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。分层便是其中之一并将在本白皮书中举例描述。此项comprehend技术针对两个主要问题：大多数系统过于复杂而难以全面掌握，而不同用户需要掌握系统的不同方面。

在许多系统上都采用了分层，并且许多著作中都对其持认可态度，Rational Unified Process（RUP）中也是如此。但是，这项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所获得的效果。

什么是“分层”?

让我们从“分层”的定义开始。术语*层*指的是体系结构模式的应用，通常称为*“层”模式*，这在包括 *RUP* 的许多著作（*[Buschmann]*、*[Herzum]*、*[PloP2]*）中均有介绍。*模式*表示存在于特定环境中的常见问题的解决方案。表 1 给出了层模式的概述。

表 1：“层”模式的概述

	层模式
环境	需要分解的系统
问题	过于复杂而难以全面掌握的系统 难以维护的系统 未隔离不稳定元素的系统大多数可重用元素难以确定的系统将由不同的团队（并可能用不同的技术）构建的系统
解决方案	将系统结构分为层

分层的最常见示例之一是由“ 国际标准化组织 ”（ISO）定义的 OSI 7 层模型：此模型（图 1 所示）定义了一组网络协议 － 每一层专注于通信的一个特定方面并构建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。本白皮书后面部分将对此策略作详细描述。

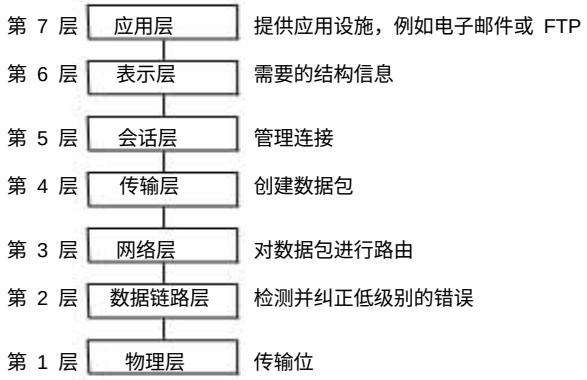

图 1：OSI 7 层模型（基于职责的分层）

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，例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的访问权的元素，例如关系数据库。

请注意，可用多种方法对层建模，本白皮书后面部分将对其作详细描述。目前，我们将使用具有构造型 «layer» 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）通常称为“级”（tier），它是分布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方向，因为它表示作为分层系统特征的一条特定规则 — 特定层中的元素只能访问同一层或以下各层中的元素 [1]。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。同时，*数据访问逻辑层*中的元素也不能访问*业务逻辑层*中的元素。此结构通常称为有向无环图（DAG）。由于依赖关系是单向，因此它是有方向的；并且由于依赖关系的路径不会形成环路，因此它是无环的。

请格外注意，当定义分层策略时，明确每一层的含义将十分重要，这样才能将元素正确分配到适当的层中。若未能将元素正确分配到适当的层中，则一开始就会降低应用该策略的价值。当更详细地讨论每个分层策略时，将给出关于每一层含义的通用指导信息。

对层建模

当研究不同的分层策略时，将逐渐明确使用特定的*模型*（因此使用特定的 UML 元素）传达每个策略是合适的。*模型*代表从特定角度对系统的完整描述。
图 3 显示了一个示例，其中包含四个模型，它们分别代表所考虑系统的不同角度：

[1] 虽然事件通知会导致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显的依赖关系。

- 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
- 实现模型：捕获系统实现

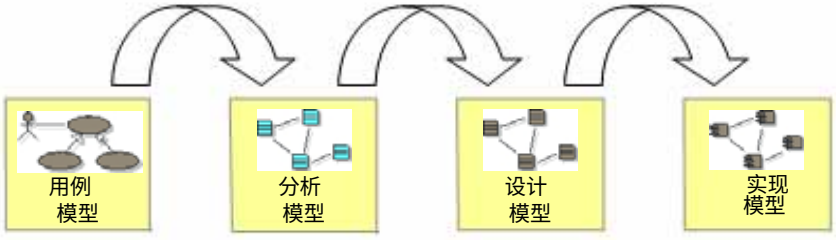

图 3：表示逐渐改进的四个模型

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。本部分讨论基于以下特征的分层：

- 职责
- 复用

每种策略的说明将作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。由于各种系统职责相互独立，因此该特定策略可以改进系统的开发和维护。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

