语言	2009-12-04
阅读 16765		阅读 51109	
• ldd 的一个安全问题	2009-11-10 阅读 6153	• 使用Flex Bison 和LLVM编...	2009-11-09

参考知识库



.NET 知识库
3783 关注 | 833 收录



Linux 知识库
11749 关注 | 3943 收录



算法与数据结构知识库
15854 关注 | 2320 收录

猜你在找

- VC++游戏开发基础系列从入门到精通
- VC++DLL动态链接库编程
- VC++Windows多线程实战图片编辑器
- C++ 虚函数表解析
- C++虚函数表解析
- C++ 虚函数表解析

深入浅出C++程序设计（基础篇）

C++ 虚函数表解析

C++程序设计

C++ 虚函数表解析

查看评论

343楼 HymanLiuTS 2017-03-22 16:23发表



有一个问题，刚刚看<<深度探索C++对象模型>>，它在第9页所说每个类所关联的type-info object会放到虚函数表的第一位，但是楼主的文章中提到表格的第一位放的是虚函数，不过代码验证楼主是对的，难道书中出错了？？？

Re: pkxpp 2017-04-21 11:14发表



回复lzhui1987：没去试验过，我的理解是：类对象里面的指针指的并不是虚表的起始地址，而是第一个函数地址项的地址。比如说type-info是放在表的第0项，第一个虚函数放在第1项，类对象的vptr指的是第1项所在的地址而已。

342楼 小朋友87 2016-10-15 17:58发表



把重载改成重写

341楼 stewewongbuaa 2016-09-06 16:42发表



写的很好，不过如果要在64位机器运行要把int换成long才行

关闭

340楼 lonely_geek 2016-09-05 20:53发表



讲的通俗易懂，但是笔者对于重载和覆盖没有区分开，这点不应该了啊。多态是基于函数的覆盖，不是重载

339楼 khing 2016-08-21 22:26发表



用VC试了下“一般继承（无虚函数覆盖）”，很惊讶地看到虚函数表里面并没有Derived的函数，大家可以试试

Re: jinshangyu 2017-02-27 18:20发表



回复khing：你肯定是继承写错了 少了virtual关键字

338楼 Annoymous_ 2016-07-18 21:32发表



多重继承部分Not Even Wrong! 多重继承的时候调用函数是要修正地址的

后面的两段安全性批评更是莫名其妙，要让程序不正常 直接用 (int*)0=0 就可以了，费这么大的功夫到底是在想批判什么

337楼 huisha25 2016-06-30 15:00发表



你好，今天看了你的虚函数讲解，受益匪浅。我从事c++开发也有一段时间了，想找个大牛学习学习，能加个好友吗？
QQ: 872569904
期待您的回复，谢谢！！

336楼 Bingogoc 2016-06-28 16:43发表



cout << "虚函数表地址: " << (int*)(&b) << endl;
cout << "虚函数表 — 第一个函数地址: " << (int*)(int*)(&b) << endl;
这个应该改成:
cout << "虚函数表地址: " << *((long*)(&b)) << endl;
cout << "虚函数表 — 第一个函数地址: " << *((long*)(long*)(&b)) << endl;
这样就能解决32位系统与64位系统的问题，两个地址应分别在原有的基础上加*，本人拙见

Re: jackqk 2016-10-14 11:08发表



回复Bingogoc: 技术还是得自己学习，别做梦了，大家都有事，陪陪家人也不会教你

335楼 qq_35053267 2016-05-31 00:07发表



是重写不是重载吧？

334楼 Shepsee 2016-05-08 17:34发表



非常清晰的解释 与插图 太感谢了

333楼 maowei117 2016-04-30 16:35发表



我认为:

```
[cpp]
01. (int*)&b
```

得到的不是虚函数表的地址，而是虚函数指针的地址。

Re: [whdugh](#) 2016-08-15 22:19发表



回复[maowei117](#)：对象取地址也不是虚函数的指针吧

Re: [g360z247j123](#) 2016-07-14 17:39发表



回复[maowei117](#)：我也认为如此。楼主原文说"C++的编译器应该是保证虚函数表的指针存在于对象实例中最前面的位置"，如果编译器真是这样做，那么对实例对象Base b取地址(&b)，得到的是指向虚函数表的指针的地址 ptr_addr；对该地址解引用(*ptr_addr)得到指向虚函数表的指针的值(即虚函数表的地址)Vtable_addr

