面向对象程序设计

虚函数

2020 年 春

耿楠

计算机科学系
信息工程学院

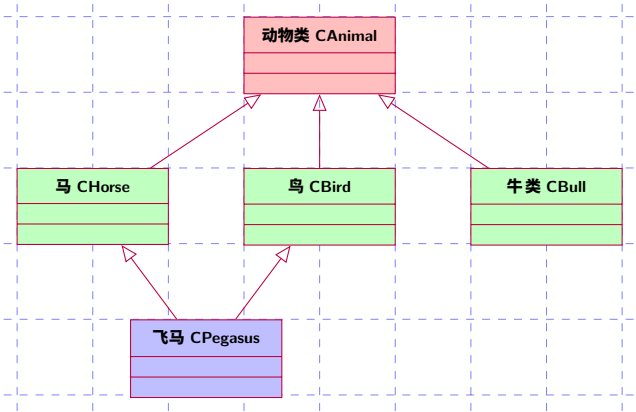
西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国 · 杨凌

- 为派生类提供一致的接口 (Uniform Interface)

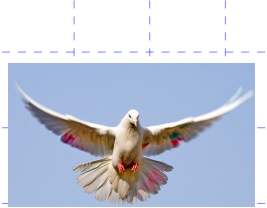
- 多态性：同一消息发送给不同对象执行不同操作

- 设计一个动物园类
  - 包含大量不同种类的动物
  - 可以显示各种动物的属性（如名字、年龄等）
  - 可以输出各种动物的叫声

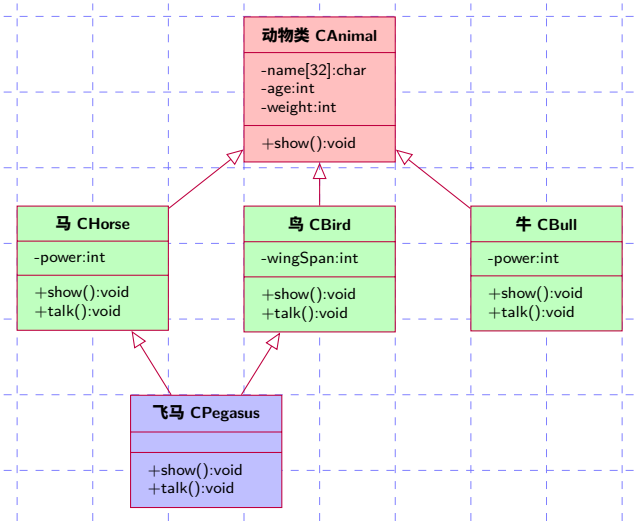
- 设计一个动物园类



- 设计一个动物园类




- 设计一个动物园类


动物类 CAnimal
-name[32]:char
-age:int
-weight:int
+show():void
马 CHorse
-power:int
+show():void
+talk():void
鸟 CBird
-wingSpan:int
+show():void
+talk():void
牛 CBull
-power:int
+show():void
+talk():void
飞马 CPegasus
+show():void
+talk():void

- 设计一个动物园类

```
// 例 01-zoo-CAnimal: ex01-zoo-CAnimal.cpp

class CAnimal
{
   char name[32];
   int age;
   int weight;
public:
   CAnimal(const char *strName = "", int a = 0, int w = 0)
   {
      strcpy(name, strName);
      age = a;
      weight = w;
   }
   void Show()
   {
      cout << name << " " << age << " " << weight  << " ";
   }
};
```

设计一个动物园类

```
// 例 01-zoo-CBird: ex01-zoo-CBird.cpp
// 虚继承的演示

class CBird: virtual public CAnimal
{
protected:
   int wingSpan;
public:
   CBird(const char *strName = "", int a = 0, int w = 0, int ws = 0):
       CAnimal(strName, a, w)
   {
      wingSpan = ws;
   }
   void Show()
   {
      CAnimal::Show();
      cout << wingSpan << endl;
   }
   void Talk()
   {
      cout << "Chirp..." << endl;
      PlaySound("Sound\\eagle.wav", NULL, SND_FILENAME | SND_SYNC);
   }
};
```

