C++程序设计基础 第八章 （更新中）

前言

本文档由 @ItsJiale 创作，作者博客：https://jiale.domcer.com/，作者依据数学与大数据学院 2024 级大数据教学班的授课重点倾向编写而成。所有内容均为手动逐字录入，其中加上了不少自己的理解与思考，耗费近一周时间精心完成。

此文档旨在助力复习 C++ 程序设计基础，为后续学习数据结构筑牢根基。信计专业的同学，也可参考本文档规划复习内容。需注意，若个人学习过程中存在不同侧重点或对重难点的理解有差异，应以教材内容为准。倘若文档内容存在任何不妥之处，恳请各位读者批评指正。

By：ItsJiale

2025.4.8

第8章 继承与派生

8.1 继承

继承是面向对象程序设计基础的重要特性之一

8.1.1 继承的概念

继承就是在原有类的基础上产生出新类，新类会继承原有类的所有属性和方法。原有的类称为基类或父类，新类称为派生类或子类（注意，为了和Java学习统一，今后将基类称为***父类***，派生类称为***子类***，***但更标准的说法是基类和派生类，此举仅仅为了方便记忆***）

声明一个类继承另一个类的格式如下

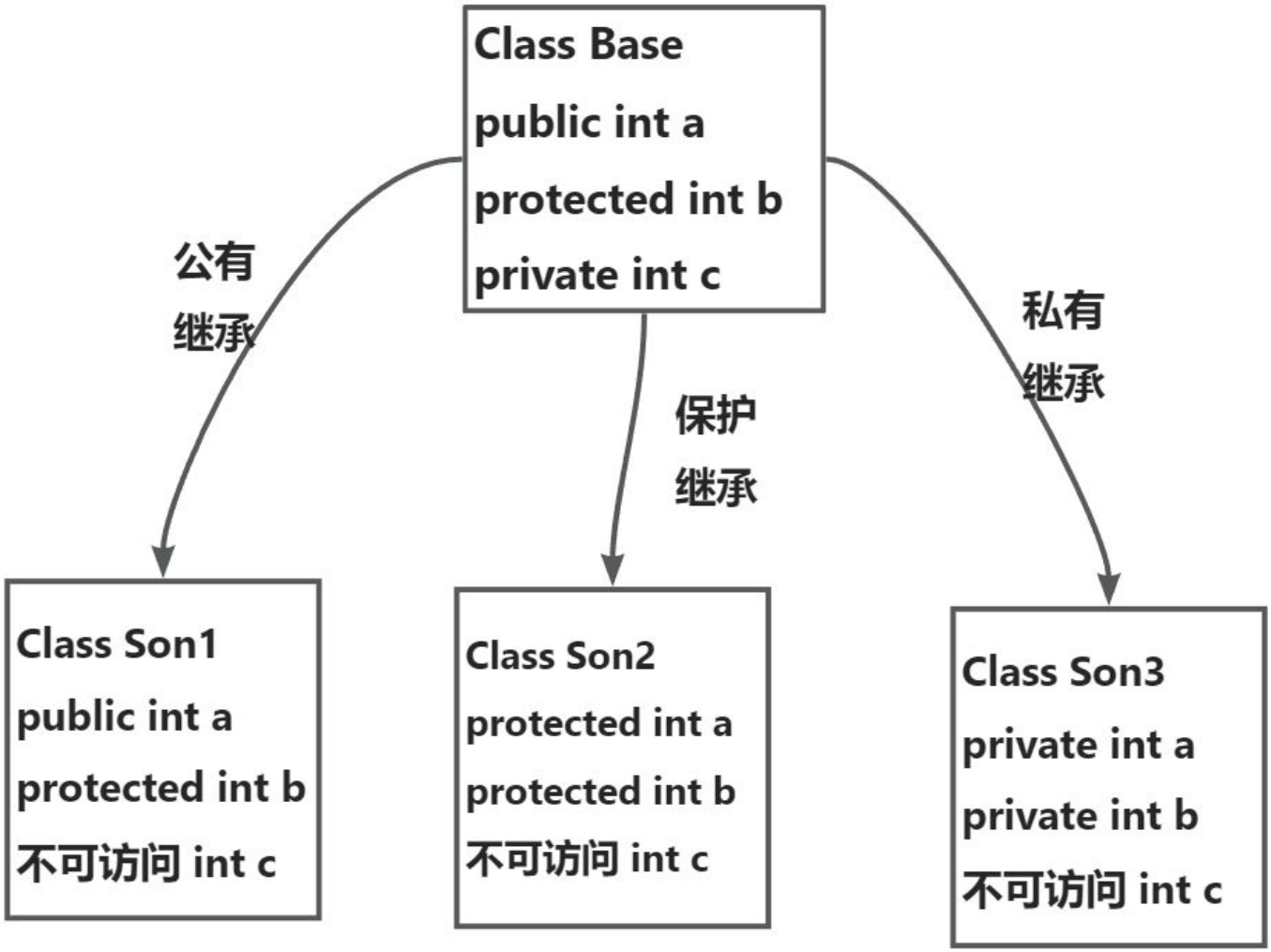
```
class 子类名称：继承方式 父类名称{
    子类成员声明
};
```

