

第4章

针对公司财务的观察模式

要想完全读懂本章的内容，你需要首先阅读第3章。

对大公司来讲，要识别其中的高层次问题是很容易的事，但要找出这些问题的根本原因却很费事。这些公司产生的海量信息，能将任何一个试图分析它们的人员迅速淹没。

例如，对一个公司业绩进行评估的主要标准是它的最终收益。如果收益与以往相比有明显减少，就应做进一步的分析，以确定减少的原因。对ACM公司所做的分析显示：尽管他们所销售的设备的价格还是比较合理的，但销售收入仍然减少了。这一点在他们的东北部子公司最为明显。进一步分析发现：他们生产的1100系列的大容量咖啡机的销售量明显低于预计的销售量，特别是在政府部门中，这种情况尤为突出。许多这样的分析都是基于数字的，但更进一步的分析应着重于定性方面，而不仅仅是定量方面。也许这次不是很好的销售业绩是由于销售赔偿计划做得不好、政府预算的削减，或者是因为夏季太炎热，又或者是在该地区又出现了一个强有力的竞争对手等。

所有这些分析都和一个医生根据他的患者的症状进行诊断这一过程很相似。我们利用所掌握的领域知识并通过一些可能的途径，来对一些明显的症状进行回溯分析。我们希望能够找到根本原因，进而对其加以处理。通过将类似过程举一反三，我们可以设想将这种诊断模式应用到公司财务分析中。

在第3章里，阐述了在一个医疗保健项目的上下文环境中如何得出关于患者的定性和定量的描述。在第3章结尾处，还简要地提到了这种模式可以应用于像公司财务分析这样类型的其它分析中。本章将着重讨论如何做到这一点。该模式的效果不错，但是仍需要做一定的改进。幸运的是，改进后的模式都是对现有模型的扩充而不是改变。

第一种模式是用待分析的企业片断来代替观察和测量模式中的人员。企业片断（参见4.1节）描述了由一系列维度所组成企业的某一部分。其中每一维度指的是对企业进行分级式分解后的产物，如地点、产品范围

或者市场等。企业片断是这些维度的组合，类似的技术在多维数据库中已经广泛使用。

测量方案模式（参见4.2节）描述的是如何使用公式（模型类型的实例）从一些测量计算出另外一些测量。在第3章中曾经讨论过每种测量是如何测量现象类型的；在这里将讨论针对特定的现象类型，测量方案是如何定义相应的测量创建方法的。我们总结了三种测量方案：因果测量方案（参见4.2.2节），该模式描述如何将不同的现象类型组合起来计算其它现象类型（比如，销售利税就是由单位销售量和平均价格计算出来的）。比较测量方案（参见4.2.2节），该模式描述单个现象类型在不同状态类型（参见4.2.3节）情况下的区别（比如，实际的销售利税与计划的销售利税间的偏差）。维度合并（参见4.2.5节），该模式利用在企业片断模式中所定义的维度来计算总值（比如，通过累计东北部各州的销售利润值来计算东北部的销售利税总额）。这些测量方案子类型都用到了多态性原理来计算其值。

我们通常利用定性现象来描述定量现象类型。此时，可以采取将现象及其类型的取值范围关联起来的方式来定义该现象。首先，我们需要一个范围（参见4.3节），该模式可以帮助我们描述两个不同的量之间的范围以及根据该范围所要做的不同操作。接下来就可以通过两种方式来定义一个带范围的现象（参见4.4节），一种方式是利用一个带范围属性的现象（参见4.4.1节），另一种方式是利用范围函数（参见4.4.2节）。

可以把本章中的模式和第3章中的模式结合起来分析公司的财务数据。4.5节讲的是如何利用这些模式去识别大型企业中的问题根源。

本章中的模型基于某个大型制造企业的一个小组的工作成果。该小组针对公司财务创新性地使用了医疗保健模型，并且发现它是一个非常有用的基础模型。本章的模型都是用C++来实现的。

关键概念：企业片断、维度、测量方案、状态类型

58

4.1 企业片断

本章所探讨的问题与第3章中所讨论的问题的最为明显的一个差别就是：在这里所观察的对象不再是一个单独的患者。有时我们着眼于整个企业，但有时我们只是关注该企业的某一部分，比如10-11浓咖啡在东北部的政府部门的销售情况。要做到这一点，可以通过把企业的每一部分以及整个企业作为独立团体来加以处理。然而，特别重要的一点就是要确保这些企业的各部分之间的关系是明确的。

这样一来，我们就必须将从护理程序到患者的映射更改为到其它一些类型的映射。我在第3章有关映射的论述把这个问题略过去了，但其实这

个问题并不是一个很简单的问题。最初的Cosmos模型[1]（它是第3章的基础）并没有考虑要将观察和人关联起来。在实际情况中，这种关联是连接到一种叫“关照对象”的类型上。关照对象本来是对患者和人群的一种泛指。人群由很多人组成，这样就可以允许观察的对象是一群人，这对公共保健非常有用。

对于公司财务，我们需要一种称为“企业片断”的新类型，它是“关照对象”的一种子类型（如图4-1所示）。一个企业片断是企业的某一部分，并且是通过一种非常特别的方式加以定义的。

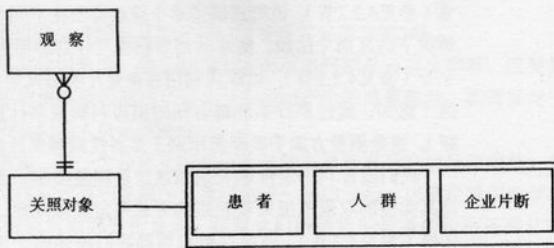


图4-1 关照对象及其子类型

第3章中的患者是可观察的关照对象中的一种。

当我们在观察一家企业时，会发现可以根据一些标准来把它划分成不同的组成部分。比如，可以按照组织单位或地理位置来划分，也可以按照产品或者产品销售对象等来划分。每一种划分方法都可以说是独立的，而且每一种划分方法都会产生一个层次结构。例如，一家跨国企业可以首先按照市场（美国），然后按照区域（东北部），最后按照地区（新罕布什尔州）来划分。其中每一个独立的层次结构都是该企业的一个维度。新罕布什尔州和东北部在地理维度中是处于不同层次上的元素。一个企业片断由多个维度元素（每个维度元素对应一个企业维度）组成。这样，ACM中的“东北部地区，11-10系列，政府部门”部分就可以定义为“地理维度上的东北部”、“产品维度中的11-10系列”，“行业维度中的政府部门”这三种维度元素所组成的企业片断（如图4-2所示）。该分析方法通常被称为“星型模式”[4]，普遍用在多维数据库[2]中。

利用所定义的企业片断，就可以构造出不同类型之间的关系模型（如图4-3所示）。可以将维度元素连接成为维度元素层次结构。可以定义多个维度元素层次结构。需要注意维度元素层次结构是如何约束父关联的，因为单靠基数还无法维持一个维度元素层次结构（尽管允许循环）。企业片

断必须从每一个维度元素层中获得一个维度元素，就像前面所提到的拥有三个维度元素的企业片断那样。针对维度元素的约束确保了维度元素层次结构的所有元素都是在同一维度中。该模型可以很好地处理这种情况，但是它也存在两个缺点。首先，还没有确切地定义维度和维度级别的概念。其次，添加一个新的维度就会导致模型的改变。

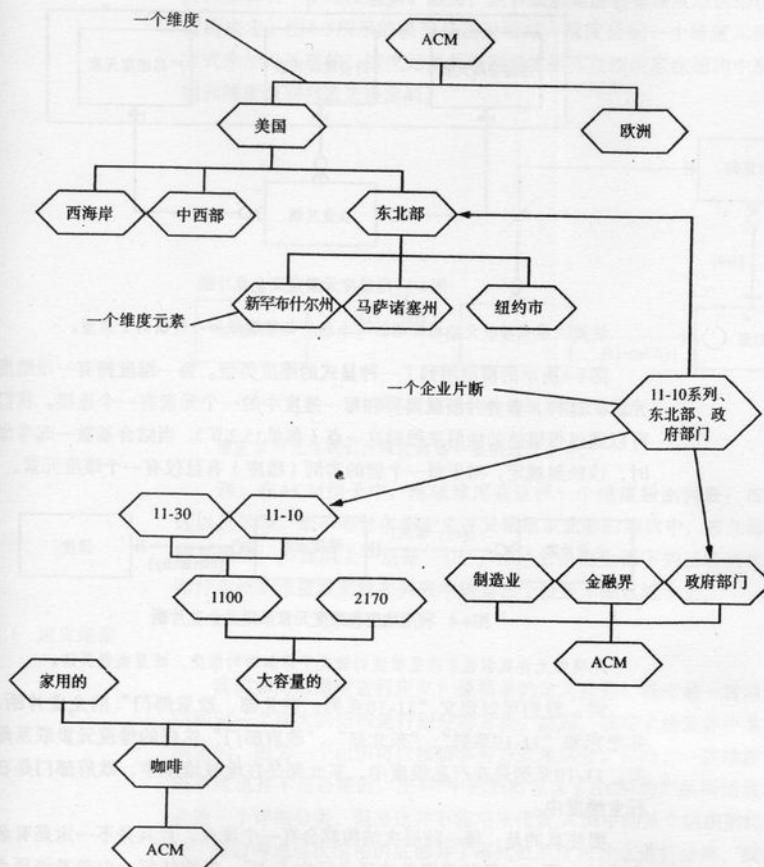


图4-2 如何将企业片断与维度元素关联起来

一个企业片断就是各维度元素的组合体。

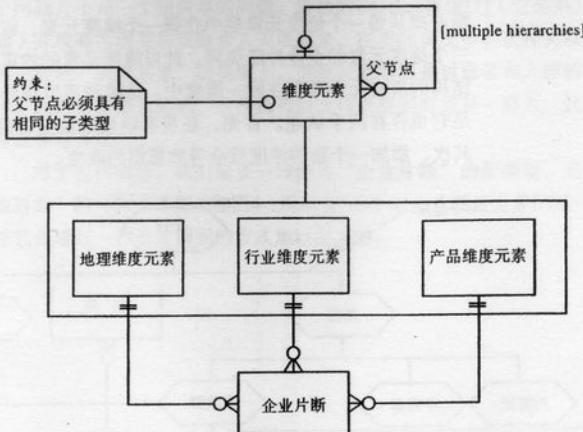


图4-3 用维度元素定义企业片断

使用该模式时，无论何时增加一个维度都需要增加一种新的子类型。

图4-4所示的模型用到了一种显式的维度类型。每一维度拥有一层维度元素。这样，企业片断就需要和每一维度中的一个元素有一个连接。我们可以通过带键值的映射来做到这一点（参见15.2节）。当结合基数一起考虑时，该映射规定，对于每一个键的实例（维度）有且仅有一个维度元素。

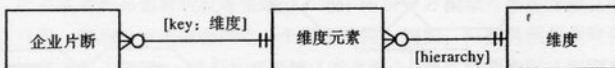


图4-4 利用维度和维度元素来定义企业片断

此模型允许我们在不改变模型的情况下添加新的维度，而且也很简洁。

例：我们可以定义“11-10系列、东北部、政府部门”的企业片断，并把它和“11-10系列”、“东北部”、“政府部门”这样的维度元素联系起来。11-10系列是在产品维度中，东北部是在地域维度中，政府部门是在行业维度中。

要注意的是：每一种层次结构都会有一个顶点，且其并不一定是有名有姓的东西。一般的习惯是将顶点标为“all”，表明任何一个参考该顶点的片段都没有进行任何的细分。另一个习惯是对维度元素的映射是可选的；这时常用“nil”来命名树的顶点。前一种方法更为可取，尽管该顶点

60

61
62

元素在一定程度上是凭空造出来的。

例：如果我们增加了渠道维度，那么企业片断“11-10系列、东北部、政府部门”就和渠道层的顶点维度元素建立起一种连接关系。我们称该维度元素为“all”。

给维度级别添加一种类型的效果并不十分明显。一般来说，每一个维度元素都有一个维度级别。然而，这种级别是由它在维度层次结构中的位置所决定。图4-5所示的模型是通过给每一维度分配一个维度元素列表的方式来加以处理的。维度元素的级别是根据其在维度层次结构中所处的级别和维度级别列表来决定的。

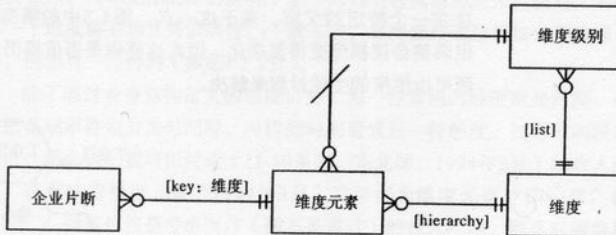


图4-5 在图4-4中添加了维度级别

维度级别允许我们对维度的每一级别进行命名。

例：在ACM例子中，地域维度有这样一个维度级别列表：市场、地区以及区域。新罕布什尔州定义在父辈是东北部的层次中，东北部的上一层是美国，美国的上一层是“all”。由于它是自上而下的三级结构，新罕布什尔州的维度级别就在列表中的第三个位置（即区域）上。

4.1.1 定义维度

我们如何对维度进行定义？最简单的定义就是：维度是一种通过组织结构对一个庞大的组织进行划分的方法。然而，该定义通常并不是最令人满意的，因为一个组织可以按很多种方式来划分。另外，一些维度对于组织系统也并不是必需的。图4-2中的模型包含了ACM的产品所销售到的行业的一个详细分类，但是这并不能用来代表ACM中的某个组织架构。

通过考虑层次结构的底部并了解此处维度对什么进行分类，我们可以找到一种更好的维度定义方法。在前面的例子中我们可以发现，ACM将重点放在了咖啡机的销售或租用上。我们可以根据销售的是哪一种机器、销售的区域以及销售到的行业对维度进行分类。这些维度来自于焦点事件

的分类，焦点事件是“星型模式”[4]中的事实表。

在决定在这类分析中使用维度时，我们首先需要理解什么是焦点事件。接下来我们可以着手研究对这些焦点事件的分类方法。从图4-2中我们可以发现，焦点事件涉及拥有产品系列的产品，该产品系列包含着一个产品组，而该产品组中又包含着一种饮料。在销售维度上，可以看到区域、地区和市场等元素。

这些维度和级别应该由业务分析人员来定义。图4-6所示的就是一种非常好的定义方法。正如图4-6所显示的那样，这种结构可以变得相当复杂，维度也不必完全相互独立。例如，你会注意到价格范围维度和产品维度存在着交叉关系。这表明，任何一种产品沿着产品维度和价格范围维度存在一个特定的父辈。基于这一点，图4-5中的模型需要做适当的调整。但调整会使模型变得复杂化，因此这样做是否值得仍然是一个问题。该问题可由维度的生成过程来解决。

63

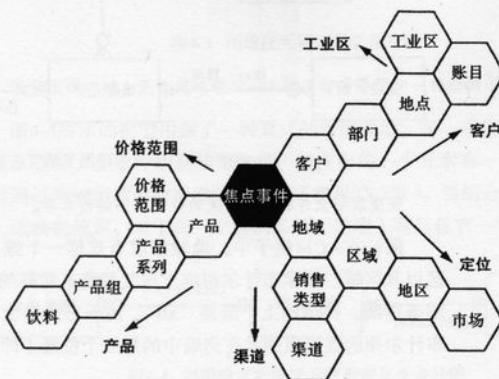


图4-6 典型的一套维度和级别

该图非常有用，它告诉我们在任何一个级别上最好不要产生6个以上的分支。实际上我们也很少这样去做。

维度不需要可测量到最低级别。因此，不值得或者甚至不可能分析到单个销售者的销售区域或单个消费者。这样，只能按维度向下分析到基础事件。为了将来的开发以及作为理解更高级别的基础，对较低级别的了解仍然是必要的。

对消费者域的全面分析将包括为消费者区域生成业务模型，其中将包括一个结构模型，而该结构模型会用于对维度的严格定义。每一维度都应

该代表一个沿着结构模型的层次路径。该过程的细节超出了本章范围，为了便于讨论，我们假设维度都已经被确定。

维度可由分析系统的用户明确地加以定义。此外，也可以根据公司的数据库来确定。对于后者，每一维度都需要一个生成操作来告诉它如何访问公司的数据库。这就允许系统随时给维度添加节点。

4.1.2 维度的属性以及企业片断

关于维度的一个重要规则是：对低级别维度的测量结果可以以适当的方式综合到高级别维度的测量之中。这样一来，如果我们想考察东北部的销售收入，可以把东北部的所有下级分区的销售值累加起来。所定义的任何一个维度都必须支持该属性。一般来讲，维度都是通过相加来实现综合的，但也有一些特例（参见4.2.5节）。

[64]

除了通过业务结构定义的维度以外，另一种常用的维度就是时间。通过把基础事件划分为时间段，可以把时间看成是一种维度。如果时间段是月份，那么我们就可以讨论（11-10系列，东北部，1994年3月）的收入问题。这意味着维度元素“1994年3月”可以作为维度元素“1994年”的“儿子”。倘若仅仅是考虑当月（而不是累计）的收入问题，那么时间维度就可以很好满足上面所讨论的组合属性。我们可以很容易地根据月份收入计算出年度收入。

企业片断和许多基础类型一样，具有一个共同的有趣的属性：所有的企业片断都只是概念上的存在。不存在数字5、数量5美元或者2314年1月1日这样的单独概念。所有这些东西都存在于我们的头脑之中，但在计算机中可能需要作为对象来生成。企业片断具有这样的共性，一旦所有的维度都确定了具体的维度元素，则所有企业片断在概念上就存在了，尽管可能并没有把它们创建成软件对象。

这种共性引出一个问题，即可否把一个企业片断当成是一种基础类型（参见A.1.5节）。如果是这样的话，就不应该有任何与非基础对象的映射。维度元素和（从关照对象中继承而来的）观察都是非基础的。虽然后者可以被排除，但前者由于是企业片断的一部分，因此不可以被排除。因为为指定的企业片断找出所有观察是一个非常普通的需求，所以需要意识到保持从企业片断到观察的映射的重要性。总而言之，虽然企业片断具有“普遍的概念存在性”这样的性质，但看上去它仍然是非基础的。

把企业片断看成是非基础的，对接口确实会存在一些影响。生成操作其实是查找或创建动作。首先要检查所需要企业片断的实例是否存在；如果存在就将其返回，如果不存在就创建它。（或者你可以认为它没有创建

操作，而仅仅是一个查找操作，在需要的时候会自动创建。)