设计一个动物园类

```
// 例 01-zoo-CHorse: ex01-zoo-CHorse.cpp
// 虚继承的演示

class CHorse: virtual public CAnimal
{
protected:
   int power;
public:
   CHorse(const char *strName = "", int a = 0, int w = 0, int pow = 0):
        CAnimal(strName, a, w)
   {
      power = pow;
   }
   void Show()
   {
      CAnimal::Show();
      cout << power << endl;
   }
   void Talk()
   {
      cout << "Whinny..." << endl;
      PlaySound("Sound\\horse.wav", NULL, SND_FILENAME | SND_SYNC);
   }
};
```

设计一个动物园类

```
// 例 01-zoo-CPegasus: ex01-zoo-CPegasus.cpp
//定义一个类 CPegasus，多重继承

class CPegasus : public CHorse, public CBird
{
public:
   CPegasus(const char *strName = "", int a = 0, int w = 0, int ws = 0, int pow = 0):
      CAnimal(strName, a, w),
      CHorse(strName, a, w, pow),
      CBird(strName, a, w, ws)

   {
   }
   void Show()
   {
      CAnimal::Show();
      cout << wingSpan << " " << power << endl;
   }
   void Talk()
   {
      CHorse::Talk();
   }
};
```

- 设计一个动物园类

```
// 例 01-zoo-CBull: ex01-zoo-CBull.cpp
// 继承的演示

class CBull: public CAnimal
{
   int power;
public:
   CBull(const char *strName = "", int a = 0, int w = 0, int pow = 0):
       CAnimal(strName, a, w)
   {
      power = pow;
   }
   void Show()
   {
      CAnimal::Show();
      cout << power << endl;
   }
   void Talk()
   {
      cout << "Moo..." << endl;
      PlaySound("Sound\\bull.wav", NULL, SND_FILENAME | SND_SYNC);
   }
};
```

设计一个动物园类

```
// 例 01-zoo-main: ex01-zoo-main.cpp
// 实例化 CBird,CHorse,CBull,CPegasus 的对象，并调用相应的方法

int main()
{
   CBird birdObj("Eagle", 5, 50, 2);
   CHorse horObj("Mogolia horse", 5, 1000, 10000);
   CBull bullObj("Africa ox", 3, 2000, 20000);
   CPegasus pegObj("Pegasus", 5, 5000, 4, 100000);

   birdObj.Show();
   birdObj.Talk();

   horObj.Show();
   horObj.Talk();

   bullObj.Show();
   bullObj.Talk();

   pegObj.Show();
   pegObj.Talk();

   return 0;
}
```

- 如果有大量的不同种类的动物，如何进行管理？
- 设计一个动物园类

```
// 例 02-zoo: ex02-zoo.cpp

class CZoo
{
   int sizeBird, sizeHorse, sizeBull, sizePeg;
   CBird *m_bird;
   CHorse *m_horse;
   CBull *m_bull;
   CPegasus *m_peg;
public:
   CZoo()
   {
      sizeBird = sizeBull = sizeHorse = sizePeg = 0;
   }
   void AddBird(CBird &bird) {}
   void AddBull(CBull &bull) {}
   void AddHorse(CHorse &hor) {}
   void AddPegasus(CPegasus &peg) {}
   void Show();
   void Talk();
};
```

