

昵称：CoderCong

园龄：2年2个月

粉丝：14

关注：2

+加关注

< 2018年4月 >						
日	一	二	三	四	五	六
25	26	27	28	29	30	31
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	1	2	3	4	5

搜索

找找看

谷歌搜索

- 常用链接
- 我的随笔

我的评论

我的参与

最新评论

我的标签

- 我的标签
- C++(2)

纯虚析构函数(1)

多态(1)

构造函数(1)

计算机网络、TCP(1)

派生(1)

析构函数(1)

虚函数(1)

虚函数表(1)

虚析构函数(1)

更多

## C++中的多态与虚函数的内部实现

### 1、什么是多态

多态性可以简单概括为“一个接口，多种行为”。  
也就是说，向不同的对象发送同一个消息，不同的对象在接收时会产生不同的行为（即方法）。  
也就是说，每个对象可以用自己的方式去响应共同的消息。所谓消息，就是调用函数，不同的行为就是指不同的实现，即执行不同的函数。这是一种泛型技术，即用相同的代码实现不同的动作。这体现了面向对象编程的优越性。

- 多态分为两种：
- (1) 编译时多态：主要通过函数的重载和模板来实现。
  - (2) 运行时多态：主要通过虚函数来实现。

### 2、几个相关概念

- (1) 覆盖、重写（override）  
override指基类的某个成员函数为虚函数，派生类又定义一成员函数，除函数体的其余部分都与基类的成员函数相同。注意，如果只是函数名相同，形参或返回类型不同的话，就不能称为override，而是hide。
- (2) 重载（overload）  
指同一个作用域出生多个函数名相同，但是形参不同的函数。编译器在编译的时候，通过实参的个数和类型，选择最终调用的函数。
- (3) 隐藏（hide）  
分为两种：
  - 1) 局部变量或者函数隐藏了全局变量或者函数
  - 2) 派生类拥有和基类同名的成员函数或成员变量。产生的结果：使全局或基类的变量、函数不可见。

### 3、几个简单的例子

```
1
/*****
2 * File:PolymorphismTest
3 * Introduction:测试多态的一些特性。
4 * Author:CoderCong
5 * Date:20141114
6 * LastModifiedDate:20160113
7
*****/
8 #include "stdafx.h"
9 #include <iostream>
10 using namespace std;
11 class A
12 {
13 public:
14     void foo()
15     {
16         printf("1\n");
17     }
18     virtual void fun()
19     {
20         printf("2\n");
21     }
22 };
23 class B : public A
24 {
25 public:
26     void foo() //由于基类的foo函数并不是虚函数，所以是隐藏，而不是重写
27     {
28         printf("3\n");
29     }
30     void fun() //重写
31     {
32         printf("4\n");
```

随笔分类
C/C++(19)
操作系统(6)
计算机基础(13)
计算机网络(15)
技术前沿
设计模式(3)
生产实践(1)
数据结构与算法(5)
数据库(2)
网络安全(1)
知识点汇总(26)
转载(5)

随笔档案
2017年8月 (1)
2016年10月 (1)
2016年9月 (4)
2016年8月 (8)
2016年7月 (3)
2016年6月 (6)
2016年5月 (5)
2016年4月 (8)
2016年3月 (10)
2016年1月 (2)

文章分类
C/C++
操作系统(1)
计算机网络
数据结构与算法(1)
数据库

最新评论
------

```
33     }
34 };
35 int main(void)
36 {
37     A a;
38     B b;
39     A *p = &a;
40     p->foo(); //输出1。
41     p->fun(); //输出2。
42     p = &b;
43     p->foo(); //输出1。因为p是基类指针，p->foo指向一个具有固定偏移量的函数。也就是基类函数
44     p->fun(); //输出4。多态。虽然p是基类指针，但实际上指向的是一个子类对象。p->fun指向的是一个虚函数。按照动态类型，调用子类函数
45     return 0;
46 }
```

