开 发 大 师 系 列

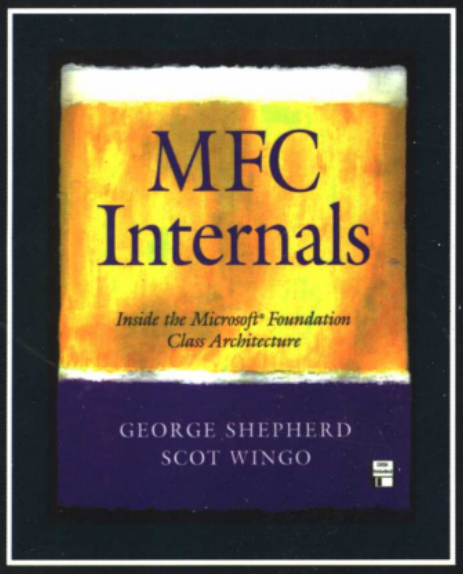
Addison Wesley

MFC Internals

Inside the Microsoft Foundation Class Architecture

深入解析MFC

[美] George Shepherd, Scot Wingo 著
赵剑云 卿瑾 译



非常简单，本书是所有严谨的MFC开发人员的必备之作。

——Dean McCrory，MFC开发组负责人

中国电力出版社
www.infopower.com.cn

开 发 大 师 系 列

MFC Internals

Inside the Microsoft Foundation Class Architecture

深入解析MFC

[美] George Shepherd, Scot Wingo 著
赵剑云 卿瑾 译

中国电力出版社

MFC Internals: Inside the Microsoft Foundation Class Architecture
(ISBN 0-201-40721-3)
George Shepherd, Scot Wingo

北京市版权局著作权合同登记号 图字：01-2002-0713 号

图书在版编目（CIP）数据

深入解析 MFC /（美）谢泼德，（美）温勾著；赵剑云，卿瑾译. —北京：中国电力出版社，2003
（开发大师系列）
ISBN 7-5083-1800-5
Ⅰ.深... Ⅱ.①谢...②温...③赵...④卿... Ⅲ.C 语言－程序设计 Ⅳ.TP312
中国版本图书馆 CIP 数据核字（2003）第 069624 号

责任编辑：乔晶
丛 书 名：开发大师系列
书　　名：深入解析MFC
编　　著：(美)George Shepherd, Scot Wingo
翻　　译：赵剑云　卿瑾
出版发行：中国电力出版社
地址：北京市三里河路6号　邮政编码：100044
电话：（010）88515918　　传真：（010）88518169
本书如有印装质量问题，我社负责退换
印　　刷：北京丰源印刷厂
开　　本：787×1092　1/16　　**印　　张**：33.75　　**字　　数**：765千字
书　　号：ISBN 7-5083-1800-5
版　　次：2003年10月北京第一版
印　　次：2003年10月第一次印刷
定　　价：59.00 元

前　言

自从 MFC 1.0 于 1990 年发布以来，微软基础类（Microsoft Foundation Class）（现在是 4.0 版了）已经发展成为性能完全、健壮、口碑甚佳、使用广泛而且非常复杂的应用程序框架。对于我们这些一直跟着它发展的人，凭借使用以前版本 MFC 的历史和经验还能帮着减轻一点复杂度。而对于今天刚开始学习 MFC 的新手，学起来就太困难了。进入 MFC 的天堂需要不懈的努力——使用 MFC 写代码，并且大量地阅读。

当然，Visual C++（和其他获得 MFC 许可的 C++产品）都带有介绍 MFC 库的参考文档和概念信息。也有很多第三方写的书讲授如何使用 MFC 创建 Windows 应用程序。然而，所有这些知识源都把焦点集中在如何使用 MFC 上：调用什么、重载什么、怎样激活特定的特性。在几个流行的在线论坛上，已经就是否有必要在文档之上理解 MFC 的实现这个问题展开过几次有趣的讨论。我的观点是，要成为一个真正的 MFC 专家，你必须理解 MFC 的实现。对任何问题的最终回答常常是在源代码本身中找到的。这也是 MFC 提供源码的主要原因之一。

