

## 第13章

# 应用外观

要完全理解本章，必须首先阅读第12章的12.3节。在12.3节中解释如何将应用分割成表示层和应用逻辑层。表示层包含所有用户界面逻辑，而应用逻辑层为表示层提供一系列自定义外观。这些应用外观负责为一个表示层选择和安排所有信息。

我们可以使用本章中描述的一个相当标准的技术来定义和建立应用外观。（因为不够典型，所以本章不包含模式。）这种技术可以被认为是对面向对象方法的一个补充。

一个应用外观看起来非常像任何其它的类型：它有属性和操作。然而，所有的属性都是从领域模型派生得来的。给出的模型是基于一个医疗保健的示例（参见13.1节）。一个外观的内容（参见13.2节）用一些连接到每个属性的方法来定义。这些方法描述如何检索属性、如何更新属性、如何得到一组合法数值、如何对属性进行验证、如何得到一个默认值。

一些公共方法（参见13.3节）可以在很多应用外观中使用，所以它们可以被转移到领域模型中。应用外观也包含一些局部于这个外观或者授权给领域模型的操作（参见13.4节）。用户界面框架一般不会涉及到领域模型中很多的相关类型，所以应用能够执行类型转换（参见13.5节）、创建用户界面能够理解的更简单的类型。一个应用经常包含多重外观（参见13.6节），它们能够用一个结构化模型来描述。

我已经在几个项目中使用了这种技术，包括英国国家健康服务系统和伦敦一家银行的交易系统。它是专门为应用逻辑层的外观设计的。它也适用于其它环境的外观，包括数据库交互。

[257]

### 13.1 一个医疗保健示例

从一个比较复杂和抽象的领域模型中最容易理解应用外观。图13-1显示了这样一个模型，它的基础结构是基于为医疗保健而设计的Cosmos模型[1]。在第3章中可以找到对大量这种思想的更多解释，在继续本章内容之前阅读那一章是值得的。

考虑从一个医院信息系统中得到的一个例子，该系统需要记录每一个患者的信息，这些信息是从医院的所有部门得到的，以便能够为每个患者保存一个完整的医疗记录。关于每个患者所能记录的信息的范围是巨大的。为了缩小模型的体积，使用一个抽象的方法，如图13-1所示。

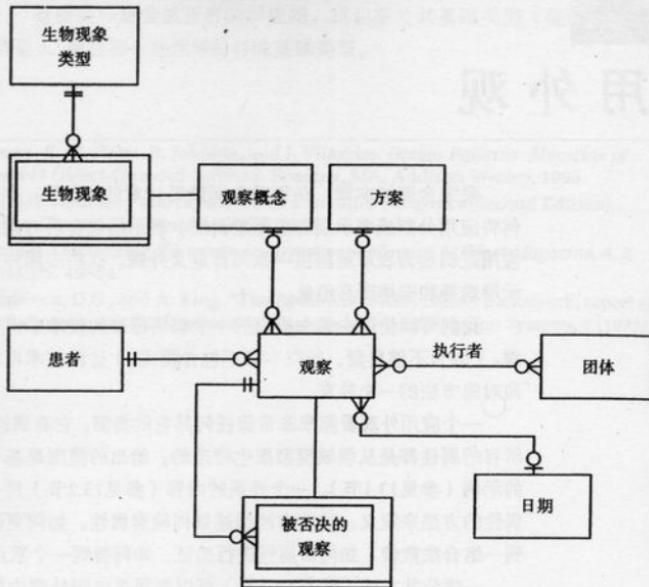


图13-1 从医疗保健而来的一个领域模型示例。领域层是建立在这个模型上

模型按照生物现象和生物现象类型的形式描述关于一个患者能够记录的所有信息。例如，一个性别的生物现象类型具有“男”和“女”两个生物现象，一个血型的生物现象类型具有的生物现象是A、B、A/B和O。为了说明一个患者的血型是O型，我们使用一个把患者连接到适当的生物现象的观察。我们也能够说明关于观察的其它适当信息，例如谁执行它（执行者），什么时候执行它（日期），如何执行它（方案）。如果后来发现这个观察是错误的并且正确的血型是A型，则我们丢弃原来的观察并且用一个新的观察替换它。这是维持一个患者的完整记录所必需的。

这样的模型适用于很多种情况。然而，输血部门的需求更简单、更集中。它只希望记录患者的一组属性。例如，考虑一个献血者的登记。一个献血者的属性包括：姓名、血型和最近一次献血日期。姓名很简单，因为它直接连接到患者类型。可是血型和最近一次献血日期却需要比较复杂的

处理，下面我们就会看到这一点。

## 外观的内容

每个应用外观都由一个指向领域模型的引用（作为外观的主题）和一些为外观的用户描述信息的属性组成，如图13-2所示。

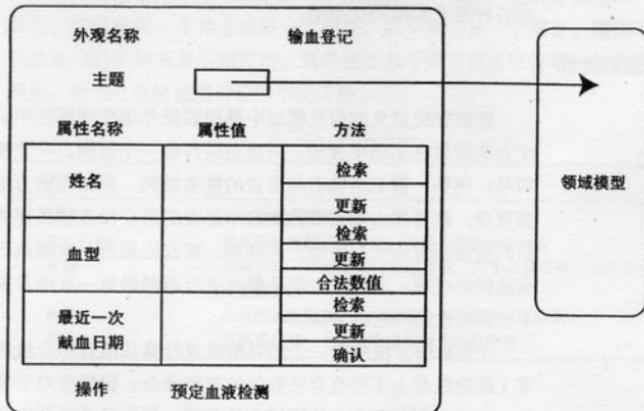


图13-2 一个应用外观的组成部分

一个应用外观的对外是领域模型中的一个特殊对象，它作为外观的主题。这个主题担当外观进行的所有处理的起始点。当我们定义外观时，我们就定义主题的类型。对于输血登记这个例子，主题就是“患者”。外观的用户从不直接访问主题，但是把外观作为主题的一个逻辑窗口来处理。

因此，外观中的每个属性都作为主题的一个逻辑属性。每个属性的类型应当已经被定义好，并且这个类型应当与领域模型上的一个类型相符。同样地，我们也能够在外观上定义操作。在献血者登记的例子中，我们有一个献血者外观如下所述：

259

**应用外观：献血者**

**主题：患者**

**属性：**

姓名：字符串

血型：生物现象

最近一次输血日期：日期

**操作：**

预定血液检测

接着我们为外观的每个属性定义一系列方法。这些方法描述如何把信息从领域模型转变成外观，以及外观如何更新共享信息。定义这些方法有不同的方式。一种方式是用一个英语句子，这种方式容易理解但会导致歧义。与之相对的极端方式是一种形式化的方法（如谓词演算），假如每个人都可以理解这种方式，那么这种方式才是合适的。在这两种方法之间存在各种形式的结构化英语。

### 13.2.1 方法的类型

检索方法定义如何从模型中得到数据并填充到属性中。我们可以把这个方法看作在主题中开始、对模型进行的一个查询。一个检索方法可能很简单；例如，登记名称会与患者的姓名相同。但是检索方法也可能变得相当复杂。患者的血型需要找出这个患者的所有没有被丢弃的观察，这些观察的生物现象类型是血型。同样地，要找出最近一次献血日期需要执行所有这样的过程：它们的方案是献血并且返回最近一次的日期。只读属性将不会拥有任何其它的方法。

合法数值方法提供一个能够用来进行确认检查的合法数值的集合。常常（就像姓名）不存在合法数值的有穷集合；但类型对于确认足够了。然而，对于血型却要求一些更复杂的东西：属性的类型是生物现象，但是只允许那些生物现象类型是血型的生物现象。因此，合法的数值是由一个返回生物现象 A、B、A/B、O 的集合的查询提供的。这些数值不仅对确认检查有用，而且能够用来填写用户界面上的一个菜单或者列表。

更新方法是一个需要最强大技术的方法。虽然检索方法是简单的，但一个更新的意义却可能变化巨大。姓名对象又提供对一个患者的属性进行更新的一种简单情况。改变血型要复杂得多，所以我们需要创建一个连接到该属性支持的生物现象的新观察。丢弃旧的观察，并且把它连接到新的对象以便显示哪一个对象丢弃了它。另外我们还能够提供一些隐藏信息。例如，血型中的改变通常由输血单元提供，并且该单元通常使用相同的方案，因此我们能够把这个信息自动地加入到记录中，方法是把登录的医生作为该过程的执行者并且使用标准的方案。显然，我们必须注意有多少信息是隐藏起来的，并且这些信息应当反馈给用户。

如果合法数值和属性类型对合法性检查都不够，那么就需要一个确认方法。需要提供的确认规则是一个专门针对某个外观的上下文。例如，最近一次献血日期可能需要比今天早并且比当前显示的献血日期晚。

默认方法在创建一条新记录时使用，与之相对的是一条现存记录的更新。为了减少复杂性，我们通常假定创建一条新记录与更新一条空记录相

同。然后用户会填写属性，并且使用的确认方法与更新记录时使用的确认方法完全相同。默认方法表明：如果用户从一条空记录开始应当提供什么信息。它的组织形式与一个检索方法非常相似。

一些属性拥有的数值不是一个而是一列。在这种情况下，有两个更新方法：一个是增加一个项目，另一个是删除一个项目。检索方法返回一个聚合，它可能是一个集合或者一个列表。如果聚合是一个集合，则排序标准是专门表示数值显示顺序的。通常这是基于数字或者字符串的标准排序准则。表13-1总结这里讨论的不同方法。[261]

表13-1 方法总结

方法名称	描述
检索	符合领域模型的数值
合法数值	如果数值范围小于类型，就是正当数值的集合
更新	为数值的一个改变如何更新模型。对于多值属性，必须指定增加和删除更新
确认	用来测试新数值，只有比合法数值复杂时才必需
默认	从外观创建一个新对象时使用的初始数值

## 2 样本方法

表13-2显示一个关于这些方法应当如何书写的例子，这个例子是从上面给出的献血者例子中提取的。这些规则没有用一种形式化的符号来表达（因此是含糊的），但是我们使用了一种伪SQL的风格来书写，已经证明这是介于严格和易于理解之间的一个合理的折衷。

表13-2 一个外观的样本方法

属性	方法名称	方法体
姓名	检索	subject.name
	更新	Change subject.name
血型	检索	subject.observations.biological_phenomenon where biological_phenomenon.biological_phenomenon_type = 'Blood Group'.
	更新	oldObs := All subject.observations where subject.observations.biological_phenomenon.biological_phenomenon_type = 'blood group'. Create new observation(newObs) where newObs.patient = subject, newObs.biological_phenomenon = new Blood Group, and newObs.rejected_observations = oldObs.
合法数值		All biological phenomena with biological_phenomenon_type = 'blood group'

(续)

最近一次 输血日期	检索	the lastest subject. observations. date from those subject. observations with protocol = 'blood transfusion'.
	更新	Create new observation with patient = subject, protocol = 'blood transfusion', and date = Date of last transfusion
	确认	new Date of last transfusion later than old Date of last transfusion

### 13.3 `公共方法`

在使用外观的应用中，我们看到很多方法具有一个类似的结构。在医疗记录模型中，血型属性就是通用情况的一个例子。血型方法检索一个患者的具体生物现象类型的一个具体生物现象，假设一个患者只有一个那种类型的生物现象。在请求血型时我们提问“这个患者的观察是针对类型血型的哪些生物现象？”这种方法存在于很多种情况下（例如一个患者的性别）。因此，如果有一个通用的服务不仅处理公共访问和更新的情况，而且负责所有特殊情况（例如一个患者存在不一致的观察）的处理，那将是很有意义的。

我们可以把这样一些服务合并到领域模型中作为针对患者的操作或者计算映射。在我们的献血者例子中，这导致一个操作：

`valueOf(aBiologicalPhenomenonType): aBiologicalPhenomenon`

注意：针对患者，可能有一个相应的更新操作也包含应用外观的更新方法。

把应用外观方法转移到领域模型中有两个方面的用处。首先，它们提供一个到外观的更高层接口，简化应用外观的开发。特别是，这意味着处理这些种类属性的公共代码可以在领域模型中只保存一次，而不用复制到很多应用外观中。这种方法的第二个价值是，它提供一种好的最优化的方法。这些代码非常通用，因而可以被保存在一个共享方法中，这个事实说明这些代码将会被频繁地执行。因此，这就是最优化的一个很好目标。当OO系统提供相对说明性查询来说更具有导航性的查询时这会变得尤其重要。

很明显，不是每个应用外观方法都应当被转移到领域模型中。应用外观的价值是它们把局部于某个上下文的内容和必须共享的内容分离开。每个转移到领域模型中的外观方法都会增加领域模型的复杂性。因而设计者必须质疑把外观方法转移到领域模型中的可行性，只有当这样做的好处超过将会增加的复杂性时才可以进行。

### 13.4 操作

和其它任何对象类型一样，应用外观既包含数据又包含过程。13.2.1节中讨论的方法是外观的私有方法，它们管理应用外观和领域模型之间的映射。还有公有方法负责访问和更新外观的属性。

应用外观中的其它操作不仅仅是对属性的处理。这些操作应当单独进行声明，并且通常都包含一些复杂的处理。需要考虑这些操作是局部的还是共享的，如图13-3所示。一个共享操作在整个组织中使用，而一个局部操作只被某个应用使用。如果操作是共享的，那么它应当在领域模型中实现并且属于最适当的共享类。共享操作应当忽略任何外观并且只操作共享对象。因而对外观的操作应当只把调用传递到共享操作上，同时提供必要的参数并且为了外观内部的使用而解释返回值。

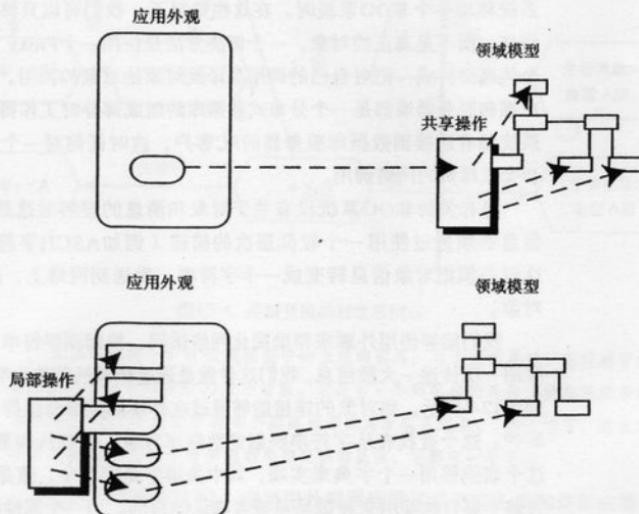


图13-3 应用外观中的操作

一个共享操作在领域模型中实现并且访问领域模型的结构和服务。外观提供对这个操作的一个引用，而一个局部操作在外观中实现并且只访问这个外观的属性和操作。

然而，一个局部操作不应当放在领域模型中，而应当在局部模型中实现。它不应当使用领域模型的结构和操作，而应当依赖于应用外观的属性和操作。由此，局部和共享代码就被明显地分隔开。

要注意的是：局部和共享操作之间的区别完全是一个对代码的概念共享的问题。它不影响环境（例如一个客户/服务器环境）的实现时所关心的事情。根据环境，局部操作能够运行在一个服务器上，或者共享操作能够在一个客户端运行。这个区别只是基于这些操作在概念上是否是共享的。共享操作被大量重用并且必须更加仔细地维护，就和领域模型的其它部分一样。局部操作可以只在外观内部进行处理。只有在它们所在的外观被重用时它们才会被重用。

### 13.5 类型转换

使用OO系统的一个困难就是在一个网络中移动对象的复杂性，尤其是在从一个对象空间移动到另一个对象空间时。这个问题发生在信息需要从一个OO系统移动另一个具有不同对象ID的OO系统时，或者从一个OO系统移动一个非OO系统时。在这些情况下，我们可以只移动关于对象的信息，而不是真正的对象。一个解决方法是使用一个Proxy（代理），把这个代理设计成：把对自己的调用翻译成对原始对象的调用。这个系统在客户端和服务器端都是一个分布式数据库的组成部分时工作得很好，但很多系统都有连接到数据库服务器的PC客户，这时任何对一个对象的调用都会变成昂贵的网络调用。

当有关的非OO系统没有关于对象和消息的理解时这是尤其重要的。信息必须通过使用一个较低层次的描述（例如ASCII字符串）来传递。这时必须把对象信息转变成一个字符串，发送到网络上，然后反译码为对象。

我们能够使用外观来帮助简化网络访问、控制到字符串的转变、允许应用一次传递一大段信息。我们这样做是通过把属性值作为字符串来处理，如图13-4所示。到对象的连接能够通过在外观的类部分保持一个查找表来维护。这个查找表从字符串映射到数据库对象，使确认和更新更加容易。这个表能够用一个字典来实现，其中关键字是字符串，值是数据库对象。关键字集合能够用来装载菜单或者确认的目的。当一个属性被改变时，这个表能够把它转变成一个对象以便进行数据库中的替换。因为这个表保存在外观的类部分中，所以它只被保存一次。如果对适当的选项进行了一个改变，那么它也需要被刷新：这种改变通常很少发生。

因此，血型属性在类部分中有一个对应的字典，BloodGroupValues。这个字典有关键字字符串‘A’、‘B’等，它们有数据库中的对象作为值。检索时血型被转变成一个字符串（使用一个名字函数或者通过字典）并且存储在属性中。当被更新时，新的字符串被用作一个到字典的查询，并且

相应的值被用来进行数据库中的更新。

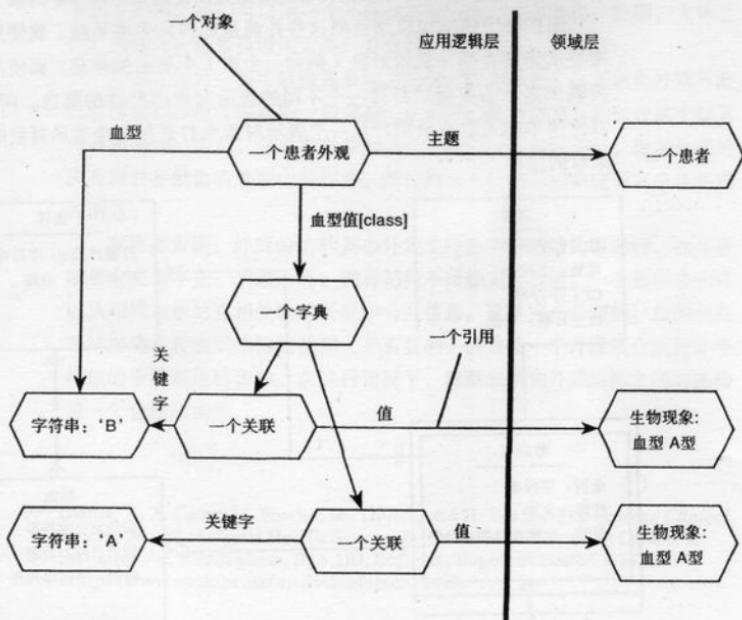


图13-4 类型变换的对象示例xx

应用外观有一个到领域模型中的主题的引用。对于它的属性，它保存字符串并且发送字符串到对应的表示对象。它还有一个到字典的连接（静态的或者通过它的类），这个字典把字符串和隐藏的领域对象联系起来。（为了清楚，这里只显示两个血型。）对每个需要这种类型变换的属性，它都有一个字典。

当使用这种方法时，应当把外观属性描述成具有内部和外部的类型。内部类型是在领域模型中的类型，而外部类型是提供给表示对象的类型。在血型的例子中，内部类型是“生物现象”，而外部类型是“字符串”。

### 13.6 多重外观

应用外观通常不单独出现，而是成组出现。一个应用由一些表示和相应的外观组成。这些构件能够用两种方法连接起来。第一种方法是使外观包含构件，如放在一个表中。例如，一个输血历史，每条都有地点和日期，可以显示为在一个全部献血者表示内的一个表。第二种方法是允许用户

从一个表示对象遍历另一个表示对象。例如，一个正在观看验血信息屏幕的用户能够打开另一个单独的屏幕观看验血所使用的血液样本的细节。

图13-5中的结构模型说明这些外观是如何发生关系的。我使用聚集来展示需要在同一表示对象（例如一个表）中显示的信息，而使用规则关联来显示需要通过打开一个不同的表示对象而获得的信息。同样地，我使用单向关联表示用户从一个表示对象中打开另一个表示对象时能够采取的路径。

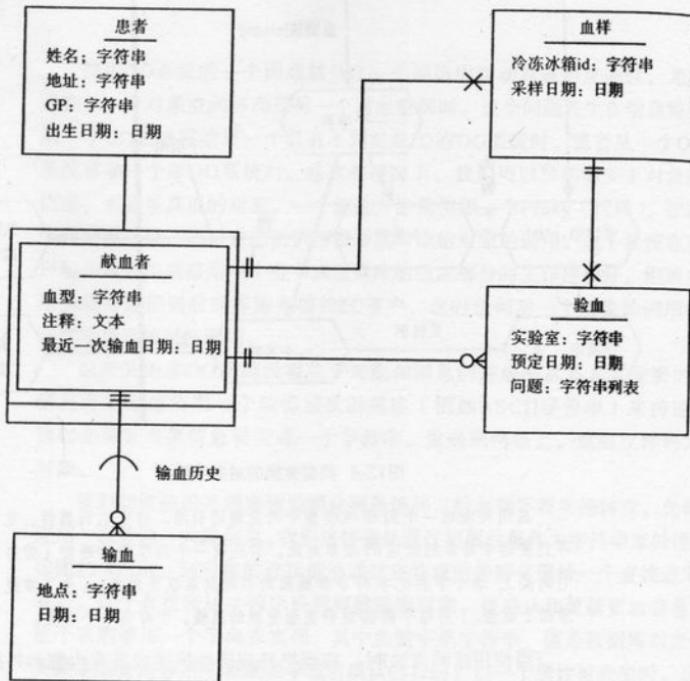


图13-5 应用外观的一个示例图

显示的类型是外部类型（参见13.5节）。这个模型表明我们有一个献血者的表示对象，它在患者的表示对象上增加了所有显示的信息。它还显示一个输血的表。用户能够遍历到一个显示验血列表的单独表示对象。从这个验血的表示对象，用户可以遍历到适当的血样的表示对象，从后者用户又可以遍历到献血者的表示对象。

使用一个结构模型是非常有用的，但是有一点很重要，即记住建模的

风格是不同的。在领域模型中应当避免复制职责，尤其在保持信息时。因为这个原因，我们可以用一个不同的符号强调不同的策略。然而，大体上我认为额外的符号会增加太多的复杂性。