4、运行时多态以及虚函数的内部实现

看了上边几个简单的例子，我恍然大悟，原来这就是多态，这么简单，明白啦！好，那我们再看一个例子：

```
1 class A
2 {
3 public:
4     virtual void FunA()
5     {
6         cout << "FunA1" << endl;
7     };
8     virtual void FunAA()
9     {
10         cout << "FunA2" << endl;
11     }
12 };
13 class B
14 {
15 public:
16     virtual void FunB()
17     {
18         cout << "FunB" << endl;
19     }
20 };
21 class C :public A, public B
22 {
23 public:
24     virtual void FunA()
25     {
26         cout << "FunA1C" << endl;
27     };
28 };

31 int _tmain(int argc, _TCHAR* argv[])
32 {
33     C objC;
34     A *pA = &objC;
35     B *pB = &objC;
36     C *pC = &objC;
37
38     printf("%d %d\n", &objC, objC);
39     printf("%d %d\n", pA, *pA);
40     printf("%d %d\n", pB, *pB);
41     printf("%d %d\n", pC, *pC);
42
43     return 0;
44 }
```

运行结果：  
5241376 1563032

1. Re:TCP的粘包现象
感谢楼主的总结
--Jarhead
2. Re:简单工厂、工厂方法、抽象工厂之小结、区别
可以
--frankchao1005
3. Re:C++ STL 迭代器失效问题
学习了
--左影尘
4. Re:进程/线程要点
楼主 (3) 若该进程还有子进程，则应将其所有的子进程终止 —— 子进程不是终止，而是其父进程变成init进程吧？
--Anmi
5. Re:C++的单例模式与线程安全单例模式（懒汉/饿汉）
楼主，singleton(){ pthread_mutex_init(&mutex);}mutex在构造中初始化singleton* singleton::initance(){ pthread.....
--jiemo1123

阅读排行榜
1. C++的单例模式与线程安全单例模式（懒汉/饿汉）(25816)
2. windows下MySQL 5.7+ 解压缩版安装配置方法(12630)
3. TCP的粘包现象(9596)
4. git与github安装、配置、pull、push(4484)
5. IPv4中IP地址分类(2533)

评论排行榜
1. C++的单例模式与线程安全单例模式（懒汉/饿汉）(10)
2. 简单工厂、工厂方法、抽象工厂之小结、区别(2)
3. TCP的粘包现象(2)

5241376 1563032

5241380 1563256

5241376 1563032

细心的同志一定发现了pB出了问题，为什么明明都是指向objC的指针，pB跟别人的值都不一样呢？是不是编译器出了问题呢？

当然不是！我们先讲结论：

- (1) 每一个含有虚函数的类，都会生成虚表(virtual table)。这个表，记录了对象的动态类型，决定了执行此对象的虚成员函数的时候，真正执行的那一个成员函数。
- (2) 对于有多个基类的类对象，会有多个虚表，每一个基类对应一个虚表，同时，虚表的顺序和继承时的顺序相同。
- (3) 在每一个类对象所占用的内存中，虚指针位于最前边，每个虚指针指向对应的虚表。

先从简单的单个基类说起：

```
1 class A
2 {
3 public:
4     virtual void FunA ()
5     {
6         cout << "FunA1" << endl;
7     }
8     virtual void FunA2 ()
9     {
10         cout << "FunA2" << endl;
11     }
12 };
13
14 class C :public A
15 {
16     virtual void FunA ()
17     {
18         cout << "FunA1C" << endl;
19     }
20 };
21 int _tmain(int argc, _TCHAR* argv[])
22 {
23     A *pA = new A;
24     C *pC = new C;
25     typedef void (*Fun) (void);
26
27     Fun fun= (Fun)*((int*)(*(int*)pA));
28     fun(); //pA指向的第一个函数
29     fun = (Fun)*((int*)(*(int*)pA) +1);
30     fun(); //pA指向的第二个函数
31
32     fun = (Fun)*((int*)(*(int*)pC));
33     fun(); //pC指向的第一个函数
34     fun = (Fun)*((int*)(*(int*)pC) + 1);
35     fun(); //pC指向的第二个函数
36     return 0;
37 }
```

