

virtual function table analysis

oel  Posted at 2007-12-18 22:07:00  481708  Favorites 1207

copyright

Category column:

Programming language

Article tags:

c++

fun

class

translator

language

iostream

Programming langu...

The column contains this content

260 subscriptions

58 articles

Subscribe to

the column

C++ virtual function table analysis

Chen Hao

<http://blog.csdn.net/haol>

ord

of virtual functions in C++ is mainly to realize the mechanism of polymorphism. Regarding this, in a nutshell, it uses the pointer of the parent type to point to the instance of its subclass, and then calls the member function of the actual subclass through the pointer of the subclass. This technique allows the pointer of the parent class to have "multiple shapes", which is a generic technique. The so-called generic technology, to put it bluntly, is trying to use a generic code to implement a variable algorithm. For example: template technology, RTTI technology, function technology, either try to do resolution at compile time, or try to do runtime resolution.

Regarding the use of virtual functions, I will not elaborate too much here. You can look at the C++ books. In this article, I just want to give you a clear analysis of the resolution mechanism of virtual functions.

Even though the same articles have appeared on the Internet, but I always feel that these articles are very easy to read, with large sections of code, no pictures, no detailed descriptions, no conclusions, and no inferences. It's not good for learning and reading, so that's why I want to write this article. I also hope that you will give me more opinions.

So here, let us enter the world of virtual functions together.

Virtual function table

Who knows C++ should know that virtual functions (Virtual Function) are implemented through a virtual function table (Virtual Table). Referred to as V-Table . In this table, we mainly need the table of the virtual function of a class. This table solves the problem of inheritance and polymorphism, and ensures that its content truly reflects the actual function. In this way, in an instance of a class with virtual functions, this table is allocated in the memory of this instance, so that when we use the pointer of the parent class to operate a subclass, this virtual function table is very important. Now, it is like a map, indicating the actual function that should be called.

So let's focus on this virtual function table. The C++ compiler should ensure that the pointer to the virtual function table exists in the frontmost position of the object instance (this is to ensure the highest performance of fetching the virtual function table - if there are multiple inheritance or multiple inheritance)). This means that



haoel

focus on

 503



 12

rough the address of the object instance, and then we can traverse the function pointer and corresponding function.

all this talk, I can tell you're probably more dizzy now than you were before. It doesn't the following is an actual example, I believe you will understand it at a glance.

we have a class like this:

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

g to the above statement, we can get the virtual function table through the instance of e is the actual routine:

```
edef void (*Fun)( void);

e b;

pFun = NULL;

t << " Address of virtual function table:" << ( int *)(&b) << endl;
t << " Virtual function table - address of the first function:" << ( int *)*( int *)(&b) << endl;

voke the first virtual function
n = (Fun)*(( int *)*( int *)(&b));
n();
```

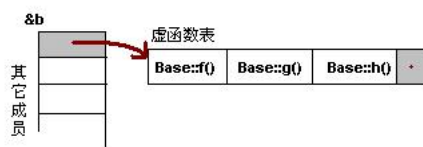
al running results are as follows: (Windows XP+VS2003, Linux 2.6.22 + GCC 4.1.3)

```
rtual function table address: 0012FED4
rtual function table -first function address: 0044F148
se::f
```

this example, we can see that we can get the address of the virtual function table by converting &b to int *, and then, by taking the address again, we can get the address of t virtual function, which is Base:f(), which is verified in the program above (casting the function pointer). Through this example, we can know that if we want to call Base:g() and , the code is as follows:

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

ld understand by now. What? Still a little dizzy. Also, such code looks too messy. No let me draw a picture to explain. As follows:



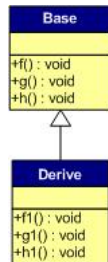
the above figure, I added an extra node at the end of the virtual function table, which is node of the virtual function table, just like the end cha e virtual function table end of. The value of this end f

s. Under WinXP+VS2003, this value is NULL. Under Ubuntu 7.10 + Linux 2.6.22 + GCC 4.1.3, if e is 1, it means there is the next virtual function table, and if the value is 0, it means virtual function table.

'll explain what the virtual function table looks like with "no coverage" and "with", respectively. Virtual functions without overriding the parent class are meaningless. The son why I'm going to talk about the case without coverage is to give a comparison. In on, we can more clearly know the specific implementation of its internal.