332楼 [ProLianGee](#) 2016-04-03 13:23发表



好文章。解决了我很多疑惑。

331楼 [JRSmith7](#) 2016-04-01 20:14发表



请教下，例程代码的最后一个模块：

[cpp]

```
01. //The tail of the vtable
02. pFun = (Fun)pVtab[2][3];
03. cout<<pFun<<endl;
```

关闭

为什么输出是1？已经是最后一个虚函数表了，不应该输出0吗？

330楼 [chenyang1231](#) 2016-03-29 00:08发表



个人觉得关于打印虚函数表地址与直接使用虚函数表调用虚函数部分有点错误，附上自己测试过的代码。评论太多了，不知道有没有人发现这个错误，望楼主在博文里改正之，否则容易误人子弟。（如果不是我理解错的话）

[cpp]

```
01. cout << "虚函数表的地址" << *((int *)&base) << endl;
02.
03. ((fun)*((int *)(&base)) + 0))();
04. ((fun)*((int *)(&base)) + 1))();
05. ((fun)*((int *)(&base)) + 2))();
```

Re: [zhangjin739](#) 2016-09-29 23:25发表



回复[chenyang1231](#)：不用改代码， 楼主应该意思是 虚函数表地址的地址罢了

Re: [xingaide520](#) 2016-06-08 15:21发表



回复[chenyang1231](#)：能解释下吗，我对这个用法不理解，太多指针了

329楼 [路过少年](#) 2016-01-24 18:57发表



mark

328楼 [01_Terry](#) 2015-12-27 16:55发表



写的真心好，感谢分享！

327楼 [fifi_nam](#) 2015-10-29 16:06发表



有个问题，关于多重继承（有虚函数覆盖）。继承有几个父类就会有几个虚函数表，然后覆盖的虚函数会取代原来父类的对应的虚函数指针。可是继承的类对象，未覆盖的那些函数并没有在虚函数表中？

326楼 [andy当我遇上你](#) 2015-10-13 18:47发表



谢谢楼主，我收藏了！

325楼 [白白皎皎](#) 2015-09-30 10:22发表



大神，学习了

324楼 [改变oo](#) 2015-09-08 15:24发表



学习了

323楼 [jerome029](#) 2015-09-01 20:15发表



顶神牛！

322楼 [woshiyisang](#) 2015-08-20 17:21发表



想问一下:虚函数表的内存是什么时候分配的?类定义的时候?

321楼 [rudy_yuan](#) 2015-08-17 07:20发表



楼主的代码似乎有点问题?

这是我写的可以运行的代码: <http://www.rudy-yuan.net/archives/128/>

大家可以参考下, 便于理解。。。

320楼 [Change_Land](#) 2015-08-06 12:53发表



陈老师您好, 这里我有一个疑问, 就是我们不能在C++的构造函数中调用虚函数, 即便是调用了, 我们调用到的也是基类本身的那个函数, 例如:

```
class Transation
{
public:
    Transation()
    {
        logTransation();
    }

    virtual void logTransation()
    {
        cout << "call Transation::logTransation" << endl;
    }
};

class BuyTransation : public Transation
{
public:
    void logTransation() override
    {
        cout << "call BuyTransation::logTransation" << endl;
    }
};

class SaleTransation : public Transation
{
public:
    void logTransation() override
    {
        cout << "call SaleTransation::logTransation" << endl;
    }
};
```

```
void main(int argc, char* argv[])
{
    BuyTransation buy;
    SaleTransation sale;
}
```

输出的结果为:

```
call Transation::logTransation
call Transation::logTransation
```

按道理, 此时调用logTransation函数的地址已经被派生类的那个logTransation函数的地址替换了, 但是并没有出现这样的情况, 那么这个虚函数表的赋值逻辑是怎样的呢?

