

## 【第三章：BAT等名企面试真题解析8讲】第8节：阿里面试真题解析之C++相关问题

已购

来自【《收割BAT：C++校招学习路线总结》】 | 150 浏览 | 0 回复 | 2019-12-06



特立独行MVP



专栏作者

+关注

### 前言

阿里虽然是国内Java的第一大厂但是并非所有的业务都是由Java支撑，很多服务和中下层的存储，计算，网络服务，大规模的分布式任务都是由C++编写。在阿里所有部门当中对C++考察最深的可能就是阿里云。在此我将以我今年实习和秋招面试阿里云的真题进行分析，通过真题对面试当中遇到的C++问题进行解答。

### C++面试重点

我本人是以C++作为主编程语言。C++是后台开发以及基础架构方向使用较多的语言之一。在我所有的面试当中，对于C++语言的考察主要集中在以下几点：

- 1.STL 容器相关实现
- 2.C++新特性的了解
- 3.多态和虚函数的实现
- 4.指针的使用

### 阿里面试真题再现

**问题1：**现在假设有一个编译好的C++程序，编译没有错误，但是运行时报错，报错如下：你正在调用一个纯虚函数(**Pure virtual function call error**)，请问导致这个错误的原因可能是什么？

答：纯虚函数调用错误一般由以下几种原因导致：

1. 从基类构造函数直接调用虚函数。
2. 从基类析构函数直接调用虚函数。
3. 从基类构造函数间接调用虚函数。
4. 从基类析构函数间接调用虚函数。
5. 通过悬空指针调用虚函数。

其中1，2编译器会检测到此类错误。3，4，5编译器无法检测出此类情况，会在运行时报错。

## ☰ 特立独行MVP

---

### 解析：

首先对关键知识点进行回顾：虚函数表，对象构造和析构过程

### 虚函数表vtbl：

1. 编译器在编译时期为每个带虚函数的类创建一份虚函数表
2. 实例化对象时，编译器自动将类对象的虚表指针指向这个虚函数表

### 构造一个派生类对象的过程：

#### 1.构造基类部分：

1. 将实例的虚表指针指向基类的vtbl
2. 构造基类的成员变量
3. 执行基类的构造函数函数体

#### 2.递归构造派生类部分：

1. 将实例的虚表指针指向派生类vtbl
2. 构造派生类的成员变量
3. 执行派生类的构造函数函数体

### 析构一个派生类对象的过程：

#### 1.递归析构派生类部分：

1. 将实例的虚表指针指向派生类vtbl
2. 执行派生类的析构函数函数体
3. 析构派生类的成员变量

#### 2.析构基类部分：

1. 将实例的虚表指针指向基类的vtbl
2. 执行基类的析构函数函数体
3. 析构基类的成员变量

由以上可知在构造函数和析构函数执行函数体过程时，实例的虚表指针指向的是构造函数和析构函数本身所属的那部分的类的虚函数表，此时执行的虚函数都实际调用的是该类本身的虚函数，所以如果在基类的析构或者构造函数当中调用虚函数且该虚函数本身在基类当中是纯虚函数那么就会出现纯虚函数调用。

### 可以运行如下代码进行验证：

```
1  #include <iostream>
2  using namespace std;
3
```

## ☰ 特立独行MVP

```
7     void helper() {
8         virtualFunc();
9     }
10    virtual ~Parent(){
11        helper();
12    }
13};
14
15class Child : public Parent{
16    public:
17    void virtualFunc() {
18        cout << "Child" << endl;
19    }
20    virtual ~Child(){}
21};
22
23
24int main() {
25
26    Child child;
27    return 0;
28}
```

运行时报错：libc++abi.dylib: Pure virtual function called!

