C++ 重载运算符和重载函数 C++ 数据抽象

C++ 多态

多态按字面的意思就是多种形态。当类之间存在层次结构，并且类之间是通过继承关联时，就会用到多态。

C++ 多态意味着调用成员函数时，会根据调用函数的对象的类型来执行不同的函数。

下面的实例中，基类 Shape 被派生为两个类，如下所示：

实例

```
#include <iostream>
using namespace std;
class Shape {
protected:
int width, height;
public:
Shape( int a=0, int b=0)
{
width = a;
height = b;
}
int area()
{
cout << "Parent class area :" <<endl;
return 0;
}
};
class Rectangle: public Shape{
public:
Rectangle( int a=0, int b=0):Shape(a, b) { }
int area ()
{
cout << "Rectangle class area :" <<endl;
return (width * height);
}
};
class Triangle: public Shape{
public:
Triangle( int a=0, int b=0):Shape(a, b) { }
int area ()
{
cout << "Triangle class area :" <<endl;
return (width * height / 2);
}
};
// 程序的主函数
int main( )
{
Shape *shape;
Rectangle rec(10,7);
Triangle tri(10,5);
// 存储矩形的地址
```

```
shape = &rec;
// 调用矩形的求面积函数 area
shape->area();
// 存储三角形的地址
shape = &tri;
// 调用三角形的求面积函数 area
shape->area();
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Parent class area
Parent class area
```

导致错误输出的原因是，调用函数 area() 被编译器设置为基类中的版本，这就是所谓的**静态多态**，或**静态链接** - 函数调用在程序执行前就准备好了。有时候这也被称为**早绑定**，因为 area() 函数在程序编译期间就已经设置好了。

但现在，让我们对程序稍作修改，在 Shape 类中，area() 的声明前放置关键字 **virtual**，如下所示：

```
class Shape {
protected:
int width, height;
public:
Shape( int a=0, int b=0)
{
width = a;
height = b;
}
virtual int area()
{
cout << "Parent class area :" <<endl;
return 0;
}
};
```

修改后，当编译和执行前面的实例代码时，它会产生以下结果：

```
Rectangle class area
Triangle class area
```

此时，编译器看的是指针的内容，而不是它的类型。因此，由于 tri 和 rec 类的对象的地址存储在 *shape 中，所以会调用各自的 area() 函数。

正如您所看到的，每个子类都有一个函数 area() 的独立实现。这就是**多态**的一般使用方式。有了多态，您可以有多个不同的类，都带有同一个名称但具有不同实现的函数，函数的参数甚至可以是相同的。

虚函数

虚函数 是在基类中使用关键字 **virtual** 声明的函数。在派生类中重新定义基类中定义的虚函数时，会告诉编译器不要静态链接到该函数。

我们想要的是在程序中任意点可以根据所调用的对象类型来选择调用的函数，这种操作被称为**动态链接**，或**后期绑定**。

纯虚函数

您可能想要在基类中定义虚函数，以便在派生类中重新定义该函数更好地适用于对象，但是您在基类中又不能对虚函数给出有意义的实现，这个时候就会用到纯虚函数。

我们可以把基类中的虚函数 area() 改写如下：

```
class Shape {
protected:
int width, height;
public:
Shape( int a=0, int b=0)
{
width = a;
height = b;
}
// pure virtual function
virtual int area() = 0;
};
```

= 0 告诉编译器，函数没有主体，上面的虚函数是**纯虚函数**。

← C++ 重载运算符和重载函数 C++ 数据抽象 →