Item 9：在析构/构造时不要调用虚函数 effective-cpp

C++ 虚函数 析构函数 构造函数

> Item 9: Never call virtual functions during construction or destruction.

父类构造期间，对虚函数的调用不会下降至子类。如果这并非你的意图，请不要这样做！ 这个问题阿里实习面试曾经问到过，看这篇文章： 2014阿里巴巴面试经历

看Scott Meyers举的例子：

```
class Transaction {                             // base class for all
public:                                         // transactions
    Transaction(){                              // base class ctor
        logTransaction();                       // as final action, log this
    }
    virtual void logTransaction() const = 0;    // make type-dependent
};

class BuyTransaction: public Transaction {      // derived class
public:
    virtual void logTransaction() const;        // how to log trans-
};
...
BuyTransaction b;
```

`b` 在构造时，调用到父类 `Transaction` 的构造函数，其中对 `logTransaction` 的调用会被解析到 `Transaction` 类。 那是一个纯虚函数，因此程序会非正常退出。

> 其实，对于构造函数中直接的虚函数调用，某些编译器会发出警告。

这一点很好理解，因为父类对象会在子类之前进行构造，此时 **子类部分的数据成员还未初始化** ， 因此调用子类的函数是不安全的，因此C++不允许这样做。除此之外， **在子类对象的父类构造期间，对象类型为父类而非子类** 。 不仅虚函数会被解析至父类，运行时类型信息也为父类（ `dynamic_cast` ， `typeid` ）。

C++提供了RTTI（Run-Time Type Identification，运行时类型识别）机制， 我们通过 `typeid` 操作符便可以得到动态的类型信息，我们在父类的构造函数中输出当前对象的类型：

```
class Transaction{
public:
```

```
    Transaction(){
        cout<<typeid(this).name()<<endl;
    }
};
class BuyTransaction: public Transaction{
public:
    BuyTransaction(){
        cout<<typeid(this).name()<<endl;
    }
};
void main(){
    BuyTransaction b;
}
```

输出：

```
P11Transaction
P14BuyTransaction
```

可见，子类对象在父类构造时期，运行时类型确实为父类。与此同时，`dynamic_cast` 也会解析到父类，父类构造函数中调用虚函数就是例子~

转载请注明来源： http://harttle.com/2015/07/27/effective-cpp-9.html 欢迎对文章中的引用来源进行考证，欢迎指出任何有错误或不够清晰的表达。可以在下面评论区评论（可能需要在能访问 disqus 服务的网络），也可以邮件至 harttle@126.com 。

> 上一篇：Item 8：析构函数不要抛出异常
> 下一篇：Item 10：赋值运算符要返回自己的引用