4.2 测量方案

65

我们所讨论的企业分析用到了大量的测量。这些测量不是用手工记录的；通常或者是从一个或多个数据库中加载而来，或者是由其它的测量计算得来。我们要记住是如何获得这些测量的，即用于创建测量的具体方案是什么。图4-7展示测量和测量方案的一个基本轮廓，其中大部分内容和第3章类似。

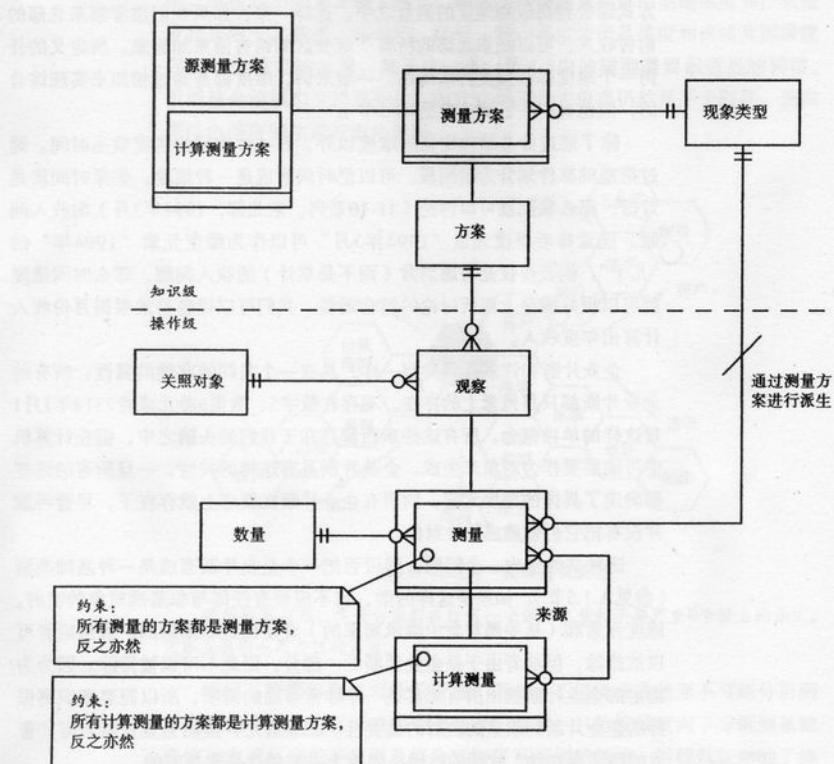


图4-7 测量和测量方案

源测量来自于数据库，而计算测量使用公式。

图4-7展示了两种测量方案。源测量方案涉及对一些企业数据库的访问。通常，一个对象在逻辑上是知道它正在访问哪一个数据库的，尽管实际的命令是在其它层。用户可以决定所要访问的数据库。计算测量方案表示要围绕该领域已经出现的测量做某种运算。

[66]

关于该模型的重要特点——其临床背景的反映——是任何一种现象类型都可以用多种测量方案来确定它的值。一些读者可能会觉得这个特点很奇怪。用多种方法计算的测量到底有什么意义？如果有多个公式，它就一定是另外一种现象类型。而这里最明显的一点就是一个现象类型可以有多种计算方案和多种源方案。我们可以在不同时期使用不同的方案。也可能会有多种源方案，选用哪一个取决于系统的可用性。有些数据库要比其它数据库更可靠，但没有任何数据库的可用性是完美的。

同样地，用户可以考虑利用不同的计算来生成相同的现象类型。选用哪一种计算取决于源的可用性，或者从用户角度考虑一些更细微的区别。关于这一点的一个很好的例子就是库存值。一般来说，库存值只有在年底才进行真正的计算，而在其它时间只是对该值进行估算。这两种情况算得的值都可以作为进一步财务分析所需的基本信息。

该模式的一些用户可能会选择指定用哪种测量方案来产生测量值。而另一些用户可能只是想得到一个现象类型，而让系统决定如何产生测量值。对于后一种情况，需要用某种方法来为现象类型进行测量方案的优先级排序。这可以通过建立一个从现象类型到测量方案的映射表来实现，映射表的前部定义首选的方案，以此类推。

要注意那些与其源测量有返回连接的计算测量。这需要遵循一个一般的规则，即当把计算结果作为一个对象时，应该知道该结果是由什么计算（方案）产生的以及该方案的输入是什么（源）。

4.2.1 保持计算的有效性

计算测量方案包括所使用的计算公式，如图4-8所示。这是一个独立的实例方法（参见6.6节）的例子。计算测量的公式通常都十分简单，所以我们可以利用一个简单的解释程序[3]并将公式刻画成电子表格形式。

该模式的一个重要特点是参数的提交方式。每一种计算测量方案都有一个参数列表。该列表代表了那些综合到公式中的现象类型。上面说的从现象类型到测量方案的映射表就是一个列表。为了使公式有意义，映射的元素必须是确定的。列表可以很好地做到这一点，当然，也可以使用字符串。

[67]

例：销售收入是一种把因果计算作为其测量方案的现象类型。因果计算的参数是一个列表，该列表有两种现象类型：销售量和平均价格。方法是公式：参数[1] × 参数[2]。

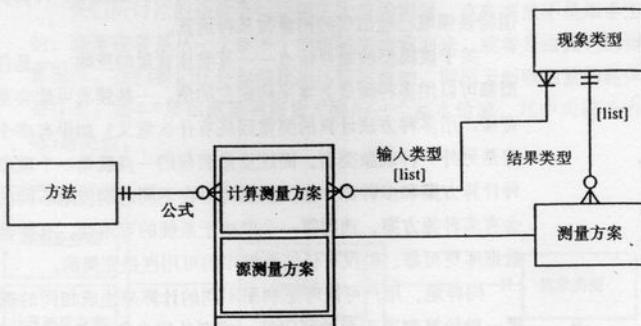


图4-8 计算测量方案中的方法

例：体质指数是医学中的一个重要参数。它的体重和身高两个参数是通过因果计算产生的。方法是公式：体重/身高²。

4.2.2 比较和因果测量方案

在公司财务应用中，测量并不都是绝对的值。用户通常并不在乎收入的具体数字，他们更想了解的是实际的和计划的数字之间的差异或者是对今年收入与往年收入的比较。

为了考虑这些比较型的测量，我们需要描述一下可能出现的不同类型的测量。典型的比较是对实际值和前段值或计划值之间的比较。前段值可以通过两种途径获得：适用时段起始点的参考值（参见3.8节），或者是寻找一个具有前段时间维度的企业片断的测量。计划测量要求我们区别看待实际值与计划值，这与我们在3.10节所讨论的临床观察和推理观察是相对应的。此外，推理观察必须记录下什么样的计划是推理的源，以便我们能够区分年度计划、季度预测等，如图4-9所示。

在这一点上，两类计算之间的基本差别应该是很明显的。一种类型是以其它现象类型值为基础确定某种现象类型值。例如，我们可以通过将销售量乘以平均价格来计算销售收入。这种类型的计算被称为因果计算，因为它遵循因果分析。在因果计算中，输入现象类型的数量和关系可以是任意的，计算所用的公式也可以是任意的表达式。

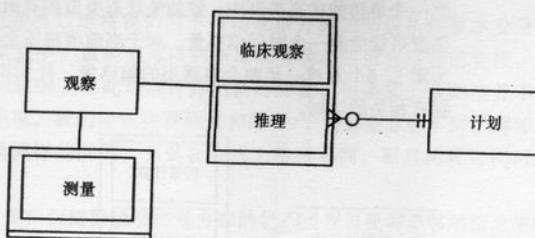


图4-9 支持计划值和实际值的观察类型

另一方面，比较计算更具有结构性。它们通常有两个输入测量，且这两个测量必须是相同的现象类型。输出测量的现象类型通常源自于计算形式以及输入现象的类型。因而，如果我们想了解销售量偏差，那么输入就是销售量现象类型，而输出就是销售量偏差现象类型。这些计算所用的公式不外乎是：绝对偏差 $(x-y)$ 或相对偏差 $((x-y)/y)$ 。

这两种计算类型之间的差别可以通过将计算测量方案子类型化来形式化地表示，如图4-10所示。计算测量方案拥有这种结构的关键元素。每一种计算测量方案都有一个单独的结果类型和一些输入类型。对于比较计算，测量方案都被限制为两个参数，且必须是相同的现象类型。所有的计算测量方案都有一种方法，该方法包含用来从输入值计算出新值的公式。两个方案可以共享一种方法，例如“参数1-参数2”这种方法就可用于确定所有现象类型的绝对误差。当然，这种情况极为普通，值得为其专门构造一种子类型来将该方法固化在类型中。

4.2.3 状态类型：定义计划的和实际的状态

由源测量方案导出的测量或计算测量方案所确定的测量通常是由它们的测量方案计算出来的。该测量方案为测量提供一种工厂方法^[3]^①。当客户要求某测量方案创建一个测量时，需要告知该测量方案他需要参考什么样的关照对象。同时还需要告知方案该测量是实际值还是计划值：如果是计划值，则要说明是何种计划；如果是实际值，则要说明是哪一天。

[69]

在这一点上讲，图4-9的模型还存在不足。我们还需要一种更简单的方法为方案提供所需的信息。图4-9的确给出了一种好办法：从已经存在的测量中确定这些信息，但是它没有提供查询该信息的简便方法。我们可以通过图4-11中的模型来解决这个问题，该模型将相关属性集中放置

① 要注意，这样做的原因是创建方法的不同，而不是最终值的类型。这也是除Gamma等人所提到的原因之外的另一个用工厂方法的原因[3]。

到一个单独的状态类型中，该抽象状态类型拥有两个子类型。实际的状态类型可能会有一个时间偏移量。对于当前值将不会有偏移量（或者偏移量为零）。6个月或一年前会有适当的偏移量。计划状态类型就像推理一样，拥有相关的计划。

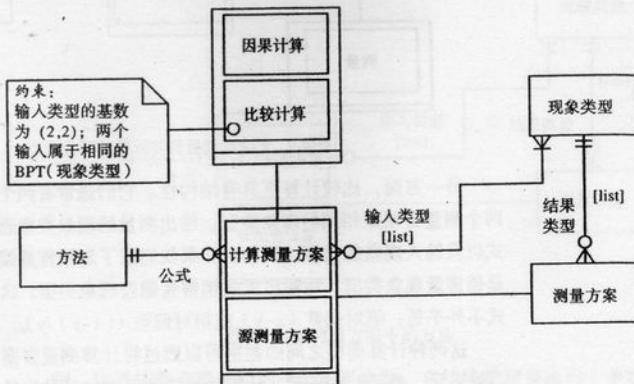


图4-10 用计算测量方案展示的计算类型

因果计算将不同的现象类型关联起来，而比较计算则显示同一现象类型的的不同状态类型之间的差异。

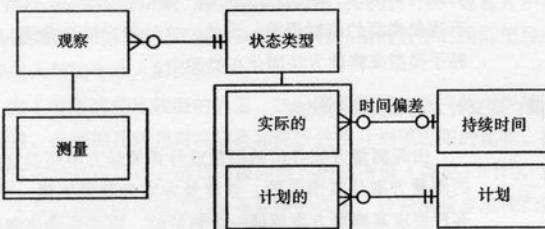


图4-11 状态类型是图4-9的一种可选方案

该可选方案使得描述所需的比较测量的种类变得更为简单（如图4-12所示）。

例：公司评估4类财务状况：实际值、前一年、年度计划以及最近的季度预测。实际值将是一个具有零时间偏移量的实际状态类型。前一年是具有一年时间偏移量的实际状态类型。年度计划是一个与年度计划相关的计划状态类型。季度预测是一个与最近的季度预测相关的计划状态类型。所有的季度预测都是计划的实例。

我们已经有效地将我们所掌握的观察方面的知识从观察本身分离到了一种类型中。该类型可以列举出现有观察的所有可能的变化。该类型属于知识级，由此可以计算出新的测量，但事实上该类型并不需要属于知识级。我们应该注意到这和图4-9中的模型有所不同。两种方法说明的是同样的问题，只是在方式上略微不同，而且两者可同时使用。

现在，客户仅需要说明测量方案的状态类型以便有足够的信息来构造测量，假定测量方案是一个因果方案。比较计算需要两个状态类型，每一个对应一个输入。一种处理方法是区分构造测量操作，使因果测量的构造操作需要一个状态类型，而比较测量的构造操作需要两个状态类型。另一种方法是允许有“比较状态类型”，如图4-12所示。我更主张后一种方法，因为这样的话，比较自身也是一个对象，构造所有测量的接口就一样了。

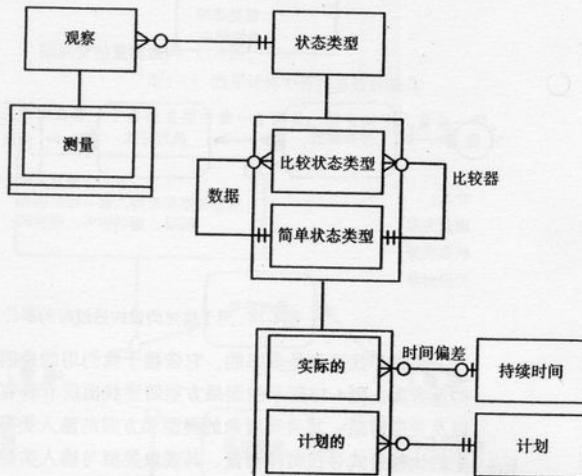


图4-12 用比较状态类型来减弱比较测量的规格说明

例：ACM的管理者想查看实际销售收入与预计销售收入之间的偏差。为满足需要，模型必须包括一种销售收入的现象类型和一种销售收入偏差的现象类型。销售收入偏差是一个用“参数[1]- 参数[2]”这样的方法得到的比较计算结果。该需求构造一种带有比较状态类型的销售收入偏差的观察。该比较状态类型将包含有预计的基准数据以及实际值比较器。

4.2.4 构造测量

既然现在我们已经知道如何识别新的测量，就可以把注意力放到测量的构造过程上，图4-13和图4-14说明这个过程。该过程有3步：寻找参数、执行公式以及用结果值构造一个新的测量对象。

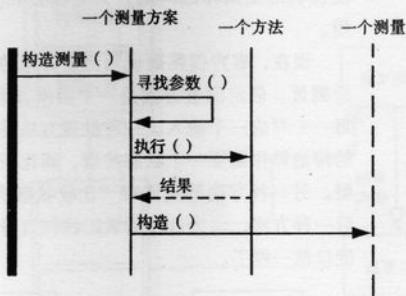


图4-13 构造测量的交互图



图4-14 用于描述测量构造过程的事件图

参数寻找操作是多态的，它依赖于我们用的是因果测量方案还是比较测量方案。图4-15所示的因果方案需要找出所有具有相同状态类型的测量以及关照对象，其关照对象的类型和方案的输入类型要相互匹配。图4-16中的比较公式寻找两种测量，其现象类型与输入类型相同，其状态类型是方案的基准数据和比较器，而且这两种测量有相同的关照对象。

在找到参数之后，就可以把它们应用到公式中，然后用所得的结果构造测量。

4.2.5 维度合并

第三类计算是要实现对维度内部值的合并。前面提到的例子是通过把东北部的所有子区域的销售值加在一起的方法来计算东北部的销售收人。

更准确地讲，对于类似于东北部这样的企业片断，其现象类型的测量是通过找出该现象类型的所有子区域的测量值，再加以计算而得到的。这些值累加在一起便生成新的值。

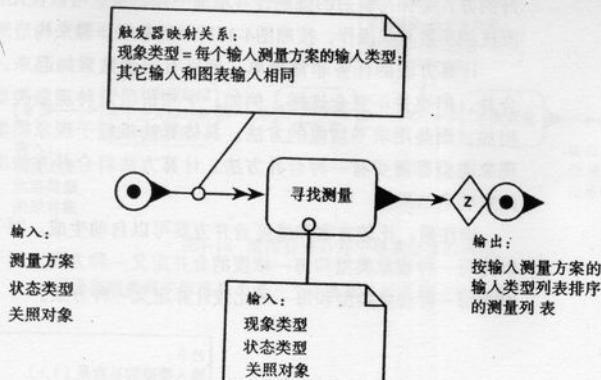


图4-15 因果计算中寻找参数的操作

该操作为每一个参数类型寻找一个测量，所有其它元素也一样。

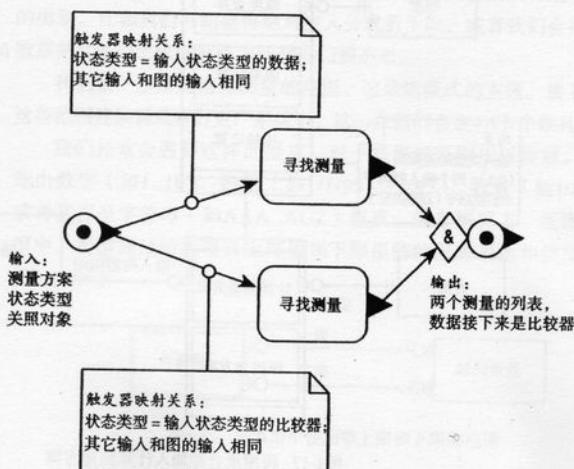


图4-16 比较计算中寻找参数的操作

该操作为比较状态类型的每一分支寻找一个参数。

这样就能像图4-17所示的那样增加维度合并方案。我们必须指明被合并的维度。该计算并不需要任何输入类型（尽管它通常和输出类型具有相同的现象类型）。可以考虑将输入类型的映射基数减小到零，但我认为更好的方式是保持映射的强制性并增加一条约束。可以使用图4-18所示经过改进的参数寻找操作，按照图4-14所示的常规步骤来构造测量。

计算方法的任务非常简单：把所有的参数累加起来。通常加法用于合并，但也并不完全这样。例如，平均价格这种现象类型就不是简单的相加，而是用求平均值的方法。具体算法依赖于现象类型，因此每一种现象类型都需要有一种合并方法。计算方法将合并方法应用于参数，以此来确定结果。

请注意：比较方案和维度合并方案可以自动生成。对于维度合并，可以为每一种现象类型和每一维度的合并定义一种方案。对于比较计算，可以为每一种现象类型和每一种比较计算定义一种方案。

