类中的静态成员真是个让人爱恨交加的特性。我决定好好总结一下静态类成员的知识点，以便自己在以后面试中，在此类问题上不在被动。

静态类成员包括静态数据成员和静态函数成员两部分。

一 静态数据成员：

类体中的数据成员的声明前加上static关键字，该数据成员就成为了该类的静态数据成员。和其他数据成员一样，静态数据成员也遵守public/protected/private访问规则。同时，静态数据成员还具有以下特点：

1.静态数据成员的定义。

静态数据成员实际上是类域中的全局变量。所以，静态数据成员的定义(初始化)不应该被放在头文件中。

其定义方式与全局变量相同。举例如下：

xxx.h文件

class base{

private:

static const int \_i;//声明，标准c++支持有序类型在类体中初始化,但vc6不支持。

};

xxx.cpp文件

const int base::\_i=10;//定义(初始化)时不受private和protected访问限制.

注：不要试图在头文件中定义(初始化)静态数据成员。在大多数的情况下，这样做会引起重复定义这样的错误。即使加上#ifndef #define #endif或者#pragma once也不行。

2.静态数据成员被 类 的所有对象所共享，包括该类派生类的对象。即派生类对象与基类对象共享基类的静态数据成员。举例如下：

class base{

public :

static int \_num;//声明

};

int base::\_num=0;//静态数据成员的真正定义

class derived:public base{

};

main()

{

base a;

derived b;

a.\_num++;

cout<<"base class static data number \_num is"<<a.\_num<<endl;

b.\_num++;

cout<<"derived class static data number \_num is"<<b.\_num<<endl;

}

// 结果为1,2;可见派生类与基类共用一个静态数据成员。

3.静态数据成员可以成为成员函数的可选参数，而普通数据成员则不可以。举例如下：

class base{

public :

static int \_staticVar;

int \_var;

void foo1(int i=\_staticVar);//正确,\_staticVar为静态数据成员

void foo2(int i=\_var);//错误,\_var为普通数据成员

};

4.★静态数据成员的类型可以是所属类的类型，而普通数据成员则不可以。普通数据成员的只能声明为 所属类类型的 指针或引用。举例如下：

class base{

public :

static base \_object1;//正确，静态数据成员

base \_object2;//错误

base \*pObject;//正确，指针

base &mObject;//正确，引用

};

5.★这个特性，我不知道是属于标准c++中的特性，还是vc6自己的特性。

静态数据成员的值在const成员函数中可以被合法的改变。举例如下：

class base{

public:

base(){\_i=0;\_val=0;}

mutable int \_i;

static int \_staticVal;

int \_val;

void test() const{//const 成员函数

\_i++;//正确，mutable数据成员

\_staticVal++;//正确，static数据成员

\_val++;//错误

}

};

int base::\_staticVal=0;

二，静态成员函数

静态成员函数没有什么太多好讲的。

1.静态成员函数的地址可用普通函数指针储存，而普通成员函数地址需要用 类成员函数指针来储存。举例如下：

class base{

static int func1();

int func2();

};

int (\*pf1)()=&base::func1;//普通的函数指针

int (base::\*pf2)()=&base::func2;//成员函数指针

2.静态成员函数不可以调用类的非静态成员。因为静态成员函数不含this指针。

3.静态成员函数不可以同时声明为 virtual、const、volatile函数。举例如下：

class base{

virtual static void func1();//错误

static void func2() const;//错误

static void func3() volatile;//错误

};

最后要说的一点是，静态成员是可以独立访问的，也就是说，无须创建任何对象实例就可以访问。