多种动物

```
// 例 02-zoo-show: ex02-zoo-show.cpp
// 实现动物园的 Show 方法

void CZoo::Show()
{
   for (int i = 0; i < sizeBird; i++)
      m_bird[i].Show();
   for (int i = 0; i < sizeHorse; i++)
      m_horse[i].Show();
   for (int i = 0; i < sizeBull; i++)
      m_bull[i].Show();
   for (int i = 0; i < sizePeg; i++)
      m_peg[i].Show();
}
```

```
// 例 02-zoo-talk: ex02-zoo-talk.cpp
// 实现动物园的 Talk 方法

void CZoo::Talk()
{
   for (int i = 0; i < sizeBird; i++)
      m_bird[i].Talk();
   for (int i = 0; i < sizeHorse; i++)
      m_horse[i].Talk();
   for (int i = 0; i < sizeBull; i++)
      m_bull[i].Talk();
   for (int i = 0; i < sizePeg; i++)
      m_peg[i].Talk();
}
```

- 缺点：对具有相同基类的派生类对象需要分别设计
- 若派生类数量增加，管理复杂度上升

```
// 例 02-zoo: ex02-zoo.cpp

class CZoo
{
   int sizeBird, sizeHorse, sizeBull, sizePeg;
   CBird *m_bird;
   CHorse *m_horse;
   CBull *m_bull;
   CPegasus *m_peg;
public:
   CZoo()
   {
      sizeBird = sizeBull = sizeHorse = sizePeg = 0;
   }
   void AddBird(CBird &bird) {}
   void AddBull(CBull &bull) {}
   void AddHorse(CHorse &hor) {}
   void AddPegasus(CPegasus &peg) {}
   void Show();
   void Talk();
};
```

- 另外一种考虑方式：类型兼容

```
// 例 03-zoo-class: ex03-zoo-class.cpp

const int MAX_ANIM_NUM = 100;

class CZoo
{
   int size;
   CAnimal *m_animal[MAX_ANIM_NUM];
public:
   CZoo()
   {
      size = 0;
   }
   void Add(CAnimal *anim)
   {
      if(size < MAX_ANIM_NUM)
      {
         m_animal[size] = anim;
         size++;
      }
   }
   void Show();
   void Talk();
};
```

```
// 例 03-zoo-show-talk:
↪  ex03-zoo-show-talk.cpp
// 实现类中的成员函数

void CZoo::Show()
{
   for (int i = 0; i < size; i++)
      m_animal[i]->Show();
}

void CZoo::Talk()
{
   for (int i = 0; i < size; i++)
      m_animal[i]->Talk();
}
```

```
In member function 'void CZoo::Talk()':
143   error: 'class CAnimal' has no member named 'Talk'
```

- 问题 1：基类不能调用派生类成员函数 Talk
- 解决办法：虚函数

```cpp
// 例 04-zoo-virtfun-show.cpp
// 实现 CAnimal 中的虚函数 Show

virtual void CAnimal::Show()
{
   cout << name << " " << age <<
   " " << weight  << " ";
}
```

```cpp
// 例 04-zoo-virtfun-talk.cpp
// 实现 CAnimal 中的虚函数 Talk

virtual void CAnimal::Talk()
{
   cout << "Do nothing!" << endl;
}
```

- 问题 1：基类不能调用派生类成员函数 Talk
- 解决办法：虚函数

```
// 例 04-zoo-main: ex04-zoo-main.cpp

int main()
{
   CBird birdObj("Eagle", 5, 50, 2);
   CHorse horObj("Mogolia horse", 5, 1000, 10000);
   CBull bullObj("Africa ox", 3, 2000, 20000);
   CPegasus pegObj("Pegasus", 5, 5000, 4, 100000);

   CZoo zoo;

   zoo.Add(&birdObj);
   zoo.Add(&horObj);
   zoo.Add(&bullObj);
   zoo.Add(&pegObj);

   zoo.Show();
   zoo.Talk();

   return 0;
}
```

定义

```
virtual 函数类型 函数表（形参表）
{
  函数体;
}
```

- 只有类的成员函数才能声明为虚函数
- 虚函数不能是静态成员函数，也不能是友元函数
- 若在基类中定义虚函数，在派生类中还需重新定义
- 构造函数不能是虚函数，析构函数可以是虚函数

