**概述：**

**1. 为什么要使用友元？**

通常对于普通函数来说，要访问类的保护成员是不可能的，如果想这么做那么必须把类的成员都生命成为 public( 共用的) ，然而这做带来的问题遍是任何外部函数都可以毫无约束的访问它操作它；另一种方法是利用 C++ 的 friend 修饰符，可以让一些你设定的函数能够对这些私有或保护数据进行操作。

**2. 使用友元有哪些缺点？**

使用友元的同时也破坏了类的封装特性，这即是友元最大的缺点。当对外声明为友元后，你的所有细节全部都暴露给了对方。

就好像你告诉你朋友你很有钱这个密秘，进而又把你有多少钱，多少古董，多少家产，多少小妾等等所有的家底全给他说了。

**3. 友元怎样理解？**

定一个友元函数，或是友元类，就是告诉对方：我的所有元素对你是开放的。这种 friend 是建立在灰常灰常灰常信任对方的基础上的。

**一. 普通函数做为类的一个友元函数**

在类里声明一个普通函数，在前面加上 friend 修饰，那么这个函数就成了该类的友元。

这时这个普通函数可以访问该类的一切成员。

**代码如下：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7591415)

1. #include <iostream >
2. using namespace std;
3. class MyClass
4. {
5. public :
6. MyClass(string name)
7. {
8. m\_name = name;
9. }
10. //声明一个友元函数
11. friend void Display(MyClass &mycalss);
12. Protected:
13. string m\_name;
14. };
15. //定义这个友元函数
16. //写成 void MyClass::Display(MyClass &mycalss)
17. void Display(MyClass &mycalss)
18. {
19. cout << "Access Protected data : "<< mycalss.m\_name << endl;
20. }
21. //测试
22. **int** main(**int** argc,**char**\* argv[])
23. {
24. MyClass test("Class A");
25. Display(test);
26. return 0;
27. }

#include <iostream >

using namespace std;

class MyClass

{

public :

MyClass(string name)

{

m\_name = name;

}

//声明一个友元函数

friend void Display(MyClass &mycalss);

Protected:

string m\_name;

};

//定义这个友元函数

//写成 void MyClass::Display(MyClass &mycalss)

void Display(MyClass &mycalss)

{

cout << "Access Protected data : "<< mycalss.m\_name << endl;

}

//测试

int main(int argc,char\* argv[])

{

MyClass test("Class A");

Display(test);

return 0;

}

**说明：**

1. 声明这个友元函数可以在任何地方，可以在 public、protected 当然也可以在 privated 里。

2. 在这个友元函数里，你可以访问这个类里的所有的成员，所有的成员函数，而不管它是不是 public、protected 或 privated 的。

3. 定义友元函数时，不能写成 void MyClass::Display(MyClass &mycalss) 这点要注意。

**二. 一个普通函数可以是多个类的友元函数**

在每个类里面都有一个友元函数的声明，声明可以有多个，但定义只能有一个。  
**代码如下：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7591415)

1. #include <iostream >
2. using namespace std;
3. class MyClass\_B;
4. class MyClass\_A
5. {
6. public:
7. MyClass\_A(string name)
8. {
9. m\_name = name;
10. }
11. //声明一个友元函数
12. friend void Display(MyClass\_A &myA, MyClass\_B &myB);
13. private:
14. string m\_name;
15. };
16. class MyClass\_B
17. {
18. public:
19. MyClass\_B(string name)
20. {
21. m\_name = name;
22. }
23. //注意，又声明一个友元函数
24. friend void Display(MyClass\_A &myA, MyClass\_B &myB);
25. private:
26. string m\_name;
27. };
28. //定义这个友元函数
29. void Display(MyClass\_A &myA, MyClass\_B &myB)
30. {
31. cout << "MyClass A : "<< myA.m\_name << endl;
32. cout << "MyClass B : "<< myB.m\_name << endl;
33. }
34. //测试代码
35. **int** main(**int** argc,**char**\* argv[])
36. {
37. MyClass\_A testA("Class A");
38. MyClass\_B testB("Class A");
39. Display(testA, testB);
40. return 0;
41. }

#include <iostream >

using namespace std;

class MyClass\_B;

class MyClass\_A

{

public:

MyClass\_A(string name)

{

m\_name = name;

}

//声明一个友元函数

friend void Display(MyClass\_A &myA, MyClass\_B &myB);

private:

string m\_name;

};

class MyClass\_B

{

public:

MyClass\_B(string name)

{

m\_name = name;

}

//注意，又声明一个友元函数

friend void Display(MyClass\_A &myA, MyClass\_B &myB);

private:

string m\_name;

};

//定义这个友元函数

void Display(MyClass\_A &myA, MyClass\_B &myB)