267

外观之间另一种可能很重要的关系是子类型关系。一个患者外观可能已经覆盖很多患者需要的一般信息。献血者需要的信息可能包含这个信息并有所增加。因此，献血者外观是患者外观的一个真正子类：患者外观的所有属性在献血者外观中都存在，并且献血者外观能够响应所有患者外观的消息。

268

在许多方面，外观的结构是由外观支持的表示的结构驱动的。在外观的集合支持不止一个表示时，两种结构不可能完全对应。一个新的表示可以从相同表示的互相关联的外观中合并信息。这是完全合理的。虽然把外观结构建立在表示结构的基础上是有益的，但是使一个外观集合支持多个相似的表示也是明智的。在这种情况下，截断表示和外观结构之间的连接是一个合理的牺牲。

## 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.

269

270

## 第14章

# 类型模型的模式——设计模板<sup>⊖</sup>

本书使用了非常概念化的模型，因此解释这些模型如何转化成软件对我来说是很重要的。本章提供能够用来为类型模型构造设计模板的转换模式。转换模式描述把一个人工制品从一种形式转换成另一种形式的原则。设计模板描述如何把一个隐式规约模型转变成一个显式规约模型和一个实现。因为隐式接口模型和概念模型基本相同，所以它们对于理解概念模型如何与实现相关联是很有价值的工具。

本章并不试图提供针对任何特殊实现环境的设计模板的一个完全集合。实现环境之间差别很大，每一个都需要不同的权衡。这不只是一个Smalltalk或者C++的问题。很多因素——硬件、数据库、网络、类库——都影响在一个项目上实际使用的模板。因此，我集中关注那些在设计模板中发现的模式——即通用的原则和执行转换时应当考虑的问题。

设计模板的不同是基于所使用的建模方法、实际实现环境、企业标准和最终系统的性能需求。对它们的使用能够以规定的方式或者以他人建议的方式进行。它们可以由一个代码生成器实现自动化（至少在理论上实现自动化），或者以手工方式使用（例如，作为编码标准）。

不是所有的方法都需要设计模板。如果所有的建模都使用一个深深植根于实现环境的方法，则转换即使需要也很少。这是使用一个基于实现的技术的主要优势。这样一个方法的问题是：人们对世界的看法和一个基于实现的模型之间存在很大的差距。而且把这样一个模型移植到另一个实现环境时通常也有问题。

设计模板有以下几个目标：

- 只要实际上可能，确保软件的构造方式与概念模型相同。
- 提供软件内部的一致性。
- 提供构造软件的指导方针，以便知识在组织中能够得到有效传播。

这些目标把我们引向一个重要的原则：设计模板应当定义软件构件的

接口并且提出实现这些构件的建议。这个过程的一个目标应当是：一个不熟悉领域但熟悉模板的程序员通过观察分析模型就可以很容易地知道所有构件的接口是什么。在实际中要完全达到这个目标或许是不可能的，但是我们应当尽可能地实现它。

**建模原则：**设计模板定义软件构件的接口并且提出实现这些构件的建议。

因此，设计模板应当提供对所需要的接口的声明并能够提供一定数量建议的实现。程序员必须接受强制的接口，但是他们能够采取任何实现，或者从建议的列表中选取，或者提出他们自己的选择。类的用户不需要知道或者关心选择了什么实现。尤其是类的实现者应当能够在不改变接口的情况下改变实现。

保持一个纯粹的概念模型是很困难的。为了确保接口能够被完全定义，模型必须是一个规约模型。它不需要是一个非常显式的规约模型，因为模板把这个模型转变成一个真正的显式规约模型。在一些情况下，接口因素会从一个纯粹概念化的角度导致模型的改变。这些改变并不重要，而且容忍它们通常比创建一些独立的模型并试图使它们同步更好。这些问题将在本章的后面进行讨论。

本章的每一节都讨论一些转换概念模型的模式。我们从讨论一个实现关联的模式（参见14.1节）开始。有三个实现：在两个方向上的指针，在一个方向上的指针，关联对象。根据基本的原则，它们都具有相同的接口。基础类型有一些特殊的考虑。关联是几乎所有技术共有的，所以这个模式可被广泛应用。

272

第二个模式讨论实现泛化（参见14.2节）。很多方法把泛化和实现的继承同样对待。本书使用多重和动态分类（参见A.1.3节），它使转换不太直接。我们考虑五个实现：继承、多重继承组合类、标志、委托给一个隐藏类、创建一个替换。我们再次定义一个公共接口，该接口包括一个测试对象类型的操作——使用时要多加小心。

剩下的模式更短并且包括对象创建的模式（参见14.3节）、对象析构的模式（参见14.4节）、使用一个入口点寻找对象的模式（参见14.5节）、实现约束的模式（参见14.6节）。我们会简要提到其它技术的设计模板（参见14.7节），但没有进行详细讨论。

如果你不使用设计模板，那么你可以把本章作为对程序员应当如何解释概念模型的范例。本章的技术对于把本书中的模型转换成更加基于实现的方法（也包括转换成OO语言）是非常有价值的。任何想要借助Booch方法（举个例子）来使用本书中分析模式的人都将需要使用这些模式，尤其

在处理泛化时更是如此。

不同的语言对不同的元素有不同名称。我使用术语域 (field) 来表达类的一个数据值 (一个Smalltalk实例变量或者一个C++数据成员)。我使用术语操作 (operation) 来指一个类能够识别的一个消息 (一个Smalltalk方法或选择器, 或者一个C++成员函数)。我区分操作 (声明) 和方法 (体); 因此一个多态的操作有很多方法。我使用术语特征 (feature) 来表示一个域或者操作。

本章假定你可以使用聚合类的一个类库。聚合 (collection), 也称容器 (container), 是一些拥有一组对象的类。传统的编程语言中所提供的最普通的、通常也是惟一的聚合是数组。对象环境能够提供大量聚合。Lewis[5]给出最普通的Smalltalk聚合的一个非常优秀的综述。很多C++版本使用类似的方法, 虽然这些将被标准模板库 (STL) [7]所取代。这些聚合包括: 集合 (无序, 无重复)、列表 (Smalltalk中的orderedCollection, STL中的vector和deque)、bag (像集合但有重复, STL中的multiset)、字典 (STL中的map)。一个字典是一个查找表或者一个关联数组, 它允许你查找一个对象时使用另一个对象作为关键字。所以我们可以有一个用名字作为索引的关于人的字典。你可以通过发送一个形式为PeopleDictionary at ("Martin Fowler") 的消息找到我。

这些聚合大大简化了编程, 并且具有这些可用性是面向对象环境最大的好处之一。很多环境 (包括所有的Smalltalk) 都具有这样的一个类库。大多数C++环境没有提供聚合类, 虽然从一些销售商那里可以很容易地买到它们。我强烈建议你熟悉并使用聚合类。在一个面向对象环境中工作而不使用聚合类就像编程时把一只手放在身后。

## 14.1 实现关联

本章以关联开始是因为它们提供了关于模板如何工作的一个简单而重要的例子。为了这一目的, 我们将假设所有对象类型用类实现; 这个假设以后将被改变。

一些面向对象的专业人员不习惯在OO分析中使用关联。他们认为关联违反了封装的OO编程原则。使用封装可以把一个类的数据结构隐藏在操作的接口后面。一些专业人员认为关联使数据结构公开化。摆脱这种困境的方法是要理解如何在OO语言的上下文中解释关联。提出关联是因为它们在概念建模中有用。如果它们被看作是用于描述某个对象类型有责任记住和改变它与另一个对象类型的联系的一种方法, 那么它们就不会与封装冲突。因此图14-1中的例子显示雇员有责任知道他的雇主并且能够改变

他的雇主。相反，组织有责任知道它的雇员并且能够改变它的雇员。在大多数的OO语言中，这个责任是通过访问者和修改者（get和set）操作实现的。当然可以使用一个数据结构，并且在大多数情况下都可以，然而一个数据结构不是概念模型所指定的。

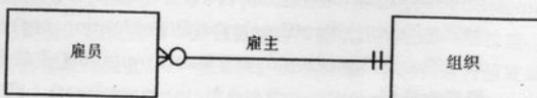


图14-1 一个关联的例子

属性能够被表示成单值映射，通常映射到基础类型。因此对方法而言，关于单值映射的讨论也适用于它们所使用的属性。

## 1 双向关联和单向关联

首先需要考虑的问题之一是使用一个双向关联还是单向关联。关于这个问题有很多辩论。单向关联更容易实现并且在软件中造成的耦合性也较少。但是它们使我们更难找到解决问题的办法。本书中的模式使用双向关联。我们可以选择把所有的关联都实现为双向的，或者都实现为单向的，或者混合使用。混合使用的一致性较少，但确实有优势。如果我们全部使用双向关联，我们会陷入耦合性问题。如果我们全部使用单向关联，我们可能会发现一些关联真的需要是双向的并且采取一个例外是值得的。

274

如果我们要使用一个单向关联，需要决定支持哪一个方向和放弃哪一个方向。应用将会对此提出建议。一个好的经验是观察关联的客户想要做什么并且顺应他们需要的方向。我不信任用很多方法学的方式对访问路径的细节进行分析。我们应当首先做最简单的事情，但是如果我们的需要后来发生变化也应当做好准备改变它。如果我们保持一个模型，我们应当更新它以显示我们正在使用的方向。

如果我们使用双向关联，必须小心使用那些穿过包的关联。如果我们维护双向性，将导致分类之间的相互可见性，这在11.2节讨论过。当我使用双向关联时，我在一个分类内部自由使用它们但避免在分类之间使用它们，因为减少分类之间的可见性更重要。

## 2 关联的接口

在一个OO语言中，关联的接口是用来访问和更新关联的一系列操作。这些操作的确切的项和结构依赖于包含的映射的基数。

一般来说，一个单值映射需要两个操作：一个访问者和一个修改者。

访问者没有参数并且返回接收者所映射的对象。修改者带一个参数并且把接收者的映射变成那个参数。不同的命名约定是可能的。Smalltalk中的约定是把两个操作都命名为mappingName，而修改者与访问者的区别是其具有参数。因此，图14-1中的雇员类应当有两个操作：employer和employer: anOrganization。C++中没有标准的约定存在，但是像getEmployer()和setEmployer(Organization org)这样的命名经常是很通用的。使用getEmployer()和setEmployer()是最自然的，但是一些人更喜欢使用employerSet()和employerGet()（或者employerOf()和employerIs()）以便在一个按字母排序的浏览器中这两种操作可以一起出现。

一个多值映射需要三个操作。也有一个访问者，但它返回一个对象的集合。所有的多值映射都被假定是集合，除非有其它的说明。非集合的接口是不同的，并且超出了本节的范围。需要两个修改者：一个负责增加一个对象，一个负责删除一个对象。访问者的命名方法通常和单值映射相同，除非我认为一个复数形式能够格外增强它多值的本质（例如，employees或者getEmployees()）。修改者的形式是addEmployee (Employee emp)、removeEmployee (Employee emp)或者employeesAdd: anEmployee、employeesRemove: anEmployee。

没有必要在一个双向关联的两边都提供修改者。修改者经常看起来好像只能用于一个方向，通常这个方向是受约束最多的（例如Employee: employer）。在一个双向关联的两个方向上都应当提供访问者；这就是它成为双向关联的原因。

在一个双向关联中，修改者必须总是确保两个映射都被更新。因此改变一个雇员的雇主就不仅要改变从雇员到组织的连接而且要改变相反的连接。我们在14.1.5节到14.1.8节中讨论这个实现。

修改者还应当确保对约束进行检查。实际上，如果上界大于等于1，那么它就会被接口自然覆盖，所以只需要为其它的数字进行检查。如果下界非零那么通常需要显式的检查。在单值映射中，下界表明是否能够把提供null（空）作为一个参数。对于多值映射，一个下界意味着删除操作中的一个检查。一个映射的基数能够影响实现其它映射的操作。例如，在图14-1中的组织上不应当有一个删除雇员的操作，因为如果不破坏对雇员的约束这是无法进行的。因为同样的原因，不应当为一个不变的关联提供修改者。

如果类型检查没有被构建在语言中，那么可以在修改者中执行它。这在根本无类型的Smalltalk中是一个争论点。为了进行类型检查，你需要某

种类型测试能力，这在14.2.6节中进行讨论。我喜欢把类型检查放在一个专门的前置条件块中。所有对象都有一个称为require: aBlock的操作。该操作估算这个块，如果它的结果出错就产生一个异常。然后我在这个子句内使用一个像self require: [aCustomer hasType: #Customer]的语句来测试类型。这使我可以很容易地为性能的原因取出类型检查，就像Eiffel中的前置条件检查。（一般来说，我就是用这种结构进行前置条件检查的。）

一个多值操作的访问者返回的集合能够用来进行进一步的处理，这种控制使用环境中现存的任何集合类提供的便利。然而，你必须确保：通过增加或者删除对象而对集合的成员关系进行修改这个行为不会改变用来生成这个集合的原始映射关系。对映射的修改只能来源于作为显式接口组成部分的修改者操作（参见6.9节）。 [276]

有时候，一个多值访问者返回的集合可能是一个性能焦点。在这些时候可以把接口扩展成包括普通集合操作（例如选取、执行和收集）和一个迭代器[4]。这些扩展应当遵循你所使用的集合类的命名约定。可是，这些接口扩展可能会使接口变得臃肿。

在C++中经常存在一个关于访问者应当返回什么的问题：对象，还是指向对象的一个指针。不管返回什么都应当由设计模板说明清楚。一个通常的约定是：对所有内置数据类型返回数值，对所有基础类返回对象，对所有其它的类返回指针。在Smalltalk中这不适用，因为你总是使用对象，或者至少看起来是这样的！在下面的讨论中我一般喜欢返回引用；实际的模板应当确切地弄清楚对于C++和类似的显式指针语言返回什么。

#### 14.1.3 基础类型

一些对象类型在一个模型的所有部分都相当简单和普遍。因此，它们与绝大多数对象类型相比需要的处理稍有不同，尤其在考虑到关联时。编程环境的典型的内置数据类型就是这种对象类型的例子：整型、实型、字符串、日期。然而，好的OO分析通常会发现其它的例子：数量、金钱、时间段、货币是一些典型的例子。难以给出使一个类型基础化的规则——它主要是来源于类型在整个模型中的存在和某种内在的简单性。这意味着如果基础类型的关联用标准方法实现，那么将会有大量把该基础类型连接到模型中其他类型的操作。因此，对于基础类型就不应当实现其到非基础类型的映射；也就是说，不应当存在操作。另外，到其它基础类型的关联应当针对每一种情况进行处理。

在一个模型中用某种方法显示出基础类型是有用的。一种方法是在词汇表中把对象类型标记为基础类型。另一种方法是使用单向关联。使

用单向关联的问题是：它们本质上是一个实现特征并且可能使非IT的分析员感到迷惑。

基础类型的一个普遍特征是：它们的关键特征是不变的，即你不能改变类型的任何属性。考虑对象5美元。如果不描述一个单独的对象，那么你既不能改变数字（5）也不能改变货币（美元）。然而并不是所有的属性都不可改变。货币能够作为一个基础类型，可是它也可能有可改变的属性，例如假期列表（为了交易的目的）。不可改变的属性得到合适的确保对于基础类型是尤其重要的。

277

#### 14.1.4 实现一个单向关联

实现一个单向关联是非常简单的。在作为单个映射的起源的类中有一个域，这个域包含到目的对象的一个引用。访问者返回这个引用，而修改者改变这个引用。

#### 14.1.5 在两个方向上都使用指针的双向实现

在这个实现中，关联是用从两个组成类引出的指针实现的。如果一个映射是单值的，那么就用从一个对象到另一个对象的简单指针，就像图14-2中从Peter到NASA的指针。如果一个映射是多值的，那么对象将有一个由指向其它对象的指针组成的集合（例如，在图14-2中NASA指向一个指针的集合，这个集合包括指向Peter、Jasper、Paul的指针）。对于支持包容的语言，更好的方法是包含这个指针的集合而不是指向它。然而，可能会存在空间的不明确，因为集合能够随意增长。与此类似，如果一个单值映射指向一个内置数据类型或者另一个基础类型，那么就能够用包含代替一个指针。

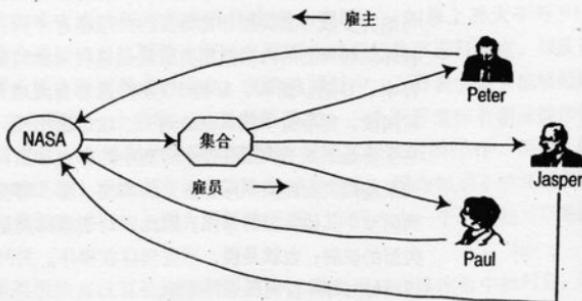


图14-2 在两个方向上都使用指针的实现

访问者操作比较简单。对一个单值映射，访问者只返回引用。对一个  
多值映射，访问者返回一个引用的集合；然而，它绝对不能返回原来的那个  
引用集合，因为这样一来用户就能够改变集合的成员关系并且破坏了封装。  
封装的边界应当适用于所有实现多值映射的集合。一种解决方法是返回该集合的一个拷贝，这样即使进行了任何改变，它们也不会影响真正的  
映射。可是，对于大集合，这可能会导致巨大的时间开销。可选择的方法是返回一个保护代理或者一个外部迭代器[3]。保护代理是一个简单的类，  
它有一个简单的域包含那个集合。所有允许的操作都在保护代理上定义，  
实现的方法是把这些调用传递到内部包含的集合上。这种方法的更新能够  
被分块。一个外部迭代器更像是一个进入聚合内部的光标。迭代器能够返回  
当前指向的对象并且能够在聚合中前进。

278

既然实现两个对象之间的每个连接都需要两个指针，那么修改者保持  
它们同步就是很重要的。因此一个要把Peter的组织改为IBM的修改者不仅  
必须用一个指向IBM的指针替换那个从Peter发出的指针，而且必须删除在  
NASA的雇员集合中指向Peter的指针并且在IBM的雇员集合中创建一个新的  
指向Peter的指针。但是这样做会使我们陷入一个OO难题。雇员需要使  
用某个只操纵集合指针而不返回一个到Peter的调用的操作（否则我们会陷  
入一个无穷循环）。然而，这个操作不可以是组织的接口的组成部分。在  
C++中，这是友元结构的一个典型应用。在Smalltalk中，我们必须创建一  
个这样的操作但是会把它标记为私有的（这样当然不会妨碍雇员使用它）。  
在这种情况下，一个有用的措施是只让一个修改者做实际的工作，它操纵  
数据和/或私有操作。其它的修改者应当只是调用这个修改者。这就确保  
更新的代码只有一个拷贝。

这种实现工作得很好。两个方向的遍历都很快。虽然确保所有的指针  
一起被更新需要一些技巧，可是一旦这一点被解决，整个解决方法就很容易  
被重复。它的主要缺点是：多值映射所需集合的规模和较慢的更新速度。

### 在一个方向上使用指针的双向实现

这个实现只在一个方向上使用指针。为了在另一个方向上遍历，我们  
需要考察类的所有实例并且选择那些向后指向源对象的实例。在图14-3中，  
雇员映射需要得到雇员的所有实例并且选择那些雇主是NASA的雇员实例。

修改者很简单。拥有指针的类的修改者只是改变指针，并且公共程序可  
以被其它类的一个修改者直接调用。走乱步伐的多重指针就不会产生危害了。

这个方案节省空间，因为每个连接只存储一个指针，但是当向指针的  
反方向遍历时速度将会变慢。它的更新速度很快。

279

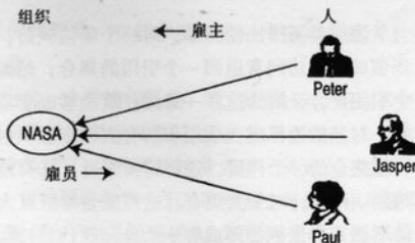


图14-3 在一个方向上使用指针的实现

#### 14.1.7 使用关联对象的双向实现

关联对象是带有两个指针的简单对象，它能够被用来连接两个其它的对象，如图14-4所示。通常，为每个关联提供一个关于这些对象的表。访问者的工作方法是：得到这个表中的所有对象，然后选择那些指向来源的对象，最后根据每一个指针找到被映射的对象。修改者很简单，只是创建或者删除关联对象。可以建立特殊的关联类；或者，可以使用具有散列表查找能力的字典类来实现它们。

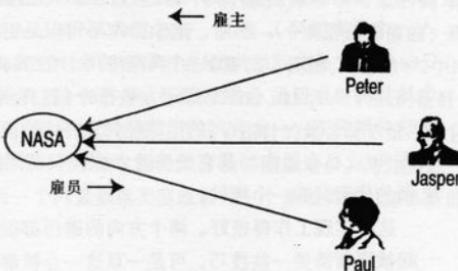


图14-4 使用关联对象的实现

关联对象在两个方向上都不是很快，但是通常能够通过使用索引（使用一个字典）来提高速度。如果大多数对象在映射中都没有被关联，那么它们是节省空间的，在这种情况下空间只在需要时才被使用。如果不可能改变两个组成类的数据结构，则它们也是有用的。

#### 14.1.8 双向实现的比较

大多数情况下，我们都在两个方向上的指针和一个方向上的指针之间进行选择。前者提供在两个方向上访问的速度，而后者对空间的利用更有

效并且更新更快。集合的基数和连接的实际数目影响着对它们的权衡。

关联对象在特殊的情况下是有用的，但总体而言它们不是第一选择。

## 9 派生映射

派生映射看起来基本上与其它种类的映射没有区别。它和基本映射一样提供访问者；它们应当是无法区别的。然而，通常不可能提供一个修改者。关于派生映射很重要的事情是：它们所隐含的位于派生映射和组成派生的其它映射组合之间的约束。

## 10 非集合映射

虽然大多数多值映射都是集合，但是也有例外。在本书中它们用短语义声明来表示，例如[list]、[hierarchy]、[key:mappingName]。这种声明意味着一个不同的接口。用[list]表示的映射将返回一个列表而不是一个集合，并且将会有像addFirst、addLast、addBefore(Object)、indexOf(anObject)这样的修改者。在这里，我并不想提供书中这些情况的所有接口。但是，如果我们使用这些结构，我们将保证会给出它们的设计模板。通常我们应当使接口基于下层聚合的接口之上。我们也能够把这些构造看作是关联模式（参见15章）。

