友元在数据库管理中的定位

王卫东,屈 洋
(暨南大学,广东 广州 510632)

摘 要:讨论在设计数据库管理系统时,如何利用C++软件的友元类的设计方法实现以授权方式进行数据库的安全管理和数据表的操作管理。利用C++软件的友元类程序设计思想的讨论,即友元的基本构想、友元的设计方法、友元类的使用规则等,阐述如何实现利用授权的手段进行数据库管理的方法。通过恰当的程序分析得出正确的友元类程序设计方法和基本规则。利用友元设计完全可以实现数据库的授权管理,但要牺牲类的封装性。

关键词:类;友元;数据库管理
中图分类号:TP311.13 **文献标识码**:A **文章编号**:1673-629X(2008)09-0016-03

Friend Function Orientation in Database Management System

WANG Wei-dong, QU Yang
(Jinan University, Guangzhou 510632, China)

Abstract: Discuss that designing the database management, how to realize authorization management by friend class programming in database security management. Utilizing friend function method in C++, that is, friend function basic idea and design method and friend class use rule, profit authorization management. Dependent on appropriate program analysis, gives right friend function design method and basic rule. Using the friend function design, completely achieve database management, but must give up encapsulation character of class.

Key words: class; friend function; database management

0 引 言

C++程序设计语言是目前广泛应用的计算机语种之一,然而在针对C++面向对象的程序设计进行深入探讨时,常常面对许多的疑惑,譬如:基类、继承与派生类、多态性与虚函数等等,友元的应用就是其中之一,如何使用友元?友元在数据库管理中如何定位?文中将通过实例给予解释。

1 C++中关于友元的概念

友元理念[1~3]的引入起源于类的设计特征,我们都知道类的基本特征就是它的封装性,故要想利用类的数据成员实现数据的处理,必须为类设计数据输入和输出接口,即为类设计数据成员函数,这样才能通过成员函数实现数据的输入,才能看到数据处理的结果,如此才能体现出类的封装性。但是这一特性在数据库管理过程中,当遇到修改数据时就会感到比较麻烦,于是想到可否使用一个外部函数,它可以帮助我们直接对指定类的数据成员内容进行修改,进而推之,可否使用其他的类,它可以帮助我们直接对指定类的数据成员内容进行修改,友元的概念由此产生。

友元:通过在指定类声明中规定某个外部函数或其他的类为其友,则这个外部函数或其他的类就可对本类的数据成员直接进行数据的赋值、修改、引用。

友元类的声明方式,为了说明声明步骤,预先假设B类为A类的友元类。

第一步:在声明B类前先对A类做一个预声明,以备B类做引用A类的数据成员的友元应用。其例如下:

```
CLASS A;//前向引用声明A,目的是为B使用友元类而定义
CLASS B
{
    //内中声明数据成员和成员函数
};
```

第二步:为B能使用A而声明对象。凡用过C++者皆知使用类必先声明对象,因此B类中准备进入A类中使用其数据成员的成员函数必预先作对象的声明。声明方法如下:

收稿日期:2007-12-16
基金项目:留学回国人员科研启动基金(教外司留[1999]363号(教育部))
作者简介:王卫东(1956-),男,山西人,教授,从事计算机的教学与软件开发应用;屈 洋,教授,从事临床医学和计算机在医学方面的应用研究。

```
CLASS A;//前向引用声明 A,目的是为 B 使用友元类而定义
CLASS B
{ PRIVATE:
      //内中声明数据成员
  PUBLIC:
      //内中声明成员函数原型
      Void Friend_Function1 (A & Object_Name);
};
```

由于在 B 类中作为友元成员函数在形参里要声明使用类 A 的对象,由此可见向前引用声明类 A 的重要性,由于是原型声明故 & 的后面不加入对象的具体名称。

第三步:在 A 类中声明 B 是友元类,此声明在 PUBLIC 区内进行。声明如下:

```
CLASS A
{ PRIVATE:
      //内中声明数据成员
PUBLIC:
      //内中声明成员函数原型
  Friend Class B;//友元类的声明
}
```

自此 B 类做为 A 类的友元类可以进入 A 类使用它的数据成员了,要注意此时通常意义上的类的封装性已经被破坏殆尽,友元在数据库管理的地位问题也由此浮出。

2　数据库管理中的权限问题

在数据库管理过程中经常遇见的问题是数据的新建、更新与删除,那么不可避免的问题是谁可以实现这些操作,进而分析,数据库中的数据从安全性或保密性角度必至少分成普通数据、中层数据和高层数据,显然它们对应不同的管理层次。于是我们看到的管理模式是一个由高到低的不可逆关系,理论上安全性是权限即授权的范围,保密性即密钥,此二者或合二为一,或并行执行,显然后者的管理措施更加严密,因为有了权限没有密钥依然不能对指定层次的数据进行任何的操作,反之亦然。

利用 C++ 如何实现如上之管理模式? 显然利用 C++ 的友元函数的特点可以较轻易地实现数据库管理中的权限问题,因为友元函数的一个基本特征就是它的不可逆性,试想有 A、B、C 三个类,B 为 A 之友,C 为 A、B 的友,则 B 可访问 A,C 可访问 A 和 B,但此过程不可逆,试想将如此的过程用数据库管理的观点解释,则不难看出 C++ 的友元函数在数据库管理中的定位本质上不是友的观念而是授权[Authorization][4~6]。

3　具体事例

有学生管理数据库,其中教师可以直接向学生数据库录入成绩,也可以随时修改,但学生只有察看的权利,本程序由于篇幅的原因将权限的审查略去,仅将相关的友元的设计提供出来供大家参考。程序实际应用中分成三部分存于相应文件中:其一,主函数部分;其二,类的声明部分;其三,数据成员函数程序部分[7]。

