

昵称：拾点阳光

园龄：3年7个月

粉丝：18

关注：20

2018年7月						
日	一	二	三	四	五	六
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

- 最新随笔
1. NumPy Nddarray对象

2. Numpy介绍与安装

3. 游戏中的人物是如何寻路的？

4. c++ 平衡二叉树的实现

5. python解析RSS

6. Jupyter Notebook

7. 编程英语之KNN算法

8. 成为伟大程序员的 10 个要点

9. spring mvc

10. sql 复习练习

- 我的标签
- c++(2)

python3(2)

java 面向对象(1)

linux(1)

- 随笔分类(81)
- android

C#(9)

c++基础(22)

git指南

Java基础(26)

python3(5)

Spring(4)

Spring MVC(1)

编程英语(2)

编程之道(5)

机器学习(3)

数据库(4)

c++ 深入理解虚函数

为什么使用虚函数？ 什么是虚函数？ 虚函数是为了解决什么问题？

面向对象的三大特征：

- 封装
 - 1. 普通虚函数
 - 2. 虚析构函数
 - 3. 纯虚函数
 - 4. 抽象类
 - 5. 接口类
 - 6. 隐藏 vs 覆盖
 - 7. 隐藏与覆盖之间的关系
 - 8. 早绑定和晚绑定
 - 9. 虚函数表
- 多态
- 继承

什么是多态？

相同对象收到不同消息或不同对象收到相同消息时产生的不同的动作。

静态多态 vs 动态多态


[:->静态多态也叫做早绑定




```
class Rect //矩形类
{
public:
    int calcArea(int width);
    int calcArea(int width,int height);
};
```



如上面的代码，他们函数名相同，参数个数不同，一看就是互为重载的两个函数





```
1 int main()
2 {
3     Rect.rect;
```

2018年5月 (4)

2018年4月 (4)

2017年8月 (3)

2017年6月 (1)

2017年2月 (1)

2016年11月 (1)

2016年10月 (20)

2016年9月 (2)

2016年8月 (14)

2016年7月 (6)

2016年6月 (18)

2016年3月 (1)

文章分类

机器学习

相册(3)

岁月流金(3)

积分与排名

积分 - 28944

排名 - 14792

最新评论

阅读排行榜

评论排行榜

推荐排行榜

```
4 rect.calcArea(10);
5 rect.calcArea(10,20);
6 return 0;
7 }
```

程序在编译阶段根据参数个数确定调用哪个函数。这种情况叫做静态多态（早绑定）

[-:>动态多态也叫做晚绑定

比如计算面积 当给圆形计算面积时使用圆形面积的计算公式，给矩形计算面积时使用矩形面积的计算公式。也就是说有一个计算面积的形状基类，圆形和矩形类派生自形状类，圆形与矩形的类各有自己的计算面积的方法。可见动态多态是以封装和继承为基础的。

```
1 class Shape//形状类
2 {
3 public:
4     double calcArea()
5     {
6         cout<<"calcArea"<<endl;
7         return 0;
8     }
9 };
10 class Circle:public Shape //公有继承自形状类的圆形类
11 {
12 public:
13     Circle(double r);
14     double calcArea();
15 private:
16     double m_dR;
17 };
18 double Circle::calcArea()
19 {
20     return 3.14*m_dR*m_dR;
21 }
22 class Rect:public Shape //公有继承自形状类的矩形类
23 {
24 public:
25     Rect(double width,double height);
26     double calArea();
27 private:
28     double m_dWidth;
29     double m_dHeight;
30 };
31 double Rect::calcArea()
32 {
33     return m_dWidth*m_dHeight;
34 }
35 int main()
36 {
37     Shape *shapel=new Circle(4.0);
38     Shape *shape2=new Rect(3.0,5.0);
39     shapel->calcArea();
40     shape2->calcArea();
41     .....
```

```
42     return 0;
43 }
```

如果打印结果的话，以上程序结果会打印两行"calcArea"，因为调用到的都是父类的calcArea函数，并不是我们想要的那样去分别调用各自的计算面积的函数。如果要想**实现动态多态则必须使用虚函数**

关键字 **virtual** ->虚函数

用virtual去修饰成员函数使其成为虚函数

所以以上函数的修改部分如下

```
class Shape
{
public:
    virtual double calcArea() {...} //虚函数
    ....
private:
    ....
};
....
class Circle:public Shape
{
public:
    Circle(double r);
    virtual double calcArea(); //此处的virtual不是必须的，如果不加，系统会自动加
    ....
private:
    ....
};
....
class Rect:public Shape
{
    Rect(double width, double height);
    virtual double calcArea();
private:
    ....
};
....
```

这样就可以达到预期的结果了

多态中存在的问题

[->内存泄漏，一个很严重的问题

例如上面的程序中，如果在圆形的类中定义一个圆心的坐标，并且坐标是在堆中申请的内存，则在main函数中通过父类指针操作子类对象的成员函数的时候是没有问题的，可是在销毁对象内存的时候则只是执行了

