CSDN 学院

C++语言基础

迁者 - 贺利坚

http://blog.csdn.net/sxhelijian/

http://edu.csdn.net

本节主题：

虚函数

指向基类的指针，为何只能访问来自基类成员？！

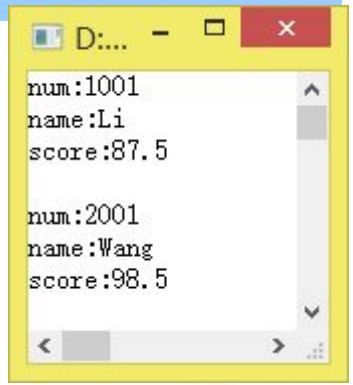
```
class Student{
public:
   Student(int, string,float);
   void display( );
protected:
   int num;
   string name;
   float score;
};
```

```
class Graduate:public Student
{
public:
   Graduate(int, string, float, float);
   void display( );
private:
   float pay;
};
```

```
int main(){
   Student stud1(1001,"Li",87.5);
   Graduate grad1(2001,"Wang",98.5,563.5);
   Student *pt=&stud1;
   pt->display( );
   pt=&grad1;
   pt->display( );
   return 0;
}
```

```
void Student::display( ) {
   cout<<"num:"<<num<<"\nname:"<<name<<"\nscore:"<<score<<"\n\n";
}
```

```
void Graduate::display( ) {
   cout<<"num:"<<num<<"\nname:"<<name<<"\nscore:"<<score<<"\npay="<<pay<<endl;
}
```



一招虚函数，从此出樊笼

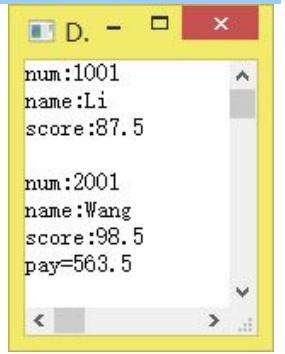
```
class Student{
public:
   Student(int, string,float);
   virtual void display( );
protected:
   int num;
   string name;
   float score;
};
```

```
class Graduate:public Student
{
public:
   Graduate(int, string, float, float);
   void display( );
private:
   float pay;
};
```

```
int main(){
   Student stud1(1001,"Li",87.5);
   Graduate grad1(2001,"Wang",98.5,563.5);
   Student *pt=&stud1;
   pt->display( );
   pt=&grad1;
   pt->display( );
   return 0;
}
```

```
void Student::display( ) {
   cout<<"num:"<<num<<"\nname:"<<name<<"\nscore:"<<score<<"\n\n";
}
```

```
void Graduate::display( ) {
   cout<<"num:"<<num<<"\nname:"<<name<<"\nscore:"<<score<<"\npay="<<pay<<endl;
}
```



虚函数使用方法

- 在基类用virtual声明成员函数为虚函数
  - 类内声明： **virtual** [类型] 函数名([参数表列])
- 在类外定义虚函数时，不再加virtual
  - 不再加，非不必再加
- 效果
  - 派生类根据需要可以重新定义函数体（函数覆盖），使用虚函数提高了程序的可扩充性
  - 成员函数被声明为虚函数后，其派生类中函数覆盖的同名函数自动成为虚函数
  - 若虚函数在派生类中未重定义，则派生类简单地继承其直接基类的虚函数
  - 指向基类的指针，当指向派生类对象时，则可以调用派生类的方法

```
class Student{
public:
   Student(int, string,float);
   virtual void display( );
protected:
   … …
};
```

```
void Student::display( )
{  … … }
```

```
class Graduate:public Student
{ … … };
```

```
void Graduate::display( )
{ … … }
```

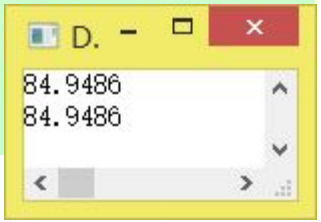
对比：未用虚函数

```
class Circle
{
public:
   Circle(double r=0):radius(r) { }
   double area ( ) const
   {
      return 3.14159*radius*radius;
   }
protected:
   double radius;
};
```

```
class Cylinder:public Circle
{
public:
   Cylinder (double r=0,double h=0)
      :Circle(r),height(h) {}
   double area() const
   {
      return 2*Circle::area( )+2*3.14159*
        Circle::radius*height;
   };
protected:
   double height;
};
```

```
int main( )
{
   Circle c1(5.2);
   Cylinder cy1(5.2, 10);

   Circle *pc=&c1;
   cout<<pc->area()<<endl;
   pc = &cy1;
   cout<<pc->area()<<endl;
   return 0;
}
```