实际上，阅读源代码是我开始学习 MFC 的方法。我这样做部分是因为有必要，部分是出于我的天性。我介入 MFC 是从它的一个内部 Alpha 版开始的，这还是在 Microsoft C/C++ 7.0 开发过程中发布的。当时还没有 MFC 的文档，甚至 CodeView 也还不能处理 C++代码。文档的匮乏没有带给我很大麻烦，因为我很想知道一切到底是怎么工作的。我一直问类似这样的问题："MFC 在处理 Windows 消息时怎么调用我的 C++成员函数？"，"MFC 的异常处理是如何实现的？"（当时 Microsoft C++的编译器还不支持异常处理）。因此，我开始浏览代码，试图理解它。但是，是不是每个使用 MFC 的人都必须这么做？是不是每个人都有这么多时间？可能不是——有的人需要享受生活！

即使你有足够的时间来研究一摞一摞的 MFC 源代码，这本书也可以引导你了解哪里有些什么。Scot 和 George 做了相当出色的工作，分析了 MFC 的源码，在本书中对 MFC 的很多关键的实现细节提供了解释。实际上，很多次我曾经试图写这样的一本书，但是我总是忙于挖掘 MFC 的新版本，没有这么多时间来创作这样一本书。幸运的是，我还有时间看完这本书，并且提供一点我的看法。

本书绝对不是对现有文档的一个重新组织，也不是一本教你如何去做的书，而是一本告诉你 MFC 是如何运作的书。很多章节都以类似这样的话开始："下面的声明已经去掉了所有有文档说明的成员函数，我们将注意力集中在没有文档说明的实现细节上。"这些实现细节会令你明白某个类内在的一些性质和限制，即使这些可能在类的公共接口中不出现。这些细节以及它们如何影响你的应用程序的知识会影响你在自己代码中使用这些类的方式。当你真正

坐下来写一个应用程序时，这些都是非常重要的细节。此外，有些细节可能只是为了趣味或展示一些你可能希望在自己的程序中使用的 C++技术（或者是你要避免的技术）。不过作者并没有揭示所有有趣的东西，有些东西留给读者自己去挖掘。在读了本书中的大量分析之后，你应该已经有能力自己探究这里没讲到的主题。掌握这个技能十分重要——很多在流行的在线服务上问到的问题（甚至在微软内部问到的问题）都可以通过简单地浏览 MFC 代码而得到解答。

简而言之，本书是所有严谨的 MFC 开发人员的必备之作。实际上，微软内部的 MFC 开发、质量保证、用户培训小组的所有新老成员都应该读一读这本书。

Dean D. McCrory

致　谢

Scot Wingo：首先感谢读者，感谢你有兴致读这样一本书。这本书写出来就像一股冲击波。感谢 Claire Horne，因为他签约雇用了一个没有任何写书和 MFC 经验的人（开玩笑的）。感谢 McCrory 院长，因为他敏锐的洞察力，他有趣的 AFX 组笑话，当然还有他提供的原始技术反馈。

尤其要感谢我的狗 Mack，因为每写一页的时候，它都给予我道义上的支持，而且还在空闲时间和我一起玩飞盘。

最后感谢我的伙伴 Stingrays，因为他是一个出色的同伴；感谢我的父母，因为他们生育了我；感谢 1203 Belle Mead 的那些人，因为好吃的红辣椒餐；也感谢可口可乐公司，因为它发明并分发了 Mello Yello——写本书过程中的最佳提神饮料；感谢 Superchunk、Small、StereoLab 和 Juliana Hatfield，因为他们写出了伟大的曲调；感谢 Dad、Todd McFarlane、John Walker、Frank Miller 和 Jim Lee，因为他们是企业家角色的榜样；最后感谢 George，因为他让我参与这个计划，从而登台成为一位大师。

