第五章 继承与派生

written by SCNU第一帅----->孔文康

什么是继承与派生

在C++中可重用性是通过继承这一机制来实现的。因此，继承是C++一个重要的组成部分。

前面介绍了类，一个类中包含了若干数据成员和成员函数。

在不同类中，数据成员和成员函数是不相同的。

但有时候两个类的基本内容相同或者有一部分相同，很多人自然而然会想到能否用已声明的类作为基础，加上新内容即可.

C++提供的继承机制，就是为了解决此类问题。

继承就是在一个已存在类的基础上建立一个新的类。已存在的类叫基类或者父类。新建立的类叫派生类或者子类。

一个派生类只从一个基类派生，称为单继承。

一个派生类有两个或多个基类称为多重继承。

派生类的声明方式

假设已经声明了一个基类Student，在此基础上通过单继承建立一个派生类Student1。

```
class Student1: public Student {
 public:
 void display_1() { //新增成员函数
 cout<<"age:"<<age<<endl;
 cout<<"address:"<<addr<<endl;
 }
 private:
 int age; //新增数据成员
 string addr; //新增数据成员
}
```

继承方式包括：public(公用的)、private(私有的)、protected(受保护的)，继承方式是可选的，如果不写此项，默认为private。

派生类的构成

派生类中的成员包括从基类继承过来的成员和自己增加的成员两大部分。

1. 从基类接收成员。

派生类把基类全部成员（不包括构造函数和析构函数）接收过来。

2. 调整从基类接收的成员

可以改变派生类中成员的访问属性。

可以覆盖同名成员。

3. 在声明派生类时增加的成员

充分利用基类的成员函数。

派生类的访问属性

派生类中包含基类成员和派生类中自己增加的成员，就产生了这两部分成员关系和访问属性的问题。

公用继承(public)

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，其私有成员仍为私有。

私有继承(private)

基类的公有成员和保护成员在派生中成了私有成员。其私有成员仍为基类私有。

受保护继承(protected)

基类的公有成员和保护成员在派生类中成了保护成员，其私有成员仍为基类私有。

保护的意思是，不能被外界引用，但可以被派生类的成员引用。

公用继承

在建立一个派生类时，将基类的继承方式指定为public，称为公用继承，用公用继承建立的派生类称为公用派生类，其基类称为公用基类。

公用基类的成员在派生类中访问属性如下表。

在基类的访问属性	继承方式	在派生类的访问属性
private	public	不可访问
public	public	public
protected	public	protected

```
Class Student { // 声明基类
 public: //基类公有成员
 void get_value() {
 cin>>num>>name>>sex; //输入基类数据的成员函数
 }
 void display() {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
 cout<<"sex:"<<sex<<endl;
  }
  private: //基类私有成员
  int num;
  string name;
  char sex;
};
class Student1 : public Student { //公有继承Student类
  public:
  void get_value () { //输入派生类数据
  cin>>age>>addr;
  }
```

```
21  void display_1 () {
22  cout<<"num:"<<num<<endl; //企图引用基类私有成员，错误
23  cout<<"name:"<<name<<endl;//企图引用基类私有成员，错误
24  cout<<"sex:"<<sex<<endl;//企图引用基类私有成员，错误
25  cout<<"age:"<<age<<endl;//引用派生类私有成员，正确
26  cout<<"address:"<<addr<<endl;//引用派生类私有成员，正确
27  }
28  private:
29  int age;
30  string addr;
31 };
```

私有继承

在声明一个派生类时将基类的继承方式定为private的，称为私有继承，用私有继承方式建立的派生类称为私有派生类，其基类称为私有基类。

私有基类的公用成员和保护成员在派生类中访问属性相当于派生类中的私有成员，即派生类的成员函数能访问他们，而在派生类外不能访问他们。

私有基类的私有成员在派生类中成为不可访问的成员，只有基类的成员函数可以引用他们。

访问属性如下表。

在基类的访问属性	继承方式	在派生类的访问属性
private	private	不可访问
public	private	private
protected	private	private

```
Class Student { // 声明基类
 public: //基类公有成员
 void get_value() {
 cin>>num>>name>>sex; //输入基类数据的成员函数
 }
 void display() {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
 cout<<"sex:"<<sex<<endl;
 }
 private: //基类私有成员
 int num;
 string name;
 char sex;
};
class Student1 : private Student { //私有继承Student类
 public:
 void get_value () { //输入派生类数据
 cin>>age>>addr;
 }
 void display_1 () {
 cout<<"num:"<<num<<endl; //企图引用基类私有成员，错误
 cout<<"name:"<<name<<endl;//企图引用基类私有成员，错误
 cout<<"sex:"<<sex<<endl;//企图引用基类私有成员，错误
 cout<<"age:"<<age<<endl;//引用派生类私有成员，正确
 cout<<"address:"<<addr<<endl;//引用派生类私有成员，正确
 }
 private:
 int age;
 string addr;
};
int main () {
 Student1 stud1;
 stud1.display(); //错误，私有基类的公用函数在派生类中是私有函数
 stud1.display_1(); //正确
 stud1.age = 18; //错误，外界不能引用派生类的私有成员
 return 0;
}
```

