虚函数

多态性的实现原理

动态绑定和静态绑定

前面例子中的函数调用，都是采用“先期绑定”的方式。所谓“绑定”就是建立函数调用和函数本体的关联。如果绑定发生于程序运行之前（由编译器和链接器完成），则称为“先期绑定”（也称为“静态绑定”）。要实现多态性，就要进行“后期绑定”（也称为“动态绑定”），即绑定发生于程序运行过程中。

C++通过虚函数实现“动态绑定”技术。虚函数的声明方法是在基类的函数声明前或函数定义的函数头前（无函数声明时）加上virtual关键字。

例如，对例3-5中Person类的DisplayInfo()函数，只要在其函数头前加上virtual关键字：

```
virtual void DisplayInfo()
{    ……    }
```

该函数即成为虚函数。虚函数具有继承性，只要基类中的函数被声明为虚函数，则在派生类中对虚函数进行重定义时，无论是否加了virtual关键字，这个函数都是虚函数。

【例3-6】多态性示例。

```
// Person.h
#include <iostream>
using namespace std;
class Person
{
public:
    Person(char *name, bool sex)
    {
        strcpy(m_name, name);
        m_sex = sex;
    }
    virtual void DisplayInfo()
    {
        cout<<"个人信息："<<endl
            <<"姓名："<<m_name<<endl
            <<"性别：";
        if (m_sex==true) cout<<"男"<<endl;
        else cout<<"女"<<endl;
    }
protected:
    char m_name[20];        // 姓名
    bool m_sex;    // 性别（true：男，false：女）
};
```

```
// Student.h
#include "Person.h"
class Student : public Person
{
public:
    Student(char *sno, char *name, bool sex, char *major)
        : Person(name, sex)
    {
        strcpy(m_sno, sno);
        strcpy(m_major, major);
    }
    void DisplayInfo()
    {
        cout<<"学生信息："<<endl
            <<"学号："<<m_sno<<endl
            <<"姓名："<<m_name<<endl
            <<"性别：";
        if (m_sex==true) cout<<"男"<<endl;
        else cout<<"女"<<endl;
        cout<<"专业："<<m_major<<endl;
    }

private:
    char m_sno[8];          // 学号
    char m_major[20];       // 专业
};
```

```cpp
// main.cpp
#include "Student.h"
void Print(Person &rp)
{
    rp.DisplayInfo();
}

int main()
{
    Student student("1210101", "张三", true, "计算机应用");
    Person person("李四", false);
    Print(student);             // 以Student类对象作为实参
    Print(person);              // 以Person类对象作为实参
    return 0;
}
```

与例3-5相同，在主函数中两次调用Print()函数，分别将基类对象person和派生类对象student作为实参传递给基类引用rp。但在例3-6中，DisplayInfo()函数被声明为虚函数，因此在使用基类引用rp调用该函数时就可以根据rp所引用对象的不同调用不同类的成员函数，即实现了多态性。

提示：

只有使用指针或引用调用虚函数时才能实现多态性。如果使用对象调用虚函数，则不具有多态性，必然是调用该对象所属类的成员函数。例如，将例3-6中的Print()函数改为：

```
void Print(Person p)
{
    p.DisplayInfo();
}
```

则运行结果与例3-5完全一样，不具有多态性。