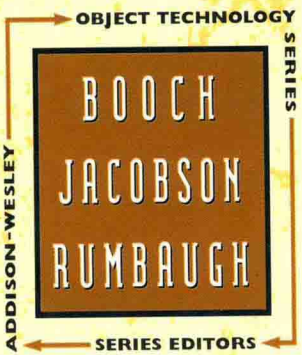
UML与面向对象设计影印丛书

COM高手心经

EFFECTIVE COM

50 WAYS TO IMPROVE YOUR COM AND MTS-BASED APPLICATIONS

DON BOX
KEITH BROWN
TIM EWALD
CHRIS SELLS
编著



科学出版社
www.sciencep.com

UML与面向对象设计影印丛书

COM高手心经

Don Box Tim Ewald
Keith Brown Chris Sells 编著

科学出版社
北京

内 容 简 介

COM 是面向对象分布式应用程序开发中重要的中间层技术。本书作者根据自己多年 COM 实际开发的经验和心得，总结出了 50 条重要的规律，并归纳为 6 大类：从 C++到 COM 的过渡，接口及 COM 开发基本要素，实施问题，apartment 有关概念，安全性，事务。另外，还通过网站提供了书中的代码。

本书适合有经验的 C++、COM 及 MTS 程序开发人员阅读。

ISBN 0-201-37968-6

图字：01-2003-2546

图书在版编目(CIP)数据

COM 高手心经=Effective COM: 50 Ways to Improve Your COM and MTS-based Applications/（美）博克斯（Box,D.）等编著.—影印本. —北京：科学出版社，2003

ISBN 7-03-011404-3

Ⅰ.C… Ⅱ.博… Ⅲ.软件接口，COM—程序设计—英文 Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字（2003）第 030821 号

策划编辑：李佩乾/责任编辑：李佩乾
责任印制：吕春珉/封面制作：东方人华平面设计室

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2003 年 5 月第 一 版 开本：787×960 1/16
2003 年 5 月第一次印刷 印张：14 3/4
印数：1—2 000 字数：281 000

定价：30.00 元

（如有印装质量问题，我社负责调换<环伟>）

影印前言

随着计算机硬件性能的迅速提高和价格的持续下降，其应用范围也在不断扩大。交给计算机解决的问题也越来越难，越来越复杂。这就使得计算机软件变得越来越复杂和庞大。20 世纪 60 年代的软件危机使人们清醒地认识到按照工程化的方法组织软件开发的必要性。于是软件开发方法从 60 年代毫无工程性可言的手工作坊式开发，过渡到 70 年代结构化的分析设计方法、80 年代初的实体关系开发方法，直到面向对象的开发方法。

面向对象的软件开发方法是在结构化开发范型和实体关系开发范型的基础上发展而来的，它运用分类、封装、继承、消息等人类自然的思维机制，允许软件开发者处理更为复杂的问题域和其支持技术，在很大程度上缓解了软件危机。面向对象技术发端于程序设计语言，以后又向软件开发的早期阶段延伸，形成了面向对象的分析和设计。

20 世纪 80 年代末 90 年代初，先后出现了几十种面向对象的分析设计方法。其中，Booch, Coad/Yourdon、OMT 和 Jacobson 等方法得到了面向对象软件开发界的广泛认可。各种方法对许多面向对象的概念的理解不尽相同，即便概念相同，各自技术上的表示法也不同。通过 90 年代不同方法流派之间的争论，人们逐渐认识到不同的方法既有其容易解决的问题，又有其不容易解决的问题，彼此之间需要进行融合和借鉴；并且各种方法的表示也有很大的差异，不利于进一步的交流与协作。在这种情况下，统一建模语言(UML)于 90 年代中期应运而生。

UML 的产生离不开三位面向对象的方法论专家 G. Booch、J. Rumbaugh 和 I. Jacobson 的通力合作。他们从多种方法中吸收了大量有用的建模概念，使 UML 的概念和表示法在规模上超过了以往任何一种方法，并且提供了允许用户对语言做进一步扩展的机制。UML 使不同厂商开发的系统模型能够基于共同的概念，使用相同的表示法，呈现彼此一致的模型风格。1997 年 11 月 UML 被 OMG 组织正式采纳为标准的建模语言，并在随后的几年中迅速地发展为事实上的建模语言国际标准。

UML 在语法和语义的定义方面也做了大量的工作。以往各种关于面向对象方法的著作通常是以比较简单的方式定义其建模概念，而以主要篇幅给出过程指导，论述如何运用这些概念来进行开发。UML 则以一种建模语言的姿态出现，使用语言学中的一些技术来定义。尽管真正从语言学的角度看它还有许多缺陷，但它在这方面所做的努力却是以往的各种建模方法无法比拟的。