这时候需要将对基类的公有函数调用放在派生类的函数中。

```
void display_1 () {
 display();
 cout<<"age:"<<age<<endl;//引用派生类私有成员，正确
 cout<<"address:"<<addr<<endl;//引用派生类私有成员，正确
}
```

保护成员和保护继承

由protected声明的成员称为受保护的成员。

受保护的成员不能被类外访问，这一点和私有成员很像。但是有一点不同，保护成员可以被派生类的成员函数引用。

在定义一个派生类时将基类的继承方式指定为protected的，称为保护继承，用于保护继承建立的派生类称为保护派生类，其基类称为受保护的基类。

保护继承的特点是保护基类的公有成员和保护成员在派生中都成了保护成员，其私有成员仍为基类私有。也就是把基类的原有的公有成员也保护起来，不让类外访问。

综合前面两个表，可得。

在基类的访问属性	继承方式	在派生类中的访问属性
private	public	不可访问
private	private	不可访问
private	protected	不可访问
public	public	public

public	private	private
public	protected	protected
protected	public	protected
protected	private	private
protected	protected	protected

```
Class Student { // 声明基类
 public: //基类公有成员
 void get_value() {
 cin>>num>>name>>sex; //输入基类数据的成员函数
 }
 void display() {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
 cout<<"sex:"<<sex<<endl;
  }
 private: //基类私有成员
 int num;
 string name;
 char sex;
};
class Student1 : protected Student { //保护继承Student类
 public:
 void get_value () { //输入派生类数据
 cin>>num>>name>>sex>>age>>addr;
 }
 void display_1 () {
 cout<<"num:"<<num<<endl; //引用基类保护成员，正确
 cout<<"name:"<<name<<endl;//引用基类保护成员，正确
 cout<<"sex:"<<sex<<endl;//引用基类保护成员，正确
 cout<<"age:"<<age<<endl;//引用派生类保护成员，正确
 cout<<"address:"<<addr<<endl;//引用派生类私有成员，正确
 }
 private:
 int age;
 string addr;
};
```

```
int main () {
 Student1 stud1;
 stud1.display(); //错误，保护基类的公用函数在派生类中是保护函数
 stud1.display_1(); //正确
 stud1.age = 18; //错误，外界不能引用派生类的私有成员
 return 0;
}
```

多级派生时的访问属性

以上只介绍了只有一级派生的情况，实际上，常常有多级派生的情况。

例如类A为基类，类B是类A的派生类，类C是类B的派生类，则类C也是类A的派生类。

类A是类B的直接基类，是类C的间接基类。

在多级派生的情况下，各成员的访问属性仍按照以上所说的原则进行确定。

派生类的构造函数和析构函数

前面提过，派生类并没有把基类的构造函数和析构函数继承过来，因此，对继承过来的基类成员初始化工作也由派生类的构造函数承担。

解决这个问题的思路就是： 在执行派生类的构造函数时，调用基类构造函数。

简单的派生类的构造函数

任何派生类都包含基类的成员，简单的派生类只有一个基类，而且只有一级派生，在派生类的数据成员中不包含基类的对象。

```
#include<iostream>
#include<string>
using namespace std;
class Student {
 public:
 Student(int n,string nam, char s):num(n),name(nam),sec(s) {}
 ~Student() {}
 protected:
 int num;
  string name;
  char sex;
  };
class Student1 : public Student {
  public:
  Student1(int n, string nam, char s, int a, string ad):Student(n,name,s)
{
  age = a;
  addr = ad;
  }
  ~Student1() {}
  private:
  int age;
  string addr;
}
```

有子对象的派生类构造函数

上面介绍过的类，其数据成员都是标准类型或者系统提供的类型，实际上，类中的数据成员还可以包含类对象，如可在声明一个类时候包含这样的数据成员

```
Student s1;
```