### 通过悬空指针调用虚函数：

悬空指针:指针最初指向的内存已经被释放了的一种指针，访问"不安全可控"的内存区域将导致未定义的行为。

如下代码显示了悬空指针调用虚函数的典型案例：

```
1  #include <iostream>
2  using namespace std;
3
4  class Parent {
5  public:
6      virtual void virtualFunc() = 0;
7      void testFunc(){};
8      virtual ~Parent(){
9          testFunc();
10     };
11 };
12
13 class Child : public Parent{
14 public:
15     virtual void virtualFunc() {
16         cout << "Child-VirtualFunc-call" << endl;
17     }
18     virtual ~Child(){};
19 };
20
21
```

## ☰ 特立独行MVP

```
25     Parent* p = child;
26     //p此时可以成功的调用Child的virtualFunc输出"Child-VirtualFunc-call"
27     p->virtualFunc();
28     delete child;
29     //在delete child之后p就是一个悬空指针
30     p->virtualFunc();
31
32     return 0;
33 }
```

上述代码当中，p指向一个前对象，该对象已经被delete。根据C++标准，它是“未定义的”：意味着任何事情都可能发生：程序可能崩溃，或者继续运行，行为可能因编译器而异，或因计算机而异，或运行时不同。有几种常见的可能性：

- 内存可能被标记为已释放。任何访问它的尝试都将立即标记为使用了悬空指针。
- 内存可能被故意加密。释放后，内存管理系统可能会将类似垃圾的值写入内存。
- 内存可能会被重用。如果在删除对象和使用悬空指针之间执行了其他代码，则内存分配系统可能已经从旧对象使用的部分或全部内存中创建了一个新对象。如果幸运的话，这看起来就像垃圾，程序立即崩溃。否则，该程序可能会在某个时间之后崩溃。
- 内存可能完全保留，没有变化。

最后一种情况就是此时对象的虚表指针指向的是基类的虚函数表，此时调用的是纯虚函数。

### 问题2：是先构造父类的虚表指针还是先构造父类的成员？

答：由问题1解析可知先构造虚表指针再构造成员变量。

对于本类来说：先设定本类虚表指针->执行初始化列表->调用成员变量构造函数->执行本身构造函数体

### 问题3：在构造实例过程当中一部分是初始化列表一部分是在函数体内，你能说一下这些的顺序是什么？差别是什么和this指针构造的顺序

答：初始化列表当中的先初始化，然后才是函数体内代码被执行。构造函数本身也只是一个函数，执行构造函数时所有成员其实都已经初始化完成。this指针属于对象，初始化列表在构造函数之前执行，在对象还没有构造完成前，使用this指针，编译器无法识别。所以this指针在初始化列表当中不应当使用，在构造函数体内部可以使用。

解析：构造函数的执行可以分成两个阶段：

- 初始化阶段：所有类类型的成员都会在初始化阶段初始化，即使该成员没有出现在构造函数的初始化列表中。
- 计算赋值阶段：一般用于执行构造函数体内的赋值操作。

可以使用如下代码进行验证：

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

## ☰ 特立独行MVP

```
6     Test1(){
7         cout << "Construct Test1" << endl;
8     }
9     Test1& operator = (const Test1& t1) {
10        cout << "Assignment for Test1" << endl;
11        this->a = t1.a;
12        return *this;
13    }
14    int a ;
15 };
16
17 class Test2 {
18 public:
19     Test1 test1;
20     Test2(Test1 &t1) {
21         cout << "构造函数体开始" << endl;
22         test1 = t1 ;
23         cout << "构造函数体结束" << endl;
24     }
25 };
26
27 int main() {
28     Test1 t1;
29     Test2 test(t1);
30     return 0;
31 }
```

输出：

```
1 Construct Test1
2 Construct Test1
3 构造函数体开始
4 assignment for Test1
5 构造函数体结束
```