父类的析构函数，子类的析构函数却没有执行，这会导致内存泄漏。部分代码如下(想去借助父类指针去销毁子类对象的时候去不能去销毁子类对象)

如果delete后边跟父类的指针则只会执行父类的析构函数，如果delete后面跟的是子类的指针，那么它即会执行子类的析构函数，也会执行父类的析构函数





```
class Circle:public Shape
{
public:
    Circle(int x,int y,double r);
    ~Circle();
    virtual double calcArea();
    ....
private:
    double m_dR;
    Coordinate *m_pCenter;    //坐标类指针
    ....
};
Circle::Circle(int x,int y,double r)
{
    m_pCenter=new Coordinate(x,y);
    m_dR=r;
}
Circle::~~Circle()
{
    delete m_pCenter;
    m_pCenter=NULL;
}
....
int main()
{
    Shape *shapel=new Circle(3,5,4.0);
    shapel->calcArea();
    delete shapel;
    shapel=NULL;
    return 0;
}










```


可见我们必须要去解决这个问题，不解决这个问题当使用的时候都会造成内存泄漏。面对这种情况则需要引入虚析构函数

虚析构函数

关键字 **virtual** ->析构函数

之前是使用virtual去修饰成员函数，这里使用virtual去修饰析构函数，部分代码如下





```
1 class Shape
2 {
```

```
3 public:
4     ....
5     virtual ~Shape();
6 private:
7     ....
8 };
9 class Circle:public Shape
10 {
11 public:
12     virtual ~Circle();//与虚函数相同, 此处virtual可以不写, 系统将会自动添加, 建议写上
13     ....
14 };
15 ....
```



这样父类指针指向的是哪个对象，哪个对象的构造函数就会先执行，然后执行父类的构造函数。销毁的时候子类的析构函数也会执行。

virtual关键字可以修饰普通的成员函数，也可以修饰析构函数，但并不是没有限制

virtual在函数中的使用限制

- 普通函数不能是虚函数，也就是说这个函数必须是某一个类的成员函数，不可以是一个全局函数，否则会导致编译错误。
- 静态成员函数不能是虚函数 static成员函数是和类同生共处的，他不属于任何对象，使用virtual也将导致错误。
- 内联函数不能是虚函数 如果修饰内联函数 如果内联函数被virtual修饰，计算机会忽略inline使它变成存粹的虚函数。
- 构造函数不能是虚函数，否则会出现编译错误。

虚函数实现原理

【：-» 首先：什么是函数指针？

指针指向对象称为对象指针，指针除了指向对象还可以指向函数，函数的本质就是一段二进制代码，我们可以通过指针指向这段代码的开头，计算机就会从这个开头一直往下执行，直到函数结束，并且通过指令返回回来。函数的指针与普通的指针本质上是一样的，也是由四个基本的内存单元组成，存储着内存的地址，这个地址就是函数的首地址。

【：-» 多态的实现原理

虚函数表指针：类中除了定义的函数成员，还有一个成员是虚函数表指针（占四个基本内存单元），这个指针指向一个虚函数表的起始位置，这个表会与类的定义同时出现，这个表存放着该类的虚函数指针，调用的时候可以找到该类的虚函数表指针，通过虚函数表指针找到虚函数表，通过虚函数表的偏移找到函数的入口地址，从而找到要使用的虚函数。

当实例化一个该类的子类对象的时候，（如果）该类的子类并没有定义虚函数，但是却从父类中继承了虚函数，所以在实例化该类子类对

象的时候也会产生一个虚函数表，这个虚函数表是子类的虚函数表，但是记录的子类的虚函数地址却是与父类的是一样的。所以通过子类对象的虚函数表指针找到自己的虚函数表，在自己的虚函数表找到的要执行的函数指针也是父类的相应函数入口的地址。

如果我们在子类中定义了从父类继承来的虚函数，对于父类来说情况是不变的，对于子类来说它的虚函数表与之前的虚函数表是一样的，但是此时子类定义了自己的（从父类那继承来的）相应函数，所以它的虚函数表当中管于这个函数的指针就会覆盖掉原有的指向父类函数的指针的值，换句话说就是指向了自己定义的相应函数，这样如果用父类的指针，指向子类的对象，就会通过子类对象当中的虚函数表指针找到子类的虚函数表，从而通过子类的虚函数表找到子类的相应虚函数地址，而此时的地址已经是该函数自己定义的虚函数入口地址，而不是父类的相应虚函数入口地址，所以执行的将会是子类当中的虚函数。这就是多态的原理。

函数的覆盖和隐藏

父类和子类出现同名函数称为隐藏。

- 父类对象.函数函数名(...); //调用父类的函数
- 子类对象.函数名(...); //调用子类的函数
- 子类对象.父类名::函数名(...);//子类调用从父类继承来的函数。

