分层策略

Peter Eeles

Rational Software 白皮书

TP 199, 08/01

Rational®
the software development company

目录

摘要

有许多技术可用于分解软件系统。分层是其中一个示例并将在本白皮书中描述。这样的技术针对两个主要问题：大多数系统太复杂以至于不能完全理解，不同用户需要系统的不同方面。

分层技术已被众多软件系统所采纳，并在许多文本以及 Rational Unified Process（RUP）中得到支持。但是，分层技术经常被误解和误用。本白皮书阐明了分层的含义并讨论了运用不同分层策略的效果。

什么是"分层"？

让我们从定义"分层"的含义开始。术语*层*指的是通常称为*"层"模式*的体系结构模式的应用，在多个文本（[Buschmann]、[Herzum]、[PloP2]）和 RUP 中有描述。*模式*表示存在于特定环境中的常见问题的解决方案。表 1 给出了层模式概述。

表 1："层"模式概述

	层模式
环境	需要分解的系统
问题	系统太复杂以至于不能完全理解 系统很难维护 系统最不稳定的元素未被隔离 系统的大多数可重用元素很难确定 系统将由可能具有不同技能的不同团队建立
解决方案	将系统结构分成层

分层的一个最熟悉的示例是由国际标准化组织（ISO）定义的 OSI 7 层模型。此模型（显示在图 1 中）定义了一组网络协议 — 每一层侧重于通信的一个特定方面并建立在下一层设施的基础之上。OSI 7 层模型使用基于职责的分层策略：每一层具有特定的职责。此策略在本白皮书后面将详细描述。

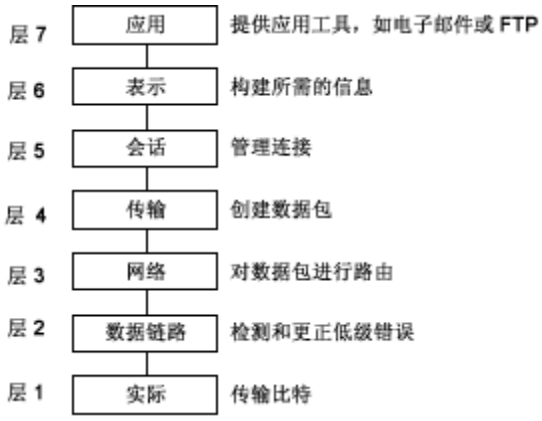

图 1：OSI 7 层模型（基于职责的分层）

图 2 显示基于职责的分层的另一个示例。

- *表示逻辑层*包含负责向人员提供某种展示形式的元素，如用户界面中的元素。
- *业务逻辑层*包含负责执行某种类型的业务处理和应用业务规则的元素。
- *数据访问逻辑层*包含负责提供对信息源（如关系数据库）的访问的元素。

应该注意的是，可用多种方法对层建模，本白皮书后面有描述。目前，我们将使用具有构造型 «层»的 UML 包来明确表示一个层。

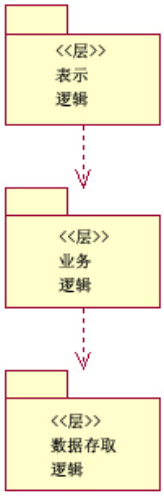

图 2：基于职责的分层

在这个基于职责分层的特定示例中，层（layer）通常被称为“级”（tier）并且是分发式系统开发中常见的概念，，其中有 *2 级*、*3 级*和 *n 级*系统。

图 2 的一个重要方面是所显示的*依赖关系的方向*，因为它暗含作为分层系统特征的某个规则 — 特定层中的元素只能访问同一层或下面各层的元素[1]。在此处给出的示例中，*业务逻辑层*中的元素不能访问*表示逻辑层*中的元素。同样地，*数据访问逻辑层*中的元素不能访问*业务逻辑层*中的元素。此结构通常被称为定向无环图（DAG）。它是*有方向的*，因为依赖关系是单向的；并且是*非循环的*，因为依赖关系的路径从没有循环。