74

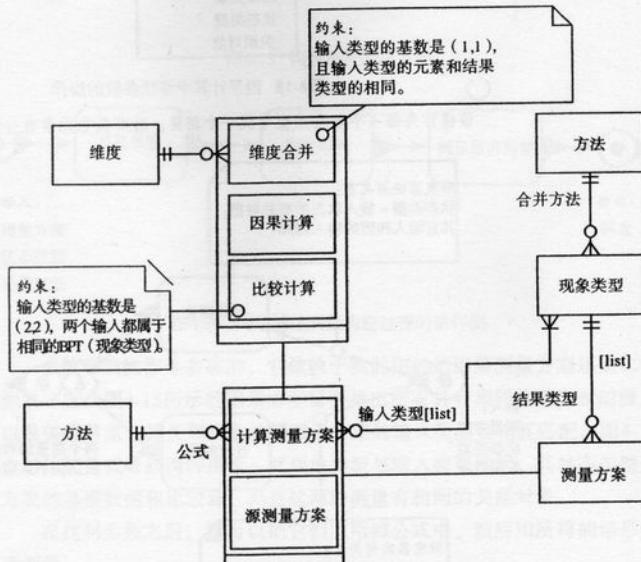


图4-17 将维度合并加入计算测量方案

计算测量在医疗保健系统中也是很有用的。我们是在本章而不是在第3章中讨论计算测量，主要原因是它们被广泛地应用在对公司财务的研究

75

中。因此，它们是一个示例，向我们展示如何把一个模型用到不同的领域，从而产生更多的思路，然后又将这些思路反馈回原来的领域。

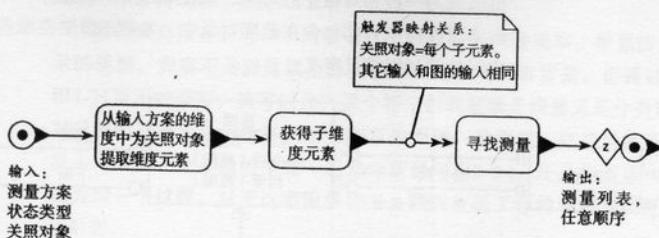


图4-18 维度合并计算中寻找参数的操作

该操作围绕指定的维度为每一个子企业片断寻找一种测量。

4.3 范围

迄今为止，我们已经探讨了如何利用计算测量和源测量来研究一个企业的财务状况。测量方案模式为我们提供了一种研究财务信息的量化方法。然而，为了搞清大量数字的具体含义，通常会对测量加以分组，形成不同的类别。比如我们可能会将绝对岁入分成若干段，或者我们会把低于基准数据的10%的比较测量作为问题专门提出来。

我们第一步是要描述测量的范围，这是该模式的主题。接下来是要把这些范围连接到观察的更广系统中，这一点我们会在4.4节中做具体的讨论。

我们经常会遇到这样的要求：对一些值的范围加以控制。该范围可能由数字（如1..10）、数据（如1/1/95..5/5/95）、数量（如10..20公斤）或者甚至是字符串（如AAA..AGZ）组成。通常情况下，范围放置在类型中，类型通过给出带有上限值和下限值的映射来表达和使用范围，如图4-19所示。

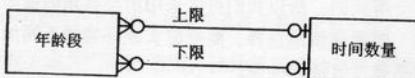


图4-19 在类型中使用带上限和下限的范围

我并不推荐这种范围表示方法，而建议采用范围类型。

该方法的问题在于：我们更多关心的是范围而不仅仅是上下限值。我们可能想知道一个实际值是否在某个范围之内，或者某两个范围是否有交

迭，或者两个范围是否相邻，或者一系列范围是否构成一个连续的范围。对于每一个带上下限值的类型都会存在这样的要求。具体的解决办法是使范围成为一种相对独立的对象，如图4-20所示。这样一来，有关范围的一些本质性的东西都包含在范围对象中，从而不需要在那些用到范围的类型中对范围加以重复定义。

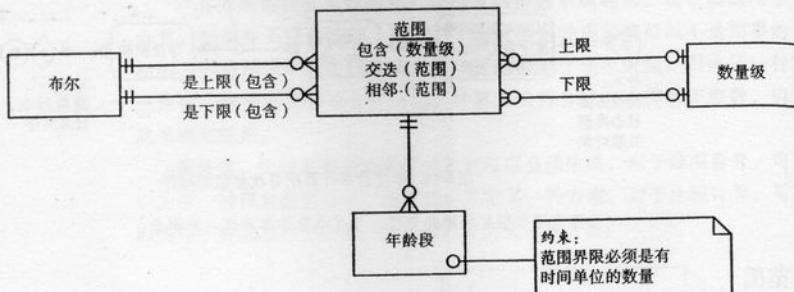


图4-20 使用一种更直观的范围对象

在需要上下限的值时通常都是这样做。上限映射和下限映射可以是随意的，这样就允许有无限大的范围，如少于6个月。要区分少于6个月和少于等于6个月时，需要使用布尔值。

一般来讲，范围可以在两个任意的数量级之间形成。在本质上，数量级就是一种定义了比较操作 ($>$ 、 $<$ 、 $=$ 、 \geq 、 \leq) 的类型。范围仅需要包含、交迭以及相邻等操作来定义自己的核心操作。类型在使用范围时，通常要指出在范围内要使用何种数量级。存在多种建模方法来表示需要何种数量级。一种方法是声明一种子类型，我在处理时段（一种数量级为时间点的范围）时即采取了这一方法；另一种方法是利用约束，如图4-20所示；第三种方法是使用类似于参数化类的方式，此时，会用到一种称为范围<整数>的类型来定义整数的范围。从概念上讲，所有这些建模技术都是等同的，所以我们可以采用所能找到的最简单的方法。在实现时，我们需要更谨慎地选择，要根据实现环境的不同加以权衡。概念模型的选择跟实现没有任何关系。

76

4.4 带范围的现象

范围为我们提供了一种定义测量分类的方法。现在，我们需要将它们同观察和测量的更广泛的模型联系起来。为了做到这一点，我们可以将某些现象类型的现象归纳成一类。如果现象类型是岁人百分比误差，就可以

构造一个有关岁入问题的现象——当岁入百分比误差低于-10%时，该现象即存在。这意味着“岁入百分比误差为-12%”的测量隐含着有岁入问题的一个分类观察（参见3.5节）。

我们需要回答的第一个问题是有一种还是有两种观察。根据图3-9所示的模型，观察不是测量就是分类观察，而不能两者皆是。但通过使用图4-21所示的模型，就可以允许某个特定的观察既是测量又是分类观察。对模型的选择取决于将概念过程首先当做一种测量，然后当做观察有岁入问题的一个单独的步骤（这意味着要用图3-9），还是将测量和观察视为同一个过程。对于这类简单情形，同我一起工作的领域专家更倾向于后者。

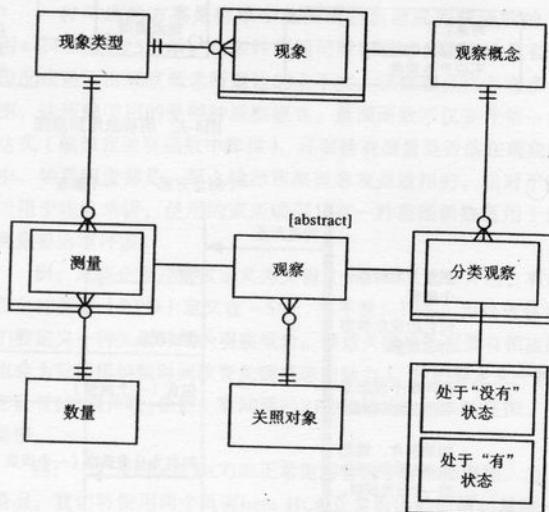


图4-21 允许观察既是测量又是分类观察

[abstract]表明一个观察必须至少是其子类型中的一种。

由于有了定义明确的范围，让计算机自动地将任意这样的测量同相关的现象联系起来就是很自然的事。为了做到这一点，我们需要一种在知识级定义范围的方法。

4.4.1 带范围属性的现象

最简单的方法是给现象增加一个范围，如图4-22和图4-23所示。如此

78

一来，当我们创建一个测量时，就可以留心观察它是否落在该测量的现象类型中的任何现象的范围之内。的确，我们需要考虑是否希望现象类型的范围不产生重叠，以及是否希望现象类型的范围是完整的。这两种情况都表明需要有一个约束。

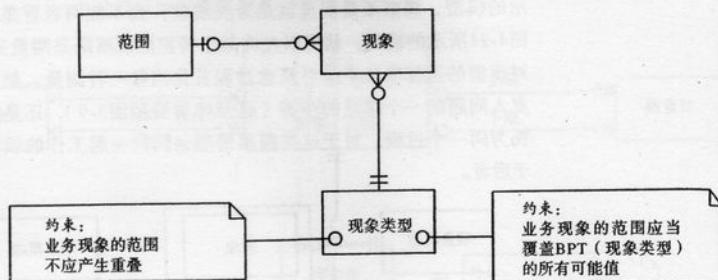


图4-22 给现象增加范围

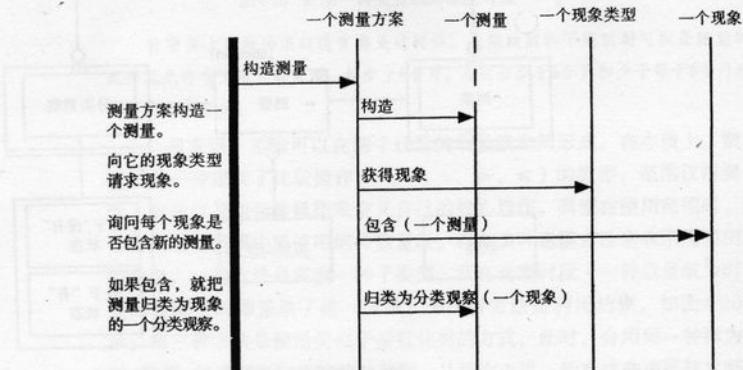


图4-23 构造测量和检查现象的交互图

检查现象的任务也可以由测量对象来完成。我倾向于测量方案，因为我认为那是一个以后很可能需要重定义的地方。

例：岁入百分比误差可分为四大类：大于5%是优等，5%到-5%是正常，-5%到-10%是要受警告，低于-10%是有问题。这可以描述成是岁入百分比误差（RPD）现象类型的四类现象。优等RPD的现象具有的范围无上限，下限为5%，正常RPD的现象具有的范围上限为5%，下限为-5%，

79

要受警告的RPD的现象具有的范围上限为 -5% ，下限为 -10% ，有问题的RPD的现象具有的范围无下限，上限为 -10% 。核实恰好为上下限时属于什么分类以及将该信息包含在范围中是非常重要的；因为我们会遇到恰好为 5% 是优等RPD还是正常RPD这样的问题？

例：体质指标通常定义成四组：正常为 $20 \sim 25 \text{ kg/m}^2$ ，超重为 $25 \sim 30 \text{ kg/m}^2$ ，肥胖为 $30 \sim 40 \text{ kg/m}^2$ ，病态肥胖为 $>40 \text{ kg/m}^2$ 。这可以描绘成四种对于体质指标现象类型的现象。超重现象具有的范围下限为 25 kg/m^2 ，上限为 30 kg/m^2 。其它每种类型都有类似的范围。

4.4.2 范围函数

一种可选的方案是将单个范围函数创建成关联函数的子类型，如图4-24和图4-25所示。在多种范围同时存在时，该方法是有用的，具体情况依赖于由观察概念所描述的上下文。该模型允许存在多个系列的范围，这依赖于用的是哪种观察概念。范围函数不仅要评估一些参数的表达式（就像在关联函数中那样），还要检查测量是否落在现象类型的范围内。如果两者都是，那么输出观察概念就是适用的。相对于将范围直接应用于现象来讲，使用约束来确保只有一种范围函数适用于任何给定的测量要困难许多。

80

例：某些企业片断被定义为关键片断。对于这些片断，有问题的岁入百分比误差（RPD）定义在 -5% ，而不是 -10% 。为处理这种情况，我们要定义一种关键片断的观察概念。那些关键片断需要有相应的观察（这也会为我们提供随时间改变关键状态的能力）。我们要定义一种范围函数，它具有{关键片断}参数、有问题的RPD的输出、 $<5\%$ 的范围、RPD的现象类型。

例：一个人的beta HCG的正常范围会由于怀孕而增加。为了表示这种情况，我们将使用两个具有beta HCG正常输出的范围。其中一个范围具有怀孕参数，而另一个具有非怀孕参数。范围函数上的现象类型是beta HCG。

这两种方法都有它们各自的优点，如果把它们混在一起使用就会显得不伦不类。直接和现象联系在一起是最简单的方法，如果它确实可以正确地描述情况，那么它就是首选的方法。范围函数要更复杂一些，但是它可以描述更复杂的情况。一般的原则是：在能够使用时应该尽量使用直接和现象关联的方式，不得已才考虑使用范围函数。如果情况变得比这里所描述的模型所能够处理的情况更复杂，就应该给范围函数增加特性，要么直接增加，要么通过子类型。

81

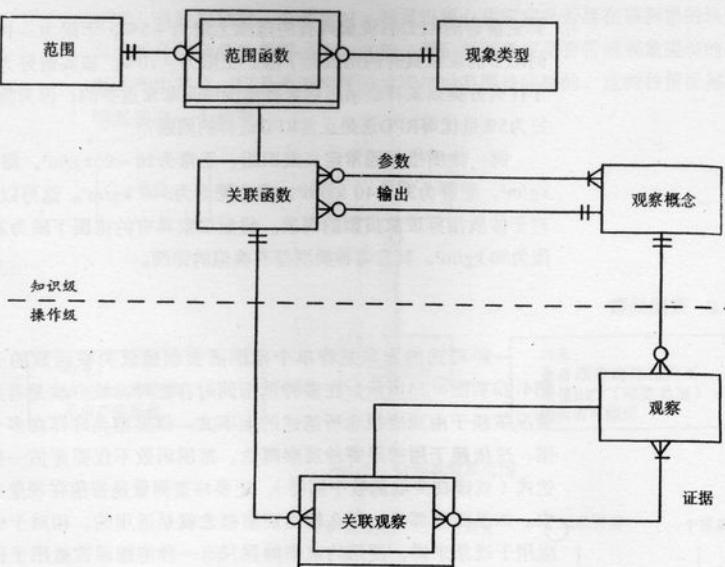


图4-24 范围函数

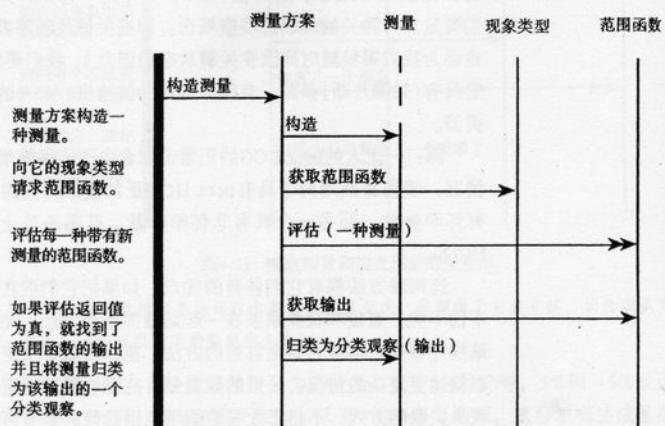


图4-25 构造一个测量并检查范围函数

4.5 使用最终框架

迄今为止，本章已经阐述了对第3章中所介绍的模式做进一步的扩展后所获得的模式的大体情况。现在我们可以来探讨如何使用这些模型。

我们从查找ACM的全部岁入开始。如此，会用到一个测量，其企业片断是整个企业；也就是说，企业片断的维度元素都处于维度层次的顶层。测量通常不是一个绝对值，而是一个和特定计划值或前段值相比较的相对值。此外，可以通过一个带范围的现象来凸显存在的问题。分析人员接下来就可以通过查找由现象所定义的有问题的观察来着手开展进一步的工作。

为了断定问题是与设备销售收入相关，我们需要从总岁入往下分析（总岁入是由销售收入减去销售成本构成的因果计算）。要注意的是：因果计算指出一种可能的分析途径，而无论测量是否由该方法确定。也许该最终数值事实上是从一个数据库中获得的。（由于脏数据的原因，也许它与公式的计算结果并不完全吻合。）

下一步是利用维度合并方案。沿着地域维度观察，会发现东北部片断的误差明显偏高。我们现在就可以将注意力集中在这样一些企业片断上：企业片断的地域维度指向东北部维度元素，并且在其它所有维度的顶层。重复这一过程两次以上，我们就会得到{东北部、1100系列产品、政府部门}的企业片断。

这里存在一定程度的间接性。比如，当涉及比较计算时，计算方式可能就不那么直接。整个岁的误差可能并不等于销售收入的误差减去销售成本的误差。一种替代方案是先单独计算出实际和计划的销售收入，然后在因果计算中使用。对于绝对误差，两种方式都是可行的，但对于百分比误差，就不行了。方案的存在与否将决定最合适的计算方式是什么。

我们还可以选择其它方式。上面我们先进行因果分析，然后做维度合并。我们还可以先分解地域维度，然后进行因果分析，然后再分解其它维度。分析的途径可能有很多种，并不是每种都需要同数据计算时所采取的途径保持完全一致。

使用3.11节所阐述的关联函数，我们可以得到定性的描述，如“一个强有力的竞争者可能导致销售收入的减少”。定性和定量的观察是通过指定的范围现象关联起来的。

具体应用中可以采用多种技术来浏览这种结构。现在的多维数据库大多都支持用户定制，从而提供最大程度的适应性。另一种可选的方式是采用由方案层次所定义的分解途径，这种技术在迅速到达问题根源方面被证实是有效的，层次分析器可以很容易地被构建在该框架的顶层。另一种

方法是使用代理在结构中游历，以找出所关注的测量。

本章做了一次具体的尝试，即将医疗保健系统中的模型应用到公司财务分析中。在公司财务分析中对这些模型所做的扩展又可以反馈回医疗保健系统模型中。测量方案的确是可用的；企业片断模式可应用于流行病学研究，尽管还需要做进一步的分析。通过允许对模式做类似于这样的跨领域应用，我希望会出现越来越有用的模式；如果我们倾向于把模式固定在特定的应用领域，估计就不会有什么模式出现。

参考文献

1. Cairns, T., A. Casey, , M. Fowler, M. Thursz, and H. Timimi. *The Cosmos Clinical Process Model*. National Health Service, Information Management Centre, 15 Frederick Rd, Birmingham, B15 1JD, England. Report ECBS20A & ECBS20B <<http://www.sm.ic.ac.uk/medicine/cpm>>, 1992.
2. Dejesus, E.X. "Dimensions of Data," in *Byte*, April 1995, pp. 139–148.
3. Gamma, E. R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
4. Peterson, S. "Stars: a pattern language for query-optimized schemas." In *Pattern Languages of Program Design*. J.O. Coplien and D.C. Schmidt, ed. Reading, MA: Addison-Wesley, 1995, pp. 163–177.