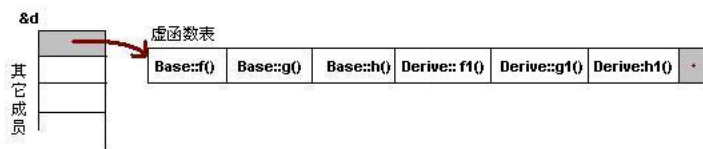
al inheritance (no virtual function override)

t's take a look at what the virtual function table looks like during inheritance. Suppose an inheritance relationship as follows:



t in this inheritance relationship, the subclass does not overload any functions of the ss. Then, in an instance of a derived class, its virtual function table looks like this:

ple: Derive d; the virtual function table is as follows:



ee the following points:

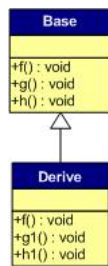
rtual functions are placed in the table in the order in which they are declared.

≥ virtual function of the parent class is in front of the virtual function of the child ass.

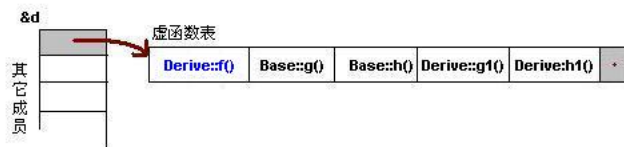
e that you are smart enough to refer to the previous program to write a program to verify.

al inheritance (with virtual function override)

vious to override the virtual function of the parent class, otherwise, the virtual function meaningless. Next, let's take a look, if there is a virtual function in the subclass that s the virtual function of the superclass, what will it look like? Suppose, we have an nce relationship like the following.



to let everyone see the effect of being inherited, in the design of this class, I only one function of the parent class: `f()`. Then, for an instance of a derived class, its function table will look like the following:



see the following points from the table,
 1. overwritten `f()` function is placed in the position of the original parent class virtual function in the virtual table.
 2. actions that are not overridden remain.

can see that for a program like the following,

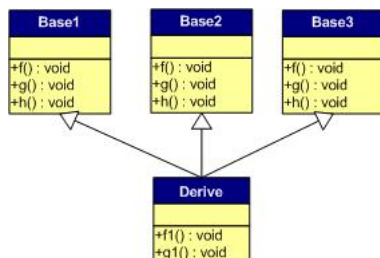
```
e *b = new Derive();
```

```
b->f();
```

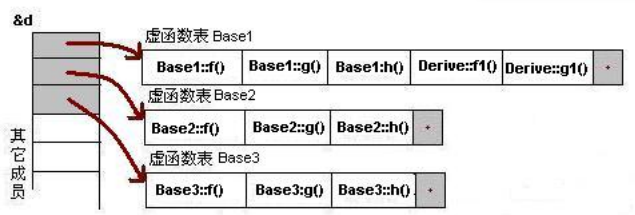
the action of `f()` in the virtual function table pointed to by `b` has been replaced by the address of the `Derive::f()` function, so when the actual call occurs, `Derive::f()` is called. This achieves polymorphism.

Multiple inheritance (no virtual function override)

Let's take a look at the situation in multiple inheritance, assuming that there is an inheritance relationship of the following class. Note: The subclass does not override the function of the superclass.



virtual function table in the subclass instance, it looks like this:



ee that:

h parent class has its own virtual table.

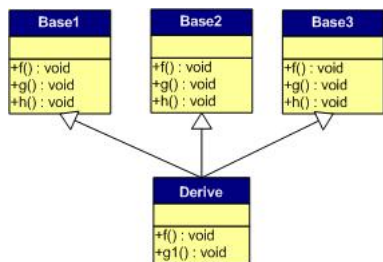
· member functions of the subclass are placed in the table of the first parent class. (The y-called first parent class is judged according to the declaration order)

to solve the problem that pointers of different parent class types point to the same instance, and can call the actual function.

ple inheritance (with virtual function override)

ke a look again, if virtual function coverage occurs.

ge below, we have overridden the f() function of the parent class in the child class .