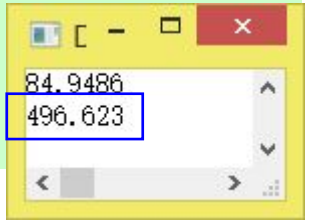
对比：用了虚函数

```
class Circle
{
public:
   Circle(double r=0):radius(r) { }
   virtual double area ( ) const
   {
      return 3.14159*radius*radius;
   }
protected:
   double radius;
};
```

```
class Cylinder:public Circle
{
public:
   Cylinder (double r=0,double h=0)
      :Circle(r),height(h) {}
   double area() const
   {
      return 2*Circle::area( )+2*3.14159*
        Circle::radius*height;
   };
protected:
   double height;
};
```

```
int main( )
{
   Circle c1(5.2);
   Cylinder cy1(5.2, 10);

   Circle *pc=&c1;
   cout<<pc->area()<<endl;
   pc = &cy1;
   cout<<pc->area()<<endl;
   return 0;
}
```



事情本就该这样——抽象的动物不会叫

```
class Animal
{
public:
  virtual void cry()
    {
      cout<<"咋叫？"<<endl;
    }
};
```

```
class Dog: publlic Animal
{
public:
    void cry()
    {
        cout<<"汪！"<<endl;
    }
};

class Cat: publlic Animal
{...};

class Mouse: publlic Animal
{...};
...
```

```
int main( ){
    Animal *p, a;
    p = &a;
    p->cry();
    Dog d;
    p = &d;
    p->cry();
    Cat c;
    p = &c;
    p->cry();
    Mouse m;
    p = &m;
    p->cry();
    return 0;
}
```

静态关联与动态关联

- 理解关联(binding)
  - 通过关联，把一个标识符和一个存储地址联系起来
  - 通过关联，可以把一个函数名与一个类对象捆绑在一起
- 静态关联(static binding)
  - 函数重载和通过对象名调用的(虚)函数，在编译时即可确定其调用的(虚)函数属于哪一个类
  - 又称为早期关联(early binding)。
- 动态关联(dynamic binding)
  - 通过基类指针与虚函数，在运行阶段确定关联关系。
  - 动态关联提供动态的多态性，即运行阶段的多态性。
  - 也称为滞后关联(late binding)。

更明白虚函数.....

- 虚函数只能是类的成员函数，而不能将类外的普通函数声明为虚函数。
  - 虚函数的作用是允许在派生类中对基类的虚函数重新定义（函数覆盖），只能用于类的继承层次结构中。
- 排他性
  - 一个成员函数被声明为虚函数后，在同一类族中的类就不能再定义**非virtual，但与该虚函数具有相同的参数(包括个数和类型)和函数返回值类型的**同名函数。
- 使用虚函数，系统要有少量的空间开销
  - 当一个类带有虚函数时，编译系统会为该类构造一个虚函数表(一个指针数组)，用于存放每个虚函数的入口地址。

何时用虚函数

- 先看成员函数所在类是否会当基类
  - 所在类不会当基类，省点事吧
- 再看成员函数在类被继承后有无可能被更改功能
  - 如果希望更改其功能的，一般应该在基类中将其声明为虚函数
  - 如果成员函数在类被继承后功能不需修改，或派生类用不到该函数，则不声明为虚函数
- 应考虑对成员函数的调用是通过对象名还是通过基类指针或引用去访问
  - 如果是通过基类指针或引用去访问的，则应当声明为虚函数。
- 有时，在定义虚函数时，并不定义其函数体，即函数体是空的，只等着被继承
  - 它的作用只是定义了一个虚函数名，具体功能留给派生类去添加——纯虚函数。

wxWidgets中利用虚函数

```
class MyApp : public wxApp
{
public:
   // Called on application startup
   virtual bool OnInit();
};

class WXDLLIMPEXP_CORE wxApp : public wxAppBase
{
public:
   wxApp();
   virtual ~wxApp();

   // override base class (pure) virtuals
   virtual bool Initialize(int& argc, wxChar **argv);
   virtual void CleanUp();
   ……
};
```

CSDN 学院

THANKS

本课程由 迁者-贺利坚 提供

CSDN网站：www.csdn.net
企业服务：http://ems.csdn.net/
人才服务：http://job.csdn.net/
CTO俱乐部：http://cto.csdn.net/
高校俱乐部：http://student.csdn.net/
程序员杂志：http://programmer.csdn.net/

CODE平台：https://code.csdn.net/
项目外包：http://www.csto.com/
CSDN博客：http://blog.csdn.net/
CSDN论坛：http://bbs.csdn.net/
CSDN下载：http://download.csdn.net/