## 实现泛化

OO类型建模和大多数传统的数据建模实践之间最显著的区别之一就是泛化的大量应用。虽然泛化作为很多数据建模方法的组成部分已经很长时间了，但是它通常都被看作一个先进的或者特殊的技术。泛化和OO继承之间的紧密联系确保它在OO分析中的核心地位。

很多OO方法把泛化作为一个等同于继承的分析来使用。然而，需要对使用动态和多重分类的方法进行更多的考虑，因为主流的OO语言只支持单一静态的分类。实现多重动态分类的方法也能够用来在不支持继承的环境中重组继承的结构和实现泛化。[281]

对于泛化，我首先描述实现，然后是接口，因为这样可以使人更容易理解接口需要支持的变形。

## 用继承实现

在大多数方法中子类型和子类是同义的，从而提供最可能的实现形式。每个类型的接口被放置在相应的类上，而方法选择由语言进行恰当支持。

因此，如果可能，我们总是推荐这种方法。它的缺点是不支持多重或者动态的分类。

#### 14.2.2 用多重继承组合类实现

图14-5显示一个多重分类的例子，我们能够用多重继承组合类处理它。在这个例子中，除了图中四个对象类型的类以外，我们还可以为优先企业和优先个人用户创建类。通过使用多重继承，类能够巧妙地捕捉所有需要的接口并且使编程环境用通常的方法来处理方法选择。

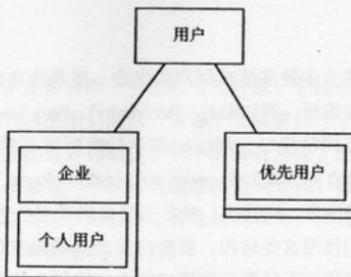


图14-5 一个多重分类的例子

这种方法有两个缺点。第一个缺点是一个有大量分解的对象类型会形成一个难以处理的组合类的集合。4个完全分解，每个有两个类型，需要2个组合类。另一个缺点是这个方法只支持静态分类。

#### 14.2.3 用标志实现

如果你询问一个从来没听说过继承的程序员如何实现记录一些用户是否是一个优先用户的需求，那么答案很可能是“用一个状态标志”。这种过时的方案仍然是有效的。它提供一种支持多重和动态分类的快速方法。标志很容易被随意修改，并且能够为每种划分定义一个标志域。实际上，这是为OO程序中不基于动态分类的状态改变而使用的方案。

使用这种方法的主要困难是我们不能使用语言内部的继承和方法选择。因此，子类型的接口中的所有操作都不得不放在超类型的类上。另外，支持子类型所需的所有域都要被包括在超类型的类中。因而，用户类实现了用户和优先用户两个对象类型。

如果接受对象不是子类型的一个实例，那么使用在子类型上定义的操作很显然是不恰当的，例如请求一个非优先用户的rep，如图14-6所示。如

果我们使用继承，将导致一个错误（Smalltalk中是运行时错误，C++中很可能是编译时错误）。在一个子类型上定义的所有操作必须由一个检查进行控制以确保接受者是属于这个子类型的。如果检查失败，则调用程序退出并产生这种问题的某种信号，通常是一个异常。这暴露出这种方案在C++中的另一个缺点——直到编译的时候才可能发现这些错误。

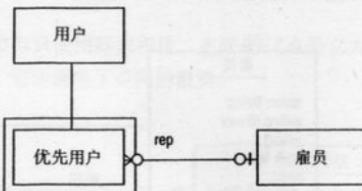


图14-6 优先用户的示例

由于继承已经被抛弃，因此它的伙伴多态性就只是一个记忆。因此，如果一个航运定价操作是多态的，那么方法选择就需要由程序员来实现。这在用户类的内部用一个case语句来实现。一个单独的航运定价操作成为用户接口的组成部分。在这个操作的方法中有一个基于用户类的子类型的逻辑测试，它可能调用内部的私有方法。如果case语句保存在类的内部并且只有一个操作被公布到外部世界，那么就保留了多态性的所有优点。由此，虽然多态性的实现机制完全变了但是其精髓得以保留。

283

这个实现的最后一个缺点是：要为子类型使用的所有数据结构都定义空间。不是这种超类型实例的所有对象都浪费了这个空间。如果在子类型上有大量的关联，这就会导致问题。

#### 2.4 用委托给一个隐藏类来实现

这种方法是用标志来实现子类型化的一个有用变体。这时，为子类型准备一个类，但这个类对除了超类型的类以外的所有类都隐藏。在超类型的类中，我们必须为指向子类型的引用（它可以又作为一个标志）提供一个域。同样，我们也必须把子类型的所有操作转移到超类型的接口上。然而，数据结构仍保持在超类型上。超类型的类上所有来自子类型的类的操作都把调用委托给子类型的类，这个子类型的类拥有实际的方法。

因此，对于图14-7中显示的概念模型，执行官的实例应当对应一个雇员实例加上一个执行官实例，如图14-8所示。除了雇员类，任何构件都看不到执行官对象和它的类。（在C++中，它的所有成员应当是私有的，并且雇员是它的友元。）定义在执行官类型上的giveStock操作应当放置在雇员类上。

当pay被用一个关联的执行官发送到一个雇员对象时，雇员上的pay方法就只需要调用执行官上的pay方法然后返回任何结果。这样一来，系统的其它部分根本不知道子类型化是如何实现的。多态操作的方法选择是用和标志相同的方法（一个内部的case语句）实现的，如果适合就用一个到执行官方法的调用。另一种方案是把所有方法都放置在雇员上，使执行官上除了一个数据结构什么都没有。然而，这样会使执行官无法成为一个自含式的模块。

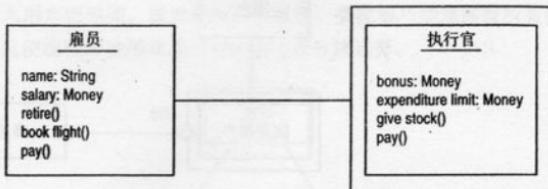


图14-7 雇员和执行官的概念模型

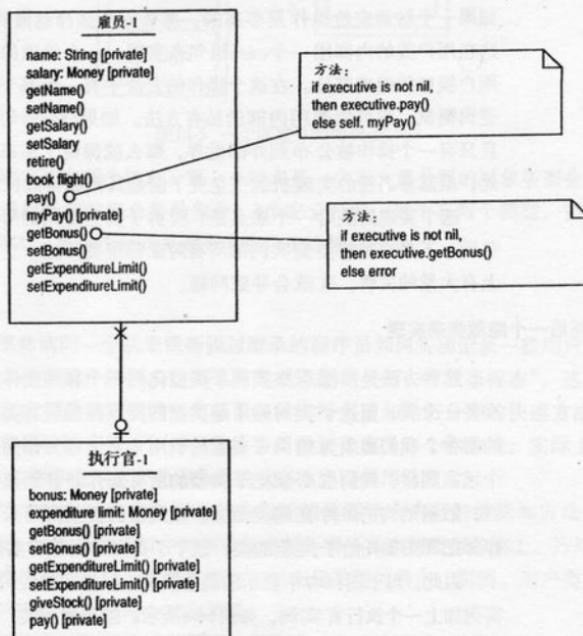


图14-8 图14-7使用委托给一个隐藏类的实现模型

这种方法的逻辑结论是图14-9中显示的状态模式[3]。在这种情况下，总有一个隐藏类存在。这些不同的隐藏类都有一个公共抽象超类，它是自隐藏的。雇员只是把pay委托给它的隐藏类。不管存在的是哪一个子类，它都会正确地响应。这使我们可以增加新的子类型而不用改变雇员类，只要新的子类型不增加到雇员的接口上（一个类似的方法是信封／信惯用法[3]）。

使用一个隐藏类与只使用标志相比，主要的优点是它为复杂的子类型提供更多的模块性。它还避免了空间的浪费。

285

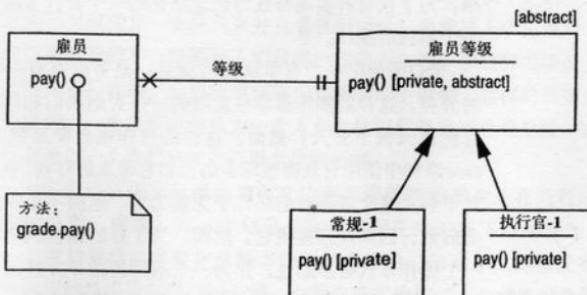


图14-9 用状态模式实现的雇员和执行官

抽象的雇员等级类的一个实例总是存在的。任何依赖状态的行为都被声明为雇员等级上的一个抽象方法并且由子类实现。我使用一个箭头来显示子类关系（来自Rational的统一建模语言（UML）[1]）以便强调子类关系和子类型关系之间的区别。

#### 通过创建一个替换来实现

处理类型改变的一个方法是：用一个子类实现子类型并且在重新分类时去删除旧的对象并用正确类的一个新对象来代替它。这样就允许程序员在仍然提供动态分类的情况下保持继承和方法选择的优点。

执行这种方法的过程是：在新的类中创建对象，把旧对象的所有公共信息复制到新对象，把所有指向旧对象的引用变为指向新对象，并最后删除旧对象。

在很多环境中最大的问题是找到所有指向旧对象的引用并把它们转移到新对象。如果没有内存管理这几乎是不可能的。任何没有捕捉到的指针都会变成悬挂指针并导致难以调试的崩溃。因此，我们不推荐将这种方法

用于C++，除非使用某个能够可靠地找到所有引用的内存管理方案。有内存管理的语言可以很容易地找到这些引用；Smalltalk提供一个方法（名字为become）进行引用的置换。

如果所有的引用都能够被找到和改变，那么这种方法似乎是可能的。它的缺点是复制公共信息和寻找并改变引用所消耗的时间。时间的总量随环境变化很大，这最终决定了这个方法是否合适。

286

#### 14.2.6 泛化的接口

所有五种实现都工作得很好，而且都在面向对象编程中得到正规的使用。为了使每种实现都成为概念泛化的一个替代方案，我们需要使它们有一个单独的接口。

OO编程中一个有争议的问题是：是否应当有一个操作返回一个对象的分类。这样的操作通常是重要的——否则我们如何能够得到一组人并过滤使其只留下女人？然而，这样的操作也会带来危险，即程序员在一个case语句中使用它从而破坏多态性和它带来的好处。在OO编程的结构内部，对于解决这个难题看起来几乎无能为力。返回一个对象的分类的操作总是必需的，因而应当提供它。然而，为了好的编程风格，我们不应当用这样一个操作来代替多态性。作为一个通用的指导方针，应当将对分类信息的请求作为一个查询之内纯粹信息收集的一部分或者是为了接口的显示。

为了找出一个对象的分类，有一些现存的惯例。Smalltalk和C++的程序员都使用名为isStateName的操作来确定一个对象是否处于某个确定的状态。Smalltalk有一个消息isKindOf: aClass可以确定类的成员资格。C++在运行时不保存类信息（虽然这会被即将发布的标准改变）。然而，只要需要，有时也提供能够有效给出这种信息的操作。

有两个主要的命名方案可供使用。第一个方案是使用命名形式isType Name。第二个方案是提供一个带参数的操作，如hasType (TypeName)。第一个方案是使用标志和隐藏类的一般约定。它在目前的情况下工作得很好，但是适用于子类型化时有一个问题。如果我们想要给一个现存的类增加一个新的子类，则我们需要给超类和子类一起增加isTypeName操作。否则在超类上调用isTypeName就会产生错误。hasType方案的可扩展性更好，因为能够增加一个子类而不用对超类进行任何改变。要记住：在任何情况下我们想要的都是类型信息，而不是类信息。

对于类型变化，不存在典型的命名标准。像makeTypeName或classify AsTypeName这样的名字都是合理的（我更喜欢前者）。这些操作应当负责从任何不相交的类型中撤消分类。因此，一个完整的划分只需要拥有

和划分中的类型一样多的修改者。不完整的划分需要某种办法达到不完整的状态。要达到该目的可以通过为划分中的每个类型提供`declassifyAsType Name`方法，或者通过提供一个单独的`declassifyInPartitionName`操作。要注意：那些不可能是动态的划分不应该拥有这些修改者。

[287]

当使用这些修改者时，关联所暗示的问题与在创建和删除之下讨论的问题相似。因此，在一个分类程序中强制映射需要参数，并且撤销分类能够导致与单个和多重删除类似的选择。

并不是所有的子类型都是动态的，但是关于是否使一个划分成为动态的决定依赖于模型是概念模型还是接口模型。在概念建模中，把一个划分标志为不可改变是一个强约束并且通常很罕见。虽然有人争论说对大多数的应用而言我们不会想要把人们从男性改变为女性，但这种类型改变在概念上不是不可能的。即使在现在的先进医学产生以前，这样的改变也可能是需要的。一个公司可能认为一个人是女性但后来发现他是男性。这样的发现应通过一个类型改变进行概念化的处理。

大多数语言很少处理类型改变的事实促使我们减少正在进行的类型改变的总数。因此，当一个划分只在非常罕见的情况下才是动态的时，在一个规约模型中声明它是静态的就是合理的。这些罕见的情况通常是由标识的错误或者用户的失误，由用户显式地创建一个替换对象就能够处理它们。这是一个纯粹的概念模型和一个基于概念的规约模型之间区别的又一个来源。

#### 14.2.7 实现hasType操作

现在需要简单说一说类型访问者的实现。系统中的每个类都将需要一个`hasType`操作。这个方法将对所有用类实现的类型进行参数检查。如果已经使用标志，那么就要检查标志以便测试这个类型。即使不存在标志，类基本上也总会实现一个特殊的类型并且必须检查这个类型。如果这些测试的任何一个都返回“真”(True)，则返回“真”。可是如果没有任何类型的类型匹配，那么就调用超类上的方法并返回其结果。如果没有超类型，就返回“假”(False)。因此实际上，一个发送到层次底端的消息将沿着层次上升直到它碰到一个匹配，或者运行完毕到达顶端并返回“假”。这个机制使扩展类型层次变得简单，因为只有实现了类型的类才需要为该类型进行检查。

[288]

### 14.3 对象创建

在创建新对象所需要的机制中，既有用一个类直接实现的机制又有间

接实现的机制。

#### 14.3.1 创建的接口

每个类必须有一种方法来创建它所实现的类型的实例。创建意味着不仅要组织一个新的实例对象，而且要满足对该对象的各种约束以使其成为一个合法对象。

所有强制的关联必须在创建操作中得到填充（一个完整的创建方法[1]）。这意味着创建操作必须有针对每一个强制操作的参数。类似地，在完全的划分中用类实现的任何子类型必须通过参数来选择。强制情况和不变的关联或者非强制的划分也应当通过参数来选择。

由于实现环境中的其它假设，有时很难使用默认对象创建机制来做到这一点。工厂方法[3]应当在这种环境中使用。

我们也应该允许在创建参数中包括可选择的、易变的特征。但是，较好的方法是首先创建对象然后向它发送必需的消息来设置这些特征。

#### 14.3.2 创建的实现

所有面向对象语言都有它们自己的创建新对象的约定。通常，这些约定提供存储的分配和域的初始化。然而，初始化程序并不总是通过参数来设置强制特征的合适地方。

在Smalltalk中，通常的习惯用法是使每个类支持一个能够带参数的创建消息（通常称为new）。在创建的过程中，它总是被安排给新对象以发送一个没有参数的初始化消息。这个初始化对于把多重映射的实例变量设置到一个新的集合是有用的，但不能支持初始化关联，因为它没有参数。最好是使用Kent Beck的创建参数方法模式[1]，它通过一个特殊的方法来设置这些初始参数。

C++为初始化提供一个构造函数。它能够做很多事情，但有时关于构造函数的语义会出现问题。通常最好只在另一个创建操作的内部使用构造函数；对这种情况，“四人帮”的创建模式[4]特别有用。

289

#### 14.4 对象析构

既然对象会产生，那么它们就可能会消亡。不是所有的对象都能够被析构，一些对象必须永远存在（例如医疗记录）。即使这样，它们也可能在一个系统中被析构而保存到其它地方。

析构对象的最大问题是忍受结果。例如，从图14-10中删除订单的一个实例，如果有任何订单项连接到它，就会产生问题。这些订单项必须有

一个订单（强制关联），所以如果我们只是删除订单，那么订单项就会违背它们的约束。

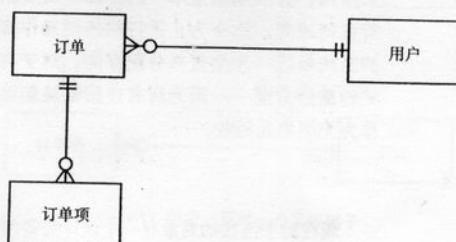


图14-10 用户和订单的例子

这个问题有两个解决方法。第一个方法是单一删除——更宽容的、更温和的方法。如果删除导致任何剩余的对象违背它们的约束，那析构就会失败。另一方面是多重（或者级联）删除——猛烈的、恶意的方法。如果这个删除使一个订单项违背它的约束，则这个对象也会被删除。如果任何事物有一个强制映射连接到这个对象，那么那些依赖对象就一起被删除——在整个信息库中产生连锁反应。

在实际中，删除能够有不同程度的级联。析构操作对于一些映射可以是多重的，而对其他的映射则是单一的。这是完全允许的，但是它必须保证析构要么全有要么全无。

这些问题增加了在没有内存管理的环境（像C++）中对引用的考虑。单一和多重删除应当确保对象不破坏它们的基数约束，并且存储管理要避免悬挂指针。

#### 14.4.1 析构的接口

不同的面向对象环境拥有它们自己的析构方法。所有能够析构的对象应当有一个完全单独的析构操作。这是一个程序员所有的需要，但是它把安排事物析构顺序的责任推到用户身上。一些强烈的删除能够和一个完全单独的析构一起被提供。然而，必须弄清楚析构对于哪些映射是多重的。[290]

#### 14.4.2 析构的实现

就是在析构中，内存管理的存在使其自身的作用充分体现出来。它几乎没有使析构方法本身有什么不同，但是确实影响了错误的结果。

在两种情况下都很重要的是：将要被析构的对象要使它们到有关联

的对象的所有连接都断开（在两个方向上）。必须进行必要的检查看看关联的对象是否会违背它的约束。如果删除是多重的，则关联的对象也要被析构。如果删除是单一的，那么就要禁止整个删除并且不对信息库进行任何改变。迄今为止进行的任何操作都要被回滚。对于没有内存管理的系统最后一步是重新分配存储。对于有内存管理的系统，不会进行显式的重新分配——因为所有连接都被删除，所以对象会孤立而死并且被作为无用单元回收。

## 14.5 入口点

现在对于连接的对象有一个设计完善的结构。从任何对象来使用类型模型都很容易决定如何遍历到另一个对象。然而，仍然存在一个重要的问题：我们在最开始的时候是如何进入对象结构的呢？对于那些使用传统的、尤其是关系数据库的人，这个问题可能显得奇怪，因为这些数据库的入口点是记录类型。获得数据包括从记录类型开始和选择单个的记录。然而，从一个类型的所有实例的列表开始并不总是最有效的方法。特别是面向对象系统能够提供不同的访问形式，这些形式更有效并能够提供其它有用的能力。

我们不需要所有类型的所有实例的一个列表。考虑图14-11中的例子。由于订单项的所有实例都连接到订单的一个实例，因此就不需要保持一个从订单项类型到其所有实例的引用。如果我们认为很少有任何人请求所有的订单项，不管订单或者产品，那么我们就能够忽略这个引用。如果那种不太可能的情况发生，即某个人确实想要一个所有订单项的列表，那么我们提供这个列表的方法是先得到订单的所有实例的一个列表然后通过映射遍历到订单项。因此，我们能够节省用于保存到订单项的所有实例的所有引用所需要的存储，代价是一层的间接引用，当然前提是可能需要订单项的所有实例。这完全是一个实现权衡。在一个关系数据库中，不需要考虑这种权衡，因为数据库使用固定的表。

同样的问题能够扩展到订单。我们可能认为：如果一个人想要通过键入一个订单号来选择一个订单，则需要订单的所有实例。既然订单号通常是一个字符串，那么经常就不保存从字符串到订单的引用而是需要订单的所有实例。然而，我们可以争论说：实际上一旦找到用户，则订单总是会被访问。至于是否保存指针又是一个实现问题。

这个问题不能扩展到用户，因为用户缺少任何强制联系。因此对于一个用户，不和任何其它对象发生联系是可能的。所以关于用户的所有实例的一个列表是必要的，这样可以确保能够找到这样的一个用户。这个保存列表的必要性就是使用户成为一个人口点的原因。

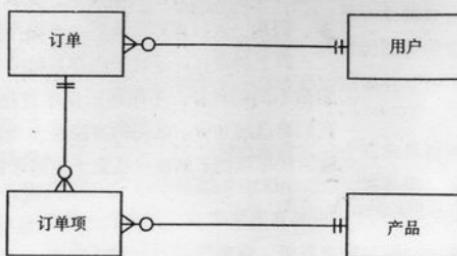


图14-11 用户、订单、产品的例子

要注意：决定哪个对象类型应当是入口点完全是一个概念上的问题。没有强制联系的对象类型必须是入口点。那些有强制联系的对象类型能够保存实例的一个列表，但是这根本不能使它们成为概念的人口点。

#### 14.5.1 查找对象的接口

对于所有类型来说，有一个返回当前类型的所有实例的操作是有用的。这样一个操作对于一个方向上的指针对于反向遍历查找指向某个对象的其它对象是很重要的。

提供根据某种标准查找一个实例的操作通常都是有用的。一个例子是 `findCustomer (customerNumber)`。虽然很难给出使用这样一个操作的通用规则，但是一般来说，最自然的方法就是使用遍历。因此，与请求“查找用户是ABC的所有订单”相比，请求“用户 ABC的所有订单”在概念上更简单。这会导致由查询的遍历表达所引发的优化问题，但是这能够在用户的访问者内部得到解决。

当查找被应用于基础类型时，这个选择就不再适用，而一个通用的查找程序是一个更好的选择。然而，即使这样也应当尽可能用通用的方法来进行。最简单的方法是请求一个类的所有实例，然后使用返回集合上的内置选择操作。对于有很多实例的类，这种方法工作得不好。下一个方法是提供一个把任何布尔操作作为一个参数的选择操作。这在类的接口上只用一个操作就实现了最大的灵活性。然而，这在一些语言中比其它语言要困难得多。只有当这些方法都用尽了并且使用一个更通用的方法太昂贵时，我们才可以使用带有特殊参数的查找。我们必须注意不要使一个类的接口膨胀。