有一点要更明确的指出，当定义分层策略时，弄清每一层的含义是很重要的，以便将元素正确放置在适当的层中。若未能将元素正确分配到适当的层将首先减少应用该策略的价值。当更详细地讨论每个分层策略时，给出关于每一层的含义的通用指南。

对层建模

当研究不同的分层策略时，将渐渐明确使用特定*模型*（因此使用特定 UML 元素）传达每个策略是合适的。 *模型*表示从特定角度对系统的完整描述。图 3 显示了四个模型的示例，分别表示基于以下考虑的系统的不同角度：

- 用例模型：捕获系统需求
- 分析模型：捕获系统需求分析
- 设计模型：捕获系统设计

[1] 虽然事件通知会产生消息，该消息从某一层中的元素发送到上层中的元素，但在此方向中不存在明确的依赖关系。

- 实施模型：捕获系统实施

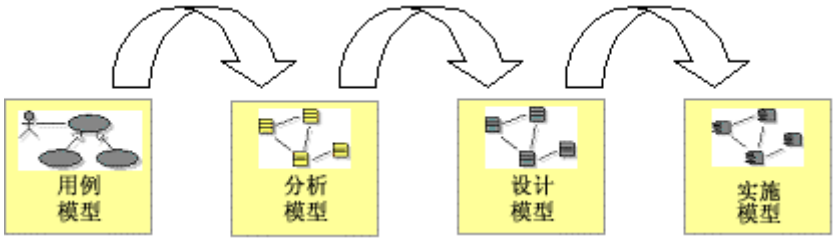

图 3：表示逐渐改进的四个模型

附加模型包括：

- *部署模型*：捕获系统的分发方面
- *数据模型*：捕获系统的持久性方面

分层策略

分层可基于多个特征。本部分讨论了基于以下特征的分层：

- 职责
- 重用

每种策略的说明将被视为对每种策略的详细讨论。

基于职责的分层

或许最常用的分层策略是基于职责的分层策略。此特定策略可以改进系统的开发和维护，因为各种系统职责是相互独立的。作为示例（请参阅图 2），可以基于以下职责对系统分层：

- 表示逻辑
- 业务逻辑
- 数据访问逻辑

这些职责中的每一个可表示为一层，如图 4 所示，该图显示了每一层的某些样本内容。在此我们考虑订单处理系统中的三个概念 — 客户、订单和产品。作为示例，*客户*概念由以下类组成：

- *CustomerView 类*：负责与客户关联的*表示逻辑*，例如在用户界面中的客户显示
- *Customer 类*：负责与客户关联的*业务逻辑*，例如客户详细信息的确认
- *CustomerData 类*：负责与客户关联的*数据访问逻辑*，例如使客户状态持久化