父类和子类出现同名虚函数称为覆盖

- 父类指针=new 子类名(...);父类指针->函数名(...);//调用子类的虚函数。

虚析构函数的实现原理

[:->虚析构函数的特点：

- 当我们在父类中通过**virtual**修饰析构函数之后，通过父类指针指向子类对象，通过**delete**接父类指针就可以释放掉子类对象

[:->理论前提：

- 执行完子类的析构函数就会执行父类的析构函数

原理：

如果父类当中定义了虚析构函数，那么父类的虚函数表当中就会有一个父类的虚析构函数的入口指针，指向的是父类的虚析构函数，子类虚函数表当中也会产生一个子类的虚析构函数的入口指针，指向的是子类的虚析构函数，这个时候使用父类的指针指向子类的对象，delete接父类指针，就会通过指向的子类的对象找到子类的虚函数表指针，从而找到虚函数表，再虚函数表中找到子类的虚析构函数，从而使得子类的析构函数得以执行，子类的析构函数执行之后系统会自动执行父类的虚析构函数。这个是虚析构函数的实现原理。

纯虚函数：

纯虚函数的定义

```
1 class Shape
2 {
3 public:
4     virtual double calcArea() //虚函数
5     {...}
6     virtual double calcPerimeter()=0; //纯虚函数
7     ....
8 };
```

纯虚函数没有函数体，同时在定义的时候函数名后面要加“=0”。

纯虚函数的实现原理：

在虚函数原理的基础上，虚函数表中，虚函数的地址是一个有意义的值，如果是纯虚函数就实实在在的写一个0。

含有纯虚函数的类被称为抽象类

含有纯虚函数的类被称为抽象类，比如上面代码中的类就是一个抽象类，包含一个计算周长的纯虚函数。哪怕只有一个纯虚函数，那么这个类也是一个抽象类，纯虚函数没有函数体，所以抽象类不允许实例化对象，抽象类的子类也可以是一个抽象类。抽象类子类只有把抽象类当中的所有的纯虚函数都做了实现才可以实例化对象。

对于抽象的类来说，我们往往不希望它能实例化，因为实例化之后也没什么用，而对于一些具体的类来说，我们要求必须实现那些要求（纯虚函数），使之成为有具体动作的类。

近含有纯虚函数的类称为接口类

如果在抽象类当中仅含有纯虚函数而不含其他任何东西，我们称之为接口类。

1. 没有任何数据成员
2. 仅有成员函数
3. 成员函数都是纯虚函数

```
class Shape
{
    virtual double calcArea()=0//计算面积
    virtual double calcPerimeter()=0//计算周长
};
```

实际的工作中接口类更多的表达一种能力或协议

比如

```
1 class Flyable//会飞
2 {
3 public:
4     virtual void takeoff()=0;//起飞
5     virtual void land()=0;//降落
6 };
7 class Bird:public Flyable
8 {
9 public:
10     ....
11     virtual void takeoff(){....}
12     virtual void land(){....}
13 private:
14     ....
15 };
16 void flyMatch(Flyable *a,Flyable *b)//飞行比赛
17 //要求传入一个会飞对象的指针，此时鸟类的对象指针可以传入进来
18 {
19     ....
20     a->takeoff();
21     b->takeoff();
22     a->land();
23     b->land();
24 }
```

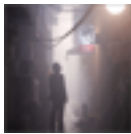
例如上面的代码，定义一个会飞的接口，凡是实现这个接口的都是会飞的，飞行比赛要求会飞的来参加，鸟实现了会飞的接口，所以鸟可以参加飞行比赛，如果复杂点定义一个能够射击的接口，那么实现射击接口的类就可以参加战争之类需要会射击的对象，有一个战斗机类通过多继承实现会飞的接口和射击的接口还可以参加空中作战的函数呢

分类: [c++基础](#)

好文要顶

关注我

收藏该文



拾点阳光
关注 - 20
粉丝 - 18

[+加关注](#)

13
推荐

0
反对

« 上一篇: [c++ 深拷贝与浅拷贝](#)
» 下一篇: [总会有一个是你需要的](#)

posted @ 2016-06-20 23:30 拾点阳光 阅读(43277) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

- 【推荐】超50万VC++源码：大型组态工控、电力仿真CAD与GIS源码库！
- 【福利】校园拼团福利，腾讯云1核2G云服务器10元/月！
- 【大赛】2018首届“顶天立地”AI开发者大赛



新注册用户域名抢购1元起

.com首年28元 .cn首年19元

立即抢购



最新IT新闻：

- 云计算亿万大单频现 今年市场怎么了
 - WhatsApp向使用原生Android API的应用发去制止信
 - 广州共从河道里打捞出超3000辆共享单车
 - 新研发屏下指纹识别传感器将会检测体温 提升识别安全性
 - 阿联酋航空推出VR体验让用户在飞机上闲逛
- » 更多新闻...



40+ 产品 免费用6个月

广告



40+ 产品 免费用6个月

广告

最新知识库文章：

- 从Excel到微服务
 - 如何提升你的能力？给年轻程序员的几条建议
 - 程序员的那些反模式
 - 程序员的宇宙时间线
 - 突破程序员思维
- » 更多知识库文章...