要注意的是：这些查找实例的操作对于非入口点和对于入口点同样正确。实际上，实例访问者应当采用相同的模式。

入口点需要一个额外的操作使一个对象装配在结构中。只是创建一个

对象并不能把它装配在结构中，尤其如果它和结构中的任何对象都没有联系。因此，入口点对象需要一个操作把它们安插在结构中。

以上的接口评论对于内存系统是正确的。当使用数据库时会出现一些稍微不同的特征。不同的数据库管理系统（OODBMS或者关系接口）有它们自己的约定。实用的方法是使用这些约定时带有以下的附带条件，即应当使尽可能多的接口独立于数据库管理系统的细节。

#### 14.5.2 查找操作的实现

实现一个人口点的通常方法是通过某个聚合类。这个聚合可以是一个特殊的单独类（例如用户列表）或是类中的一个静态域。向一个类型请求它的实例意味着返回聚合的对象。因为存在多值关联，所以重要的一点是：聚合应当是不可改变的除非通过入口点的接口。

一个非入口点通常也有一个返回所有实例的操作。这能够通过从一个入口点开始的遍历来实现。选择和查找工作使用类似的方法。

#### 14.5.3 使用类或者登记表对象

人口点的接口和实现都可以用类或者登记表对象来完成。入口点的一个基于类的实现导致每个人口点类都保存它的实例的一个聚合作为一个类或者静态变量。另一种方法是拥有一个单独的登记表对象，它为每个人口点类保存一个聚合。使用登记表方法的主要优点是它允许存在独立的登记表，可能针对不同的上下文。因此，如果两个临床部门想要维护疾病的同实例，这就能够通过为每个临床部门使用一个单独的登记表来实现。

在接口中，区别存在于程序员把查找消息发送给类还是发送给一个登记表对象。使用一个登记表就从每个类上除去了这个职责，但是登记表对于每个人口点类都需要至少一个查找操作。如果查找操作也被用于非入口点，那么登记表对于每个类都需要至少一个查找操作。当程序员需要理解和在不同的上下文之间切换时使用一个登记表是有用的。如果只使用一个单独的上下文，则它就能够被设置成全局的并且基于类的操作能够委托给适当的登记表。

### 14.6 实现约束

类型模型帮助定义一个类型必须满足的约束。基数和划分都意味着约束。越复杂的情况需要越复杂的类型。本书中使用的短的和长的语义声明都常常意味着更复杂的约束。

约束一般不影响编程语言中类的显式接口。Eiffel是一个例外，其中约束定义了类不变式。对于没有Eiffel特征的语言，所有的修改者都必须考虑约束。程序员编写修改者时必须确保应用修改者使对象处于一个不违背它的任何约束的状态。

通常使用的方法是实现一个显式的访问者来确定一个对象是否适合它的约束。应当为所有类提供一个名称类似checkInvariant的操作，如果发生了错误，这个操作就产生一个例外，而如果所有都正常，则什么都不做。它能够被用在不同的地方为系统进行健康检查，包括在调试期间作为一个后置条件检查的一部分和在操作中作为系统健全检查的一部分——这对数据库系统尤其重要。

Smalltalk和C++不像Eiffel一样具有显式的约束和断言的能力。我们可以用一种弱的但是相当有效的方法来建立它们。在Smalltalk中你能够建立一个把一个块作为参数的操作（名称如同require: aBlock）。这个方法能够被编写在类对象中以便执行那个块并且如果它返回错误就抛出一个异常。然后需要的方法就能够用来进行前置条件检查、不变式检查和一些后置条件检查。C++中有一个名为断言的宏能够用来实现相同的目的。

[294]

## 4.7 其它技术的设计模板

本书主要讨论类型模型。因此本章中的设计模板是从类型模型进行的转换。类似的原则能够应用于其它技术。虽然这样的一个直接映射似乎并不正确，但它能够为事件图[6]提供设计模板模式。在最近的几年中关于不同种类状态模型的设计模板已经进行了大量的讨论，尽管我们仍然在等待对于这个问题的一致结论。交互图十分接近于实现，这在它们与代码的关系中相当明显。

在最近的几年中，已经有人数较少但是重要的一群开发者强调这种转换方法。Shlaer和Mellor已经站在这个群体的最前面[8]。我希望随着时间的推移这个题目会得到更多的关注并且我们将会看到更多的模式和一些完整的设计模板。我怀疑模板的一个完全集合更有可能作为一个商业工具（很可能与CASE工具相连接）或者一个内部的成就而被生产出来。我希望这些模板的模式将会成为文献的一个常规部分。

## 参考文献

- Beck, K. *Smalltalk Best Practice Patterns Volume 1: Coding*. Englewood Cliffs, NJ: Prentice-Hall, in press.
- Booch, G., and J. Rumbaugh. *Unified Method for Object-Oriented Development*. Rational Software Corporation, Version 0.8, 1995.

3. Coplien, J.O. *Advanced C++ Programming Styles and Idioms*. Reading, MA: Addison-Wesley, 1992.
4. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
5. Lewis, S. *The Art and Science of Smalltalk*. Hemel Hempstead, UK: Prentice-Hall International, 1995.
6. Martin, J. and J.J. Odell. *Object-Oriented Methods: Pragmatic Considerations*. Englewood Cliffs, NJ: Prentice-Hall, 1996.
7. Musser, D.R., and A. Saini. *STL Tutorial and Reference Guide*. Reading, MA: Addison-Wesley, 1996.
8. Shlaer, S., and S.J. Mellor. "A deeper look at the transition from analysis to design." *Journal of Object-Oriented Programming*, 5, 9 (1993), pp. 16-21.

# 关联模式

---

在分析和设计方法中，关联是一个通用构造。通常，一个关联意味着一个相应特殊的情况会不断重现。可以引进一个专门的符号，但是对这种情况建模而不用这个符号也是可能的。思考这个问题的一个有用方法是把这种情况看作一个模式。这个关联模式能够以一个基础形式进行描述，或者能够引进一个新的符号作为一个简写。这两者在意义上是相等的。

本章关注三个这样的情况。当你想要把一个关联看作一个类型，通常是给它一些特征时，一个关联类型（参见15.1节）就出现了。一个带键值的映射（参见15.2节）是用来为一个映射提供一个类似查找表或者字典的行为。这些模式中的每个模式都使用大量带有额外符号的方法。重要的是理解在符号之后的模式。一个方法可能不支持一个额外的符号，所以知道没有这个符号时如何工作是很重要的。尤其当你已经习惯于一个支持某个符号的方法而要转变到一个不支持该符号的方法时，或者当你在两个方法之间转变而一个方法不支持一个符号时，更显示其重要性。

即使你的方法为一个关联模式使用一个符号，理解符号如何联系到更简单的思想也是重要的。如果情况是罕见的，那么通常更好的方法是不引入一个额外的符号来记忆，而是使用基础形式。

第三个关联模式是历史映射（参见15.3节）。我们能够用历史映射来记录一个映射的值的改变历史（例如一个雇员的工资历史）。这是我所知道的任何方法中的专门符号都不支持的。然而，这对很多信息系统来说都是一个必需的模式。当需要一个历史映射时，引进一个符号作为关联模式的一个简写是很有价值的。如果不仅领域改变而且我们关于领域的知识在不同的地方也发生改变时，就会出现特殊的复杂性；这就产生了二维历史（参见15.3中的子小节）。

有几个因素影响使用符号或者基础形式之间的选择。在概念上，主要的权衡在于符号提供的简明性和符号带来的额外记忆之间。在一个规约模型中，一个符号意味着软件中的一个不同接口。这个接口使用起来很可能比从基础形式转变而来的接口更方便。然而，我们总是能够把操作添加到

规约模型上提供更方便的接口。这样，在规约模型上添加了额外的显式操作，但是避免了额外的符号。

使用符号还是基础形式是一个选择的问题。在本章中，我会指出我的偏爱（我要强调的是这些偏爱与客户的愿望相比只能放在第二位）。作为一名顾问，使客户的生活更便利是我的工作。

关联模式在元层起作用：它们是用来描述建模语言的模式，而不是用来描述模型本身的模式。我用术语元模型模式来概括地描述这类模式。其它的元模型模式可以用来描述泛化、状态模型或者任何其它建模技术中的元层概念。

## 15.1 关联类型

当我们想要给一个关系增加一个属性时，就会出现一种普通的建模情况。例如，一个早期的模型表明一个人被一个公司雇佣，如图15-1所示。后来的工作显示我们应当记录雇员开始受雇的日期，并且它必须位于这个关系上。我们能够使用一个符号（例如Rumbaugh的领形符号[2]）把开始日期属性添加到关系上，如图15-2所示。

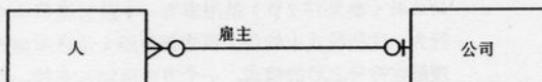


图15-1 人和公司之间的简单关系

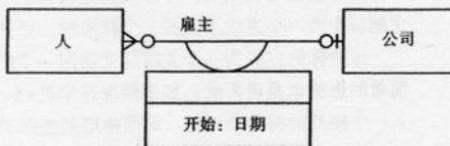


图15-2 在图15-1中添加一个开始日期属性

本图使用了Rumbaugh的领形符号。

如果一个建模方法不支持用这种方法把一个属性添加到一个关系上，那么还有许多替代方案。在我们的例子中，一个替代方案是把开始日期添加到人上。因为依据定义一个人只能有一个公司，所以不存在多义性的危险。我们可能会认为开始日期属性确实是关系的一部分，但是这只能被理解为语义上的吹毛求疵。一个更合理的反对理由是：除非存在一个雇主否则开始日期不应当有数值。这可以用一个规则来解决，虽然这通常是一个不够完美的解决方法，特别是因为大多数方法都不能很好地支持这种规则。

当两个映射都是多值映射时，这个方法就不能在关系上使用，如图15-3所示。既然一个人的每个技能都有一个不同的资格，那么就不可能把数字放在人上。

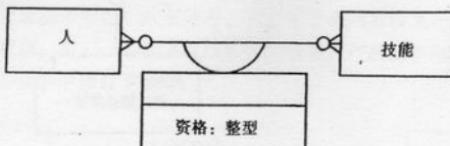


图15-3 两个映射都是多值映射的关系

在不支持关联类型的方法中，我们可以引进一个额外的类型，如图15-4所示（注意基数是如何从图15-1中转变过来的）。这样非常好地处理了这种情况。新的类型可能稍微有些不自然、但是所有的模型都包含一定数量的人工成分，因为它们描述一个真实情况所用的形式化程度比自然语言中存在的形式化程度要大得多。这两个模型之间最重要的区别之一在于接口含意。在图15-2中，人有一个返回关联的公司的getEmployer操作。  
图15-4中的模型有一个不同的接口，它返回雇佣关系对象。雇佣关系对象需要一个额外的消息来得到公司，所以我们需要把原来的关联变成一个派生的关联，如图15-5所示。

299

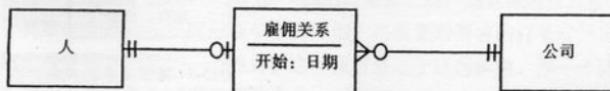


图15-4 增加一个雇佣关系类型作为开始日期的拥有者

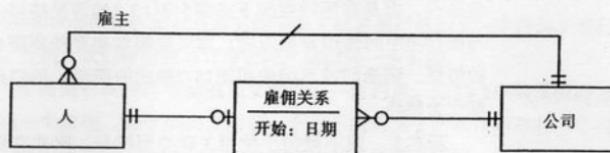


图15-5 用一个派生的映射重建雇主映射

通过考虑图15-3中显示的多对多关联，我们能够考虑得更巧妙一点。图15-6同样引进一个新的类型。如果只是增加资格类型，那么在第一个检查上会工作得很好，因为它允许一个人有很多资格，并且因此可以有多种技能，每个技能都有一个资格值。问题是这个模型允许的太多，因为它也允许相同的技能对应多个资格。为了消除这个问题，我们需要建立针对资格的附加的

惟一性规则，以便指明每个资格必须有一个惟一的人和技能的组合。

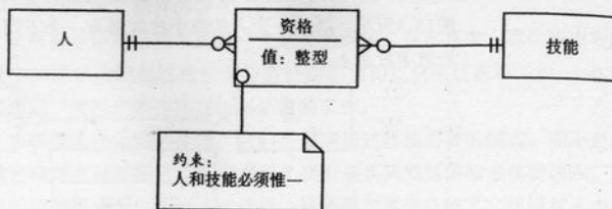


图15-6 用一个新类型来处理图15-3

这个问题通常不被使用关联类型符号的建模者所注意。图15-7是这个符号的另一个典型应用，这里关系处理这样的一个理解，即一个人可以是很多公司的雇员并且这些雇佣关系中的一部分可能已经结束，这样以来我们就有了雇佣关系的一个历史。一个人对同一家公司有两个时期的工作经历是非常可能的。因此我们不能增加一个图15-6中所显示的那种风格的约束。一般来说，问题是我不知道是否要将一个关联类型解释为拥有约束。

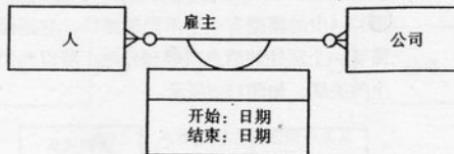


图15-7 雇佣关系的关联类型

实际上，建模者使用关联类型符号时同时应用这两种解释。这并不是它本身的缺点，但是他们应当说清楚他们所指的是哪一个。使用图15-7是合理的，但是在那种情况下必须为图15-3的情况使用一个规则，这个规则与图15-6中的规则是类似的。如果建模者想要使用图15-3的例子作为通用的解释，则他们就不能使用图15-7形式的模型；他们必须使用一个新的类型作为替换。

基本上，我不倾向于使用关联类型符号。除非它们包括一个明确的规则，例如惟一性的规则，否则我不认为它们为额外的符号增加了很多价值。惟一性是有用的，但是它很少被正确使用，以至于我宁愿使用一个额外的类型然后增加惟一性规则以使其明确。

## 15.2 带键值的映射

带键值的映射描述一种技术，它在分析中反映为使用字典（带索引的

查找表，也称为maps[1]或者关联数组）实现关系的技术。在图15-8和图15-9中显示使用它的例子。我们主要考虑的是记录某种特殊的产品在一个特殊的订单上有多少。这个例子的标准数据模型显示在图15-8中。图15-9中显示的模型使用带键值的映射符号，它集中于询问和改变一个订单拥有某种产品的数目。图15-8对此进行结算的方法是：产品能够回答它被订单预定并在每个订单中预订了多少。

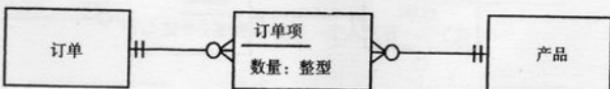


图15-8 一个标准的订单、订单项的模型

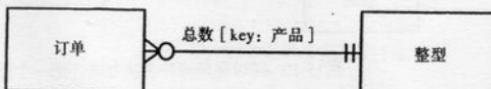


图15-9 使用一个字典来建模图15-8

这些模型解释的一个重要部分是它们如何影响类型的接口。图15-8的模型意味着在订单和产品上有一个`getLineItems`接口。图15-9的模型意味着在订单上有一个`getAmount (product)`的接口。对产品意味着没有接口。  
[301]  
为了找出一个产品在不同的订单中的使用，将需要对所有的订单实例询问它们是否有这个产品的一个数目，这样做有点过于迂回曲折。另一个区别在于询问一个订单上现存有哪些产品。对于图15-8，这只需要询问一个订单找到它的所有订单项然后询问每个订单项的产品。对于图15-9，这需要向一个订单询问数目的字典，然后请求它的键；订单将必须提供`getAmounts`操作以便允许对它的字典（或者更严格的是一个拷贝）进行访问。否则，我们将需要对订单访问产品的每一个实例。

带键值的映射符号能够用来处理惟一性约束。图15-8中的模型通常都伴随一个规则，这个规则说明在一个订单内对某个产品只能存在一个订单项。我们不会想要为30个小配件设置一个订单项而为同一个订单中20个小配件设置另一个单独的订单项。一个更好的建议是针对50个小配件只有一个订单项。对于图15-8，这需要一个规则，但在图15-9中这很明显，因为一个订单对一个产品只能有一个数量。

我们需要考虑的是：如果向一个订单询问一个不在它上面的产品的数目，这个订单应当如何响应。在这个例子中，一种看起来合理的方法是返回0，并使带键值的映射成为强制的。在其它的情况下，我们可能想要一

个无效的 (null) 返回, 这将使那个映射成为可选的。

如果两种表达方法都是有价值的, 那么就意味着可以同时使用它们两个。可以使用一个规则或者一个派生标记注释出冗余, 如图15-10所示。同时使用两种表达方法支持这样的事实, 即图15-8的方法在一般情况下更具有适应性而图15-9的方法增加一个非常有用简写行为, 并且使惟一性很明显。

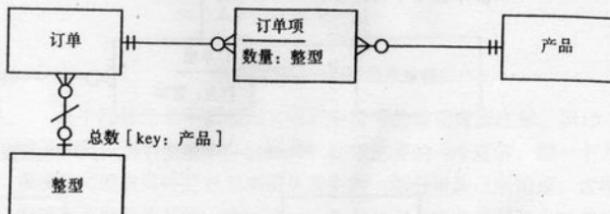


图15-10 同时使用两种表达方法, 把一个标记为派生的

我发现带键值的映射符号是一个非常有用的构造。使用它还是使用一个额外的类型依赖于情况和我所要强调的。虽然没有它我肯定能够生存, 但是我经常发现它是一个便利的构造。然而, 注意不要过度使用它。通常, 额外的类型对于额外的信息和行为是重要的。在图15-8中, 我们可以容易地为订单项增加一个价钱, 若使用图15-9这将是难以使用的。自然地, 图15-10中“鱼和熊掌兼得”的答案是一个通常的选择。

### 15.3 历史映射

对象不仅代表真实世界中存在的对象; 它们还经常代表对那些曾经存在但已经消失的对象的记忆。使用对象来代表记忆是完全可以接受的——关于存在的记忆对人们来说通常和存在本身一样真实——但重要的是能够说明两者的区别。考虑记录一个人的工资的问题。在任何一个时刻个人都有一个单独的工资, 如图15-11所示。然而, 随着时间流逝工资可能会改变。这本质上并不能阻止图15-11作为一个模型, 除非我们需要记录工资的历史。如果所有我们想要的就是记录过去的工资, 那么图15-12将实现这个功能, 前提是给工资的修改者增加把旧的工资添加到一个旧工资列表中的能力。通过使用一个列表, 我们不仅能够记录以前的工资而且能够保存它们所应用的顺序。

图15-12可能对很多情况都适用, 但是它无法帮助我们回答这个问题: “John Smith在1997年1月2日的工资是多少?”为了回答这个问题我们需要图

15-13中所建议的更复杂的方法。这个模型赋予我们记录工资和它们的历史的能力。然而我们需要一个额外的规则：一个人的工资不能有重叠的时间段。这个规则通常是隐含假定的，不经常被显式指出——并因此常被遗忘。

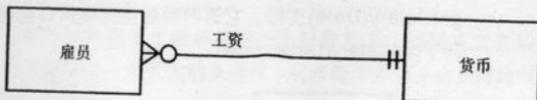


图15-11 在任何时间点一个人只有一份工资

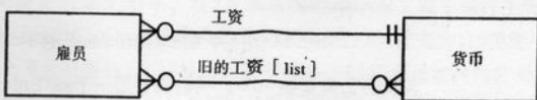


图15-12 一个记录过去工资的模型

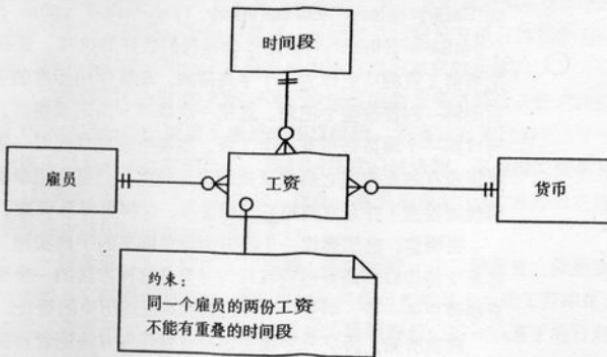


图15-13 工资历史的一个完全记录

图15-13中显示的模型提供了我们所需要的能力，但是它相当笨拙。由于没有查看隐含的规则，一个雇员每次只能有一个工资的要点被丢失。两个类型之间的一个关联现在变成四个类型和三个关联。这会极大增加一个图的复杂性，尤其如果有很多这样的历史关系。为此所建议的接口也是相当笨拙的。上一段中问题的答案包括：询问John Smith的所有工资，然后选择时间段包括1997年1月2日的那个。

我经常使用图15-14中显示的模型，它合并图15-13中方法的灵活性和图15-11中快照图表的经济性。所有细节都隐藏在小而重要的历史关键字的后面。我已经引入一个新的符号，只要我适当定义，它是完全可容许的。

我将放弃一个数学的定义，取而代之的是指出用关键字定义的接口。图15-11意味着一个getSalary()的访问者返回工资的数值，而由一个setSalary(Money)的修改者改变它。图15-14意味着一个不同的接口：访问者getSalary()仍然存在但是这次它返回工资映射的当前数值。这由getSalary(Date)支持，它返回映射在所提供的日期的数值。getSalary()等同于getSalary(Date:: now)。

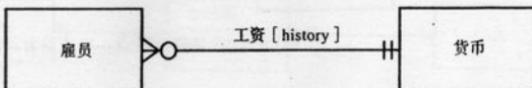


图15-14 用一个更简单的符号来描述图15-13的能力

304 更新要稍微复杂一些。我们能够用一个setSalary(Money, Date)的操作把一个开始于某个特殊日期的新的工资添加到历史中。这对于附加改变是一个好接口，但如果旧的记录也需要修改，它就不够有效。一个setSalaryHistory(Dictionary(key: TimePeriod, value: Money))操作和一个getSalaryHistory()的操作会是我们最好的选择。这样一来，客户能够把当前工资的历史作为一个字典得到，能够使用标准的字典操作，而且随后能够一起修改整个记录。这比一次修改一条记录要好，因为规则是在任何日期一个雇员必须有一个工资。如果更改是每次对一条记录进行，那么将很难在每次改变之后保证规则的正确性。把整个记录取出，改变它（不用规则检查）并且立刻把它全部替换，这样更容易管理。

很明显，这里建议一个带时间段关键字的字典实现。这样一个实现很容易支持接口所需要的所有行为并且是这种方法的一个简单应用。我们甚至能够更进一步，引进一个专门的类来处理历史的聚合。

