|  |  |
| --- | --- |
| 1. 编写Basket类 2. 隐藏指针 3. 模拟虚拷贝 | Virtual Quote\* clone() const &  {  Return new Quote(\*this);  }  Virtual Quote\* clone() const &&  {  Return new Quote(move(\*this));  } |
| 1. 编写Basket类 2. 无法直接使用对象进行面向对象编程。必须使用指针和引用。 3. 但指针会增加程序的复杂性，所以经常定义一些辅助的类来处理这种复杂情况   定义一个表示购物篮的类   1. 代码细节：   items.upper\_bound(\*be))，upper\_bound 同样关键字的下一位置，要获取关键字，所以需解引用得到具体的对象值  \*\*be 是一个quote对象，\*be得到智能指针，\*\*be得到对象，是一个动态的对象  count,统计有多少个元素的健值相同   1. 隐藏指针 2. basket的用户还是要处理动态内存。若想basket的用户不处理动态内存，则需在basket中重新add\_item的定义，它来处理动态内存   bsk.add\_item(make\_shared<Quote>(“123”,34));   1. 重新定义后：   void add\_item(const Quote &sale);// 拷贝给定的对象  void add\_item(Quote &&sale);// 移动给定的对象   1. 但是存在一个问题，当定义add\_itme时：   bs.add\_item(Bulk\_quote("aaa", 20, 3, 0.5));// main中  items.insert(shared\_ptr<Quote>(new Quote(sale)));// item方法里  这里new Quote(sale),若sale是bulk\_quote类型，只是把sale中的基类部分给Quote，并不能获取一个bulk\_quote的指针，会被切掉一部分，只能得到基类的指针  模拟虚拷贝   1. 为了解决上述问题，给Quote类添加一个虚函数，该函数获取一份当前对象的拷贝 2. 需要定义左值和右值版本，右值版本中虽然sale是右值引用类型，但是sale本身是左值，所以需要std::move来把一个右值引用绑定到sale上 3. sale的动态类型决定到底运行quote函数还是bulk\_quote函数，无论拷贝还是移动，clone都返回一个新分配对象的指针，把shared\_ptr绑定到这个对象上，然后insert。   items.insert(shared\_ptr<Quote>(sale.clone()));   1. shared\_ptr支持派生类向基类的类型转换，所以shared\_ptr<Quote>可以绑定到Bulk\_quote\*上 | |