设计模式的逆向特征

一、创建型

1.工厂模式

- 单继承

```
Fruit *__fastcall Apple::Apple(Apple *this)
{
  Fruit *result; // rax
  Fruit *v2; // [rsp+30h] [rbp+10h]

  v2 = this;
  Fruit::Fruit(this);
  result = v2;
  *(_QWORD *)v2 = &off_404570;
  return result;
}
```

  实际上单继承就是把 baseClass 的成员变量完全copy了一份放在了childClass的前面。
- 多继承
  其实也都是成员变量按顺序往后排
- 虚函数与虚表
  C++实现虚函数的方法是：为每个类对象添加一个隐藏成员，隐藏成员保存了一个指针，这个指针叫虚表指针（vptr），它指向一个虚函数表（virtual function table, vtbl）。在运行时创建对象时，对象的虚表指针将设置为指向合适的虚表。如果该对象调用一个虚函数，则通过在该对象的虚表中进行查询来选择正确的函数。
  每个类使用一个虚函数表，每个类对象用一个虚表指针。

对于包含虚函数的类，必须将一个虚表指针作为类中的第一个字段。在计算对象的总大小时，也必须考虑到虚表指针。这种情况在使用new操作符动态分配对象时最为明显，这时，传递给new的大小值不仅包括类（以及任何超类）中的所有显式声明的字段占用的空间，而且包括虚表指针所需的任何空间

```
.rdata:0000000000404530 ; `typeinfo name for'Apple
.rdata:0000000000404530 _ZTS5Apple      db '5Apple',0           ; DATA XREF: .rdata:00000000004044E8↑o
.rdata:0000000000404530                                         ; type descriptor name
.rdata:0000000000404537                 align 20h
.rdata:0000000000404540                 public _ZTS5Fruit
.rdata:0000000000404540 ; `typeinfo name for'Fruit
.rdata:0000000000404540 _ZTS5Fruit      db '5Fruit',0           ; DATA XREF: .rdata:0000000000404508↑o
.rdata:0000000000404540                                         ; type descriptor name
.rdata:0000000000404547                 align 10h
.rdata:0000000000404550                 public _ZTS6Banana
.rdata:0000000000404550 ; `typeinfo name for'Banana
.rdata:0000000000404550 _ZTS6Banana     db '6Banana',0          ; DATA XREF: .rdata:0000000000404518↑o
.rdata:0000000000404550                                         ; type descriptor name
.rdata:0000000000404558                 align 20h
.rdata:0000000000404560                 public _ZTV5Apple
.rdata:0000000000404560 ; `vtable for'Apple
.rdata:0000000000404560 _ZTV5Apple      dq 0                    ; offset to this
.rdata:0000000000404568                 dq offset _ZTI5Apple    ; `typeinfo for'Apple
.rdata:0000000000404570 off_404570      dq offset Apple__getFruit
.rdata:0000000000404570                                         ; DATA XREF: Apple::Apple(void)+18↑o
.rdata:0000000000404578                 align 20h
.rdata:0000000000404580                 public _ZTV5Fruit
.rdata:0000000000404580 ; `vtable for'Fruit
.rdata:0000000000404580 _ZTV5Fruit      dq 0                    ; offset to this
.rdata:0000000000404588                 dq offset _ZTI5Fruit    ; `typeinfo for'Fruit
.rdata:0000000000404590 off_404590      dq offset __cxa_pure_virtual
.rdata:0000000000404590                                         ; DATA XREF: Fruit::Fruit(void)+8↑o
.rdata:0000000000404598                 align 20h
.rdata:00000000004045A0                 public _ZTV6Banana
.rdata:00000000004045A0 ; `vtable for'Banana
.rdata:00000000004045A0 _ZTV6Banana     dq 0                    ; offset to this
.rdata:00000000004045A8                 dq offset _ZTI6Banana   ; `typeinfo for'Banana
.rdata:00000000004045B0 off_4045B0      dq offset _ZN6Banana8getFruitEv
.rdata:00000000004045B0                                         ; DATA XREF: Banana::Banana(void)+18↑o
.rdata:00000000004045B0                                         ; Banana::getFruit(void)
.rdata:00000000004045B8                 align 20h
.rdata:00000000004045C0 aGccX8664Win32S db 'GCC: (x86_64-win32-seh-rev0, Built by MinGW-W64 project) 8.1.0',0
```

```
.rdata:0000000000404000 ; std::piecewise_construct
.rdata:0000000000404000 _ZStL19piecewise_construct db    0
.rdata:0000000000404001 asc_404001      db '我是香蕉',0            ; DATA XREF: Banana::getFruit(void)+C↑o
.rdata:000000000040400A byte_40400A     db 0CEh                 ; DATA XREF: Apple__getFruit+C↑o
.rdata:000000000040400B                 db 0D2h
.rdata:000000000040400C                 db 0CAh
.rdata:000000000040400D                 db 0C7h
.rdata:000000000040400E                 db 0C6h
.rdata:000000000040400F                 db 0BBh
.rdata:0000000000404010                 db 0B9h
.rdata:0000000000404011                 db 0FBh
.rdata:0000000000404012                 db    0
.rdata:0000000000404013 ; char byte_404013[]
.rdata:0000000000404013 byte_404013     db 62h                  ; DATA XREF: main+1F↑o
.rdata:0000000000404013                                         ; Factory::CreateFruit(char *)+1D↑o
.rdata:0000000000404014                 db 61h
.rdata:0000000000404015                 db  6Eh ; n
.rdata:0000000000404016                 db  61h ; a
.rdata:0000000000404017                 db  6Eh ; n
.rdata:0000000000404018                 db  61h ; a
.rdata:0000000000404019                 db    0
.rdata:000000000040401A ; char[]
.rdata:000000000040401A                 db  61h ; a             ; DATA XREF: Factory::CreateFruit(char *)+4E↑o
.rdata:000000000040401B                 db  70h ; p
.rdata:000000000040401C                 db  70h ; p
.rdata:000000000040401D                 db  6Ch ; l
.rdata:000000000040401E                 db  65h ; e
.rdata:000000000040401F                 db    0
.rdata:0000000000404020 qword_404020    dq '误错入输'               ; DATA XREF: Factory::CreateFruit(char *):loc_402EB8↑o
.rdata:0000000000404028                 db    0
.rdata:0000000000404029 aHello          db 'hello',0            ; DATA XREF: main+67↑o
.rdata:000000000040402F                 align 10h
```

- this指针
  this可以看成是传递到所有非静态成员函数的第一个隐藏参数,Microsoft Visual C++利用thiscall调用约定，并将this传递到ECX寄存器中。
  从逆向工程的角度看，在调用函数之前，将一个地址转移到ECX寄存器中可能意味着两件事情。首先，该文件使用Visual C++编译；其次，该函数是一个成员函数。如果同一个地址被传递给两个或更多函数，我们可以得到结论，这些函数全都属于同一个类层次结构。
  如果发现一个函数向其他函数传递this指针，则这些函数可能和传递this的函数属于同一个类。

纯虚函数的标志

```
.rdata:0000000000404588                 dq offset _ZTI5Fruit    ; `typeinfo for'Fruit
.rdata:0000000000404590 off_404590      dq offset __cxa_pure_virtual
.rdata:0000000000404590                                         ; DATA XREF: Fruit::Fruit(void)+8↑o
.rdata:0000000000404598                 align 20h
.rdata:00000000004045A0                 public _ZTV6Banana
.rdata:00000000004045A0 ; `vtable for'Banana
.rdata:00000000004045A0 _ZTV6Banana     dq 0                    ; offset to this
.rdata:00000000004045A8                 dq offset _ZTI6Banana   ; `typeinfo for'Banana
.rdata:00000000004045B0 off_4045B0      dq offset _ZN6Banana8getFruitEv
.rdata:00000000004045B0                                         ; DATA XREF: Banana::Banana(void)+18↑o
.rdata:00000000004045B0                                         ; Banana::getFruit(void)
.rdata:00000000004045B8                 align 20h
.rdata:00000000004045C0                 public _ZTV9AbFactory
.rdata:00000000004045C0 ; `vtable for'AbFactory
.rdata:00000000004045C0 _ZTV9AbFactory  dq 0                    ; offset to this
.rdata:00000000004045C8                 dq offset _ZTI9AbFactory ; `typeinfo for'AbFactory
.rdata:00000000004045D0 off_4045D0      dq offset __cxa_pure_virtual
```