1)主函数文件(project1.cpp):

```
#include<iostream.h>//此预处理命令
#include<iomanip.h>
#include<math.h>
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<string>
using namespace std;
#include"e:\project1\hfunction.h"//指定类头文件的路径
void main(){
  f_teacher wang(23,"照明");//教师输入本人的工作证号码和姓名
  wang.t_show();//此时检验合格则允许进行下一步工作(仅保留显示)
  f_student stu1(10,"无用",87.3); //输入学生成绩
  stu1.s_show();//显示该学生的成绩
  wang.assign_grades(stu1);//具备权限的教师修改学生成绩
  stu1.s_show();
  f_student stu2(12,"理由",67.3);
  stu2.s_show();
  wang.assign_grades(stu2);
  stu2.s_show();
}
```

2)类的声明文件(hfunction.h):

```
class f_student;//前向引用声明,目的是为 f_teacher 定义友元函数
class f_teacher//教师的类声明
{
private:
    char ft_name[20];
    int ft_id;
public:
    f_teacher(int,char[]);//f_teacher 的构造函数
    void t_show();//显示用成员函数
    void assign_grades(f_student &);//f_student 的友元函数
    ~f_teacher(){cout<<"this teacher class over!"<<endl;}
      //f_teacher 的析构函数
};
class f_student//学生类的声明
{
private:
```

```
    int fs_id;
    char fs_name[20];
    float fs_grades;
public:
    f_student(int,char[],float);//学生成绩输入的成员函数
    void s_show();//成绩显示成员函数
    friend class f_teacher;//声明类 f_teacher 是类 f_student 友元类
    ~f_student(){cout<<"this student class over!"<<endl;}//析构函数
};
```

3)类的成员函数文件(functionp.cpp):

```
#include<iostream.h>//预处理命令
#include<iomanip.h>
#include<math.h>
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<string>
using namespace std;
#include"e:\project1\hfunction.h"//类的头文件
f_teacher::f_teacher(int ftid,char ftna[]) //f_teacher 的构造函数
{
  ft_id=ftid;
  strcpy(ft_name,ftna);
}
void f_teacher::t_show()
{cout<<"教师工作证号码:"<<ft_id<<"教师姓名:"<<ft_name<<endl;}
void f_teacher::assign_grades(f_student &ts)//f_student 的友元函数
{float n;
  cout<<"显示该学生的原成绩:"<<ts.fs_grades<<endl;
  cout<<"输入修改的成绩:";
  cin>>n;
  ts.fs_grades=n;//教师修改学生成绩
}
f_student::f_student(int fsid,char fsna[],float fsgr)//f_student 的输入成员函数
{
  fs_id=fsid;
  strcpy(fs_name,fsna);
  fs_grades=fsgr;
}
void f_student::s_show()//f_student 的显示成员函数
{
    cout<<"学生学号:"<<fs_id<<endl;
    cout<<"学生姓名:"<<fs_name<<endl;
    cout<<"学生成绩:"<<fs_grades<<endl;
}
```

4 结束语

通过上述分析和程序实例,可以清晰地看到教师类(f_teacher)作为学生类(f_student)的友元类可以对学生的成绩随时加以修改,然而学生只能够浏览个人成绩,于是实现了我们的预期要求即利用友元实现数据库的授权管理模式。

参考文献:

[1] Stevens A. Wiley's Teach C++[M]. 北京:电子工业出版社,2004.

[2] Pohl I. Object-Oriented Programming Using C++[M]. 2nd Edition. [s.l.]:Addison-Wesley Pub Co,2004.

[3] Stroustrup B. The C++ Programming Language[M]. 3rd Edition. [s.l.]:Addison-Wesley Pub Co,1997.

[4] Rabin M O. Efficient dispersal of information for security, load balancing, and fault tolerance[J]. Journal of the ACM,1989,36(2):335-348.

[5] Dongarra J. The Top 10 Algorithms[J]. IEEE Computing in Science & Engineering,2000,2(1):22-23.

[6] 张志勇. 委托授权在PMI体系架构中的研究与应用[J]. 计算机工程,2006,32(5):152-154.

[7] 聂 青. 现代程序设计C++数据结构面向对象的方法与实现[M]. 北京:北京理工大学出版社,2002.

(上接第15页)

电子学报,2000,28(4):102-106.

[2] 刘明宝,姚鸿勋,高 文. 彩色图像的实时人脸跟踪方法[J]. 计算机学报,1998,21:527-531.

[3] Hager G D, Belhumeur P N. Efficient region tracking with parametric models of geometry and Illumination[J]. IEEE Trans. PAMI,1998,20:1025-1039.

[4] Antoszczyszyn P M, Hannah J M, Grant P M. Tracking of the motion of important facial features in model-based coding[J]. Signal Processing,1998,66:249-260.

[5] Bradski G R. Computer vision face tracking as a component of a perceptual user interface[C]//Proceedings of IEEE Workshop Applications of Computer Vision. Princeton, NJ: IEEE, 1998:214-219.

[6] Viola P, Jones M J. Rapid Object Detection Using a Boosted Cascade of Simple Features[J]. Computer Vision and Pattern Recognition,2001,1:8-14.

[7] 陆其明. DirectShow开发指南[M]. 北京:清华大学出版社,2003.

[8] 刘伟玮. Visual C++视频/音频开发实用工程案例精选[M]. 北京:人民邮电出版社,2004:311-312.