George Shepherd：每当我读一本关于软件和编程的书，就不可避免地看到书中写着类似“感谢我的配偶，因为在我写这本书时她容忍着我的离开”这样的话。我过去认为他们是在开玩笑，或者只是善意地这样做。但是，现在我不这样想了。完成这本书是我所做过的事情中最艰难的一件。确实，我应该感谢我的妻子 Sandy，因为每当我消失在办公室埋头苦干这个项目的时候，她都给予了很大的耐心。另外 Sandy 也是一个很好的评论家，她帮助我澄清那些令人混淆的文字。既然现在我有了时间，从现在开始，我要为自己离开的时间和她给予我的帮助回报她。

感谢我的儿子 Teddy，因为他理解我为什么有时不能陪他去动物园。

感谢 Don Box，因为他以一种易于理解的方式向我解释 OLE；感谢 Mary Kirtland，因为她帮助我使得这个项目可以正常运作；感谢 Andrew Schulman，因为他提供了反馈意见并且推荐了能接收本书的 Addison-Wesley；感谢 Jeff Duntemann，因为他发表了我的前两篇文章，帮助我开始写作；感谢 J.D.Hildebrand 和 Oakley Publishing 的人们，因为他们帮助过我鼓励过我；感谢 Addison-Wesley 的 Claire Horne，因为在我们因仔细推调敲而影响进度（至少我们没有按她喜欢的速度写出来）的时候她仍然保持了极大的耐心；感谢 Dean McCrory，因为他编辑了本书的技术部分；感谢我的兄弟 Patrick，因为他是一个关于想法的宣传者；感谢我的父母，因为他们激励我去尽最大努力做到最好。最后我还要感谢 Scot Wingo，因为他是一个伟大而又饱含激情的写作伙伴。

简　介

这是又一本关于 MFC（Microsoft Foundation Class，Microsoft 基础类）的书，里面又写了些什么呢？

当你在喜爱的书店浏览 C++和 Windows 方面的书时，你会发现有很多书都是关于 Microsoft Visual C++和 MFC 的。其中大部分书籍都集中介绍了向导代码生成工具，并包含了使用 MFC 的基本方法。这本书是不同的——你不会发现一本和它相似的书。它的不同之处就在于它是关于 MFC 内幕的。它会告诉你 MFC 是如何整合在一起，以及在表象后面发生的事件。

为什么需要本书？

为什么你需要一本关于 MFC 内幕的书？为了回答这个问题，首先让我们来看一个典型的 MFC 编程例子。

使用 AppWizard，点击某些按钮，做出一些选择，在 10 分钟之内，你就可以得到一个 Windows 应用程序。这个应用程序支持 MDI（Multiple Document Interface，多文档界面）和 OLE（Object Linking and Embedding，对象链接和嵌入），还有一个浮动的工具栏、一个状态栏、打印对话框、打印预览对话框和关于对话框。在利用 MFC 编程的早期，事实上你已经成为超级程序员——比一个快速的 C 程序员更快，而且能够在一个小范围内跳过 OLE。

下面你决定修改应用程序内容，从而完成一些类似于处理多文档类型的事情：例如，增加一个 GIF/BMP 图像浏览器。你感觉像突然撞到一堵砖墙。MFC 将文档链表放在哪里了呢？将它们的类型添加到“打开文件”对话框需要做什么呢？你的第一反应可能是检查 MFC 的文档，但你会发现里面没有任何这样的细节内容。幸运的是，微软将 MFC 的源代码封装在类库中，所以你现在要做的就是在类库中查找它并弄清楚，对吗？

错，很快你就会意识到 MFC 的代码十分复杂，仅是一个单独的源文件都有超过 120000 行的代码，还不算头文件和.H 扩展文件。所以，你要再花一周左右的时间增加一个新特性，而在你掌握了 AppWizard 经验后只花几分钟就可以完成。现在你已经失去了那种神速超级程序员的感觉，你需要花上几个白天（或晚上）专心理解 MFC 的源代码并学习有关如何让 MFC 按照你的意愿为你服务的知识。

