C 箴言：如何访问模板化基类中的名字 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/245/2021_2022_C___E7_AE_B4_E8_A8_80_EF_c97_245965.htm 假设我们要写一个应用程序，它可以把消息传送到几个不同的公司去。消息既可以以加密方式也可以以明文（不加密）的方式传送。如果我们有足够的信息在编译期间确定哪个消息将要发送给哪个公司，我们就可以用一个 template-based（模板基）来解决问题： class CompanyA { public: ... void sendCleartext(const std::stringamp. msg). ... }. class CompanyB { public: ... void sendCleartext(const std::stringamp. msg). ... }. ... // classes for other companies class MsgInfo { ... }. // class for holding information // used to create a message template class MsgSender { public: ... // ctors, dtor, etc. void sendClear(const MsgInfoamp. info) // similar to sendClear, except { ... } // calls c.sendEncrypted }. 这个能够很好地工作，但是假设我们有时需要在每次发送消息的时候把一些信息记录到日志中。通过一个 derived class（派生类）可以很简单地增加这个功能，下面这个似乎是一个合理的方法： template class LoggingMsgSender: public MsgSender { public: ... // ctors, dtor, etc. void sendClearMsg(const MsgInfoamp. msg). ... }. 一般的 MsgSender template（模板）不适用于 CompanyZ，因为那个模板提供一个 sendClear function（函数）对于 CompanyZ objects（对象）没有意义。为了纠正这个问题，我们可以创建一个 MsgSender 针对 CompanyZ 的特化版本： template // a total specialization of class MsgSender { // MsgSender. the same as the public: // general template, except ... // sendCleartext is omitted

void sendSecret(const MsgInfo&amp. info) { ... } }. 注意这个 class definition（类定义）开始处的 "template " 语法。它表示这既不是一个 template（模板），也不是一个 standalone class（独立类）。正确的说法是，它是一个用于 template argument（模板参数）为 CompanyZ 时的 MsgSender template（模板）的 specialized version（特化版本）。这以 total template specialization（完全模板特化）闻名：template（模板）MsgSender 针对类型 CompanyZ 被特化，而且这个 specialization（特化）是 total（完全）的只要 type parameter（类型参数）被定义成了 CompanyZ，就没有剩下能被改变的其它 template ’s parameters（模板参数）。