C++ 数据抽象 C++ 接口（抽象类）

C++ 数据封装

所有的 C++ 程序都有以下两个基本要素：

- **程序语句（代码）：**这是程序中执行动作的部分，它们被称为函数。
- **程序数据：**数据是程序的信息，会受到程序函数的影响。

封装是面向对象编程中的把数据和操作数据的函数绑定在一起的一个概念，这样能避免受到外界的干扰和误用，从而确保了安全。数据封装引申出了另一个重要的 OOP 概念，即**数据隐藏。**

数据封装是一种把数据和操作数据的函数捆绑在一起的机制，**数据抽象**是一种仅向用户暴露接口而把具体的实现细节隐藏起来的机制。

C++ 通过创建**类**来支持封装和数据隐藏（public、protected、private）。我们已经知道，类包含私有成员（private）、保护成员（protected）和公有成员（public）成员。默认情况下，在类中定义的所有项目都是私有的。例如：

```
class Box
{
public:
double getVolume(void)
{
return length * breadth * height;
}
private:
double length; // 长度
double breadth; // 宽度
double height; // 高度
};
```

变量 length、breadth 和 height 都是私有的（private）。这意味着它们只能被 Box 类中的其他成员访问，而不能被程序中其他部分访问。这是实现封装的一种方式。

为了使类中的成员变成公有的（即，程序中的其他部分也能访问），必须在这些成员前使用 **public** 关键字进行声明。所有定义在 public 标识符后边的变量或函数可以被程序中所有其他的函数访问。

把一个类定义为另一个类的友元类，会暴露实现细节，从而降低了封装性。理想的做法是尽可能地对外隐藏每个类的实现细节。

数据封装的实例

C++ 程序中，任何带有公有和私有成员的类都可以作为数据封装和数据抽象的实例。请看下面的实例：

实例

```
#include <iostream>
using namespace std;
class Adder{
public:
// 构造函数
Adder(int i = 0)
```

```
{
total = i;
}
// 对外的接口
void addNum(int number)
{
total += number;
}
// 对外的接口
int getTotal()
{
return total;
};
private:
// 对外隐藏的数据
int total;
};
int main( )
{
Adder a;
a.addNum(10);
a.addNum(20);
a.addNum(30);
cout << "Total " << a.getTotal() <<endl;
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Total 60
```

上面的类把数字相加，并返回总和。公有成员 **addNum** 和 **getTotal** 是对外的接口，用户需要知道它们以便使用类。私有成员 **total** 是对外隐藏的，用户不需要了解它，但它又是类能正常工作所必需的。

设计策略

通常情况下，我们都会设置类成员状态为私有（ private ），除非我们真的需要将其暴露，这样才能保证良好的**封装性。**

这通常应用于数据成员，但它同样适用于所有成员，包括虚函数。

← C++ 数据抽象　　C++ 接口（ 抽象类 ）→

用关系，故而重写是被允许的。

编译器不检查虚函数的各类属性。被virtual修饰的成员函数，不论他们是private、protect或是public的，都会被统一的放置到虚函数表中。对父类进行派生时，子类会继承到拥有相同偏移地址的虚函数表（相同偏移地址指，各虚函数相对于VPTR指针的偏移），则子类就会被允许对这些虚函数进行重载。且重载时可以给重载函数定义新的属性，例如public，其只标志着该重载函数在该子类中的访问属性为public，和父类的private属性没有任何关系！

纯虚函数可以设计成私有的，不过这样不允许在本类之外的非友元函数中直接调用它，子类中只有覆盖这种纯虚函数的义务，却没有调用它的权利。

柠檬怪 5天前