从 UML 的早期版本开始，便受到了计算机产业界的重视，OMG 的采纳和大公司的支持把它推上了实际上的工业标准的地位，使它拥有越来越多的用户。它被广泛地用

于应用领域和多种类型的系统建模，如管理信息系统、通信与控制系统、嵌入式实时系统、分布式系统、系统软件等。近几年还被运用于软件再工程、质量管理、过程管理、配置管理等方面。而且它的应用不仅仅限于计算机软件，还可用于非软件系统，例如硬件设计、业务处理流程、企业或事业单位的结构与行为建模，等等。

在 UML 陆续发布的几个版本中，逐步修正了前一个版本中的缺陷和错误。即将发布的 UML2.0 版本将是对 UML 的又一次重大的改进。将来的 UML 将向着语言家族化、可执行化、精确化等理念迈进，为软件产业的工程化提供更有力的支撑。

本丛书收录了与面向对象技术和 UML 有关的 12 本书，反映了面向对象技术最新的发展趋势以及 UML 的新的研究动态。其中涉及对面向对象建模理论研究与实践的有这样几本书：《面向对象系统架构及设计》主要讨论了面向对象的基本概念、静态设计、永久对象、动态设计、设计模式以及体系结构等近几年来面向对象技术领域中的新的理论知识与方法；《用 UML 进行用况对象建模》主要介绍了面向对象的需求阶段、分析阶段、设计阶段中用况模型的建立方法与技术；《高级用况建模》介绍了在建立用况模型中需要注意的高级的问题与技术；《UML 面向对象设计基础》则侧重于经典的面向对象理论知识的阐述。

涉及 UML 在特定领域的运用的有这样几本：《UML 实时系统开发》讨论了进行实时系统开发时需要对 UML 进行扩展的技术；《用 UML 构建 Web 应用程序》讨论了运用 UML 进行 Web 应用建模所应该注意的技术与方法；《面向对象系统测试：模型、视图与工具》介绍了将 UML 应用于面向对象的测试领域所应掌握的方法与工具；《对象、构件、框架与 UML 应用》讨论了如何运用 UML 对面向对象的新技术——构件-框架技术建模的方法策略。《UML 与 Visual Basic 应用程序开发》主要讨论了从 UML 模型到 Visual Basic 程序的建模与映射方法。

介绍面向对象编程技术的有两本书：《COM 高手心经》和《ATL 技术内幕》，深入探讨了面向对象的编程新技术——COM 和 ATL 技术的使用技巧与技术内幕。

还有一本《Executable UML 技术内幕》，这本书介绍了可执行 UML 的理念与其支持技术，使得模型的验证与模拟以及代码的自动生成成为可能，也代表着将来软件开发的一种新的模式。

总之，这套书所涉及的内容包含了对软件生命周期的全过程建模的方法与技术，同时也对近年来的热点领域建模技术、新型编程技术作了深入的介绍，有些内容已经涉及到了前沿领域。可以说，每一本都很经典。

有鉴于此，特向软件领域中不同程度的读者推荐这套书，供大家阅读、学习和研究。

北京大学计算机系　蒋严冰 博士

Preface

The evolution of the Component Object Model (COM) has in many ways paralleled the evolution of C++. Both movements shared a common goal of achieving better reuse and modularity through refinements to an existing programming model. In the case of C++, the preceding model was procedural programming in C, and C++'s added value was its support for class-based object-oriented programming. In the case of COM, the preceding model was class-based programming in C++, and COM's added value is its support for interface-based object-oriented programming.

As C++ evolved, its canon evolved as well. One notable work in this canon was Scott Meyers' *Effective C++*. This text was perhaps the first text that did not try to teach the reader the basic mechanics and syntax of C++. Rather, *Effective C++* was targeted at the working C++ practitioner and offered 50 concrete rules that all C++ developers should follow to craft reasonable C++-based systems. The success of *Effective C++* required a critical mass of practitioners in the field working with the technology. Additionally, *Effective C++* relied on a critical mass of supporting texts in the canon. At the time of its initial publication, the supporting texts were primarily *The C++ Programming Language* by Stroustrup and *The C++ Primer* by Lippman, although a variety of other introductory texts were also available.