Re: [从来不作](#) 2015-08-07 14:54发表



回复Manistein: 对象在构造的时候, 首先构造它的基类部分, 然后构造自身。你举的例子应该是这样运行的: 首先构造基类, 设定vptr初值, 然后调用基类构造函数, 最后构造自身, 重新设定vptr的值。

319楼 [Lilyss21](#) 2015-07-26 21:17发表



学习了, 记录下。

318楼 [稚梟天卓](#) 2015-07-26 15:55发表

请教下, 虚函数表是一个类的对象共享, 还是一个对象就拥有一个虚函数表?

关闭



Re: 从来不作 2015-08-07 14:56发表



回复u013630349：每个类一个虚表，每个类的实例拥有一个指向虚表的指针。

Re: Isfv001 2016-06-11 15:54发表



不错,不过有个地方.
当类有 数据的时候.
比如int.
我的编译器
会这样排.
第一个虚表的地址.
int 的数据
第一个虚表的地址.

317楼 tppppppppp1 2015-06-23 16:30发表



我做实验的时候发现子类继承多个父类的时候子类对象的内存中存放的是“父类1虚函数表，父类1成员变量，父类2虚函数表，父类2成员变量。。。”，为何博主的图中的顺序是“父类1虚函数表，父类2虚函数表，成员变量”？

关闭

316楼 Hellen1101 2015-06-01 23:37发表



不错

315楼 Hellen1101 2015-06-01 23:38发表



不错

314楼 smzx_boy2012 2015-05-23 23:59发表



困扰许久的问题终于被解决了，多谢分享，希望我也有一天成长到分享这样的知识的人

313楼 东东同学 2015-05-06 21:38发表



<https://github.com/mudongliang/CppLearn>

其中testvtable*.cpp都是关于这个文章中几个例子写的demo，有什么问题，大家给我提一下！
另外看见有人说第一个代码有问题，不知道是什么意思？在我64bit和32bit机器上都运行没问题啊！

Re: s985507995 2016-01-14 15:07发表



回复mudongliangabcd：本来想把代码也给你放到github上的，提交试了下没权限。贴出来修改代码。

```
#include<iostream>
using namespace std;

class Base{
public:
    Base(){}
    // virtual ~Base(){}
    virtual void f(int a, int b){
        cout<<"Base::f()<<endl;
        int c = a + b;
        cout << "add " << c << endl;
    }
    virtual void g(){
        cout<<"Base::g()<<endl;
    }
    virtual void h(){
        cout<<"Base::h()<<endl;
    }
};

typedef void(*Fun)(int, int);

int main(){
    Base b;
    Fun pFun = NULL;
    cout << "虚函数地址: "<<(int *)&b<<endl;
    cout << "虚函数-第一个函数地址: "<<(int *)*(int *)&b <<endl;

    pFun = (Fun)*((int *)*(int *)&b);
    pFun(3, 4);
    return 0;
}
```

Re: [s985507995](#) 2016-01-14 15:02发表



回复mudongliangabcd: 1. 上述所有代码都不带构造和析构函数。(如果类中有虚函数,析构函数同样要设计成虚函数,否则可能会造成内存泄漏)
在加入虚析构函数后,按照上述逻辑,在64位机上跑testvtable.cpp,会有一些问题,你可以尝试一下;
2. 即便如文中所述,找到了虚函数的函数地址。可以尝试在函数内部做运算,我在64位机上,运算结果没有跑对的。这个结果还不知道有没有可以帮忙解释下。

312楼 [chenzhg33](#) 2015-04-02 23:46发表



cout<<"虚函数表地址: "<<(int*)(&b)<<endl;
这句有问题吧? &b是b对象的地址,就算强制转为int*,还是b的地址,虚函数表地址应该是*(int*)&b,对于64位机器应该是*(long*)&b

Re: [契约无罪](#) 2015-07-25 11:50发表



cout<<"虚函数表地址: "<<(int*)(&b)<<endl;
cout<<"虚函数表 — 第一个函数地址: "<<(int*)(int*)(&b)<<endl;
其实改成
cout<<" pV-Table = "
cout<<" V-Table" 或者 " V-Table[0]

Re: [Narsil](#) 2015-04-03 17:29发表



回复u010078776: 没错,这里很显然错误了,当然,无论 32 位还是 64 位,我们使用 intptr_t 就可以通用了

311楼 [wangzhongjunjun](#) 2015-03-25 16:36发表



请教下大神:上面的图片用什么思维导图软件做出来的?