- 定义

```
virtual ~类名 ()
{
  函数体;
}
```

虚析构函数

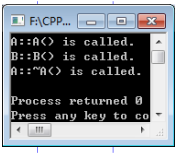
```
// 例 05-dtor-base: ex05-dtor-base.cpp
// 构造函数和析构函数的演示

class A
{
public:
   ~A()
   {
      cout << "A::~A() is called." << endl;
   }
   A()
   {
      cout << "A::A() is called." << endl;
   }
};
```

```
// 例 05-dtor-derived: ex05-dtor-derived.cpp
// 定义类 B，该类继承类 A

class B: public A
{
private:
   int *ip;
public:
   B(int size = 0)
   {
      ip = new int[size];
      cout << "B::B() is called." << endl;
   }
   ~B()
   {
      cout << "B::~B() is called." << endl;
      delete []ip;
   }
};
```

```
// 例 05-dtor-main: ex05-dtor-main.cpp
int main()
{
   A *b = new B(10);
   delete b;
   return 0;
}
```

②

①



虚析构函数

```
// 例 06-vdtor-base: ex06-vdtor-base.cpp
// 类虚析构函数的演示

class A
{
public:
   virtual ~A() //虚析构函数
   {
      cout << "A::~A() is called." << endl;
   }
   A()
   {
      cout << "A::A() is called." << endl;
   }
};
```

```
// 例 05-dtor-derived: ex05-dtor-derived.cpp
// 定义类 B，该类继承类 A

class B: public A
{
private:
   int *ip;
public:
   B(int size = 0)
   {
      ip = new int[size];
      cout << "B::B() is called." << endl;
   }
   ~B()
   {
      cout << "B::~B() is called." << endl;
      delete []ip;
   }
};
```

```
// 例 05-dtor-main: ex05-dtor-main.cpp
int main()
{
   A *b = new B(10);
   delete b;
   return 0;
}
```

②

①

```
F:\CPP...
A::A() is called.
B::B() is called.
B::~B() is called.
A::~A() is called.

Process returned 0
```

- 定义基类的析构函数是虚析构函数，当程序运行结束时，通过基类指针删除派生类对象时，先调用派生类析构函数，然后调用基类析构函数

- 纯虚函数定义

```
virtual 函数类型 函数名 (参数表)=0;
```

- 抽象类定义
  - 类中含有至少一个纯虚函数

- 回顾动物类中的虚函数 Talk→ 函数体无意义

```
virtual void CAnimal::Talk()
{
  cout << "Do nothing!" << endl;
}
```

- 解决办法：纯虚函数

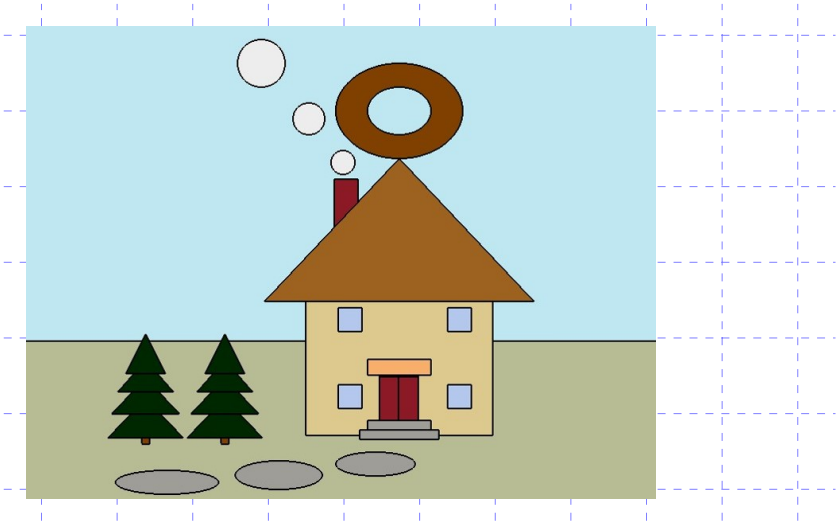
```
virtual void CAnimal::Talk() = 0;
```