The COM programming movement has reached a similar state of critical mass. Given the mass adoption of COM by Microsoft as well as many other development organizations, the number of COM developers is slowly but surely approaching the number of Windows developers. Also, five years after its first public release, there is finally a sufficiently large canon to lay the tutorial groundwork for a more advanced text. To this end, *Effective COM* represents a homage to Scott Meyers' seminal work and attempts to provide a book that is sufficiently

approachable that most working developers can easily find solutions to common design and coding problems.

Virtually all existing COM texts assume that the reader has no COM knowledge and focus most of their attention on teaching the basics. *Effective COM* attempts to fill a hole in the current COM canon by providing guidelines that transcend basic tutorial explanations of the mechanics or theory of COM. These concrete guidelines are based on the authors' experiences working with and training literally thousands of COM developers over the last four years as well as on the communal body of knowledge that has emerged from various Internet-based forums, the most important of which is the DCOM mailing list hosted at `DCOM-request@discuss.microsoft.com`.

This book owes a lot to the various reviewers who offered feedback during the book's development. These reviewers included Saji Abraham, David Chappell, Steve DeLassus, Richard Grimes, Martin Gudgin, Ted Neff, Mike Nelson, Peter Partch, Wilf Russell, Ranjiv Sharma, George Shepherd, and James Sievert. Special thanks go to George Reilly, whose extensive copyediting showed the authors just how horrible their grammar really is. Any errors that remain are the responsibility of the authors. You can let us know about these errors by sending mail to `effectiveerrata@develop.com`. Any errata or updates to the book will be posted to the book's Web page, `http://www.develop.com/effectivecom`.

The fact that some of the guidelines presented in this book fly in the face of popular opinion and/or "official" documentation from Microsoft may at first be confusing to the reader. We encourage you to test our assertions against your current beliefs and let us know what you find. The four authors can be reached *en masse* by sending electronic mail to `effectivecom@develop.com`.

Intended Audience

This book is targeted at developers currently using the Component Object Model and Microsoft Transaction Server (MTS) to develop software. *Effective COM* is not a tutorial or primer; rather, it assumes that the reader has already tackled at least one pilot project in COM and has been humbled by the complexity and breadth of distributed object computing. This book also assumes that the reader is at least somewhat familiar with the working vocabulary of COM as it is described in *Essential COM*. The book is targeted primarily at developers who work in C++; however, many of the topics (e.g., interface design, security, trans-

actions) are approachable by developers who work in Visual Basic, Java, or Object Pascal.

What to Expect

The book is arranged in six chapters. Except for the first chapter, which addresses the cultural differences between "100% pure" C++ and COM, each chapter addresses one of the core atoms of COM.

Shifting from C++ to COM

Developers who work in C++ have the most flexibility when working in COM. However, it is these developers who must make the most adjustments to accommodate COM-based development. This chapter offers five concrete guidelines that make the transition from pure C++ to COM-based development possible. Aspects of COM/C++ discussed include exception handling, singletons, and interface-based programming.

Interfaces

The most fundamental atom of COM development is the interface. A well-designed interface will help increase system efficiency and usability. A poorly designed interface will make a system brittle and difficult to use. This chapter offers 12 concrete guidelines that help COM developers design interfaces that are efficient, correct, and approachable. Aspects of interface design discussed include round-trip optimization, semantic correctness, and common design flaws.

Implementations

Writing COM code in C++ requires a raised awareness of details, irrespective of the framework or class library used to develop COM components. This chapter offers 11 concrete guidelines that help developers write code that is efficient, correct, and maintainable. Aspects of COM implementation discussed include reference counting, memory optimization, and type-system errors.

Apartments

Perhaps one of the most perplexing aspects of COM is its concept of an apartment. Apartments are used to model concurrency in COM and do not have analogues in most operating systems or languages. This chapter offers nine concrete guidelines that help developers ensure that their objects operate properly in a multithreaded environment. Aspects of apartments discussed include real-world lock management, common marshaling errors, and life-cycle management.

Security

One of the few areas of COM that is more daunting than apartments is security. Part of this is due to the aversion to security that is inherent in most developers, and part is due to the fairly arcane and incomplete documentation that has plagued the security interfaces of COM. This chapter offers five concrete guidelines that distill the security solution space of COM. Aspects of security discussed include access control, authentication, and authorization.

Transactions

Many pages of print have been dedicated to Microsoft Transaction Server, but precious few of them address the serious issues related to the new transactional programming model implied by MTS. This chapter offers eight concrete pieces of advice that will help make your MTS-based systems more efficient, scalable, and correct. Topics discussed include the importance of interception, activity-based concurrency management, and the dangers of relying on just-in-time activation as a primary mechanism for enhancing scalability.