就我所知，历史符号现在还没有被任何方法论者所提出。它是很有价值的，因为它简化一种既普遍又棘手的情况。理想的解决方法是拥有一个有完全“时间旅行”能力的对象系统。这样一个系统不是完全牵强的，它的到来将消除对历史的任何专门处理的需要。

本节也是一个通用要点的一个特殊情况。在建模中，你可能遇到一个对模型来说既普通又棘手的重复情况。不要害怕引入一个新符号来简化这种情况，但是你必须正确定义它。要考虑的关键权衡是在新结构带来的简明性和符号需要的额外记忆之间。一个好的符号是一个折衷，提供优雅的表示，但不是一个很复杂的符号。这个权衡对所有的项目是不同的，因此不要害怕对这些事情做出你自己的决定。

**建模原则：**如果你遇到一个难以建模的重复情况，可以定义一个符号。

然而，我们只有当造成的简化性超过了记忆额外符号的困难时才会定义一个符号。

## 二维历史

上面的讨论集中于能够检索一个对象的某个属性在过去某一点的数值。很多系统有一个更大的复杂性，它来源于系统不会及时地收到改变信息这样一个事实。

想像我们有一个工资单系统，其中记录一个雇员有一个开始于1月1日的100美元/天的工资率。在2月25日我们用这个工资率运行工资单系统。在3月15日我们得知，该雇员的工资率从2月15日起变为110美元/天。当询问这个雇员对象在2月25日的工资率是多少时它应当如何回答呢？这个问题有两个答案：这个雇员在那时认为工资率是多少和这个雇员现在认为工资率是多少。这两个工资率都很重要。如果我们需要回顾2月25日工资单的运行以便看到那些数字是如何计算的，那么我们就需要看到旧的数字。如果我们要处理一个新的授权，可能为了以前没有报告的几个小时的加班，那么我们就需要这个工资率成为我们现在对它的理解。

其实按照生活中的实际情况，事情可能会变得更糟糕。假定我们进行了相关的调整并且制定了最近的加班报酬，所有这些都已经在3月26日的一次工资单的运行中被执行。在4月4日我们被告知，雇员的工资率又被变为112美元/天，从2月21日生效。现在，雇员对象对于在2月25日它的工资率是多少就能够给出三个答案了！

一般来说，为了处理这种问题，我们需要一个二维历史。依照我们在过去的其它时间点的知识，我们询问雇员在过去的某个点的工资率是多少。因此这需要两个日期：工资率可适用的日期和我们的知识基于的日期，如表15-1所示。

表15-1 例子的二维工资率

可适用的日期	知识基于的日期	结果
2月25日	2月25日	100美元/天
2月25日	3月26日	110美元/天
2月25日	4月26日	112美元/天

一维的例子实际上不得不在以下两者之间进行选择：把可适用的日期和知识的日期等同看待，或者总是把知识的日期看作“现在”。

给历史增加完全的二维能力肯定会增加大量的复杂性，并且并不总是值得的。重要的是，要观察为什么这些不同的比率可能是需要的。在这个

例子中，除了我们现在的关于过去的知识之外我们需要知道任何其它事情的惟一原因可能是要解释调整和把调整记入到以前的工资运行中。处理这种情况的另一个方法是把关于如何制造一个工资单计算的所有信息嵌入到这个工资单计算的结果中。如果这个信息只会被人检查而不会被执行，那么可以把它作为一个文字的属性。计算调整能够通过指向计算结果的引用实现——使用的比率不是必需的。即使需要比率，制作一个拷贝也被认为是更安全的。如果所有这些都就绪，那么只需要一个一维历史就能够执行有追溯效力的授权（例如迟报的两个小时的加班）。

二维历史也影响放置在事件上的时间点。除非我们确信我们总是在事件一发生就知道，否则关于任何事件我们需要两个时间点：当事件发生时的时间点和当我们的系统注意到这个事件的时间点。（这个的例子包括在6.1节讨论的条目的两个时间点，和在3.8节讨论的双时间记录模式。）

## 参考文献

1. Musser, D.R., and A. Saini. *STL Tutorial and Reference Guide*. Reading, MA: Addison-Wesley, 1996.
2. Rumbaugh, J. "OMT: The object model." *Journal of Object-Oriented Programming*, 7, 8 (1995), pp. 21-27.

## 第16章

# 后记

---

你认为这本书怎么样？你发现这些模式有用并且对它们感兴趣么？虽然我希望这样，但我也希望你感到不满足——还有更多的要说，更多的要理解。本节讨论下一步将如何去做。

你能做的一件事情是试验这些模式。阅读一本关于模式的书实际上只是让你感觉到有哪些模式存在。当我阅读“四人帮”的《设计模式》时，我体验到了他们的思想。然而，为了学习那些模式是如何工作的，我需要试验它们。在阅读之后，关于“四人帮”的模式仍然有很多方面我并不真正赞同和理解，但是我知道，实践和更多的阅读将增长我的理解。

当你试验这些模式时，请让我了解你的工作。是否有些模式没有被很好解释？还有我应当考虑的其它变形吗？请给我发电子邮件，让我知道你的想法，以便我能更广泛地传播这个信息。（我的电子邮件地址是100031.3311@compuserve.com。）

本书的最大问题之一是存在如此多的疏漏。我已经从一些领域描述了模式，但是还有很多没有提到的其它领域存在的需要理解的模式。即使在我已经提到的领域中也还能发现更多的模式。并且我所描述的模式是不完全的；关于如何使用它们、存在什么变形、出现什么实现问题、能够怎样测试它们和如何得到最优的性能，都有很多需要学习。

本书反映了我的知识的不完全的状态。为更进一步，你需要看一看模式团体正在进行的不断增长的工作。其它的模式书籍正在出版，并且更多的模式书籍会在今后的几年中很快出现。虽然现在像这样的关于分析模式的书还没有很多，但我希望本书将会鼓励更多这样的书出现。在很多方面，如果本书能够终止无休止地连续出现分析和设计书籍，并且开创新的模式书籍的热潮，那么这可能是本书的最大意义。

获得模式信息的最好途径之一是万维网。Ralph Johnson的模式主页<sup>⑨</sup>是模式信息的核心来源。Ward Cunningham的波特兰模式库<sup>⑩</sup>也包含了大量有价值在线信息。

现在，许多会议都包括关于模式的演讲和讨论。然而最关注模式的会议是Pattern Language of Programming (PLoP)，每年的9月在伊利诺伊州的Allerton Park召开。这个会议是一个特殊的事件，最特别的是论文的表述方法。每篇论文不是进行正式的陈述，而是在一个由多位作者参加的专题讨论会上由其他作者对其进行批评。结果是对每篇论文进行了引人入胜的讨论，在讨论中作者们大量地学习到其他人是如何看待他们的工作。

下一步是写一些你自己的模式。这不是一个令人畏惧的实践。我发现模式团体对新思想是开放的，并且很热切地希望更多的人们编写模式。PLoP是提交模式的一个极好的论坛，它为了解模式开发的整个领域提供了一个顶级的会议地点。你也可以在网络上发布模式——波特兰模式库就是专门为这个目的设计的。我也想在本书的网络站点上发布其他人的分析模式。实际上我希望本书将来的版本会包括其他作者的模式，并且我的角色能够变得更像一个编辑而不是一个作者。

一开始，我之所以写这本书是因为我想要读像这样的一本书。现在仍然是这样。我希望本书和其后续的相关书籍将意味着未来一代的软件项目不用在一片空白的基础上开始。

## 参考文献

[310]

- Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

<sup>⑨</sup> <http://st-www.cs.uiuc.edu/users/patterns> (或<http://hillside.net/patterns/>)  
<sup>⑩</sup> <http://c2.com/ppr/index.html>

# 第三部分

## 附录

311  
312



## 附录 A

# 技术和符号

为了编写像这样的一本书，我需要使用一些建模技术，但是我并不想用太多的时间来讨论它们。毕竟这是一本关于模式的书，而不是一本关于建模技术的书（有大量关于这方面的书）。对于这些技术至今都没有标准，所以我不得不选择那些我感觉正确并且不过于异类的东西。我发现没有一种方法包含所有的东西并且我喜欢从不同的方法中融合一些技术。在这个附录中我将讨论我所使用的技术和它们的符号。

### A.1 类型图

类型图显示一个系统的结构视图。它集中描述系统中对象的类型和它们之间存在的不同种类的静态关系。两种最重要的关系是关联（一个客户租借了许多盘录像带）和子类型（护士是一种人）。

在这个区域存在着关于符号的最有争议的论点。每个人都选择他们自己的、完全不同的符号。因此存在大量广泛类似的技术可供本书选择。从中挑选一种不是一件容易的事情。

一个强有力的竞争者是Rational 软件公司的统一建模语言（UML）[2]。但本书使用这种方法存在两个问题。首先是时机的问题。本书写于1994年到1995年间，而UML在本书完全起草之后才刚刚发布。就在我写这些时，那些符号只是适用于一个预先发布的形式，并且Rational正在讨论在发布正式版本之前对UML进行重大改变。第二个问题是UML集中于实现建模而不是概念建模——而本书集中于概念模式。

313

我为类型图选择了Odell[5]的符号，主要是因为他的方法在主要的OO方法中是最概念化的。然而为了更好地适应我的需要，我在一些地方对它进行了修改。

绝大多数的方法都有某种结构化建模技术的形式。由于我的目的是写一本这方面的教科书，因此Odell[5]是最适合本书的，他使用了一个非常

概念化的方法。开发者也应当阅读一本更加面向实现的书，例如Booch[1]，以便提供实现的视角。Cook和Daniels[4]提供关于结构化建模的最严格定义的描述，因此值得阅读。

### A.1.1 类型和类

起点是一个类型的概念，用一个矩形描述。重要的是我用词语类型而不是类。理解这两者之间的区别是很重要的。一个类型描述一个类的接口。一个类型能够由很多类实现，并且一个类能够实现很多类型。一个类型能够由很多类用不同的语言、性能权衡等来实现。一个类也能够实现很多类型，尤其是包括子类型化时。类型和类之间的区别在一些基于授权的设计技术中是很重要的，这在“四人帮”的书[3]中进行了讨论。这两个术语经常被混淆，因为大多数语言并不明显地区分它们。实际上，大多数的分析和设计方法都不明显地区分它们。

我发现从三个视角来思考类型图的建造<sup>①</sup>是有用的，这三个视角是：概念层、规约层、实现层[4]。概念模型以人们思考世界的方法建模。它们完全是精神上的图像，忽略了任何技术问题。概念模型的不同依赖于它们表达的是真实世界还是我们对世界的了解。这方面的一个例子是一个人和一个生日。在真实世界中所有的人都有生日，因此建模时把生日作为人的强制属性是合理的。然而，我们可能知道一个人却不知道他的生日。因此对于很多领域，生日在一个反映我们对世界的了解的概念模型中是可选的。这个区别对于历史信息很重要。一个把世界的结构按其本来面貌描述的模型能够被抽出作为某个时刻最好的及时快照。然而，如果它描述了我们所知道的，它通常也需要反映我们的记忆。本书中的模型采用的视角是捕捉一个关于我们对世界的了解的模型，因为这在信息系统中是最有用的视角。

规约模型是可以用来定义系统中软件构件的接口的模型。它可以是隐式的，也可以是显式的。显式规约模型的一个例子是C++头文件，它详细说明具体存在什么操作、它们的参数和它们的返回类型。隐式规约模型需要和一些说明它们如何转化成为一个显式接口的约定组合起来。例如，一个隐式规约模型上的生日属性，对于Smalltalk转化成为操作birthdate和birthdate: aDate，对于C++转化成为操作Date getBirthdate()const和void setBirthDate(Date)。

<sup>①</sup> 这个区别也能够适用于其它的技术，但它总是和结构化模型一起发表。

隐式规约模型能够比显式规约模型更接近于概念模型，并且它们能够比很多显式接口装载更多的信息。C++和Smalltalk的接口丢失了大量关于使用部分接口的规则信息。具有断言的Eiffel可以更完整，但是与一个紧密跟随概念模型的隐式模型相比它不容易理解。

实现模型直接位于一个类的内部<sup>Θ</sup>。无论作为文档还是对于类的设计者，它们都是有用的。它们不应当被类的任何客户使用，除非它们是在阐明整个项目中使用的通用实现原则。

概念模型和隐式规约模型几乎一样。因此你能够把本书中的类型图既看作概念模型又看作隐式规约模型。如果这两者之间显然有区别，那么我就在正文中指出。本书中只有少量的实现模型，我都在文中标记出来。不过，我使用和其他模型相同的符号来实现模型。

第14章讨论类型模型如何联系到实现模型。如果一个模式的实现引入了超出第14章范围的内容，则此时实现将与模式一起被讨论。

### A.1.2 关联、属性和聚集

关联描述类型的实例之间的关系（一个人为一家公司工作，一家公司有一些办公室，诸如此类）。关联的一个精确解释依赖于它们是否是一个概念模型、规约模型或者实现模型的一部分。一个概念上的解释仅仅声明在对象之间存在一个概念的关系。在职责方面，它们负责相互之间的了解。因此，在一个订单和一个客户之间的一个关联被理解为这个订单知道它的客户并且反之亦然。在一个规约模型中，存在访问和更新关系的操作；一个显式规约模型显示模型上的操作和它们的名称。一个实现模型把一个关联解释为一个指针或者其它引用的存在。要注意的重要一点是：在概念和规约模型中关联并不表明数据结构。因此封装得到保护。[315]

我想要区分关联和映射。一个映射（有时称为角色）是从一个类型到另一个类型的有向连接。一个关联包含一个或者两个映射。一个单向的关联只是一个映射并且能够被看作等同于一个映射。一个双向的关联包含两个映射，这两个映射称为是相互反向的。这个反向与数学上的反函数是完全不同的。它在本质上意味着如果你遍历一个映射和它的反向映射，你将得到一个包含你的起点的对象聚合。因此如果一个客户通过它制定的订单的集合来遍历，则这些订单中的每个都指向那个客户。术语“源”（或者“领域”）指出映射来自的类型，术语“目标”（或者“值域”）指出映射指向的类型。（例如，在一个从客户到订单的映射中，客户是源而订单是目

<sup>Θ</sup> 一个实现模型可以被更确切地称为一个类图。

标。) 当一个名称和一个关联一起出现时, 这个名称是其中一个映射的名称。你能够通过这个名称到关联的位置来说明它是哪一个映射: 目标在前边, 源在后边, 名称在左边。

关于双向关联的价值存在一些争论。在概念上所有的关系都是双向的。考虑一个人和他的生日之间的一个关联。如果说在一个日期和那天出生的人们之间存在一个关联, 这在概念上是完全具有意义的。但是在一个规约模型中却不是正确的。一个日期, 如果对所有引用它的事物都在这个日期中建立一系列操作, 那么将使这个日期的接口膨胀到一个不合理的程度。关于双向关联的另一个问题是它增加了类型之间的耦合。这会使复用更加困难。很多人使用单向关联来减少类型之间的依赖性。相反的论证是在信息系统中大量的工作是通过类型之间的连接进行遍历。当这些连接主要是单向时, 就更难以找到回来的道路。一个类似的情况是试图在一个城市中找到你自己的位置: 单向的道路使整个事情变得更加困难, 即使你了解这个城市。

本书中的模式指明双向关联。但你使用这些模式时, 你能够选择使用双向关联或者单向关联。你工作的应用应当对使用什么方向的关联和丢弃哪些关联提出建议。你的选择并不真正影响模式。如果你使用双向关联, 你能够使用14.1节中的模式来帮助你实现它们。

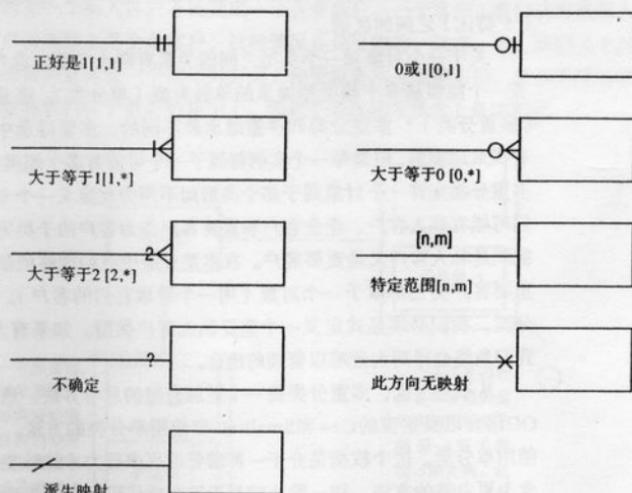
关联的一个关键方面是基数(有时称为多重性)。它规约的事情就像一个人能够为多少家公司工作和一个母亲能够有多少个孩子。一个基数是映射而不是关联的一个特征: 每个映射有它自己的基数。有很多基数的符号; 图A-1显示本书中使用的一些符号。上界为1的映射被称为是单值的, 上界大于1的映射被称为是多值的。多值的映射被假定去描述一个集合, 除非有其它的指定(通过一个简短的语义说明)。

在本书中我把一个属性和一个单值映射等同看待。有时我在一个类型的矩形内部显示一个属性, 有时使用一个关联。区别只是在于符号上的便利。

一些方法使用聚集关系, 它们是部分/整体的关系(例如, 一个锤子由一个头部和一个手柄组成)。在本书中我很少使用聚集。我并没有发现这个概念对领域模型有很大的用处, 因为它的大多数语义是关于所有关联的。因此它成为另一个需要记忆和争论的符号, 并且争论的结果通常对两方面都不很重要。然而, 我确实在应用层使用了它(参见13.6节)。

派生的(或者计算的)关联描述如何能够基于基础关联来定义其它的关联。(因此祖父关联是用父亲关联所跟随的父辈关联定义的。)一个概念模型上的派生映射表明一个映射是基于这个模型上存在的其它映射。在一个规约模型上, 它表明对一个派生映射而言访问者的结果和使用下层映射的组合相同。这样一来, 派生映射也能够被看作是在派生映射和基础映射

之间的一个约束。对于下层数据结构而言，把一个映射标记为派生映射除了这个约束外没有别的重要意义。实现者可以选择任何数据结构，只要类型的使用者得到的印象是：派生映射是依据这个模型而派生的。在一个实现模型上，派生映射表明被存储的数据和数据上的方法之间的区别。



图A-1 本书中使用的基数的符号

在关联主题上存在很多其它的变形。我试图尽可能使问题简单。在第15章中，把一些有用的变形作为关联模式进行讨论。

### A.1.3 泛化

让我们把一个商业实体的私人客户和企业客户作为泛化的一个典型例子。客户的这两个类型是不同的，但也具有很多相同点。这些相同点可以放在一个通用的客户类型中，而私人客户和企业客户作为子类型。

同样，这个现象在建模的不同级别有不同的解释。如果企业客户的所有实例通过定义也是客户的实例，那么在概念上我们能够说企业客户是客户的一个子类型。在一个规约模型中，企业客户的接口必须符合客户的接口。也就是说，企业客户的一个实例能够在使用一个客户的任何场合，并且调用者不需要担心实际上存在的一个子类型（可替换性原则）。企业客户对某个命令的响应可以和另一个客户不同（多态性），但是调用者不用担心这个区别。

在OO语言中，继承和子类化（subclassing）是一种子类继承超类的数据和操作的实现方法。它和子类型化（subtyping）有大量的相同之处，但是也存在重要的区别。子类化只是实现子类型化的一种方法（参见14.2节）。没有子类型化也能够使用子类化——但是大多数的作者确实都不赞成这种实践。更新的语言和标准越来越强调接口继承（子类型化）和实现继承（子类化）之间的区别。

关于一个对象和一个类型之间的关系有两个问题。首先，一个对象具有一个能够从多个超类型继承的单独类型（单分类），还是具有多个类型（多重分类）？多重分类和多重继承是不同的。多重继承中一个类型能够有很多超类型，但是每一个实例都属于一个可能有多个超类型的单独类型。多重分类允许一个对象属于多个类型而不用为此定义一个专门的类型。我们可能有私人客户、企业客户和重要客户作为客户的子类型。一个客户可能既是私人客户又是重要客户。在多重分类中我们能够把私人客户类型和重要客户类型都赋予一个对象（用一个继承它们的客户）。如果没有多重分类，我们必须显式定义一个重要私人客户类型。如果有大量的子类型，我们最终会得到大量难以管理的组合。

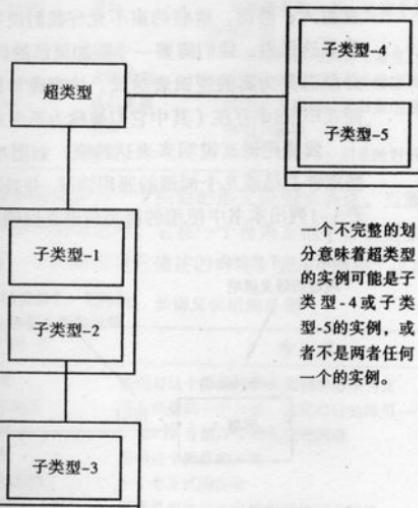
从概念上说，多重分类是一个更加自然的思考方法。然而，大多数的OO语言以及主流的C++和Smalltalk都使用单分类的方法。大量的方法也使用单分类。这个权衡是介于一种需要花更多努力才能转变成代码的概念上更自然的方法，和一种比较易于转变成代码但是受实现束缚更多的方法之间。我更喜欢那种更加概念化的方法，并且在本书中使用多重分类。

当使用多重分类时，我们必须说明在把子类型分组到划分中时哪些组合是合法的，如图A-2所示。同一个划分中的类型是不相交的；也就是说，没有对象可以是单个划分中多于一个类型的实例。因此超类型不可能既是子类型-1又是子类型-2。一个不完整的划分意味着超类型的一个实例不一定是在这个划分中的某个子类型的实例。一个完整的划分意味着超类型的每个实例必须也是这个划分中某个子类型的实例。

第二个问题是一个对象是否能够改变它的类型。例如，当一个银行的账户透支时，它在本质上改变它的行为，有几个操作（撤销，关闭）被覆盖。动态分类允许对象在子类型化结构的内部改变类型，但静态分类不可以。同样地，主要的OO语言和大多数OO方法都是静态的，并且这里适用的是和单/多重分类相同的权衡。本书采用更加概念化的动态分类方法。