- 不能实例化（不能创建类的对象）

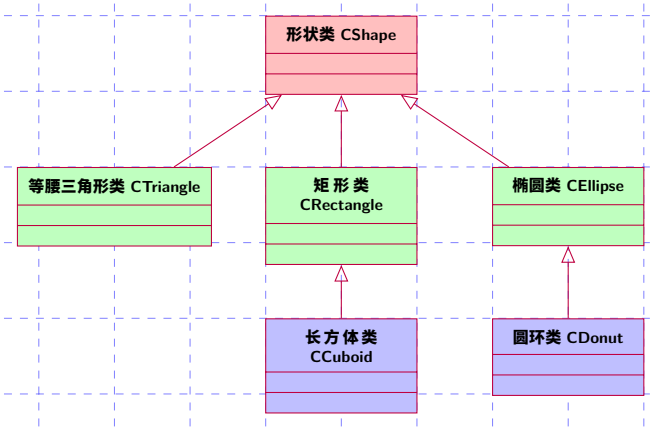
```
// 例 07-abstract-main.cpp

int main()
{
  CAnimal anim("God");

  anim.Show();
  anim.Talk();

  return 0;
}
```

```
In function 'int main()':
142   error: cannot declare variable 'anim' to be of abstract type 'CAnimal'
8     note:   because the following virtual functions are pure within 'CAnimal':
24    note:     virtual void CAnimal::Talk()
      === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

- 例子：甜甜圈小屋分析



- 例子：甜甜圈小屋类设计


形状类 CShape
等腰三角形类 CTriangle
矩形类
CRectangle
椭圆类 CEllipse
长方体类
CCuboid
圆环类 CDonut

- 例子：甜甜圈小屋代码设计

```
// 例 08-abstract-Shape: ex08-abstract-Shape.cpp
// 纯虚函数的定义

class CShape{
protected:
   ULONG textColor;
   string strText;
   CPoint2D wPos;
   ULONG objColor;
public:
   CShape():wPos(CPoint2D(400,300)){
      textColor = 0x000000;
      objColor = 0xBBE0E3;
      strText = "";
   }
   CShape(CPoint2D w, string strText="",
        ULONG objColor = 0xBBE0E3,
        ULONG textColor = 0):wPos(w){
      this->textColor = textColor;
      this->objColor = objColor;
      this->strText = strText;
   }
   void Translate(float x, float y);
   void DrawText();
   virtual void Draw() = 0;
   virtual void ShowPos() = 0;
};
```

- 例子：甜甜圈小屋代码设计

```
// 例 08-abstract-ShapeArr: ex08-abstract-ShapeArr.cpp

#ifndef SHAPEARR_H
#define SHAPEARR_H

#include "Donut.h"
#include "Triangle.h"
#include "Rectangle.h"
//定义类 CShapeArr，该类具有类 CShape 的成员变量
const int MAX_SHAPE_NUM = 1000;

class CShapeArr
{
  CShape *m_shape[MAX_SHAPE_NUM];
  int size;
public:
  CShapeArr();
  void Add(CShape *node);
  bool GetInput();
  void ShowPos();
  void Translate(int x, int y);
  void Draw();
  ~CShapeArr();
};