Acknowledgments

First and foremost, Chris would like to thank his wife, Melissa, for supporting him in his various extra-circular activities, including this book.

Thanks to J. Carter Shanklin and the Addison Wesley staff for providing the ideal writing environment. I couldn't imagine writing for another publisher.

Thanks to all of the reviewers for their thoughtful (and thorough) feedback.

Thanks to all my students as well as the contributing members of the DCOM and ATL mailing lists. Whatever insight this book provides comes from discussing our mutual problems with COM.

Last but not least, thanks to my fellow authors for their hard work and diligence in seeing this project through to the end. It is truly a pleasure and an honor to be included as an author with professionals of such caliber.

Don would like to thank the three other Boxes that fill up his non-COM lifestyle.

Thanks to my coauthors for sharing the load and waiting patiently for me to finish my bits and pieces (the hunger strike worked, guys).

A tremendous thanks to Scott Meyers for giving us his blessing to leverage[1] his wildly successful format and apply it to a technology that completely butchers his life's work.

Thanks to all of my cohorts at DevelopMentor for tolerating another six months of darkness while I delayed yet another book project.

Thanks to J. Carter Shanklin at Addison Wesley for creating a great and supportive environment.

Thanks to the various DCOM listers who have participated in a long but fun conversation. This book in many ways represents an executive summary of the megabytes of security bugs, MTS mysteries, and challenging IDL puzzles that have been posted by hundreds of folks on the COM front lines.

A special thanks goes to the Microsoft folks who work on COM and Visual C++, for all of the support over the years.

Keith would like to thank his family for putting up with all the late nights. The joy they bring to my life is immeasurable.

Thanks to Don, Tim, and Chris, for thinking enough of me to extend an invitation to participate in this important project.

Thanks to Mike Abercrombie and Don Box at DevelopMentor for fostering a home where independent thought is nourished and the business model is based on honesty and genuine concern for the community.

Thanks to everyone who participates in the often lengthy threads on the DCOM list. That mail reflector has been incredibly useful in establishing a culture among

[1] Leverage is a nice-sounding euphemism used in the Windows development world, whose real meaning should be obvious.

COM developers, and from that culture has sprung forth a wealth of ideas, many of which are captured in this book.

Thanks to Saji Abraham and Mike Nelson for their dedication to the COM community.

Thanks, Carter, this book is so much better than it possibly could have been if you had pressed us for a deadline.

And finally, thanks to all the students who have participated in my COM and security classes. Your comments, questions, and challenges never cease to drive me toward a deeper understanding of the truth.

First and foremost, Tim would like to thank his coauthors for undertaking this project and seeing it through to completion. As always, gentlemen, it's been a pleasure.

Also, thanks to friends and colleagues Alan Ewald, Owen Tallman, Fred Tibbitts, Paul Rielly, everyone at DevelopMentor, students, and the participants on the DCOM mailing list for listening to me go on and on about COM—nodding sagely, laughing giddily, or screaming angrily as necessary.

A special thanks to Mike, Don, and Lorrie for suffering through the earliest days of DM to produce an extraordinary environment for thinking.

And, of course, thanks to my family: Sarah for letting me wear a COM ring too, Steve and Kristin for reminding me about the true definition of success, Alan and Chris for allowing me to interrupt endlessly to ask geeky questions, and Nikke and Stephen Downes-Martin for accepting phone calls from any airport I happen to be in.

Finally, thank you J. Carter Shanklin and Addison Wesley for letting us do our own thing.

Chris Sells
Portland, OR
August 1998
http://www.sellsbrothers.com

Don Box
Redondo Beach, CA
August 1998
http://www.develop.com/dbox

Keith Brown
Rolling Hills Estates, CA
August 1998
http://www.develop.com/kbrown

Tim Ewald
Nashua, NH
August 1998
http://www.develop.com/tjewald

Contents

Shifting from C++ to COM

Moving from pure C++ development to the world of COM can seem especially constraining. Many of the language constructs you have come to know and love are yanked from your arsenal and replaced with a whole new set of language constructs called attributes in a language that looks closer to C than C++. The mindset of a C++ developer is typically focused on implementing objects. It takes a considerable amount of time before one's focus shifts to thinking in terms of components that communicate through request and response messages. This chapter discusses several of the more important attitude shifts that are needed to survive the transition from "100% pure" C++ to the stylized subset we have come to know as the Component Object Model.