main()用到AppleFactory 所以data段不会出现AppleFactory类

```
.rdata:00000000004045D0 ; `vtable for'PesiBottle
.rdata:00000000004045D0 _ZTV10PesiBottle dq 0                   ; offset to this
.rdata:00000000004045D8                 dq offset _ZTI10PesiBottle ; `typeinfo for'PesiBottle
.rdata:00000000004045E0 off_4045E0      dq offset _text_51      ; DATA XREF: PesiBottle::PesiBottle(void)+18↑o
.rdata:00000000004045E8                 align 10h
.rdata:00000000004045F0                 public _ZTV11PesiFactory
.rdata:00000000004045F0 ; `vtable for'PesiFactory
.rdata:00000000004045F0 _ZTV11PesiFactory dq 0                  ; offset to this
.rdata:00000000004045F8                 dq offset _ZTI11PesiFactory ; `typeinfo for'PesiFactory
.rdata:0000000000404600 off_404600      dq offset _ZN11PesiFactory10CreateColaEv
.rdata:0000000000404600                                         ; DATA XREF: PesiFactory::PesiFactory(void)+18↑o
.rdata:0000000000404600                                         ; PesiFactory::CreateCola(void)
.rdata:0000000000404608                 dq offset _ZN11PesiFactory12CreateBottleEv ; PesiFactory::CreateBottle(void)
.rdata:0000000000404610                 public _ZTV4Cola
.rdata:0000000000404610 ; `vtable for'Cola
.rdata:0000000000404610 _ZTV4Cola       dq 0                    ; offset to this
.rdata:0000000000404618                 dq offset _ZTI4Cola     ; `typeinfo for'Cola
.rdata:0000000000404620 off_404620      dq offset __cxa_pure_virtual
.rdata:0000000000404620                                         ; DATA XREF: Cola::Cola(void)+8↑o
.rdata:0000000000404628                 align 10h
.rdata:0000000000404630                 public _ZTV6Bottle
.rdata:0000000000404630 ; `vtable for'Bottle
.rdata:0000000000404630 _ZTV6Bottle     dq 0                    ; offset to this
.rdata:0000000000404638                 dq offset _ZTI6Bottle   ; `typeinfo for'Bottle
.rdata:0000000000404640 off_404640      dq offset __cxa_pure_virtual
.rdata:0000000000404640                                         ; DATA XREF: Bottle::Bottle(void)+8↑o
.rdata:0000000000404648                 align 10h
.rdata:0000000000404650                 public _ZTV8PesiCola
.rdata:0000000000404650 ; `vtable for'PesiCola
.rdata:0000000000404650 _ZTV8PesiCola   dq 0                    ; offset to this
.rdata:0000000000404658                 dq offset _ZTI8PesiCola ; `typeinfo for'PesiCola
.rdata:0000000000404660 off_404660      dq offset _ZN8PesiCola7SaynameEv
.rdata:0000000000404660                                         ; DATA XREF: PesiCola::PesiCola(void)+18↑o
.rdata:0000000000404660                                         ; PesiCola::Sayname(void)
.rdata:0000000000404668                 align 10h
.rdata:0000000000404670                 public _ZTV9AbFactory
.rdata:0000000000404670 ; `vtable for'AbFactory
.rdata:0000000000404670 _ZTV9AbFactory  dq 0                    ; offset to this
.rdata:0000000000404678                 dq offset _ZTI9AbFactory ; `typeinfo for'AbFactory
.rdata:0000000000404680 off_404680      dq offset __cxa_pure_virtual
```

2.单例模式

单例模式的构造函数是私有函数，但是在反汇编看不出私有属性。

- 类继承权限

类的继承权限并不会影响子类继承父类子类所拥有的父类的成员变量个数，换句话说，不管父类的成员变量是什么权限，子类都完全拥有一份父类的成员变量的拷贝

所以只能通过观察其属性成员是static定义的，大概率是单例模式

```
0000000040803C                  align 20h
0000000040803C
0000000040803C                  public _ZN10Singelton28m_singerE
0000000040803C ; void *Singelton2::m_singer
0000000040803C _ZN10Singelton28m_singerE dq ?           ; DATA XREF: __static_initialization_and_destruction_0(int,int)+4C↑w
0000000040803C                                          ; _text_54+A↑r ...
0000000040803C                  public _ZN10Singelton27m_countE
0000000040803C ; Singelton2::m_count
0000000040803C _ZN10Singelton27m_countE dd ?            ; DATA XREF: Singelton2::FreeInstance(void)+33↑w
0000000040803C                                          ; Singelton2::printT(void)+1E↑r ...
0000000040804C ; std::ios_base::Init std::__ioinit
0000000040804C _ZStL8__ioinit  db    ? ;               ; DATA XREF: __tcf_0+8↑o
0000000040804C                                          ; __static_initialization_and_destruction_0(int,int)+1F↑o
0000000040804D                  db    ? ;
0000000040804E                  db    ? ;
0000000040804F                  db    ? ;
0000000040804F
0000000040804F initialized      dd ?                    ; DATA XREF:   main↑r
```

release模式(静态成员的动态封装类)

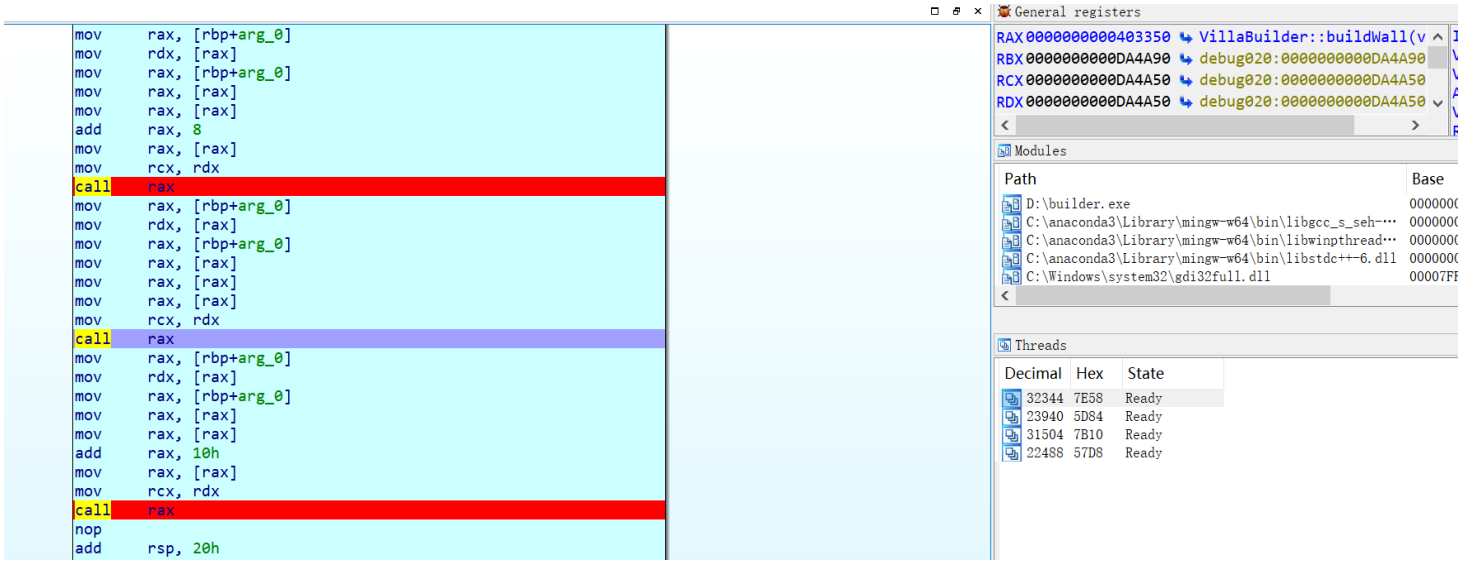
```
.data:004053FC ; int Singelton2::m_count
.data:004053FC ?m_count@Singelton2@@0HA dd ?             ; DATA XREF: _dynamic_initializer_for__Singelton2__m_singer__+54↑w
.data:004053FC                                          ; _main+63↑w ...
.data:00405400 ; Singelton2 *Singelton2::m_singer
.data:00405400 ?m_singer@Singelton2@@0PAV1@A dd ?        ; DATA XREF: _dynamic_initializer_for__Singelton2__m_singer__+4A↑w
.data:00405400                                          ; _dynamic_initializer_for__Singelton2__m_singer__+6B↑w ...
.data:00405404 ; void (__stdcall *const __dyn_tls_dtor_callback)(void *, unsigned int, void *)
.data:00405404 ___dyn_tls_dtor_callback dd ?             ; DATA XREF: ___scrt_get_dyn_tls_dtor_callback↑o
.data:00405408 ; void (__stdcall *const __dyn_tls_init_callback)(void *, unsigned int, void *)
```

3.建造者模式

指导者类里含有一个建造者对象，建造者类里含有一个建造物对象

。可以通过看构造函数观察特征

另一个特征是director类会按照逻辑顺序依次调用builder类里的函数，如图



```
VillaBuilder *__fastcall VillaBuilder::VillaBuilder(VillaBuilder *this)
{
  House *v1; // rbx
  VillaBuilder *result; // rax
  VillaBuilder *v3; // [rsp+40h] [rbp-40h]

  v3 = this;
  Builder::Builder(this);
  *(_QWORD *)v3 = off_4055C0;
  v1 = (House *)operator new(0x60ui64);
  House::House(v1);
  result = v3;
  *((_QWORD *)v3 + 1) = v1;
  return result;
}
```

4.原型模式

- 实质就是找拷贝构造

由此可见调用子类的拷贝构造函数会先调用父类的构造函数，然后在调用当前类的拷贝构造，这里的off_85600就是 vptr ，从虚函数表中也可以看见，子类覆盖了父类的虚函数就会指向子类的虚函数。

若没有覆盖，表项中依旧是指向父类的函数地址，而且顺序是按照父类的虚函数表顺序排列，子类中父类没有的虚函数会按顺序继续排在后面，不同类的虚函数表其实都是在编译期就已经确定了的，不同类的虚函数表处于临近的内存区域。