#endif // SHAPEARR_H
```

基类类型指针数组

基类类型指针形参

- 例子：甜甜圈小屋代码设计

```
// 例 08-abstrace-ShapeArr: ex08-abstract-ShapeArr.cpp
// 类成员函数的实现

void CShapeArr::Draw()
{
   for (int i=0; i<size; i++)
   {
      m_shape[i]->Draw();
   }
}

void CShapeArr::ShowPos()
{
   for (int i=0; i<size; i++)
      m_shape[i]->ShowPos();
}

void CShapeArr::Translate(int x, int y)
{
   for (int i=0; i<size; i++)
   {
      m_shape[i]->Translate(x,y);
   }
}
```

基类类型指针指向派生类对象，调用派生类中的函数

- 例子：甜甜圈小屋代码设计

```
// 例 08-abstract-dtor: ex04-abstract-dtor.cpp
// 实现类 CShapeArr 的析构函数

CShapeArr::~CShapeArr()
{
   //dtor
   if(size != 0)
   {
      for(int i=0; i< size; i++)
      {
         if(m_shape[i]!=NULL)
            delete m_shape[i];
      }
   }
}
```

基类类型指针指向派生类对象，调用派生类中的函数

- 例子：甜甜圈小屋-测试数据

```
/*
Rectangle 800 400 400 400 sky 0xC0E8F2 0x000000
Rectangle 800 200 400 100 earth 0xB8BD95 0x000000
Donut 0.5 80 60 474 490 rooftop 0x814000 0x000000
Rectangle 30 100 407 354 chimney 0x8C1926 0x000000
Ellipse 15 15 403 425 smoke1 0xFFEEEE 0x000000
Ellipse 20 20 360 480 smoke2 0xFFEEEE 0x000000
Ellipse 30 30 300 550 smoke3 0xFFEEEE 0x000000
Rectangle 236 200 474 181 wall 0xDECA8F 0x000000
Triangle 340 180 474 340 roof 0x9E6120 0x000000
Rectangle 30 30 412 227 window11 0xB5C9EE 0x000000
Rectangle 30 30 550 227 window1 0xB5C9EE 0x000000
Rectangle 30 30 412 130 window12 0xB5C9EE 0x000000
Rectangle 30 30 550 130 window2 0xB5C9EE 0x000000
Rectangle 80 20 474 167 eave 0xF9B06B 0x000000
Rectangle 25 55 462 128 door1 0x8C1926 0x000000
Rectangle 25 55 486 128 door2 0x8C1926 0x000000
Rectangle 80 12 474 94 step1 0x9F9E99 0x000000
Rectangle 100 12 474 82 step2 0x9F9E99 0x000000
Ellipse 50 15 444 45 stone1 0x9F9E99 0x000000
Ellipse 55 18 322 31 stone2 0x9F9E99 0x000000
Ellipse 65 15 181 22 stone3 0x9F9E99 0x000000
Rectangle 10 15 154 78 trunk11 0x814000 0x000000
Triangle 95 50 154 103 crown11 0x002800 0x000000
Triangle 85 45 154 131 crown12 0x002800 0x000000
Triangle 70 40 154 156 crown13 0x002800 0x000000
Triangle 50 50 154 184 crown14 0x002800 0x000000
Exit
*/
```

- 练习：形状类的设计
  - 设计要求
    - 设计一个 CShape 抽象类，类中包含纯虚函数
    - 从 CShape 类派生出三角形类 CTriangle、矩形类 CRectangle 和椭圆类 CEllipse
    - 使用一个公共接口计算三角形对象、矩形对象和椭圆对象面积之和
    - 重载运算符 $>$ 用于判断两个形状面积的大小，返回 true 或 false

- 运行时刻类型识别 (RunTime Type Identification)：允许使用基类指针或引用操纵对象的程序获得这些指针或引用实际所指对象的类型。

- **dynamic_cast**运算符

- **typeid**运算符

- static_cast
- dynamic_cast
  - 向下类型转换
- reinterpret_cast
- const_cast

▶ 向下类型转换

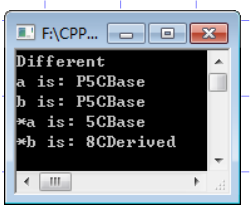
```
// 例 09-rtti-class: ex09-rtti-class.cpp

class CBase
{
   // Since RTTI is included in the virtual method table
   // there should be at least one virtual function.
public:
   virtual void foo() {}
   void methodBase()
   {
      cout << "method Base" << endl;
   }
};

class CDerived : public CBase
{
public:
   void methodDerived()
   {
      cout << "method Derived" << endl;
   }
};
```