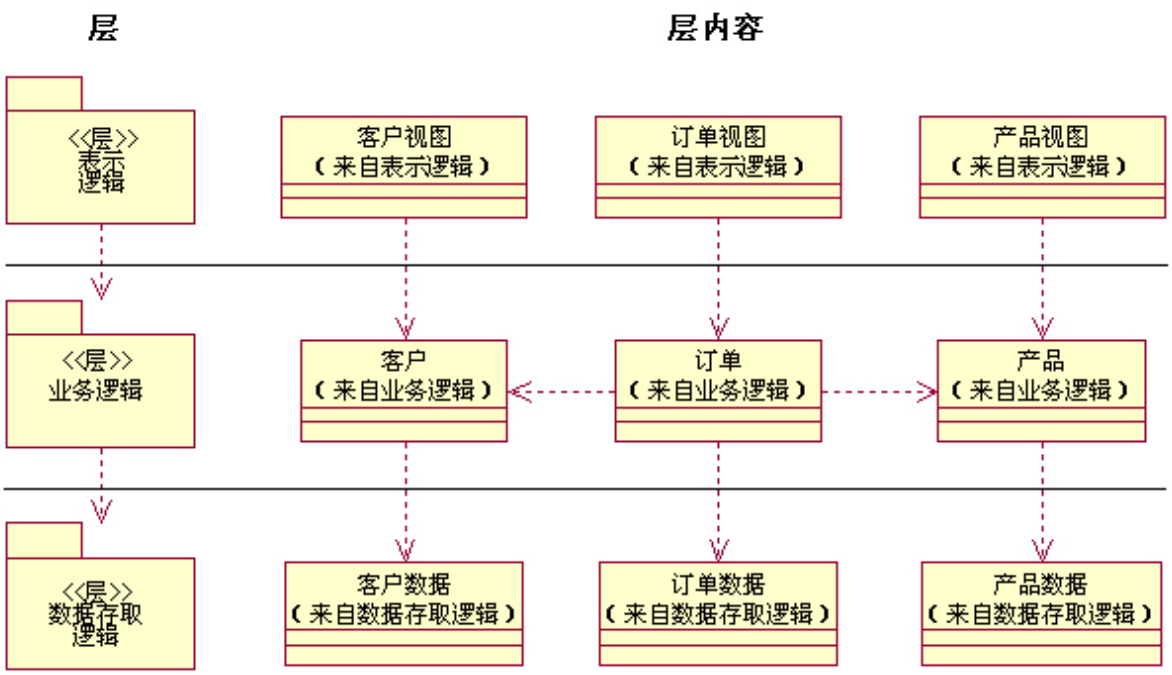

图 4：基于职责的分层的层和内容

现在我们将考虑关于此特定分层策略的某些“谬论”。

谬论 1：层和级是不同的

此特定谬论是经常遇到的混淆来源。事实是级*即是*层，虽然层是基于特定策略 — 一个职责的。混淆是由于可以多种方式应用级的概念这一事实造成的，如表 2 所示。

表 2：级定义

应用	层（级）
2 级	结合了表示逻辑和业务逻辑 数据访问逻辑
3 级	表示逻辑 业务逻辑 数据访问逻辑
n 级	表示逻辑 业务逻辑（*分发式*） 数据访问逻辑

谬论 2：层（级）表示物理分发

另一个常见的误解是逻辑分层表示物理分发。请考虑一个 3 级分层。即使各种元素将驻留其中一层中，每一级本身也可以多种方式运用，如表 3 所示，表中使用通常用于描述特定物理分发的特征的名称（如“瘦客户机”）。

表 3：3 级分层的应用

应用	层	
	客户端	服务器端
单个系统	表示逻辑 业务逻辑 数据访问逻辑	
瘦客户机	表示逻辑	业务逻辑 数据访问逻辑
胖客户机	表示逻辑 业务逻辑	数据访问逻辑

以下说法也是正确的，即单个系统可使用多个物理分发策略，其中某些元素将被归类概括为“瘦客户机”分发，而其它的是“胖客户机”分发。通常，选择是基于非功能需求（如性能）的。

对基于职责的层建模

正如我们将看到的，此策略的应用会影响*设计模型*、*实施模型和部署模型*。*设计模型*通常使用两个方法中的一种构造。

第一种方法显示“包含”在层中的元素。结果包含在图 5（Rational Rose 浏览器屏幕快照）中，它显示：

- *驻留在表示逻辑包中的演示类*（CustomerView、OrderView 和 ProductView）
- *驻留在业务逻辑包中的业务逻辑类*（Customer、Order 和 Product）
- *驻留在数据访问逻辑包中的数据访问逻辑类*（CustomerData、OrderData 和 ProductData）

图 5：包含在层中的元素

第二种方法并入了*业务组件*的概念（在本例中是客户、订单和产品）作为首要概念，由此涉及的主要元素是系统支持的与领域相关的概念。例如，概念*客户*可与表示逻辑、业务逻辑和数据访问逻辑的元素关联。此业务组件概念在 [Eeles] 和 [Herzum] 中进一步讨论。这种思考方式产生了显示在图 6 中的模型结构。在此示例中，分层是按元素名称来*表示*的。例如，所有 *View* 类（如 *CustomerView*）暗指表示逻辑层，而所有 *Data* 类（如 *CustomerData*）暗指数据访问逻辑层。未限定的类名（如 *Customer*）暗指业务逻辑层。