第5章

引用对象

大多数面向对象方法都将注意力放在对象识别上。在面向对象计算机系统中，每个对象都拥有惟一的ID，利用该ID能够确保可直接访问到任何对象。这种观念同时也影响了我们的概念思维。只有很少的对象方法使用主键和次键概念，虽然它们在传统的数据建模中扮演了非常重要的角色。我们也需要有某种方法来引用一个特定的对象：例如，我可能需要找一个要付给他账单的人，而一个医生可能需要将一个患者标记为糖尿病患者。对象系统为我们提供了强大的浏览能力，这种浏览能力揭示了概念对象之间的自然联系，但有时候确实需要一种更直观的标识符。

针对对象的最简单的标识符是名称（参见5.1节），名称由一串字符组成，通常用来标识一个对象。问题在于并不能确保名称可适用于所有的情形。可能需要一种更人工的构造物：由标识方案（参见5.2节）所限定的标识符。

我们本来认为对象是易定义的和静态的，但当我们意识到通常情况下并非如此的时候，问题就变得更为复杂。在计算机以外的世界中，很容易找到我们认为是两个对象而实际是一个对象的情形。对于这种情况，我们需要做对象合并（参见5.3节）。既然我们在合并过程中可能会犯错误，那么我们就可能需要到后面将它们分拆。可以通过复制并替换、替代或者本质/表象等方式来合并对象。有时候，有一些分离的对象可能本应该是一样的，但是我们不能十分肯定，或者我们不能和其他相关人员达成一致。此时，我们只能说存在对象等价（参见5.4节）。

要记住，本章所阐述的内容是对对象的概念性引用，即人们通常所使用的引用方式。这些引用出现在除软件所使用的任何对象标识方案之外的模型中。本章不讨论任何软件标识技术，但是会假设它们存在于任何面向对象的实现中。我还会假设这些软件标识技术对于用户是隐藏的。

关键概念：标识符、标识方案、替代对象、对象本质、对象等价

5.1 名称

在我所讲授的一门面向对象设计的课程中，我使用过一个习题，包括

对一个人的出生情况的各项细节的记录。其中一个要求就是需要记录某个人的出生医院和出生城市。就常识而言，如果知道某个人的出生医院，那么很自然地，就应该知道其出生的城市，因为每所医院只能在一个城市中。但事实并非如此，有人会指出情况不是这样的，因为世界上很多城市都有一所圣·玛丽医院。

该错误是逻辑上和哲学上最古老的问题的一个表现，即事物名称和事物本身之间出现的混淆。一所医院不是一串字符所能代表的：正是因为建筑、组织、人员、法人实体等众多的事物使得怀特岛的圣·玛丽医院和伦敦的圣·玛丽医院不同。很明显，如果人们实际遇到了这些对象，就不会真的把这些对象搞混。就像可能存在拥有同样名称的医院对象一样，名称仅仅是一串和医院相关的字符，而不是医院本身。我们为对象本身建模，而不是为对象名称建模，这样就可以有充分的理由称每一所医院都坐落在惟一的一所城市中。

什么是名称？名称只是一种标识对象的非正式的方法。我强调“非正式”是因为相对于其它特性，名称用起来更加随意和方便。“Martin”这个字符串是一个有用的标识符，它在许多场合都足以用来标识我。但是我曾经和另外一个叫Martin的人共住一所房子。两位居住者共享了该字符串，所以它作为标识符的价值就减小了。在我们的朋友圈中，“Martin”仍然是最通常的用来标识我们俩的方式，但是由此而产生的混乱也的确发生过。在很多应用中，我们认为有必要给一个人起一个单独的名称，如图5-1所示，尽管该名称可能是有结构的。更复杂的例子是允许一个人有别名，给他很多名称，如图5-2所示。例如，可以用字符串“Martin F”来标识我，以便和另一个Martin区分开。

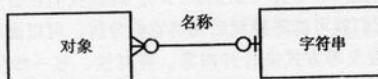


图5-1 有一个名称的对象

该模型表明并不是所有的对象都必须要有一个名称。你可以争辩说：没有名称就暗示着和一个空串存在必然的联系，因此映射是强制的。在任何一种情况下，这种模型都表明一个字符串可以被作为多个对象的名称。在概念上，所有相等的字符串都是相同的字符串；也就是说，你没有完全相同的复制。

通常，名称是标识对象的一种有效方法，但是在建立人事档案系统时，不会把一个人的名字作为这个人的唯一标识。人可以有很多名字，同样的名字也可能被不同的人使用，而且人还会改名。所有这些因素使得名称

成为不可靠的标识符，但是到目前为止，名称仍然是最普通的标识符。

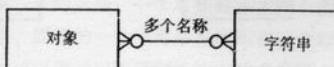


图5-2 有多个名称的对象

该模型模拟有多个别名的对象。一种变形可以是有一个（常用的）名称和多个别名。

要特别留意名称和标识符的另外一方面：名称是刻画对象的一种简洁的方式。它可以描述一个对象的某些属性。比如，某款汽车叫16GL，是指其引擎大小和舒适度。尽管该名称是对该款车的简洁描述，但它不是一个标识符，因为许多车型都可以称为16GL。

一个真正的标识符有这样几个特点：它必须可靠地为用户指明一个且是惟一的一个对象，同时无论何时使用该标识符，它都必须是指称同样的对象。图5-3显示了一个标识符的公共模型。与通常的拥有基础对象的情形不同的是，从标识符到对象的映射是单值的。

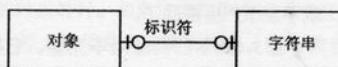


图5-3 一个标识符对应一个对象

该模型表明，并不是所有的对象都有一个标识符，这在概念上可以是真实的，即使它在软件系统中并不是真实的。标识符是一个字符串，但并不是所有的字符串都用来标识一个对象，要成为一个真正的标识符，它应该标识的是一个惟一的对象。如果使用一个标识符类型，将是非常可取的一种做法，这时这个标识符类型与一个对象之间就存在强制的映射关系。

87

5.2 标识方案

在简单系统中，典型的做法是给每个对象一个单独的标识符；但是在更复杂的系统中，一个对象往往会有多个标识符。医疗保健行业就有很多套用来标识患者的方案：比如每家医院会分配给患者一个病例号，而各科室又会分配给患者单独的号码；银行界也有好几套用来标识银行的方案：如SWIFT、sort codes（分类编码）、CHAPS等。图5-4中的模型给出了更一般化的描述方法。

例：世界卫生组织的疾病国际分类法用E10编码来标识I型糖尿病。这可以以字符串“E10”作为标识符，而标识方案是ICD-10，对象是I型糖尿病。

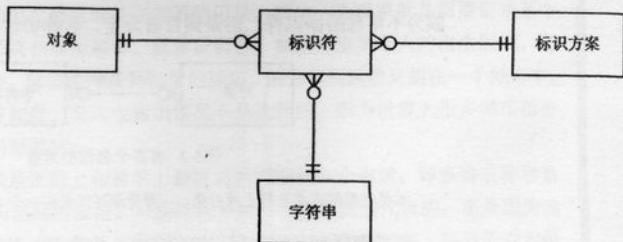


图5-4 标识方案

例：假设我有一个编号为“123456”的护照。这可以用字符串“123456”作为标识符，标识方案是英国护照，对象是我自己。当然，根据实际情况，对象可以就是我的护照。

标识方案表征了用于标识一个对象的上下文。一个账户会有单独的SWIFT和CHAPS编码。同样的字符串序列在SWIFT和CHAPS编码体系中表征的是两个不同的银行，但是如果这些字符串处于不同的标识方案中，就不至于产生混乱了。

图5-4所示的模型尚不完整。它所给定的格式还很粗略，不足以阻止在同一方案中用同一个字符串来表征多个对象。在这里还需要用到一个概念，即惟一性约束[1]，用以表明映射的一个特定组合必须有针对一个对象类型的惟一值。

我们可以考虑给针对标识方案和字符串组合的映射关系一个惟一性约束。该约束将声明两个标识符不能具有相同的标识方案和字符串组合。由于从标识符到对象的映射是单值的，因此标识方案和字符串的组合就标识了一个惟一的对象——这也正是我们所需要的。另外一种思路也值得考虑，即假定对象和字符串存在惟一性约束关系。这种约束关系将声明一个特定的对象和一个特定的字符串只能引用一个特定的标识方案。换句话说，一个对象不能在两个不同的标识方案中用同样的字符串来表征。这种惟一性约束类型既不太行得通，也不太方便。人们通常喜欢在不同的方案中使用相同的字符串，使得他们不必记住太多的标识符。银行卡中的个人身份号码（PIN）和社会保障号码就是两个实例。

标识方案和对象的惟一性约束表明，在每个标识方案中只能用一个字符串来标识一个对象。这样在一个标识方案中就不能使用别名。别名是有其一定的用处，但并不是必需的；它们也有不方便之处，特别是在人们将标识符和对象搞混淆的时候，当然，它们也未必会带来灾难性的后果。针对所有三元映射的约束可以防止产生无用的标识符副本，但不会在本质上

改变整个方案。

惟一性约束的第二条约定是：一个对象的标识符不能改变。这就意味着在一个标识方案中，同样的一个字符串只能用在一个对象上，而不能再用在别的对象上。这可以通过以下强制性的方式来保证：即确保标识符不能被删除，并且起自标识符的各种映射也是不可改变的——也就是说，它们一旦产生就不能再改变。一旦一个标识符被分配出去，那么它就永久地被分配了。在现实生活中，一些标识方案的确重新使用了标识符，但是只有那些从未被使用过的标识符才可以被重新使用。

在一个典型的面向对象语言中，惟一性约束是如何实现的呢？其中，标识符的不变性就起了相当大的作用。不变性不允许软件中的映射更新，所以没有一个公共的调整操作。映射必须在创建操作过程中通过以参数的方式来传递值而建立。在创建过程中，需要进行合法性检查，以确保其它的标志符不会跟它一样，具有相同的（组成惟一性约束关系的）映射组合。

通常，标识方案要负责检查由其标识符所用的字符串的格式。这种检查要在创建标识符的时候进行。如果该字符串中嵌入了任何关于所引用对象的有意义的信息，那么该信息也应该被检查。我可能拥有一个叫U123的标识符，这里U表示我住在美国。如果我返回英国的话，该标识符将会带来问题。一般来说，把一个对象的特征信息嵌入到标识符的字符串中是一种不好的做法，因为这样做就意味着该字符串将随着特征的改变而改变。明智的做法是生成一个单独的字符串来提供这类简洁的信息。

89

5.3 对象合并

我们习惯把对象想像成是一成不变的：一旦对象被标识，就永远这样去标识了。现实生活中，事情并没那么简单。请设想一个患者到某家医院去看病，几天之后，医院的工作人员发现这个患者同时还是另一个科室的非住院患者，但是此时他们已经在同一个计算机系统中为他建立了两个单独的记录。这种情况很普遍，而且在工作人员发现针对这个患者有重复记录前可能已经过去了几个星期或几个月的时间。

存在重复记录不仅影响计算机系统，也影响医院工作人员的理解。意识到你现在按照左心室功能衰竭症对待的患者在一年前还曾是一名甲状腺功能亢进患者这一点，不仅对计算机系统，更对整个治疗过程有着非常重要的意义。我们需要有一种概念上的机制来把两个对象联系在一起。

我将概略地论述针对这种情形的三种策略：复制并替换、替代以及本质/表象。

5.3.1 复制并替换

通常，我们认为第一种策略是把一个对象的所有属性复制到另一个对象，并且删除被复制的对象（复制并替换）。旧的被删除对象的标识符要发生改变，使它映射到保留的对象，这样就打破了不变性规则。在允许别名标识符时，这种策略是可行的，但是在处理软件中对被删除对象的任何引用时，就会出现问题。这样就存在“虚悬引用”的风险，它通常会带来不良后果，除非你可以捕获到所有的引用关系。

例：John Smith进入急诊室接受治疗，医院分配给他JS777这个号码。后来该医院发现他以前在本医院的注册号是JS123。JS777对象的信息必须要加载到JS123对象的记录上，所有对JS777对象的引用要转移到JS123对象上，而且要删除JS777对象。

5.3.2 替代

第二种策略是进行对象替代（如图5-5所示）。一个对象被分类为被替代对象，并且和另一个对象（主动对象）链接。将来，所有的工作都是针对主动对象的，被替代对象因为历史原因而保留下。不需要替换对被替代对象的引用。可以把被替代对象中的当前数据复制到主动对象中，也可以是任何到主动对象的消息都必须反映到所有被替代对象的数据中。所有给被替代对象的消息都被授权由主动对象来处理。如果所有的数据都被复制了，那么主动对象就可以完全忽略被替代对象的存在。

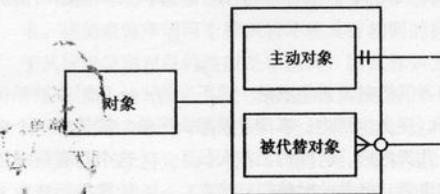


图5-5 被另一个对象替代的对象

例：利用替代策略，对象JS777被标记为被替代对象，而对象JS123被标记为活动对象。任何发送到JS777的消息都被授权由JS123来处理。

例：研究人员曾发现两种不同类型的肝炎：血液传染型肝炎和非A非B型肝炎。后来这两种肝炎被认为是同一类型的肝炎，并且被称为C型肝炎。这种情况可以通过用主动对象C型肝炎替代血液传染型肝炎和非A非B型肝炎，并且将它们联系起来的方式来描述。

从概念上来说，复制并替换策略和替代策略在很大程度上是相同的。惟一的区别在于你可以看清楚最早附属于被替代对象的东西。以下这一点是重要的：如果医院在没有发现Smith先生是两种病的患者时就做了一些治疗的话，只有替代策略可以准确地反映已经发生过的事情。

5.3.3 本质/表象

最后一种可以考虑的策略是本质/表象模型，如图5-6所示。对象保留了自己的大部分内容，但是它的背后还有另一个对象——对象本质。对象本质的存在仅仅是为了把各个相关的对象关联起来；它没有其它的属性。在这种策略中，是通过把多个对象同一个单独的对象本质连接起来而实现合并的。这就意味着当一些消息传递了更改信息时，对象必须知道它们的其它表象，而且在响应时会对它们加以考虑。

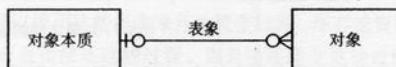


图5-6 对象本质和表象

91

例：利用本质/表象策略，就产生了一个将JS123和JS777作为其表象的新的对象本质。

例：这种模型并不能很好地适用于前面所举的肝炎的例子，因为血液传染型肝炎和非A非B型肝炎的概念已逐渐不再使用，而C型肝炎的概念已渐渐地被人们所接受。

上面的讨论主要集中在合并对象上；然而，到后来可能又需要取消合并。比如，在将两个患者的记录合并后，医院可能会在几个月后发现他们的确是两个不同的患者。如果采用的是本质/表象策略的话，那么再将对象分离是很容易的事，因为它保留了原始对象。可以这样讲，如果在很长一段时间内都无法确定是否真的要进行对象合并的话，本质/表象策略是最好的一种策略。

例：如果发现两个John Smith是完全不同的人，就必须消除把他们联系在一起的对象本质。

5.4 对象等价

前面几节重点都放在阐述一个对象如何才能由不同的人用不同的方式来标识。而此处将着重讨论不同对象的相似性问题。例如，医学术语使用许多意义相近但程度不同的标准词汇来定义不同的临床情况。然而，重点就在于“程度不同”上。“意义相近但程度不同”的具体定义可以是非常

准确的，但有时也不完全是这样。为了处理这种不精确性，人们为医学术语建立了不同的编码方案，这意味着有多个编码方案可供选择。

我们可以选择其中一套编码方案作为自己的术语标识方案。这样一来，如果某个临床医生要描述一系列特定的生物现象，他就可以通过把编码方案当成是一种标识方案来标识这些生物现象。其他的临床医生也可以这样做。这样就便于信息的交流和传递，最起码是可以做到编码方案粒度这一级。在这里，可能忽略一个重要的问题，即此时等价还没有被普遍认可。也就是说，有些团体可能会认为两个对象是相同的，而别的团体并不这么认为。图5-7的模型通过定义一个由某些团体所认同的等价来解决这个问题。一个团体只有在认同了该等价之后才可以使用它。

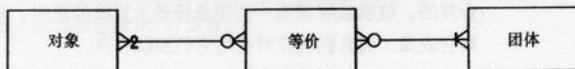


图5-7 对象之间的等价

例：许多医生认为G型肝炎和GBC型肝炎是一种病，但这并没有被普遍认可。这时就可以用这两种病症之间的一个等价来表征。如果一个医生想要一个G型肝炎患者的清单，而且该医生是这种等价的认同者，那么那些GBC型肝炎患者也会被列入其中。

参考文献

- 92
93
94
1. Martin, J. and J. Odell. *Object-Oriented Methods: A Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

第6章

库存与账务

在商业计算系统中，大多数程序都被设计用于跟踪企业的资金流动，也就是记录钱是怎样赚来的，又是怎样花掉的。而跟踪库存和账务的主要思想就是要记录钱和货物是如何一笔笔流动的。

本章中的库存和账务模式就是因此而产生的。它们给出一个核心的概念集合，我们可以使用这些概念来作为资金核算、库存或资源管理的基础。这些模式并没有直接描述这些过程，而是描述建立这些过程的基本思想。第7章介绍一个简单的例子，该例子使用这些思想来记录电话账目清单。

在本章中，我使用一个简单的个人财务示例来解释库存和账务的基本思想。我所使用的术语有别于传统的财经账务术语，尽管它们大同小异。因为我要建立一个更为抽象的模型，而这需要一些新的名词和术语。本章中模式的独特之处在于：它们展示了处理规则是如何被嵌入到账目系统中的。这种方法允许账目自行更新和管理，从而将一个传统的被动型账目记录系统转变成一个主动型账目记录系统，我们可以按照合适的方式，通过搭配账目清单来配置该系统。

第一个模式是有关账目（参见6.1节）的。一个账目包含着有价值的东西——货物或者资金——它们只能按条目方式进行增加和删除。条目保存着账目的所有变更历史。当我们用一个账目来记录一个值的变更历史时，很重要的一条就是要确保不能丢失任何条目。事务（参见6.2节）把所有条目关联起来，提高了可审计性。在一个事务中，从一个账目上取下的条目必须存放到另一个账目上；条目不能被随意创建或销毁。

