* 我希望你没有刻意为追求一个数字而生活，我希望你找到了真正的价值所在
* 你一定要坚持做自己，静下心来做自己喜欢的事，然后把自己交给命运

@[toc]

# 多态

## 多态的基本概念

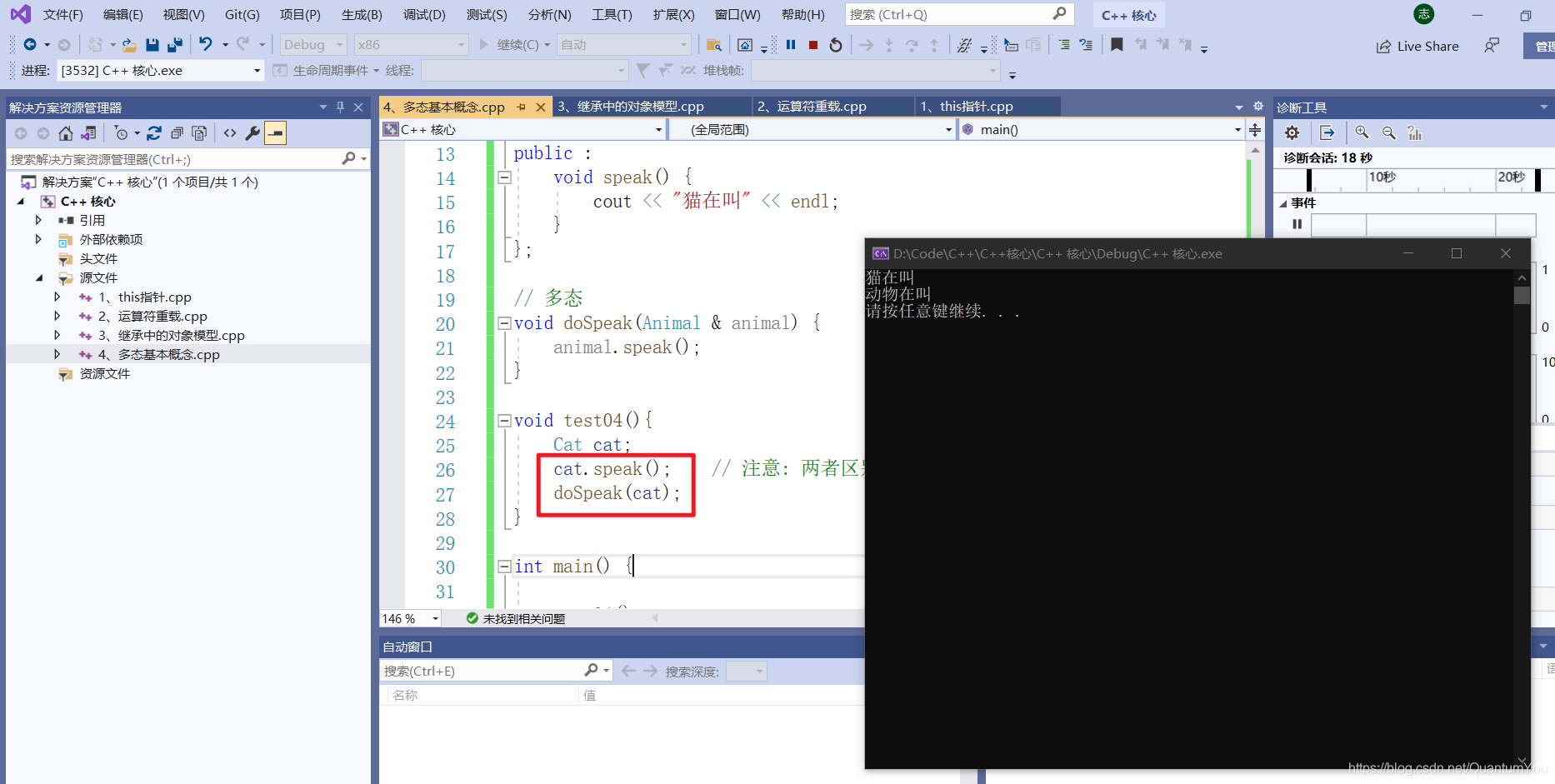
**多态是C++面向对象三大特性之一**

多态分为两类

* 静态多态: 函数重载 和 运算符重载属于静态多态，复用函数名
* 动态多态: 派生类和虚函数实现运行时多态

静态多态和动态多态区别：

* **静态多态的函数地址早绑定 - 编译阶段确定函数地址**
* **动态多态的函数地址晚绑定 - 运行阶段确定函数地址**

  
下面通过案例进行讲解多态

**注意引入虚函数的使用**

class Animal  
{  
public:  
 //Speak函数就是虚函数  
 //函数前面加上virtual关键字，变成虚函数，那么编译器在编译的时候就不能确定函数调用了。  
 virtual void speak()  
 {  
 cout << "动物在说话" << endl;  
 }  
};  
  
