《面向对象程序设计》朋辈课程 · 第八辑

友元与静态成员

信息学院 赵家宇

PART 01

友元

友元

- 友元：类外的一般函数或是另一个类的成员函数能够对该类体中的私有成员和其他成员能直接访问；友元以牺牲信息隐藏原则，削弱封装性来提高编程效率，增大了类与类之间的藕合度；
- 友元的种类：
  - 一般函数
  - 另一个类的成员函数
  - 另一个类

友元

- 友元的关系是单向的，友元的关系不能传递；
- 友元的利弊分析：
  - 弊：友元可以访问其他类中的私有成员，是对封装性原则的破坏；
  - 利：实现数据共享，提高程序的效率；

友元函数

- 在本类以外的其他地方定义了一个函数（可以是不属于任何类的一般函数，也可以是其他类的成员函数），在对本类进行声明时在类体中用friend对该函数进行声明，此函数就成为本类的友元函数.
- 友元函数说明格式：

```
friend <类型> <函数名>(<参数列表>)
```

- 特性：可以访问与其有好友关系的类中的私有成员；

友元函数

- 根据函数的定义位置：
  - 普通友元函数：将普通函数声明为友元函数；
  - 友元成员函数：友元函数可以是另一个类中的成员函数；
- 一个函数（普通函数或成员函数）可以被多个类声明为友元，因此可以引用多个类中的私有数据。

友元成员函数

```
class Time {
    public:
        Time(int,int,int);
        void display(Date &);
    private:
        int hour;
        int minute;
        int sec;
};
```

```
class Date{
    public:
        Date(int,int,int);
        friend void Time::display(Date &);
    private:
        int month;
        int day;
        int year;
};
```

- 声明Time类中的display函数是本类的友元成员函数；display是Date类外的函数，但是能够访问date私有成员。

友元成员函数

```
class Time {
    int hour, minute, sec;
    public:
        Time(int,int,int);
        void display(Date &);
};
```

```
class Date{
    int month, day, year;
    public:
        Date(int,int,int);
        friend void Time::display(Date &);
};
```

```
void Time::display(Date &t) {
    cout<<t.month<<t.day<<t.year;
    cout<<hour<<minute<<sec;
}
int main() {
    Time t1(17,45,50);
    Date d1(5,8,2024);
    t1.display(d1);
    return 0;
}
```

友元类

- 可以将一个类声明为另一个类的友元，即友元类；若B类是A类的友元类，友元类B中所有函数都是A类的友元函数，可以访问A类中的所有成员；

```
class Screen {
    friend class Window_Mgr;
    …
};
```

- Window_Mgr是Screen的友元类，则Window_Mgr中的所有成员函数都是Screen的友元函数。

PART 02

静态数据成员

回顾：已经接触过的变量修饰符

- auto
- static ：

  只能初始化一次；值可保留和刷新
- register
- extern

```
#include <stdio.h>
void f(){
  auto int a=3;
  static int b=3;
  a=a+1;
  b=b+1;
  printf("%d %d ",a,b);
}

int main( ) {
   for (int i=0;i<3;i++)
       f();
 }
```

静态数据成员

- 一个类有若干个对象，每个对象都分别有自己的数据成员，不同对象的数据成员各自有值。但是，有时希望有某一个或几个数据成员为所有对象所共有，以便实现数据共享。
- 数据共享的方式：
  - 使用全局变量（难以辨别，不安全）
  - 使用全局对象（操作麻烦，不安全）
  - 使用静态成员（静态数据成员/静态成员函数）

静态数据成员

- 静态数据成员是一种特殊的数据成员，它以关键字static开头；
- 语句格式：

```
class 类名 {
    …
    static 类型说明符 成员名;
    …
};
```

```
class Student {
    int studentID;
    char* name;
    static int schoolID;
};
```

静态数据成员

- 静态数据成员为全体对象所共有，不只属于某个对象，所有对象都可以引用它；在定义对象之前，就有静态数据成员。
- 静态数据成员保证每个类只有一个实体，并不为每个对象都保留了空间；静态数据成员在所有对象之外单独开辟空间，只要在类中定义了静态数据成员，即使不定义对象，也为静态数据成员分配空间，可以被引用；
- 对于同一个类的所有对象而言，静态数据成员的值是一样的。

静态数据成员

- 由于静态数据成员是类的所有对象共享的，因此必须对静态数据成员进行初始化，且只能在类体外进行，格式如下：

  数据类型　类名::成员名=初值;

- 若未对静态数据成员赋初值，则编译器自动赋予初值。

```
class Student {
    int studentID;
    char* name;
    static int schoolID;
};
int Student::schoolID = 10384;
```

静态数据成员

- 静态数据成员并不是属于某个对象，而是属于类的，但同类的对象可以引用它；

使用静态数据成员具有访问属性限制，主要有以下方式：

- 在类的public部分说明的静态数据成员，可以通过对象名或是类名访问，如对象名.静态数据成员 或 类名::静态数据成员。
- 在类的非public部分说明的静态数据成员，只能由类的成员函数访问，保证了安全性。

静态数据成员和全局变量

- 通过静态数据成员实现数据共享，可以保证按照设计者的要求进行保护和操作，保证了与普通数据成员一致的安全性；因此可以不使用全局变量。全局变量破坏了封装的原则，使得类外操作也会影响到类内数据。
- 静态数据成员的作用域只限定于定义该类的作用域内；全局变量自生成起就会一直持续到程序结束。

静态成员函数

- 在类中声明函数的前面加static就成了静态成员函数;
- 语句格式:

```
class 类名 {
    …
    类型 函数名(形参) {
        函数体
    }
    …
};
```

```
class Student {
    int studentID;
    char* name;
    static int schoolID;
    static int check(){ return schoolID;}
};
Student::check();
```

静态成员函数

- static成员函数是类的一部分，而不是对象的一部分，因此没有this形参；它可以直接访问所属类的static成员，但不能直接使用非static成员；
- 若是在类外定义static成员函数，则无须重复指定static保留字；
- static成员函数不能被声明为const；
- 在类外调用静态成员函数的使用方法：

  类名::静态成员函数　或　对象名.静态成员函数

静态成员函数和静态数据成员

- 引进静态成员函数的目的，不是为了对象之间的沟通（不是访问某对象），而是访问类中的静态数据成员；但对于static的数据成员并非一定要static的成员函数才能访问，使用普通的成员函数也是能访问的；只是使用静态成员函数去访问是更合适的！
- 如果想让静态成员函数也能访问私有数据，则需要向静态成员函数传递一个对象（此举在功能上类似友元）；

感谢倾听

给个好评！