|  |  |
| --- | --- |
| 1. 类型转换与继承 2. 静态类型与动态类型 3. 不存在从基类向派生类的隐式类型转换 4. 在对象之间不存在类型转换 | 动态类型  Bulk\_quote bulk;  Quote \*itemP = &bulk;  Bulk\_quote \*bulkP =itemP;  Dynamic\_cast  Static\_cast  切掉 |
| 1. 类型转换与继承 2. 如果想把引用或指针绑定到一个对象上，则引用或指针的类型应该与对象的类型一致   或者对象的类型含有一个可接受const类型转换规则。   1. 继承关系是一个重要的例外：可以把基类的指针或引用绑定到派生类对象上 2. 重要含义：当使用基类的引用（或指针）时，实际上我们并不清楚该引用（或指针）所绑定的真实类型   note：和内置指针一样，智能指针也支持派生类向基类的类型转换   1. 静态类型与动态类型 2. 静态类型与动态类型区分开来。 3. 表示静态类型在编译时时已知的，变量声明时的类型或表达式生成的类型 4. 动态类型：直至运行时才可知，变量或表达式表示的内存的对象的类型 5. 如果表达式既不是引用也不是指针，则它的动态类型与静态类型一致。 6. 不存在从基类向派生类的隐式类型转换 7. 简单的说：就是派生类的指针或引用指向基类，不是隐式的 8. 一个基类的对象既可以独立的形式存在，也可以作为派生类对象的一部分存在 9. 因为基类的对象可能是派生类对象的一部分，也可能不是，所以不存在基类向派生类的自动类型转换。 10. 即使一个基类指针或引用绑定在一个派生类对象上，我们也不能执行从基类向派生类的转换编译器检查静态类型来推断该转换是否合法。Bulk\_quote &b = base;,但是是安全的 11. 如果基类中含有虚函数，基类向派生类转换就是安全的,可以用dynamic\_cast来控制运行时执行。 12. 若我们已知基类向派生类的转换是安全的，static\_cast强制覆盖编译器的检查工作 13. 在对象之间不存在类型转换   Chil cl;  Father fa(cl);//调用父类的拷贝构造函数  fa = cl;//调用父类的拷贝赋值函数  派生类向基类的自动类型转换只对指针或引用类型有效，在派生类类型和基类类型之间不存在这样的转换。  但不过确实有将派生类对象转换成它的基类类型，但是与我们期望的有所差别   1. 初始化或赋值一个类类型的对象时，实际调用的是某个函数 2. 初始化时调用构造函数，赋值时调用赋值运算符，通常包含一个const引用 3. 因为接受引用，所以派生类向基类类型转换时允许我们给基类的拷贝、移动操作传递一个派生类的对象 4. 给基类的构造函数传递一个派生类对象时，运行的构造函数是基类中定义的那个 5. 同样赋值的话，运行的也是基类中定义的那个是将派生类中基类的那一部分给基类，但是派生类自己的成员则忽略了，称为被切掉了。因为切掉了，不能访问派生类的成员   note:派生类对象为一个基类对象初始化或赋值时，只有该基类对象中的基类部分会被拷贝、赋值、移动，它的派生类部分会忽略掉了  关键概念：存在继承关系的类型之间的类型转换  1.派生类向基类的类型转换只对指针或引用有效  2.基类向派生类不存在隐式类型转换  3.派生类向基类的类型转换也可能由于访问受限而变得不可行  可以将一个派生类对象拷贝、赋值 移动给一个基类对象，但只处理派生类对象的基类部分。  即：  Bulk\_quote bulk3;  Quote item2(bulk3);  item2 = bulk3;  cout << item2.val1 << endl;  //cout << item2.val2 << endl;// 不能val2  dynamic\_cast与static\_cast区别  1.派生类指针或引用指向基类的操作，而基类又指向另一个派生类，即派生类先向上转的基类再用强制转换向下转给派生类  基类必须有虚函数才可以执行dynamic，static没有要求  dynamic\_cast与static\_cast一样,不为空，派生类和基类的成员都默认初始化了或者构造函数初始化  2.若是直接用基类的对象给派生类指针或引用  dynamic\_cast，为空的指针或引用，无法创建  static\_cast,不是空的指针或引用，但是对于基类的数据成员会进行初始化，但是对于派生类的成员不会进行默认初始化，不会实例派生类成员所以未定义的数据 | |