有些情况下你需要更多地了解 MFC 内幕。这里给出了你在用 MFC 编程时可能已经遇到的几个例子：

- 当你调试应用程序时，会经常停在 MFC 的源代码中，其中都是你所不熟悉的类和没有文档说明的内部结构。知道这些没有文档说明的 MFC 结构正在做什么，并知

道它们如何影响了你所跟踪的问题不是更好吗？

- 在 MFC 中，许多类都是用来继承的。换句话说，你不需要直接创建它们的实例，但需要创建一个派生类，并将这个派生类实例化。在这一过程中，你必须覆盖基类的一些虚函数，这样才能在你的派生类中获得一些需要的行为。问题就在于 MFC 的文档没有清晰地说明何时你应该完全替代函数、何时你应该第一次调用这个函数以及何时增加自己的代码。例如，当创建 CDialog 的一个派生类时，需要重载 OnInitDialog()来完成一些初始化工作，那么你是应该先调用 CDialog::OnInitDialog()呢？还是完全替代这个函数而不去理会那些冲突呢？显然，要做这个决定，你必须理解 CDialog::OnInitDialog()都做了些什么——只有清楚了这些才能使得是否调用基类的函数变得一目了然。
- MFC 对 OLE 的支持非常完备，但如果你还想要扩展它，或者写自己的 COM 对象，那么就要放弃已有记录的 MFC OLE 类的安全性，还必须在自己的类中指出 MFC OLE 代码。

本书要解决的问题

尽管 MFC 隐藏了细节，并很好地组织了 Windows 代码的功能，但是它没有削减了解其内部的需要。如果你想要创建热门的商用应用程序，程序不只是弹出一些对话框那么简单，你就必须知道 MFC 是如何工作的。有许多工作组致力于研究 MFC 内幕。其中许多还不十分明显，有一些甚至没有记录。这些也就是我们这本书所要探讨的。

我们已经搜索了 MFC 的源代码，以求找到能帮助解决 MFC 问题的有用信息。我们已经发现了很多有趣的内容，例如没有文档说明的类、功能函数和数据成员，有趣的 C++技术，有用的代码技术和若干关于各种 MFC 类如何工作以及它们如何互相合作的细节。我们将总结所有的关键事实和至关重要的发现，以便于你能快速地解决在 MFC 编程中所遇到的问题（再次成为超级程序员！）。

本书读者对象

本书是为那些决定使用 MFC 2.0 或更高版本的、认真的 Windows 开发人员而编写的（我不十分关心你是否认真，只要你想学）。

我们对读者假设以下两件事情：

1. 理解 C++。MFC 是 C++的类库，所以你需要掌握 C++工作的方式。你必须能够读懂这些代码，遵从语法，并理解底层的面向对象编程的封装、继承和多态性等基本思想。MFC 在它的实现中十分依赖这些概念。按照这种方法，如果我们找到任何 MFC 使用的十分高级或有趣的 C++技术，我们就将它指出，以便你能使用这种技术。

2. 从开发的角度看，你必须掌握基本的 Windows。你需要理解一些概念，例如 window 句柄、消息、设备环境以及矩形等等。

如何使用本书

本书关注服务于 Windows 95 和 Windows NT 的 32 位 MFC 4.0。如果你所使用的 MFC 的版本较低，请不必担心，因为经过这么多年，大多数的内部概念都没有发生变化。如果有的发生了变化，我们会附带一个关于版本的提示。

贯穿本书，我们介绍了那些在标准文档中漏掉的重要概念。我们还指出了那些没有文档说明的类、特性和成员函数等等，我们还会告诉你如何在你的应用程序中使用它们 www.infopower.com.cn 中有一些关于如何使用 MFC 的示例。

本书的第一部分（第 1 章~ 第 8 章）集中讨论了核心的 Windows 图形用户界面类和支持类。在第二部分（第 9 章~ 第 14 章），我们介绍了对基本 Windows 支持的扩展内容，如 OLE。最后，在第 15 章，我们给出了 MFC 的高级内幕。

我们设计这本书的目标有很大弹性。你必须一页一页地读这本书，或者将它作为一本参考书。无论哪种情况，如果你从来没有看过 MFC 的源代码，那么附录 A 可能是一个好的起点。它回顾了 MFC 源代码的组织，还包含了一些微软的编码方针。