二、创建型

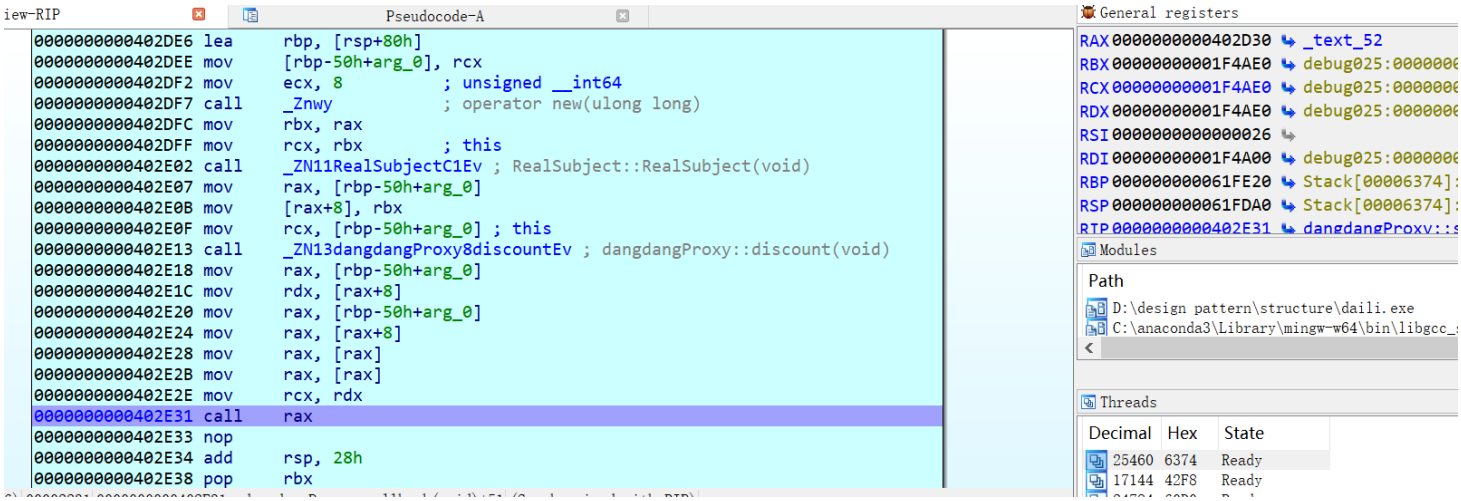
1.代理模式

- 代理类a包含被代理类b的实例，a,b类实现协议类protocol
- 代理类a和被代理类b接口相同

代理类a包含被代理类b的实例

```
04500                   public _ZTI13dangdangProxy
04500 ; `typeinfo for'dangdangProxy
04500 _ZTI13dangdangProxy dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
04500                                           ; DATA XREF: .rdata:0000000000404588↓o
04500                                           ; reference to RTTI's type class
04508                   dq offset _ZTS13dangdangProxy ; reference to type's name
04510                   dq offset _ZTI7Subject  ; reference to parent's type name
04518                   align 20h
04520 ; public Subject
04520                   public _ZTI7Subject
04520 ; `typeinfo for'Subject
04520 _ZTI7Subject      dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
04520                                           ; DATA XREF: .rdata:00000000004044F0↑o
04520                                           ; .rdata:0000000000404510↑o ...
04520                                           ; reference to RTTI's type class
04528                   dq offset _ZTS7Subject  ; reference to type's name
04530                   public _ZTS11RealSubject
04530 ; `typeinfo name for'RealSubject
```

代理类接口的汇编，会发现call了被代理类的接口(实际情况应该会有相同的输入参数)



2.装饰模式

- 分为若干个具体类和若干个装饰类，都是继承自同一个抽象类。

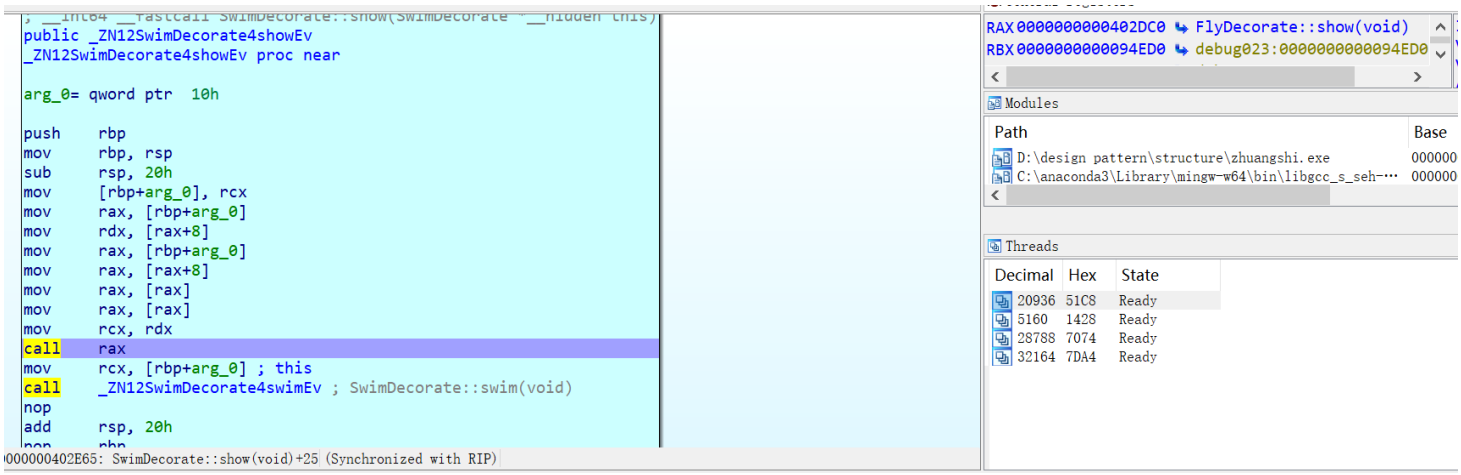
```
.rdata:00000000004054E0 ; public FlyDecorate :
.rdata:00000000004054E0 ;   public /* offset 0x0 */ Car
.rdata:00000000004054E0                 public _ZTI11FlyDecorate
.rdata:00000000004054E0 ; `typeinfo for'FlyDecorate
.rdata:00000000004054E0 _ZTI11FlyDecorate dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
.rdata:00000000004054E0                                         ; DATA XREF: .rdata:0000000000405598↓o
.rdata:00000000004054E0                                         ; reference to RTTI's type class
.rdata:00000000004054E8                 dq offset _ZTS11FlyDecorate ; reference to type's name
.rdata:00000000004054F0                 dq offset _ZTI3Car      ; reference to parent's type name
.rdata:00000000004054F8                 align 20h
.rdata:0000000000405500 ; public SwimDecorate :
.rdata:0000000000405500 ;   public /* offset 0x0 */ Car
.rdata:0000000000405500                 public _ZTI12SwimDecorate
.rdata:0000000000405500 ; `typeinfo for'SwimDecorate
.rdata:0000000000405500 _ZTI12SwimDecorate dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
.rdata:0000000000405500                                         ; DATA XREF: .rdata:00000000004055B8↓o
.rdata:0000000000405500                                         ; reference to RTTI's type class
.rdata:0000000000405508                 dq offset _ZTS12SwimDecorate ; reference to type's name
.rdata:0000000000405510                 dq offset _ZTI3Car      ; reference to parent's type name
.rdata:0000000000405518                 align 20h
.rdata:0000000000405520 ; public Car
.rdata:0000000000405520                 public _ZTI3Car
.rdata:0000000000405520 ; `typeinfo for'Car
.rdata:0000000000405520 _ZTI3Car        dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
.rdata:0000000000405520                                         ; DATA XREF: .rdata:00000000004054F0↑o
.rdata:0000000000405520                                         ; .rdata:0000000000405510↑o ...
.rdata:0000000000405520                                         ; reference to RTTI's type class
.rdata:0000000000405528                 dq offset _ZTS3Car      ; reference to type's name
.rdata:0000000000405530 ; public RunCar :
.rdata:0000000000405530 ;   public /* offset 0x0 */ Car
.rdata:0000000000405530                 public _ZTI6RunCar
.rdata:0000000000405530 ; `typeinfo for'RunCar
.rdata:0000000000405530 _ZTI6RunCar     dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
.rdata:0000000000405530                                         ; DATA XREF: .rdata:00000000004055F8↓o
.rdata:0000000000405530                                         ; reference to RTTI's type class
.rdata:0000000000405538                 dq offset _ZTS6RunCar   ; reference to type's name
.rdata:0000000000405540                 dq offset _ZTI3Car      ; reference to parent's type name
.rdata:0000000000405548                 align 10h
.rdata:0000000000405550                 public _ZTS11FlyDecorate
```