图 6：每个业务组件包中的*隐式*分层

也可在表示业务组件的每个包中明确表示分层，如图 7 所示。*当给定业务组件的每一层中包含多个元素时，最好使用此结构*。虽然在本示例中只扩展了客户业务组件包，但订单和产品包具有类似的结构。

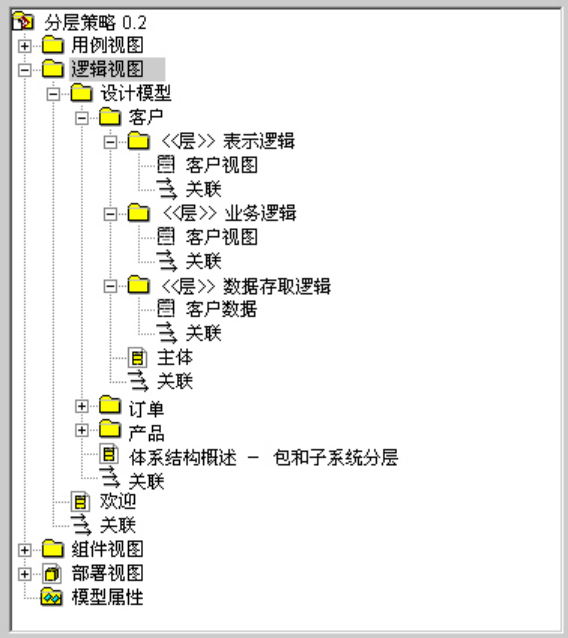

图 7：业务组件包中的*显式*分层

当有必要实际划分实施每个职责的元素时，基于职责的分层策略除了影响设计模型，通常还影响*实施模型*。例如，请考虑展示“瘦客户机”的物理分发的系统：确定支持客户机和服务器上的执行所需的实施单元是很有用的。在本示例中，*表示逻辑层*中的元素驻留于部署在客户机上的应用程序中，而*业务逻辑层*和*数据逻辑层*中的所有元素驻留于部署在服务器上的另一个应用程序中。

此场景表明*实施模型*（如图 8 所示），它表示 Rational Rose 浏览器图像和组件图，该图像和组件图显示了部署再客户机上的应用程序元素。在本示例中，碰巧存在设计模型中的类和实施模型中的 UML 组件之间的一一映射。不过请注意，此映射通常依赖于所用的实施技术。