如果你是一个 MFC 初级程序员，你可能想从第 1 章开始，然后在读其他部分之前先读第 2 章和第 5 章。第 2 章阐述 MFC 如何封装 Windows，第 5 章讨论大多数 MFC 类的基类 CObject 如何工作，以及它在原有的 Windows 基础上增加了哪些内容。

如果你是一个 MFC 中级程序员，你可以简单地复习前 5 章，然后直接进入你最感兴趣的题目。例如，如果你需要一些关于如何实现 MFC 文档/视图结构的信息，就可以直接跳进第 7 章。

最后，如果你是一个高级 MFC 程序员，你可能想要阅读那些包含你尚未了解的内幕的章节。

警　告

微软没有将这里所指出的内容归档也是有原因的，他们希望这些内容能够发生变化。事实上，在写书的过程中，我们必须修改很多章节，因为 MFC 变化得太快（一年发布 4 次）。如果你决定在自己的应用程序中使用我们给出的任何内幕，请注意要使用注释，或者使用#ifdef，这样在新版本的 MFC 破坏你的用法时，你能够知道究竟发生了什么变化。例如，在 MFC 以前的版本中，文档和视图的链表保存在文档模板中。在 4.0 版本中，微软将这些链表移动到新类 CDocManager 中。如果你已经使用了以前链表的位置，你就不得不重写这段代码以使用新的类。

联系作者

我们已经尽了最大努力提供那些适时的实际信息。如果你发现书中有问题，或者你有什么意见以及想要讨论某个话题，你可以访问我们的网页 http://www.stingsoft.com/mfc_internals 并反馈给我们。这个站点还拥有对本书的修改和其他一些有趣的 MFC 站点的链接。如果你不能通过网络访问，还可以通过 e-mail：mfc_internals@stingsoft.com 联系我们。我们会尽快地回复你。

现在，冒险开始了。

目　录

CHAPTER 1

MFC 的概念性总括

你如果理解了 MFC 设计者的意图，就会更清楚地看到 MFC 的整体框架结构。掌握了它的框架结构，你就能更好地利用 MFC 的性能。这一章从提供一个 MFC 的高层的概念性总括开始，介绍了 MFC 的一些历史。我们首先回顾一下 OOP（Object-Oriented Programming，面向对象编程）的基本原则和 MFC 的设计目标（如果你很熟悉 OOP 的原则，就可以直接跳到“应用程序框架与 MFC”一节）。然后我们会展现一副更大的图像：MFC 的组件以及这些基本部分是如何共同工作的。

面向对象编程的一些背景

目前，似乎每个软件包的广告都宣称它们的程序中使用了对象。当然，其中一些确实是面向对象的，而另一些却不是。但说一个程序是面向对象的意味着什么呢？MFC 是面向对象的。为了更彻底地理解 MFC，你需要理解 OOP 的基本原理，包括抽象、封装、继承、多态性和模块化等概念。本节将介绍 OOP 的总括性的背景和一些面向对象的概念。

结构化设计、分析和编程在 20 世纪 70 年代中期到 80 年代后期很流行。它是每个人从计算机课程中所学到的。它也帮助实现了无数的系统，其中有许多至今还在使用。但随着软件开发者所处理的工程越来越大，结构化技术逐渐显露出它的缺点。这并不是说它们已经没有用了，只是目前更大、更复杂的系统需要新的技术和概念来辅助控制系统的复杂性。OOP 可以对现实世界中不可能使用结构化技术解决的问题建模。因而，许多开发人员开始使用面向对象分析、设计和编程来帮助他们管理这样庞大而复杂的系统。