存在两种事务：双腿事务——把一笔账从一个账目转移到另一个账目；多腿事务——可以在多个账目中拥有条目，只要事务在总体上始终保持平衡。

可以使用汇总账目（参见6.3节）把多个账目组合集中起来，汇总账目可把大多数针对单个账目的处理（报告行为）应用到成组账目上。有时我们需要生成一些不要求其保持平衡的账目条目，这时可以使用备注账目（参见6.4节）。

账目可以包含固定的规则，由这些规则来控制账目间的金额转移。我们可以使用记入规则（参见6.5节）来建立能互相更新并能反映业务规则的动态账目网络。为此，记入规则的实例要有自己的可执行方法，该需求又引出一个重要的建模概念——个体实例方法（参见6.6节）。个体实例方法可以用单个子类型、策略模式、内部的case语句、解释程序和参数化方法的某种组合来实现。

记入规则的执行模式（参见6.7节）描述触发记入规则的具体方式：一旦创建事务时触发；请求账目来处理相关规则；请求记入规则来触发；或者请求账目自动更新，从而向后链式触发其前驱。

要在多个账目中使用记入规则，需要定义一种多个账目的记入规则（参见6.8节）。一种方式是使用知识级，即将记入规则定义在账目类型上。另一种方式是将记入规则和汇总账目关联起来。

在账务系统中，各种对象都需要提取账目条目和账目结算的子集，两者都需要一个选择条目（参见6.9节）的模式。在我们想从多值映射中获取对象选集的时候，这个模式就会派上用场。替代方案还有：返回整个集合并由用户自行选择，针对账目做一些额外的操作，或使用账目过滤器。

我们可以使用账务实践模式（参见6.10节）把记入规则的巨大网络划分成若干的组。在长时间的计算中，我们经常需要返回头去了解为什么不同的事务会给出这样或那样的结果，此时我们需要用到条目来源模式（参见6.11节）。

结算单和所得计算书（参见6.12节）用于区分现有资产账目和流动资产账目。不同的人对账目的看法是相近的；例如，对于我的银行账户，我和银行的看法就基本雷同。其中一个是另一个的对应账目（参见6.13节）。

作为最终结果的模式非常抽象；而在实际应用中，一些特殊情形需要专门化的账目模型（参见6.14节）。这样的账目就通过子类型化通用的账务模式而得到。

本章最后一个模式描述如何登记条目到多个账目（参见6.15节）。在有不只一种报告条目轨迹的方式时，该模式就能派上用场。另外还有两种替代技术：使用备注条目或使用派生账目。当我们只想要账目的报告功能（不要结算和审计功能）时，我们可以选用派生账目来替代账务模式。

这些模型是总结几个项目中的想法而得到的。它们起初是我在为一个美国服务公司设计开发客户服务系统时想到的，而且我在为一个国际电话公司检查账务结构的过程中对它们做了进一步的发展和完善。最近，我在为一个美国制造企业开发付账系统时又一次对这些模型做了深入的完善。

关键概念：账目、事务、条目、记入规则

6.1 账目

在很多领域，不仅强调要记录某个事物的当前值，而且还强调要记录影响该值的每次变化的具体细节。一个银行账目就需要记录每一次的存款和取款；而库存记录要记住每一次的物品出/入库情况。

一个账目类似于一个数量属性，再加上一系列条目（数量属性的值每做一次改变，都会附加一个条目）（如图6-1所示）。结余反映账目的当前值，是所有与账目相关的条目共同作用的结果。这并不意味着结余值在每一次被访问时都需要重新计算。结余的派生值可以被临时缓存，即便该缓存值对用户是不可见的。通过使用条目，用户可以查看一段时间内账目的变化情况以及存取的总量（参见6.9节）。数额的符号表示该条目属于存入还是取出。一个账目清单包含一段时间内账目上的所有存取条目。

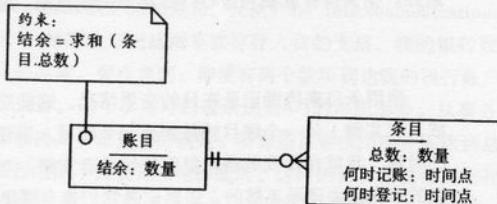


图6-1 账目和条目

条目记录账目的每次改变。

[97]

例：我从自己的户头上取出100美元。这可以用一个连接到我的账目的金额为 -100美元的条目来表示。

例：我从商店买了4令^Θ标准信纸。商店会用一个连接到标准信纸的数量为 -4令的条目来表示。

例：一月份我用了350KWH（千瓦小时）的电。这可以用一个连接到我家用电账目的数量为350KWH的条目来表示。

建模原则：要记录一个值的变更历史，可以为这个值设立一个账目。

计算结余的一种具体实现方式就是把条目集中起来形成一个数值集合。Smalltalk语言就有一个特殊操作collect来做这件事。这样做有一个危险，那就是collect操作总是把对象按原样收集到集合中；这样一来，在一个条目集合上运行collect就会产生一个数量集合；而集合是不允许有重复数值的，所以如果存在有相同金额的条目，就只会把一个条目的数额计算

^Θ 令是纸张的一种计数单位，以前以480张为一令，现在以500张为一令，而印刷中的一令为516张。——编辑注

在内，这样得出的结余值就不正确。为了把这些基础数据都收集起来，通常采取的方式是使用允许重复的袋（bag）。在C++中，这个问题不太常见，因为在C++中很少使用集合操作；C++使用者可以使用一个没有这个问题的外部迭代器[1]来解决这个问题。此外，为验证正误，在测试时所用的测试用例也要包含具有相同数额的条目（甚至是所有属性都相同的条目）。

图6-1表明，每个条目需要记录两个时间点：一个是办理时间，另一个是条目登记到账目的时间。这在需要追溯收费^⑥时尤为重要。办理费用在办理时间和登记时间之间可能会有所变化，所以这两个时间都是需要的。我们既需要知道事件的历史记录，也需要知道有关该历史记录的其它情况（参见15.3.1节）。时间点既要包含日期也要包含时间；很多应用只需要包含日期就足够了。

例：4月1日我在Jae's Café吃了顿饭，信用卡公司在4月4日收到付账通知。相关条目就要包含4月1日作为记账日期，而4月4日作为登记日期。

6.2 事务

使用条目来协助记录账目的变更情况。这些变更通常包括把物品（钱或其它实物）从一个账目转移到另一个账目。当我从我的银行账目中取款的时候，我就在往我的钱包里面放钱或者支票。对于很多物品仅仅记录它们的来来去去还是不够的，还要记录它们来自哪里和去向哪里。

如图6-2所示，事务明确地把取款条目和存款条目联系起来。双条目法反映一个基本的账务原理，就是钱（或其它我们要核算的东西）永远不能凭空生成和消失，它仅仅是從一个地方移到另一个地方。

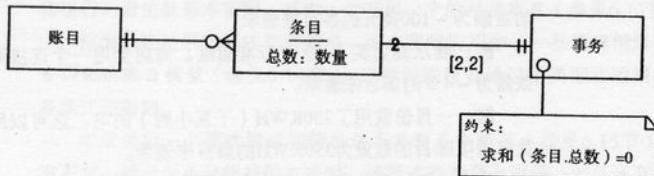


图6-2 带两个条目的事务

例：我用信用卡购买机票，要支付给波士顿航空公司500美元。这是一个金额为500美元、从信用卡账目到波士顿航空公司账目的事务。稍后，我将生成一个从我的支票到信用卡的事务来把信用卡的结算值恢复到零。

例：芳香咖啡机制造公司（ACM）从纽约运送5吨阿拉伯穆哈咖啡到

⑥ 追溯收费是指从以往某天开始生效收费。——编辑注

波士顿。这是一个数量为5吨、从纽约账目到波士顿账目的事务。

在复杂的账务结构中，我们的目的是要使账目达到平衡，也就是说，在业务周期上的各点都要使结算值归零。通过守恒原理建立模型，我们很容易发现系统中的不守恒之处。虽然在使用账目的时候事务并不是必需的，但我倾向于多使用这一概念。

建模原则：在使用账目时要遵守守恒原理：需要清算的物品不能凭空生成和消失，它仅仅是从一个地方转移到另一个地方。这使得发现和防止不守恒变得容易。

多腿事务

图6-2表明，每一个事务由一个取走和一个相应的存入动作组成。事实上，我们在一个事务中可以包含很多个取走和存入动作。比如，我可以从Megabank取走3000美元，又从Total Telecommunications那里取走2000美元。然后决定把这两笔款都存入我的支票。我的银行账单将显示5000美元的存款。要注意到：即便有两个款项到达我的银行账户，但只会显示一个条目。这个事务可以表示成图6-3的多腿事务。从事务到条目的映射上限被提高。最重要的就是：所有条目要依照事务来达到总体平衡，对于各条目之间不需要是完全匹配的。这样的话，我就可以把我的银行账目状态建模成有三个条目的事务：[账目：支票账目，金额：5000美元]、[账目：Megabank，金额：3000美元]、[账目：Total Telecommunications，金额：2000美元]。事务负责确保钱没有凭空产生和消失。

99

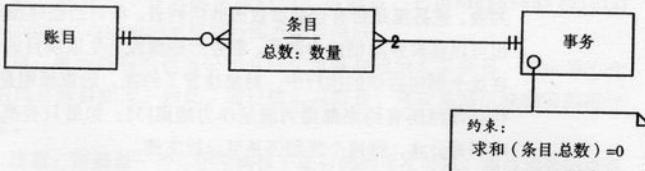


图6-3 多腿事务

生成事务时，多腿事务模型比双腿事务模型更有弹性。

例：芳香咖啡机制造公司（ACM）从纽约运走5吨爪哇咖啡，2吨运往波士顿，3吨运往华盛顿。这是单个事务，并带有三个条目：[账目：纽约，-5吨]、[账目：波士顿，2吨]、[账目：华盛顿，3吨]。

双腿模型是多腿模型的特例。在双腿模型中，事务只有两个条目。在

有些应用中，双腿模型占主流，这种情况与图6-4中的模型类似。其它一些应用可能会有大量的多腿事务。因为多腿事务更加有弹性，所以我建议使用多腿事务。在多腿事务中通过一个特殊的创建操作可以很轻易地生成双腿事务，具有很强的方便性与适用性。本节的其余部分都假定使用多腿模型。

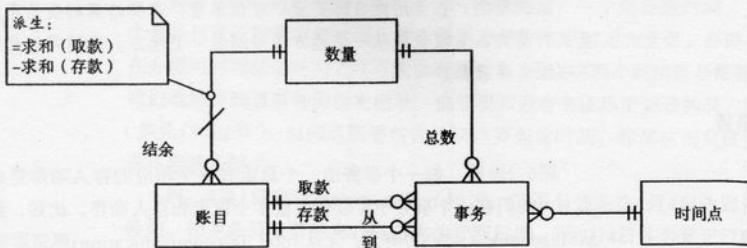


图6-4 不使用条目的双腿模型

在所有使用双腿事务的地方都能见到这个模型。该图和图6-2具有相同的效能。但我还是乐意使用图6-2，因为可以更容易地将它转化为多腿事务。

100 事务和条目之间的相互制约关系引出一个“先有鸡还是先有蛋”的问题。因为存在约束关系，所以如果不创建一个事务就不能生成一个条目。类似地，如果没有一个条目就不能创建一个事务，因为对事务有同样的约束。

一种解决方法是：提供事务的一个创建操作，它使用局部定义的条目列表，甚至是带有合适参数的数组列表。条目的创建操作被声明为私有，但可以在事务的创建中调用。事务的创建成为生成条目的惟一地方。显然，在这个创建操作的执行中，对象违背了约束。约束规则是：所有的公共操作必须以所有约束都得到满足作为结束[5]。如果只有事务的创建例程被声明为公共，则这个规则还是可以被实施。

6.3 汇总账目

在账目系统中，把账目组织在一起往往会有很大的用处。比如：我可能想把我的Total Telecommunications和Megabank的账目组合成一个业务收入账目；类似地，我想把租金和食物放到我的个人花销账目中，把差旅费和办公费放到业务花费账目中。由细目账目和汇总账目所构成的简单层次模型就可以支持这类结构，如图6-5所示。

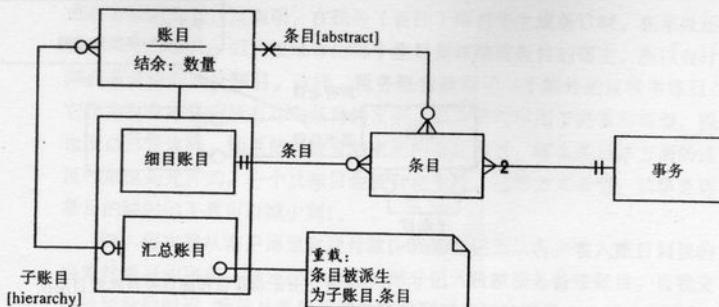


图6-5 汇总账目和细目账目

一个汇总账目既可以由汇总账目组成，也可以由细目账目组成。这就形成一个以细目账目作为叶子的层次关系（组成模式[1]的例子）。汇总账目的条目是由子账目的条目递归地继承来的。

101

利用该层次结构，可以把多个账目组合成汇总账目。同时强制规定：条目仅允许登记到细目账目中，而不能登记到汇总账目中。汇总账目仍然可以像细目账目一样进行处理，因为它们的条目是从组成它们的子账目的条目继承而来的。一个汇总项目的子账目中如有汇总账目，则它将在它的子账目中或是在子账目的子账目中递归地查找条目。这种条目映射的继承关系，可以允许我们在父类型层次上描述基于条目的结余属性或是其它操作和属性。

例：我有一个有关航空旅行的汇总账目，它由Megabank和Total Telecommunications的航空旅行细目账目组成。

例：芳香咖啡机制造公司有一个爪哇咖啡的汇总账目，它由每个仓库的细目账目组成。因此，通过该汇总账目，可以算出爪哇咖啡的库存总量。

注意：需要做一些标记来确保子账目间的层次关系。通过集合的基数还不足以实现这个约束。在这样的结构中，绝对不允许有循环的存在。

汇总账目和细目账目的区分在账务中很常见，但并不是必须如此的。图6-6中的模型显示如何去除这种区分。其中，任何账目都可以有条目，所有的账目都可以放入层次结构中。这可以通过提供从账目到条目的两个映射来实现：一个映射用来表示一层中都有哪些条目，另一个映射可以把子账目的条目累积起来。第一个是可以更新的，第二个是继承来的，不可以更新，常用于结余、账目清单和图6-5模型中父类的其它特征。

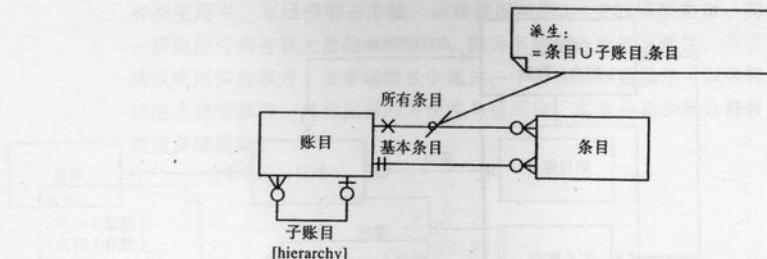


图6-6 不区分汇总账目和细目账目的账目层次

[102]

我们可以使用这个模型把条目记入汇总账目中。

至此，我们已经依照惯例介绍了账目要按层次结构来组织，并且条目要登记到账目中。我们将基于以上假设继续我们后面的话题，不过，在6.15节会介绍其它一些替代方案的可能性。

6.4 备注账目

Benjamin Franklin曾经说过，“在这个世界上，除了死亡和征税，没什么事情是笃定会发生的。”我们无法消除交税的痛苦，但我发现如果能够在见到退税单时不产生惊讶，那么这种痛楚可以得到一定程度的缓解。我是这样做的，每当赚到钱时，我都把其中一部分归入我的税务账目；这样我就知道哪些钱才真正是我的，哪些钱是用来交税的。

请注意，采取这种做法，“真正”属于我的钱不会被拿走。如果不交税的话，我也不必从经常性账户支付款项。此外，这里所讲的税务项目包括州税和联邦税。当真的需要交税（或做预算）时，我就会从自己的经常性账户中支取款项，打入到州税和联邦税的银行账户；同时，需要从我的税务账目中减去相同数量。这样，正式账目（经常性账目、联邦税、州税）和税务账目间其实并没有款项流动。税务账目对我来讲就像是一个备忘录，提醒我该交多少税，因此称之为备注账目。

一个备注账目中的钱并不是“真”钱。事实上，并不会在备注账目中存取真正的款项，这一点很重要。所以，在税款示例中，在我从收入账户取钱放入经常性账目的同时，我也会在税务备注账目上生成条目。备注账目成为账目的另外一种子类型，我还必须确保在备注账目和别的账目间不会产生款项移动。可以这样来实现，即在事务的结算约束中不包含备注账目。

如果使用事务，我们要确保仅仅在账目间移动钱款，并且不能凭空生

成或者销毁款项。这表明，在税务（备注）账目中生成条目时，在某处还要生成结算条目。因为很难看出哪个账目是该结算条目的宿主，所以会计师们通常会使用反账目。这样，税务账目就有了一个额外的反税务账目，它作为税务账目中所有存取条目的反面。该方式可以用于通常的模型，但也不必总是这样。如果结算检查约束忽略备注账目，那么条目单方面的违反约束也是允许的。一个反账目会被自动生成。这种方式表明：从事务到条目的映射的下界可以减少到1。

103

例：每次我从客户那里拿到付款，我把它记为从客户收入账目到我的经常性账目的事务。我还把数量的一部分记入税款债务备注账目。当我交预计税款的时候，我就从我的经常性账目到联邦税款账目上生成一个事务。在这个事务中，我还要生成第三个条目以在我的税款债务账目上减少同样的数目。

当然，如果我们不使用事务，我们就不会遇到结算问题，并可以放心地把条目记入。但是危险在于这么做可能会轻易地把“真”钱流入备注账目，甚至凭空消失。

6.5 记入规则

使用备注账目，我们可以将款项记入到税款债务账目上，但我还是要记着去做。由于我每次都是把收入条目的45%记入税款债务账目，因此计算机系统应该能为我自动做这些事情。

这里需要的就是一个规则：规则检查特定的账目，当发现一个条目时，就生成另一个条目。图6-7表示一个这类规则的简单示例。记入规则通过指定一个账目作为触发器来描述。触发器账目中的任何条目都会生成一个新条目，新条目的值就是原来条目的值和乘数的积。