如图 4 所示，这些职责中的每一个职责均可表示为一层，该图显示了每一层的一些样本内容。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*概念由以下内容组成：

- *CustomerView* 类：负责与客户关联的*表示逻辑*，例如在用户界面中的显示客户
- *Customer* 类：负责与客户关联的*业务逻辑*，例如验证客户详细资料

= *CustomerData* 类：负责与客户关联的*数据访问逻辑*，例如使客户关系持久化

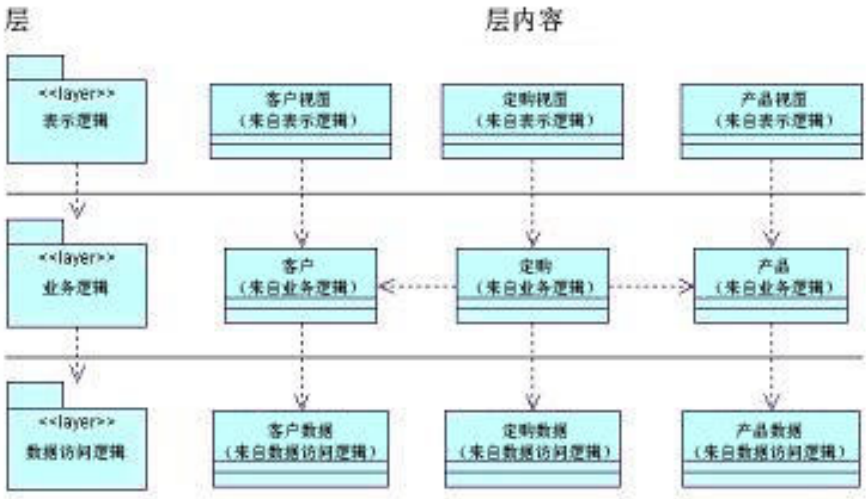

图 4：基于职责分层的层和内容

现在，我们来讨论一下关于此特定分层策略的一些“谬论”。

谬论 1：层和级是不同的

此典型谬论是造成混淆的常见源头。虽然层基于某个特定策略（即，一个职责），但实际上级就是层。造成此混淆的原因是由于可以用很多方式应用级的概念，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（*分布式*） 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。请思考一下 3 级分层。即使各种元素位于其中一层，但是每一层本身仍可以用多种方式应用（如表 3 所示），表中使用通常用于描述特定物理分布特征的名称（例如“ 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布策略，其中某些元素将归类概括为“ 瘦客户机 ”分布，而其他则是“ 胖客户机 ”分布。通常情况下，选择基于非功能性需求（例如性能）。

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*、*实现模型*和*部署模型*。*设计模型*通常使用以下两种方法之一进行构造。

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，它显示了：

- 位于表示逻辑包中的表示类（CustomerView、OrderView 和 ProductView）
- 驻留在业务逻辑包中的业务逻辑类（Customer、Order 和 Product）
- 位于数据访问逻辑包中的数据访问逻辑类（CustomerData、OrderData 和 ProductData）