owing is a diagram for the virtual function table in the subclass instance:



e that the position of f() in the virtual function table of the three parent classes is replaced by tion pointer of the child class. In this way, we can point to the subclass from any ly typed superclass and call f() of the subclass. like:

```
ive d;
e1 *b1 = &d;
e2 *b2 = &d;
e3 *b3 = &d;
>f(); //Derive::f()
>f(); //Derive::f()
>f(); //Derive::f()
```

```
>g(); //Base1::g()
>g(); //Base2::g()
```



haoel

focus on

503



12

```
>g(); //Base3::g()
```

Ÿ

me I write an article on C++, it is inevitable to criticize C++. This article is no n. Through the above description, I believe we have a more detailed understanding of the function table. Water can carry a boat or capsize it. Next, let's see what bad things we ith virtual function tables.

access the virtual function of the subclass itself through the pointer of the pe that it is a meaningless thing that the subclass does not overload the virtual function of rclass. Because polymorphism is also based on function overloading. Although in the above e can see that there is a **Derive virtual function** in the virtual table of **Base1**, it is le for us to use the following statement to call the subclass's own virtual function:

```
sel *b1 = new Derive();
>f1(); // Compile error
```

mpt to use the parent class pointer to call the member function of the subclass that t override the parent class will be regarded as illegal by the compiler, so such a cannot be compiled at all. But at runtime, we can access the virtual function table through to achieve behavior that violates C++ semantics. (For an attempt at this, by reading the the appendix below, I believe you can do it)

ss **non-public** virtual functions

ion, if the virtual function of the parent class is **private** or **protected**, but these non- public functions will also exist in the virtual function table, so we can also use the method of g the virtual function table to access these **non-public** virtual functions virtual function, easy to do.

```
ss Base {
private :
    virtual void f() { cout << "Base::f" << endl; }
```

```
ss Derive : public Base{
```

```
edef void (*Fun)( void );
```

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

iding remarks

magic language. For programmers, we never seem to know w s. To be familiar with this language, we must understand



haoel

focus on

👍 503



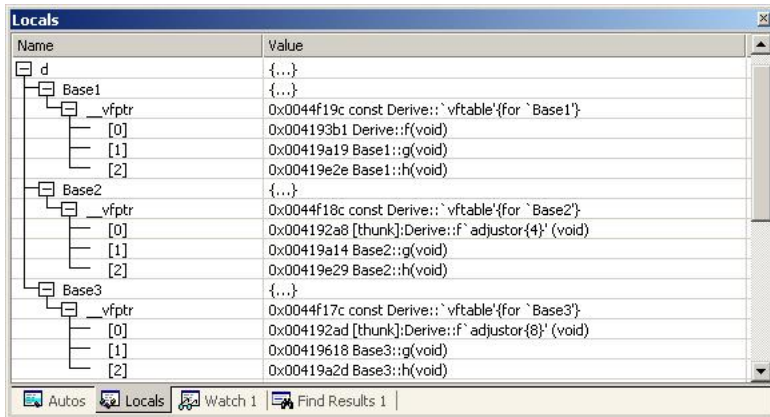
🌟 12

those dangerous things in C++. Otherwise, it's a programming language that shoots itself in

he end of the article, let me introduce myself. I have been engaged in software research
lopment for ten years. I am currently the technical director of software development. In
technology, I mainly focus on Unix /C /C++. related stuff. In terms of management, he is
team building, technology trend analysis, and project management. Welcome everyone to
ate with me, my MSN and Email are: haael@hotmail.com

lix 1: View the virtual function table in VC

xpand the instance of the class in the Debug state of the VC IDE environment to see the
function table (not very complete)



lix II: Routines

an example of a virtual function table access for multiple inheritance:

```
>stream>
space std;

{

jal void f() { cout << "Base1::f" << endl; }
jal void g() { cout << "Base1::g" << endl; }
jal void h() { cout << "Base1::h" << endl; }

{

jal void f() { cout << "Base2::f" << endl; }
jal void g() { cout << "Base2::g" << endl; }
jal void h() { cout << "Base2::h" << endl; }

{

jal void f() { cout << "Base3::f" << endl; }
jal void g() { cout << "Base3::g" << endl; }
jal void h() { cout << "Base3::h" << endl; }

: public Base1, public Base2, public Base3 {

jal void f() { cout << "Derive::f" << endl; }
```



haael focus on

503 12

```
jal void g1() { cout << "Derive::g1" << endl; }
```

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

```
pFun = NULL;
```

```
ive d;
```