1. Define your interfaces before you define your classes (and do it in IDL).

One of the most basic reflexes of a C++ programmer is to begin the coding phase of a project in a "dot-H" file. It is here that the C++ programmer typically begins defining both the public operations of his or her data types as well as their core internal representations. When working on an exclusively C++-based project, this is a completely reasonable approach. However, when working on a COM-based project, this approach usually leads to pain and suffering.

The most fundamental concept in COM is that of separation of interface from implementation. Although the C++ programming language supports this style of programming, it has very little *explicit* support for defining interfaces as separate entities from the classes that implement them. Without such explicit support for interfaces, it is easy to blur the distinction between interface and implementation.

It is common for novice COM developers to forget that interfaces are intended to be abstract definitions of some functionality. This implies that the definition of a COM interface should not betray implementation details of one particular class that implements the interface. Consider the following C++ class definition:

```
class Person {
  long m_nAge;
  long m_nSalary;
  Person *m_pSpouse;
  list <Person*> m_children;
public:
  Person(void);
  void Marry(Person& rspouse);
  void RaiseSalary(long nAmount);
  void Reproduce(void);
  Person *GetSpouse(void) const;
  long GetAge(void) const;
  long GetSalary(void) const;
  const list<Person*>& GetChildren(void) const;
};
```

This is a completely reasonable class definition; however, if it were used as the starting point for a COM interface definition, the most direct mapping would look like this:

```
DEFINE_GUID(IID_IPerson, 0x30929828, 0x5F86, 0x11d1,
        0xB3, 0x4E, 0x00, 0x60, 0x97, 0x5E, 0x6A, 0x6A);
DECLARE_INTERFACE_(IPerson, IUnknown) {
  STDMETHOD(QueryInterface)(THIS_ REFIID r,void**p)PURE;
  STDMETHOD_(ULONG, AddRef)(THIS) PURE;
  STDMETHOD_(ULONG, Release)(THIS) PURE;
  STDMETHOD_(void, Marry)(THIS_ IPerson *pSpouse) PURE;
  STDMETHOD_(void, RaiseSalary)(THIS_ long nAmt) PURE;
  STDMETHOD_(void, Reproduce)(THIS) PURE;
  STDMETHOD_(IPerson *, GetSpouse)(THIS) PURE;
  STDMETHOD_(long, GetAge)(THIS) PURE;
  STDMETHOD_(long, GetSalary)(THIS) PURE;
  STDMETHOD_(list<IPerson*> *, GetChildren)(THIS) PURE;
};
```

The DECLARE_INTERFACE_ macros are used to emphasize that this interface definition is meant to appear in a C/C++ header file.

The first deficiency of this interface definition is that it uses a standard template library (STL) list to return the collection of children. While this method is reasonable in a closed single-binary system in which all constituent source code will be compiled and linked as an atomic unit, this technique is fatal in COM. Since in COM the component may be built with one compiler and used by client code compiled with a different compiler, it is impossible to guarantee that both the client and the object will agree on the representation of an STL list. Even if both entities are compiled with the same vendor's compiler, it is not guaranteed that the same version of the STL is in use.[1]

A more obvious problem related to returning an STL list is that this interface potentially betrays an implementation detail of the underlying class, namely, that it stores its collection of children using an STL list. For the implementation shown earlier, this is not an issue. However, if other class implementers wish to implement the `IPerson` interface, they too must store their children in STL lists or create a new list every time the `GetChildren` method is called. Since STL lists are not the most space-efficient representation for a collection, this constraint imposes an undue burden on all implementations of `IPerson`.

One last flaw related to data types has to do with the results of each method. The definition of each of the `IPerson` methods uses the `STDMETHOD_` macro that allows the interface designer to indicate the physical result type of the method explicitly. Thus, the following method definition,

```
STDMETHOD_(long, GetAge)(THIS) PURE;
```

will expand as follows once the C preprocessor performs its magic:

```
virtual long __stdcall GetAge(void) = 0;
```

The problem with this method is that it does not return an HRESULT. As detailed in Item 2, COM overloads the physical result of the method to indicate communication failures; since this method does not return an HRESULT, COM has no way to inform the client of any communication errors that may occur.

None of the flaws noted previously would have occurred had the developer defined the interface in COM's Interface Definition Language (IDL). It is tempting to look at IDL and think, Why must I master yet another language? This concern is valid. Most developers older than age 25 have already mastered (at least) one programming language and are reluctant to learn yet another syntax for writing conditionals and loops. This reluctance is understandable, because every decade new programming languages are produced that promise vast increases in programmer productivity but often turn out to be syntactic monstrosities.

[1] Many developers eschew their compiler's STL implementation in favor of higher-performance versions available from third-party vendors.