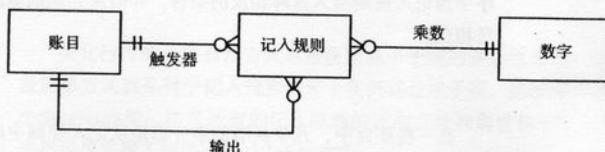


图6-7 有乘积因子的记入规则的简单结构

对于在触发器账目中的每个条目，我们都把它与乘数的乘积结果值作为新条目记入到输出账目中。

例：我的税款债务可以用一个记入规则来处理，收入账目是触发器，税款债务账目是输出账目，乘数是0.45。

104

标量乘积可用于记入规则的很多有用情况，但是过程很容易复杂化。考虑一个递增的收入所得税：最初的300英镑不征税，接下来的2500英镑征收20%税，其余征收40%税。简单的标量乘积并不适用。我们想要记入规则执行任意多样的算法，这会带给我们最大限度的灵活性。

要使记入规则具有这种灵活性，必须把计算连接到每个记入规则的实例，因为规则计算条目数量的方式不同。从概念上讲，这意味着每个记入规则的实例都有自己的计算方式，如图6-8所示。微妙的图符掩饰了一个重要问题。主流的对象系统允许因多态和继承而有不同的方法，但这是基于类的：不同类的对象方法有所不同。我们需要一个类的每个单独实例的方法都可以有所不同，这需要个体实例方法模式，这将在6.6节介绍。（我将在9.2节讨论类似的问题。）

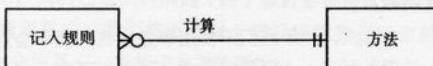


图6-8 带有计算方法的记入规则可以为条目计算值

每个记入规则的实例都有自己的计算方法。

6.5.1 可逆性

记入规则的一个重要属性就是它们必定是可逆的。通常我们不能删除一个错误的条目，因为它要么是支付的一部分，要么出现在账目清单上。我们清除它的作用的唯一方法就是输入一个同样的但是相反的条目。这样，任何记入规则必须确保：两个相同但符号相反的条目放进了同一个触发器账目中，并且在进一步处理中恰好相互抵消。我们可以这样来测试：在程序中为记入规则插入这样相反的条目，并确保它们的输出是大小相等但符号相反。

6.5.2 不使用事务

在一些账目中，几乎所有的事务都是从记入规则中产生出来的。输入账目被用来记录来自外部世界的初始条目。更深层的账目条目都由记入规则产生。由于无论是外部条目还是记入规则，所有条目都是可预测的，因此降低了不使用事务的风险。避免疏漏的责任就由系统的操作使用转变为记入规则的设计。如果我们不使用事务，那么记录删除原因并跟踪条目间的影响是很有用的。总的来说，我乐意用事务，因为这使得审计更为容易而且代价并不高。如果你不用事务，你仍然需要某种审计机制。

105

6.6 个体实例方法

为了领域专家的方便，概念模型应该尽可能自然。我们应该尽可能减少对特定实现环境的依赖。计算机设计应该反映人的思想，而不是人的思想反映计算机设计。这个理念反映在图6-8中。在定义完这个概念模型以后，我们需要找到一个实现它的方式。这样问题就不再是“我们怎样在每个记入规则实例里进行计算”，而是变成“如何在实例上加上方法”。这便引出我们将在第14章讨论的转换方法。我们想得到几种方法来在不改变一个接口的情况下实现图6-9中的模型。这就引出基于模板的设计的重要原理：模型应该定义类的接口。我们在不改变接口的情况下，应该能够更换实现方式。

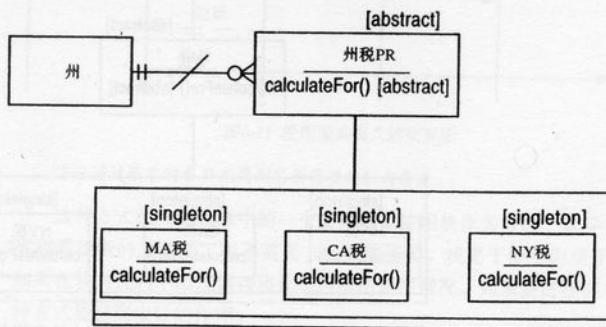


图6-9 使用singleton类实现个体实例方法

6.6.1 使用Singleton类实现

变化行为的一种自然方式就是使用基于子类的多态性操作。这样做的最简单方式就是对于记入规则的每个实例都分成子类，这样就创建了很多个Singleton类。这里所有的记入规则的标准方法和属性都在记入规则中，子类仅仅实现不同的calculateFor方法。

这样做的主要问题是子类有点太人工化。它们的存在仅仅是因为我们不能用实例改变calculateValue。这种人工性使这种方式并不完美。另一个问题是这种方式产生很多的类，这也会使一些人感到很不舒服。因为这些类很小并被加以限制，所以类不会表现出特别大的问题。计算方法的共享可以通过调整类的继承来实现。但是，记入规则的过程操作也可能受害于多态性，而且两个多态性也不一定能匹配。

6.6.2 使用策略模式实现

乍一看，图6-10所示的策略模式[1]实现和使用Singleton类实现很相似。主要的不同点就是，图6-10是在一个独立的方法对象（或称策略对象）上进行子类型化。记入规则变简单了，因为整个方法选择问题不存在了。记入规则仅知道它能用一个方法对象来进行计算。

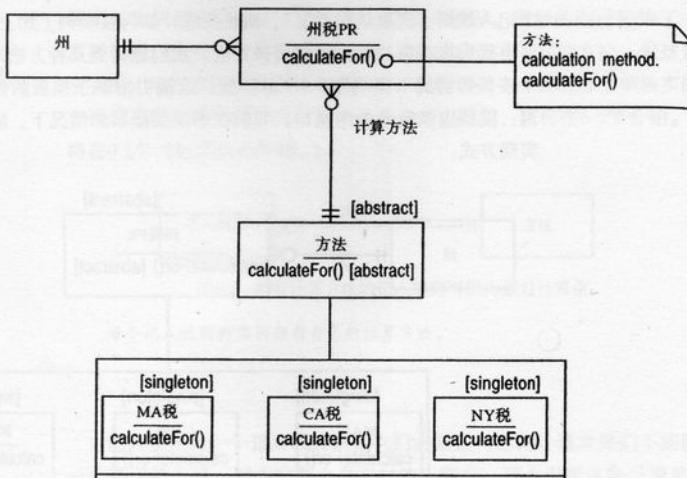


图6-10 使用策略模式〔1〕实现个体实例方法

图6-11显示发生在示例中的相互作用。一个账目请求一个处理规则来处理它。处理规则得到所有没被这个规则处理过的条目（参见6.7.2节）。对每个这样的条目，它都将调用自己的方法来计算新条目的值。方法可能要确定一些问题；例如，税率常常依照结婚与否而有浮动。把结果传回记入规则，记入规则就会构造新的条目。

应该强调的是：从函数设计（或从面向对象方法）的角度说，方法对象不是一个“自由子程序”。方法被封装在记入规则中，只有记入规则能够引用这个方法。

记入规则的方法可以在两个对象间共享。这个方法的一个实例就是统一税率方法，统一税率方法应用某种标准折扣税率。如果几个类有同样的方法、仅仅是税率不同的话，可以设计一个方法，它向记入规则询问统一税率，否则的话，就允许过程重用。这可以看成是方法对象和参数化方法（参见6.6.4节）实现的折衷。

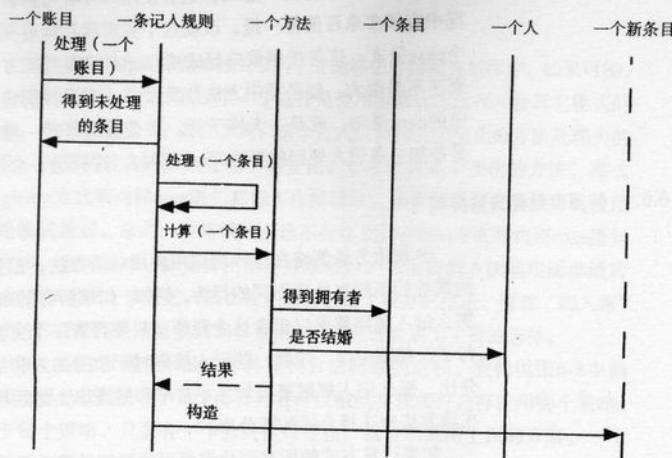


图6-11 使用策略模式的交互图

方法通过请求的条目来得到它所需要的所有信息。

这种方式在Smalltalk中的一个变体就是使用块作为方法。这样，我们不再需要新的方法类，也不需要方法类的子类。块易于使用但难于调试：如果在块内代码中出现错误将很难跟踪出现的情况。如果块还算简单，这种方式能起到很好的作用。

6.6.3 使用内部case语句实现

面对生成子类仅仅是为了处理一个多态方法这样一个情况，我们可能会疑惑，我们为什么应该如此大费周张。我们可以为记入规则定义一些私有操作：computeFederalTax、computeMassTax、computeSalesCommission等。如图6-12所示，记入规则中只要一个包含简单case语句的computeFor函数，依照接受实例来决定使用哪个私有方法。108

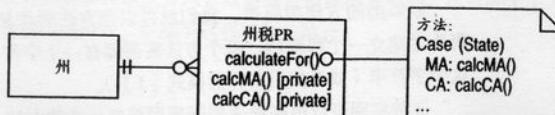


图6-12 使用内部case语句实现个体实例模型

只要把case语句封装在记入规则中，就不算是违反面向对象的原理。

对象的设计者们一想到像这样使用case语句就被吓跑了，但在这种情况下有很多东西值得一提。改动这个实现就意味着要加一个私有操作和一个case分支。这与使用策略模式或singleton类实现方式的新子类相比，区别还不是很大。如果类中方法数量太多，那么我们将有一个很大但还算简单的case语句，或是一大堆子类。要么就是管理一堆singleton类，要么就是每加一条记入规则就要改写case语句，这只是一个权衡问题。

6.6.4 使用参数化方法实现

109 参数化方法策略在记入规则中使用一个方法，根据记入规则或相关类的属性的不同来处理不同的行为。例如，如果所有的条目都是统一税率的，那么记入规则就可以包含这个税率，只要再有一个方法按这个税率扣除就行了，如图6-13。如果一些记入规则对于单身的人和结婚的人有不同的百分比，那么记入规则里再加入单身率和结婚率，然后可以询问婚姻状况并在该方法中选择合适的百分率。

如果计算方式的所有变化都可以通过简单修改几个参数来实现，则这个策略很起作用。这种情况下，我们就这么建模。如果情况更加复杂，就需要个体实例方法。这是一个很有潜力的实现手段，因为在有些情况下，我们可以把参数化方法和其它技术结合使用。

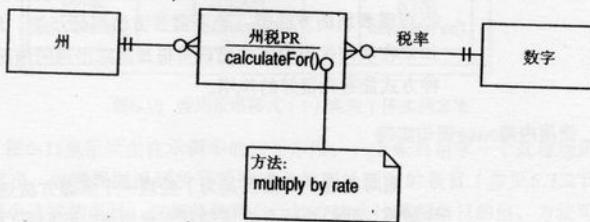


图6-13 使用参数化的记入规则

6.6.5 使用解释器实现

如果类的方法很简单，我们就可以把方法看作简单语言中的字符串，再为之建立一个解释器。每个方法实例都有一个字符串，方法类能够解释这个字符串（或者使用解释器模式[1]）。

这种实现可以很好地适用于采用简单公式的方法，这些方法使用数学运算符、括号和几个简单过程。如果语言简单，建立解释器并不难。唯一的限制就是语义能表述些什么。

6.6.6 实现方式的选择

所有的实现方式都能够很好工作，并能够很好隐藏在操作中。如果可能，我会使用参数化方法。我的下一个选择是使用参数化方法和一种其它模式的结合，看能否发现一个混合方式，这个方式仅使用几个变化的方法处理大的变化，并使用较多的参数处理小的变化。如果仅有几个变化的方法，那么 singleton方式和内部case语句都能工作得很好。如果有很大变化，那么使用策略模式最好。总之，策略模式永远不会比用 singleton方式和内部case语句差太多，但可能乍一看觉得稍稍有点难理解。如果类的方法能用简单语言（比如数学公式）来表述，那么使用解释器是个很好的想法。随着“四人帮”模式的广泛使用，策略模式和参数化方法的结合将成为主要的选择。

以上四种策略显示处理个体实例方法问题的途径。我们说图6-8中画的模型是特定的分析声明，而设计者可以根据实现条件选择使用哪个策略。对于每个策略，只要有一个公共接口存在，就可以采用上面的方案。一个分析模型定义一个可以有多种方式实现的接口，这个原理是使用设计模板的开发方式的基础。

110

很多建模者乐意使用问题建模的别的方法，而不使用图6-8。他们可能乐意使用接近于其它策略的表述。如果在同一个接口下把实现替换掉，他们仍可以把分析和实现分开。其他的建模者使用与实现相同的模式。这种情况下，他们牺牲实现的独立性以换取分析模型和实现的更好吻合。

对分析和设计进行精确划分是很困难的。正如各种类的组合就可以满足特定的软件接口一样，我们对同一情况进行概念建模时也可以使用多种类型的结合。类型的选择会影响到类的选择。主要影响是，类型的选择就定义类的接口，但是接口下面的实现不必与概念模型一致。

6.7 记入规则的执行

至此，我们已经讨论如何构造记入规则和记入规则在被触发或告知执行后如何反应。我们在这里可以回顾一些记入规则的触发策略。记入规则应该被设计为有多种触发方式，这是我要强调的第一点。尽可能把记入规则和其触发策略分开，以减少这些机制的结合。

6.7.1 急切触发

在这种方式中，触发账目中一旦生成合适的条目就立即触发记入规则。我们可以有两种方式这样做。第一种方式是：将职责放入事务或条目的创建方法中，如图6-14所示。创建事务可以导致几个条目被记入账目。条目

的每次记入都会搜索记入规则，记入规则使用账目作为触发器。这时每条记入规则都被触发。

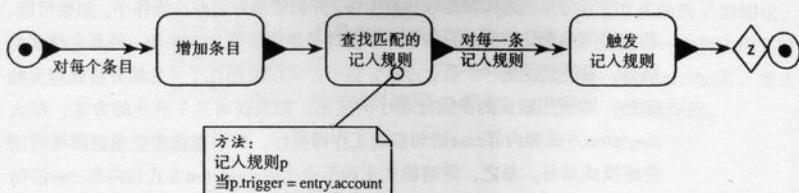


图6-14 显示构造事务能触发记入规则的事件图

记入规则的查找和触发可以在事务构造时也可以在单个条目构造时进行，如图6-15。后者更好一些。

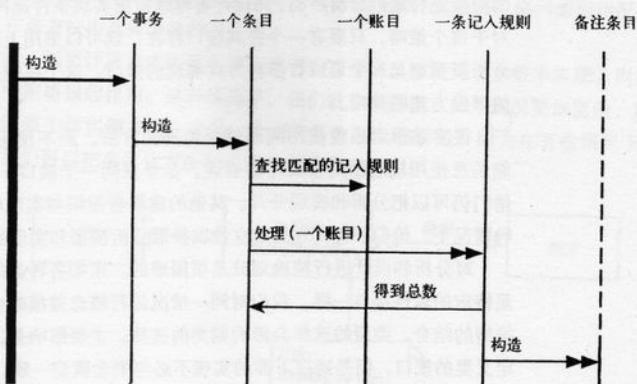


图6-15 在条目构造中触发记入规则的交互图

第二种方法是：使记入规则成为触发器账目的观察者[1]。当生成记入规则时，记入规则在触发器账目中登记。当一个条目被放入账目中时，账目对所有的观察者广播这个事件。记入规则便检查账目，发现新的条目。然后就在备注账目中生成合适的新条目。这个方案的优点是事务不必再激活记入规则。观察者是一个很有用的机制，但只有当能从观察者看到被观察者的时候我才使用这个机制，尤其是观察者与被观察者处于不同的包中。在不是必要的时候，我并不乐意使用观察者，因为许多这样的程序难于调试。我也不想把记入规则放入独立的包中，因此也就不需要使用观察者。

6.7.2 基于账目的触发

基于账目的触发把触发的职责从事务移到了账目。条目可以被添加到账目上，而不触发任何记入规则。在有些时候，如图6-16，账目被告知处理自己并接着触发其上次处理自身以来新加入的所有条目的向外的记入规则。

112

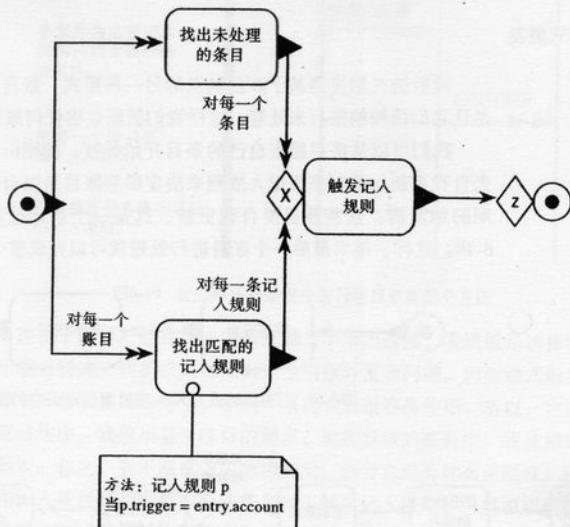


图6-16 账目的循环触发

图中X号表示对于每条记入规则和未处理条目的组合执行触发记入规则的操作。

基于账目的触发需要账目跟踪未处理的条目。它可以通过维护一个独立的未处理条目集合（保存条目列表并跟踪最后一个要处理的条目）来达到目的，或者通过记录上次处理的时间点以便能返回这个时间点之后登记的条目（使用登记时间属性）来达到目的。

基于账目的触发可被用于循环的账务系统，这种账务系统中，账每天处理一次。这种情况要注意的是账目要以正确的顺序处理。在可能受到向外的处理规则影响之前，账目必须被处理。通过查看处理规则可以自动地决定这些依赖关系。

6.7.3 基于记入规则的触发

在基于记入规则的触发中，利用外部代理明确告知记入规则开始执行。
113 这个代理将查看输入并找出新条目。这样，基于记入规则的触发和基于账目的触发很相似，有很多一样的优点和缺点。主要的不同是：既然账目可以有很多记入规则，那决定哪些条目未被处理的责任就由账目转到了条目。这通常会使得问题更复杂，所以我愿意使用基于账目的触发。

