国外经典教材

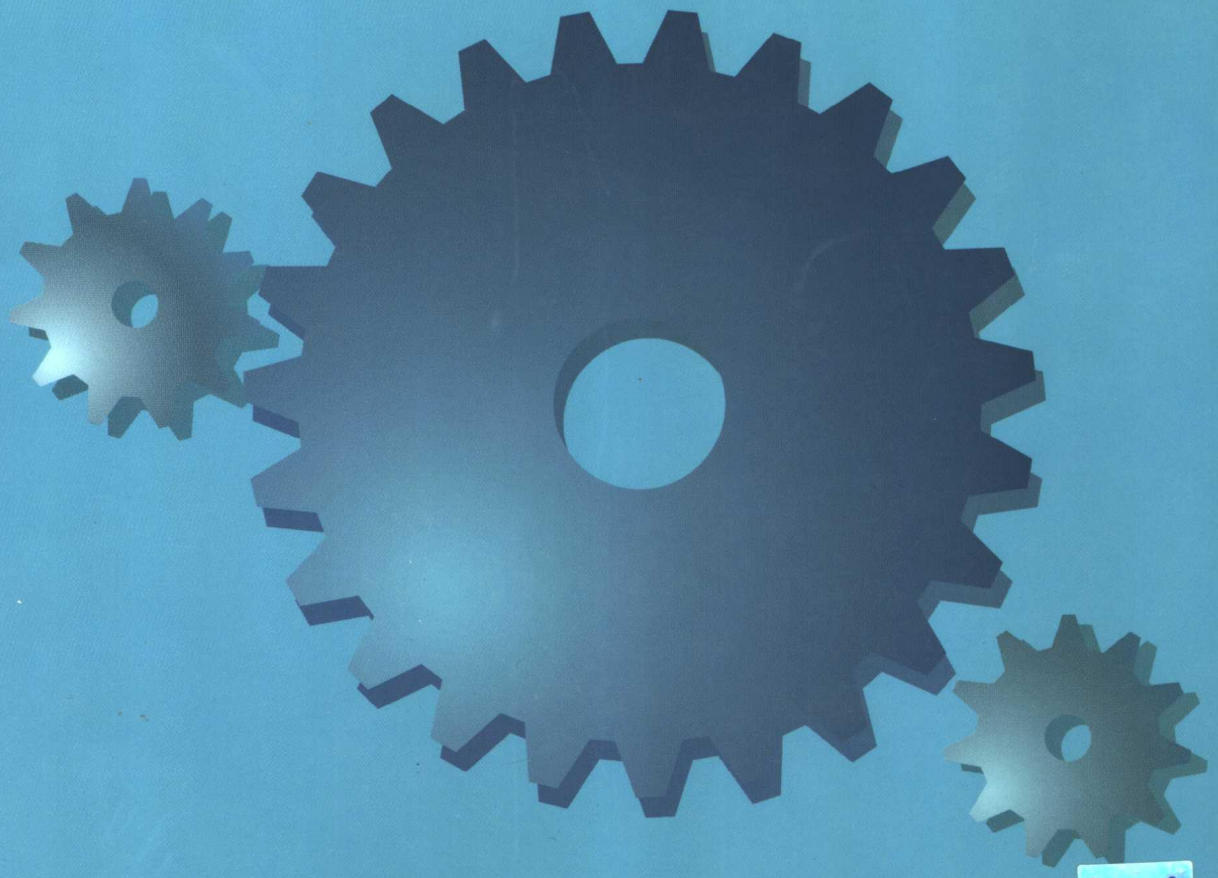
PEARSON Prentice Hall

组件级编程

COMPONENT-LEVEL PROGRAMMING

(美) Peter Maurer 著

施 诺 译



Pearson Education

清华大学出版社

国外经典教材

组 件 级 编 程

（美）Peter Maurer 著

施 诺 译

清华大学出版社

北 京

内 容 简 介

本书全面介绍了组件级编程和设计的方法，而且在这种重要的、最新的应用程序开发风格的各个方面，都为读者提供了一个坚实的基础。本书首先介绍了可视化编程，说明了如何由胶连逻辑和现有组件创建一个程序。与该问题的其他论述方法不同，本书的内容并非仅此而已。