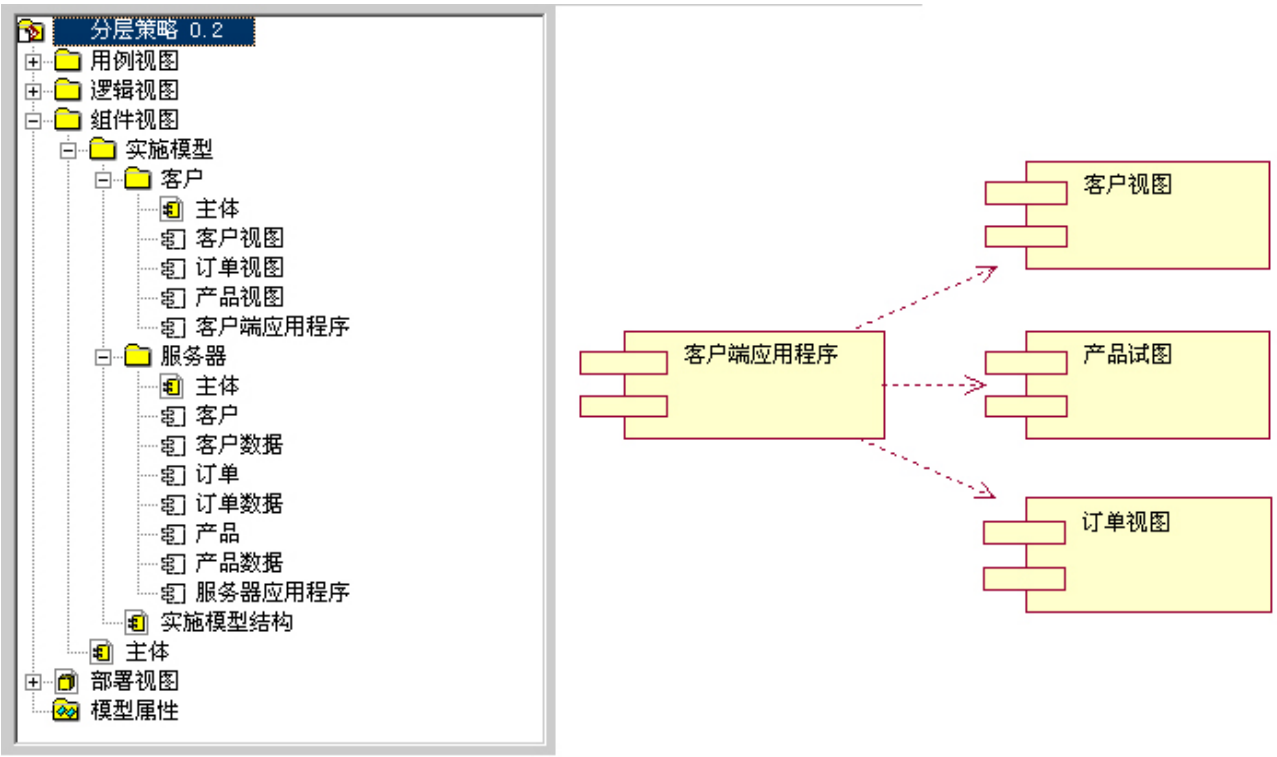

图 8：实施模型中的*瘦式*分层

类似地，当需要描述职责的物理分发时，基于职责的分层策略也会影响*部署模型*。在图 9 中，使用上面的示例，我们可以看到定义了六个节点。三个*客户机节点*中的每一个包括一个 ClientApplication 流程。FrontEndServer 节点包括 LoadBalancer 流程，该流程负责将客户机请求分配到两个服务器节点中的一个。每个*服务器节点*包括一个 ServerApplication 流程。

