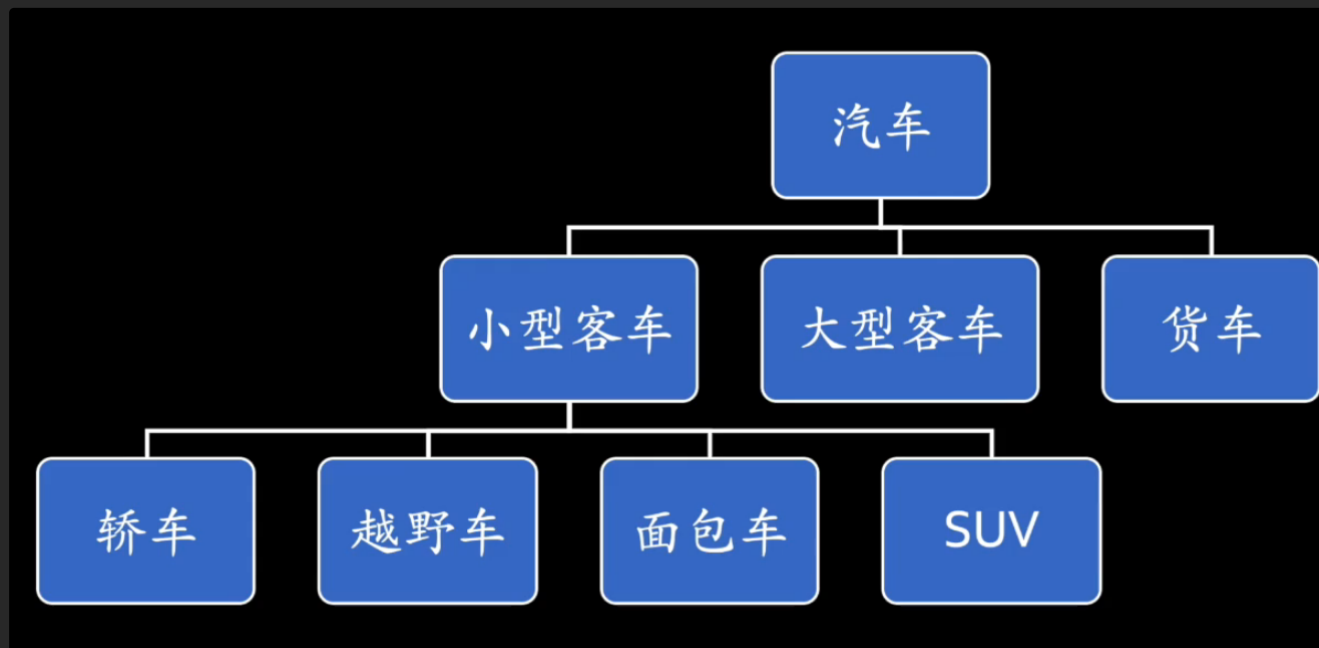


## 3.1继承与派生

### 继承与派生

- **派生**是指利用已有的类，通过继承的方式定义一个新类，新类继承了原来的类的属性和方法。
- 继承和派生体现了类的层次结构，表现了人们认识事物由一般到具体、由简单到复杂的过程。



- 在继承关系中，新定义的类称为被继承类的 **派生类** (derived class) 或 **子类** (subclass)，而被继承的类称为新定义类的 **基类** (base class) 或 **父类** (fatherclass)。
- 派生类继承了基类的所有成员。一个派生类也可以作为另一个派生类的基类。

```
1  class<派生类名>:[<继承方式>]<基类名>
2  {
3  <派生类类体>
4  }
```

- 继承方式有3种：public、private和protected。
  - public：基类成员的访问权限在派生类中保持不变。
  - private：基类中所有公有、保护成员在派生类中都成为私有成员。
  - protected：基类中所有的公有、保护成员在派生类中都成为保护成员。
- 派生类对基类成员的访问权限 **不超过** 继承方式。

### 例子：继承与派生

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  class CPoint
5  {
6  private:
7      double x, y;
8
9  public:
10     CPoint(double x=0,double y=0)
11     {
12         this->x=x;
13         this->y=y;
14     }
```