图书在版编目（CIP）数据

组件级编程/（美）莫瑞尔著；施诺译. —北京：清华大学出版社，2003
（国外经典教材）
书名原文：Component-Level Programming
ISBN 7-302-07224-8
Ⅰ.组… Ⅱ.①莫… ②施… Ⅲ.程序设计—教材 Ⅳ.TP311.1

中国版本图书馆 CIP 数据核字（2003）第 080080 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969

文稿编辑：车立红　王松
封面设计：立日新设计公司
印 刷 者：北京市清华园胶印厂
装 订 者：三河市化甲屯小学装订二厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：26　**字数**：628 千字
版　　次：2003 年 11 月第 1 版　2003 年 11 月第 1 次印刷
书　　号：ISBN 7-302-07224-8/TP・5254
印　　数：1～3500
定　　价：49.00 元

前　言

本书以2000年秋季我在南佛罗里达大学所讲授的一门组件级设计课程为基础，该课程是面向对象设计课程的后继课程（在学习本书前，了解一些面向对象的设计知识是有益的，但并非必需）。该课程的主要学习对象是高年级的在校大学生，但也可以作为研究生的预备课程。

我们有必要开设有关组件级设计的课程，这一需求已出现了很长时间。基于组件的语言（诸如Visual Basic）已经在商业领域作为开发引擎而广泛使用，而且计算机专业的学生也需要透彻地理解这种语言。然而，有关Visual Basic（或其他一些基于组件的开发语言）的课程却并不完善。这些课程只是向学生说明如何使用现有的组件，但并没有说明如何创建自己的组件。

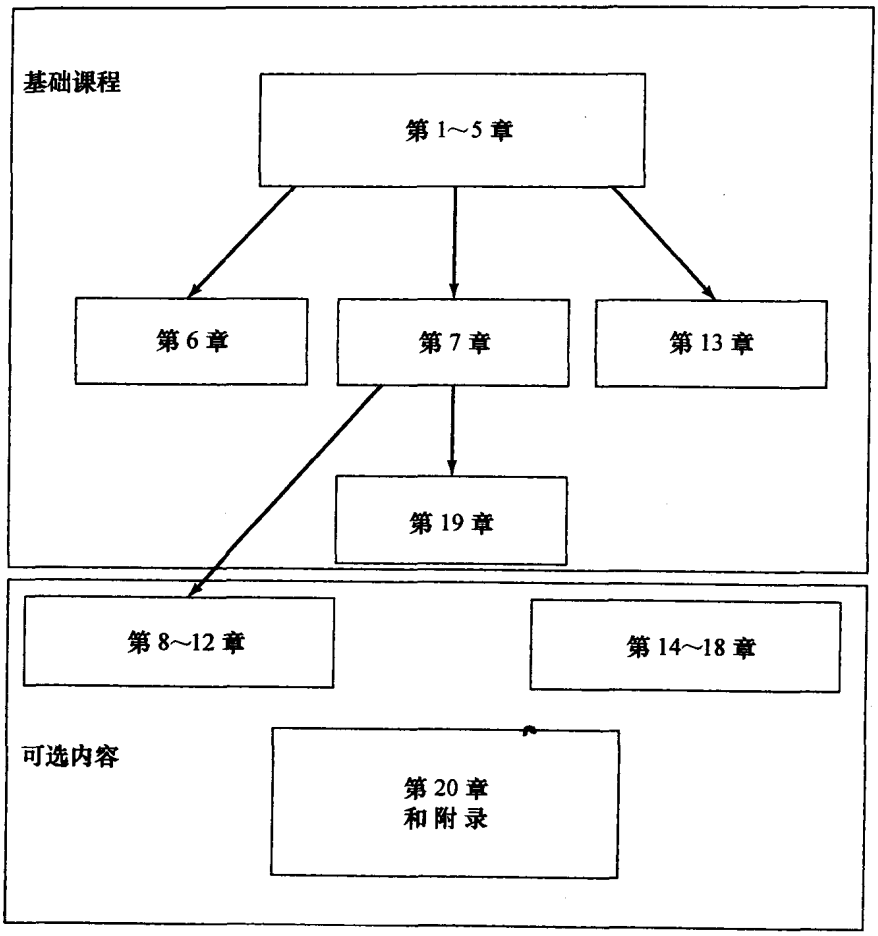
在开发新的基于组件的应用程序时，必须将组件和利用宿主语言（Visual Basic及其他开发语言）编写的代码结合起来。应当作为组件组成部分实现的功能与应当采用宿主语言编程实现的功能之间，有着本质的差别。通常情况下，虽然大家更愿意购买第三方开发的组件，而不愿意开发自己的组件，但仍有许多应用程序所需的组件并不存在。对于定制软件来说，这种趋势是一种规律，而不是个别现象，也没有理由认为这种趋势在将来会有所改变。