310楼 [bama2488313716](#) 2015-03-23 10:46发表



博客写的非常好 但是想问下如果通过普通指针定位到虚函数位置,并且通过普通指针调用该虚函数,如何在虚函数中访问对象的成员变量呢 我知道用类的成员函数指针取代普通指针可以做到。

309楼 [ixshells](#) 2015-03-12 16:06发表



好文章, Mark一下

308楼 [jlstmac3](#) 2015-03-05 10:17发表



感谢大神,你的文章,把本来难以理解的知识点变得非常易懂

307楼 [969722243](#) 2015-01-18 16:51发表



非常好,学习了,多来几篇这样用的文章

306楼 [sunny_ss12](#) 2014-12-20 12:28发表



请好,对于虚函数表中虚函数的地址是不是应该写成
(Fun)*((int**)*(int**)(&b)+0); // Base::f()
(Fun)*((int**)*(int**)(&b)+1); // Base::g()
(Fun)*((int**)*(int**)(&b)+2); // Base::h()
否则在64位机器下是有错的。

Re: [hahaxiaohuo2015](#) 2015-02-02 14:41发表



回复sunny_ss12: 同意,但是int**这样很难理解,这位大神能否详细解释下
(Fun)*((int**)*(int**)(&b))的含义呢? 多谢多谢

Re: [sunny_ss12](#) 2015-02-26 17:01发表



回复sunnykuan_cafuc: 而(int*)(&b)相当于把Base的第一个元素看成了int,在32位机器下int和指针都是4个字节不会出问题,而在64位机器下,int和指针占用的空间大小不同,int是4个字节,而在64位机器下指针是8个字节,这样转换就有问题了

Re: [sunny_ss12](#) 2015-02-26 16:57发表



回复sunnykuan_cafuc: 我是这样想的:Base对象空间存储的第一个元素是虚函数表地址,即一个指针变量,对Base对象取地址实际上就是对该指针变量取地址,即应该是指向指针的指针,所以是(int**)(&b)而不是(int*)(&b)。不知道说的对不对

Re: [chenzhg33](#) 2015-04-02 23:25发表



回复sunny_ss12: 关键在于int** ptr是指向指针的指针,ptr+1就是移动8个字节,如果(Fun)*((int**)(int*)(&b)+2);也是移动8个字节,就是函数地址

305楼 [jiangjun12345shi](#) 2014-12-13 14:39发表



随便复制别人的 真没意思

关闭

304楼 [jiangjun12345shi](#) 2014-12-13 14:38发表



随便复制别人的 没他妈的意思

Re: [syj52417](#) 2015-03-05 22:16发表



回复jiangjun12345shi: 那你能给下原始链接?

303楼 [cheyiliu](#) 2014-12-11 17:17发表



学习了

302楼 [hothuangting](#) 2014-12-02 09:00发表



好啊

301楼 [忧伤还是快乐](#) 2014-11-20 08:32发表



您好: 麻烦请问一下即使访问到了子类中非继承的函数和访问到了基类中的private或者protected的函数, 这样做有什么实际的意义? 谢谢

300楼 [827fast](#) 2014-11-15 02:06发表



大体上是对的。有一个地方误导人啊。

299楼 [damotiansheng](#) 2014-11-10 00:16发表



崩溃, 第一个代码就有两个错误, 竟然没人提, 搞得我没看懂, 一直纠结

298楼 [Full_Speed_Turbo](#) 2014-10-29 10:14发表



讲解的很清楚, 通俗易懂! 博主V5!