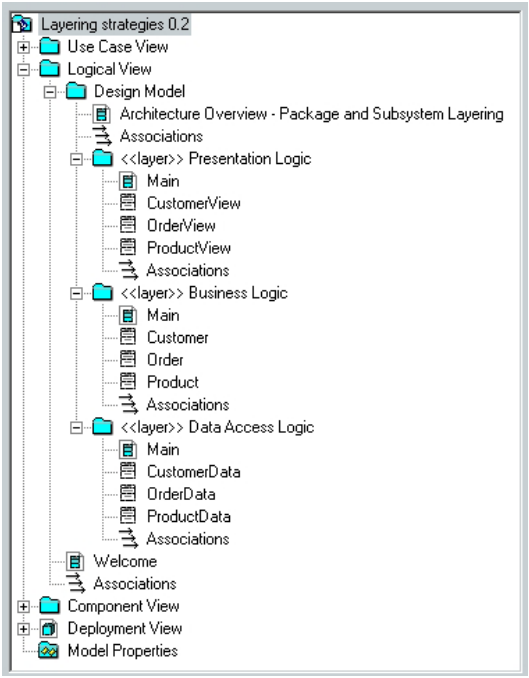

图 5：包含在层中的元素

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由系统支持并与领域相关的概念。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] 中作进一步讨论。这种思考方式将形成图 6 中所示的模型结构。在此示例中，分层将由元素名称来*表示*。例如，所有的*视图*类（例如 *CustomerView*）暗指表示逻辑层，而所有的*数据*类（例如 *CustomerData*）暗指数据访问逻辑层。未限定的类名（例如 *Customer*）暗指业务逻辑层。

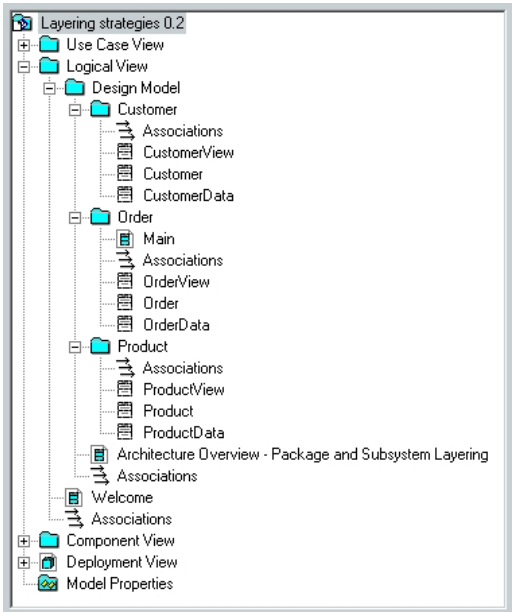

图 6：每个业务组件包中的隐式分层

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。
*当给定的业务组件的每一层中包含许多元素时，更能显示出这种结构的优越性。*虽然在本示例中只扩展了客户（Customer）业务组件包，但订单（Order）和产品（Product）包应具有类似的结构。

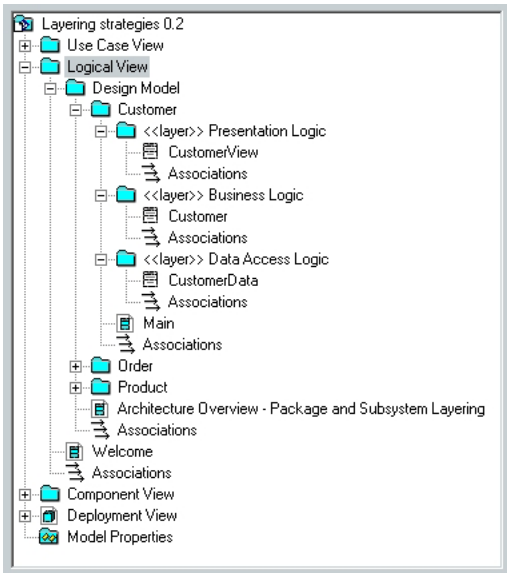

图 7：业务组件包中的显式分层

当有必要在物理上划分实现每个职责的元素时，基于职责的分层策略除了影响设计模型，通常还影响*实现模型*。例如，请思考一个展示“瘦客户机”物理分布的系统：确定支持客户机和服务器上的执行所需的实现单元将十分有用。在本示例中，*表示逻辑层*中的元素位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*中的所有元素则位于部署在服务器上的另一个应用程序中。

此方案表示*实现模型*（如图 8 所示），它代表 Rational Rose 浏览器图像和组件图，该图像和组件图显示了部署在客户机上的应用程序的元素。在本示例中，设计模型中的类和实现模型中的 UML 组件之间恰好存在一对一映射关系。但请注意，此映射通常取决于所用的实现技术。

