2021/05/18_x86逆向C++_第5课_多重继承

笔记本: x86逆向-C++
创建时间: 2021/5/18 星期二 15:07
作者: ileemi

- 类成员初始化的初始化时机
- 寻找虚表的方法
- 构造函数、成员函数的重载
- 运算符重载
- 模板
- 静态成员
- 静态成员函数
- 单重继承的识别
  - 继承构造的特征
  - 继承析构的特征
  - 成员对象构造的特征
  - 无需表情况
- 多重继承构造的特征
- 多重继承析构的特征
- 多重成员函数的特征
- 成员对象析构的特征
- 无需表的还原
- 抽象类的特征
- purecall 函数的识别

类成员初始化的初始化时机

- 在构造中初始化（在填充虚表之后）
- 使用初始化表（在填充虚表之后）
- 定义时直接赋值（在填充虚表之后）

> 再Debug编译的程序中，填充虚表之前有初始化代码，可能是基类的初始化代码被内联。

寻找虚表的方法

通过构造函数获取虚表位置
通过析构函数获取虚表位置
通过调用虚函数的位置获取虚表位置

构造函数、成员函数的重载

- 构造函数重载有还原依据
- 成员函数重载没有还原依据（无法获取函数名，只能按照功能还原）

运算符重载

没有还原依据，除非有符号（.pdb）。根据可读性选择还原方式。

- 成员运算符重载
- 全局运算符重载
- 友元运算符重载（是一个全局函数，不是成员函数，不能直接调用成员）

模板

没有还原依据，根据可读性选择还原方式。

- 函数模板
- 类模板

静态成员

和全局变量一样，没有还原依据。当发现全局变量和某个类有较大的相关性时，可以考虑将其定义为该类的静态成员。

静态成员函数

静态成员函数的特征和全局函数无区别，根据功能相关性选择可读性好的还原方式。

单重继承的识别

继承构造的特征

填充虚表之前有函数调用
this指针无变化
出现覆盖虚表

继承析构的特征

填充虚表之后有函数调用
this指针无变化
出现覆盖虚表

成员对象构造的特征

填充虚表之前有函数调用
this指针有变化
未出现覆盖虚表

无需表情况

继承的方式还原
组合的方式还原

多重继承构造的特征

有虚表（虚函数）的前提下：

- 填充虚表之前有多次函数调用（构造子类前需要先构造父类）
- **this指针有变化**
- **多次出现覆盖虚表**

示例：

```
class A{ ... }
class B{ ... }
class AB:public A, public B{ ... }

// 虚表结构（构造）：
A::vatable = {A::~A, A::fun1}
A::member

B::vatable = {B::~B, B::fun2}
B::member
// 覆盖虚表 新增加的虚函数存储到第一个覆盖虚表的后面
A::vatable = {AB::~AB, AB::fun1, AB::fun3}
A::member
B::vatable = {AB::~AB, AB::fun2} // 没有覆盖虚表，填写父类虚表 this指针
+8
B::member
AB::member
```

类成员对象的构造在填写虚表之后调用（this指针会增加）。

多重继承析构的特征

有虚表（虚函数）的前提下：

- 填充虚表之后有多次函数调用（先析构子类，在依次析构父类）
- **this指针有变化**
- **多次出现覆盖虚表**

多重成员函数的特征

- 填充虚表之前有多次函数调用
- this指针有变化
- 未多次出现覆盖虚表

成员对象析构的特征

- 填充虚表之后有多次函数调用
- this指针有变化
- 未多次出现覆盖虚表

无需表的还原

继承的方式还原
组合的方式还原

抽象类的特征

> 抽象类有纯虚函数，派生类必须实现。

对于抽象类的识别（代码运行库为动态库时），虚表中会出现 "**purecall**" 字符标志，一个 "purecall" 字符就标志一个纯虚函数，也就代码改类为抽象类（接口类没有函数实现）。

抽象类的析构也可以为纯虚函数，但是子类必须实现（用于还原虚表），以下写法可以绕过编译器：

```
virtual ~Test() = 0 {
}
```

非动态编译抽象类纯虚函数的识别，ida观察虚表中没有"purecall" 字符标志，可查看虚表项指向的地址是否一致（如果一致可判定为纯虚函数），还可通过错误码识别

（VC++ 6.0）：

```
__purecall      proc near               ; DATA XREF: .rdata:off_425050↓o
                                        ; .rdata:00425054↓o ...
                push    ebp
                mov     ebp, esp
                push    19h
                call    __amsg_exit
; ---------------------------------------------------------------------------
                add     esp, 4
                pop     ebp
                retn
__purecall      endp
```

purecall 函数的识别

- 低版本返回0x19错误
- 高版本出现0x40000015错误

```
                int     29h             ; Win8: RtlFailFast(ecx)
; ---------------------------------------------------------------------------

loc_4B11C3:                             ; CODE XREF: _abort+4A↑j
                push    1
                push    40000015h
                push    3
                call    j____acrt_call_reportfault
                add     esp, 0Ch

loc_4B11D4:                             ; CODE XREF: _abort+3E↑j
                push    3               ; Code
                call    j___exit_0
_abort          endp
```