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

base1's vtable

```
un = (Fun)*((int*)*(int*)((int*)&d+0)+0);
```

```
n = (Fun)pVtab[ 0 ][ 0 ];
```

```
n();
```

```
un = (Fun)*((int*)*(int*)((int*)&d+0)+1);
```

```
n = (Fun)pVtab[ 0 ][ 1 ];
```

```
n();
```

```
un = (Fun)*((int*)*(int*)((int*)&d+0)+2);
```

```
n = (Fun)pVtab[ 0 ][ 2 ];
```

```
n();
```

base's vtable

```
un = (Fun)*((int*)*(int*)((int*)&d+0)+3);
```

```
n = (Fun)pVtab[ 0 ][ 3 ];
```

```
n();
```

the tail of the vtable

```
n = (Fun)pVtab[ 0 ][ 4 ];
```

```
t<<pFun<<endl;
```

base2's vtable

```
un = (Fun)*((int*)*(int*)((int*)&d+1)+0);
```

```
n = (Fun)pVtab[ 1 ][ 0 ];
```

```
n();
```

```
un = (Fun)*((int*)*(int*)((int*)&d+1)+1);
```

```
n = (Fun)pVtab[ 1 ][ 1 ];
```

```
n();
```

```
n = (Fun)pVtab[ 1 ][ 2 ];
```

```
n();
```

the tail of the vtable

```
n = (Fun)pVtab[ 1 ][ 3 ];
```

```
t<<pFun<<endl;
```

base3's vtable

```
un = (Fun)*((int*)*(int*)((int*)&d+1)+0);
```

```
n = (Fun)pVtab[ 2 ][ 0 ];
```

```
n();
```

```
un = (Fun)*((int*)*(int*)((int*)&d+1)+1);
```

```
n = (Fun)pVtab[ 2 ][ 1 ];
```



haoel

focus on

503



12


```
n();

n = (Fun)pVtbl[ 2 ][ 2 ];
n();

// the tail of the vtable
n = (Fun)pVtbl[ 2 ][ 3 ];
t<<pFun<<endl;

return 0 ;
```

indicate the author and source when reprinting. Please do not use for commercial
s without permission)

articles, please visit my Blog: <http://blog.csdn.net/haael>

explanation of **c++ virtual functions (you must understand it)** lyztyycode's Blog 240,000+
· **virtual function** Daniel's article is easy to understand and concise. Preface The role of **virtual functions in C++** is ...

morphism **virtual function virtual function table** is the most detailed 10-01
by **C++** polymorphism explanation, detailed explanation of **virtual functions** , **virtual function tables** , **virtual function** in...

explanation of **virtual function table**_Gigi Hermes **Blog_Virtual function table** 9-22
cludes two **virtual functions** , so A vtbl includes two pointers, pointing to A::v **fun** c1() and A::v **fun** c2() respectively. ...

nction and **virtual function table**_tiankong19999 's **blog_virtual function list** 9-16
; object shares the **virtual function table** of the class, each class object has a **virtual function** pointer vptr, and the **virt...**

explanation of **C++ virtual functions (dynamic binding)** TABE_'s Blog 10,000+
itle of the directory here. Overview of the **virtual table** of the class General inheritance (without **virtual function** cover...

al **function** and **virtual function table** principle Popular recommendation 7now_'s Blog 40,000+
sses of **virtual functions** are stored in the **virtual function table** . Runtime polymorphism is achieved through **virtual fu...**

nction **table** 5-19
lass with **virtual functions** will generate a **virtual function table** to store pointers to **virtual** member **functions** . (2) Eac...

nction **table** structure __ __ 9-21
e out the first 4 bytes of the b1 and d1 objects, which are the pointers to the **virtual table** , and the **virtual function ta...**

al **function table** example analysis 01-21
ism is an important feature of **C++** object-oriented programming. It was amazing to see **virtual functions** before , wh...

explanation of **virtual functions** , **virtual pointers** and **virtual tables in C++** bob's blog 2672
id knowledge about **virtual functions** A **function** declared with the virtual keyword is called a **virtual function** , and a **vi...**

explanation of **virtual functions** The Last of Qingping's Blog 20,000+
ctory 1. **Virtual function** instance 2. Implementation of **virtual function** (memory layout) 1. No inheritance situation 2. ...