注意：

1. ***父类的构造函数与析构函数不能被继承***
2. 子类对父类成员的继承没有选择权，不能选择继承或不继承某些成员
3. 子类可以添加新的成员，用于实现新的功能，保证子类的功能在父类的基础上有所拓展
4. 一个父类可以有多个子类，一个子类可以继承多个父类

```cpp
#include <iostream>
using namespace std;

// 父类 Animal
class Animal {
public:
    void eat() {
        cout << "Animal is eating." << endl;
    }
};

// 子类 Dog 继承自 Animal
class Dog : public Animal {
public:
    void bark() {
        cout << "Dog is barking." << endl;
    }
};

int main() {
    Dog dog;
    dog.eat(); // 调用从父类继承的方法
    dog.bark(); // 调用子类自己的方法

    return 0;
}

```

8.1.2 继承方式



观察子类的变化

注意：

1. ***在类内：子类成员对父类成员的访问权限***

父类成员有三种访问权限，分别是 `public` （公有）、 `private` （私有）和 `protected` （保护）。子类对父类不同权限成员的访问情况如下：

- **公有成员**：子类可以随意访问。
- **保护成员**：子类能够访问。
- **私有成员**：子类无法访问。

2. ***在类外：通过子类对象对父类成员的访问权限***

在类外，只能通过子类对象访问父类的公有成员，因为公有成员在类外部是可见的，而私有和保护成员不可见。

8.1.3 类型兼容

（公有继承下，派生对象可以当做基类对象使用，此时派生对象只能发挥基类对象的作用）——> 不说人话

稍微改写一下：公有继承时，子类对象能当作父类对象用，不过只能发挥父类对象的作用。

```C++
#include <iostream>
using namespace std;
// 父类
class Parent {
public:
    int parentValue;
};

// 子类
class Child : public Parent {
public:
    int childValue;
};

// 函数参数传递示例
void printValue(Parent p) {
    cout << "Parent value: " << p.parentValue << endl;
}

int main() {
    Child c;
    c.parentValue = 10;
    c.childValue = 20;

    // 赋值兼容
    Parent p = c;
    cout << "Assigned parent value: " << p.parentValue << endl;

    // 函数参数传递
    printValue(c);

    return 0;
}
```

- **赋值兼容**：在 `main` 函数里，`Parent p = c;` 把 `Child` 类的对象 `c` 赋值给 `Parent` 类的对象 `p`，此时 `p` ***只能访问父类中定义的成员*** `parentValue`。
- **函数参数传递**：`printValue(c);` 可以将 `Child` 类的对象 `c` 传递给期望接收 `Parent`

