文章目录

1.简介

子类为完成基类初始化，在 C++11 之前，需要在初始化列表调用基类的构造函数，从而完成构造函数的传递。如果基类拥有多个构造函数，那么子类也需要实现多个与基类构造函数对应的构造函数。

```
class Base {
public:
        Base(int v): _value(v), _c('0'){}
        Base(char c): _value(0), _c(c){}
private:
        int _value;
        char _c;
};

class Derived: public Base {
public:
        // 初始化基类需要透传参数至基类的各个构造函数，非常麻烦
        Derived(int v) :Base(v) {}
        Derived(char c) :Base(c) {}

        // 假设派生类只是添加了一个普通的函数
        void display() {
                // dosomething
        }
};
```

书写多个派生类构造函数只为传递参数完成基类初始化，这种方式无疑给开发人员带来麻烦，降低了编码效率。从 C++11 开始，推出了继承构造函数（Inheriting Constructor），使用 using 来声明继承基类的构造函数，我们可以这样书写。

```
class Base {
public:
        Base(int v) :_value(v), _c('0'){}
        Base(char c): _value(0), _c(c){}
```

```
private:
        int _value;
        char _c;
};

class Derived: public Base {
public:
        // 使用继承构造函数
        using Base::Base;

        // 假设派生类只是添加了一个普通的函数
        void display() {
                // do something
        }
};
```

我们通过 using Base::Base 把基类构造函数继承到派生类中，不再需要书写多个派生类构造函数来完成基类的初始化。

更为巧妙的是，C++11 标准规定，继承构造函数与类的一些默认函数（默认构造、析构、拷贝构造函数等）一样，是隐式声明，如果一个继承构造函数不被相关代码使用，编译器不会为其产生真正的函数代码。这样比通过派生类构造函数“透传构造函数参数”来完成基类初始化的方式，总是需要定义派生类的各种构造函数更加节省目标代码空间。

2.注意事项

（1）继承构造函数无法初始化派生类数据成员。

这个很好理解，因为继承构造函数的功能是初始化基类，对于派生类数据成员的初始化则无能为力。解决的办法主要有两个：

一是使用 C++11 特性就地初始化成员变量，可以通过 =、{} 对非静态成员快速就地初始化，以减少多个构造函数重复初始化变量的工作，注意初始化列表会覆盖就地初始化操作。

```
class Derived: public Base {
public:
        // 使用继承构造函数
        using Base::Base;

        // 假设派生类只是添加了一个普通的函数
        void display() {
                // do something
        }
private:
```

```
        // 派生类新增数据成员
        double _double{0.0};
};
```

二是新增派生类构造函数，使用构造函数初始化列表。

```
class Derived :public Base {
public:
        // 使用继承构造函数
        using Base::Base;

        // 新增派生类构造函数
        Derived(int a, double b):Base(a), _double(b){}

        // 假设派生类只是添加了一个普通的函数
        void display() {
                // do something
        }
private:
        // 派生类新增数据成员
        double _double{0.0};
};
```

相比之下，第二种方法需要新增构造函数，明显没有第一种方法简洁，但第二种方法可由用户控制初始化值，更加灵活。各有优劣，两种方法需结合具体场景使用。

（2）构造函数拥有默认值会产生多个构造函数版本，且继承构造函数无法继承基类构造函数的默认参数，所以我们在使用有默认参数构造函数的基类时必须要小心。

```
class A {
public:
        A(int a = 3, double b = 4): _a(a), _b(b){}
        void display() {
                cout<<_a<<" "<<_b<<endl;
        }

private:
        int _a;
        double _b;
};

class B:public A {
public:
```

```
        using A::A;
};
```

那么 A 中的构造函数会有下面几个版本：

```
A()
A(int)
A(int, double)
A(const A&)
```

那么 B 中对应的继承构造函数将会有如下几个版本：

```
B()
B(int)
B(int, double)
B(const B&)
```

可以看出，参数默认值会导致多个构造函数版本的产生，因此在使用时需格外小心。

（3）多继承的情况下，继承构造函数会出现“冲突”的情况，因为多个基类中的部分构造函数可能导致派生类中的继承构造函数的函数签名（函数名与参数）相同。

```
class A {
public:
        A(int i){}
};

class B {
public:
        B(int i){}
};

class C : public A, public B {
public:
        using A::A;
        using B::B;  //编译出错，重复定义C(int)

        // 显示定义继承构造函数 C(int)
        C(int i):A(i),B(i){}
};
```

为避免继承构造函数冲突，可以通过显示定义来阻止隐式生成的继承构造函数。

此外，使用继承构造函数还需要注意：如果基类构造函数被申明为私有成员函数，或者派生类是从虚基类继承而来 ，那么就不能在派生类中申明继承构造函数。

参考文献

Michael Wong, IBM XL编译器中国开发团队.深入理解C++11[M].C3.1 继承构造函数.P57-62