4.3 多态的实现原理_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 4.3 多态的实现原理涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

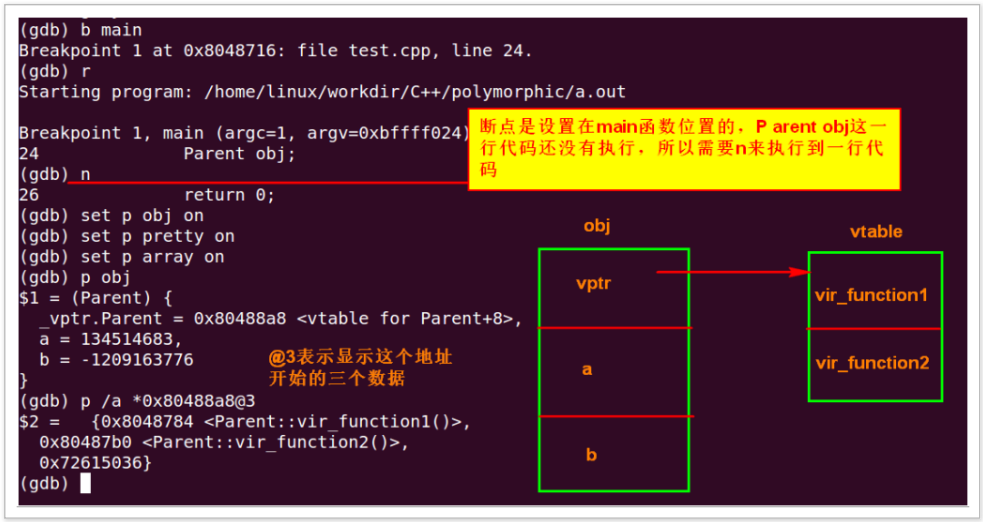
3. 多态的实现原理

一、虚函数表与 vptr 指针

```
class Parent{


public:
  virtual void vir_function1()  {
      cout << "vir_function1" << endl;
  }


  virtual void vir_function2() {
      cout << "vir_function2" << endl;
  }


private:
     int a;
     int b;
};
```

用 64bit 编译器编译出 32bit 程序：

```
linux@ubuntu:~/cpp/lesson17$ g++ Test.cpp -g -m32
In file included from Test.cpp:1:0:
/usr/include/c++/7/iostream:38:10: fatal error: bits/c++config.h:
No such file or directory
 #include <bits/c++config.h>
          ^~~~~~~~~~~~~~~~~~


解决方案：
sudo apt-get install g++-multilib
gdb a.out
```



gdb 调试:

- 增加调试信息
  - -g
- 设置断点
  - b function/line
- 按照层次打印
  - set p pretty <on/off>
- 显示虚函数表
  - set p obj <on/off>
- 数组内容层次显示
  - set p array on
- 数据显示格式
  - x 按十六进制格式显示变量
  - d 按照十进制格式显示变量
  - u 按十六进制格式显示无符号整型
  - o 按八进制格式显示变量
  - t 按二进制格式显示变量
  - a 按十六进制格式显示变量
  - c 按字符格式显示变量
  - f 按浮点数格式显示变量
- 查看函数入口地址
  - info line 10(查看第 10 行函数的入口地址)

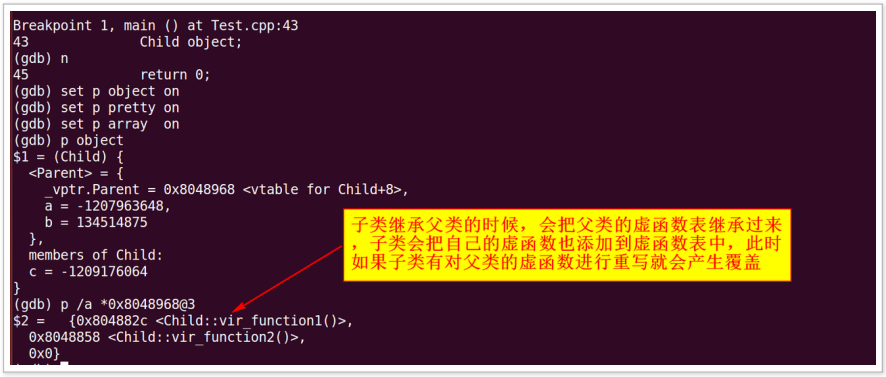
总结:

- 当类中声明虚函数时，编译器会在类中生成一个虚函数表
- 虚函数表存放的是虚函数的入口地址
- 虚函数表是由编译器自动生成与维护的
- virtual 成员函数会被编译器放入虚函数表中
- 存在虚函数时，每个对象中都有一个指向虚函数表的指针 (vptr 指针)

二、子类继承父类的虚函数表

```
#include <iostream>


using namespace std;


class Parent
{
        public:
                virtual void vir_function1()
                {
                        cout << "virtual_function1" << endl;
                }


                virtual void vir_function2()
                {
                        cout  << "vir_function2" << endl;
                }




        private:
                int a;
                int b;
};


class Child:public Parent
{
        public:
                virtual void vir_function1()
                {
                        cout << "virtual_function1" << endl;
                }


                virtual void vir_function2()
                {
                        cout  << "vir_function2" << endl;
                }


        private:
                int c;
};


int main(void)
{
```

```
        Child object;


        return 0;
}
```