class Cat :public Animal  
{  
public:  
 void speak()  
 {  
 cout << "小猫在说话" << endl;  
 }  
};  
  
class Dog :public Animal  
{  
public:  
  
 void speak()  
 {  
 cout << "小狗在说话" << endl;  
 }  
  
};  
//我们希望传入什么对象，那么就调用什么对象的函数  
//如果函数地址在编译阶段就能确定，那么静态联编  
//如果函数地址在运行阶段才能确定，就是动态联编  
  
void DoSpeak(Animal & animal) // Animal & animal = cat   
{  
 animal.speak();  
}  
//  
//多态满足条件：   
//1、有继承关系  
//2、子类重写父类中的虚函数  
//多态使用：  
//父类指针或引用指向子类对象  
  
void test01()  
{  
 Cat cat;  
 DoSpeak(cat);  
  
  
 Dog dog;  
 DoSpeak(dog);  
}  
  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**执行说话的函数** : 地址早绑定在编译阶段确定函数地址, 如果想执行让猫说话，那么这个函数地址就不能提前绑定，需要在运行阶段进行绑定，地址晚绑

**总结**：  
**多态满足条件**

* 有继承关系
* 子类重写父类中的虚函数

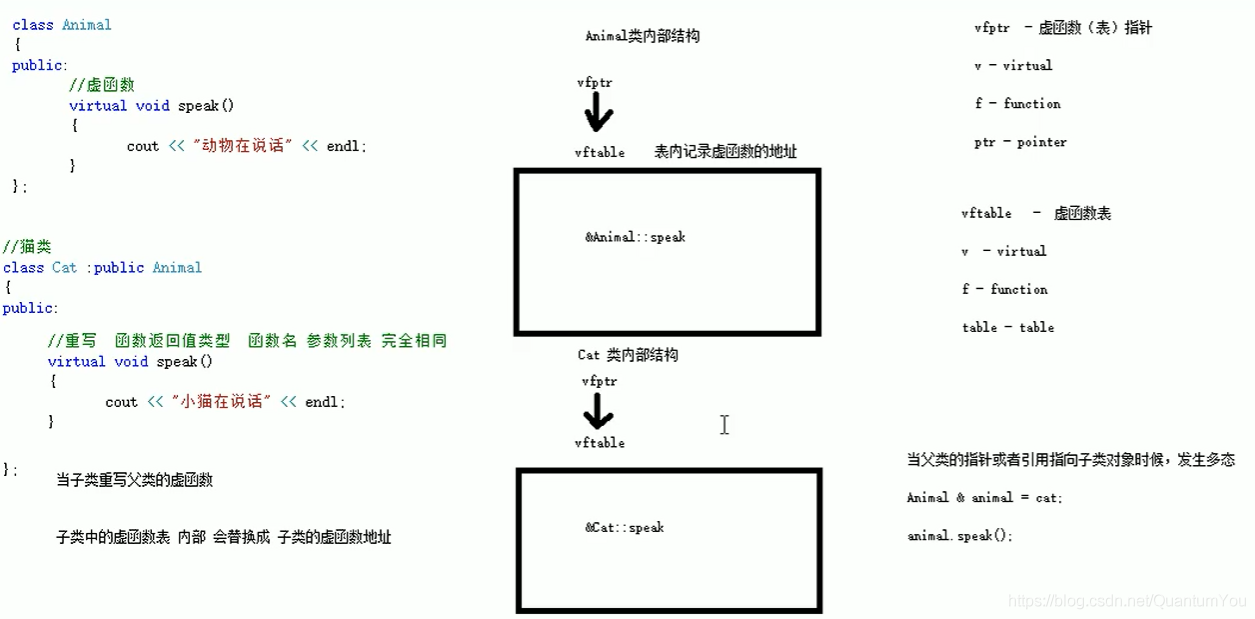
**多态使用条件**

* 父类指针或引用指向子类对象

重写：函数返回值类型 函数名 参数列表 完全一致称为重写

**Tips** : 32位windows内存地址采用平坦结构，内存地址都是32位整数，而指针变量的实质就是内存地址。所以，不管哪一种类型的指针变量都是4个字节大小。

## 多态原理分析



## 纯虚函数和抽象类

* 在多态中，通常父类中虚函数的实现是毫无意义的，主要都是调用子类重写的内容

因此可以将虚函数改为**纯虚函数**

纯虚函数语法：virtual 返回值类型 函数名 （参数列表）= 0 ;

当类中有了纯虚函数，这个类也称为抽象类

**抽象类特点**：

* 无法实例化对象
* 子类必须重写抽象类中的纯虚函数，否则也属于抽象类

**示例：**

class Base  
{  
public:  
 //纯虚函数  
 //类中只要有一个纯虚函数就称为抽象类  
 //抽象类无法实例化对象  
 //子类必须重写父类中的纯虚函数，否则也属于抽象类  
 virtual void func() = 0;  
};  
  
class Son :public Base  
{  
public:  
 virtual void func()   
 {  
 cout << "func调用" << endl;  
 };  
};  
  
void test01()  
{  
 Base \* base = NULL;  
 //base = new Base; // 错误，抽象类无法实例化对象  
 base = new Son;  
 base->func();  
 delete base;//记得销毁  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