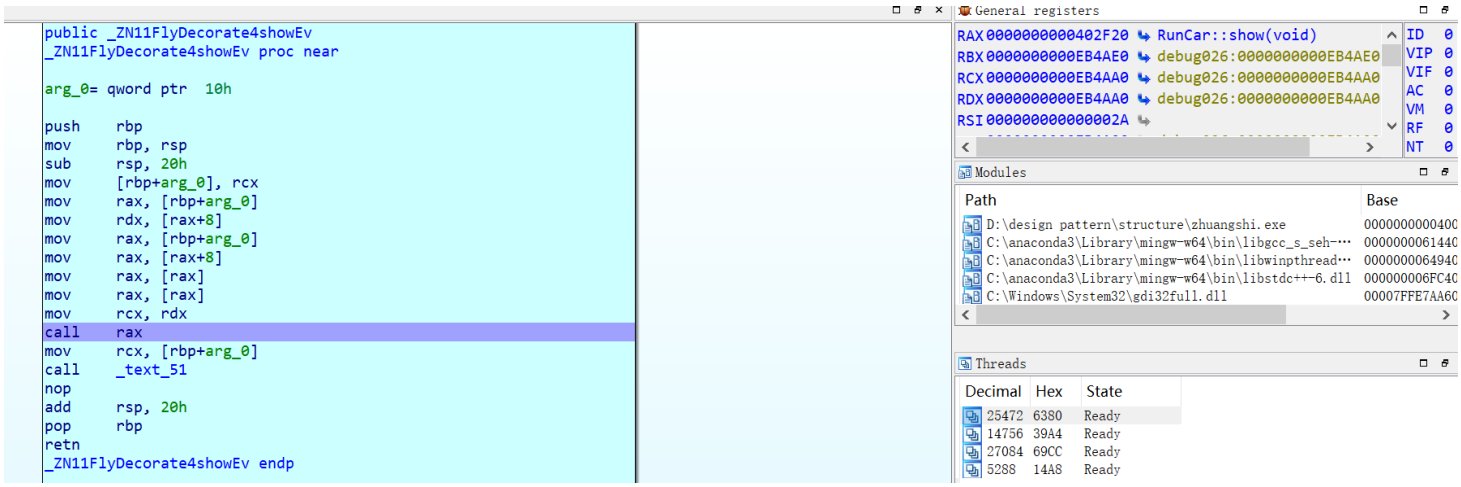
- 每个装饰类都含有一个抽象类对象
- 每个装饰类都含有与抽象类有相同接口的虚函数，不同于具体类的虚函数是对抽象类的具体实现，装饰类的这个虚函数一定会调用抽象对象的虚函数，同时加上装饰类新增的自定义函数。

具体类对抽象类纯虚函数的实现

```
; Attributes: bp-based frame

; __int64 __fastcall RunCar::show(RunCar *__hidden this)
public _ZN6RunCar4showEv
_ZN6RunCar4showEv proc near

arg_0= qword ptr  10h

push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+arg_0], rcx
lea     rdx, asc_405001 ; "跑车"
mov     rcx, cs:_refptr__ZSt4cout
call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operator<<<std::char_traits<char>>(std::basic_ostream<char,std::char_traits<char>>
mov     rdx, cs:_refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
mov     rcx, rax
call    _ZNSolsEPFRSoS_E ; std::ostream::operator<<(std::ostream & (*)(std::ostream &))
nop
add     rsp, 20h
pop     rbp
retn
_ZN6RunCar4showEv endp
```

装饰类的相同接口，调用了抽象对象(可以是父类指针指向子对象)的虚函数





实现了自定义的装饰

```
; Attributes: bp-based frame

; __int64 __fastcall SwimDecorate::swim(SwimDecorate *__hidden this)
public _ZN12SwimDecorate4swimEv
_ZN12SwimDecorate4swimEv proc near

arg_0= qword ptr  10h

push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+arg_0], rcx
lea     rdx, asc_405006 ; "游车"
mov     rcx, cs:_refptr__ZSt4cout
call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operator<<<std::char_traits<char>>(std::basic_ostream<char,std::char_traits<char>> &,char const*)
mov     rdx, cs:_refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
mov     rcx, rax
call    _ZNSolsEPFRSoS_E ; std::ostream::operator<<(std::ostream & (*)(std::ostream &))
nop
add     rsp, 20h
pop     rbp
retn
_ZN12SwimDecorate4swimEv endp
```

```
; Attributes: bp-based frame

public _text_51
_text_51 proc near

arg_0= qword ptr  10h

push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+arg_0], rcx
lea     rdx, asc_40500B ; "飞车"
mov     rcx, cs:_refptr__ZSt4cout
call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operator<<<std::char_traits<char>>(std::basic_ostream<char,std::char_traits<ch
mov     rdx, cs:_refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
mov     rcx, rax
call    _ZNSolsEPFRSoS_E ; std::ostream::operator<<(std::ostream & (*)(std::ostream &))
nop
add     rsp, 20h
pop     rbp
retn
_text_51 endp
```

3.适配器模式

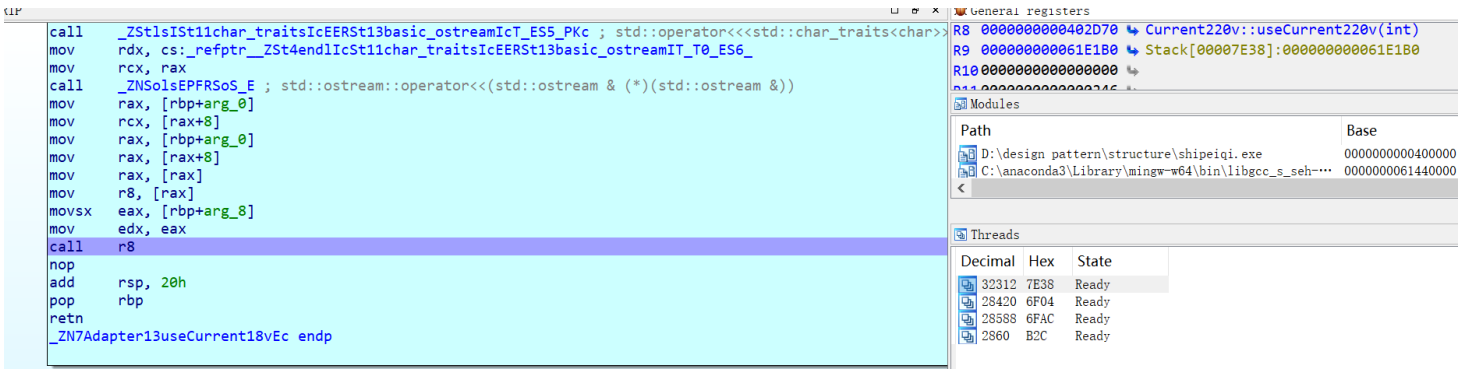
- 将一个接口(已有接口类)转换成客户希望的另一个接口(用户接口类)
- 适配器类里会有一个已有接口类对象

```
1  Adapter *__fastcall Adapter::Adapter(Adapter *this, Current220v *a2)
2  {
3    Adapter *result; // rax
4    Adapter *v3; // [rsp+30h] [rbp+10h]
5    Current220v *v4; // [rsp+38h] [rbp+18h]
6
7    v3 = this;
8    v4 = a2;
9    text_51(this);
10   *(_QWORD *)v3 = &off_4045A0;
11   result = v3;
12   *((_QWORD *)v3 + 1) = v4;
13   return result;
14 }
```

- 适配器类的父类是用户接口类(抽象类)

```
; public Current18v
                public _ZTI10Current18v
; `typeinfo for'Current18v
_ZTI10Current18v dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
                                        ; DATA XREF: .rdata:0000000000404510↓o
                                        ; .rdata:0000000000404558↓o
                                        ; reference to RTTI's type class
                dq offset _ZTS10Current18v ; reference to type's name
; public Current220v
                public _ZTI11Current220v
; `typeinfo for'Current220v
_ZTI11Current220v dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
                                        ; DATA XREF: .rdata:0000000000404578↓o
                                        ; reference to RTTI's type class
                dq offset _ZTS11Current220v ; reference to type's name
; public Adapter :
;   public /* offset 0x0 */ Current18v
                public _ZTI7Adapter
; `typeinfo for'Adapter
_ZTI7Adapter    dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
                                        ; DATA XREF: .rdata:0000000000404598↓o
                                        ; reference to RTTI's type class
                dq offset _ZTS7Adapter  ; reference to type's name
                dq offset _ZTI10Current18v ; reference to parent's type name
                align 20h
                public _ZTS10Current18v
```

- 适配器类虚函数是对用户接口类纯虚函数的具体实现，且里面一定会调用已有接口类的函数(且接口一般与该虚函数的接口不同)



4.桥接模式

- 将抽象部分与它的实现部分分离,分为桥接类(抽象部分，如car)和实现类(实现部分，如enginee),两个大类各自可有抽象类和具体类。如图，找到了两个抽象类，分别为car和enginee

```
004044E0 ; public Car
004044E0                 public _ZTI3Car
004044E0 ; `typeinfo for'Car
004044E0 _ZTI3Car        dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
004044E0                                         ; DATA XREF: .rdata:0000000000404500↓o
004044E0                                         ; .rdata:0000000000404588↓o
004044E0                                         ; reference to RTTI's type class
004044E8                 dq offset _ZTS3Car      ; reference to type's name
004044F0 ; public WBM6 :
004044F0 ;   public /* offset 0x0 */ Car
004044F0                 public _ZTI4WBM6
004044F0 ; `typeinfo for'WBM6
004044F0 _ZTI4WBM6       dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
004044F0                                         ; DATA XREF: .rdata:00000000004045A8↓o
004044F0                                         ; reference to RTTI's type class
004044F8                 dq offset _ZTS4WBM6     ; reference to type's name
00404500                 dq offset _ZTI3Car      ; reference to parent's type name
00404508                 align 10h
00404510 ; public Engine
00404510                 public _ZTI6Engine
00404510 ; `typeinfo for'Engine
00404510 _ZTI6Engine     dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
00404510                                         ; DATA XREF: .rdata:0000000000404530↓o
00404510                                         ; .rdata:00000000004045C8↓o
00404510                                         ; reference to RTTI's type class
00404518                 dq offset _ZTS6Engine   ; reference to type's name
00404520 ; public cc_4400 :
00404520 ;   public /* offset 0x0 */ Engine
00404520                 public _ZTI7cc_4400
00404520 ; `typeinfo for'cc_4400
00404520 _ZTI7cc_4400    dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
00404520                                         ; DATA XREF: .rdata:00000000004045E8↓o
00404520                                         ; reference to RTTI's type class
00404528                 dq offset _ZTS7cc_4400  ; reference to type's name
00404530                 dq offset _ZTI6Engine   ; reference to parent's type name
00404538                 align 20h
```