看待动态分类的一个方式是它统一了状态和类型的概念。当使用静态分类时，我们必须注意把依赖状态的行为与子类化区别对待。动态分类把它们同样对待。

对动态类的使用发现概念模型和实现模型之间的一个细微区别。在一个概念模型中所有子类型化都被看作动态的，除非用一个简短的语义说明[immutable]（不变的）显式地定义。这不仅反映世界上可能发生的变化而且反映我们关于它们的不断变化的知识。对一些业务而言，真实的情况可能是一个私人客户不能变成一个企业客户。一个客户，我们认为是私人客户而实际上是企业客户，这种情况也是可能的。这样一来，我们关于世界的知识意味着一个动态的分类，即使世界本身是静态的。信息系统通常建立在我们关于世界的知识上，因此子类型化在概念上是动态的。



图A-2 泛化的符号

超类型的一个实例可能是子类型-1和子类型-4，但不可能是子类型-1和子类型-2。

然而，我们不能忽略处理动态分类的额外的复杂性。因而，概念上的动态子类型化在一个规约模型中通常被声明为静态的。这有效地说明虽然我们知道分类可能改变，但是它的发生如此罕见足以使我们不想付出额外的努力（和费用）来支持它。如果它真的发生，用户不得不用复制和替换来处理这种情况。在很多情况下，这种动态的罕见性足以使这种方法值得使用。长期的灵活性能够通过确保访问者接口在两种情况下相同来维持。

最后，决定一个划分是静态的还是动态的依赖于应用，所以我没有试

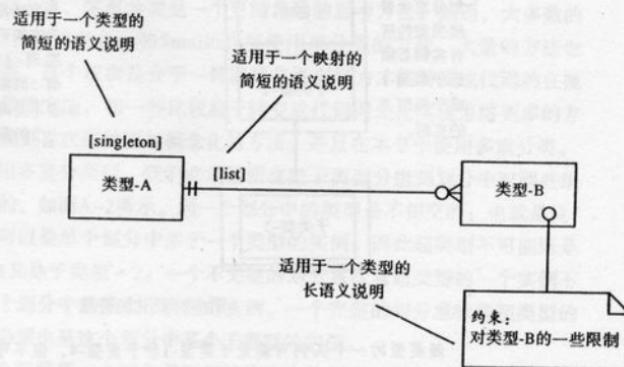
图在模式中制定任何通用的声明。当你在一个静态分类的语言中工作时，为了简单化，我建议你尽可能使用静态分类。

如果你正在使用一个不使用多重动态分类的方法，那么你将需要用14.2节中提出的模式来转变这些模型。

#### A.1.4 规则和语义说明

虽然在谈论类型时我们会大量谈到关联和子类型，但它们并不是全部。我可能有一个人寿保险对象，它具有针对保险客户和受益人的映射。我能够使用基数约束来捕捉一些说明，例如只有一个保险客户但是可能有很多受益人；然而，这些约束不允许我们说明保险客户一定不是一个受益人。为了这样做，我们需要一个更加灵活的约束。约束是关于一个类型的一个必须总是为真的逻辑表达式。约束通常被OO方法忽略，虽然它们早就已经在Eiffel中存在（其中它们被称为不变式）。

我使用语义说明来表达约束，如图A-3所示。简短的语义说明表示那些能够总结成几个词语的通用情况，并且被放在一个方括号中添加到图上。表A-1列出本书中使用的简短的语义说明。



图A-3 语义说明的符号

表A-1 简短的语义说明

标记	附属于	含义
[abstract]	类型	类型的所有实例都是某个子类型的实例
[abstract]	映射	应当被领域的子类型覆盖。源也是抽象的
[class]	映射	映射是起源于类而不是实例。这等价于类变量或者静态成员

(续)

标记	附属于	含义
[Dag]	递归关联	用这个关联连接起来的对象组成一个有向无循环图
[Dag]	映射	映射返回对象的一个有向无循环图
[global]	包	这个包对所有其它的包都可见
[hierarchy]	递归关联	用这个关联连接的对象组成一个层次结构
[hierarchy]	多值映射	映射返回一个对象的层次结构
[historic]	历史映射	保存以前连接的历史(参见15.3节)
[immutable]或者[imm]	映射	映射在创建一个实例之后就不能被修改
[immutable]或者[imm]	划分	子类型是静态的。在这个划分内对象不能改变类型
[key: 一个类型]	映射	一个带键值的映射(参见15.2节)
[list]	多值映射	映射返回对象的一个排序聚合(list)
[multiple hierarchies]	递归关联	用这个关联连接的对象组成几个层次结构
[singleton]	类型	类型只能有一个实例
[数字1, 数字2]	映射	数字1和数字2分别是映射的上界和下界

不是所有的事情都能用一个简短的语义说明来表达。当需要更多的空间时,我使用一个长语义说明,它在一个卷角方框中容纳更多的文本。一个长语义说明有一个头部指明它描述的内容。表A-2列举了这些头部。

322

表A-2 长语义说明的头部

头部	附属于	含义
约束	类型	说明对这个类型的所有实例都必须为真
派生	派生映射	导出映射的一个方法。实现可以选择另一个等价的方法
实例	类型	类型的所有被许可的实例的列表
方法	操作	指明这个操作的方法
注释	任何事物	一个非正式的注释
重载	类型	指明类型如何重载超类型的某个特征

不是所有的方法都提供捕捉语义说明中显示的那种信息的方法。然而,不要使大量这样的信息丢失是很重要的。方法渐渐提供某种能够用这种方式使用的可视化注释,它与长语义说明类似。

## 1.5 基础类型

在传统的数据建模中世界通常被划分为实体和属性。这个划分有些随意。实际上它总是被归结为属性,这些属性是由环境支持的基础数据类型——通常是整数、实数、字符串、日期或者可能其它的一些类型。

用对象系统,我们能够很容易地定义新类型,它们具有大量与这些内置类型相同的特征。Smalltalk中一个经典的例子是分数。在Smalltalk中一

个分数就像任何其它数字一样工作；实际上如果我们在Smalltalk中执行 $1/3$ 那么答案就是分数 $1/3$ ，而不是某个假无穷循环小数。

在开发系统时，我们必须使用这些类型。一个典型的例子是对货币价值的处理。在一个数据库中一辆小汽车的价值通常被作为一个数字处理，然而说一辆小汽车价值 $10\ 000$ 是没有意义的。币种总是重要的。通过使用对象，我们实际上可以定义一个既知道数字又知道币种的货币类型。它能够执行加法（核对币种匹配）并且能够创建一个用正确方法安排格式的打印输出。

表A-3列举了本书中使用的基础类型。

表A-3 本书中使用的基础类型

类 型	描 述
布尔	true（真）或者false（假），具有通常的操作
币种	单位的子类型，描述货币币种（例如，美元、英镑、日元）
日期	通常的日期（例如，1-Apr-1995）
持续时间	数量的子类型，其单位是时间（例如，5天3小时）。注意我们不能从天转变成月
整数	通常的整数（……-1, 0, 1, 2……）
量级	一个支持比较运算符（例如<、>、=、 $\geq$ 、 $\leq$ ）的类型
货币	数量的子类型，其单位是币种（例如，5美元、250法国法郎）。
数字	整数、实数、分数的超类型
数量	具有一个数字和一个单位的类型（例如，4英寸）（参见3.1节）
范围	两个量级之间的范围（参见4.3节）
实数	通常的实数
字符串	一小段文本。没有固定的限制，但是我通常把它解释为一小段单行文本。更长的文本项使用文字类型
文字	一大段文本，通常带有格式
时间	时刻（例如，下午1点20分）。不固定于一个特定的日期（参见时间点）
时间点	时间的一点。它可能只是一个日期，或者可能是一个日期和时间的组合
时间段	一段有开始和结束时间点的时间区间。一个时间段能够指出它是否和另一个时间段重叠，或者一个时间段在它的内部。它是时间点的一个范围
时间基准	时间段和时间点的超类型
单位	一个数量的单位（例如，英寸、牛顿）

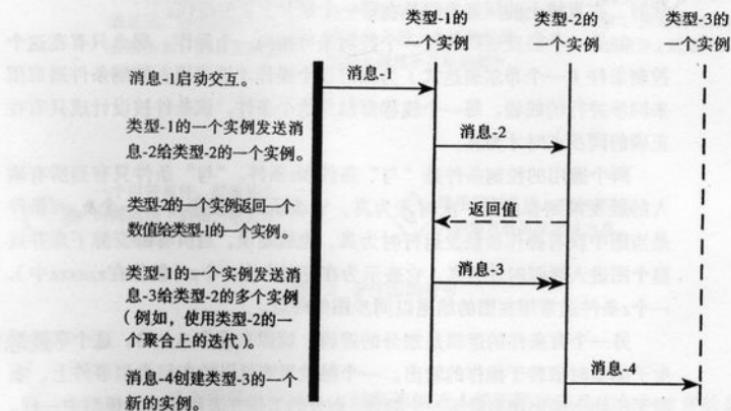
关于基础类型的重点是：从一个基础类型到一个非基础类型的映射是不会被实现的。否则，基础类型会得到一个巨大的接口，这个接口挤满到每个使用它的类型的访问者。这样既难操纵又不可复用。一个概念模型能够显示存在一个映射，因为概念上这个映射确实存在，但是一个相应的履约模型却不能存在。

一些作者喜欢把这些种类的类型称为文字 (literal)；然而，其它一些作者用术语“文字”代表非对象类型（例如C++中的实型），这就是我使用术语“基础类型”的原因。

我不想对本书中的基础类型进行完整详细的说明。这作为留给读者的一个练习（或者一个未来的版本）。这些类型的一部分在Cook 和 Daniels 的书[4]中有详细的说明。

## A.2 交互图

交互图显示几个对象如何合作完成某件事情。一个交互图中有一些代表对象的垂直线。这些线之间的箭头代表对象之间发送的信息，顺序是沿着页面从上到下，如图A-4所示。交互图被广泛地应用并且易于遵循。我所做的一个不寻常的事情是用一个双头箭头来显示同一个信息被发送到多个对象，这可能发生在一个循环中或者在一个聚合上迭代时。我也偶尔使用一个虚线来显示一个返回值；我并不总是这样做，但是当事情纠缠不清的时候，这样做是有用的。



图A-4 交互图使用的符号

在本书中我大量使用交互来显示行为。通常，我把它们和一个事件图联合使用（参见A.3节），因为这两种方法可以很好地相互补充。事件图用一种鼓励并行的方式定义行为，然而它们没有指明哪一个对象做什么。交互图显示这种行为如何能够在对象之间进行分配，但是取消了并行性和准确的行为逻辑。

你可能更熟悉用方框之间带编号的消息表示的交互图，这些方框与沿页面的直线是等同的。我更喜欢沿着页面的直线这种形式，因为我认为它使人更容易看到消息的顺序。

既然交互图如此简单，那么如果你以前没有使用过它们，你就不再需要关于它们的更多指南。更多的细节来自Booch[1]。

### A.3 事件图

事件图是我使用的另一种形式的行为模型。虽然它们比交互图更加复杂，但是它们确实允许规约完整的控制。它们也能够表达在业务建模中非常有用的并行行为。

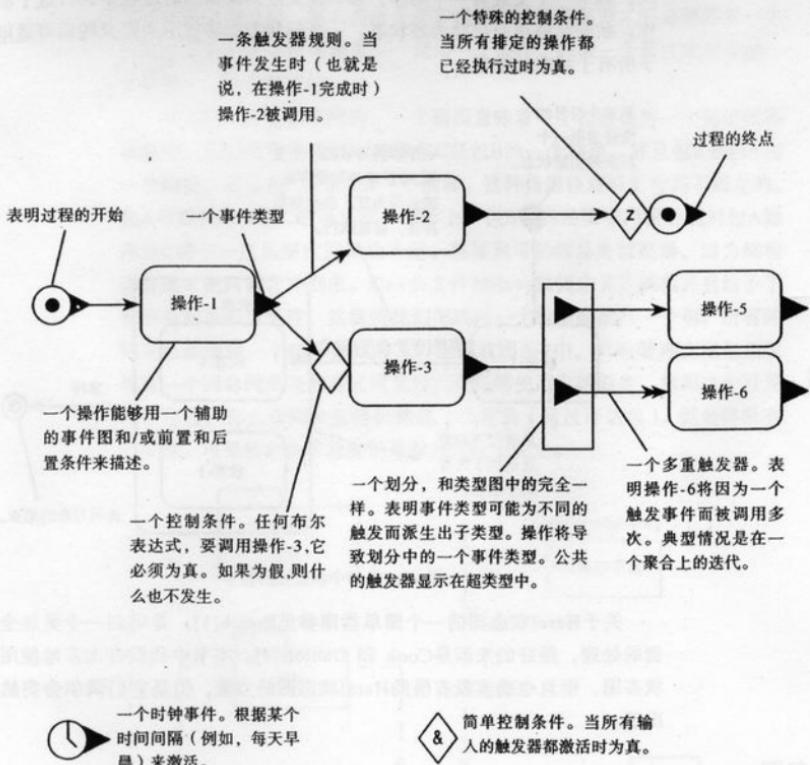
一个事件图中的方框表示操作，这些操作的完成标记了一个事件。一个触发规则表明一个事件触发一个操作。当一个事件类型上定义了多于一条的触发规则时就出现并行性。因此在图A-5中标记在操作-1末端的事件类型并行触发了操作-2和操作-3。这意味着操作-2和操作-3能够以任何顺序发生或者同时发生。并行性也会伴随着一个多重触发而发生，它用一个双头箭头表示。这表明这个事件多次触发操作，例如在一个聚合上发生迭代。在直线上的标签表明是在哪一个聚合上发生迭代。

如果一个触发规则经过一个控制条件指向一个操作，那么只有在这个控制条件（一个布尔表达式）为真时这个操作才被调用。控制条件通常用来同步并行的线程。每一个线程都触发这个条件，该条件被设计成只有在正确的同步点时才为真。

两个通用的控制条件是“与”条件和z条件。“与”条件只有当所有输入的触发规则都激活一次时才为真。它表示为在菱形中有一个&。z条件是当图中没有操作被触发运行时为真，也就是说，当所有都安静下来并且整个图进入睡眠时才为真。它表示为在菱形中有一个z（就像在zzzzzz中）。一个z条件通常用在图的结尾以同步图的终点。

另一个有条件的逻辑是划分的逻辑，就像在操作-3上的。这个事件派生子类型时依赖于操作的输出。一个触发规则可以放在超类型事件上，表明无论什么输出都会激活一个触发。划分的工作方法和在结构模型中一样。一个事件能够有很多定义在它上面的划分，一个划分内能够有任意数量的事件，并且划分可在相互之间任何想要的深度上定义。任何事件都是每个划分中惟一一个事件类型的一个实例。

事件图是概念上的，在图中只说明某个过程是如何工作的，而没有说明哪一个对象执行这个过程。因此它很好地补充了交互图。关于事件图的指南参见Odell[5]。



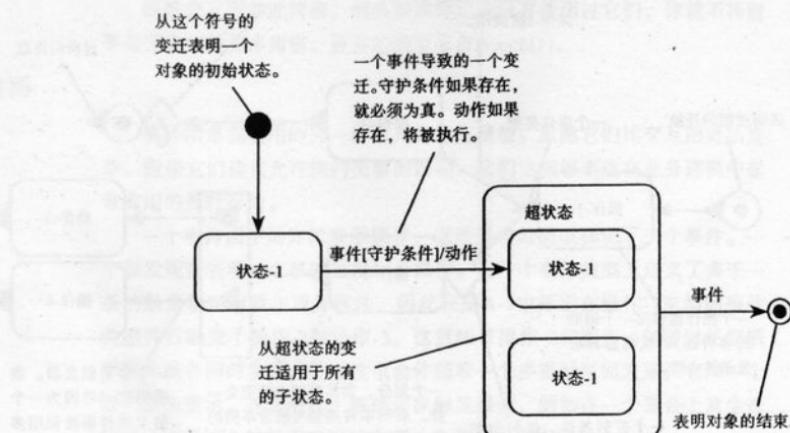
图A-5 事件图的符号

## A.4 状态图

状态图通过描述一个对象能够进入的不同状态和它如何改变状态来定义单个对象的行为。状态图在OO方法中最广泛的应用形式是Harel状态图。在本书中我使用这种形式的一个子集。一个状态图是为单个类型画的，它描述这个类型的每个实例的行为。

每个状态图用一个方框来表示，如图A-6所示。方框用变迁连接起来，这些变迁显示一个对象如何从一个状态变迁到另一个状态。变迁用造成这个变迁的事件来标记。如果一个变迁有一个守护条件，那么只有当事件发生并且守护条件的计算值为真时变迁才能发生。守护条件是一个布尔表达

式。如果一个变迁有一个动作，那么在迁移到新状态的过程中执行这个动作。状态能够被泛化成为超状态。一个超状态能够被用来定义随后可适用于所有子状态的变迁。



图A-6 本书中状态图的符号

关于Harel状态图的一个简单指南参见Booch[1]。要得到一个更加全面的处理，最好的来源是Cook 和 Daniels[4]。本书中我没有太多地使用状态图，而且也确实没有借助Harel状态图的力量，但是它们偶尔会突然出现。

## A.5 包图

在大型模型中我们需要一种方法来组织类型图中出现的大量类型。一个单独的大型类型图不仅对人来说太复杂而难以理解，而且对软件来说也难以管理。虽然一个大型的类型图对一个人来说能够被分割成很多页，但是这些页的随意选择对于控制软件没有什么作用。如图A-7所示，包图提供了一种更加可控的机制。

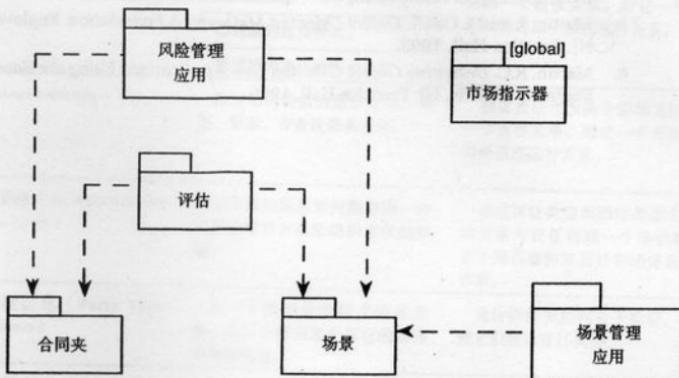
一个包（也称为分类领域、群集或者子系统）是一组类型（或者类）。一个类型只能属于一个包。通常类型被指定到一些包中，这样—来那些合作的类型一般也被放在相同的包中。在一个包中，任何类型都能够访问同一个包中的任何其它类型的所有特征。

包通过可见性关系连接起来。如果一个客户类型想要使用另一个包中

的一个服务器类型，那么在这个客户类型和服务器类型之间必须存在一个可见性关系。这对任何服务——调用一个操作、约束一个属性或者传递一个参数——都是需要的。

可见性和前提是不同的。一个前提意味着一个包需要另一个包出现在函数中。前提是可传递的：如果包C是包B的一个前提，并且包B是包A的一个前提；那么包C是包A的一个前提。这种传递性对可见性是不成立的。包A可能没有到包C的可见性；实际上，包B可能是被专门设计成对包A隐藏包C的——这是层次构架的本质。前提和可见性总是被混淆，因为编程语言通常把两者合并起来。C++头文件和Envy前提定义了前提并且给予了到所有前提的可见性，这就使我们无法用一个包来隐藏另一个包。所有的可见性必须在一个包中显式定义。因此在图A-7中，风险管理应用包必须具有一个到合同夹包的显式可见性以便能够使用它的服务。如果这个可见性不存在，那么合同夹包将仍然是一个前提（通过评估包），但是将没有可见性。可见性意味着前提但是反过来却不成立。

[329]



图A-7 描述包的符号

这个图是从图11-3中得到的。这里我对包使用Rational软件公司的统一建模语言（UML）的符号[2]，因为我发现它比Booch的原始符号更清晰。

在一个包的内部，类型可以是公有的也可以是私有的。公有类型可以被具有可见性的包看到；私有类型只能被同一个包中的类型使用。包能够被指定为全局的，在这种情况下所有其它的包都具有到它的可见性。这对通用的构件（例如整数、字符串和聚合）是必需的。

当开发一个大型系统时，我们试图把包之间的可见性最小化以便使这

个系统具有更少的依赖性并且因此变得更加容易管理。在本书中我主要在第11章和第12章中讨论包。

虽然这种类型的模型主要是针对较大的系统，但是它在方法中被讨论的次数并不多。Booch[1]介绍了我在这里所使用的基本思想，但是它的描述非常简要，主要是因为没有一个实际的例子很难讨论这个课题。Robert Martin解决了这个缺乏的问题，他给出一些使用包模型的例子[6]。

## 参考文献

1. Booch, G. *Object-Oriented Analysis and Design with Applications* (Second Edition). Redwood City, CA: Benjamin/Cummings, 1993.
2. Booch, G. and J. Rumbaugh. *Unified Method for Object-Oriented Development*. Rational Software Corporation, Version 0.8, 1995.
3. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
4. Cook, S. and J. Daniels. *Designing Object Systems: Object-Oriented Modelling with Syntropy*. Hemel Hempstead, UK: Prentice-Hall International, 1994.
5. Martin, J. and J. Odell. *Object-Oriented Methods: A Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
6. Martin, R.C. *Designing Object-Oriented C++ Applications Using the Booch Method*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

## 附录B

# 模式列表

章节	名称	问题	解决方案
2.1	团体 (Party)	人和组织单位有相似的职责。	创建一个团体类型作为人和组织的超类型。
2.2	组织层次 (Organization Hierarchies)	描述一个分层次的组织结构。	在组织上创建一个递归关联。
2.3	组织结构 (Organization Structure)	<p>一个组织结构具有层次或者更复杂的连接。</p> <p>出现新的连接种类。</p> <p>保留结构变化的历史。</p>	创建组织结构作为两个团体之间的一个直接关系。给它一个组织结构类型来描述这种关系。
2.4	责任 (Accountability)	描述结构类似的组织结构、雇佣、管理、专业注册和合同。	创建责任作为两个团体之间的一个直接关系。给它一个责任类型来描述这种关系。
2.5	责任知识级 (Accountability Knowledge Level)	记录那些描述如何能够用一种容易改变的方法来组织责任的规则。	通过责任类型和团体类型之间的关联为责任创建一个知识级。这个知识级约束责任和团体的操作级。
2.6	团体类型泛化 (Party Type Generalizations)	在一个模型中的很多团体类型，大部分都和某个其它的团体类型相类似。	允许团体类型拥有子类型，以便它们继承责任类型。
2.7	层次型责任 (Hierarchical Accountability)	把一些责任类型约束为一个层次结构。	定义包含分层约束的责任类型的一个子类型。级别的列表可以使你为层次中的每一级命名。
2.8	操作范围 (Operating Scope)	描述一个责任所意味的职责。	给责任增加一些操作范围。操作范围的类型依赖于责任的类型。
2.9	职位 (Post)	责任对应于工作职位而不是做这项工作的人。	创建一个职位作为团体的另一个子类型。指定某个人到一个职位并分配给他一个责任。职位的拥有者在职期间担负这个职位的职责。

