C 箴言:用成员函数模板接受兼容类型 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/464/2021_2022_C___E7_AE_B4_E8_A8_80__c67_464893.htm smart pointers（智能指针）是行为很像指针但是增加了指针没有提供的功能的 objects。例如，《C 箴言：使用对象管理资源》阐述了标准 auto_ptr 和 tr1::shared_ptr 是怎样被应用于在恰当的时间自动删除的 heap-based resources（基于堆的资源）的。STL containers 内的 iterators（迭代器）几乎始终是 smart pointers（智能指针）；你绝对不能指望用 " " 将一个 built-in pointer（内建指针）从一个 linked list（线性链表）的一个节点移动到下一个，但是 list::iterators 可以做到。 real pointers（真正的指针）做得很好的一件事是支持 implicit conversions（隐式转换）。derived class pointers（派生类指针）隐式转换到 base class pointers（基类指针）， pointers to non-const objects（指向非常量对象的指针）转换到 pointers to const objects（指向常量对象的指针），等等。例如，考虑在一个 three-level hierarchy（三层继承体系）中能发生的一些转换： class Top { ... }. class Middle: public Top { ... }. class Bottom: public Middle { ... }. Top *pt1 = new Middle. // convert Middle* => Top* Top *pt2 = new Bottom. // convert Bottom* => Top* const Top *pct2 = pt1. // convert Top* => const Top* 在 user-defined smart pointer classes（用户定义智能指针类）中模仿这些转换是需要技巧的。我们要让下面的代码能够编译： template class SmartPtr { public: // smart pointers are typically explicit SmartPtr(T *realPtr). // initialized by built-in pointers ... }. SmartPtr pt1 = // convert SmartPtr => SmartPtr (new

Middle). // SmartPtr SmartPtr pt2 = // convert SmartPtr => SmartPtr (new Bottom). // SmartPtr SmartPtr pct2 = pt1. // convert SmartPtr => // SmartPtr 在同一个 template（模板）的不同 instantiations（实例化）之间没有 inherent relationship（继承关系），所以编译器认为 SmartPtr 和 SmartPtr 是完全不同的 classes，并不比（比方说）vector 和 Widget 的关系更近。为了得到我们想要的在 SmartPtr classes 之间的转换，我们必须显式地为它们编程。 在上面的 smart pointer（智能指针）的示例代码中，每一个语句创建一个新的 smart pointer object（智能指针对象），所以现在我们就集中于我们如何写 smart pointer constructors（智能指针的构造函数），让它以我们想要的方式运转。一个关键的事实是我们无法写出我们需要的全部 constructors（构造函数）。在上面的 hierarchy（继承体系）中，我们能从一个 SmartPtr 或一个 SmartPtr 构造出一个 SmartPtr，但是如果将来这个 hierarchy（继承体系）被扩充，SmartPtr objects 还必须能从其它 smart pointer types（智能指针类型）构造出来。例如，如果我们后来加入 class BelowBottom: public Bottom { ... }. 我们就需要支持从 SmartPtr objects 到 SmartPtr objects 的创建，而且我们当然不希望为了做到这一点而必须改变 SmartPtr template。 大体上，我们需要的 constructors（构造函数）的数量是无限的。因为一个 template（模板）能被实例化而产生无数个函数，所以好像我们不需要为 SmartPtr 提供一个 constructor function（构造函数函数），我们需要一个 constructor template（构造函数模板）。这样的 templates（模板）是 member function templates（成员函数模板）（常常被恰如其分地称为 member templates（成

员模板））生成一个 class 的 member functions（成员函数）的 templates（模板）的范例： template class SmartPtr { public: template // member template SmartPtr(const SmartPtr &amp. other). // for a "generalized ... // copy constructor" }. 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