- 桥接类的抽象类里含有实现类的抽象类对象
  通过子类的构造函数找到了抽象类car的构造函数，改名为car::car,注意a2是enginee类强转到int64

```
1  WBM6 *__fastcall WBM6::WBM6(WBM6 *this, Engine *a2)
2  {
3    WBM6 *result; // rax
4    WBM6 *v3; // [rsp+30h] [rbp+10h]
5
6    v3 = this;
7    text_52(this, (__int64)a2);
8    result = v3;
9    *(_QWORD *)v3 = &off_4045B0;
10   return result;
11 }
```

```
1 _QWORD *__fastcall text_52(_QWORD *a1, __int64 a2)
2 {
3   _QWORD *result; // rax
4
5   *a1 = &off_404590;
6   result = a1;
7   a1[1] = a2;                                   // a2 是enginee类
8   return result;
9 }
```

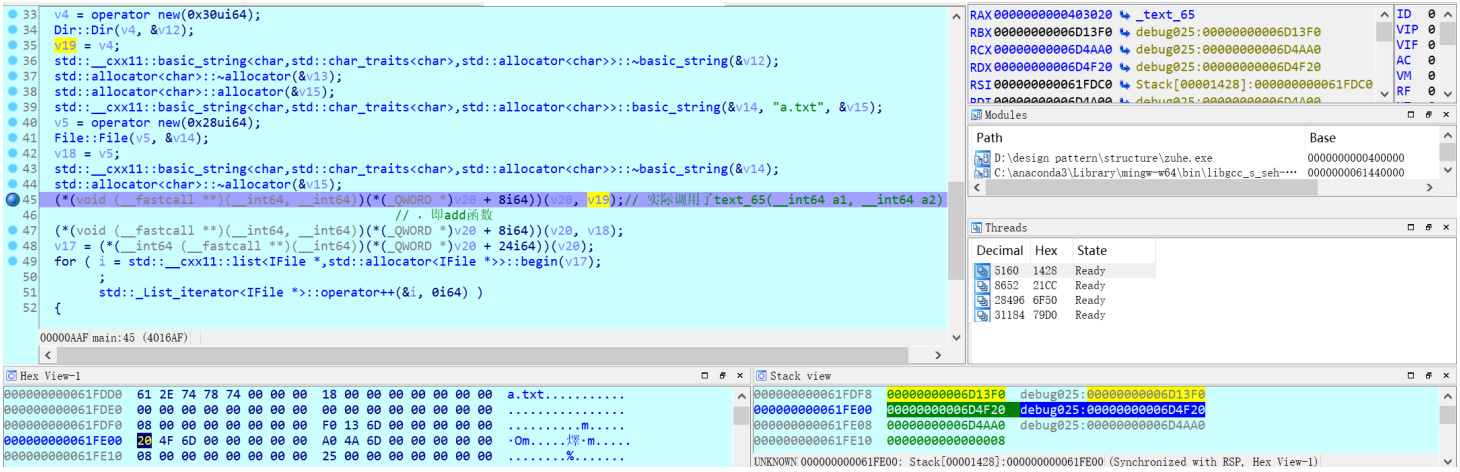
- 且两者纯虚函数接口相同，桥接类的虚函数具体实现一定会调用实现类对象的虚函数

5.组合模式

- 单个对象和组合对象继承自同个抽象类，接口一致，但实现可能不一致（如返回NULL）

```
; public Dir :
;   public /* offset 0x0 */ IFile
                public _ZTI3Dir
; `typeinfo for'Dir
_ZTI3Dir        dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
                                        ; DATA XREF: .rdata:0000000000405568↓o
                                        ; reference to RTTI's type class
                dq offset _ZTS3Dir      ; reference to type's name
                dq offset _ZTI5IFile    ; reference to parent's type name
                align 20h
; public File :
;   public /* offset 0x0 */ IFile
                public _ZTI4File
; `typeinfo for'File
_ZTI4File       dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
                                        ; DATA XREF: .rdata:0000000000405598↓o
                                        ; reference to RTTI's type class
                dq offset _ZTS4File     ; reference to type's name
                dq offset _ZTI5IFile    ; reference to parent's type name
                align 20h
; public IFile
                public _ZTI5IFile
; `typeinfo for'IFile
_ZTI5IFile      dq offset __imp__ZTVN10__cxxabiv117__class_type_infoE+10h
                                        ; DATA XREF: .rdata:00000000004054F0↑o
                                        ; .rdata:0000000000405510↑o ...
                                        ; reference to RTTI's type class
                dq offset _ZTS5IFile    ; reference to type's name
```

- main函数定义的实例是树形对象，对象往往包含类似于装有抽象类对象的list容器



如图，v20是根节点root,v19是子节点Dir,v18是子节点File

6.外观模式

- 外观类中有多个同层次的子系统类

直接查看Facade的构造函数，如果是系统自动定义的会显示表示出创建的成员对象,但在这里我们是通过new的方式的默认构造函数，所以会出现三行分配空间的代码，比较难读懂

可以转变思路，直接看Facade的函数，发现dothing函数里调用了三个子系统的dothing函数，说明外观类中有三个子系统类的对象

```
__int64 __fastcall Facade::dothing(Facade *this)
{
  __int64 v1; // rax
  Facade *v3; // [rsp+30h] [rbp+10h]

  v3 = this;
  v1 = *(_QWORD *)this;
  SubSystemA::dothing();
  SubSystemB::dothing(*((SubSystemB **)v3 + 1));
  return SubSystemC::dothing(*((SubSystemC **)v3 + 2));
}
```

7.享元模式

> 音乐服务根据收费划分出免费用户和会员用户，免费用户只能听部分免费音乐，会员用户可以听全部的音乐，并且可以下载。虽然权限上二者间有一些区别，但是他们所享受的音乐来是自于同一个音乐库，这样所有的音乐都只需要保存一份就可以了。另外如果出现音乐库里没有的音乐时，则需要新增该音乐，然后其他服务也可以享受新增的音乐，相当于享元池或缓存池的功能

可以动调，通过创建相同对象来看看享元池是否只保存了一次。或者输入条件相同时查询的是否是同一个对象。

```
        v16);
  v11 = std::map<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>,Music *,std::less<std::__cxx11::basic_string<char,s
  if ( (unsigned __int8)std::_Rb_tree_iterator<std::pair<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>> const,Music
                          &v8,
                          &v11) )
  {
    v3 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "请输入音乐的作者、流行度");
    std::ostream::operator<<(v3, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v6);
    v4 = std::operator>><char,std::char_traits<char>,std::allocator<char>>(refptr__ZSt3cin, &v6);
    std::istream::operator>>(v4, &v7);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v12, &v6);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v13, v16);
    v14 = FlyWeightMusicFactory::AddMusic(v15);// 只有满足条件才保存到库中
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v13);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v12);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v6);
  }
```

可看出数据结构进而推断出享元池的保存形式

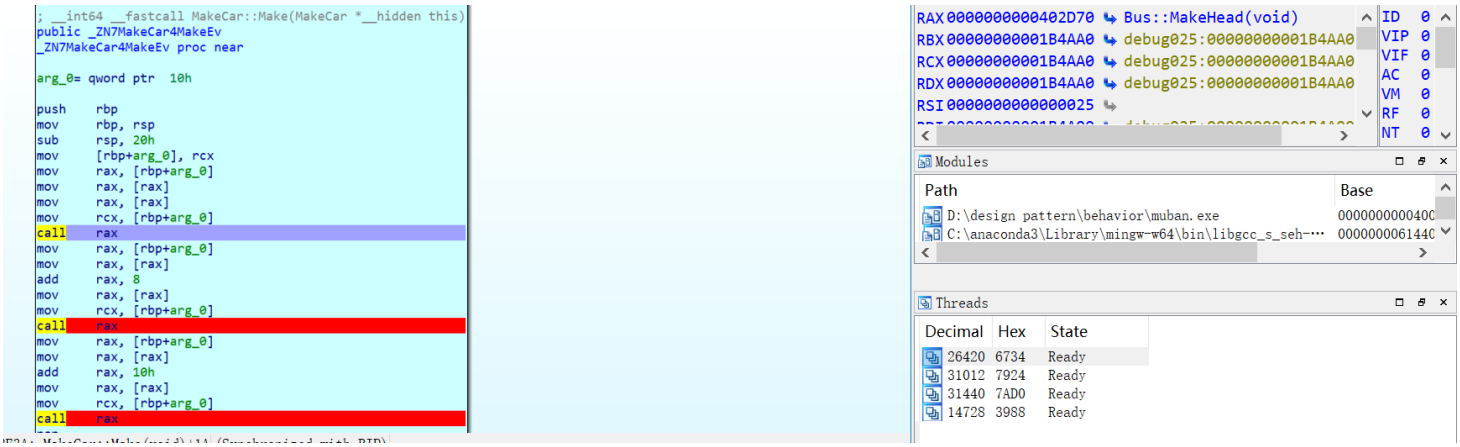
```
__int64 __fastcall FlyWeightMusicFactory::AddMusic(__int64 a1, __int64 a2, __int64 a3, signed int a4, _BYTE *a5)
{
  __int64 v5; // rbx
  __int64 v6; // rbx
  __int64 v8; // [rsp+28h] [rbp-58h]
  char v9; // [rsp+30h] [rbp-50h]
  char v10; // [rsp+50h] [rbp-30h]
  char v11; // [rsp+70h] [rbp-10h]
  __int64 v12; // [rsp+80h] [rbp+0h]
  char v13; // [rsp+B0h] [rbp+30h]
  char v14; // [rsp+D0h] [rbp+50h]
  char v15; // [rsp+F0h] [rbp+70h]
  char v16; // [rsp+100h] [rbp+80h]
  __int64 v17; // [rsp+160h] [rbp+E0h]
  __int64 v18; // [rsp+168h] [rbp+E8h]
  unsigned int v19; // [rsp+178h] [rbp+F8h]

  v17 = a1;
  v18 = a2;
  v19 = a4;
  v8 = 0i64;
  if ( a4 > 4 )
  {
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v13, a3);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v14, v18);
    v6 = operator new(0x50ui64);
    VipMusic::VipMusic(v6, &v13, v19, &v14);
    v8 = v6;
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v14);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v13);
    std::pair<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>,Music *>::pair<std::__cxx11::basic_string<char,std::char_tr
      &v16,
      v18,
      &v8);
    std::map<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>,Music *,std::less<std::__cxx11::basic_string<char,std::char_
      &v15,
      v17 + 48,
      &v16);
    std::pair<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>,Music *>::~pair(&v16);
    *a5 = 1;
  }
```