(续)

章 节	名 称	问 题	解 决 方 案
3.1	数量 (Quantity)	描述一个数值, 如6英尺或5美元。	使用一个包含数字和单位的数量类型。货币是一种单位。
3.2	转换率 (Conversion Ratio)	在不同单位的数量之间进行转换。	记录单位之间的转换率。
3.3	复合单位 (Compound Units)	描述像kg/m这样的单位。	使用一个由其它单位组合而成的单位。
3.4	测量 (Measurement)	<p>一个对象有大量的数量属性。</p> <p>记录关于一个属性的单个测量的信息。</p> <p>跟踪一个属性的数值随时间的变化。</p>	<p>创建一个对象来描述一个单独的测量。它与被测量的对象和描述这种测量的现象类型相联系。</p>
3.5	观察 (Observation)	属性是定性的, 并因而不能用数字来测量。	创建一个把对象和现象连接起来的观察类型。每一个现象都是某个现象类型对应的数值。
3.6	观察概念的子类型化 (Subtyping Observation Concepts)	有一些现象是另一种现象的特殊情况。	允许现象通过知识级中的一个关联进行子类型化。
3.7	观察方案 (Protocol)	<p>当观察的方法偶尔会产生不同的解释时, 要处理类似的现象。</p> <p>记录一个测量的精确性和灵敏性。</p>	记录用来决定观察的方案。
3.8	双重时间记录 (Dual Time Record)	一个观察为真的时刻和你注意到它的时刻是不同, 并且一个事件发生的时刻和你注意到它的时刻也是不同的。	对所有这样的对象分别记录这两种时刻。
3.9	被否决的观察 (Rejected Observation)	观察被错误地执行, 但是不能被擦除。	保存它们, 把它们标记为被否决的, 并且记录是什么观察否决了它们。
3.10	临床观察、假设与推理 (Active Observation, Hypothesis, and Projection)	<p>观察的确定性。</p> <p>描述那些你认为可能发生的观察, 如果你的处理必须基于这种可能性。</p>	将观察进行子类型化, 变为临床观察(我将要处理它)、假设(我将要更深入研究它)和推理(我认为它可能发生)。
3.11	关联观察 (Associated Observation)	为诊断记录证据。	把诊断作为一个观察, 它具有一个关联, 连接到作为证据的观察。

(续)

章 节	名 称	问 题	解 决 方 案
3.12	观察过程 ( Process of Observation )	确定观察和诊断的过程。	每一个观察都可能导致要求进行更多观察和干预的建议, 以及对反面性观察的重新评价。因为这些步骤产生更进一步的观察, 所以导致一个连续的观察过程。
4.1	企业片断 ( Enterprise Segment )	用不同的标准和各种粒度把一个大的企业分成碎片。	把分割的每个标准定义为一个维度, 并且将它描述为一个元素的层次结构。把一个企业片断定义为每个维度的一个元素的组合。
4.2	测量方案 ( Measurement Protocol )	指明测量是计算得到的还是从数据库中读出的。 记录计算的公式。 同样的现象类型能够用依赖上文的不同方法来决定。	定义一个测量方案, 描述如何为一个现象类型创建一个测量。测量方案能够被提供或者计算得到, 而计算应当是因果关系的、可比较的或者维度的组合。
4.3	范围 ( Range )	描述两个数值之间的范围。	定义一个具有上下界和适当操作的范围类型。
4.4	带范围的现象 ( Phenomenon with Range )	把一个现象描述成在一个现象类型上的范围。	赋予这个现象一个范围属性。 在由其它现象描述的条件下创建一个把范围和现象连接起来的范围函数。
5.1	名称 ( Name )	引用一个对象。	把一个字符串作为对象的名称。
5.2	标识方案 ( Identification Scheme )	确保一个标识指向一个对象, 而不同的团体可以用不同的方式指向对象。	创建包含标识符并且每个标识符只指向一个单元的标识方案。一个团体可以使用任何标识方案。
5.3	对象合并 ( Object Merge )	两个对象实际上是相同的。	把一个对象的属性复制到另一个对象, 关闭所有从前者到后者的引用, 并且删除前者。 把一个对象标志为“被替代”, 并且赋予它到另一个对象的连接。 把两个对象的出现用一个表明它们相同的本质的对象连接起来。
5.4	对象等价 ( Object Equivalence )	一些人认为两个对象是相同的, 但其他人认为它们不同。	为这些对象创建等价关系。
6.1	账目 ( Account )	记录某个数量改变的历史。	创建一个账目。记录每次改变作为账目的一个条目。账目结算得出账目的当前值。

(续)

章 节	名 称	问 题	解 决 方 案
6.2	事务 ( Transaction )	确保一个账目没有任何丢失。	在账目之间使用事务来传递条目。
6.3	汇总账目 ( Summary Account )	一组账目看起来就像一个单独的账目。	创建一个汇总账目，把其它账目作为它的子账目。
6.4	备注账目 ( Memo Account )	不使用事务而把某个数量标记在一个副账目中。	创建一个不影响真正事务并且没有真正条目的备注账目。
6.5	记入规则 ( Posting Rules )	在账目之间自动传递。	定义账目之间的一个记入规则。
6.6	个体实例方法 ( Individual Instance Method )	对一个类型的每个实例，为某个操作赋予它自己独有的方法。	为每个方法定义一个 singleton 类的子类。 使用策略模式。
			创建一个隐藏在对象内部的 case语句。
			用参数隔不同的行为。
			建立一个简单的解释器。
6.7	记入规则的执行 ( Posting Rule Execution )	确保记入规则都在正确的时间执行。	当一个条目被放入一个账目时，激活所有向外的规则。 明确地请求激活一条记入规则。 请求一个账目激活它向外的记入规则。 当需要一个账目时，向后链式触发记入规则。
6.8	多个账目的记入规则 ( Posting Rules for Many Accounts )	为多个账目定义相同的记入规则。	在一个账目类型上定义规则。 在一个汇总账目上定义规则。
6.9	选择条目 ( Choosing Entries )	向一个账目请求它的条目子集。 向一个对象请求它的一个聚合中选择的一些对象。	账目返回所有的条目，而且调用者选择它所要的条目。 账目为每个可能的子集提供一个操作。 调用者通过一个过滤器对象到达账目。
6.10	账务实践 ( Accounting Practice )	指定几条记入规则组成一组。	创建一个账务实践把它们组合起来。
6.11	条目来源 ( Sources of an Entry )	了解一个事务是如何计算的。	用新的事务来记录记入规则的创建和计算中使用的条目。

(续)

章 节	名 称	问 题	解 决 方 案
6.12	结算单和所得计算书 (Balance Sheet and Income Statement)	描述结算单和所得计算书。	创建账目的子类。
6.13	对应账户 (Corresponding Account)	协调对同一个账户的两个团体的视图。	把每个视图作为单独的互相对应的账户。
6.14	专门化的账户模型 (Specialized Account Model)	在特殊的情况下使用通用的账务模式。	对该模式类型进行子类型化来支持专门化的需求。
6.15	登记条目到多个账户 (Booking Entries to Multiple Accounts)	把一个条目记录到多个账户中。	把一个账户作为真正的账户，为另一个账户使用备注账户。 把一个账户作为真正的账户，为另一个账户使用派生账户。
8.1	提议和执行的动作 (Proposed and Implemented Action)	描述你想要做的和已经做的。	为提议的和执行的动作使用相互独立的对象。
8.2	完成和放弃的动作 (Completed and Abandoned Actions)	指明一个动作是如何结束的。	一个动作如果按计划执行就是完成的，否则就是放弃的。
8.3	挂起 (Suspension)	把一个动作暂时停止。	把挂起记录在一个动作上。用一个时间范围来显示挂起持续的时间。
8.4	计划 (Plan)	记录一组你想要一起执行的提议的动作。 描述动作之间的依赖性。 允许不同的人们互相协调计划。	一个计划是用依赖关系连接起来的建议的动作组成的聚合。几个团体可以有不同的计划指向同一个建议的动作。
8.5	方案 (Protocol)	很多次以同样的方法执行标准的过程。	一个动作可以依据一个方案来执行。一个方案可以被分成用依赖关系连接的子方案。
8.6	资源分配 (Resource Allocation)	把资源分配到计划、方案和动作。	一般的资源分配分派一个资源类型的一个数量。特定的资源分配分派特定的资源。
8.7	输出和启动函数 (Outcome and Start Functions)	知道何时执行一个方案，以及方案或任何动作的结果将是什么。	启动函数和输出函数把一个方案和观察概念连接起来，这个观察概念触发方案或者方案的结果。
9.1	合同 (Contract)	从买方和卖方两者的观点来记录交易。	通过买方团体和卖方团体来使用订约人。
9.2	合同夹 (Portfolio)	为不同的目的动态选择合同。 动态选择对象。	定义一个合同夹作为合同的聚合。选择合同时使用一个过滤器——一个决定哪个合同适合合同夹的布尔表达式。

(续)

章 节	名 称	问 题	解 决 方 案
9.3	报价 (Quote)	区分购买和出售所提供的价格。	把两个价格合并成一个单独的报价。
9.4	场景 (Scenario)	交易物的价格随时间而改变。 考虑价格的假设组合。 一个交易物的价格能够影响另一个交易物的价格。	创建一个场景来捕捉真实的和虚拟的市场状态。一个场景提供了当时状态下任何交易物的价格，并且包括了在虚拟市场状态下导出价格的规则。
10.1	期货合同 (Forward Contracts)	一个合同可能在将来以今天的价格交付。	使用一个合同时，交易日期和交付日期相互独立。
10.2	期权 (Options)	一个团体可能选择在将来的某个时刻以一个事先确定的价格买卖某种东西。	期权是合同的一个拥有附加行为的子类型。 期权是一个单独的对象，拥有一个合同作为一个属性。
10.3	产品 (Product)	一个组合期权被推销员看作一个条目，但是销售商却把它看作较简单的合同的一个聚合。  一个推销员看到一个包，但是在包内部只有条目。	把一个推销员销售的东西看作一个产品，但把它内部需要估价的一个成分看作一个合同。
10.4	子类型状态机 (Subtype State Machines)	一个关卡期权与一个期权相比有不同的行为，但是它看起来像一个子类型。要处理子类型和状态机。	确保子类型和超类型对象对相同的事件都有响应。
10.5	并行的应用和领域层次结构 (Parallel Application and Domain Hierarchies)	你在一个用户界面中显示一个对象列表。这些对象是各种不同的子类型，一些子类型属性需要被显示。你的用户界面对象不能因为向一个不正确的对象发送消息而失败。	应用对象检查领域对象的类型以确保它能够理解消息。  给超类型一个包含所有子类型行为的接口。  把属性作为一个运行时属性。  使用一个由领域对象装载的中间对象。  使用异常处理包。
11.1	对一个包的多重访问 (Multiple Access Levels to a Package)	一个包的不同客户需要不同数目的行为。	把包按访问的每一级分成单独的包。  允许包有不止一个接口。
11.2	相互可见性 (Mutual Visibility)	两个包中的类型需要相互可见。	把两个包合并。  拥有两个相互可见的包。  决定一个类型不能看到另一个类型。

(续)

章节	名称	问题	解决方案
11.3	包的子类型化 (Subtyping Packages)	将子类型与包一起使用。	子类型可以放在一个单独的包中。对包的可见性适用于超类型，但反之不行。
12.1	两层构架 (Two-Tier Architecture)	在一个客户/服务器系统上划分软件。	把用户界面置于客户端，把数据库置于服务器端。用户界面类直接访问数据库。
12.2	三层构架 (Three-Tier Architecture)	两层构架把用户界面和数据库设计耦合得太紧密。  数据库界面不能支持一个丰富的领域模型。	由三个逻辑层：应用、领域、数据库。
12.3	表示层和应用逻辑层 (Presentation and Application Logic)	应用软件既处理领域模型的解释又处理导出的用户界面。	把应用层分割成表示层（用户界面）和应用逻辑层（处理领域模型）。把应用逻辑层构成一组针对表示的外观。
12.4	数据库交互 (Database Interaction)	使用数据库工作。	让领域类负责在数据库中保存自己。  创建一个单独的层次来处理数据库和领域对象之间的交互。
14.1	实现关联 (Implementing Associations)	实现一个概念关联。	选择一个方向来实现，使用一个操作和一个指针。  在两个方向上都使用操作和指针。  在两个方向上都使用操作，但只在一个方向使用指针，另一个方向使用查找。  在两个方向上都使用操作，对指针使用一个表进行查找。
14.2	实现泛化 (Implementing Generalizations)	实现泛化，尤其是在包含多重的和动态的分类时。	使用继承。  为具有多重继承的子类型的每个组合使用类。  使用一个内部标志。  委托给一个隐藏类（状态模式）。  复制和替换。
14.3	对象创建 (Object Creation)	创建一个对象。	对所有强制的和不可改变的映射都使用一个带参数的创建方法。

(续)

章 节	名 称	问 题	解 决 方 案
14.4	对象析构 (Object Destruction)	破坏一个对象。	拥有一个专门的析构方法。定义应该在多大程度上级联删除。
14.5	入口点 (Entry Point)	开始寻找对象。	让类负责保存和查找它的实例。 由一个登记表来查找和保存对象。
14.6	实现约束 (Implementing Constraints)	实现约束。	给每个对象一个检查其约束的操作。当调试时，在修改者完成后调用它。
15.1	关联类型 (Associative Type)	给一个关联增加特征。	为关联创建一个类型。 使用一个专门的符号。
15.2	带键值的映射 (Keyed Mapping)	在一个映射中描述一些数值，它们的键值是另一个类型。	使用一个带键值的映射。
15.3	历史映射 (Historic Mapping)	记录一个映射的以前值。	使用一个历史映射。

# 索引

索引中的页码为英文原书的页码，与书中边栏的页码一致。

## A

Abandoned actions (被放弃的动作), 157, 160-161, 337  
Absence (缺乏)  
category observation (分类观察), 46  
observation concepts (观察概念), 47  
Abstract (抽象)  
mapping (映射), 136, 322  
posting rule (记入规则), 151  
type (类型), 322  
Accessors (访问者), 275-277, 278, 280  
Account (账目)。参见Summary account  
booking entries to multiple accounts, 97, 127-132, 337  
corresponding (对应的), 96, 124-125, 337  
derived (派生的), 130-131  
filter (过滤器), 119, 120  
generally (通常), 95, 97-98  
memo (备注), 96, 103-104, 336  
pattern (模式), 335  
posting (记入), 141  
sign (符号), 97  
specialized model (专门化模型), 96, 125-127, 337  
statement (声明), 97  
Account-based firing (基于账目的触发), 112-113, 143  
Accountability (责任)  
abstraction (抽象), 23  
generally (通常), 17-18, 22-24  
hierarchic (层次的), 17, 28-30, 332  
knowledge level (知识级), 17, 24-27, 332  
operating scopes (操作范围), 30-32

organization hierarchies (组织层次), 17, 19-21, 331  
organization structure (组织结构), 17, 21-22, 331  
party (团体), 17, 18-19  
party type generalizations (团体类型泛化), 17, 27-28, 332  
pattern (模式), 331  
post (职位), 17, 32-33  
Accounting and inventory (账务和库存)。参见Account; Entry; Individual instance method  
balance sheets and income statements (结算单和所得计算书), 96, 123-124, 337  
patterns (模式), 134  
posting rule execution (记入规则的执行), 96, 111-115, 336  
posting rules (记入规则), 96, 104-105, 336  
posting rules for many accounts (到多个账目的记入规则), 116-118, 336  
practice (实践), 119-122  
practice pattern (实践模式), 96, 337  
specialized account model (专门化的账目模型), 96  
Total Telecommunications example (TT集团的例子), use in (用在), 133-134  
transactions (事务), 95-96, 98-101  
Accounting framework (账目框架), 132  
ACM。参见Aroma Coffee Makers (ACM)  
Action (动作)  
abandoned (放弃的), 157, 160-161, 337  
completed (完成的), 157, 160-161, 337  
implemented (执行的), 157, 158-160, 168, 337  
proposed (建议的), 157, 158-160, 168, 337

- Active observation (临床观察), 36, 49-50, 334  
 Actual status (实际状态), 69-71  
**Acyclic graph structure (非循环图结构)。参见DAG (directed acyclic graph)**  
 Aggregation in type diagrams (类型图中的聚集), 315-318  
 Alexander, Christopher (亚历山大·克里斯托夫), 5, 6  
**Analysis (分析)**  
 design techniques (设计技术), 3  
 generally (通常), 1  
 pattern (模式), 310  
 Anderson, Bruce, 5  
**Application (应用层)。参见Parallel application**  
**Application facade (应用外观)**  
 common methods (公共方法), 257, 262-264  
 contents of a facade (外观的内容), 257, 259-262  
 domain model (领域模型), visibility to (到……的可见性), 221  
 generally (通常), 257-258  
 health care example (医疗保健的例子), 257-259  
 methods for facade attributes (外观属性的方法), 260-262  
 multiple facades (多重外观), 257, 267-269  
 operations (操作), 257, 264-265  
 type conversion (类型转换), 257, 265-267  
**Application logic (应用逻辑层)。参见Presentation and application logic**  
**Architecture (构架)。参见Layered architecture for information systems; Three-tier architecture; Two-tier architecture**  
**Arguments (参数), 67**  
**Aroma Coffee Makers (芳香咖啡机制造公司) (ACM)**  
 accountability (责任), 31  
 actual versus planned status (实际状态与计划状态), 71  
 dimensions of (维度), 60  
 enterprise segments (企业片断), 61-65  
 framework (框架), use of resulting (使用最终框架), 82-83  
 location dimension (位置维度), 63  
 organization hierarchies (组织层次), 19-20  
 performance analysis (性能分析), 57  
**Asset (资产), 157, 168-172**  
**Associated observation (关联观察)**  
 defined (定义的), 36  
 linking of knowledge and operational levels (知识级和操作级的连接), 50-51  
**pattern (模式), 334**  
**trigger rule (触发器规则), 52**  
**Association objects (关联对象), bidirectional implementation (双向实现), 280**  
**Association patterns (关联模式)**  
 associative type (关联类型), 297, 298-301  
 generally (通常), 297-298  
 historic mapping (历史映射), 297, 303-307  
 keyed mapping (带键值的映射), 297, 301-303  
 two-dimensional history (二维历史), 298  
**Associations (关联)**  
 defined (定义的), 297  
 one-way (单向), 277  
 quantity in modeling (建模中的数量), 38  
 recursive (递归的), 322  
**Associations (关联), implementation of (关联的实现)。**  
 参见 Bidirectional associations  
 derived mappings (派生映射), 281  
 fundamental types (基础类型), 277  
 generally (通常), 272, 274  
 interface (接口), 275-277  
 nonset mappings (非集合映射), 281  
 pattern (模式), 341  
 type diagrams (类型图), use in (用在), 315-318  
 unidirectional (单向的), 274-275, 278, 316  
**Associative type (关联类型), 297, 298-301, 342**  
**Atomic unit (原子单位), 39-41**  
**Attributes (属性)**  
 object information (对象信息), 35  
 phenomenon with range (带范围的现象), 78-80

quantity (数量), 97-98  
 quantity in modeling (建模中的数量), 38  
 type diagrams (类型图), use in (用在), 315-318

## B

Backward-chained firing (向后链式触发), 114  
 Balance sheets (结算单), 96, 123-124, 337  
 Bags (袋)  
   account (账目), 98  
   collections (集合), 273  
   mappings with (映射), 39-41  
   protocol components (方案构件), 165  
 Bank (银行)  
   derivatives trading system (衍生交易系统), 240, 245-246  
   foreign exchange derivatives trading system (外汇派生交易系统)。参见Trading

Barings Bank collapse (巴林银行倒闭), 205-206  
 Barrier option (关卡期权), 211

Beck, Kent, 5, 133, 289

Behavioral meta-model (行为的元模型), 163

Bidirectional associations (双向关联)

  implementation (实现), 274-275, 278-281  
   type diagrams (类型图), 316

Black-Scholes analysis (Black-Scholes分析)

  derivative contract (派生合同), determining value of (确定派生合同的价值), 245  
   options (期权), determining value of (确定期权的价值), 201-202  
   risk evaluation (风险评估), 205

Block method in Smalltalk (Smalltalk中的块方法), 108

Booch, 324

Booking entries to multiple accounts (登记条目到多个账户), 97, 127-132, 337

Boolean (布尔型),

  contract attribute (合同属性), 177  
   fundamental type (基础类型), 324  
   portfolio (合同夹), use in (用在), 180-182

BPR (Business process reengineering) (业务过程重组) 10  
 Broker (代理), interface (接口), 253  
 Browser (浏览器), 217, 218, 221-222  
 Business process reengineering (BPR) (业务过程重组 (BPR)), 10

## C

C++  
   association interface (关联接口), 277  
   collection classes (聚合类), 273-274  
   constraints (约束), implementation of (实现), 294  
   contracts (合同), use in (用在), 181  
   exception handling (异常处理), 223  
   external iterator (外部迭代器), 98  
   history of (历史), 5  
   model prototypes (模型原型), 58  
   object creation (对象创建), 289  
   quotes (报价), use in (用在), 186, 188  
   Total Telecommunications example (TT集团的例子), use in (用在), 133  
   type checking (类型检查), 218  
 Calculated measurement protocol (计算测量方案), 66-70  
 Call (看涨(期权)), 在期权合同中 (in option contract), 202-204  
 Calls (呼叫), phone (电话)  
   separation into day and evening (分成白天和夜晚两类), 143-145  
   setting up of (建立), 134, 142-143  
   time (时间), charging for (按时间收费), 145-148  
 Cardinality (基数)  
   keyed mapping (带键值的映射), use in (用在), 60  
   type diagrams (类型图), use in (用在), 317  
 Cascading delete (级联删除), 290  
 Category (分类)  
   absence (缺乏), 46  
   mapping to phenomenon type (到现象类型的映射), 43-45  
   pattern (模式), 8

