1.类的一个特征就是封装，public和private作用就是实现这一目的。所以：

用户代码（类外）可以访问public成员而不能访问private成员；private成员只能由类成员（类内）和友元访问。

2.类的另一个特征就是继承，protected的作用就是实现这一目的。所以：

protected成员可以被派生类对象访问，不能被用户代码（类外）访问。

现来看看如下示例：

[?](https://www.jb51.net/article/54224.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | #include<iostream>  #include<assert.h>  **using** **namespace** std;  **class** A{  **public**:  **int** a;    A(){      a1 = 1;      a2 = 2;      a3 = 3;      a = 4;    }  **void** fun(){      cout << a << endl;    //正确      cout << a1 << endl;   //正确      cout << a2 << endl;   //正确，类内访问      cout << a3 << endl;   //正确，类内访问    }  **public**:  **int** a1;  **protected**:  **int** a2;  **private**:  **int** a3;  };  **int** main(){    A itema;    itema.a = 10;    //正确    itema.a1 = 20;    //正确    itema.a2 = 30;    //错误，类外不能访问protected成员    itema.a3 = 40;    //错误，类外不能访问private成员  **system**("pause");  **return** 0;  } |

**继承中的特点：**

先记住：不管是否继承，上面的规则永远适用！

有public, protected, private三种继承方式，它们相应地改变了基类成员的访问属性。

**1.public继承：**基类public成员，protected成员，private成员的访问属性在派生类中分别变成：public, protected, private

**2.protected继承：**基类public成员，protected成员，private成员的访问属性在派生类中分别变成：protected, protected, private

**3.private继承：**基类public成员，protected成员，private成员的访问属性在派生类中分别变成：private, private, private

但无论哪种继承方式，上面两点都没有改变：

1.private成员只能被本类成员（类内）和友元访问，不能被派生类访问；

2.protected成员可以被派生类访问。

再来看看以下代码：

**1.public继承**

代码如下：

[?](https://www.jb51.net/article/54224.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | #include<iostream>  #include<assert.h>  **using** **namespace** std;    **class** A{  **public**:  **int** a;    A(){      a1 = 1;      a2 = 2;      a3 = 3;      a = 4;    }  **void** fun(){      cout << a << endl;    //正确      cout << a1 << endl;   //正确      cout << a2 << endl;   //正确      cout << a3 << endl;   //正确    }  **public**:  **int** a1;  **protected**:  **int** a2;  **private**:  **int** a3;  };  **class** B : **public** A{  **public**:  **int** a;    B(**int** i){      A();      a = i;    }  **void** fun(){      cout << a << endl;       //正确，public成员      cout << a1 << endl;       //正确，基类的public成员，在派生类中仍是public成员。      cout << a2 << endl;       //正确，基类的protected成员，在派生类中仍是protected可以被派生类访问。      cout << a3 << endl;       //错误，基类的private成员不能被派生类访问。    }  };  **int** main(){    B b(10);    cout << b.a << endl;    cout << b.a1 << endl;   //正确    cout << b.a2 << endl;   //错误，类外不能访问protected成员    cout << b.a3 << endl;   //错误，类外不能访问private成员  **system**("pause");  **return** 0;  } |

**2.protected继承：**

代码如下：

[?](https://www.jb51.net/article/54224.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | #include<iostream>  #include<assert.h>  **using** **namespace** std;  **class** A{  **public**:  **int** a;    A(){      a1 = 1;      a2 = 2;      a3 = 3;      a = 4;    }  **void** fun(){      cout << a << endl;    //正确      cout << a1 << endl;   //正确      cout << a2 << endl;   //正确      cout << a3 << endl;   //正确    }  **public**:  **int** a1;  **protected**:  **int** a2;  **private**:  **int** a3;  };  **class** B : **protected** A{  **public**:  **int** a;    B(**int** i){      A();      a = i;    }  **void** fun(){      cout << a << endl;       //正确，public成员。      cout << a1 << endl;       //正确，基类的public成员，在派生类中变成了protected，可以被派生类访问。      cout << a2 << endl;       //正确，基类的protected成员，在派生类中还是protected，可以被派生类访问。      cout << a3 << endl;       //错误，基类的private成员不能被派生类访问。    }  };  **int** main(){    B b(10);    cout << b.a << endl;       //正确。public成员    cout << b.a1 << endl;      //错误，protected成员不能在类外访问。    cout << b.a2 << endl;      //错误，protected成员不能在类外访问。    cout << b.a3 << endl;      //错误，private成员不能在类外访问。  **system**("pause");  **return** 0;  } |

**3.private继承：**

代码如下：

[?](https://www.jb51.net/article/54224.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | #include<iostream>  #include<assert.h>  **using** **namespace** std;  **class** A{  **public**:  **int** a;    A(){      a1 = 1;      a2 = 2;      a3 = 3;      a = 4;    }  **void** fun(){      cout << a << endl;    //正确      cout << a1 << endl;   //正确      cout << a2 << endl;   //正确      cout << a3 << endl;   //正确    }  **public**:  **int** a1;  **protected**:  **int** a2;  **private**:  **int** a3;  };  **class** B : **private** A{  **public**:  **int** a;    B(**int** i){      A();      a = i;    }  **void** fun(){      cout << a << endl;       //正确，public成员。      cout << a1 << endl;       //正确，基类public成员,在派生类中变成了private,可以被派生类访问。      cout << a2 << endl;       //正确，基类的protected成员，在派生类中变成了private,可以被派生类访问。      cout << a3 << endl;       //错误，基类的private成员不能被派生类访问。    }  };  **int** main(){    B b(10);    cout << b.a << endl;       //正确。public成员    cout << b.a1 << endl;      //错误，private成员不能在类外访问。    cout << b.a2 << endl;      //错误, private成员不能在类外访问。    cout << b.a3 << endl;      //错误，private成员不能在类外访问。  **system**("pause");  **return** 0;  } |

通过以上的代码都备有较为详尽的注释，读者应该能够理解。仔细看代码中派生类B中定义了和基类同名的成员a，此时基类的a仍然存在，可以验证。

[?](https://www.jb51.net/article/54224.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **int** main(){    cout << **sizeof**(A) << endl;    cout << **sizeof**(B) << endl;    **system**("pause");  **return** 0;  } |

输出：

16

20

所以派生类包含了基类所有成员以及新增的成员，同名的成员被隐藏起来，调用的时候只会调用派生类中的成员。

如果要调用基类的同名成员，可以用以下方法：

[?](https://www.jb51.net/article/54224.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **int** main(){      B b(10);    cout << b.a << endl;    cout << b.A::a << endl;    **system**("pause");  **return** 0;  } |

输出：

10

4

记得这里是在类外访问，而a在基类中是public，所以继承方式应该为public，使得a在派生类中仍然为public，在类外可以访问。