C++多态--虚函数virtual及override

C++ 专栏收录该内容 3 订阅 39 篇文章 订阅专栏

C++多态

C++多态(polymorphism)是通过虚函数来实现的，虚函数允许子类重新定义成员函数，而子类重新定义父类的做法称为覆盖(override)，或者称为重写。

最常见的用法就是声明基类的指针，利用该指针指向任意一个子类对象，调用相应的虚函数，动态绑定。由于编写代码的时候并不能确定被调用的是基类的函数还是哪个派生类的函数，所以被成为"虚"函数。如果没有使用虚函数的话，即没有利用C++多态性，则利用基类指针调用相应的函数的时候，将总被限制在基类函数本身，而无法调用到子类中被重写过的函数。

用下面代码演示多态和非多态。

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

class A
{
public:
    void foo()
    {
        printf("1\n");
    }
    virtual void fun()
    {
        printf("2\n");
    }
};
class B : public A
{
public:
    void foo()  //隐藏：派生类的函数屏蔽了与其同名的基类函数
    {
        printf("3\n");
    }
    void fun()  //多态、覆盖
    {
        printf("4\n");
    }
};
int main(void)
{
    A a;
    B b;
    A *p = &a;
    p->foo();  //输出1
    p->fun();  //输出2
    p = &b;
    p->foo();  //取决于指针类型，输出1
    p->fun();  //取决于对象类型，输出4，体现了多态
    return 0;
}
```

C++纯虚函数及虚函数

纯虚函数是在基类中声明的虚函数，它在基类中没有定义，但要求任何派生类都要定义自己的实现方法。在基类中实现纯虚函数的方法是在函数原型后加“=0”。

引入纯虚函数的原因：

（1）为了方便使用多态特性，我们常常需要在基类中定义虚拟函数。

（2）在很多情况下，基类本身生成对象是不合情理的。例如，动物作为一个基类可以派生出老虎、孔雀等子类，但动物本身生成对象明显不合常理。

包含纯虚函数的类称为**抽象类**。由于抽象类包含了没有定义的纯虚函数，所以不能定义抽象类的对象。抽象类的主要作用是将有关的操作作为结果接口组织在一个继承层次结构中，由它来为派生类提供一个公共的根，派生类将具体实现在其基类中作为接口的操作。

虚函数的作用是允许在派生类中重新定义与基类同名的函数，并且可以通过**基类**指针或引用来访问基类和派生类中的同名函数。

虚函数是C++中用于实现多态的机制。核心理念就是通过基类访问派生类定义的函数。如果父类或者祖先类中函数func()为虚函数，则子类及后代类中，函数func()是否加virtual关键字，都将是虚函数。为了提高程序的可读性，建议后代中虚函数都加上virtual关键字。

C++保留字override

override 仅在成员函数声明之后使用时才是区分上下文的且具有特殊含义；否则，它不是保留的关键字。使用 override 有助于防止代码中出现意外的继承行为。以下示例演示在未使用override 的情况下，可能不打算使用派生类的成员函数行为。编译器不会发出此代码的任何错误。

```
class BaseClass
{
  virtual void funcA();
  virtual void funcB() const;
  virtual void funcC(int = 0);
  void funcD();
};

class DerivedClass: public BaseClass
{
  virtual void funcA(); // ok, works as intended

  virtual void funcB(); // DerivedClass::funcB() is non-const, so it does not
                        // override BaseClass::funcB() const and it is a new member function

  virtual void funcC(double = 0.0); // DerivedClass::funcC(double) has a different
                                    // parameter type than BaseClass::funcC(int), so
                                    // DerivedClass::funcC(double) is a new member function
};
```

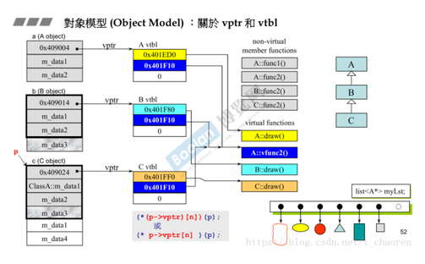
当使用 override时，编译器会生成错误，而不会在不提示的情况下创建新的成员函数。