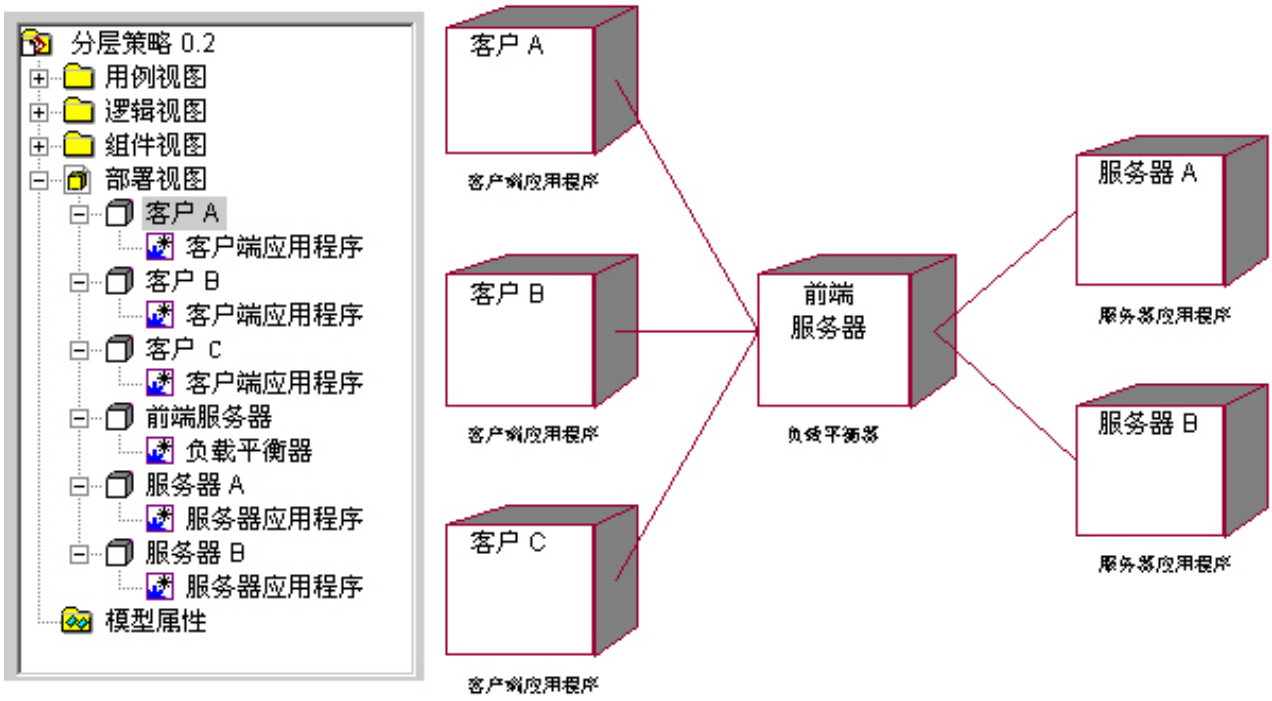

图 9：部署模型描述职责的物理分发

基于重用的建模

另一个常用的分层是基于重用的。此策略尤其适合于具有可识别的目标以在整个组织内重用组件的组织。使用此分层策略的效果在于组件的可重用性是高度可见的，因为组件是按照重用级别明确分组的。图 10 中显示了从 [Jacobson] 中描述的策略中派生出的分层示例。在此我们看到三个层：基本、特定于业务的和特定于应用程序的。

- *基本层*包含可能跨组织应用的元素（如数学）。此类元素将被广泛重用。
- *特定于业务的层*包含适用于特定组织但独立于应用程序的元素（如地址簿）。此类元素将在同一个组织的应用程序内重用。
- *特定于应用程序的层*包含适用于特定应用程序或项目的元素（如个人组织者）。这些元素最少重用。

