C++类成员指针

C++ 同时被 2 个专栏收录
29 订阅 96 篇文章 订阅专栏

1.成员指针简介

成员指针是C++引入的一种新机制，它的申明方式和使用方式都与一般的指针有所不同。成员指针分为 成员函数 指针和数据成员指针。

2. 成员函数指针

在事件驱动和 多线程 应用中被广泛用于调用回调函数。在多线程应用中，每个线程都通过指向成员函数的指针来调用该函数。在这样的应用中，如果不用成员指针，编程是非常困难的。成员函数指针的定义格式:

```
成员函数返回类型 （类名::*指针名）（形参）= &类名::成员函数名
```

成员指针使用示例:

```
#include <iostream>
#include <string>
using namespace std;

class A
{
    string name;

public:
    A(string s)
    {
        name=s;
    }

    void print()
    {
        cout<<"name:"<<name<<endl;
    }
};

int main()
{
    A a("lvlv");
    void (A::*memP)()=&A::print; //定义类成员函数指针并赋初值
    (a.*memP)();
}
```

程序正常运行并输出:

```
name:lvlv
```

使用成员函数指着注意两点：

（1）使用成员函数指针时需要指明成员函数所属的类对象，因为通过指向成员函数的指针调用该函数时，需要将对象的地址用作this指针的值，以便进行函数调用；

（2）为成员函数指针赋值时，需要显示使用&运算符，不能直接将"类名::成员函数名"赋给成员函数指针。

3. 数据成员指针

一个类对象生成后，它的某个成员变量的地址实际上由两个因素决定：对象的首地址和该成员变量在对象之内的偏移量。数据成员指针是用来保存类的某个数据成员在类对象内的偏移量的。它只能用于类的非静态成员变量。数据成员指针的定义格式:

```
成员类型 类名::*指针名=&类名::成员名;
```

数据成员指针使用示例:

```
#include <iostream>
using namespace std;

class Student
{
public:
    int age;
    int score;
};

double average(Student* objs,int Student::*pm,int count)
{
    int result=0;
    for(int i=0;i<count;++i)
    {
        result+=objs[i].*pm;
    }
    return double(result)/count;
}

int main()
{
    Student my[3]={{16,86},{17,80},{18,58}};
    double ageAver=average(my,&Student::age,3);//求平均年龄
    double scoreAver=average(my,&Student::score,3);//求平均成绩
    cout<<"ageAver:"<<ageAver<<endl;
    cout<<"scoreAver:"<<scoreAver<<endl;
}
```

程序输出如下结果:

```
ageAver:17
scoreAver:74.6667
```

使用数据成员指针时，需要注意以下几点：

（1）数据成员指针作为一个变量，在底层实现上，存放的是对象的数据成员相对于对象首地址的偏移量，因此通过数据成员指针访问成员变量时需要提供对象的首地址，即通过对象来访问。从这个意义上说，数据成员指针并不是一个真正的指针。

（2）对象的数据成员指针可以通过常规指针来模拟，例如上面的程序中，可以讲average()函数的形参pm可以申明为int型变量，表示数据成员的偏移量，那么原来的 `obj.*pm` 等同于 `*(int*)((char*)(&obj)+pm)` ，显然，这样书写可读性差，可移植性低且容易出错。

（3）使用数据成员指针时，被访问的成员往往是类的公有成员，如果是类的私有成员，容易出错。考察如下程序。

```
#include <iostream>
using namespace std;
class ArrayClass
{
    int arr[5];
public:
    ArrayClass()
    {
        for(int i=0;i<5;++i)
            arr[i]=i;
    }
};

//使用数据成员指针作为形参
void printArray(ArrayClass& arrObj,int (ArrayClass::* pm)[5])
{
    for(int i=0;i<5;++i)
    {
        cout<<(arrObj.*pm)[i]<<" ";
    }
}

int main()
{
    ArrayClass arrObj;
    printArray(arrObj,&ArrayClass::arr);//编译出错，提示成员ArrayClass::arr不可访问
}
```

以上程序无法通过编译，因为成员arr在类ArrayClass中的访问权限设置为private，无法访问。要解决这个问题，将函数printArray()设置为类ArrayClass的友元函数是不行的，因为是在调用该函数时访问了类ArrayClass的私有成员，而不是在函数体内用到类ArrayClass的私有成员。因此，可以定义一个调用printArray()函数的友元函数。该函数的参数中并不需要传递类ArrayClass的私有成员。修改后的程序如下。

```
#include <iostream>
using namespace std;

class ArrayClass
{
int arr[5];
public:
    ArrayClass()
    {
        for(int i=0;i<5;++i)
            arr[i]=i;
    }

    friend void print(ArrayClass& arrObj);
};

//使用数据成员指针作为形参
void printArray(ArrayClass& arrObj,int (ArrayClass::* pm)[5])
{
    for(int i=0;i<5;++i)
        cout<<(arrObj.*pm)[i]<<" ";
}

//定义友元函数
void print(ArrayClass& arrObj)
{
    printArray(arrObj,&ArrayClass::arr);
}

int main()
{
    ArrayClass arrObj;
    //printArray(arrObj,&ArrayClass::arr);//编译出错，提示成员ArrayClass::arr不可访问
    print(arrObj); //通过友元函数调用打印数组函数printArray()来访问私有成员
}
```

程序通过编译，运行输出0,1,2,3,4。

一般函数指针和类的成员函数指针 09-08
C++中一般函数指针和类的成员函数指针的区别