继承和类的派生

派生类的声明方式

其声明方式与函数重载类似，在类声明中，派生类名后跟一个冒号和基类名。
,形式如下

```
class DerivedClass : public BaseClass {
    // 派生类的成员声明
};
```

例子如下:

```
class Student : public Person {
    // 派生类的成员声明
};
```

派生类的构成

派生类中的成员包括以下几个部分:

1. 基类的公有成员和保护成员
2. 派生类自己增加的成员

构成一个派生类包括以下三部分工作:

1. 从基类接收成员(接收的是不包括构造函数和析构函数的成员)
2. 调整从基类接收的成员的访问属性

注意

- 派生类中不能直接访问基类的私有成员
- 派生类中不能直接访问基类的保护成员
- 成员函数的覆盖是指函数名和参数列表都相同

3. 添加新的成员

派生类成员的访问属性

对于基类成员和派生类新增的成员是按不同的原则处理的

在访问属性时,需要注意以下情况

- 基类的成员函数可以访问基类成员
- 派生类的成员函数可以访问派生类新增成员
- 基类的成员函数只能访问基类的成员,而不能访问派生类新增成员
- 派生类外可以访问派生类的公有成员
- 派生类外不能访问派生类的保护成员和私有成员
- 对于派生类访问基类成员的权限,取决于基类成员在基类中的访问属性

1. 基类成员是公有成员,则派生类可以访问

2. 基类成员是保护成员,则派生类可以访问
3. 基类成员是私有成员,则派生类不能访问

同时,我们还要考虑派生类成员的继承方式:

- 公有继承:基类的公有成员和保护成员在派生类中保持原有的访问属性
- 保护继承:基类的公有成员和保护成员在派生类中变为保护成员
- 私有继承:基类的公有成员和保护成员在派生类中变为私有成员

派生类的构造函数和析构函数

构造函数的主要作用是初始化派生类对象,因为构造函数不能被继承,所以派生类不能自动调用基类的构造函数,只能由派生类自己定义构造函数

在创建派生类对象时,总是先调用基类的构造函数,再调用派生类的构造函数
,调用基类构造函数的方式有两种:

1. 显式调用基类构造函数,形式如下

```
DerivedClass::DerivedClass(arglist):BaseClass(arglist) {
    // 派生类构造函数体
}
```

例子如下

```
Student::Student(string name, int age, string id, float score):Person(name, age)
{
    // 派生类构造函数体
}
```

2. 隐式调用基类构造函数

在派生类构造函数中,可以不显式调用基类构造函数,此时将调用基类的默认构造函数

简单的派生类的构造函数

单一继承时的构造函数形式如下

```
DerivedClass::DerivedClass(arglist):BaseClass(arglist) {
    // 派生类构造函数体
}
```

以下是一个完整的代码例子

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

class Person {
    private:
        string name;
        int age;
    public:
        Person(string n, int a):name(n), age(a) {}
        void display() const {
```

```
            cout << "Name: " << name << ", Age: " << age << endl;
        }
};

class Student:public Person {
    private:
        string id;
        float score;
    public:
        Student(string n, int a, string i, float s):Person(n, a), id(i), score(s)
{}
        void display() const {
            Person::display();
            cout << "ID: " << id << ", Score: " << score << endl;
        }
};

int main() {
    Student s("Zhang", 20, "1234567890", 85.5);
    s.display();
    return 0;
}
```

有子对象的派生类的构造函数

例子如下

```
#include <iostream>
using namespace std;
class Student {
    private:
        string name;
        int age;
        Address addr;
    public:
        Student(string n, int a, string city, string street, string zip):name(n),
age(a), addr(city, street, zip) {}
};

class Student1:public Student {
    private:
        string id;
        float score;
    public:
        Student1(string n, int a, string city, string street, string zip, string
i, float s):Student(n, a, city, street, zip), id(i), score(s) {}
};

int main() {
    Student1 s("Zhang", 20, "Beijing", "Haidian", "100081", "1234567890", 85.5);
    s.display();
    return 0;
}
```

多层派生时的构造函数

全代码例子如下,还是以上述的学生类为例子

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

class Student {
    private:
        string name;
        int age;
        Address addr;
    public:
        Student(string n, int a, string city, string street, string zip):name(n),
age(a), addr(city, street, zip) {}
};

class Student1:public Student {
    private:
        string id;
        float score;
    public:
        Student1(string n, int a, string city, string street, string zip, string
i, float s):Student(n, a, city, street, zip), id(i), score(s) {}
};

class Student2:public Student1 {
    private:
        string major;
    public:
        Student2(string n, int a, string city, string street, string zip, string
i, float s, string m):Student1(n, a, city, street, zip, i, s), major(m) {}
};

int main() {
    Student2 s("Zhang", 20, "Beijing", "Haidian", "100081", "1234567890", 85.5,
"Computer Science");
    s.display();
    return 0;
}
```

派生类的析构函数

析构函数的调用顺序与构造函数相反,先调用派生类的析构函数,然后调用子对象的析构函数,最后调用基类的析构函数,差不多是一个玩意

多重继承

虚基类

为了解决多重继承时可能产生的二义性问题,可以使用虚基类