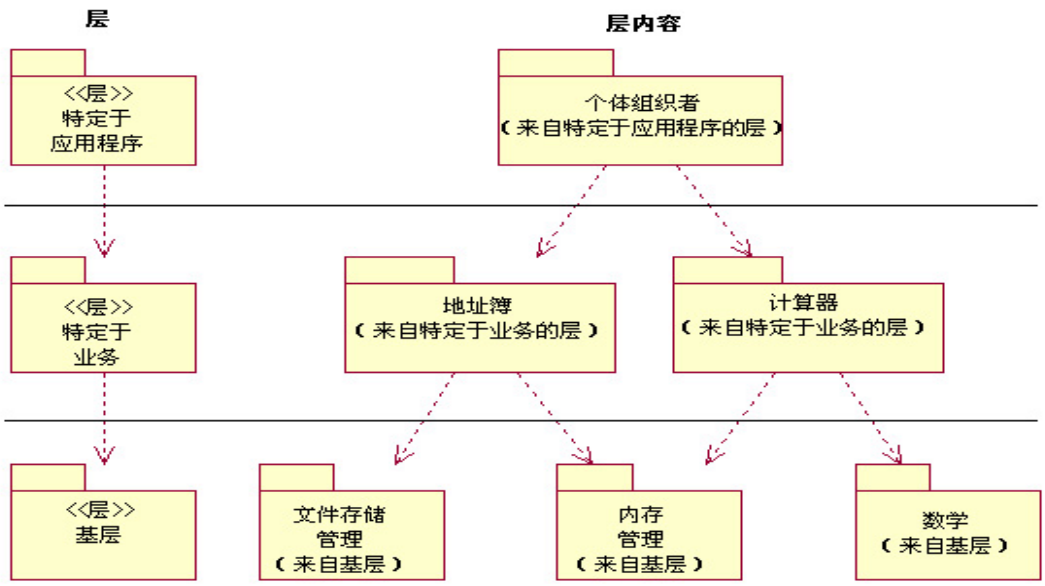

图 10：基于重用的分层示例

我们可以得出：基本层中的元素是最常重用的，而特定于应用程序的层中的元素是更特定于项目的，因此较少重用。

对基于重用的层建模

重用策略的应用主要影响*设计模型*。并入基于重用的分层的设计模型结构可直接构想并显示在图 11 中，它反映图 10 中的示例。

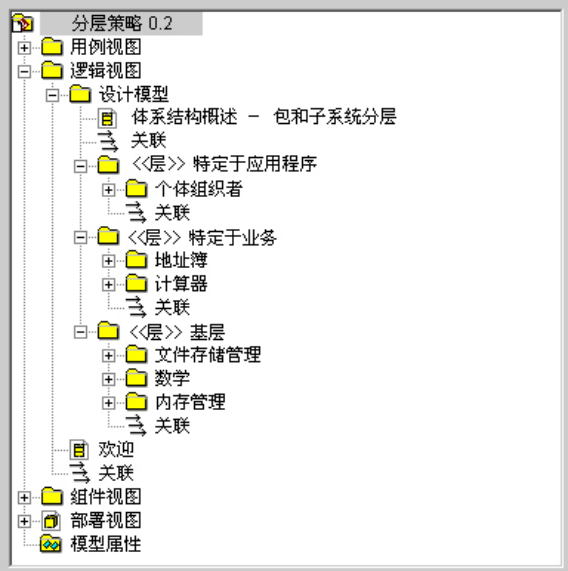

图 11：并入基于重用的分层的设计模型

其它分层策略

本白皮书旨在使用两个最广泛使用的策略作为示例，简单指出存在的不同分层策略的“特色”。不过，类似的方法可作为确认特征（例如安全性、所有权和技能集）的策略。

多维分层

也可结合之前描述的策略来创建新的分层策略。图 12 中的示例显示：

- 来自上文中示例的基于重用的两个层
  - 特定于应用程序
  - 特定于业务
- 基于职责的三个层（级）
  - 表示逻辑
  - 业务逻辑
  - 数据访问逻辑

