|  |  |
| --- | --- |
| 1. 访问控制与继承 2. 受保护的成员 3. 公有、私有和受保护继承 4. 派生类向基类转换的可访问性 5. 友元与继承 6. 改变个别成员的可访问性 7. 默认的继承保护级别 | 可访问  Private控制派生类的访问权限  Class D : private/protected/public A{  }  Class Base{  Friend class Pal  }  class Sneaky1 :public Base1 {  int j;  void testfun() {  prot\_mem = 2;  }  };  class Pal {  public:  int f(Base1 b) { return b.prot\_mem; }  //int f2(Sneaky1 s) { return s.j; }  int f3(Sneaky1 s) { return s.prot\_mem; }// 对基类的访问权限由基类本身控制，派生类的基类部分也是如此,即基类对Pal友元，那么派生类对象的基类部分也对Pal友元，但是派生类本身的不对Pal友元。  };  Struct D1:Base{}  Class D2:Base{} |
| 1. 访问控制与继承 2. 每个类分别控制自己的成员初始化过程，每个类还分别控制着其成员对于派生类来说是否可访问 3. 受保护的成员   protected:   1. 受保护的成员对于类的用户来说是不可访问的，就是类的对象不可访问。对于类的派生类和友元来说可以访问 2. 重要的一点：派生类的成员或友元只能通过派生类对象来访问派生类对象中基类部分的protected成员，派生类对于一个基类对象中的受保护成员没有任何访问特权。   class Sneaky1 :public Base1 {  friend void clobber(Sneaky1&);  //friend void clobber(Base1&);  int j;  };  void clobber(Sneaky1 &s) { s.j = s.prot\_mem = 0; }  //void clobber(Base1 &s) { s.prot\_mem = 0; }// 不可，因为不是通过派生类对象   1. 公有、私有和受保护继承   某各类对其继承而来的成员的访问权限收到两个因素影响：   1. 在基类中成员的访问说明符 2. 在派生类的派生列表中的访问说明符   对于派生类的成员（及友元）（this或对象）能否访问直接基类的public,protected，private成员：  1.this   1. 派生类派生列表访问说明符没什么影响，对基类中成员的访问说明符有影响 2. 基类中成员说明符：public,protected 能访问，private就不能访问 3. 前提注意：派生类的成员或友元不能让派生类的对象是const的！不然不可访问public的成员函数，不兼容   2.对象  派生类列表访问说明符的目的是控制派生类的用户(对象)（包括派生类的派生类在内下一点讨论，主要是派生类的对象）对于基类的成员的访问权限：   1. 若派生类列表访问说明符是：   public，派生类的对象可以访问基类的publc，protected  protected,派生类的对象不可以访问基类的public，protected  private，派生类的对象不可以访问基类的public，protected   1. \*就是若D私有继承B ，D d，这样的d对象不可访问基类的public成员，但是若在D内部使用D d可以访问基类的public成员(上一点)！ 2. private，将基类的public成员私有化了，所以派生类的对象不可访问基类的public 3. protected，将基类的public变为protected的，所以派生类的对象不可访问基类的protected   3.派生类的派生类  派生访问说明符还可以控制继承自派生类的新类的访问权限：   1. 继承private的派生类的新派生类无法访问间接基类的public，protect成员 2. 因为private将基类的protected与public成员进行私有化了，相当于重新有了成员的访问说明符   因为private的直接基类将其私有了  若访问说明符是protected或者public则可以   1. 派生类向基类转换的可访问性。就是Base\*b=new Derived; 2. 由该转换的代码决定（位置），同时派生类的访问说明符也有影响 3. 只有D公有的继承B时   用户代码才能使用派生类向基类的转换(在main中或其它地方)，若是私有或受保护的则不能   1. 不论D以什么方式继承B，D的成员函数和友元都能让派生类向基类的转换，是永远可行的。 2. 若D公有或者受保护的继承B，对于D的“派生类E”，E的成员和友元可以使用D（或E）向B的类型转换。反之，如果D继承B的方式是私有的，则不能使用 3. note:在上述的3种情况某个给定节点的代码中，如果基类的公有成员可以访问的，则派生类向基类的转换是可行的，反之不可。嗯，注意派生类的对象不可访问派生类的protected与private成员。   关键概念：类的设计与受保护的成员  1.不考虑继承,一个类有两种不同的用户  1）普通用户：负责类的对象，只能访问类的公有接口成员  2）类的实现者：负责编写类的成员和友元的代码，能访问类的公有，也可以私有  2.考虑继承，第三种用户，派生类  则多了一个受保护的成员  1）普通用户：不能访问受保护的成员  2）类的实现者：可以访问受保护，但不能访问private  3.基类应该将其接口成员为公有的  实现的部分分两组：  一组可供派生类访问,应该声明protected  一组由基类及基类的友元访问，应该声明private   1. 友元与继承 2. 友谊关系不能传递，同样不能继承。 3. 基类的友元不可访问派生类成员，派生类的友元也不能随意访问基类的成员 4. 每个类负责控制自己的成员的访问权限。 5. 派生类内嵌的基类部分也受基类部分控制，即上面的右边例子 6. 友元关系不能继承！ 7. 改变个别成员的可访问性 8. D私有继承B   则D的派生类不可访问B的成员，若将B的成员重新定义则D的派生类或用户就可以访问B的成员   1. 使用方法：   public:  using B::a;  note：派生类只能为那些它可以访问的名字提供using声明,不能对private，重新public   1. 默认的继承保护级别 2. 不声明派生类访问说明符，则class与struct默认实现 3. class：private 4. struct：public 5. struct与class的差别就是默认成员访问说明符及默认派生访问说明符   note：应该显示的声明访问说明符不去依赖默认的，因为这样清晰明了 | |