{

cout << "MyClass A : "<< myA.m\_name << endl;

cout << "MyClass B : "<< myB.m\_name << endl;

}

//测试代码

int main(int argc,char\* argv[])

{

MyClass\_A testA("Class A");

MyClass\_B testB("Class A");

Display(testA, testB);

return 0;

}

同样的，这个友元函数，可以访问这两个类的所有元素。

**三. 一个类的成员函数也可以是另一个类的友元**

从而可以使得一个类的成员函数可以操作另一个类的数据成员

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7591415)

1. #include <iostream >
2. using namespace std;
3. class MyClass\_B;
4. //A 类
5. class MyClass\_A
6. {
7. public:
8. MyClass\_A(string name)
9. {
10. m\_name = name;
11. }
12. void Function(MyClass\_B &myB);
13. private:
14. string m\_name;
15. };
16. //B 类
17. class MyClass\_B
18. {
19. public:
20. MyClass\_B(string name)
21. {
22. m\_name = name;
23. }
24. //友元函数声明，注意和普通函数的区别
25. friend void MyClass\_A::Function(MyClass\_B &myB);
26. private:
27. string m\_name;
28. };
29. //函数定义
30. void MyClass\_A::Function(MyClass\_B &myB)
31. {
32. cout<<myB.m\_name<<endl;
33. }
34. //测试代码
35. **int** main(**int** argc,**char**\* argv[])
36. {
37. MyClass\_A testA("Class A");
38. MyClass\_B testB("Class B");
39. testA.Function(testB);
40. return 0;
41. }

#include <iostream >

using namespace std;

class MyClass\_B;

//A 类

class MyClass\_A

{

public:

MyClass\_A(string name)

{

m\_name = name;

}

void Function(MyClass\_B &myB);

private:

string m\_name;

};

//B 类

class MyClass\_B

{

public:

MyClass\_B(string name)

{

m\_name = name;

}

//友元函数声明，注意和普通函数的区别

friend void MyClass\_A::Function(MyClass\_B &myB);

private:

string m\_name;

};

//函数定义

void MyClass\_A::Function(MyClass\_B &myB)

{

cout<<myB.m\_name<<endl;

}

//测试代码

int main(int argc,char\* argv[])

{

MyClass\_A testA("Class A");

MyClass\_B testB("Class B");

testA.Function(testB);

return 0;

}

我们可以看到，B 类，对 A 类其中的一个函数开放，其结果是这个函数可以访问 B 类的所有元素。

**四. 整个类也可以是另一个类的友元**

友类的每个成员函数都可以访问另一个类的所有成员。

**示例代码如下：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7591415)

1. #include <iostream >
2. using namespace std;
3. //类 A
4. class MyClass\_B;
5. class MyClass\_A
6. {
7. public:
8. MyClass\_A(string name)
9. {
10. m\_name = name;
11. }
12. //友元类声明
13. friend class MyClass\_B;
14. private:
15. string m\_name;
16. };
17. //类 B
18. class MyClass\_B
19. {
20. public:
21. MyClass\_B(string name)
22. {
23. m\_name = name;
24. }
25. void Display(MyClass\_A &myA);
26. private:
27. string m\_name;
28. };
29. //成员函数
30. void MyClass\_B::Display(MyClass\_A &myA)
31. {
32. cout<<myA.m\_name<<endl; //访问A的私有成员
33. MyClass\_A test("test");
34. cout<<test.m\_name<<endl; //好像A的所有元素在B里都存在一样
35. }
36. //测试代码
37. **int** main(**int** argc,**char**\* argv[])
38. {
39. MyClass\_A testA("Class A");
40. MyClass\_B testB("Class B");
41. testB.Display(testA);
42. return 0;
43. }

#include <iostream >

using namespace std;

//类 A

class MyClass\_B;

class MyClass\_A

{

public:

MyClass\_A(string name)

{

m\_name = name;

}

//友元类声明

friend class MyClass\_B;

private:

string m\_name;

};

//类 B

class MyClass\_B

{

public:

MyClass\_B(string name)

{

m\_name = name;

}

void Display(MyClass\_A &myA);

private:

string m\_name;

};

//成员函数

void MyClass\_B::Display(MyClass\_A &myA)

{

cout<<myA.m\_name<<endl; //访问A的私有成员

MyClass\_A test("test");

cout<<test.m\_name<<endl; //好像A的所有元素在B里都存在一样

}

//测试代码

int main(int argc,char\* argv[])

{

MyClass\_A testA("Class A");

MyClass\_B testB("Class B");

testB.Display(testA);

return 0;

}

此时B可以访问A的所有元素，就好像A在B里面一样。

**五. 总结**

简单的说就是：**声明一个友元函数或者是友元类，就是要把自己完全暴露给对方**。