三、行为型

1.模板方法模式

- 抽象类里有模板函数来决定其他函数的执行顺序和逻辑

```
560 ; `vtable for'MakeCar
560 _ZTV7MakeCar    dq 0                    ; offset to this
568                 dq offset _ZTI7MakeCar  ; `typeinfo for'MakeCar
570 off_404570      dq offset __cxa_pure_virtual
570                                         ; DATA XREF: MakeCar::MakeCar(void)+8↑o
578                 dq offset __cxa_pure_virtual
580                 dq offset __cxa_pure_virtual
588                 dq offset _ZN7MakeCar4MakeEv ; MakeCar::Make(void)
590 aGccX8664Win32S db 'GCC: (x86_64-win32-seh-rev0, Built by MinGW-W64 project) 8.1.0',0
5CF                 align 10h
```



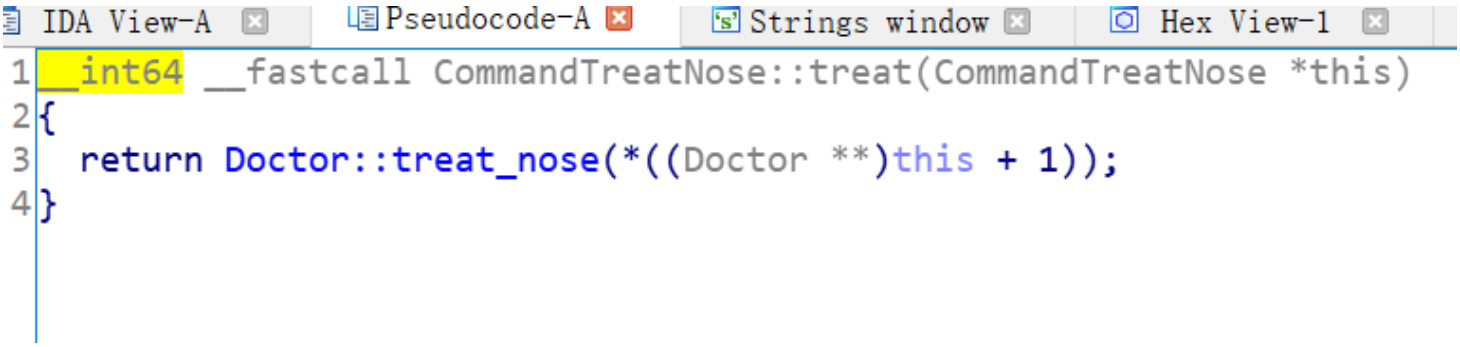
2.命令模式

> 一个对象(Car)调用另一个对象(Engine)的过程是：创建目标对象实例；设置调用参数；调用目标对象的方法。如果调用过程较繁琐，或者有多处调用，有必要用一个专门的类对这种调用过程进行封装，即command类

- 虽然有命令排序、批量提交等复杂操作，但本质还是看command类的特征
- command类(抽象类或具体类)一般包含一个要调用的目标对象

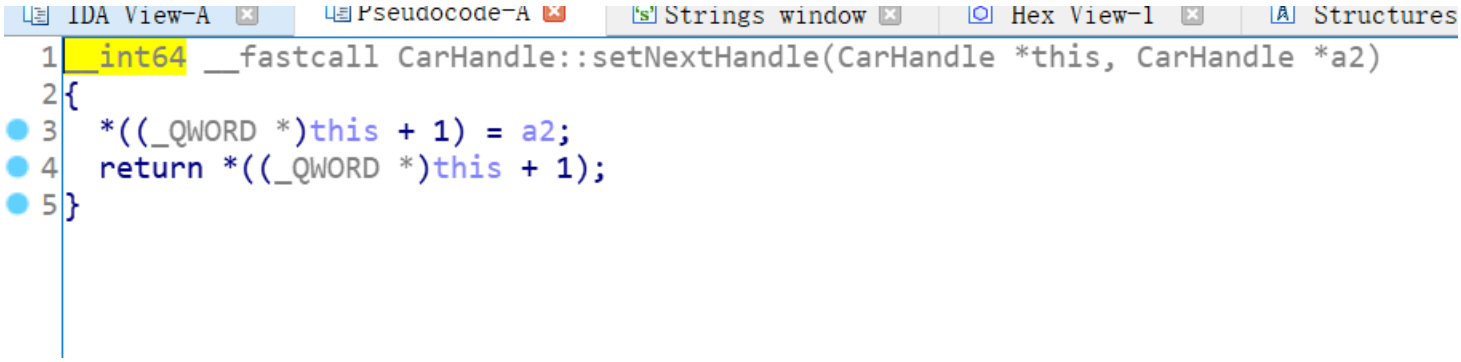
```
CommandTreatNose *__fastcall CommandTreatNose::CommandTreatNose(CommandTreatNose *this, Doctor *a2)
{
  CommandTreatNose *result; // rax
  CommandTreatNose *v3; // [rsp+30h] [rbp+10h]
  Doctor *v4; // [rsp+38h] [rbp+18h]

  v3 = this;
  v4 = a2;
  Command::Command(this);
  *(_QWORD *)v3 = &off_4055B0;
  result = v3;
  *((_QWORD *)v3 + 1) = v4;
  return result;
}
```

- 命令的执行函数一定会调用目标对象的方法，一般比较繁琐或调用得比较多



3.职责链模式

- 子类完全平等，即哪个都有可能是下一个事件，即所有类都会有一个设置下一个处理单元的函数，(可能在抽象类实现然后继承抽象类)且参数必须是抽象类




```
v3 = (HeadCarHandle *)operator new(0x10ui64);
HeadCarHandle::HeadCarHandle(v3);
v4 = v3;
v5 = (BodyCarHandle *)operator new(0x10ui64);
BodyCarHandle::BodyCarHandle(v5);
v6 = v5;
v7 = (TailCarHandle *)operator new(0x10ui64);
TailCarHandle::TailCarHandle(v7);
CarHandle::setNextHandle(v4, v6);
CarHandle::setNextHandle(v6, v7);
CarHandle::setNextHandle(v7, 0i64);
(**(void (__fastcall ***)(CarHandle *))v4)(v4);
```

- 一般抽象类或子类会有一个抽象类对象，来作为下一个处理单元

```
CarHandle *__fastcall BodyCarHandle::BodyCarHandle(BodyCarHandle *this)
{
  CarHandle *result; // rax
  CarHandle *v2; // [rsp+30h] [rbp+10h]

  v2 = this;
  CarHandle::CarHandle(this);
  result = v2;
  *(_QWORD *)v2 = &off_4055A0;
  return result;
}
```

- 子类的执行函数Handle()里会调用下一个处理单元的执行函数,即调用了相同的函数(除非恰好到了最后一个任务，下一个是结束任务NULL)

4.策略模式

> 定义一系列算法类，将每一个算法封装起来，并让它们可以相互替换，策略模式让算法独立于使用它的客户而变化

- 要做到可替换性，实现类含有的策略类对象或函数参数一定要是抽象类

```
1 Context *__fastcall Context::setStrategy(Context *this, Strategy *a2)
2 {
3   Context *result; // rax
4
5   result = this;
6   *(_QWORD *)this = a2;
7   return result;
8 }
```

5.中介者模式

> 中介者模式将一个网状的系统结构变成一个以中介者对象为中心的星形结构，在这个星型结构中，使用中介者对象与其他对象的一对多关系来取代原有对象之间的多对多关系。所有成员通过中介者交互