可以看出Test2在构造函数体执行之前已经使用了Test1的默认构造函数初始化好了t1。

### 问题4：初始化列表的写法和顺序有没有什么关系？

答：成员初始化的顺序和它们在类定义中出现的顺序一致，构造函数初始值列表中的前后位置不会影响实际的初始化顺序。当数据成员是 `const`、引用，或者属于某种未提供默认构造函数的类类型的话，就必须通过构造函数的初始值列表为这些成员提供初始值，否则就会引发错误。

### 问题5：如果父类有一个虚函数叫func\_A,子类也实现这个函数，在子类的构造函数当中去调用这个func\_A，运行的是谁的实现？

答：运行的是子类的实现。因为子类构造函数调用的时候对象的虚表指针指向的是子类的虚函数表，因为子类实现了func\_A所以调用的是子类自己的func\_A。

## ☰ 特立独行MVP

答：虚表指针先构造。

### 问题7：c++运行构造函数的时候虚函数表被构造出来了么？

答：构造出来了。因为虚函数表是在编译时由编译器创建，在运行时肯定已经创建完成。

### 问题8：在普通的函数当中调用虚函数和在构造函数当中调用虚函数有什么区别？

答：普通函数当中调用虚函数是希望运行时多态。而在构造函数当中不应该去调用虚函数因为构造函数当中调用的就是本类型当中的虚函数，无法达到运行时多态的作用。

### 问题9：成员变量，虚函数表指针的位置是怎么排布？

答：如果一个类带有虚函数，那么该类实例对象的内存布局如下：首先是一个虚函数指针，接下来是该类的成员变量，按照成员在类当中声明的顺序排布，整体对象的大小由于内存对齐会有空白补齐。其次如果基类没有虚函数但是子类含有虚函数此时内存子类对象的内存排布也是先虚函数表指针再各个成员。如果将子类指针转换成基类指针此时编译器会根据偏移做转换。

可以使用如下代码验证：

```
1  #include <iostream>
2  using namespace std;
3
4  class Parent{
5  public:
6      int a;
7      int b;
8  };
9
10 class Child:public Parent{
11 public:
12     virtual void test(){}
13     int c;
14 };
15
16 int main() {
17     Child c = Child();
18     Parent p = Child();
19     cout << sizeof(c) << endl;//24
20     cout << sizeof(p) << endl;//8
21
22     Child* cc = new Child();
23     Parent* pp = cc;
24     cout << cc << endl;//0x7fbe98402a50
25     cout << pp << endl;//0x7fbe98402a58
26     cout << &(cc->a) << endl;//0x7fbe98402a58
```

## ☰ 特立独行MVP

```
30 | }
```

输出：

```
1 | 24
2 | 8
3 | 0x7fbe98402a50
4 | 0x7fbe98402a58
5 | 0x7fbe98402a58
6 | 0x7fbe98402a5c
7 | 0x7fbe98402a60
```

我的测试环境是64位，所以指针为8个字节。转换之后pp和cc相差一个虚表指针的偏移。

## 总结

阿里考察C++的问题集中在以下几点：

1. 虚函数的实现
2. 虚函数使用出现的问题原因
3. 带有虚函数的类对象的构造和析构过程
4. 对象的内存布局
5. 虚函数的缺点：相比普通函数，虚函数调用需要2次跳转，会降低CPU缓存的命中率。运行时绑定，编译器不好优化。

## 结语

感谢阅读，如果文中有任何错误欢迎你给我留言指出，也欢迎分享给更多的朋友一起阅读。

[举报](#)

收藏



赞

### 相关专栏



《收割BAT：C++校招学习路线总结》

19篇文章 | 95订阅

已订阅

0条评论

🔼 默认排序 ▼

☰ 特立独行MVP



没有回复

请留下你的观点吧~

发布


专栏推荐



《收割BAT：C++校招学习路线总结》

《收割BAT：C++校招学习路线总结》，专栏共计17节。专栏分为五大主要内容，包括后台开发学习...

19篇文章 | 95阅读

 牛客博客，记录你的成长

[关于博客](#) | [意见反馈](#) | [免责声明](#) | [牛客网首页](#)