```

15
16     double getX()
17     {
18         return x;
19     }
20     double getY()
21     {
22         return y;
23     }
24     void setX(double x = 0)
25     {
26         x=x;
27     }
28     void setY(double y = 0)
29     {
30         y=y;
31     }
32 };
33
34
35
36 class CCirle:public CPoint // CCirle is derived from CPoint
37 {
38 private:
39     double r;
40
41 public:
42     CCirle(double r);
43     void setR(double r);
44     void setCenter(double x = 0, double y = 0);
45     double getR();
46     double getUpperLeftX();
47     double getUpperLeftY();
48 };
49
50 CCirle::CCirle(double r)
51 {
52     this->r=r;
53 }
54
55 void CCirle::setCenter(double x, double y)
56 {
57     this->setX(x);
58     this->setY(y);
59 }
60
61 double CCirle::getR()
62 {
63     return this->r;
64 }
65
66 double CCirle::getUpperLeftX()
67 {
68     return this->getX() - this->r;
69 }
70
71 double CCirle::getUpperLeftY()
72 {
73     return this->getY() + this->r;
74 }
75
76 int main()
77 {

```

```

78     CCircle c1(100);
79     c1.setCenter(100, 200);
80
81     cout << "O(" << c1.getX() << ", " << c1.getY() << ")" << endl;
82     cout << "R = " << c1.getR() << endl;
83     cout << "UpperLeft(" << c1.getUpperLeftX() << ", " << c1.getUpperLeftY() << ")" << endl;
84     return 0;
85 }

```

本例中，使用组合类的方式要优于继承的方式。

## 同名覆盖 (override)

- 同名覆盖：派生类可对基类的成员重新定义。
- 派生类的成员会把基类成员覆盖掉

```

class A {
public:
    void show() { cout << "A::show" << endl; }
};

class B : public A{
public:
    void show() { cout << "B::show" << endl; }
    void display() { show(); }
};

```

```

A::show
B::show
B::show

```

```

void eg4_2() {
    A a;
    B b;
    a.show();
    b.show();
    b.display();
}

```

```

B
void show() { cout << "A::show" << endl; }
void show() { cout << "B::show" << endl; }
void display() { show(); }

```

Eric\_

## 派生类的构造函数和析构函数

- 基类的构造函数、析构函数 **不能被继承**，故派生类须调用基类的构造函数对基类进行实例化。
- 编译时，先生成基类构造函数的调用代码，再生成派生类构造函数的调用代码。
- 基类构造函数的调用：
  - 隐式调用**，派生类未指定基类构造函数，贝则调用基类的默认构造函数。
  - 显式调用**，派生类指定基类构造函数，派生类将参数传递给基类的构造函数。
- 析构过程与构造过程相反。

## 构造函数的显式调用

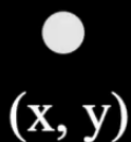
- 派生类显示定义构造函数的形式如下：

```

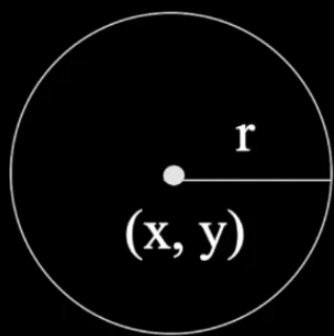
1  <派生类名>::<派生类名>(<形参声明>) :<基类名>(<参数表>)
2  <派生类构造函数的函数体>

```

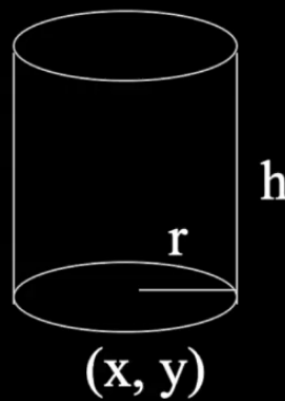
- <形参声明>的部分参数，传递给基类构造函数
- 基类有多个构造函数时，编译器根据<参数表>来确定调用基类的哪一个构造函数。



Point



Circle



Cylinder

## 例子：派生类的构造函数

### 写在类中