基于重用的分层策略中呈现的依赖关系通常是由业务逻辑层中的元素之间的依赖关系产生的，如图 12 所示，在此可看到个人组织者和地址簿之间的依赖关系。

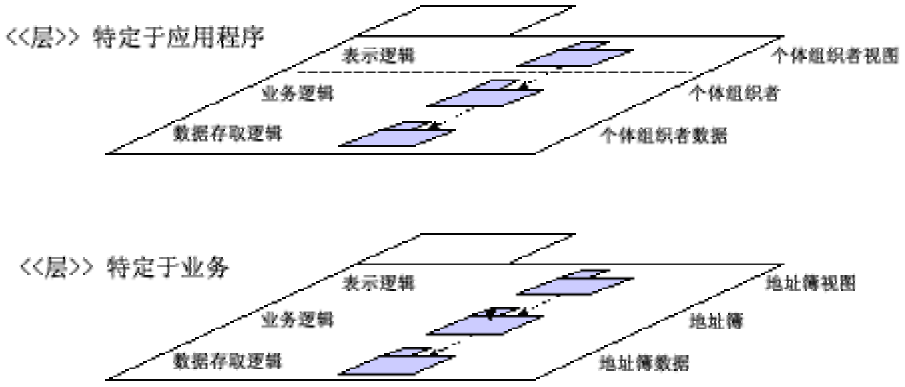

图 12：多维分层

对多维层建模

在此，我们考虑两维*设计模型*中的多维分层方面的说明。也考虑由此并入*业务组件*概念的结构。

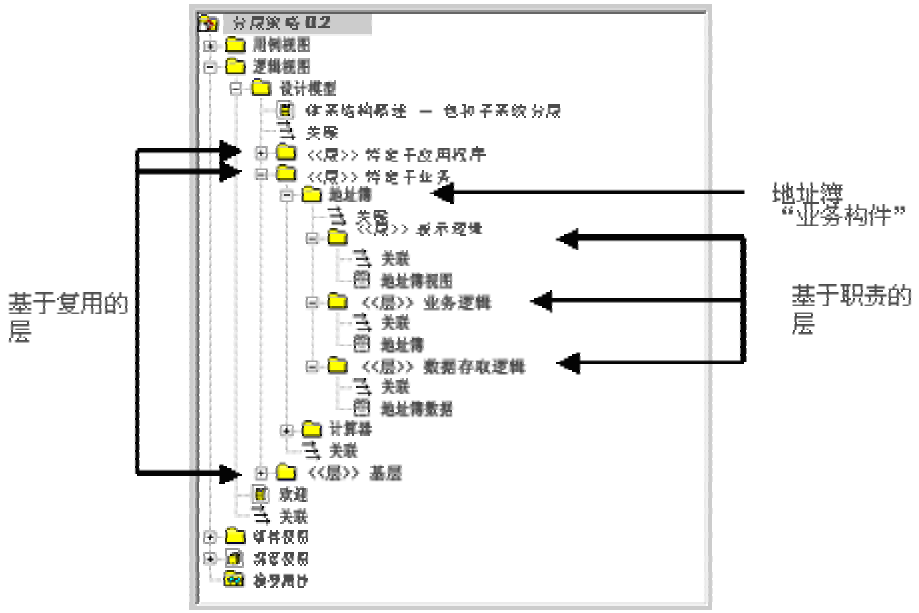

图 13：并入多维分层的设计模型

采纳多维分层策略需要确定主要策略。在示例中，主要分层策略是基于复用的。首先基于给出*特定于应用程序*、*特定于业务*和*基本三层*的此策略来组织设计模型。然后用驻留在每一层中的元素进一步组织每一层；例如，图 13 显示了包含*地址簿*和*计算器*的特定于业务的层。接着，基于次级策略（基于职责的分层）进一步组织每个元素。例如，*地址簿*包包含三个层：*表示逻辑*、*业务逻辑*和*数据访问逻辑*。

每一层包含驻留在该层的所有元素：

- *表示逻辑层*包含 AddressBookView 类
- *业务逻辑层*包含 AddressBook 类
- *数据访问逻辑层*包含 AddressBookData 类

结论

设计人员必须作出的最重要的决策之一是选择一个合适的分层策略，因为它将对生成的模型结构产生重大影响。不过，更重要的是所选的分层策略可直接带来商业利益（如可维护性和重用）。例如，如果通过采纳基于职责的分层策略，将系统的不同职责彼此隔离开，将可能开发出更具维护性的系统。同样地，使用基于重用的分层策略，可清晰地确定出可重用的系统元素。

致谢

作者对于 Kelli Houston、Wojtek Kozaczynski、Philippe Kruchten、Bran Selic 和 Catherine Southwood（所有 Rational 软件）对本白皮书的早期草稿所提的有深刻见解的意见表示感谢。

参考书目

[Buschmann] Buschmann, Frank, et al. *A System of Patterns*. 1996. New York: John Wiley & Sons. ISBN 0-471-95869-7.

[Edwards] Edwards, Jeri. *3-Tier Client/Server at Work*. 1999. New York: John Wiley & Sons. ISBN 0-471-31502-8.

[Eeles] Eeles, Peter, and Oliver Sims. *Building Business Objects*. 1998. New York: John Wiley & Sons. ISBN 0-471-19176-0.

[Herzum] Herzum, Peter, and Oliver Sims. *The Business Component Factory*. 2000. New York: John Wiley & Sons.

[Jacobson] Jacobson, Ivar, et al. *Software Reuse*. 1997. Reading, Massachusetts: Addison-Wesley. ISBN 0-201-92476-5.

[PLoP2] Vlissides, John, James Coplien, and Norman Kerth. *Pattern Languages of Program Design 2*. 1996. Reading, Massachusetts: Addison-Wesley. ISBN 0-201-89527-7.

Rational®
the software development company

Dual Headquarters:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
International Locations: www.rational.com/worldwide