C 箴言：理解隐式接口和编译期多态 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/220/2021_2022_C___E7_AE_B4_E8_A8_80_EF_c67_220271.htm 计算机等级考试训练软件《百宝箱》object-oriented programming（面向对象编程）的世界是围绕着 explicit interfaces（显式接口）和 runtime polymorphism（执行期多态）为中心的。例如，给出下面这个（没有什么意义的）的 class（类）。 class Widget { public: Widget(). virtual ~Widget(). virtual std::size_t size() const. virtual void normalize(). void swap(Widgetamp. w){ if (w.size() > 10 amp. w != someNastyWidget) { Widget temp(w). temp.normalize(). temp.swap(w). }} 我们可以这样谈论 doProcessing 中的 w： 因为 w 被声明为 Widget 类型的引用，w 必须支持 Widget interface（接口）。我们可以在源代码中找到这个 interface（接口）（例如，Widget 的 .h 文件）以看清楚它是什么样子的，所以我们称其为一个 explicit interface（显式接口）它在源代码中显式可见。 因为 Widget 的一些 member functions（成员函数）是虚拟的，w 对这些函数的调用就表现为 runtime polymorphism（执行期多态）：被调用的特定函数在执行期基于 w 的 dynamic type（动态类型）来确定（参见《C 箴言：绝不重定义继承的非虚拟函数》）。 templates（模板）和 generic programming（泛型编程）的世界是根本不同的。在那个世界，explicit interfaces（显式接口）和 runtime polymorphism（执行期多态）继续存在，但是它们不那么重要了。作为替代，把 implicit interfaces（隐式接口）和 compile-time polymorphism（编译期多态）推到了前面。为了了解这是怎样一种情况，

看一下当我们把 doProcessing 从一个 function（函数）转为一个 function template（函数模板）时会发生什么：templatevoid doProcessing(Tamp.&amp. w != someNastyWidget) { T temp(w). temp.normalize(). temp.swap(w). }} 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com