学生通过本书将学习到基于组件开发的其他方面内容——也就是说，组件本身的开发。本书不仅讨论了如何开发各种类型的组件，而且还讨论了如何将应用程序划分为组件和宿主级编码。

本书的第1～5章包含了各种课程所必备的基本知识。教员应该在学习后面章节前先介绍这部分内容。第6～18章相对较为独立，因而在介绍时可以不分次序。然而，第7章是其他许多章节的基础，特别是第8章、第9章和第11章内容的基础，以及第10章和第12章部分内容的基础。我建议每门课程都应当讲授第6章、第7章和第13章的内容。（因为第6章介绍的内容可用于大量有趣的学生项目，我喜欢先介绍这些内容，以使学生在开始时能够接触较大的项目）。第19章和第20章的内容适用于准备进一步学习组件级设计的学生。

第13章的内容为组件级软件与更典型的编程类型架起了一座桥梁。通过使用本章的技术，可将针对基于字符的接口（诸如MS-DOS或者UNIX外壳）开发的软件转换为组件级软件。对于其余章节，学生可以根据自己的兴趣有选择地进行学习。我已尽可能让本书的内容丰富多彩。

介绍完第7章的内容后，教员随时都可以介绍第19章的内容。实际上，这些章节是相互独立的，但如果没有一些范例，就很难生动地介绍第19章的内容。第7章和第13章提供了这些范例。下面的流程图有助于安排本书的课程。



学生们将会发现，这是一门引人入胜的课程，因为该课程允许学生开发出他们在自己计算机上用过的同类软件。此外，该课程还允许学生们开发出各种具有精美外观的软件，而不用学习 GUI 编程的所有细节知识。随着基于组件开发系统课程的普及，本书的内容也会相应地变得更重要。

Peter M. Maurer

目　录

第1章 绪 论

1.1 组件革命

计算技术的发展经历了三次革命：存储程序计算机、高级编程语言和组件级编程。前两次革命是将“猜想”变为现实的范例。许多学者团体、著名大学和研究所为此做出了重大贡献。各种原则的阐明和各项工作的开展沿着那些伟大预言家所设计的思路进行。经历过这些革命的人都清楚，世界正在注视着他们。他们知道自己正在创造历史。

与此相反，组件革命的发生几乎是偶然的。尽管目前有很多可用的组件技术，但组件革命是一个有关 VBX 及日后称为 ActiveX 控件的技术的故事，VBX 技术不是最早的或最好的技术（尽管这些观点值得商榷），但正是这项技术向世人证明了组件级编程是个不错的想法。

术语“VBX”代表 Visual Basic eXtension，它是一种为 Visual Basic 编程语言的早期版本添加新特性的机制。为了创建这种扩展，需要创建一个具有某些标准接口函数的运行函数库（run-time function library）。这些函数库存储在以“.vbx”为扩展名的文件中，因而有了 VBX 这个名字。一旦创建了 VBX，就可以通过将其复制到自己的硬盘上，并在 Visual Basic 项目文件中插入一个对该 VBX 的引用，来实现将其添加到 Visual Basic 项目中。在大多数情况下，这并不需要复杂的安装程序。一旦在项目中包含了 VBX，就可以为 Visual Basic 程序员提供一个或多个定制控件。控件在过去和现在都是 Visual Basic 编程的支柱。VB 编程的一个主要工作就是向应用程序窗口添加控件。标准控件可以从可用控件的菜单中选择，包括按钮控件、文本框控件和标签控件等。VBX 的定制控件在使用方式上与标准控件完全相同。我们可以将这些定制控件添加到应用程序窗口，通过添加这些定制控件的代码段，就可以将定制控件的功能集成到程序中。事实上，我们可以毫不费力地的创建多个互不干扰的控件实例。除了极少数情况外，在某个程序中集成不同类型的定制控件时，无需担心控件间的兼容性。Visual Basic 推出后不久，就出现了各种各样极好的定制控件，提供了从电子表格功能到字处理功能等各种类别的功能。几乎所有类似独立程序存在的东西都可以作为 VBX 定制控件的形式存在。它们并不在自己的窗口中执行控件功能，而是在 Visual Basic 应用程序的子窗口中执行功能。