这时候，s1就是类对象中的内嵌对象，称为子对象，即对象中的对象。

```
#include<iostream>
#include<string>
using namespace std;
class Student {
 public:
 Student(int n, string nam):num(n),name(nam) {}
 void display () {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
  }
  protected:
  int num;
  string name;
};
class Student1: public Student {
  public:
  Student1(int n, string nam, int n1, string nam1, int a, string ad)
  :Student(n, nam),monitor(n1, nam1) {
  age = a;
  addr = ad;
  }
  void show () {
  cout<<"This student is:"<<endl;
  display();
  cout<<"age:"<<age<<endl;
  cout<<"address:"<<addr<<endl;
  }
  void show_monitor () {
  cout<<endl<<"class Monitor is:"<<endl;
  monitor.display();
  }
  private:
  Student monitor;
  int age;
  string addr;
}
```

多层派生的构造函数

一个类不仅可以派生出一个派生类，派生类还可以继续派生，形成派生的层次结构。

在上面叙述的基础上，不难写出在多级派生情况下的派生类构造函数。

```
#include<iostream>
#include<string>
using namespace std;
class Student {
 public:
 Student (int n, string nam):num(n),name(nam) {}
 void display () {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
  }
 protected:
 int num;
 string name;
};
class Student1 : public Student {
 public:
 Student1 (int n, char nam[10],int a):Student(n, nam) {
 age = a;
 }
 void show () {
 display();
 cout<<"age:"<<age<<endl;
 }
 private:
 int age;
}
class Student2 : public Student1 {
 public:
 Student2 (int n, string nam, int a, int s): Student1(n, nam, a) {
```

```
30   score = s;
31   }
32   void show_all () {
33   show();
34   cout<<"score: "<<score<<endl;
35   }
36   private:
37   int score;
38  };
```

派生类的析构函数

在派生时，派生类时不能继承基类的析构函数的，也需要通过派生类的析构函数去调用基类的析构函数。

在派生类中可以根据需要定义自己的析构函数，用来对派生类中的所增加的成员进行清理工作。基类的清理工作仍然由基类的析构函数负责。

在执行派生类的析构函数时，系统会自动调用基类的析构函数和子对象的析构函数。

调用的顺序与构造函数正好相反：

先执行派生类自己的析构函数，然后调用子对象的析构函数，最后调用基类的析构函数。

多重继承

允许一个派生类同时继承多个基类，这种行为称为多重继承。

声明多重继承的方法

如果已声明类A，类B，类C，可以声明多重继承的派生类D

```
class D :public A, private B, protected C {

}
```

多重继承派生类的构造函数

```
#include<iostream>
#include<string>
using namespace std;
class Teacher {
 public:
 Teacher (string nam, int a, string t)
 :name(nam),age(a),title(t) {}
 void display () {
 cout<<"age:"<<age<<endl;
  cout<<"name:"<<name<<endl;
  cout<<"title:"<<title<<endl;
  }
  protected:
  string name;
  int age;
  string title;
}
class Student {
 public:
 Student(char nam[], char s, float sco) {
 strcpy(name1, nam);
 sex = s;
 score = sco;
 }
 void display1 () {
 cout<<"name:"<<name1<<endl;
 cout<<"sex:"<<sex<<endl;
 cout<<"score:"<<score<<endl;
 }
 protected:
```

```
 string name1;
 char sex;
 float score;
}
class Graduate : public Teacher, public Student {
 public:
 Graduate (string nam, int a, char s, string t, float sco, float w)
 :Teacher(nam, a, t),Student(nam,s,sco),wage(w) {}
 void show () {
 display();
 display1();
 cout<<"wages:"<<wage<<endl;
 }
 private:
 float wage;
}
```

多重继承引起的二义性问题

多重继承可以反映现实生活中的情况，能够有效地处理一些比较复杂的问题，使程序具有灵活性。

但是它增加了程序的复杂度，程序的编写和维护变得相对困难，容易出错。

最常见的就是二义性问题。

```
class A {
 public:
 int a;
 void display();
};
class B {
 public:
 int a;
 void display();
}
```

```
class C : public A, public B {
 public:
 int b;
 void show ();
}
// 如果在main函数中调用
C c1;
c1.a = 3;
c1.display();
// 无法判断要访问的是哪一个基类的成员
// 增加作用域运算符
c1.A::a = 3;
c1.A::display();
```

