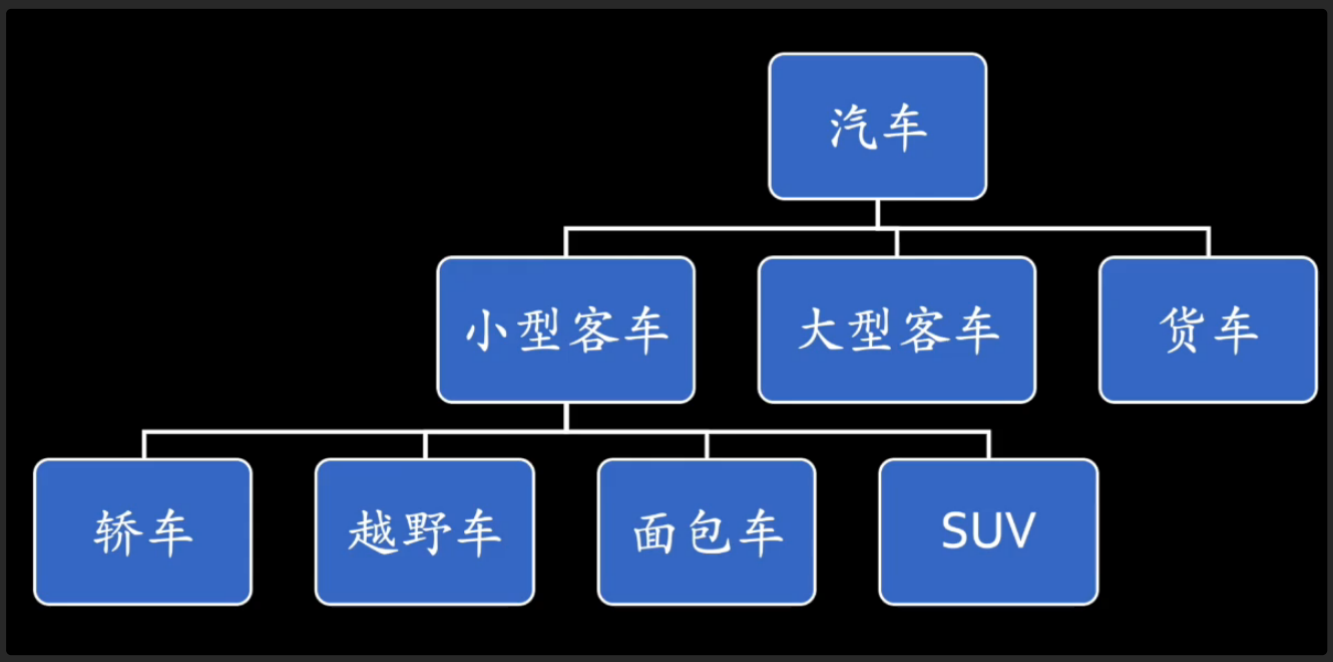
3.1继承与派生

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。

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

- 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

```
#include <bits/stdc++.h>
using namespace std;

class CPoint
{
private:
    double x, y;

public:
    CPoint(double x=0,double y=0)
    {
        this->x=x;
        this->y=y;
    }
```

```
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  {
```

```
    CCirle c1(100);
    c1.setCenter(100, 200);

    cout << "O(" << c1.getX() << ", " << c1.getY() << ")" << endl;
    cout << "R = " << c1.getR() << endl;
    cout << "UpperLeft(" << c1.getUpperLeftX() << ", " << c1.getUpperLeftY() << ")" << endl;
    return 0;
}
```

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

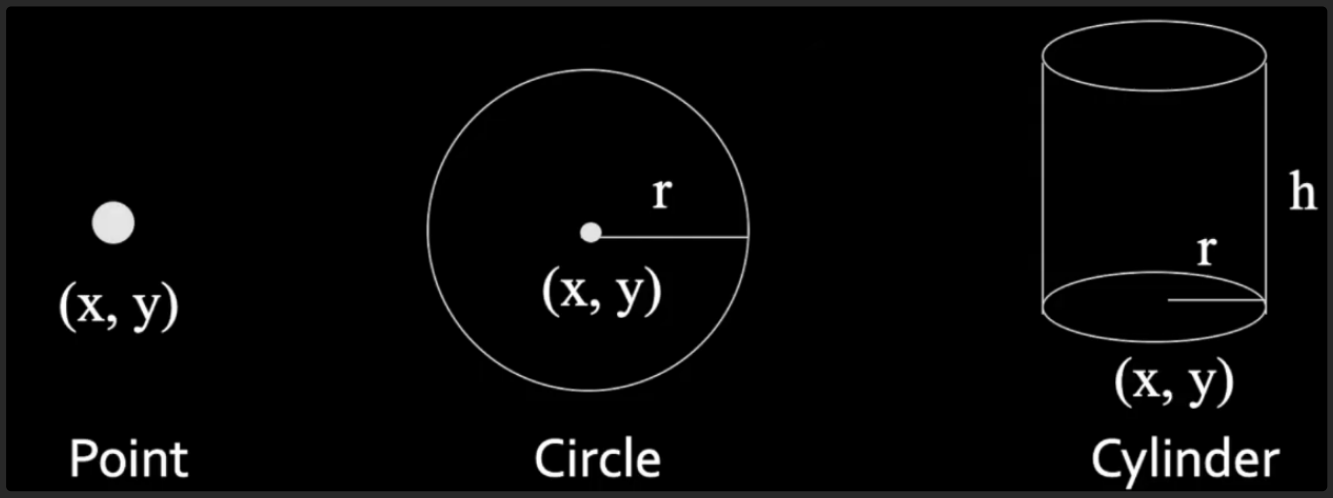
- 基类的构造函数、析构函数**不能被继承**，故派生类须调用基类的构造函数对基类进行实例化。
- 编译时，先生成基类构造函数的调用代码，再生成派生类构造函数的调用代码。
- 基类构造函数的调用：
  - **隐式调用**，派生类未指定基类构造函数，贝则调用基类的默认构造函数。
  - **显式调用**，派生类指定基类构造函数，派生类将参数传递给基类的构造函数。
- 析构过程与构造过程相反。