cription of **virtual function table** wufeifan_learner's Blog 2657
; ago, I briefly talked about the **virtual function of C++** in my blog. The so-called **virtual function** is the method of **C++**...

nction weixin_38607041's blog 3386
le of Contents **Virtual Function** Definition of **Virtual Function** Precautions for the Use of **Virtual Functions** __ __ Overr...

al **function table** analysis 03-03
re role of **virtual functions in C++** is mainly to realize the mechanism of polymorphism. Regarding polymorphism, in ...

al **function** and **virtual function table** analysis 07-24
l **function** and **virtual function table analysis** , detailed content, clear analysis, recommended to everyone.

nction **table** in **C++** cz's blog 38
of polymorphism is the **virtual function table** 1. Overview In order to realiz



haoel

focus on

503



12

virtual function latest release
virtual function declared as virtual in a base class and redefined in one or more derived classes is called a virtual function ...

virtual function table
virtual function is a specific form of object-oriented programming function , and it is an effective mechanism for C++ to realize ...

explanation of C++ virtual functions
virtual functions ? 1.1 Definition of virtual functions Virtual functions are used when implementing C++ polymorphism...

explanation of C++ virtual functions
the role of virtual functions in C++ is mainly to realize the mechanism of polymorphism. Regarding polymorphism, in ...

concept and use of virtual functions
uses virtual functions in order to use polymorphism . The role of virtual functions is to allow redefinition of functions ...

concept and use of virtual functions
from: http://c.biancheng.net/cpp/biancheng/view/244.html We know that it is impossible to define two functions with ...

virtual function table
this article mainly analyzes and summarizes the virtual function table . Virtual functions have been summarized in previous ar...

Did "Related Recommendations" help you?

very unhelpful didn't help generally helpful very helpful

©2022 CSDN skin theme: Dabai Designer: CSDN official blog Return to the home page

Careers Business seek 400-660- online Working hours 8:30-
Cooperation coverage 0108 service 22:00
Public security record number 11010502030143 Beijing ICP No. 19004658 Beijing Net Wen [2020] No. 1039-165
Operating website filing information Beijing Internet Illegal and Bad Information Reporting Center parental supervision
Network 110 Alarm Service China Internet Reporting Center Chrome store download Account Management Specifications
Copyright and Disclaimer Copyright complaint Publication license business license
©1999-2022 Beijing Chuangxin Lezhi Network Technology Co., Ltd.



haoel

Code age 23

120

original

4634

Weekly Rank

1.15

million+ Overall ranking

6.54

million+ access

grade

30,000+ 20,000+ 3115 5429 8085
integral fan Liked Comment collect



Private letter focus on

Search blogger articles

popular articles







- Debugging programs with GDB (1) 647990
- Write Makefile with me (1) 613780
- C++ virtual function table parsing 481701
- Actually Unix is very simple 160759
- Memory layout of C++ objects (on)



haoel

focus on

Category column

	Technology Trends	13 articles
	Plagiarism	7 articles
	Programming Tools	19 articles
	Programming langu...	58 articles
	professional mood	23 articles
	software development	28 articles



latest comment

Write Makefile with me (14)

beyondbss: Thank you boss, I finished it in one go

Write Makefile with me (10)

beyondbss: You ask me if I own porcelain, of course I belong to porcelain~

C++ virtual function table analysis

businia08130: Transparent

Programming training (1)

Xue Bingbing: This one probably doesn't know who the author is. I suggest you ch ...

C++ virtual function table analysis

Prison Code Division: I wrote the complete test code of the four cases and put it on ...

Would you like to recommend the Blog Detail Page to a friend?



strongly
not



Not
recommen



so so



recommen



highly
recommen

latest articles

Talk about "how do I recruit programmers"

C skills: convert structure parameters into indeterminate parameters

List of free ebooks

4 articles in 2011

16 articles in 2010

48 articles in 2009

6 articles in 2008

9 articles in 2007

10 articles in 2006

20 articles in 2004

28 articles in 2003

Table of contents

foreword

virtual function table

一般继承 (无虚函数覆盖)

一般继承 (无虚函数覆盖)



haoel

focus on

503



12

多重继承 (有虚函数覆盖)

安全性

结束语

附录一: VC中查看虚函数表

附录二: 例程



haoel

focus on

👍 503



★ 12