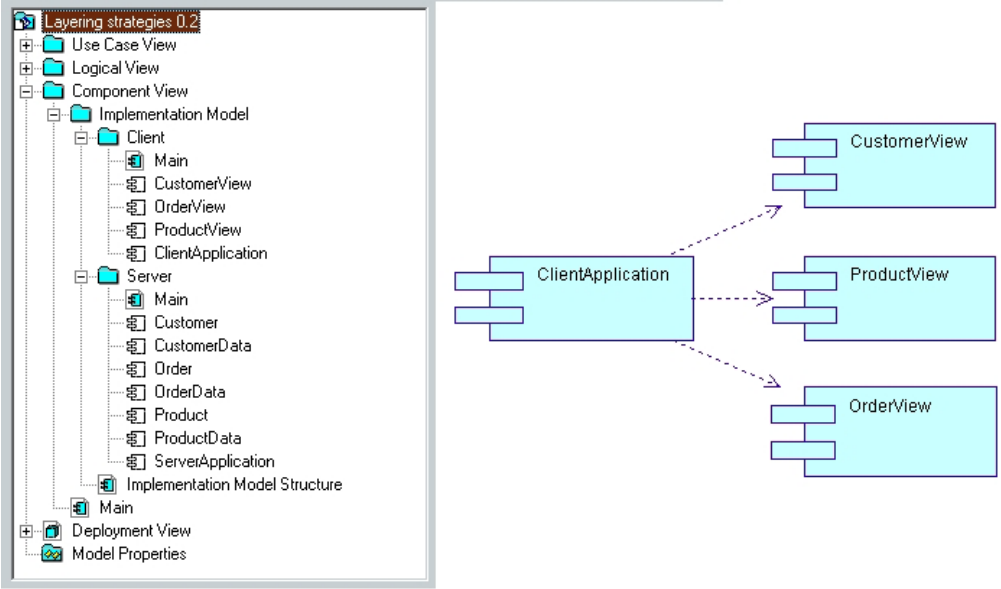

图 8：实现模型中的隐式分层

同样，当需要描述职责的物理分布时，基于职责的分层策略也可能影响*部署模型*。在图 9 中，并使用上面的示例，我们可以看到定义了六个节点。三个*客户机节点*中的每一个节点都包括一个 ClientApplication 流程。FrontEndServer 节点包括一个 LoadBalancer 流程，该流程负责将客户机请求分发给两个服务器节点中的一个节点。每个服务器节点包括一个 ServerApplication 流程。

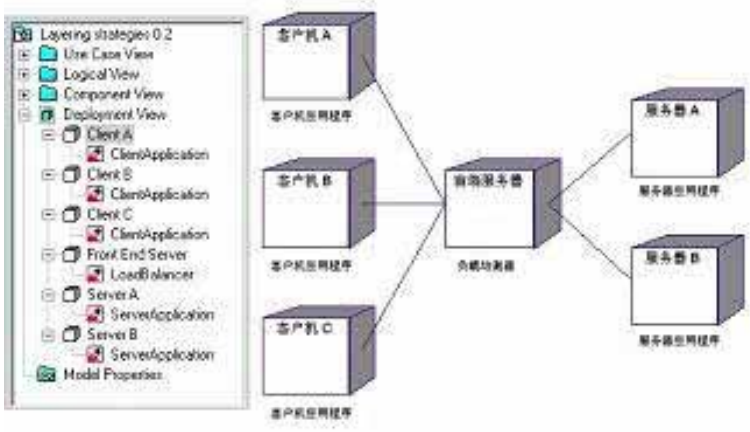

图 9：部署模型描述职责的物理分发

基于重用的建模

另一个常用的分层是基于重用的分层。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。由于组件是按照重用级别进行明确分组，因此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。图 10 中显示了从 [Jacobson] 中描述的策略派生出的示例分层。在此我们看到三个层：基础层、特定于业务的层和特定于应用程序的层。

- *基本层*包含可能跨组织应用的元素（例如数学）。此类元素将得到广泛重用。
- *特定于业务的层*包含适用于特定组织但与应用程序无关的元素（例如地址簿）。此类元素将在同一个组织中的应用程序内重用。
- *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（例如个人备忘录）。此类元素最少重用。