297楼 [johnnyforonline](#) 2014-10-23 13:41发表



mark

296楼 [Sylvanas_XV](#) 2014-10-14 00:50发表



挺好的

295楼 [JakeMiao](#) 2014-10-05 11:01发表



很好。但是重载、重写、覆盖三者没有分清楚。

294楼 [xuweiqun](#) 2014-09-30 14:48发表



今天在 vs2003 and vs2013 + win7 下测试, 发现一个小问题:
经过多次 debug release测试, 虚函数表的结束结点 是个不确定值。

293楼 [liaoruiyan](#) 2014-09-28 18:53发表



其实应该解释下(Fun)*((int*)*(int*)&d)+0);这样的代码, 不然看完整篇博文还是迷迷糊糊不懂虚函数表

292楼 [Sylvernass](#) 2014-09-23 23:20发表



很不错, 这样就大致理解虚函数了

291楼 [kekey1210](#) 2014-09-21 11:32发表



好

290楼 [superbin](#) 2014-09-19 15:04发表



好牛的技术文章!

289楼 [hustcalm](#) 2014-09-14 11:35发表



好文!
不过还是觉得看《深入理解C++对象模型》是最靠谱的!


288楼 [yonggeno1](#) 2014-09-14 10:06发表



是一篇好文章, 支持作者!


287楼 [lanshanwanghao](#) 2014-09-12 20:48发表

关闭



楼主太牛逼了。讲的简单明了

286楼 lyzsyr 2014-08-27 10:57发表




毕业两年了，终于要啃一啃c++了。
楼主，写的很清楚很透彻。
学习，膜拜了！

285楼 picy2008 2014-08-19 19:35发表




good

284楼 寒山-居士 2014-08-12 18:31发表




写的很清晰，也很形象，就需要这样的 讲的很活，不像其他文章喜欢咬文嚼字

283楼 u0116snail 2014-08-12 11:17发表



不错，谢谢楼主

282楼 最怕认真 2014-08-08 19:10发表



很详细啊。我感觉一下子清晰了很多。


281楼 whs2004789 2014-08-01 15:55发表



[cpp]

```
01. typedef void(*Fun)(void);    //void类型的函数指针
02.
03. class Base
04. {
05. public:
06.     virtual void f() { cout << "Base::f" << endl; }
07.     virtual void g() { cout << "Base::g" << endl; }
08.     virtual void h() { cout << "Base::h" << endl; }
09. private:
10.     virtual void j() { cout << "Base::j" << endl; }
11. };
12.
13. class dev: public Base
14. {
15. public:
16.     virtual void k() { cout << "dev::k" << endl; }
17. };
18.
19. int main()
20. {
21.
22.     //Base b1;
23.     //b1.j();                //compile error
24.     dev d;
25.     //d.f();                 //compile error
26.     //通过函数指针访问到私有的j(), j()对于对象来讲本来是不可见的,指针太强大
27.     Fun pFun2 = (Fun)*((int*)(int*)&d)+3);
28.     pFun2();
29.
30.     Base *b2 = new dev();
31.     //b2->k();                //compile error,父类指针无法call子类特有的虚函数
32.     //通过函数指针访问到子类特有的虚函数k(), 指针太强大
33.     Fun pFun3 = (Fun)*((int*)(int*)b2+4);
34.     pFun3();
35.
36.     return 0;
37. }
```

280楼 catTom 2014-07-31 16:59发表



```
typedef void(*Fun)(void);


void main() {
    Derive d;
    Fun pFun = (Fun)*((int*)(int*)&d)+0);
    pFun();
}
```



```
void main() {
    Derive d;
    Fun pFun = (Fun)*((int*)(int*)&d)+0);
    pFun();
}
```

如果把虚函数部分改成cout一个自己的成员变量，那么我试过会出现段错误，可是如果把函数指针改成typedef void(*Fun)(void*);调用改成pFun(&d);其他地方都不改，就可以出现正确结果，这是为什么呢？