VBX 编程最重要的一点是，它允许程序员在远远超过语句级编程的一个崭新层次上编写程序。为了确保程序运行，仍然需要某些语句级编程，但是程序中最复杂和细微的部分现在已经封装到一个 VBX 组件的集合中。语句级代码主要作为集成各种组件的“胶水”。很多采用基于 VBX 程序的编程实践，已经采用了基于 ActiveX 的程序和其他一些

基于组件的技术。

为了理解这次革命的重要性，必须认识到在“语句级以上”编程多年来一直是编程语言开发者的目标。传统的观点认为，语言开发者可以通过一系列增量式步骤实现此目标：从一种常规的编程语言开始，通过添加新的、更强大的原语，创建一种更为强大的编程语言。然后以新的编程语言为基础，开发另一种更为强大的编程语言。语言开发者可以按照这种方式继续下去，直至编程目标远远超过语句级编程所能达到的层次为止。

尽管这种增量式方法看起来很合理，但它从未起过任何作用。难以捉摸的“下一步”从未实现过。尽管引入了许多新原语，但其中大部分原语具有局限性。尽管创建了许多新的编程语言，其中的一些编程语言完全不同于传统的编程语言，但大多数只是昙花一现。安排在“下一步”中进行的变革没有达到它们最初的期望，或者由于过于繁杂而难以在实际中使用（功能性语言就是这种变革）。

一项有用的变革是从面向过程的编程转变为面向对象的编程。编程观点也随这种变革从增加新的原语转变为代码重用。这种观点认为程序员可以在“远远高于语句级”的层次上进行编程，不是使用新的语言原语，而是重用其他程序员编写的主要代码。最初，一些激动人心的成功案例说明这种观点的确是正确的。其中给人印象最深的成功案例就是为 MS-DOS 程序开发的图形用户接口（graphical user interface，简称 GUI）包。使用这些程序包，程序员可以在非常不适合 GUI 编程的环境中为程序开发出复杂的 GUI，而且这种开发毫不费力。遗憾的是，诸如此类的成功案例并不多见，而且 MS-DOS 应用程序的消失使得该成功案例与本书主题毫不相干。现代的 GUI 操作系统确实有面向对象的开发包，但这些包通常只是封装了现有的 GUI 功能。

这并非意味着面向对象编程不重要。事实并非如此，面向对象编程是一项重要的变革，它采用一种我们可以理解的逻辑方式组织程序，而且在程序的编写和维护上比面向过程的编程更加容易。然而，面向对象编程并没有改变编写程序的方式。它改变了我们考虑数据的方式，以及我们考虑数据和代码之间关系的看法。但在编写代码时，仍然要用赋值语句、while 语句、for 循环以及其他语言要素。尽管面向对象编程让代码重用更加容易，但代码重用仍是一个说得多用得少的概念。

随后出现了 Visual Basic 及其 VBX 组件。突然间，所有人都在重用代码，所有人都在“远远高于语句级”的层次上进行编程。程序开发的速度更快了，而且程序员获得了多于其他面向对象编程的特性。为什么会出现这种情况？答案似乎就是“VBX 组件”。

真正的问题在于，为什么 VBX 可以取得成功，而其他经过深思熟虑的方法却没有成功？这个问题的答案非常重要，因为它将深深地影响我们看待组件的方式。我们希望进行组件开发的观点是，在不损失任何现有促使 VBX 成功的技术的前提下，通过适当的启发，就可以开发出新的组件技术。更为重要的是，如果我们完全理解了 VBX 技术及其成功的原因，就会发现这项技术是那些适用范围更广、最终更有用的技术的一个子集。

1.2 工程和计算机科学

尽管我们喜欢自称为计算机科学家，但严格地讲，程序设计不是一门科学，而是一

种工程规范。程序设计并不采用科学方法，但它却采用科学原理和数学原理。程序员用所学的设计原则构造问题的解决方案。除了极为罕见的情况外，他们并不用公式来表示猜想或者做实验来证明猜想。