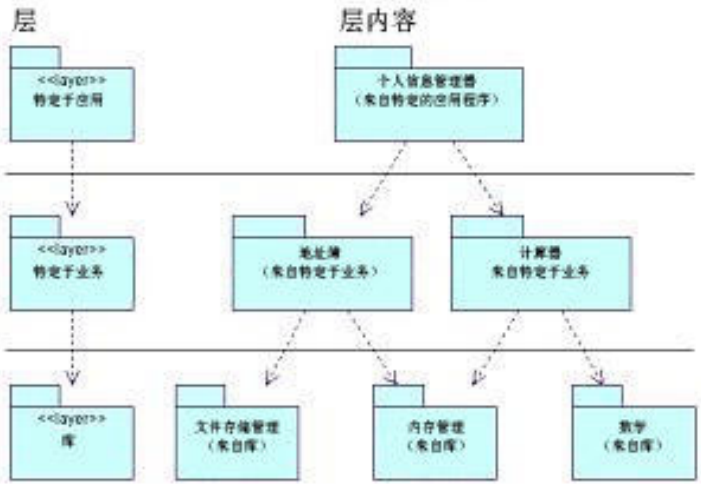

图 10：基于重用的分层示例

我们可以得出：基本层中的元素最为常重用，而特定于应用程序的层中的元素更特定于项目，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。

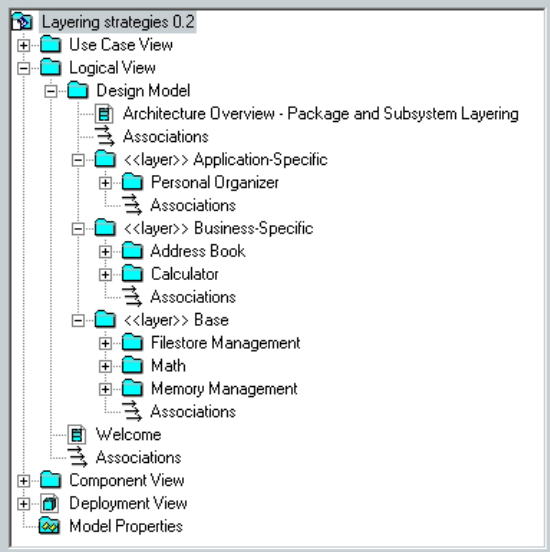

图 11：并入基于重用的分层的设计模型

其他分层策略

本白皮书旨在使用两个最广泛使用的策略作为示例，简单描绘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所产生，如图 12 所示，可看到 PersonalOrganizer 和 AddressBook 之间的依赖关系。