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：

```
<派生类名>::<派生类名>(<形参声明>):<基类名>(<参数表>)
<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。



例子：派生类的构造函数

写在类中

```
class CPoint
{//定义基类Point
protected:
    int x, y;
public:
    //含有缺省参数值的构造函数也是默认的构造函数
    CPoint(int x=0, int y=0)
    {
        this->x = x;
        this->y = y;
        cout << "Point constructor: " << '[' << x << ',' << y << ']' << endl;
    };
    ~CPoint()
    {
        cout << "Point destructor: " << '[' << x << ',' << y << ']' << endl;
    };
};


class CCirle: public CPoint
{//定义Point派生类Circle
protected:
    int radius;
public:
    //显式调用基类的构造函数
    CCirle(int x = 0, int y = 0, int r = 0): CPoint(x, y):CPoint(x, y)
    {
        radius = r;
        cout << "Circle constructor: " << '[' << radius << "] [" << x << ',' << y << ']' << endl;
    };
    ~CCirle()
    {
        cout << "Circle destructor: " << '[' << radius << "] [" << x << ',' << y << ']' << endl;
    };
};
```

分开来写

```
//CCircle.hpp
class CCirle: public CPoint
{//定义Point派生类Circle
protected:
    int radius;
```

```
public:
    //显式调用基类的构造函数
    CCirle(int x = 0, int y = 0, int r = 0);
    ~CCirle();
};


//CCircle.cpp
CCirle::CCirle(int x, int y, int r) : CPoint(x, y)
{
    radius = r;
    cout << "Circle constructor: " << '[' << radius << "] [" << x << ',' << y << ']' << endl;
};
CCirle::~CCirle()
{
    cout << "Circle destructor: " << '[' << radius << "] [" << x << ',' << y << ']' << endl;
};
```

圆柱

```
class CCylinder: public CCirle
{//定义Circle派生类Cylinder
protected:
    int height;
public:
    //显式调用基类的构造函数
    CCylinder(int x = 0, int y = 0, int r = 0, int h = 0): CCirle(x, y, r)
    {
        height = h;
        cout << "Cylinder constructor: " << '[' << height << "] [" << radius << "] [" << x << ',' << y << ']' << endl;
    };
    ~CCylinder()
    {
        cout << "Cylinder destructor: " << '[' << height << "] [" << radius << "] [" << x << ',' << y << ']' << endl;
    };
};
```

完整源码

```
#include <bits/stdc++.h>
using namespace std;

class CPoint
{//定义基类Point
protected:
    int x, y;
public:
    //含有缺省参数值的构造函数也是默认的构造函数
    CPoint(int x=0, int y=0)
    {
        this->x = x;
        this->y = y;
        cout << "Point constructor: " << '[' << x << ',' << y << ']' << endl;
    };
    ~CPoint()
    {
        cout << "Point destructor: " << '[' << x << ',' << y << ']' << endl;
    };
};

//CCircle.hpp
class CCirle: public CPoint
```

```
{//定义Point派生类Circle
protected:
    int radius;
public:
    //显式调用基类的构造函数
    CCirle(int x = 0, int y = 0, int r = 0);
    ~CCirle();
};

//CCircle.cpp
CCirle::CCirle(int x, int y, int r) : CPoint(x, y)
{
    radius = r;
    cout << "Circle constructor: " << '[' << radius << "] [" << x << ',' << y << ']' << endl;
};
CCirle::~CCirle()
{
    cout << "Circle destructor: " << '[' << radius << "] [" << x << ',' << y << ']' << endl;
};

class CCylinder: public CCirle
{//定义Circle派生类Cylinder
protected:
    int height;
public:
    //显式调用基类的构造函数
    CCylinder(int x = 0, int y = 0, int r = 0, int h = 0): CCirle(x, y, r)
    {
        height = h;
        cout << "Cylinder constructor: " << '[' << height << "] [" << radius << "] [" << x << ',' << y << ']' << endl;
    };
    ~CCylinder()
    {
        cout << "Cylinder destructor: " << '[' << height << "] [" << radius << "] [" << x << ',' << y << ']' << endl;
    };
};

int main()
{   //调用了类Point、CircLe和CyLinder的构造函数
    CCylinder c(200, 300, 100, 400);
    return 0;
}
```