我个人认为，尽管没有发现“下一级”的编程语言，但我们在无意间发现了一条在其他工程规范中早已获得公认的原则。换句话说，利用中间级部件进行设计是困难的或是不可能的。当你打算组建一个车队时，你不可能从三菱公司购买发动机，从福特公司购买车门，从克莱斯勒公司购买轮胎，从通用公司购买装饰盖。如果真的这么做了，那么这些部件无法装配到一起（汽车制造商确实要向其他厂家购买部件，但只是针对那些经过仔细设计，可以使用这些部件的产品）。然而在组建车队时，你可以从福特公司购买一些汽车，再从克莱斯勒公司、三菱公司和通用公司购买其余的汽车。然后就可以用成品“组建”车队，而不是用中间级部件。

这表明我们重新发现了另一个公认的原则——也就是说，成品在更大型的设计中可以作为组件使用。在 VBX 及其他组件技术出现之前，最接近于组件的技术是子程序包（这里指广义的子程序包）。标准 C 库就是这样一个程序包。在该库中可以找到一个中等规模的部件—quicksort 函数。我曾经多次使用该函数，但总是要查参数。我发现，有时为了运行该函数必须调整函数原型；我还发现，让比较器正确运行简直就是一种魔法。我经常自言自语：“哦，这是个短整型数组，我最好自己编写一个快速插入排序的函数”。这就是说，使用中间级部件比使用普通的原语或成品部件的要求更为苛刻。

还有更为复杂的子程序包，它们提供诸如文字处理或图像分析的特性。通常说来，使用 X 程序包的应用程序必须设计成 X 程序包的应用程序类型。程序必须满足程序包的要求，因为程序包不会也不可能满足程序的要求。毫无疑问，在单个程序中结合两个或多个这样的程序包也不可行。

与子程序包相比，组件可以更容易地满足程序的要求。如果决定向程序中添加 X 组件，我们可以在开发过程的任何时刻添加该组件，而不必在开始时就添加。如何使用组件取决于我们自己。我们也不用担心由于需要满足组件的要求而重新构建该程序（大多数 Visual Basic 程序员会认为这是个荒谬的想法）。当使用组件时，在单个程序中将文字处理器、电子表格编辑器、Web 浏览器、动画视频播放器和 MP3 文件播放器结合在一起，并期望所有组件正常工作是可行的，也是非常合理的。

尽管一个复杂的子程序包能够提供与 VBX 或其他组件相同的功能，但子程序包不是一个成品，它只是某些问题的一个局部解决方案。为了创建一个成品，子程序包还需要结合其他的编码。从另一方面来说，组件是成品。即便一些简单的组件只能提供按钮或文本框，但它仍是成品；尽管组件需要容器程序提供扩展支持，但它仍是独立的程序。

基于这样的观点，我们开始分析组件级编程。

1.3 组件的定义

在本书中，术语**组件**将定义为一个独立的程序，而且该程序很容易作为其他程序的组成部分使用。这种定义最初可能难以接受，因为大多数组件只能在宿主环境中运行，

而且需要该环境提供大量的服务。这似乎同“独立程序”的称号相矛盾。然而这一点很容易解释，所有程序都对环境有设置要求，而组件所设定的要求只是程度问题。例如，一个在 UNIX 命令行中运行的 C 程序在 UNIX 外壳程序上设置要求。该外壳程序必须为该程序打开 stdin、stdout 和 stderr 文件。这种服务超出了标准操作系统的 I/O 支持、内存管理和其他运行时服务，而且并非所有在 UNIX 环境中运行的程序都要求提供这种服务。当组件对环境的要求比命令行程序对环境的要求更复杂时，这些要求可以视为操作系统服务，正如打开 stdin、stdout 和 stderr 文件是一种操作系统服务那样。

每个 Visual Basic 程序都包含 Visual Basic 运行环境。该环境提供了扩展的操作系统服务，支持组件实例化和执行、粘连代码的调度和执行以及粘连代码和组件间的通信接口。总而言之，该扩展服务集构成了 Visual Basic 的**组件框架**。每种组件技术都有自己的组件框架。正是这种组件框架的设计定义了特定的组件技术。

1.4 组件框架

与所有其他程序类似，组件必须由某个外部代理加载和执行，而且必须向组件提供 I/O 和通信服务。组件必须能够与环境、其他组件或者同时与两者进行交互。组件框架的设计定义了组件技术的基本特性。