```
// 例 09-rtti-fun: ex09-rtti-fun.cpp
// 动态类型转换的演示

void my_function(CBase* my_a)
{
   CDerived *my_b = dynamic_cast<CDerived*>(my_a);
   if(my_b != NULL)
      my_b->methodDerived();
   else
      my_a->methodBase();
}
```

向下类型转换

```
// 例 09-rtti-main: ex09-rtti-main.cpp
//实例化 CBase 和 CDerived 的对象，并测试他们的类型

#include <iostream>
#include <typeinfo>

using namespace std;

int main( void )
{
   CBase* a = new CBase;
   CBase* b = new CDerived;
   if(typeid(* a) == typeid(* b))
      cout << "Identical" << endl;
   else
      cout << "Different" << endl;
   cout << "a is: " << typeid( a ).name() << endl;
   cout << "b is: " << typeid( b ).name() << endl;
   cout << "*a is: " << typeid( *a ).name() << endl;
   cout << "*b is: " << typeid( *b ).name() << endl;
   return 0;
}
```

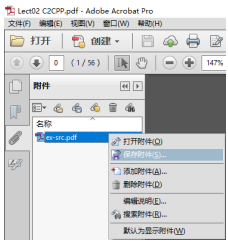
编译器不同，输出可能不同!

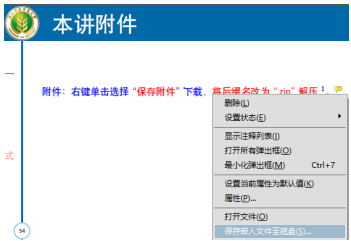


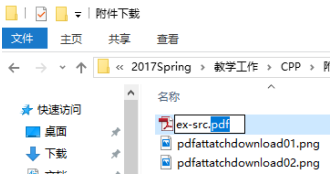
- 静态联编 (static binding or early binding)
  - 在编译时确定同名函数的具体操作对象
    - 重载函数
    - 函数模板
    - 运算符重载
  - 特点：执行效率高，但灵活性差
- 动态联编 (dynamic binding or late binding)
  - 在程序运行时 (run time) 确定对象所调用的函数
    - 虚函数
    - 纯虚函数
  - 特点：灵活性强，但效率可能会降低

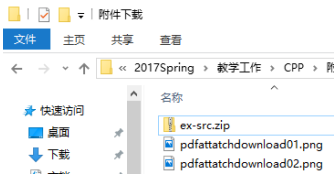
- 多态性是面向对象程序设计的重要特性之一，多态是指同样的消息被不同类型的对象接收时导致完全不同的行为。
- 虚函数是在基类中定义的以 virtual 关键字作为开头的成员函数，需要在派生类中重新定义。
- 纯虚函数是一个在基类中声明的没有定义具体实现的虚函数，带有纯虚函数的类称为抽象类。
- 抽象类为一个类族提供统一的操作界面。抽象类处于类层次的上层，自身无法实例化，只能通过继承机制，生成抽象类的具体派生类，然后再实例化。通过抽象类的指针和引用，可以指向并访问各派生类成员，实现多态性。
- 联编可以在编译和连接时进行，称为静态联编；联编也可以在运行时进行，称为动态联编。

附件：右键单击该链接，选择“保存附件”下载，将后缀名改为“.zip”解压[1] [2]。









[1] 请退出全屏模式后点击该链接。

[2] 以 Adobe Acrobat Reader 为例。

本讲结束，谢谢!
欢迎多提宝贵意见和建议

西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国 · 杨凌