面向对象编程术语

通常用来描述一个面向对象系统的术语是抽象、封装、继承、多态性和模块化。一种编程语言只有支持了这些特性才能称之为面向对象编程语言。下面给出了每个概念的细节描述。

抽象（abstraction）

抽象定义了一个对象区别于其他对象的重要特性，它与用户的观点有关。抽象从外部观察对象，忽略了那些关于对象是如何实现的细节。抽象还为对象类之间的交互定义了约定和

协议。选择适当的抽象或类是面向对象设计的主要目标。

封装（encapsulation）

封装就是将代码和数据组合成一个整体的能力。抽象和封装二者紧密结合。没有封装，就不能定义任何用于对现实世界对象建模的抽象。在 C++中，封装的单元是类（class）和结构（struct）。理想情况下，应该能在没有看到对象的实现前使用对象。了解对象的实现会增加复杂度，因为你必须把它看作是类的一个客户。

然而，这个世界不是理想的。正如 Grady Booch（《Object-Oriented Analysis and Design》的作者）所说：在实际情况下，一个人要研究一个类的实现从而理解它的意义需要很长时间，尤其是在缺少额外的文档时（这也就是为什么有了这本关于 MFC 的书的原因）。

继承（inheritance）

继承就是从已存在类的基础上获得新的类的能力。继承使得你能够建立类层次结构，从而使你能够重用已有的代码。在 C++中，继承一个类很简单，新的类具有基类的所有功能。一旦你有了所有的好功能在手，你就可以按照自己的需要来修改。

多态性（polymorphism）

多态性的本意就是“有很多形状”。这意味着在 OOP 中，在类层次结构中，一个类可以向上或向下共享指定的函数名，虽然每个特定的类所表现出的行为是不同的。例如，假设使用 OOP 来开发一些画各种形状图形的类。每种形状的图形有自己的绘画方法。即使每个对象的绘画方法看起来很相似，你还是希望每个图形有自己的绘画行为。你需要一个圆对象来画圆，需要一个正方形对象来画正方形。每个对象以自己的绘画方法完成绘画的能力就是多态性的例子。在 C++中，多态性是通过虚拟成员函数来完成的。虚拟成员函数是在运行时而不是在编译时与类绑定在一起的（像一般的 C 函数一样）。多态性之所以存在于 C++中，是因为究竟选择哪个类的成员函数来调用是在运行时决定的。

模块化（modularity）

虽然类形成了一个系统的组件，但它们每个都不能组成一个完整的系统。为了定义系统的体系结构，需要将这些类分成模块。模块对于控制复杂度很重要（尤其是对大系统），所以确定如何将你的类分成模块就和选择类一样难。模块化实质上是一种封装，它在你将相关的类组合在一起以提供更高层次的行为时发生。一个好的模块功能结合紧密，并提供了满足客户需求的最小接口。

通常的对象

对象就是一个有属性并展现一定行为的实体。以你每天使用的计算机为例，它就是一个对象：它有属性并且展现一定的行为。即使计算机内部的工作情况很复杂，也不防碍你使用它。在你和计算机之间有一层抽象。计算机的实现细节被安全地封装在组成机器的金属、塑

料和硅中，但你和计算机之间的交互被很清楚地定义。这也就是说机器被封装了。你发送一条消息给计算机（例如开机、按键、移动鼠标），计算机就会反馈相应的行为（它会启动、在显示器上显示一个字符、鼠标指针移动）。你的计算机还从那些在它之前出现的计算机上继承了属性和行为，这样就形成了层次结构。例如，今天在你桌子上放着的 Pentium 计算机是从原来的 80x86 机器中派生的。这样，每台新机器都基本上提供了与之前的计算机相同的特性，而它本身又有一些新的特性。PC 机的进步就形成了层次结构，即每台新机器从之前的一代继承属性。

迄今为止，大多数计算机还执行许多相同的基本命令，即使底层的实现是不同的。这又称为多态性。例如，IBM-PC 不仅仅是个人电脑的品牌。假设你的朋友有一台 Macintosh，他的 Macintosh 和 IBM 的机器共享大多数相同的属性和行为，但每台机器都以自己的方式实现这些行为。例如，Macintosh 和 IBM 兼容的 PC 都会响应鼠标的移动，但底层的实现略有不同。这也是多态性的例子。