- 中介者类一般含有多个相同的抽象类或子类，或者是装着这些类的容器等，一般是一对一或一对多
- 中介者类的函数方法会用到这些其他类的对象以实现交互

```
1  __int64 __fastcall Mediator::getPair(Mediator *this)
2  {
3    char v1; // bl
4    __int64 v2; // rax
5    __int64 result; // rax
6    int v4; // ebx
7    __int64 v5; // rax
8    __int64 v6; // [rsp+0h] [rbp-80h]
9    char v7; // [rsp+20h] [rbp-60h]
10   char v8; // [rsp+40h] [rbp-40h]
11   Mediator *v9; // [rsp+80h] [rbp+0h]
12
13   v9 = this;
14   ZN6Person6getSexB5cxx11Ev(&v6 + 4, *((_QWORD *)this + 1));// 存储着其他类的对象
15   ZN6Person6getSexB5cxx11Ev(&v8, *((_QWORD *)v9 + 2));// 存储着其他类的对象
16   v1 = std::operator==<char>(&v8, &v7);
17   std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_
18   std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_
```

- 其他的对象可能含有一个中介者类对象，且会用到中介者类的函数，并把自身的类作为参数传进去，以实现交互

```
v3 = (Mediator *)operator new(0x18ui64);
Mediator::Mediator(v3);
v18 = v3;
std::allocator<char>::allocator(&v10);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v9, "小芳", &v10);
v4 = (Person *)operator new(0x58ui64);
Woman::Woman(v4, (__int64)&v9, 5u, (__int64)v18);
v17 = v4;
```

```
; Attributes: bp-based frame

; __int64 __fastcall Woman::getPair(Woman *__hidden this, Person *)
public _ZN5Woman7getPairEP6Person
_ZN5Woman7getPairEP6Person proc near

arg_0= qword ptr  10h
arg_8= qword ptr  18h

push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+arg_0], rcx
mov     [rbp+arg_8], rdx
mov     rax, [rbp+arg_0]
mov     rax, [rax+50h]
mov     rdx, [rbp+arg_0] ; Person *
mov     rcx, rax         ; this
call    _ZN8Mediator8setWomanEP6Person ; Mediator::setWoman(Person *)
mov     rax, [rbp+arg_0]
mov     rax, [rax+50h]
mov     rdx, [rbp+arg_8] ; Person *
mov     rcx, rax         ; this
call    _ZN8Mediator6setManEP6Person ; Mediator::setMan(Person *)
mov     rax, [rbp+arg_0]
mov     rdx, [rax+50h]
mov     rax, [rbp+arg_0]
```

6.观察者模式

> 定义对象之间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。

- 一般通知者类里会有一个装有被通知者类对象的容器

```
__int64 __fastcall Secretary::Secretary(Secretary *this)
{
  Secretary *v2; // [rsp+30h] [rbp+10h]

  v2 = this;
  std::__cxx11::list<Employee *,std::allocator<Employee *>>::list(this);// 装有被通知者类对象的容器
  return std::__cxx11::list<Employee *,std::allocator<Employee *>>::clear(v2);
}
```

- 通知者类的执行函数会遍历容器内的每一个被通知者类对象，并调用被通知者类对象函数(找循环)

IDA View-A | Pseudocode-A | Strings window | Hex View-1 | Structures | Enums | In

```
__int64 __fastcall Secretary::setaction(__int64 a1, __int64 a2)
{
  __int64 result; // rax
  __int64 v3; // rbx
  __int64 i; // [rsp+20h] [rbp-60h]
  __int64 v5; // [rsp+28h] [rbp-58h]
  char v6; // [rsp+30h] [rbp-50h]
  __int64 v7; // [rsp+70h] [rbp-10h]
  __int64 v8; // [rsp+78h] [rbp-8h]

  v7 = a1;
  v8 = a2;
  for ( i = std::__cxx11::list<Employee *,std::allocator<Employee *>>::begin(a1);// 遍历每一个容器内的对象
        ;
        std::_List_iterator<Employee *>::operator++(&i, 0i64) )
  {
    v5 = std::__cxx11::list<Employee *,std::allocator<Employee *>>::end(v7);
    result = std::_List_iterator<Employee *>::operator!=(&i, &v5);
    if ( !(_BYTE)result )
      break;
    v3 = *(_QWORD *)std::_List_iterator<Employee *>::operator*(&i);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v6, v8);
    Employee::receive(v3, &v6);                 // 调用对象的函数
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v6);
  }
  return result;
}
```

7.备忘录模式

- 分为原发器类和存档类，两者的成员大体相同(看构造函数)

```
__int64 __fastcall MememTo::MememTo(__int64 a1, __int64 a2, int a3)
{
  __int64 result; // rax
  __int64 v4; // [rsp+40h] [rbp-40h]
  __int64 v5; // [rsp+48h] [rbp-38h]
  int v6; // [rsp+50h] [rbp-30h]

  v4 = a1;
  v5 = a2;
  v6 = a3;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(a1);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(v4, v5);
  result = v4;
  *(_DWORD *)(v4 + 32) = v6;
  return result;
}
```

```
__int64 __fastcall Person::Person(__int64 a1, __int64 a2, int a3)
{
  __int64 result; // rax
  __int64 v4; // [rsp+40h] [rbp-40h]
  __int64 v5; // [rsp+48h] [rbp-38h]
  int v6; // [rsp+50h] [rbp-30h]

  v4 = a1;
  v5 = a2;
  v6 = a3;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(a1);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(v4, v5);
  result = v4;
  *(_DWORD *)(v4 + 32) = v6;
  return result;
}
```

- 原发器类含有一个存档函数和读档函数，分别以存档类对象作为返回值和函数参数。且调用的都是原发器类中的函数方法，存档类对象一般只作为参数和返回值

```
void main2(void)
{
  Person *v0; // rbx
  MememTo *v1; // rsi
  Manager *v2; // rbx
  __int64 v3; // rax
  MememTo *v4; // rbx
  Person *v5; // rbx
  char v6; // [rsp+20h] [rbp-70h]
  char v7; // [rsp+4Fh] [rbp-41h]
  char v8; // [rsp+50h] [rbp-40h]
  char v9; // [rsp+77h] [rbp-19h]
  Manager *v10; // [rsp+78h] [rbp-18h]
  MememTo *v11; // [rsp+80h] [rbp-10h]
  Person *v12; // [rsp+88h] [rbp-8h]

  std::allocator<char>::allocator(&v7);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v6, "zhangsan", &v7);
  v0 = (Person *)operator new(0x28ui64);
  Person::Person(v0, &v6, 25i64);
  v12 = v0;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v6);
  std::allocator<char>::~allocator(&v7);
  Person::printT(v12);
  v11 = 0i64;
  v10 = 0i64;
  v1 = (MememTo *)Person::createMememTo(v12);
  v2 = (Manager *)operator new(8ui64);
  Manager::Manager(v2, v1);
  v10 = v2;
  std::allocator<char>::allocator(&v9);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v8, "lisi", &v9);
  Person::setName(v12, &v8);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v8);
  std::allocator<char>::~allocator(&v9);
  Person::printT(v12);
  v3 = Manager::getMememTo(v10);
  text_63(v12, v3);
  Person::printT(v12);
  v4 = v11;
```

IDA View-A | Pseudocode-A | Strings window | Hex View-1 | Structures | Enums

```
__int64 __fastcall setMememTo(__int64 a1, MememTo *a2)
{
  int v2; // edx
  __int64 result; // rax
  char v4; // [rsp+20h] [rbp-20h]
  __int64 v5; // [rsp+50h] [rbp+10h]
  MememTo *v6; // [rsp+58h] [rbp+18h]

  v5 = a1;
  v6 = a2;
  ZN7MememTo7getNameB5cxx11Ev(&v4, a2);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(v5, &v4);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v4);
  v2 = MememTo::getAge(v6);
  result = v5;
  *(_DWORD *)(v5 + 32) = v2;
  return result;
}
```

- main函数多次调用存档-读档函数，且用对象自身上一个状态来调用存档函数，保存到一个存档类对象中，然后修改属性变动到下一个状态。若要读档，则把该存档类对象作为读档函数的参数。

8.访问者模式

- 包含访问者和被访问元素两个主要组成部分，被访问的元素通常具有不同的类型(或不同的子类)

```
public ParkA :
  public /* offset 0x0 */ ParkElement
               public _ZTI5ParkA
 `typeinfo for'ParkA
ZTI5ParkA      dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
                                       ; DATA XREF: .rdata:00000000004066F8↓o
                                       ; reference to RTTI's type class
               dq offset _ZTS5ParkA    ; reference to type's name
               dq offset _ZTI11ParkElement ; reference to parent's type name
               align 10h
public ParkB :
  public /* offset 0x0 */ ParkElement
               public _ZTI5ParkB
 `typeinfo for'ParkB
ZTI5ParkB      dq offset __imp__ZTVN10__cxxabiv120__si_class_type_infoE+10h
                                       ; DATA XREF: .rdata:0000000000406718↓o
                                       ; reference to RTTI's type class
               dq offset _ZTS5ParkB    ; reference to type's name
               dq offset _ZTI11ParkElement ; reference to parent's type name
               align 10h
```

