链滴

C++ 的虚函数与多态性

作者：heyang5188
原文链接：https://ld246.com/article/1504868546047
来源网站：链滴

C++多态性

oop的核心思想是多态性，多态性代表多种形式

我们把具体有继承关系的多个类型称为多态性，因为我们能使用这些类型的多态性而无需在意他们的异

引用和指针的静态类型和动态类型的不同正是C++语言支持态性的根本所在

只有当通过函数指针的或者引用调用函数时，才会在运行时候解析该运用，也只有在这种情况下，动类型和静态类型才会不相同

静态类系和动态类型

● 静态类型和动态类型的区别是，静态类型是在编译的时候是已知的，他是变量声明时的类型或表达生成的类型；

动态类型是变量或者表达式表示的内存中的对象类型，动态类型直到运行的时候才知道。

● 只有指针和引用调用虚函数的时候，才会根据这个引用和指针绑定的不同对象来调用不用的虚函数

那么问题来了什么样的函数称为虚函数呢?

在C++语言中，在基类的函数声明上添加 virtual 声明符就可以定义一个虚函数

<code>在C++中，基类希望它的派生类都各自定义适合自的版本的函数的话，就应该将这个函数声明称为虚函数</cod>

派生类对象及派生类向基类类型的转换

假如说，我们定义了一个基类的名字叫人 ,他的派生类的名字叫男人和女人

男人照镜子非常快，但是女人就不一样了，所以这里就体现了多态性

```
class man {
public:
    man()=default;
    virtual ~man();
    virtual void lookatmirror() {
        /*
        */
    }
};
class boy : public man {
public:
    boy() = default;
    virtual ~boy();
    virtual void lookamirror()override {
    //very fast;
    }
};
```

原文链接：c++ 的虚函数与多态性

```
class girl : public man {
public:
    girl() = default;
    virtual ~girl();
    virtual void lookmirror() override{
        sleep(10000);  //看个10000秒
    }
};
```

比如说我们现在创建了一个指向基类的指针，

```
 man * goodman = new man;
 boy herb();
 goodman = & herb; //动态绑定，虽然goodman 的类型是 指向基类的指针，但是这里绑定了一个
```

态类型；

```
 goodman->lookatmirror();//执行的是boy的 虚函数；
```

如果这里绑定的类型是另外一个基类，那么就会执行另外一个派生类的函数。

我们对象在内存中的可以理解成两个部分，一部分是从基类继承而来的，一部分派生类自己定义的，

一个派生类可以在需要的时候转换称为基类，转换的同时就会把属于自己那部分给切掉

而基类不能转换称为一个派生类

。

```
double printtotle(std::ostream & os, const Quote & item, std::size_t t)
{
    double ret = item.net_price(t);
    os << "ISBN: " <<item.isbn()<<" number: " <<t << " total due:" << ret << std::endl;
    return ret;
}
```

定义一个基类

```
class Quote {
    friend double print_totle(std::ostream& os, const Quote &item, std::size_t t);
public :
    Quote() = default;
    Quote(const std::string &book, double sales_price):bookNo(book),price(sales_price){}
    std::string isbn() const { return bookNo; }
    Quote& operator=(const Quote&rhs);
    //return sales_data
    virtual double net_price(std::size_t n) const //虚函数
    {
        return n*price;
    }
    virtual void debug() const;
    virtual ~Quote() = default; //基类通常都需要定义一个析构函数，即使该函数不执行任何实际操
```

也是如此

```
private:
    std::string bookNo;
protected:
    double price = 0.0;
```

```
};
```