6.7.4 向后链式触发

向后链式触发是基于账目触发的另一种形式：账目自己并不处理，而是让它们依赖的账目来处理。这样我们就能查出任何账目的最新状态。

我们可以从账目检查自己的条目开始处理，如图6-17所示。这个账目先自我更新，然后使用记入规则来决定哪些账目是以自己为输出的记入规则的触发器。这些账目要自我更新，这是一个迭代过程，如图6-18和图6-19。这样，请求最后一个账目进行处理就可以完成整个账目图的更新。

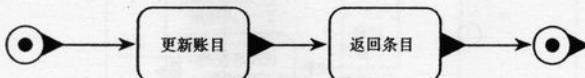


图6-17 用向后链式触发向细目账目请求它的条目

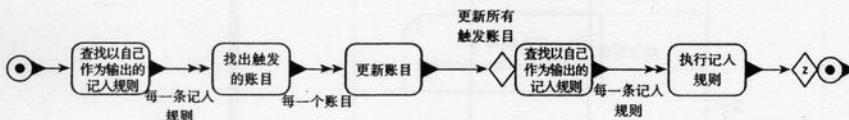


图6-18 使账目更新的方法

对于处理账目中的每一个输入账目都要递归地调用账目更新操作。

6.7.5 触发手段的比较

选择触发方法的核心因素是何时要执行记入规则（实现决策），以及何时要捕获错误。急切触发使我们可以尽早发现错误。这就意味我们可以有更多的时间查出出错的原因。这也要求我们在生成条目的时候要做全部计算。在何时进行计算上，基于账目的触发和向后链式触发给我们更多的灵活性。如果我们成批处理账目，可以把条目从文件中全部读入，然后在休闲时间（比如夜晚）触发记入规则。触发得越早，就越早发现错误。

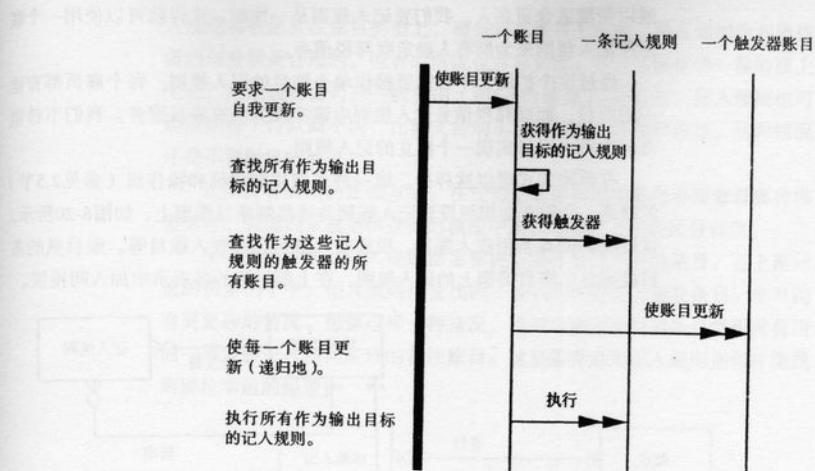


图6-19 基于账目的触发中进行账目更新的交互图

在基于账目的触发和向后链式触发中做出选择，实质就是选择我们是否乐意处理建立向后链式触发时产生的额外复杂问题。向后链式触发比基于账目的触发更难于建立，但是一旦建立就很容易使用。所以，在简单的账目结构中，我使用基于账目的触发；而在复杂的账目中，我使用向后链式触发。总之，我不愿意使用急切触发，因为它没有什么灵活性。通过确保在加入条目时（不是在生成条目时）触发记入规则，一样可以得到急切触发带来的所有好处。尽管这是额外的，但的确让我在不希望这样做的时候可以不这样做。急切触发没有提供这样的选择。但如果我能有足够的处理能力，而让记入规则不会引入任何代价，这就没什么区别了。

你可以混和使用触发方式。收入账目在进入两层财产账目中可能使用急切触发，而在其余部分可能使用向后链式触发。但是，使用多种触发方案将会使系统更加复杂，所以我一般不会无缘无故地混和使用这些触发方式。

这类方式还算是新的，而且人们还在探索各种触发方案中固有的折衷方式。这是一个多变化的领域，重要的是要保留一点灵活性，以便在观察系统运行时改变触发方案。

6.8 多个账目的记入规则

至此，我介绍了自己的示例和我自己的账目表。我们需要进行一些扩

展以便能适合更多人。我们要记入规则是一致的，这样就可以使用一个联邦税记入规则来为所有人确定联邦税债务。

经过这个扩展，不再需要操作单个账目的记入规则。每个雇员都有唯一的账目，而联邦税债务记入规则应该实现为所有雇员服务。我们不希望必须为每个雇员实现一个独立的记入规则。

有两种方式可以这样做。第一种是使用知识级和操作级（参见2.5节）的概念。我们在知识级设置记入规则并连接到账目类型上，如图6-20所示。这样我们就有费用收入账目、税前收入账目、净收入账目等。账目里的条目检查自己账目类型上的记入规则，在上面讨论的各表示中加入间接级。

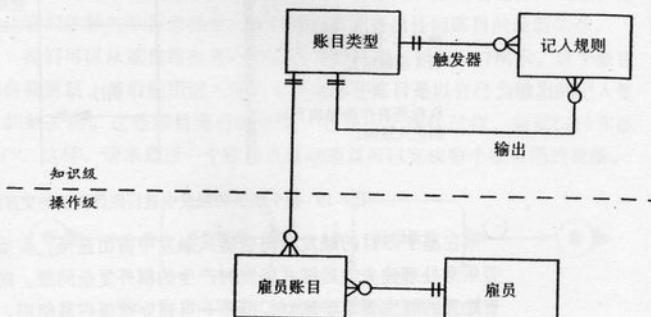


图6-20 使用账目类型

这里介绍可以在其上定义记入规则的知识级。

例：所有的雇员每工作18天就增加一天假期。这可以表示为一条记入规则，它的触发器是账目类型“工作天数”，它的输出是账目类型“附加的假期”。这个方法保证假期的账目结算是工作账目结算的 $1/18$ ，雇员账目每次被触发的时候，它就依照触发类型，查找定义在账目类型中的记入规则。

知识级和操作级的划分，虽然很吸引人，但不是处理这种状况的惟一办法。另一种方式就是使用汇总账目。在条目加入汇总账目的子账目（或是汇总账目本身，如果汇总账目记入是允许的）时，定义在汇总账目里的记入规则被触发。输出账目可以类似地被定义在汇总账目中，这可能导致在合适的子账目中生成条目。

例：这个示例包含工作天数汇总账目和附加的假期账目。记入规则和上例一样。要检查的是汇总账目，而不是为记入规则检查账目类型。

这两个方式的选择取决于账目和账目类型的区分程度。如果所有的记

入规则都被定义在账目类型上，而条目在账目上产生，那么知识级和操作级的划分就是合理的。但有时情况也并非如此。条目可能在更一般的级别上生成，用来表明公司的一般费用（图6-6的模型）。类似地，记入规则也可能依照每个付款而不同：比如支持购车贷款的减少就需要这样。这种情况还是不要做区分的好。

这里没有通用的正确方案。在任何给定的情况下都有必要看看哪种模型更好。关键因素就是供选择的模型中账目和账目类型的区分程度。

无论哪种方式，记入规则都需要决定怎样输出正确的条目。在上面讨论的很多例子中，记入规则仅查找同一雇员的账目作为触发条目。也可能有更复杂的情况。想像这样一种情况，给初级顾问的费用条目将引发费用的一部分被记入顾问经理的备注账目。这里需要通知记入规则如何才能找到那位幸运的经理。

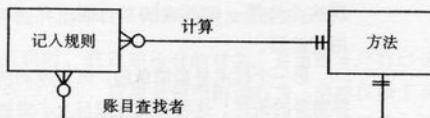


图6-21 使用账目查找方法

使用分开的方法来查找输出账目和计算事务的值。

一种处理方式就是提供第二个方法来查找合适的输出账目，如图6-21所示。这个方法向源条目请求其雇员，然后再通过雇员找到经理。这就提供了最大程度的灵活性，但需要第二个方法对象，而且要像6.6节介绍的那样实现这个方法对象。

117

这就暗示另一个问题。在通常的记入规则中，并非所有的雇员都有资格触发记入规则。例如，记入规则可被设置为处理州税。那么伊利诺伊州税收的记入规则就只有在雇员是伊利诺伊州的居民时才能被触发。这就引出第三个方法，它用于表述资格条件，如图6-22和图6-23所示。

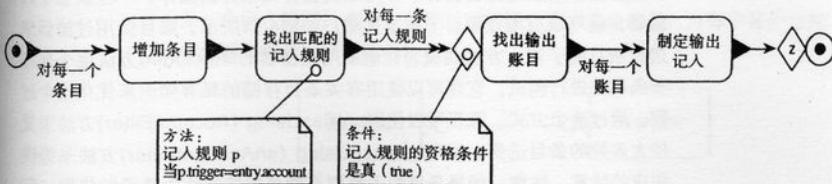


图6-22 在图6-14中加入账目查找者和资格条件的事件图

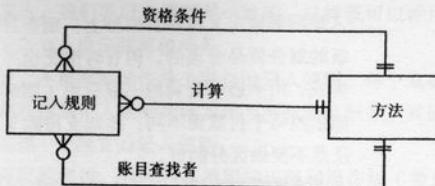


图6-23 在上面的规则中加入资格条件

6.9 选择条目

在很多情况下，记入规则需要在触发器账目的条目中选择一个子集。比如，可能要查看从某个归档日期后的所有条目（如7月所有条目的结算），或是危险货品的条目（用条目的一个子类型表示）。有三种方式实现这类选择：把所有的条目取出并选择，提供针对具体选择的方法，使用过滤器。

第一个技术是最简单的：账目返回所有的条目，客户处理这个集合以选择需要的条目。这种方式在账目上不需要额外的行为，而是把所有的事情交给客户。如果有很多客户都需要进行类似的选择，就会有很多重复。如果存在很多个条目，把集合输出也是很大的开销，尤其在需要复制集合的时候。记住，账目永远也不要把自己存储条目的方式未加保护的输出（参见14.1节）。使用这种带条目的方式也意味着由客户负责条目的统计和结算。

如果很多客户都在请求类似的集合，比如某段时间的条目，那么可以在账目上加上一个额外的行为来满足这个需要（比如entriesChargedDuring(TimePeriod)）。这样有个好处就是客户省去了重复遍历同一个集合的麻烦。如果加入一个处理某段时间上的结算的方法（比如balanceChargedDuring(TimePeriod)），那么客户就可以更省力。这种解决方案带来的问题是：如果有许多集合，那么账目的接口会变得很大。

过滤器（参见9.2节）是封装了查询的对象。使用这个模式将导致账目过滤器。账目过滤器包含多种用来设置查询条件的操作。一旦设置了过滤器，就可以应用在账目上实施过滤，如图6-24所示。账目使用过滤器来选择条目的子集，方法是概念性地采用过滤器的isIncluded方法逐个地检查条目并进行测试。它还可以使用有关条目存储的私有知识来优化这个过程。通过这种方式，账目可以使用entriesUsing(AccountFilter)方法来支持大多数的条目选择，使用balanceUsing(anAccountFilter)方法来提供相应的结算。注意：如果条目的子类型有额外的特性作为选择的依据，那么每个条目类型就需要过滤器的子类型。

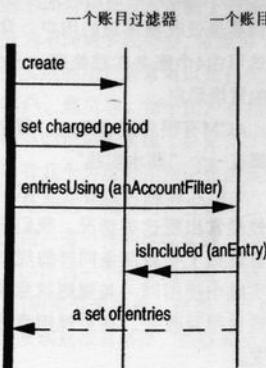


图6-24 使用账目过滤器的交互图

使用多值关联时，我返回所有的对象，并留给客户自己选择。如果有几个常用的选择方式，我愿意使用附加行为，但也仅限于有少数几个行为。如果某个选择方式导致太多的重复以返回所有对象，还要附加太多的行为的话，我就使用过滤器。设置和维护过滤器的确需要额外的工作，所以我只在真正需要的时候使用过滤器。这种需要经常在账目和条目中出现。

6.10 账务实践

在我们遇到有很多记入规则的较大的账目网络时，网络可能很大并难于应付。在这种情况下，我们需要用某种方式把网络分成一个个的部分。考虑一个使用账目清单的例子。我们对各类客户使用不同的记账程序。这可以表示为一个账目网络。每种客户类型有不同的规则，可以使用稍微不同的账目网络来处理。

119

一种特殊的账目网络是一个账务实践。从概念上讲，一个账务实践只是一个记入规则的集合，如图6-25所示。基本思想就是，为每个客户分配一个账务实践来处理账目清单。

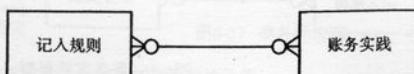


图6-25 账务实践

这些账务实践被用于对记入规则进行逻辑分组。

例：电力部门把居民用户分为常规类和生命线类。生命线类指的是该州认为需要最低收费等级的用户。常规用户依照居住区域不同被分为3个等级。这可由4个账务实践处理：一个给生命线用户，其余3个分别给那个等级的常规用户。

例：ACM有很多工会工人，每个工会都协商不同的待遇。ACM对每个工会都有一个“薪水实践”。

同一个记入规则可以存在于不只一个实践中。在实践间需要类似行为的时候经常出现这类情况。我们需要区分以下这两种情况：从一个实践中拷贝规则（导致两条同样的规则），在多个实践中使用同一条规则。在多个实践中使用同一条规则就意味着：当规则改变的时候，所有使用它的实践也跟着改变。拷贝规则意味着一个拷贝的改变不会导致其它拷贝的改变。

账务实践被分配给用户对象，这样每个用户都有一个账务实践。因此，每个电力部门的用户或是公司雇员都有一个特定的账务实践。这种分配可以手工完成，或是由规则来决定。

例：在ACM，薪水实践依照工人所属工会的不同分配给工人。

如果不使用账务实践，你还可以使用能够依照雇员某个属性把条目分类的记入规则。如果不为每个工会都分配实践，你也可以仅使用一个实践。第一条记入规则查看条目所在工会，并在相应的工会账目里生成条目（参见7.6节这类分离记入规则的示例）。

如果问题复杂，我会使用分离的实践，只要我们能在某段时间内把实践分配给用户。任何在条目到条目的基础上进行改变的划分（比如在7.6节里面夜晚/白天的划分），都必须有记入规则来处理。如果一个用户改变了账务实践，我们可以使用历史映射来跟踪这些改变（参见15.3节）。

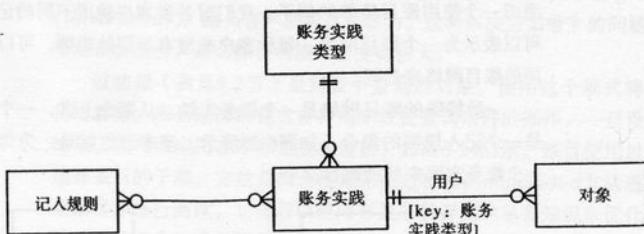


图6-26 账务实践类型

在比较大的账目网络里，我们定义一个账务实践的配置文件，配置依照每个使用它的对象而不同。

当处理过程的不同阶段具有在逻辑上分离的多组记入规则的时候，我们可以把规则划分为不同的实践类型，依照每个类型给用户分配一个实践。在图6-26中，账务实践的用户通常可以是任何对象。当然，在特定的模型中，用户可以是客户、雇员等。每个用户有相应的一类账务实践，这个约束条件可以用带键值的映射来实现（参见15.2节）。

例：一个应用有几个为居民用户记费的实践，但所有的居民用户都按同样的方式交税。处理这个情况时，我们可以使用分开的收费实践和税收实践。所有的居民用户有一样的税收实践，即便他们有不同的收费实践。

最后，我们可以得到一个合理的结论，就是把账务实践和记入规则作为同一组成模式[1]的部分来处理。这也允许有很多级别的实践组合。到目前为止，我也没发现这很有必要，所以我也没进一步研究。

6.11 条目来源

有时，知道为什么一个条目以某个形态存在是很重要的。例如：如果一个用户查询一个特定的条目，当前的模型能给我们很多关于这个条目生成的信息。我们通过查看其它条目的日期就可以决定当时账目所处的状态。我们还能确定哪条记入规则计算了该条目。

图6-27所示的模型可以处理这样的客户需求，它可以得到每个事务以便记住是由哪条记录产生的这个事务，还能记住哪些条目作为事务的输入（如果你没有使用事务，那么关联就是条目到条目的）。

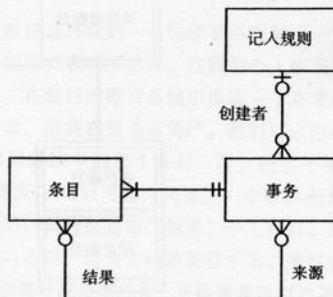


图6-27 事务的来源

这里记录了每个条目双方向的计算。

例：我给ACM做些工作并得到2000美元的报酬，我把它记为一个从费用收入账目到经常性账户的事务。我的记入规则在我的税款债务账目里

122

产生一个单独的事务。这个事务的创建者是45%的记入规则，这个事务的来源包括从费用收入账目里取款。

使用这个模式，我们可以在账目结构中形成条目和事务链。每个条目可以递归地使用来源和结果间的映射来确认原因和效果。

建模原则：为了确认计算是如何进行的，把计算的结果表示成对象，由它来记住是由哪个计算产生的结果以及计算所使用的输入值是什么。

6.12 结算单和所得计算书

在使用账目来描述一个系统时，区分结算单和所得计算书账目是很有用的，如图6-28所示。我的经常性账目是资产账目，我的信用卡账目是债务账目。这样，它们反映了某段时间内我有多少钱（或在信用卡里表示我没有）。这些都在我的结算单上。收入和花费账目反映了我的钱从哪里来以及花到哪里去。我有一个收入账目记录老板给我的工资，另一个收入账目记录存款利息，一个花费账目记录旅游花费，还有为购买食品所设立的花费账目等。我的收入与花费的结算账目并不能反映我目前有多少钱，仅仅反应我的收入的来源与花费。



图6-28 资产、收入、花费账目

这些是在财务账目中常见的账目。即便在别处，这些概念也是有用的，用来区分它们持有的东西和它们来源去处的分类。

123

账目通常用在这样的模式里，在收入账目里产生各个条目，条目被传

递到几个资产账目，然后消失在花费账目里。系统保存的资金被存放在特定的资产账目里，但很多资产账目仅仅用于定期结算。债务账目几乎总是被用来在今后某个时间进行结算（它也可能在今后用于长期债务，比如抵押）。

