

单重虚继承无虚函数

我们用三个类来描述这个问题，

基类，家具类 `CFurniture` ，

派生类，沙发类 `CSofa` 和床类 `CBed` ，

虽然沙发和床确实有一些共性的地方，这里我们仅仅做演示。

代码：

```
1 // 2. 单重虚继承无虚函数.cpp : 定义控制台应用程序的入口点。
2 //
3
4 #include "stdafx.h"
5 #include <iostream>
6 using namespace std;
7
8 // 家具类
9 class CFurniture {
10 public:
11     CFurniture() {
12         printf("CFurniture::CFurniture()\r\n");
13         m_nFurniture = 3;
14         //m_nFurniture2 = 4;
15     }
16
17     ~CFurniture() {
18         printf("CFurniture::~~CFurniture()\r\n");
19         m_nFurniture = 0;
20     }
21
22     int m_nFurniture;
23     //int m_nFurniture2;
24 };
25
26 // 沙发类，虚继承
27 class CSofa :virtual public CFurniture {
28 public:
29     CSofa() {
30         printf("CSofa::CSofa()\r\n");
```

```
31         m_nSofa = 1;
32     }
33
34     ~CSofa() {
35         printf("CSofa::~~CSofa()\r\n");
36         m_nSofa = 0;
37     }
38
39     void sit() {
40         printf("CSofa::sit()\r\n");
41     }
42
43 private:
44     int m_nSofa;
45 };
46
47 // 床类，虚继承
48 class CBed :virtual public CFurniture {
49 public:
50     CBed() {
51         printf("CBed::CBed()\r\n");
52         m_nBed = 2;
53     }
54
55     ~CBed() {
56         printf("CBed::~~CBed()\r\n");
57         m_nBed = 0;
58     }
59
60     void sleep() {
61         printf("CBed::sleep()\r\n");
62     }
63
64 private:
65     int m_nBed;
66 };
67
68 int main()
69 {
70     // 实例化对象
71     CFurniture furniture;
72     CSofa sofa;
```

```

73     CBed bed;
74
75     // 对象大小
76     int nSizeFurniture = sizeof(CFurniture);
77     cout << nSizeFurniture << endl; // 4
78     int nSizeSofa = sizeof(CSofa);
79     cout << nSizeSofa << endl; // 12
80     int nSizeBed = sizeof(bed);
81     cout << nSizeBed << endl; // 12
82
83     return 0;
84 }

```

对象实例化

```

1     CFurniture furniture;
2     CSofa sofa;
3     CBed bed;

```

构造,

基类, `CFurniture furniture` 访问自身构造函数 `CFurniture()`

派生类, `CSofa sofa` 访问自身构造函数, 向上查找父类, 访问父类的构造函数, 执行完毕后, 再回到自身访问构造函数。

`CBed bed` 同理。

程序执行完毕, 析构:

析构对象 `CBed ~CBed()`, 析构父类对象 `CPerson ~CPerson()`,

析构对象 `CSofa ~CSofa()`, 析构父类对象 `CPerson ~CPerson()`,

析构父类对象 `CPerson ~CPerson()`

对象大小

```

1 // 对象大小
2     int nSizeFurniture = sizeof(CFurniture);
3     cout << nSizeFurniture << endl; // 4

```

```

4     int nSizeSofa = sizeof(CSofa);
5     cout << nSizeSofa << endl; // 12
6     int nSizeBed = sizeof(bed);
7     cout << nSizeBed << endl; // 12

```

基类，`CFurniture` 有一个 `int` 型成员变量 `m_nFurniture`，那对象的大小 `sizeof(nSizeFurniture)` 等于 4 个字节大小。

派生类，

`CSofa` 自身有一个 `int` 型成员变量 `m_nSofa`，大小为 4bytes，

`CSofa` 类继承自 `CFurniture` 类，

所以 `CSofa` 类还有一个基类的 `int` 类型的成员变量 `m_nFurniture`，那这里应该是 8bytes 啊，咋就是 12bytes 呢？

`CBed` 类同 `CSofa`，

让我们来看看 虚继承 是怎么回事儿。

对象内存分布

`CFurniture furniture` 对象内存

The screenshot shows a debugger window with the following code:

```

68 // 实例化对象
69 CFurniture furniture;
70 CSofa sofa;
71 CBed bed;
72
73 // 对象大小
74 int nSizeFurniture = sizeof(CFurniture); 已用时间 <= 1ms

```

The memory window shows the address `0x00B7FB2C` with the value `00 00 00 00`. The watch window shows the following variables:

名称	值
&furniture	0x00b7fb2c {m_nFurniture=3}
&sofa	0x00b7fb18 {m_nSofa=1}

`CFurniture furniture;`

对象的 `int` 类型的成员变量 `m_nFurniture` 被初始化为 0

`CSofa sofa;`

The screenshot shows a debugger window with the following code:

```

68 // 实例化对象
69 CFurniture furniture;
70 CSofa sofa;
71 CBed bed;
72
73 // 对象大小
74 int nSizeFurniture = sizeof(CFurniture); 已用时间 <= 1ms

```

The memory window shows the address `0x0116FE4C` with the value `74 8b 01 01`. The watch window shows the following variables:

名称	值
&furniture	0x0116fe60 {m_nFurniture=3}
&sofa	0x0116fe4c {m_nSofa=1}
CFurniture	{m_nFurniture=3}
m_nFurniture	3
m_nSofa	1
&bed	0x0116fe38 {m_nBed=-858993460}

我们来着重分析下这里面的内存都表示的什么。

The screenshot shows a C++ code editor with the following code:

```

69 CFurniture furniture;
70 CSofa sofa;
71 CBed bed; 已用时间 <= 1ms
72
73 // 对象大小
74 int nSizeFurniture = sizeof
75 cout << nSizeFurniture << endl;

```

Below the code is a memory dump window showing the memory layout of the `CSofa` object. The address `0x0116FE4C` is highlighted. The memory dump shows the following values:

地址	值
0x0116FE4C	74 8b 01 01
0x0116FE50	01 00 00 00
0x0116FE54	03 00 00 00
0x0116FE58	cc cc cc cc
0x0116FE5C	cc cc cc cc
0x0116FE60	03 00 00 00

Annotations in the image point to specific memory locations:

- `0x0116FE4C` is labeled as "CSofa对象的虚表地址" (Virtual table address of CSofa object).
- `0x0116FE50` is labeled as "CSofa自身成员变量m_nSofa" (CSofa's own member variable m_nSofa).
- `0x0116FE54` is labeled as "基类成员变量 m_nFurniture" (Base class member variable m_nFurniture).

On the right, a memory dump window shows the memory layout of the `CSofa` object. The address `0x01018B74` is highlighted. The memory dump shows the following values:

地址	值
0x01018B74	00 00 00 00
0x01018B78	08 00 00 00
0x01018B7C	00 00 00 00
0x01018B80	43 53 6f 66 CSof
0x01018B84	61 3a 3a 43 a::C
0x01018B88	53 6f 66 61 Sofa

An annotation points to the value `08 00 00 00` at address `0x01018B78`, labeled as "偏移量, 从虚表地址访问到父类成员变量的偏移量" (Offset, from virtual table address to parent class member variable offset).

看到这里明白了, 为什么是 12bytes 了吧。

偏移量:

`0x0116FE54` 这个位置是父类成员变量在当前类中的内存位置。

`0x0116FE4C` 是当前类的成员变量在内存的位置。

`0x0116FE54 - 0x0116FE4C = 8(HEX)`, 就是 `08 00 00 00` 了。

得到的就是, 当前类对象地址到访问父类成员在当前类对象中的成员变量的偏移量。也就是说, 需要偏移多少才可以访问到父类成员变量。

对象组成:

给父类 `CFurniture` 再加一个成员变量 `m_nFurniture2` 来帮助分析。

The screenshot shows a C++ code editor with the following code:

```

67
68 int main()
69 {
70     // 实例化对象
71     CFurniture furniture;
72     CSofa sofa;
73     CBed bed; 已用时间 <= 8ms
74
75     // 对象大小

```

Below the code is a memory dump window showing the memory layout of the `CSofa` object. The address `0x0079F83C` is highlighted. The memory dump shows the following values:

地址	值
0x0079F83C	74 8b b8 00
0x0079F840	01 00 00 00
0x0079F844	03 00 00 00
0x0079F848	04 00 00 00
0x0079F84C	cc cc cc cc
0x0079F850	cc cc cc cc

Annotations in the image point to specific memory locations:

- `0x0079F840` is labeled as "CSofa自身成员" (CSofa's own member).
- `0x0079F844` is labeled as "父类CFurniture成员变量" (Parent class CFurniture member variable).
- `0x0079F848` is labeled as "组合成了CSofa sofa的整个对象" (Combined into the entire CSofa sofa object).

On the right, a memory dump window shows the memory layout of the `CSofa` object. The address `0x00B88B74` is highlighted. The memory dump shows the following values:

地址	值
0x00B88B74	00 00 00 00
0x00B88B78	08 00 00 00
0x00B88B7C	00 00 00 00
0x00B88B80	43 53 6f 66 CSof
0x00B88B84	61 3a 3a 43 a::C
0x00B88B88	53 6f 66 61 Sofa