```
1  class CPoint
2  { //定义基类Point
3  protected:
4      int x, y;
5  public:
6      //含有缺省参数值的构造函数也是默认的构造函数
7      CPoint(int x=0, int y=0)
8      {
9          this->x = x;
10         this->y = y;
11         cout << "Point constructor: " << '[' << x << ',' << y << ']' << endl;
12     };
13     ~CPoint()
14     {
15         cout << "Point destructor: " << '[' << x << ',' << y << ']' << endl;
16     };
17 };
18
19
20 class CCircle: public CPoint
21 { //定义Point派生类Circle
22 protected:
23     int radius;
24 public:
25     //显式调用基类的构造函数
26     CCircle(int x = 0, int y = 0, int r = 0): CPoint(x, y):CPoint(x, y)
27     {
28         radius = r;
29         cout << "Circle constructor: " << '[' << radius << "]" [" << x << ',' << y << ']' << endl;
30     };
31     ~CCircle()
32     {
33         cout << "Circle destructor: " << '[' << radius << "]" [" << x << ',' << y << ']' << endl;
34     };
35 };
```

### 分开来写

```
1  //CCircle.hpp
2  class CCircle: public CPoint
3  { //定义Point派生类Circle
4  protected:
5      int radius;
```

```

6     public:
7         //显式调用基类的构造函数
8         CCircle(int x = 0, int y = 0, int r = 0);
9         ~CCircle();
10    };
11
12
13    //CCircle.cpp
14    CCircle::CCircle(int x, int y, int r) : CPoint(x, y)
15    {
16        radius = r;
17        cout << "Circle constructor: " << '[' << radius << "]" [" << x << ',' << y << ']' << endl;
18    };
19    CCircle::~~CCircle()
20    {
21        cout << "Circle destructor: " << '[' << radius << "]" [" << x << ',' << y << ']' << endl;
22    };

```

## 圆柱

```

1     class CCylinder: public CCircle
2     { //定义Circle派生类Cylinder
3     protected:
4         int height;
5     public:
6         //显式调用基类的构造函数
7         CCylinder(int x = 0, int y = 0, int r = 0, int h = 0): CCircle(x, y, r)
8         {
9             height = h;
10            cout << "Cylinder constructor: " << '[' << height << "]" [" << radius << "]" [" << x << ',' << y << ']' << endl;
11        };
12        ~CCylinder()
13        {
14            cout << "Cylinder destructor: " << '[' << height << "]" [" << radius << "]" [" << x << ',' << y << ']' << endl;
15        };
16    };

```

## 完整源码

```

1     #include <bits/stdc++.h>
2     using namespace std;
3
4     class CPoint
5     { //定义基类Point
6     protected:
7         int x, y;
8     public:
9         //含有缺省参数值的构造函数也是默认的构造函数
10        CPoint(int x=0, int y=0)
11        {
12            this->x = x;
13            this->y = y;
14            cout << "Point constructor: " << '[' << x << ',' << y << ']' << endl;
15        };
16        ~CPoint()
17        {
18            cout << "Point destructor: " << '[' << x << ',' << y << ']' << endl;
19        };
20    };
21
22    //CCircle.hpp
23    class CCircle: public CPoint

```

```

24  { //定义Point派生类Circle
25  protected:
26      int radius;
27  public:
28      //显式调用基类的构造函数
29      CCircle(int x = 0, int y = 0, int r = 0);
30      ~CCircle();
31  };
32
33  //CCircle.cpp
34  CCircle::CCircle(int x, int y, int r) : CPoint(x, y)
35  {
36      radius = r;
37      cout << "Circle constructor: " << '[' << radius << "]" [" << x << ',' << y << ']' << endl;
38  };
39  CCircle::~~CCircle()
40  {
41      cout << "Circle destructor: " << '[' << radius << "]" [" << x << ',' << y << ']' << endl;
42  };
43
44  class CCylinder: public CCircle
45  { //定义Circle派生类Cylinder
46  protected:
47      int height;
48  public:
49      //显式调用基类的构造函数
50      CCylinder(int x = 0, int y = 0, int r = 0, int h = 0): CCircle(x, y, r)
51      {
52          height = h;
53          cout << "Cylinder constructor: " << '[' << height << "]" [" << radius << "]" [" << x << ',' << y << ']' << endl;
54      };
55      ~CCylinder()
56      {
57          cout << "Cylinder destructor: " << '[' << height << "]" [" << radius << "]" [" << x << ',' << y << ']' << endl;
58      };
59  };
60
61  int main()
62  { //调用了类Point、Circle和Cylinder的构造函数
63      CCylinder c(200, 300, 100, 400);
64      return 0;
65  }

```