例：我用信用卡从波士顿航空公司订购了一张机票。我的信用卡账目就是一个债务账目，波士顿航空公司账目是一个花费账目。这是我来划分的，我是信用卡账目的拥有者（这是我的债务账目）。

例：ACM从印度尼西亚咖啡进口商那里购买了3吨爪哇咖啡。ACM有一个印度尼西亚咖啡进口商的收入账目，记录从印度尼西亚咖啡进口商转移3吨爪哇咖啡到ACM的纽约账目。纽约账目是一个资产账目，由ACM拥有。

在此，我可以简要解释一下我为什么不用术语“借方”和“贷方”。这些是众所周知的账目术语，但是我没用它们，而是使用“从哪”、“到哪”、“取款”和“存款”。原因是术语“借方”和“贷方”不能表达同“取款”和“存款”一致的含义。对于所得计算书账目，贷方增加账目，而借方减少账目，这样对于外行是容易理解的。对于结算单账目，借方增加资产（就是他们存入资产），而贷方减少资产。对于没有会计知识的人来说可能有些奇怪，但这的确是通常的账目习惯。所以我不用术语“借方”和“贷方”，部分是因为它们可能会把没有会计知识的人搞糊涂，部分是因为我们在讨论一个更抽象的模型，而非一般的金融财务问题。

6.13 对应账目

收入和花费账目是外在的——钱还不是我的——只是我用来分类的账目。银行对账目的观点表明了这点。在我的个人账目系统中，我有一个资产的经常性账目。在银行的账目系统中也有一个非常类似的账目。银行是银行账目的归类者，但我在里面有资产。我们可以把银行系统中的这个账目想成和我的账目系统中的那个账目一样，但这不能起作用。在3月1日（这是我取款的那天）ATM可能为我登记一个取款条目。银行在3月2日进行同样的取款登记，因为这是银行的第二个工作日。这两个账目指向同一个资产，但是它们不同，因为它们的条目不同。最好试着把它们看作对应账目。对应账目在某种程度上要匹配，并且通常可以在某个时间点相互消解。比如在我的账本（即我的账目）和银行的账本（即银行的账目）进行匹配的时候。这个消解过程要精确，但也可能留有不严密之处，比如日期上可能稍有不同。

图6-29表示这种情况。只有结算账目才有拥有者；所得计算书账目没有资产，所以就没有拥有者的问题。所有的账目都有分类器来指示谁生成

和操作了账目；我曾使用过团体（参见2.1节）。对应关系表示一对特殊的属性：对称性和传递性[3]。首先是对称性，如果账目 x 是账目 y 的对应者，那么账目 y 一定是账目 x 的对应者。通常默认的关联是不对称的。传递性表明：如果账目 y 是账目 x 的一个对应者，而账目 z 又是账目 y 的一个对应者，那么账目 z 是账目 x 的一个对应者。

对应者[对称的，传递的]

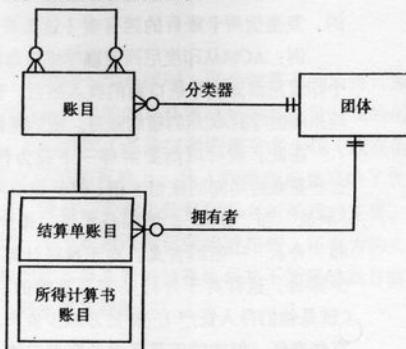


图6-29 对应账目

6.14 专门化的账目模型

我已经给出多个示例来表明：这个模型既可以被用作财务账目的基础，也可以被用作库存跟踪的基础。使用账目模型，通常通过子类型化来提供特定域的信息。例如，考虑库存管理——一个很适用账目的问题。我们对每个货物和地点的组合都形成一个账目（给它一个非财务名称，比如财产）。这样，如果我们在伦敦、巴黎和阿姆斯特丹之间跟踪 Macallans、Talisker 和 Laphroig 威士忌的销售情况，我们就需要 9 个财产（资产账目，比如伦敦-Macallans、伦敦-Talisker、巴黎-Talisker 等）。无论什么时候，我们把货物从一个位置移动到另一个位置，我们就创建一个传输（事务）来处理这个移动。因为涉及钱的问题，所以事务要进行结算。另外，这类对象必须在整个过程中保持不变。图 6-30 表示这个领域对账目模型的扩展。

125

用这种方式跟踪订单（包括收入和支出）也很有作用。每个供应商有一个收入账目，如果供应商的位置重要的话，可能还不只一个。类似地，每个客户有一个支出账目。我们可以在两个方向上跟踪订单：我们允许传

[126]

输出的子类型，或是预定的或是实际的，或者我们为订单提供另一个财产的集合，这样我们会有如伦敦-Talisker-Ordered和伦敦-Talisker-Actual这样的财产。当生成订单的时候，我们生成一个从供应商的预定财产到交付目的地的预定财产的传输。订单被提交后，我们就在我们的地点上生成从预定财产到实际财产的传输。这和金融书上使用的接收账户是一样的。

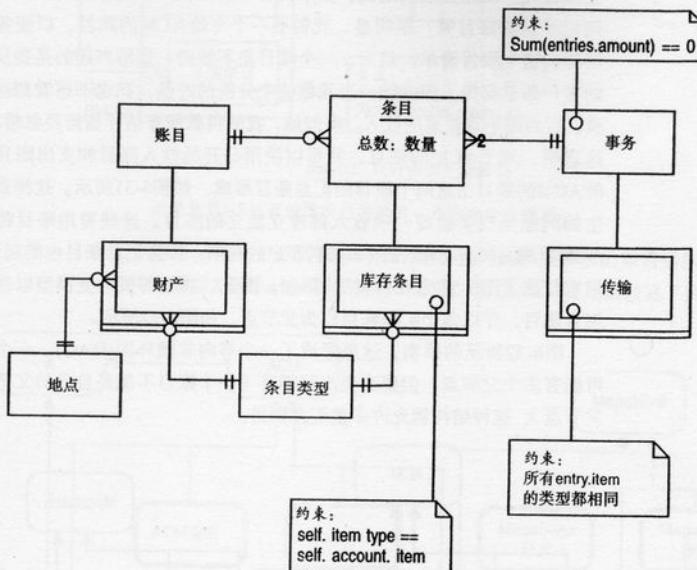


图6-30 专门化账目模型以支持库存管理

在特定领域使用账目模型需要进行这类的专门化。

我们可以使用汇总财产来得到全面的情况。所有预定财产的一个汇总财产能给出所有预定位置，一个Talisker的汇总财产能给出所有地点的Talisker的总数。

6.15 登记条目到多个账目

处理账目时一个常见问题就是：有时有多个地方可以登记条目。例如，假定我支付500美元购买机票参加OOPSLA的招待会。我是把它记在OOPSLA账目上呢（以便我能算出我花了多少钱参加OOPSLA）还是把它记在旅行账目上呢（以便我能算出我花了多少钱在航空旅行上）？有几种

方式可以处理这种情况。这几种方式体现一些关于使用账目的有用办法，也引出比前面介绍的账目结构更加复杂的账目结构。

一个顾问的账目清单表明这个问题。比如说，我在ACM做了三天的顾问，我收费6000美元。另外，我还需要额外的花费：500美元的飞机票，250美元的住宿费，150美元的汽车租赁费，100美元的餐费。我怎么算这些账目呢，或是更准确地说，如果我有一个像样的账目系统的话，我该如何记录这些账目呢？很明显，我需要一个专给ACM的账目，以便我能寄给他们这个账目清单。然而，一个账目是不够的。我感兴趣的是我从不同的客户那里都挣了多少钱。当我做这个分析的时候，我也不想看到这些花费，因为那不算是真的收入。类似地，我的税款债务估计也需要忽略花费。这表明，对于ACM的账目，我可以使用分开的收入账目和支出账目。我的ACM的账目由这两个账目的汇总账目形成，如图6-31所示。这样做所产生的问题是我要给每一项收入都建立独立的账目。这些费用账目将包含ACM报酬、Megabank报酬和其他客户的报酬。作为汇总账目也能起作用，但它打破了图6-5和图6-6的结构限制。因此，我就需要改变模型以便使用细目账目，并以多个汇总账目作为父节点，如图6-32所示。

图6-32所示的模型，这里形成了一个有向非循环图(DAG)。一个账目可能有多个父节点，但要避免出现循环（一个账目不能是自己的父节点的父节点）。这种结构就允许多重汇总账目。

127

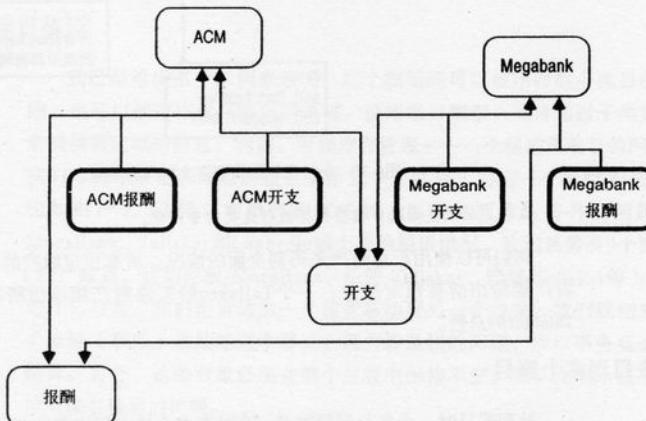


图6-31 一个典型的报酬/开支账目结构

粗边图表示细目账目，由箭头表示汇总。

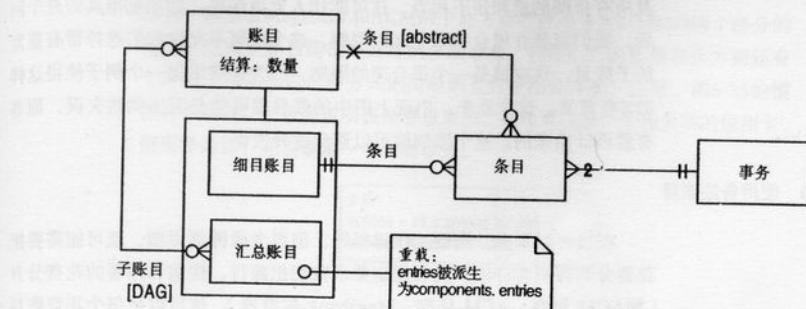


图6-32 允许多重汇总账目

这里把图6-5里面的层次结构换成一个有向非循环图。

然而，有一个必须考虑的小问题。我用图6-33所示的账目结构会怎么样呢？账目X合计ACM以及所有的报酬，所以ACM费用就被算了两次。

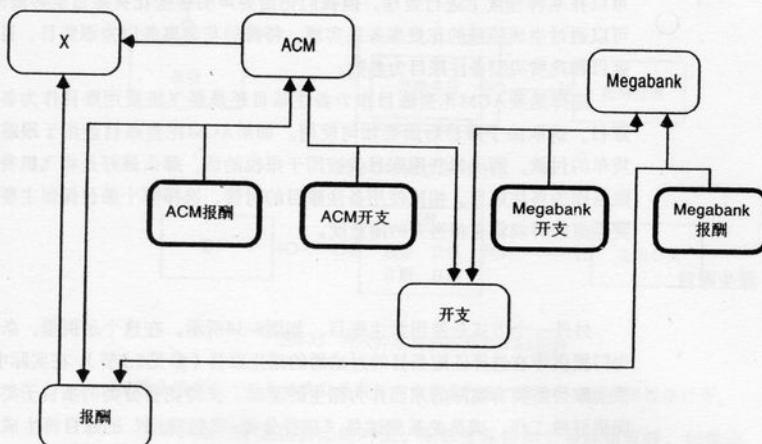


图6-33 显示一个问题的账目结构

如果使用多重汇总账目，有可能会定义一个具有重叠细目账目的汇总账目。

依照图6-32的模型，我们仍然能得到X的正确结算。结算是在派生的条目集合里定义的。集合元素不允许重复，所以在ACM报酬里的条目都只能在X中出现一次，这样就能给出正确的结算。然而，这个结算和ACM

及所有报酬的总和还不相等，这可能让人觉得诧异。如果觉得真的是个问题，我们就要在组合关系上加以限制，这个限制不允许我们选择带有重复的子账目。这应该是一个很合理的限制，因为很难举出一个例子使得这样的X有意义。这样看来，出现上图中的账目很可能是因为偶然失误，而非有意设计出来的。这个限制就可以避免这种失误。

6.15.1 使用备注账目

在这一级别上，模型工作得很好，但要考虑得更详细，就可能需要把花费分解得更为详细。税收规定要求我们把旅行、住宿和用餐的花费分开（如ACM-航空、ACM-住宿、Megabank-航空等）。这可以把每个花费账目分解为多个细目账目，但这可能会使花费账目因为这些复杂的账目组合而变得难于管理。有必要探索其它的可选方案。

129 一种选择是使用备注账目里的条目。这样访问ACM总部的500美元的飞机票就既要出现在ACM花费账目上，也要出现在飞机费用账目上。这种方式消除了对ACM-航空账目的需要，但是需要额外的条目。记入规则可以在某种程度上进行处理，但我们仍需要声明哪些花费账目是必需的。可以通过生成特殊的花费事务来实现，特殊的花费事务以来源账目、目标账目和花费类型备注账目为参数。

选择是要ACM花费账目作为备注账目还是要飞机费用账目作为备注账目，这取决于账目后面要如何使用。如果ACM花费账目被用于跟踪发货单的付款，而飞机费用账目仅被用于报税的话，那么最好是将飞机费用账目作为备注账目。相比使用备注账目的时候，选择哪个账目保留主要的资金流的时候就有相当大的随意度。

6.15.2 派生账目

130 另外一个方式是使用派生账目，如图6-34所示。在这个示例里，条目专门提供带有选择匹配条目的过滤器的派生账目（参见6.9节）。在实际中，派生账目需要有实际的东西作为派生的基础。支持花费分类的条目子类型能很好地工作，成员关系测试是“花费分类=飞行费用”的账目将生成需要的信息。

我们可能考虑更深一层地使用这个方式。为什么不完全放弃账目而使用像图6-35的样子呢？我可以通过查询花费搞清楚一切。

这个问题帮助定义为什么账目是有用的以及为什么派生账目是有价值的。账目在需要跟踪资产流向的相对静态的结构里面能很好地起作用。如果这个流向简单，比如分配机票花费，那么还不至于需要账目。然而，考

虑我一趟就去Megabank和ACM两个地方而把费用分为2/3和1/3两个部分的情况。这就是账目能很好处理的多腿事务。但图6-35的模型在这方面还有个问题。我如何以这种方式把简单的支付费用分开呢？注意，图6-35的模型有个问题：它没有说明钱是哪里来的。我还可以加一个相关联的信用卡，但那样的话，花费就像一个双腿事务。

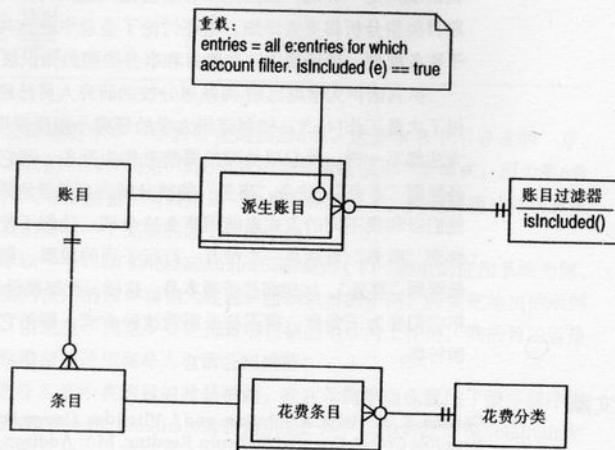


图6-34 派生账目介绍

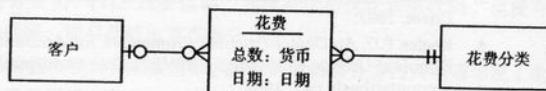


图6-35 为放弃账目模型而定义的花费

派生账目仍可以允许我们使用所有的报表行为，但我们不能使用跟踪行为。

在账目结构不是很静态的时候，给派生账目加上属性很有效。如果有很多信息片，那么使用账目所拥有的同一个报表功能，可以很容易地计算派生账目。然而，派生账目只有报表功能，不能在其中进行登记，所以不能用于跟踪资产的消长。

所以，无论何时我们要表现条目的某一方面，我们就要在条目属性和新一级账目之间进行选择。这个选择是基于你要什么样的账目行为。如果我们仅仅需要报表，我们可以使用属性，必要时也可以使用派生账目。否

[131]

则，我们就需要新一级的账目。

进一步阅读

这里我可以推荐两个和账目相关的信息资源，它们给出和本章观点不同的看法。Hay[2]就有关于账目的一章。他关于账目和事务的基本概念和我的观点是一样的，虽然他没有给出记入规则的任何东西。他对企业中的账目类型分析得更为详细。他还讨论了企业中的公共事务和它们如何适用于账务模型。他还给出这些账目和事务类型的知识级。

伊利诺伊大学厄巴纳-尚佩恩分校的研究人员已经在开发账务框架方面做了大量工作[4]^Θ。伊利诺伊大学的研究小组所采取的方法与我和Hay的方法都不一样。他们以处理发票信息作为开头，把它作为到高级别账目的高级别“事务”。这个“事务”能被分解为到低级别账目的低级别“事务”。他们以和我不同的方式来使用事务这个词：他们不使用守恒原理。一个高级别“事务”可能是一个带有一行行子项的发票。框架致力于把它分解为低级别“事务”，比如那些子项本身。这样，框架被设计用于分解一簇条目，把它们变为子条目，而不是采用我这种方式，即把它们作为账目网络并来回转移。

参考文献

1. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
2. Hay, D. *Data Model Patterns: Conventions of Thought*. New York, NY: Dorset House, 1996.
3. Langer, S.K. *An Introduction to Symbolic Logic*, Third Edition. New York, NY: Dover, 1967.
4. Keefer, P.D. *An Object-Oriented Framework for Accounting Systems*. University of Illinois at Urbana-Champaign <ftp://st.cs.uiuc.edu/pub/Smalltalk/st80_vw/accounts/thesis.ps>, 1994.
5. Meyer, B. "Applying 'Design by Contract,'" In *IEEE Computer*, 25, 10 (1992), pp. 40-51.

[132]

Θ 参见<http://st-www.cs.uiuc.edu/users/johnson/Accounts.html>。