与组件类似，我们可以从用户的角度或者开发者的角度看待组件框架。框架的**用户**是 Visual Basic 程序员或 Web 页面设计人员——也就是说，使用框架创建新应用程序的人员。**开发者**是设计和构建框架的人员。这两种看待组件的角度截然不同。在理想情况下，用户很少知道框架，而是使用简单的编程范例来建立所需的组件交互。相对而言，开发者必须完全熟悉框架和组件的实现细节。该领域中的用户简单视图必须转变为大量复杂的基本实现。

组件级编程标准为组件和框架提供了规范。例如，ActiveX 规范详细描述了某个模块必须实现的服务，该服务提供组件与环境的正常交互；同时该规范还规定了那些必须由框架实现的服务，以便和其他 ActiveX 组件交互。

组件交互有两种广义上的类型：（1）组件－容器交互，（2）组件－组件交互。一个**容器**是一个实体，它实现了组件框架并对一个或多个组件集进行维护。一个 Visual Basic 程序就是一个容器。组件－容器交互的机制已经完全实现了标准化，但组件－组件交互的机制由于一些特殊应用而有所保留。几乎没有组件－组件直接交互的通用标准。最常见的组件－组件直接交互的例子是 Visual Basic 数据控件与数据感知组件之间的通信。

在最底层，实体间相互通信有两种方式（实体既可以是组件，也可以是容器）。第一种方式是**主动通信**，其中**实体 A** 调用**实体 B** 所包含的子程序。第二种方式是**被动通信**，其中**实体 A** 修改**实体 B** 也可以访问的全局变量。实际上，所有的组件技术都采用**主动**通信，尽管用户在必要时可能会模拟**被动**通信。在主动通信中，有几类公认的子程序，列举如下：

Set_Property_P 用户认为 P 是一个变量，可作为组件内部状态的一部分进行维护。**Set_Property_P** 函数用于为 P 赋一个新值。P 可以是表示组件内

	部状态的一个真实变量；也可以是一个虚属性，其值可以在需要时进行计算。设置一个虚属性值通常会引发组件的某些复杂行为。该函数有一个参数(新的属性值)，但是没有返回值。数组属性需要有额外的参数，以便指定该数组的索引值。在某些技术中，函数返回值用于表示函数运行的成功或失败。在这种情况下，**Set_Property_P** 函数将有一个合适的返回值。
Get_Property_P	与 **Set_Property_P** 函数对应，用于读取属性值。该函数没有参数，其返回值就是属性 P 的值。如果属性是个数组，那么该函数会有一个表示数组索引的参数。
Event_X	事件机制允许组件调用另一个实体（通常是容器）的子程序。该事件子程序可以有参数，但通常没有返回值。事件的接收实体不需要实现事件子程序，此时该实体必须用一个空操作子程序进行代替。在组件技术中，事件机制通常是最复杂的交互。
Initialize_State	该子程序用一个包含初始化数据的参数进行调用，对组件的内部状态进行初始化。尽管用户可以指定初始化数据的内容，但这种机制通常对用户是不可见的。
Save_State	该子程序用一个指向容器的参数进行调用，以保存初始化数据。该数据随后将传递给 **Initialize_State** 函数。**Initialize_State** 函数和 **Save_State** 函数都不是必需的。
Method	方法是一个可以被组件用户显式调用的任意函数。该函数有输入参数、返回值或两者均有。输入值和返回值的实现通常需要框架开发人员付出大量努力。

1.5 编程模型

迄今为止，大多数有用的组件框架是那些完全支持可编程容器的框架。这些框架的实用性取决于编程模型的简单性和有效性。编程模型应该支持几种公认的特性。最基本的特性是能够创建组件实例，包括创建同一组件的多个互不干扰的实例。该模型应该支持“数据和函数”接口，这类似于面向对象编程（在组件技术中，其数据元素称为**属性**，其函数元素称为**方法**）。尽管一般认为组件实例不是对象，但编程者可以将其视为对象。例如，如果 Text 是称为 Editor 组件的一个属性，而且在程序中有两个 Editor 组件的实例（Editor1 和 Editor2），应该可以用下述语句或类似语句，将数据从一个实例传递到另一个实例：

```
Editor1.Text := Editor2.Text;
```

如果 **Editor** 组件有一个用于清除其内容的 **Clear** 方法，则在特定的实例中可用下述语句调用该方法：

```
Editor1.Clear();
```