类对象的函数 `printValue`，在函数内部，***也只能访问父类的成员*** `parentValue`。

总结

公有继承时子类对象能当作父类对象使用，是因为子类包含了父类的所有成员，但在使用过程中***只能发挥父类对象的作用***，即只能访问和调用父类中定义的成员。

在将子类对象当作父类对象使用的情境下，***通常是无法直接访问子类特有的方法的***

C++

```cpp
#include <iostream>
using namespace std;

// 定义父类 B0
class B0 {
public:
    // 定义成员函数 display，用于输出 "B0.display()"
    void display() {
        cout << "B0.display()" <<endl;
    }
};

// 定义子类 B1，公有继承自 B0
class B1 : public B0 {
public:
    // 定义与父类同名的成员函数 display，用于输出 "B1.display()"
    void display() {
        cout << "B1.display()" << endl;
    }
};

// 定义子类 D1，公有继承自 B1
class D1 : public B1 {
public:
    // 定义与父类同名的成员函数 display，用于输出 "D1.display()"
    void display() {
        cout << "D1.display()" << endl;
    }
};
// 定义函数 fun，该函数接受一个 B0 类型的指针作为参数
// 函数的功能是调用指针所指向对象的 display 函数
void fun(B0 *p) {
    p->display();
}

int main()
{
    // 创建 B0 类的对象 b0
    B0 b0;
    // 创建 B1 类的对象 b1
    B1 b1;
    // 创建 D1 类的对象 d1
    D1 d1;
    // 定义一个 B0 类型的指针 pt
    B0 *pt;
```

```
    // 让指针 pt 指向对象 b0
    pt = &b0;
    // 调用 fun 函数，传入指向 b0 的指针
    // 由于指针类型是 B0*，会调用 B0 类的 display 函数
    fun(pt);

    // 让指针 pt 指向对象 b1
    pt = &b1;
    // 调用 fun 函数，传入指向 b1 的指针
    // 虽然 pt 指向的是 B1 对象，但由于指针类型是 B0*，
    //仍然会调用 B0 类的 display 函数
    fun(pt);

    // 让指针 pt 指向对象 d1
    pt = &d1;
    // 调用 fun 函数，传入指向 d1 的指针
    // 同样，因为指针类型是 B0*，会调用 B0 类的 display 函数
    fun(pt);

    return 0;
}
```

上述例子类型兼容的体现：

1. 赋值兼容

在 `main` 函数里，你可以把 `B1` 和 `D1` 类的对象地址赋值给 `B0` 类型的指针 `pt`，这就体现了赋值兼容。代码如下：

C++

```
B0 b0;
B1 b1;
D1 d1;
B0 *pt;
pt = &b0;  // 指向 B0 类对象，这是常规操作
pt = &b1;  // 把 B1 类对象的地址赋值给 B0 类型的指针
pt = &d1;  // 把 D1 类对象的地址赋值给 B0 类型的指针
```

这里 `B1` 是 `B0` 的直接父类， `D1` 是 `B0` 的间接子类（通过 `B1` 继承）。由于是公有继承， `B1` 和 `D1` 类的对象包含了 `B0` 类对象的所有成员，所以可以将它们的地址赋值给 `B0` 类型的指针。这意味着在赋值时， `B1` 和 `D1` 类对象被视为 `B0` 类对象

2. 函数参数传递

函数 `fun` 接收一个 `B0` 类型的指针作为参数，在调用 `fun` 函数时，可以传入 `B0` 、 `B1` 或 `D1` 类对象的指针，这体现了函数参数传递方面的类型兼容。代码如下：

```C++
void fun(B0* p) {
    p->display();
}

// 在 main 函数中调用 fun 函数
pt = &b0;
fun(pt);
pt = &b1;
fun(pt);
pt = &d1;
fun(pt);
```

当把 `B1` 或 `D1` 类对象的指针传递给 `fun` 函数时，函数内部将其当作 `B0` 类对象的指针来处理。这是因为在公有继承关系下， `B1` 和 `D1` 类对象在一定程度上可以替代 `B0` 类对象使用，也就是满足类型兼容的要求。

8.2 派生类（子类）

8.2.1 派生类（子类）的构造函数和析构函数