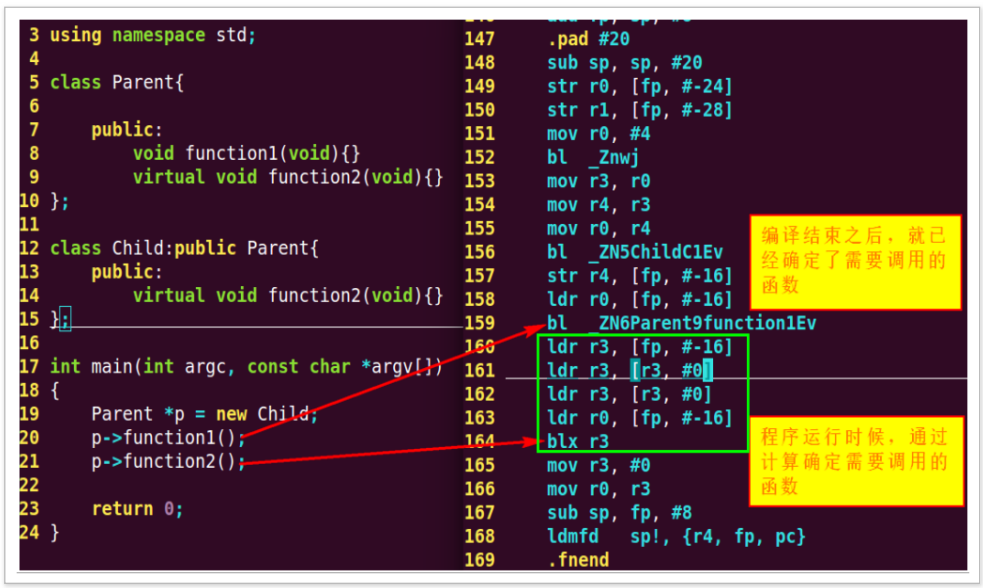


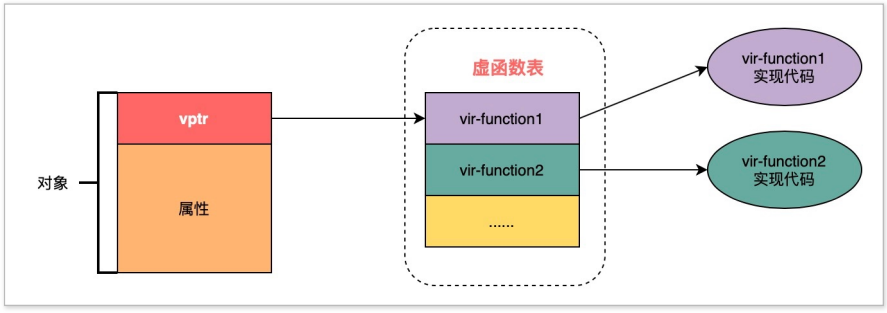
三、静态绑定与动态绑定

静态绑定：程序编译结束之后，就已经确定了需要调用的函数（根据对象类型或指针的类型来调用函数）

动态绑定：程序编译的时候，没有确定需要调用的函数，程序在运行的时候，才确定需要调用的函数

> 只有在多态的场合才会出现动态绑定，否则就是静态绑定





四、思考

问题：为什么用父类指针或引用指向子类对象的时候，可以根据子类对象的不同，调用不同子类对象的函数?

** 回答:** 因为在多态的场合是动态绑定的，** 程序运行的时候，会根据子类对象里面的 vptr 找到虚函数表，然后通过查虚函数表找到要调用的函数。** 由于是不同的子类对象，每个子类对象对应的虚函数表中函数实现不一样，最终调用的函数也就不一样了。

五、任务

请写出如下程序 (32bit 和 64bit) 运行的结果

```
#include <iostream>


using namespace std;


class A{
public:
    virtual void function1(void)
    {
        cout << "A::function1" << endl;
    }


private:
    int a;
    static int b;
};


int A::b = 0;


class B{
public:
    void function2(void)
    {
        cout << "B::function2" << endl;
    }


    virtual void function(void)
    {
        cout << "B::function" << endl;
    }
private:
```

```
    int b;
};

class C:public A,public B
{
public:
    void function1(void)
    {
        cout << "C:function1" << endl;
    }

    void function2(void)
    {
        cout << "C:function2" << endl;
    }

private:
    int c;
};

void function1(A &obj)
{
    obj.function1();
}

void function2(B &obj)
{
    obj.function2();
}

int main(void)
{
    C obj;

    cout << "sizeof(obj) : " << sizeof(obj) << endl;

    function1(obj);
    function2(obj);

    return 0;
}
```

- 划线
- 写笔记

学习要认真，笔记应当先

公开笔记 0/1000 提交

Sunny_SunshineX

删除 编辑

全文完