|  |  |
| --- | --- |
| 1. 使用RTTI 2. 类的层次关系 3. 类型敏感的相等运算符 4. 派生类虚equal函数 5. 基类equal函数 | 使用typeid(lhs) == typeid(rhs) &&lhs.equal(rhs);  Auto r = dynamic\_cast<const Derived&>(rhs) |
| 1. 使用RTTI 2. 想为具有继承关系的类实现相等运算符。比较两个对象的每个成员是否相等 3. 半对的思路： 4. 在继承体系中各个层次分别执行相等性判断，为基类的引用定义一个相等运算符(整体)，工作委托给动态类型的虚函数equal 5. 但是，由于虚函数需要形参一致，那么当基类定义的equal形参是Base&，那么派生类的equal形参也是Base&，equal只能使用基类的成员，这样就不能比较派生类独有的成员 6. 理清思路： 7. 需要两个对象类型相同 8. 在派生类equal虚函数的形参基类的引用得转换这个派生类的对象才能使用派生类独有的成员 9. 那么可以在整体引用中，先比较两个对象类型是否相同再继续使用equal虚函数 10. 在equal虚函数将基类的引用转换为当前类，才能使用派生类独有的成员，那么再比较每一个成员 11. 基类的equal虚函数不用转换，直接比较每个成员 12. 类的层次关系 13. 类型敏感的相等运算符 14. 就是使用typeid(lhs)==typeid(rhs) && lhs.equals(rhs) 15. typeid(lhs)==typeid(rhs) 先判断动态类型是否相等，才进行下一步   3) lhs.equals（rhs）执行lhs动态类型的虚函数  4.派生类虚equal函数  1) 需要将类型dynamic\_cast成自己的类型在判断成员是否相等  5.基类equal函数  只需简单的判断成员是否相等  代码:  Derived d1(1,"123"),d2(1, "123");  bool bln = d1 == d2;  cout << bln << endl;  Base\* b1 = new Derived(1,"12");  Base\* b2 = new Derived(1, "12");  bool bln2 = (\*b1 == \*b2);  cout << bln2 << endl;  Base\* b1 = new Base(1, "12");  Base\* b2 = new Base(2, "12");  bool bln3 = (\*b1 == \*b2);  cout << bln3 << endl; | |