279楼 [huangzhe10](#) 2014-07-28 12:06发表




```
//The tail of the vtable
pFun = (Fun)pVtab[2][3];
cout<<pFun<<endl;

----->

//The tail of the vtable
pFun = (Fun)pVtab[0][3];
cout<<pFun<<endl;
```


278楼 [huangzhe10](#) 2014-07-28 11:43发表



```
[cpp]
01. //The tail of the vtable
02. pFun = (Fun)pVtab[2][3];
03. cout<<pFun<<endl;
04.
05. // 应该是这样的吧 呵呵
06. //The tail of the vtable
07. pFun = (Fun)pVtab[0][3];
08. cout<<pFun<<endl;
```


关闭

277楼 [shawn_hao](#) 2014-07-18 23:34发表



是这样理解吗？每个类都会根据自己的虚函数建立一个虚函数地址表。基类会，派生类也会。只不过建立的时候，先把所有基类的拿过来，然后再根据有覆盖没有，再去覆盖虚函数表中的地址。至于运行的时候，编译器只看对象的实例地址，然后根据实例对象的基类，去扩展类中相应的虚表中找虚地址，然后，调用相应的函数去执行。跟C中的函数指针是一样的吧。里面的函数表可是要自己亲自去编写的额。

276楼 [QQ51931373](#) 2014-07-17 11:40发表




```
cout<<"虚函数表地址："<<(int*)&b<<endl;
```

```
cout<<"虚函数表 — 第一个函数地址："<<(int*)(int*)&b<<endl;
```


这两句代码根本就是错误的，大家都在提，居然不改正，BS啊BS，什么？你的意思是，改正了就危及到了“ ”是做人态度的博弈。

Re: [关门兔](#) 2015-05-11 11:22发表




回复qq51931373：确实错了哦，第一次博主的文章看就有错误，看来以后要自己调试一下才行。

Re: [ldming](#) 2014-07-20 09:08发表




回复qq51931373：请看看这篇文章的发表时间是2007年，博主现在有新的博客在维护，你可以去看一下；博主写了这么多文章，如果每天都看评论去改文中的错误应该是很耗时的一件事情，况且大神也有很多事情要忙；既然评论中都已经说出问题所在，那看到这篇文章的朋友如果发现错误就可以通过评论来得到证实，也可以跟其他的小伙伴沟通，为什么非要让作者在原文中更正呢？有时候遇到错误不是更能够激发思考，使人印象深刻么？博主已经贡献了这么好的文章，我们又何必吹毛求疵，抱着感恩的态度给博主个赞不是很好~

Re: [QQ51931373](#) 2014-07-21 11:26发表




回复ldming：你说的也有道理，但是对许许多多的新人产生的误导呢？

275楼 [evcowlai](#) 2014-07-14 11:06发表



好，谢谢楼主，很有用！

274楼 [zzyoucan](#) 2014-06-23 18:48发表



不错，但还是有许多不懂的

查看更多评论

发表评论

用户名: u010442388

评论内容:



提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

- 全部主题
- Hadoop
- AWS
- 移动游戏
- Java
- Android
- iOS
- Swift
- 智能硬件
- Docker
- OpenStack
- VPN
- Spark
- ERP
- IE10
- Eclipse
- CRM
- JavaScript
- 数据库
- Ubuntu
- NFC
- WAP
- jQuery
- BI
- HTML5
- Spring
- Apache
- .NET
- API
- HTML
- SDK
- IIS
- Fedora
- XML
- LBS
- Unity
- Splashtop
- UML
- components
- Windows Mobile
- Rails
- QEMU
- KDE
- Cassandra
- CloudStack
- FTC
- coremail
- OPhone
- CouchBase
- 云计算
- iOS6
- Rackspace
- Web App
- SpringSide
- Maemo
- Compuware
- 大数据
- aptech
- Perl
- Tornado
- Ruby
- Hibernate
- ThinkPHP
- HBase
- Pure
- Solr
- Angular
- Cloud Foundry
- Redis
- Scala
- Django
- Bootstrap

关闭

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved



18