最后，你的计算机又是由其他对象组成的。CPU、硬盘、显示器等等，它们本身也是对象。你可以认为你的计算机是由所有这些独立的对象组成的模块，这也就是模块化的例子。

对象与 C++

在软件中，对象就是一个数据实体，它知道如何对自己处理。它还是搜集相关数据和用于对数据进行操作的函数的一种方法。和现实世界一样，软件对象就是抽象的实体，它有属性、行为和清楚定义的接口。

这里有一个例子：假设你想用 C 写一个小的程序来记录人的年龄。一个解决方法是定义一个结构：

```
typedef struct _PERSON[25];
   char szName[25];
   int nAge;
} PERSON;
```

然后定义一些函数来初始化和显示一个人的数据：

```
void SetData(PERSON *pPerson, char *pszName, int nAge);
void ShowData(PERSON *pPerson);
```

注意我们都做了些什么。一方面，我们定义了人员记录，另一方面又定义了对这些数据操作的函数。它们是独立的实体。C++做了修改，它将数据和函数结合为一个实体——一个 C++类。类很像 C 中的 struct，只是它将对数据进行操作的方法也合并了进来。Person 结构可以被定义为这样一个类：

```
class Person {
   char m_szName[25];
   int m_nAge;
public:
   void SetData(char *pszName, int nAge);
   virtual void ShowData();
};
```

现在，函数就是类的一部分。SetData()和 ShowData()被称为类的成员函数。

注意：C++的结构也可以有成员函数。在 C++中，类和结构的区别就在于成员变量和成员函数在默认情况下是否可见。C++结构的成员函数和成员变量在默认情况下是可见的，而 C++类的成员变量和成员函数在默认情况下是私有的（private）。

将数据结构和函数结合在一起的能力也称为封装，它是面向对象编程语言的重要性质。一旦定义了 Person 类，并实现了它的成员函数，那么初始化和显示数据就变成创建一个类的实例并调用相应的成员函数。

```
main()
{
   Person person;
   person.SetData("Sam", 218);
   person.ShowData();
)
```

但是这并不是使用类可以做的一切。C++还允许从已有的类派生出新的类，从而创建一个和基类具有相同数据并展现相同行为的类。这一性质被称为继承性，它也是面向对象编程语言的另一个重要的性质。使用 Person 数据库的例子，假设你想创建一种新的 Person，叫做雇员（Employee）。每个 Employee 与 Person 共享相同的属性和行为，另外它还有一个新的属性是职业（occupation）。在 C++中，你可以从 Person 类派生出 Employee 类，做法如下：

```
class Employee : public Person {
   char m_szOccupation[25];
public:
   void SetData(char *pszName, int nAge, char *pszOccupation);
   virtual void ShowData();
};
```

此时，Employee 类继承了 Person 类的所有属性和行为，而且它有一个自己的新数据成员 m_szOccupation。这就是继承性的一个例子，也是面向对象语言的重要组成部分。

现在有一个新的方法可以用来显示雇员的数据。这对于解决成员变量 occupation 的显示问题是必需的。因为 Person 类不知道任何关于职业的信息，所以 Employee 类需要重新定义成员函数 ShowData()，使它能显示雇员的职业。这是多态性的一个例子，其中在类层次结构中，对象向上或向下共享函数名，层次结构中每个类根据自己的情况来实现这个函数。

当然，Employee 类需要在一些地方存在。这样就引入了模块化。在 C++中，模块化是通过头文件（header file）和源文件（source file）来完成的。头文件中包含了类定义和理解一个 C++类所必需的其他信息。具体的实现代码存放在一个或多个源文件中。在这个例子中，Employee 类的定义可以存放在一个叫 EMPLOYEE.H 的文件中，具体实现可能存放在一个叫 EMPLOYEE.CPP 的文件中。

尽管这个例子只给出了抽象、封装、继承、多态性和模块化的简单解释，但这些基本的概念是面向对象的核心。

为什么使用 OOP

使用面向对象技术开发软件能帮助构建更接近于现实世界的系统。现实世界不是由那些