虚基类

虚基类的作用

如果一个派生类有多个直接基类，而这些直接基类又有一个共同的基类，则在最终的派生类中会保留该间接基类数据成员的多份同名成员。

在引用这些同名成员时，必须在派生类对象名后直接加基类名，以避免二义性。

在一个类中保留间接共同基类的多份同名成员，虽然有时候是必要的，可以在不同的数据成员中分别存放不同的数据，也可以通过构造函数分别对他们进行初始化。

但是在大多数情况下，这种现象是人们不希望出现的，因为保留多份数据，不仅占用较多的存储空间，还增加了访问的困难，容易出错。

C++提供虚基类的方法，使得在继承间接共同基类时只保留一份成员。

```
#include<iostream>
#include<string>
using namespace std;
class Person {
 public:
 Person(string nam, char s, int a)
 :name(nam),sex(s),age(a) {

 }
  protected:
  string name;
  char sex;
  int age;
};
Class Teacher : virtual public Person {
  public:
  Teacher(string nam, char s, int a, string t)
  :Person(nam,s,a) {
  title = t;
  }
  protected:
  string title;
};
class Student : virtual public Person {
  public:
  Student (string nam, char s, int a, float sco)
  :Person(nam,s,a),score(sco) {}
  protected:
  float score;
};
class Graduate:public Teacher, public Student {
  public:
  Graduate (string nam, char s, int a, string t,float sco,float w)
  :Person(name,s,a),Teacher(nam,s,a,t),Student(nam,s,a,sco),wage(w) {
  }
  void show () {
  cout<<"name:"<<name<<endl;
  cout<<"age:"<<age<<endl;
  cout<<"sex:"<<sex<<endl;
```

```
  cout<<"score:"<<score<<endl;
  cout<<"title:"<<title<<endl;
  cout<<"wages:"<<wage<<endl;
  }
  private:
  float wage;
};
```

在类Graduate中，只保留了一份基类的成员，因此可以直接引用，不会产生二义性。

基类与派生类的转换

只有公有派生类才是基类的真正子类型，它完整地继承了基类的功能。

前面已说明，不同类型数据之间的自动转换和赋值，称为赋值兼容。

基类与派生类对象之间有赋值兼容关系，由于派生类中包含从基类继承的成员，因此可以将派生类对象的值赋给基类对象，在用到基类对象的时候，可以用其子类对象代替，具体表现在以下几个方面。

1、派生类对象可以向基类对象赋值

可以用子类（公用派生类）对象对其基类对象赋值

```
A a1; //定义基类对象a1
B b1; //定义A类的公用派生类B的对象b1
a1 = b1; //用派生类B对象b1对基类对象a1赋值
```

在赋值时，舍弃派生类自己的成员。

实际上，所谓赋值只是对数据成员赋值，对成员函数不存在赋值问题。

请注意

赋值后不能企图通过对象a1去访问派生类对象b1的成员因为b1的成员与a1的成员是不同的。

只能用子类对象对其基类对象赋值，而不能用基类对象对其子类对象赋值。

同一基类的不同派生类对象之间不能赋值。

2、派生对象可以替代基类对象向基类对象的引用进行赋值或初始化

如定义了基类A对象a1,可以定义a1的引用变量

```
A a1;
B b1;
A& r = a1;
```

请注意

这时候，r是a1的引用，r和a1共享同一段存储单元。

```
A& r = b1;
```

请注意

此时r并不是b1的别名，也不是与b1共享同一段存单元。它只是b1中基类部分的别名，r与b1中基类部分共享同一存储单元。

r与b1具有相同的起始地址。

3、如果函数的参数是基类对象或基类对象的引用，相应的实参可以用子类对象

```
void fun (A& r) {
 cout<<r.num<<endl;
}
```

函数的形参是类A的对象引用，本来实参应该为类A的对象。由于子类对象与派生类对象赋值兼容，派生类对象能自动转换类型，在调用fun函数时可以用派生类B对象b1作为实参：

```
fun (b1);
```

4、派生类对象的地址可以赋给指向基类对象的指针变量，也即是说，指向基类对象的指针变量也可以用来指向派生类对象。

```
A a1;
B b1;
A* pt = &a1;
pt = &b1;
```

继承与组合

在一个类中以另一个类的对象作为数据成员的，叫做类的组合。

```
class Teacher {
 public:
 private:
 int num;
 string name;
 char sex;
}
class BirthDate {
```

```
9  public:
10  private:
11  int yaer;
12  int month;
13  int day;
14 }
15 class Professor : public Teacher {
16  public:
17  private:
18  BirthDate birthday;
19 }
```

类的组合和继承一样，是软件重用的重要方式。

通过继承建立了派生类与基类的关系。
通过组合建立了成员类和组合类的关系。
在本例中BirthDate是成员类，Professor是组合类。