1、实例化方式 Son son ; 2、堆栈方式 Base \* base = new Son ;

## 虚析构和纯虚析构

* 多态使用时，如果子类中有属性开辟到堆区，那么父类指针在释放时无法调用到子类的析构代码（导致内存泄露）,原因在于释放的是父类（父类指针在析构的时候不会调用子类中析构函数，导致子类如果有堆区属性，出现内存泄漏）

解决方式：将父类中的析构函数改为**虚析构**或者**纯虚析构**

**虚析构和纯虚析构共性**：

* 可以解决父类指针释放子类对象
* 都需要有具体的函数实现

**虚析构和纯虚析构区别**：

* 如果是纯虚析构，该类属于抽象类，无法实例化对象
* 纯虚析构也需要实现

**虚析构语法**：

virtual ~类名(){}

**纯虚析构语法**：

virtual ~类名() = 0;

类名::~类名(){}

**示例：**

class Animal {  
public:  
  
 Animal()  
 {  
 cout << "Animal 构造函数调用！" << endl;  
 }  
 virtual void Speak() = 0;  
  
 //析构函数加上virtual关键字，变成虚析构函数  
 //virtual ~Animal()  
 //{  
 // cout << "Animal虚析构函数调用！" << endl;  
 //}  
  
  
 virtual ~Animal() = 0;  
};  
  
Animal::~Animal()  
{  
 cout << "Animal 纯虚析构函数调用！" << endl;  
}  
  
//和包含普通纯虚函数的类一样，包含了纯虚析构函数的类也是一个抽象类。不能够被实例化。  
  
class Cat : public Animal {  
public:  
 Cat(string name)  
 {  
 cout << "Cat构造函数调用！" << endl;  
 m\_Name = new string(name);  
 }  
 virtual void Speak()  
 {  
 cout << \*m\_Name << "小猫在说话!" << endl;  
 }  
 ~Cat()  
 {  
 cout << "Cat析构函数调用!" << endl;  
 if (this->m\_Name != NULL) {  
 delete m\_Name;  
 m\_Name = NULL;  
 }  
 }  
  
public:  
 string \*m\_Name;  
};  
  
void test01()  
{  
 Animal \*animal = new Cat("Tom");  
 animal->Speak();  
  
 //通过父类指针去释放，会导致子类对象可能清理不干净，造成内存泄漏  
 //怎么解决？给基类增加一个虚析构函数  
 //虚析构函数就是用来解决通过父类指针释放子类对象  
 delete animal;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

Animal 的构造函数无 ？

总结：

1. 虚析构或纯虚析构就是用来解决通过父类指针释放子类对象

2. 如果子类中没有堆区数据，可以不写为虚析构或纯虚析构

3. 拥有纯虚析构函数的类也属于抽象类