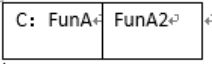
运行结果：

FunA1  
FunA2  
FunA1C  
FunA2

是不是有点晕？没关系。我一点一点解释：pA对应一个A的对象，我们可以画出这样的表：

A: FunA	FunA2
---------	-------

这就是对象\*pA的虚表，两个虚函数以声明顺序排列。pA指向对象\*pA，则\*(int\*)pA指向此虚拟表，则(Fun)\*((int\*)(\*(int\*)pA))指向FunA，同理，(Fun)\*((int\*)(\*(int\*)pA) + 1)指向FunA2。所以，出现了前两个结果。根据后两个结果，我们可以推测\*pC的虚表如下图所示：



也就是说，由于C中的FunA重写(override)了A中的FunA，虚拟表中虚拟函数的地址也被重写了。就是这样，这就是多态实现的内部机制。

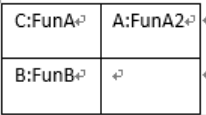
我们再回到最初的问题：为什么\*pB出了问题。  
根据上边的结论，我们大胆地进行猜测：由于C是由A、B派生而来，所以objC有两个虚拟表，而由于表的顺序，pA、pC都指向了对应于A的虚拟表，而pB则指向了对应于B的虚拟表。做个实验来验证我们的猜想是否正确：  
我们不改变A、B、C类，将问题中的main改一下：

```
1 int _tmain(int argc, _TCHAR* argv[])
2 {
3     C objC;
4     A *pA = &objA;
5     B *pB = &objC;
6     C *pC = &objC;
7
8     typedef void (*Fun)(void);
9
10    Fun fun = (Fun)*((int*)(*(int*)pC));
11    fun(); //第一个表第一个函数
12    fun = (Fun)*((int*)(*(int*)pC)+1);
13    fun(); //第一个表第二个函数
14    fun = (Fun)*((int*)(*(int*)pC+1));
15    fun();> //第二个表第一个函数
16    fun = (Fun)*((int*)(*(int*)pB));
17    fun(); //pB指向的表的第一个函数
18    return 0;
19 }
```

哈哈，和我们的猜测完全一致：

FunA1C  
FunA2  
FunB  
FunB

我们可以画出这样的虚函数图：



暂且这样理解，编译器执行B \*pB = &objC时不是仅仅是赋值，而是做了相应的优化，将pB指向了第二张虚表。

说了这么多，我是只是简单地解释了虚函数的实现原理，可究竟对象的内部的内存布局是怎样的？类数据成员与多个虚表的具体内存布局又是怎样的？编译器是如何在赋值的时候作了优化的呢？我在以后的博客里会讲一下。  
补充一下链接：<http://www.cnblogs.com/qiaoconglifelife/p/5299959.html>  
理解好多态，对理解面向对象编程的思想有很大的帮助！


分类: [C/C++](#)

标签: [C++](#), [派生](#), [多态](#), [虚函数](#), [虚函数表](#)

好文要顶

关注我

收藏该文

 **CoderCong**

关注 - 2

粉丝 - 14

+加关注

0 0

« 上一篇: [构造函数、析构造函数、虚析构造函数、纯虚析构造函数要点](#)  
» 下一篇: [32位机与64位机数据大小](#)

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！

【报名】2050 大会 - 博客园程序员聚会（5.25 杭州·云栖小镇）

【招聘】花大价钱找技术大牛我们是认真的！

【腾讯云】买域名送解析+SSL证书+建站



#### 最新IT新闻：

- 走出领英后，沈博阳的目标是打造租房领域的超级独角兽
  - 天宫一号：我这一辈子
  - Spotify CEO：上市首日股价可能表现不佳，已做好准备
  - 印度网约车巨头Ola已收购Ridlr，未来或推出公共交通服务
  - 三星电子CEO去年薪酬2290万美元 太子李在镕仅拿到82万
- » 更多新闻...



#### 最新知识库文章：

- 写给自学者的入门指南
  - 和程序员谈恋爱
  - 学会学习
  - 优秀技术人的管理陷阱
  - 作为一个程序员，数学对你到底有多重要
- » 更多知识库文章...