- 一般接收的函数参数是抽象类指针，然后根据不同的访问者可以对它们进行不同的访问操作。

```
__int64 __fastcall ParkA::accept(ParkA *this, Visitor *a2)
{
  char v2; // si
  _BOOL1 v3; // di
  __int64 v4; // rax
  __int64 v6; // [rsp+0h] [rbp-80h]
  char v7; // [rsp+20h] [rbp-60h]
  char v8; // [rsp+40h] [rbp-40h]
  ParkA *v9; // [rsp+90h] [rbp+10h]
  Visitor *v10; // [rsp+98h] [rbp+18h]

  v9 = this;
  v10 = a2;
  v2 = 0;
  (*(void (__fastcall **)(__int64 *, Visitor *))(*(_QWORD *)a2 + 8i64))(&v6 + 4, a2);
  v3 = 1;
  if ( !(unsigned __int8)std::operator==<char,std::char_traits<char>,std::allocator<char>>(&v7, "A清洁工") )
  {
    (*(void (__fastcall **)(char *, Visitor *))(*(_QWORD *)v10 + 8i64))(&v8, v10);
    v2 = 1;
    if ( !(unsigned __int8)std::operator==<char,std::char_traits<char>,std::allocator<char>>(&v8, "管理者") )
      v3 = 0;
  }
  if ( v2 )
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v8);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v7);
  if ( v3 )
  {
    (**(void (__fastcall ***)(Visitor *, ParkA *))v10)(v10, v9);
    v4 = std::operator<<<char,std::char_traits<char>,std::allocator<char>>(refptr__ZSt4cout, (char *)v9 + 8);
  }
  else
  {
    v4 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "无权访问");
  }
  return std::ostream::operator<<(v4, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
}
```

9.状态模式

- 分为对象类和状态类，对象类中含有一个状态类对象，表示对象当前状态，且有设置状态和得到状态等函数方法

```
1  Worker *__fastcall Worker::Worker(Worker *this)
2  {
3    State1 *v1; // rbx
4    Worker *result; // rax
5    Worker *v3; // [rsp+40h] [rbp-40h]
6
7    v3 = this;
8    v1 = (State1 *)operator new(8ui64);
9    State1::State1(v1);
10   result = v3;
11   *((_QWORD *)v3 + 1) = v1;
12   return result;
13 }
```

```
1  __int64 __fastcall Worker::getState(Worker *this)
2  {
3    return *((_QWORD *)this + 1);                 // this+1内存空间存储的是状态类对象
4  }
```

```
1  __int64 __fastcall Worker::setState(__int64 a1, __int64 a2)
2  {
3    __int64 result; // rax
4
5    result = a1;
6    *(_QWORD *)(a1 + 8) = a2;                     // a2是状态类对象，a1是this指针
7    return result;
8  }
```

- 在状态类里实现对象类具体要做的功能，功能函数以对象类对象作为函数参数。状态类的不同子类表示不同的状态

```
__int64 __fastcall State1::doSomeThing(State1 *this, Worker *a2)
{
  _BOOL1 v2; // al
  __int64 v3; // rax
  __int64 result; // rax
  void *v5; // rax
  State2 *v6; // rbx
  __int64 (__fastcall ***v7)(_QWORD, Worker *); // rax
  Worker *v8; // [rsp+48h] [rbp-38h]

  v8 = a2;
  v2 = (unsigned int)Worker::getHour(a2) == 7 || (unsigned int)Worker::getHour(v8) == 8;
  if ( v2 )
  {
    v3 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "吃早餐");
    result = std::ostream::operator<<(v3, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
  }
  else
  {
    v5 = (void *)Worker::getState(v8);
    operator delete(v5);
    v6 = (State2 *)operator new(8ui64);
    State2::State2(v6);
    Worker::setState((__int64)v8, (__int64)v6);
    v7 = (__int64 (__fastcall ***)(_QWORD, Worker *))Worker::getState(v8);
    result = (**v7)(v7, v8);
  }
  return result;
}
```

- 功能函数会判断对象类对象的当前状态是否应该是该子类状态，如果不是就设置对象类对象的状态为下一个状态，并再次调用下一个状态的功能函数来判断。子类的功能函数高度相似，只是状态之间是环形结构

```
  }
  else
  {
    v5 = (void *)Worker::getState(v8);
    operator delete(v5);
    v6 = (State2 *)operator new(8ui64);
    State2::State2(v6);
    Worker::setState((__int64)v8, (__int64)v6);
    v7 = (__int64 (__fastcall ***)(_QWORD, Worker *))Worker::getState(v8);
    result = (**v7)(v7, v8);
  }
  return result;
}
```

```
__int64 __fastcall State2::doSomeThing(State2 *this, Worker *a2)
{
  _BOOL1 v2; // al
  __int64 v3; // rax
  __int64 result; // rax
  void *v5; // rax
  State3 *v6; // rbx
  __int64 (__fastcall ***v7)(_QWORD, Worker *); // rax
  Worker *v8; // [rsp+48h] [rbp-38h]

  v8 = a2;
  v2 = (unsigned int)Worker::getHour(a2) == 9 || (unsigned int)Worker::getHour(v8) == 10;
  if ( v2 )
  {
    v3 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "工作");
    result = std::ostream::operator<<(v3, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
  }
  else
  {
    v5 = (void *)Worker::getState(v8);
    operator delete(v5);
    v6 = (State3 *)operator new(8ui64);
    State3::State3(v6);
    Worker::setState((__int64)v8, (__int64)v6);
    v7 = (__int64 (__fastcall ***)(_QWORD, Worker *))Worker::getState(v8);
    result = (**v7)(v7, v8);
  }
  return result;
}
```

```
__int64 __fastcall State3::doSomeThing(State3 *this, Worker *a2)
{
  _BOOL1 v2; // al
  __int64 v3; // rax
  __int64 result; // rax
  void *v5; // rax
  State1 *v6; // rbx
  __int64 (__fastcall ***v7)(_QWORD, Worker *); // rax
  Worker *v8; // [rsp+48h] [rbp-38h]

  v8 = a2;
  v2 = (unsigned int)Worker::getHour(a2) == 11 || (unsigned int)Worker::getHour(v8) == 12;
  if ( v2 )
  {
    v3 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "午睡");
    result = std::ostream::operator<<(v3, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
  }
  else
  {
    v5 = (void *)Worker::getState(v8);
    operator delete(v5);
    v6 = (State1 *)operator new(8ui64);
    State1::State1(v6);
    Worker::setState((__int64)v8, (__int64)v6);
    v7 = (__int64 (__fastcall ***)(_QWORD, Worker *))Worker::getState(v8);
    result = (**v7)(v7, v8);
  }
  return result;
}
```

10.解释器模式

- 分为语言类和解释器类，解释器类的子类表示不同的解释方法。
- 解释器类的解释函数以语言类对象作为函数参数

```
__int64 __fastcall SubExpreesion::interpreter(SubExpreesion *this, Context *a2)
{
  int v2; // ST2C_4
  Context *v4; // [rsp+48h] [rbp+18h]

  v4 = a2;
  v2 = (unsigned __int64)Context::getNum(a2) - 1;
  Context::setNum(v4, v2);
  return Context::setRes(v4, v2);
}
```

11.迭代器模式

- 迭代器可理解为指针，迭代器类里存有数组类的对象指针和一个游标表示迭代器当前所指向的位置

```
ConcreIterator *__fastcall ConcreIterator::ConcreIterator(ConcreIterator *this, Aggregate *a2)
{
  ConcreIterator *result; // rax
  ConcreIterator *v3; // [rsp+30h] [rbp+10h]
  Aggregate *v4; // [rsp+38h] [rbp+18h]

  v3 = this;
  v4 = a2;
  text_52(this);
  *(_QWORD *)v3 = off_4055C0;
  *((_DWORD *)v3 + 2) = 0;                      // 游标 默认构造时归零
  result = v3;
  *((_QWORD *)v3 + 2) = v4;
  return result;
}
```

- 数组类类似数据库，函数方法只实现存储和返回数据等功能

```
; `vtable for'ConcreAggregate
_ZTV15ConcreAggregate dq 0                      ; offset to this
                      dq offset _ZTI15ConcreAggregate ; `typeinfo for'ConcreAggregate
off_4055F0            dq offset _ZN15ConcreAggregate10CreateIterEv
                                                ; DATA XREF: ConcreAggregate::ConcreAggregate(void)+18↑o
                                                ; ConcreAggregate::CreateIter(void)
                      dq offset _ZN15ConcreAggregate7getItemEi ; ConcreAggregate::getItem(int)
                      dq offset _ZN15ConcreAggregate7getSizeEv ; ConcreAggregate::getSize(void)
                      align 10h
                      public _ZTV9Aggregate
```

- 迭代器类有Next函数表示游标的移动

```
ConcreIterator *__fastcall ConcreIterator::Next(ConcreIterator *this)
{
  ConcreIterator *result; // rax
  ConcreIterator *v2; // [rsp+30h] [rbp+10h]

  v2 = this;
  result = (ConcreIterator *)((*(unsigned int (__fastcall **)(ConcreIterator *))(*(_QWORD *)this + 16i64))(this) ^ 1);
  if ( (_BYTE)result )
  {
    result = v2;
    ++*((_DWORD *)v2 + 2);
  }
  return result;
}
```

- 创建迭代器的函数方法在数组类里

IDA View-A | Pseudocode-A | Strings window | Hex View-1 | Structures

```
ConcreIterator *__fastcall ConcreAggregate::CreateIter(ConcreAggregate *this)
{
  ConcreIterator *v1; // rbx
  ConcreAggregate *v3; // [rsp+40h] [rbp-40h]

  v3 = this;
  v1 = (ConcreIterator *)operator new(0x18ui64);
  ConcreIterator::ConcreIterator(v1, v3);
  return v1;
}
```