```
class BaseClass
{
  virtual void funcA();
  virtual void funcB() const;
  virtual void funcC(int = 0);
  void funcD();
};

class DerivedClass: public BaseClass
{
  virtual void funcA() override; // ok

virtual void funcB() override; // compiler error: DerivedClass::funcB() does not
                  // override BaseClass::funcB() const
  virtual void funcC( double = 0.0 ) override; // compiler error:
                        // DerivedClass::funcC(double) does not
                        // override BaseClass::funcC(int)

  void funcD() override; // compiler error: DerivedClass::funcD() does not
              // override the non-virtual BaseClass::funcD()
};
```



下面代码展示了手动调用虚函数的过程

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

class A {
public:
    virtual void vfunc1() { cout << "A::vfunc1()" << endl; };
    virtual void vfunc2() { cout << "A::vfunc2()" << endl; };
    void func1() { cout << "A::func1()" << endl; };
    void func2() { cout << "A::func2()" << endl; };
private:
    int data1_;
    int data2_;
};

class B :public A {
public:
    virtual void vfunc1() override { cout << "B::vfunc1()" << endl; };
    void func2() { cout << "B::func2()" << endl; };
private:
    int data3_;
};

class C :public B {
public:
    virtual void vfunc1() override { cout << "C::vfunc1()" << endl; };
    void func2() { cout << "C::func2()" << endl; };
private:
    int data1_, data4_;
};

//演示了手动调用虚函数的过程
int main() {
    B a;
    typedef void(*Fun)(void);
    Fun pFun = nullptr;
    cout << "虚函数表地址：" << (int*)(&a) << endl;
    cout << "虚函数表第1个函数地址："<<(int*)*(int*)(&a) << endl;
    cout << "虚函数表第2个函数地址：" << (int*)*(int*)(&a) + 1 << endl;
    pFun = (Fun)*((int*)*(int*)(&a));
    pFun();
    pFun = (Fun)*((int*)*(int*)(&a) + 1);
    pFun();
    return 0;
}
```


参考文章：

[1] http://blog.csdn.net/hackbuteer1/article/details/7475622

[2] http://blog.csdn.net/hackbuteer1/article/details/7558868

[3] http://www.jb51.net/article/78489.htm

C++11新特性之虚函数的override指示符 公众号：君子黎 1873
1. C++11中新增加了许多新的特性，这里对override这个新特性做一个简单的描述；override的英文意思为"覆...

评论 8

请发表有价值的评论，博客评论不欢迎灌水，良好的社区氛围需大家一起维护 评论

ShadowFlowYan 2022.05.10
很清晰，多谢博主

daydayup_666 2021.01.30
(Fun)*((int*)*(int*)(&a)) 请教一下：为什么这儿写成pFun = (Fun)(int*)(&a); 这样就不行

sinth 回复 daydayup_666 2021.02.09
这样强制转换，首先你得有子类的声明。而虚函数，没有子类的声明，也能且仅能执行子类的函数。

weixin_46005091 2020.12.02
class BaseClass
{
virtual void funcA();
virtual void funcB() const;
virtual void funcC(int = 0);
void funcD();
};

class DerivedClass: public BaseClass
{
virtual void funcA() override; // ok

virtual void funcB() override; // compiler error: DerivedClass::funcB() does not
// override BaseClass::funcB() const

virtual void funcC(double = 0.0) override; // compiler error:
// DerivedClass::funcC(double) does not
// override BaseClass::funcC(int)

void funcD() override; // compiler error: DerivedClass::funcD() does not
// override the non-virtual BaseClass::funcD()
};

请问最后这个重写为什么会报错呢？应该是正确的吧，希望讲解一下

< 1 2 3 >

C++ 纯虚函数 虚函数 override_软件工程小施同学的博客... 3-29
C++ 纯虚函数 虚函数 override override在C++中它是覆盖了一个方法并且对其重写,从而达到不同的作用。在...

C++11新特性之虚函数的override指示符_君子黎的博客

C++ 多态: 虚函数, 纯虚函数 最最详细

C++ override使用详解

...final看这一篇_陆六宗的博客_c++虚函数 override

【c++】虚函数描述符override_wall的博客_虚函数override

C#里virtual和override

C++ override及其虚函数的详解

C++11显式虚函数重载override与final_TOMACAT的博客

c++虚函数override的(和重载函数overload的比较_简单...

C++中virtual和override的作用

c++的virtual和override作用

virtual和override作用和使用场景及理解

C++中override和overwrite的探究

虚函数、纯虚函数、override

c++虚函数详解（你肯定懂了）

C++ 纯虚函数 虚函数 override

virtual与override

C++override关键词和virtual在C#(应用的区别

c# virtual和override的多态用法

virtual override

管理虚函数的一些小工具（override）（final）

在虚函数声明中写override的作用

C# virtual和override

C++学习（一百二）virtual和override

“相关推荐”对你有帮助么？
非常没帮助 没帮助 一般 有帮助 非常有帮助

©2022 CSDN 皮肤主题: 大白 设计师: CSDN官方博客 返回首页