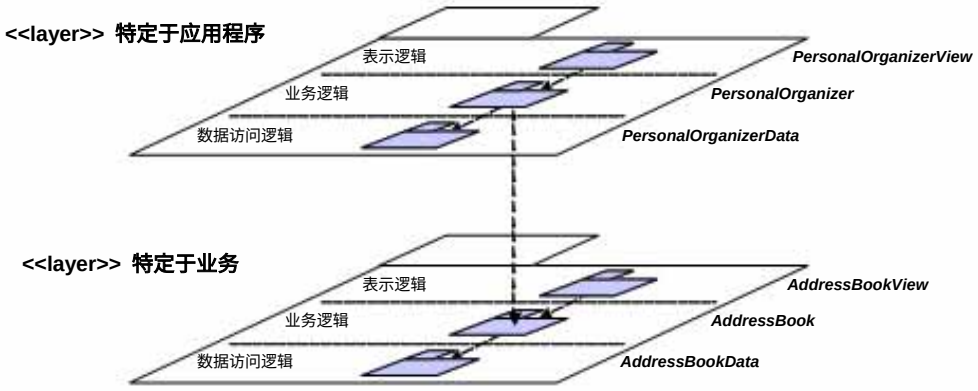

图 12：多维分层

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。我们还考虑一种结构，通过该结构并入业务组件概念。

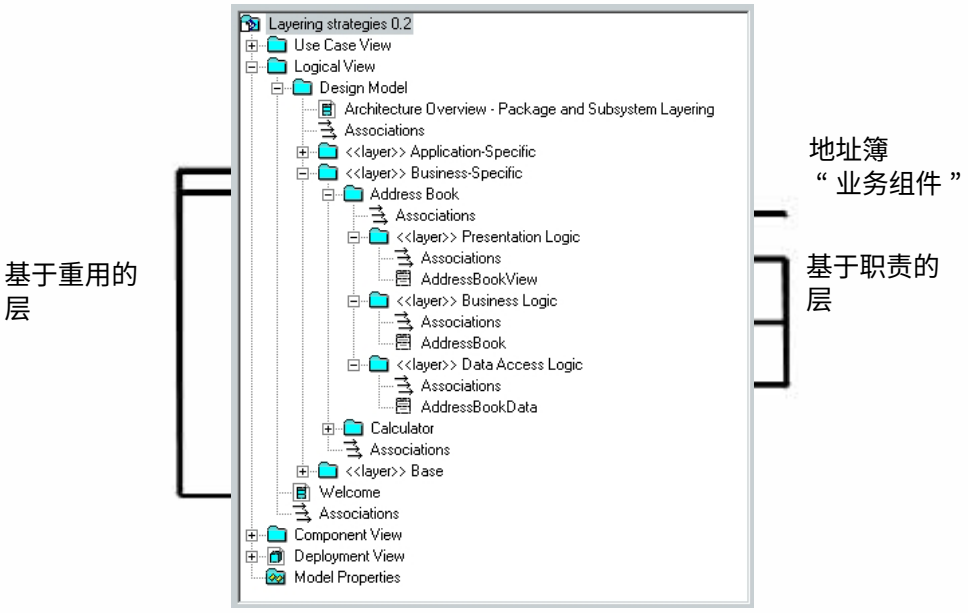

图 13：并入多维分层的设计模型

采纳多维分层策略需要确定主要策略。在我们的示例中，主要分层策略基于重用。首先根据给出*特定于应用程序*的层、*特定于业务*的层和*基础*层的此策略来组织设计模型。然后用位于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，根据次级策略（基于职责的分层）进一步组织这些元素。例如，地址簿包中含有三个层：*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

总结

架构设计师必须作出的最重要的决策之一就是选择一个合适的分层策略，因为它将对所生成模型的结构产生重大影响。不过，更重要的是所选的分层策略可直接带来商业利益（例如可维护性和重用）。
例如，如果通过采用基于职责的分层策略区分系统中的不同职责，则可能开发出的系统的可维护性更好。同样，使用基于重用的分层策略，可清楚地确定可重用的系统元素。

致谢

作者对于 Kelli Houston、Wojtek Kozaczynski、Philippe Kruchten、Bran Selic 和 Catherine Southwood（均来自 Rational Software）对本白皮书的初稿所提出的中肯意见表示感谢。

参考资料

[Buschmann] *A System of Patterns*，Buschmann，Frank 等，1996 年，John Wiley & Sons，纽约，ISBN 0-471-95869-7。

[Edwards] *3-Tier Client/Server at Work*，Edwards，Jeri，1999 年，John Wiley & Sons，纽约，ISBN 0-471-31502-8。

[Eeles] *Building Business Objects*，Eeles，Peter 和 Oliver Sims，1998 年，John Wiley & Sons，纽约，ISBN 0-471-19176-0。

[Herzum] *The Business Component Factory*，Herzum，Peter 和 Oliver Sims，John Wiley & Sons，纽约，2000 年。

[Jacobson] *Software Reuse*，Jacobson，Ivar 等，1997 年，选自 Reading，Addison-Wesley，马萨诸塞州，ISBN 0-201-92476-5。

[PLoP2] *Pattern Languages of Program Design 2*，Vlissides，John，James Coplien 和 Norman Kerth，1996 年，选自 Reading，马萨诸塞州 Addison-Wesley，ISBN 0-201-89527-7。

Rational
the software development company

两家总部：

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

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

免费电话：(800) 728-1212
电子邮件：info@rational.com
Web：www.rational.com
全球网址：www.rational.com/worldwide