- presence (存在), 46
- Causal measurements protocol (因果测量方案), 58, 68-73
- Class mapping (类映射), 322
- Classes (类)
- collection (聚合), 273
  - combination (组合), 282
  - entry point (入口点), use in (用在), 293-294
- Classification (分类), 318-321
- Coad, Peter, 6
- Combination option (组合期权), 206
- Comparative measurements protocol (比较测量方案), 58, 68-75
- Comparative status type (比较状态类型), 71-72
- Completed actions (完成的动作), 157, 160-161, 337
- Compound unit (复合单位)
- bags (袋), use of (使用), 39-41
  - defined (定义的), 39
  - pattern (模式), 332
  - quantity pattern extension (数量模式扩展), use in (用在), 35
- Conceptual model (概念模型)
- analysis and design (分析和设计), compared (比较的), 1
  - analysis and design techniques (分析和设计技术), use in (用在), 3
  - business process reengineering and (业务过程重组), 10
  - contract (合同), 176-177
  - creation of (创建), 2
  - individual instance method (个体实例方法), 106
  - quantity (数量), use in (用在), 38
  - software language (软件语言), expression of (表示), 3
  - software technology (软件技术), independent of (独立于), 4
  - type diagrams (类型图), use in (用在), 314-315
- Conceptual schema (概念模式), 242
- Concurrency (并发性), 244
- Conformance (一致性), 211-214, 215-216
- Constraint (约束)
- implementation of (实现), 294, 342
- phenomenon with range attribute (带范围属性的现象), 79
- type (类型), 323
- Constructor parameter method (构造器参数方法), 138
- Consumable (消耗品), 157, 168-172
- Containers (容器), 273, 278
- Contract (合同)。参见Derivative contract; Forward contract
- generally (通常), 175, 176-180
  - package (包), 231-233
  - pattern (模式), 338
  - selectors (选择器), 182-184
  - spot (现货), 198
- Control condition (控制条件), 140
- Conversion ratio (转换率)
- generally (通常), 35
  - individual instance method (个体实例方法), 39
  - monetary values (金钱数值), 39
  - pattern (模式), 332
  - scenarios (场景), used to convert (用来转换), 39
  - unit conversion (单位转换), 38
- Cook, 211, 213, 324-325
- Coplien, Jim, 5
- Copy and replace (复制并替换), in object merge (在对象合并中), 90
- Corporate finance observations (公司财务观察)。参见  
Aroma Coffee Makers (ACM); Measurement protocol
- enterprise segment (企业片断), 58, 59-65
- framework (框架), use of (使用), 82-83
- generally (通常), 57-58
- range (范围), 58, 76-77
- Corresponding account (对应账户), 96, 124-125, 337
- Cosmos Clinical Process Model (Cosmos临床过程模型), patterns used in (使用的模式), 158
- Cosmos project (Cosmos项目)
- accountability model (责任模型), development of (开发), 18

- application facade (应用外观), model based on (基于的模型), 258-259  
 health care modeling (医疗保健建模), 36  
 layered architecture for information systems (信息系统的分层构架), 240  
 object of care (关照对象), 59  
 observations (观察), 49  
 Counterparty (对方团体), 178  
 Creation of objects (对象创建), 289, 342  
 Creation parameter method (构造参数方法), 138, 289  
 Cross-product control condition (叉积控制条件), 140  
 Cross-rate element (交叉汇率元素), 192-194  
 Cunningham, Ward, 5, 310  
 Currency (货币), fundamental type (基础类型), 324
- D
- DAG (directed acyclic graph) (有向非循环图), 166-168, 322  
 Daniels, 211, 213, 324-325  
 Database (数据库), 用在两层构架中 (use in two-tier architecture), 240-242  
 Database interaction (数据库交互)  
     domain tier (领域层), linking to data sources (连接到数据源), 252  
     generally (通常), 240, 251-252  
     interaction diagram (交互图), 254  
     interface tier (接口层), 252-256  
     pattern (模式), 341  
 Date fundamental type (日期基础类型), 324  
 Default method (默认方法), 261-262  
 Deletion of objects (对象删除), 290  
 Dependence (依赖性), 162, 166-167  
 Derivative contract (派生合同)  
     domain hierarchies (领域层次结构), 198, 216-223  
     forward contract (期货合同), 197, 198-200  
     options (期权), 197, 200-205  
     parallel application (并行应用层), 198, 216-223  
     product (产品), 197-198, 205-211  
 subtype state machines (子类型状态机), 198, 211-216  
 Derivative trade (派生交易)。参见Derivative contract  
 Derivatives trading system for a bank (一个银行的派生交易系统), 240, 245-246  
 Derived account (派生账户), 130-131  
 Derived mappings (派生映射), 281, 317-318, 323  
 Design analysis (设计分析), 1  
 Design templates (设计模板)  
     associations (关联), implementation of (实现), 272, 274-281  
     constraints (约束), implementation of (实现), 273, 294  
     design templates for other techniques (其它技术的设计模板), 273, 295  
     entry point (入口点), 273, 291-294  
     generalization (泛化), implementation of (实现), 273, 281-288  
     generally (通常), 271-272  
     goals of (目标), 272  
     model implementation (模型实现), use in (用在), 137  
     object creation (对象创建), 273, 289  
     object destruction (对象析构), 273, 290-291  
 Destruction of objects (对象的析构), 290-291, 342  
 Diagrams (图)  
     event (事件), 326-327  
     interaction (交互), 325-326  
     package (包), 328-330  
     state (状态), 327-328  
 Diagrams (图), type (类型)  
     associations (关联), attributes (属性), aggregation (聚集), 315-318  
     fundamental types (基础类型), 323-325  
     generalization (泛化), 318-321  
     generally (通常), 313-314  
     semantic statements (语义声明), 321-323  
     type and class (类型和类), 314-315  
 Dictionary (字典)

- collection (聚合), 273  
 historic mapping (历史映射), use in (用在), 305  
 keyed mappings (带键值的映射), use in (用在), 301  
**Digitalk Smalltalk**, 参见Smalltalk  
**Dimension** (维度)  
 combination (组合), 58  
 combination protocol (组合方案), 74-75  
 defined (定义的), 63  
 enterprise segment (企业片断), 58, 60-65  
 properties of (属性), 64-65  
**Directed acyclic graph** (有向非循环图) (DAG), 166-168, 322  
**Domain experts** (领域专家), involvement in conceptual modeling (包含在概念建模中), 3  
**Domain framework** (领域框架), 243-244  
**Domain hierarchies** (领域层次结构), 198, 216-217, 340  
 参见 Parallel application  
**Domain tier** (领域层), 242-245, 252  
**Double entry accounting** (双条目账务), 98-99  
**Dual time record** (双时间记录), 36, 47-48, 333  
**Duration** (持续时间), fundamental type (基础类型). 324  
**Dynamic classification** (动态分类), 320
- E**
- Each-entry posting rule (每条日记入规则), 143  
**Eager firing** (急切触发), 111-112  
**Edwards, John**, 10  
**Eiffel**, 294, 321  
**Einsteinian model** (爱因斯坦模型), developer use of (开发人员使用), 2  
**Eligibility condition method** (资格条件方法), use in posting rules (用在记入规则中), 118  
**Encapsulation** (封装), 274  
**Enterprise segment** (企业片断)  
 dimension (维度), defining of (定义), 63-64  
 dimension elements (维度元素), 60-62  
 dimension level type (维度层类型), 63  
 generally (通常), 58, 59  
 hierarchies of (层次), 59-60  
 object of care (关照对象), 59  
 pattern (模式), 334  
 properties of (属性), 65  
 top of hierarchy (顶层), 60  
**Enterprise-wide modeling** (企业范围内的建模), 235  

**Entry** (条目)

 accounting (账务), use in (用在), 95  
 booking to multiple accounts (登记到多个账目), 97, 127-132, 337  
 choosing of (选择), 96, 118-119, 337  
 double entry approach (双条目方法), 98-99  
 memo (备注), 129-130  
 sources of (来源), 96, 122-123, 337  
 storing of (存储), 119  

**Entry point** (入口点)

 classes (类), use of (使用), 293-294  
 find operations (查找方法), implementation of (实现), 293  
 generally (通常), 273, 291-292  
 interface for finding objects (查找对象的接口), 292-293  
 pattern (模式), 342  
 registrar objects (登记表对象), use of (使用), 293-294  
**Equivalence of objects** (对象等价), 85, 92-93, 335  
**Essence/appearance model in object merger** (对象合并中的本质/表象模型), 91-92  
**Event diagrams** (事件图), 326-327  
**External iterator** (外部迭代器), 98, 279  
**External schema** (外部模式), 242
- F**
- Facade** (外观), 参见Application facade  
 application logic tier (应用逻辑层), use in (用在), 247  
 client/server environments (客户/服务器环境), stretching in (延伸于), 250-251  
 database interface tier (数据库接口层), use in (用在), 253

multiple (多重), 267-269

Filter (过滤器)

account (账目), 119, 120

portfolio (合同夹), 181-184

Find arguments operation (查找参数操作), 74-75

Find operations (查找操作), 293

Firing approaches in posting rule execution (记入规则执行中的触发方法)

account-based (基于账目), 112-113

backward-chained (向后链式), 114

comparison of (比较), 114-115

eager (急切), 111-112

posting-rule-based (基于记入规则), 113-114

Fixed format of a pattern (一个模式的固定格式), 6

Flags (标志), generalization implementation (泛化实现), 283-284

Focal event (焦点事件), 63

Foreign exchange derivatives trading system for a bank (一个银行的外汇衍生交易系统), 176

Forward contract (期货合同)

date calculation (日期计算), 199-200

defined (定义的), 198

generally (通常), 197

pattern (模式), 339

tenor (期限), 198-199

Framework (框架), accounting (账务), 132

Frameworks and patterns (框架和模式), 11-13

Function (函数)

outcome (输出), 157, 172-174, 338

range (范围), 58, 80-81

start (启动), 157, 172-174, 338

Fundamental (基础的)

enterprise segment (企业片断), 65

types (类型), 277, 324

## G

Gang of Four ("四人帮")

creation patterns (创建模式), 289

delegation (委托/授权), used in design (用在设计中),

314

initial publication of (初始发布), 5

patterns of (模式), 110, 309

software interface and implementation differences (软件接口和实现差别), 4

software patterns (软件模式), influence on (影响), 6

Generalization (泛化), implementation of (实现)

delegation to a hidden class (授权给一个隐藏类), 284-286

flags (标志), 283-284

generally (通常), 273, 281-282

hasType operation (hasType操作), 288

inheritance (继承), 282

interface for (接口), 287-288

multiple inheritance combination classes (多重继承组合类), 282

pattern (模式), 342

replacement (替换), creation of (创建), 286

type diagrams (类型图), use in (用在), 318-321

Global package (全局包), 322

Graphs (图)

directed acyclic graph (有向非循环图) (DAG), 166-168, 322

plans and protocols used as (用作计划和方案), 166-168

## H

Hard-coding (硬编码), 194

HasType operation (hasType操作), implementation of (实现), 288

Hay, David, 4, 5, 132

Health care example of application facade (应用外观的医疗保健的例子), 258-259

Hedge (套利), defined (定义), 177-178

Hidden class (隐藏类), 284-286

Hierarchic accountability (层次型责任)

acyclic graph structure (非循环图结构), 28

generally (通常), 17

- leveled (分级的), 29-30  
 multivalued mapping (多值映射), 322  
 pattern (模式), 332  
 recursive association (递归关联), 322  
 rebalancing of subtypes (子类型的重新权衡), 30  
 summary accounts (汇总账目), 101-103  
 type (类型), 28-29  
 Hillside Group (山边小组), history of (历史), 5  
 Historic mapping (历史映射)  
     generals (通常), 297, 303-305, 322  
     pattern (模式), 342  
     two-dimensional history (二维历史), 305-307  
 Human artifact (人工制品), conceptual model as (概念模型作为), 2  
 Hypothesis (假设), 36, 49-50, 334
- I
- Idea (思想), defined (定义), 8  
 Identification scheme (标识方案), 85, 88-89, 335  
 Identifier (标识符), 85-87  
 Immutable (不变的)  
     mapping (映射), 322  
     partition (划分), 322  
 Implemented action (执行的动作)  
     generally (通常), 157  
     pattern (模式), 337  
     planning (计划), 158-160  
     resource allocation (资源分配), 168  
 Implementing associations (实现关联)。参见Associations, implementation of  
 Implementing generalization (实现泛化)。参见Generalization, implementation of  
 Income statement (所得计算书), 96, 123-124, 337  
 Individual instance method (个体实例方法)  
     calculated measurement protocol (计算测量方案), 68  
     conversion from Celsius to Fahrenheit (从摄氏温度到华氏温度的转换), 39  
     generally (通常), 96, 106
- implementation (实现), choosing of (选择), 110-111  
 internal case statement (内部case语句), 108-109  
 interpreter implementation (解释器的实现), 110  
 parameterized method of implementation (使用参数化方法实现), 109-110  
 pattern (模式), 336  
 posting rules (记入规则), use in (用在), 105  
 singleton class of implementation (使用Singleton类实现), 106-107  
 strategy pattern of implementation (使用策略模式实现), 107-108  
 Information systems (信息系统), layered architecture (分层架构)。参见Layered architecture for information systems  
 Inheritance (继承), generalization implementation (泛化实现), 282  
 Instantiation of knowledge level (知识级的实例化), 26  
 Instances (实例), 323  
 Integer (整型), fundamental type (基础类型), 324  
 Interaction diagram (交互图), 325-326  
 Interface (接口)  
     broker (代理), 253  
     destruction (析构), 290-291  
     generalization (泛化), 287-288  
 Internal case statement (内部case语句), use in individual instance method (用在个体实例方法中), 108-109  
 Internal schema (内部模式), 242  
 Interpreter implementation (解释器实现), use in individual instance method (用在个体实例方法中), 110  
 Intervention (干预), defined (定义), 53-54  
 Invariant check (不变式检查), 138  
 Inventory and accounting (库存和账务)。参见Accounting and inventory  
 Iterator (迭代器), 98, 279
- J
- Johnson, Ralph, 5, 310

## K

Keyed mapping (带键值的映射)

association patterns (关联模式), use in (用在), 297, 301-303

enterprise segment (企业片断), use in (用在), 60  
pattern (模式), 342

semantic statement (语义声明), 323

Keyed output (带键值的输出), use in Total Telecommunications example (用在TT公司的例子中), 135

Knock-in (插入型), 212-214, 215

Knock-out (删除型), 211

Knowledge level (知识级)

accountability (责任), 24-27, 332

generally (通常), 17

instantiation of (实例化), 26

operational levels and (操作级和), 24-26, 50-51

outcome functions (输出函数), 173

phenomenon type (现象类型), 41-42

planning patterns (计划模式), 165

posting rules (记入规则), 116-118

start functions (启动函数), 173

## L

Layered architecture for information systems (信息系统的分层构架)

database interaction (数据库交互), 240, 251-256

generally (通常), 225, 239-240

presentation and application logic (表示层和应用逻辑层), 240, 245-251

three-tier architecture (三层构架), 240, 242-245, 255-256

two-tier architecture (两层构架), 240-242

Lazy checking (消极检查), 170-171

Legal values method (合法数值方法), 261, 262, 263

Lewis, 273

List (列表), collection (聚合), 273, 322

Logic (逻辑)。参见Presentation and application logic

Logical data model (逻辑数据模型), 243

Long (多头)

contracts (合同), use in (用在), 177-178

options (期权), use in (用在), 202-204

## M

Magnitude (数量), fundamental type (基础类型), 324

Mapping (映射)。参见Historic mapping; Keyed mapping;

Multivalued mapping; Single-valued mapping

abstract (抽象), 136, 322

arguments (参数), list of (列表), 67

association (关联), comparison to (用来比较), 317

category to phenomenon type (到现象类型的分类), 43-45

class (类), 322

derived (派生的), 281, 318

directed acyclic graph (有向非循环图), 322

identification scheme (标识方案), use in (用在), 88-89

immutable (不变的), 322

nonset (非集合的), 281

number (数目), 322

trigger (触发/触发器), 137

Mapping with bags (使用 bags 的映射), 39-41

Measurement (测量)

calculated (可计算的), 67

generally (通常), 35, 41

operational level (操作级), 42

pattern (模式), 333

phenomenon type (现象类型), 41-42

Measurement protocol (测量方案)

arguments (参数), list of (列表), 67

calculated (计算出来的), 66, 67-68, 69-70

causal (因果关系的), 58, 68-70

comparative (比较的), 58, 68-70

corporate analysis (公司分析), 65-66

creation for a phenomenon type (一个现象类型的创建),

- creation of (创建), 71-73  
 defined (定义的), 58  
 dimension combination (维度组合), 73-76  
 pattern (模式), 334  
 range functions (范围函数), 81  
 source (来源), 66  
 status type (状态类型), 58, 69-71  
 Mellor, 211, 295  
 Memo entry (备注条目), 129-130  
 Memo account (备注账目), 96, 103-104, 336  
 Mental model (智力模型), creation of (创建), 1-2  
 Meta-model (元模型)  
     behavioral (行为的), 163  
     defined (定义的), 26  
     pattern (模式), 298  
 Method (方法), operation (操作), 323  
 Methods for facade attributes (针对外观属性的方法), 260-262  
 Model (模型)。参见Conceptual model  
     choosing of (选择), 2  
     Einsteinian (爱因斯坦理论的), 2  
     implementation (实现), 315  
     logical data (逻辑数据), 243  
     Newtonian (牛顿学说的), 2  
     specialized account (专门化的账目), 96, 125-127, 337  
     specification (规约), 314-315  
     structural (结构的), in Total Telecommunications example (在TT集团的例子中), 134-136  
     type (类型), 10  
 Modeling (建模)  
     derivatives (派生的), 197  
     enterprise-wide (企业范围内的), 235  
     examples for (例子), 8  
     implementation technique (实现技术), 177  
 Modeling principles (建模原则)  
     abstract interface (抽象接口), providing of (提供), 182, 196  
     abstract supertypes (抽象超类型), use of (使用), 187  
     abstract type (抽象类型), providing of (提供), 187  
     account value (账户数值), 98  
     alternative approaches (替代方法), choice of (选择), 204  
     association (关联), one-way or two-way decision (单向或者双向决定), 232  
     attributes combined into new type (组合成一个新类型的属性), 186  
     conceptual models linked to interfaces (连接到接口的概念模型), 4  
     conservation (守恒), principle of (原理), 99  
     date calculations in forward contracts (期货合同中的日期计算), 200  
     derived features (派生特征), 179  
     derived markers (派生标记), use of (使用), 203  
     design templates (设计模板), 272  
     feature (特征), marking of (标记), 203  
     model (模型), divide into levels (划分成不同的层次), 26  
     model modification and type changes (模型修改和类型改变), 22  
     modeling alternatives (建模选择), choice of (选择), 204  
     models (模型), usefulness of (使用), 2, 13  
     multiple attributes interacting with behavior (与行为交互的多重属性), 38  
     mutually visible packages (相互可见的包), 232  
     notation (符号), defining of (定义), 305  
     operational level (操作级), 42  
     patterns as starting point (作为起点的模式), 13  
     portfolio (合同夹), use of (使用), 181  
     postcondition of objects (对象的后置条件), 216  
     process (过程), making into a feature of a type (被放置在类型的一个特征中), 195  
     product/contract split (产品/合同分割), 210  
     responsibilities (职责), allocation of (分配), 211  
     responsibilities (职责), separation of (分离), 210  
     scenarios (场景), use of (使用), 191  
     sets of features (成组的特征), 179

- state charts (状态图), generalization effects (泛化效果), 216  
 subtyping (子类型化), use of (使用), 208  
 supertype and subtype generalizations (超类型和子类型泛化), 186  
 supertype logic (超类型逻辑), 24  
 type associations (类型关联), 42  
 Modifiers (修改者), 275, 276, 280  
 Monetary values (金钱数值), 37-39  
 Money (金钱), fundamental type (基础类型), 324  
 Multilegged transaction (多腿事务)  
     defined (定义的), 96  
     generally (通常), 99-101  
 Total Telecommunications example (TT集团的例子),  
     用在 (use in), 138  
 Multiple (多重的)  
     access levels to a package (到一个包的访问层次), 226-230, 340  
     classification (分类), 319-321  
     delete (删除), 290  
     hierarchies (层次结构), 322  
     source protocol (起源方案), 67  
     visibility (可见性), 227-230  
 Multiplicity (多重性), 317  
 Multivalued mapping (多值映射)  
     accounting structure (账务结构), 137  
     associations (关联), interface for (接口), 275-276  
     associative type (关联类型), use in (用在), 299  
     bidirectional implementation (双向实现), 278  
     hierarchy (层次结构), 322  
     type diagrams (类型图), use in (用在), 317  
 Mutual visibility (相互可见性), 230-233, 340
- N**
- Name of objects (对象的名称), 85, 86-87, 335  
 Newtonian model (牛顿学说的模型), 2  
 NHS Common Basic Specification (NHS通用基础规约), 158
- Non-entry point (非入口点), 293  
 Non-scenario approach (非场景方法), 190  
 Nonfundamental object (非基础对象), 65  
 Nonsingleton mappings (非集合映射), 281  
 Note (注意), 323  
 Number (数字型), fundamental type (基础类型), 324  
 Number mapping (数字映射), 322
- O**
- Object creation (对象创建), 289, 342  
 Object destruction (对象析构), 290-291, 342  
 Object equivalence (对象等价), 85, 92-93, 335  
 Object merge (对象合并)  
     copy and replace (复制并替换), 90  
     essence/appearance model (本质/表象模型), 91-92  
     generally (通常), 85, 90  
     pattern (模式), 335  
     superseding (替代), 85, 90-91  
 Object-oriented (面向对象)  
     analysis (分析), 38  
     language (语言), 89  
     technique (技术), 4  
 Object technology reuse (对象技术复用), 11  
 Objects (对象), finding of (寻找), 292-293  
 Objects (对象), referring to (引用)  
     equivalence (等价), 92-93  
     identification scheme (标识方案), 88-89  
     merger (合并), 90-92  
     name (名称), 86-87  
 Observation (观察)  
     active (临床的), 36, 49-50, 334  
     associated (关联的), 36, 50-51, 334  
     category (分类), 43-46  
     generally (通常), 42-43  
     pattern (模式), 333  
     phenomenon types (现象类型), 43  
     planning process (计划过程), use in (用在), 172-174  
     process of (过程), 36, 51-55, 334

- qualitative information (定性的信息), use in (用在), 35  
 qualitative measurements (定性的测量), 43  
 rejected (被否决的), 36, 48, 333  
 Observation concept (观察概念)  
 absence and presence (缺乏和存在), use in (用在), 46-47  
 control condition (控制条件), 53  
 subtyping (子类型化), 35, 46, 333  
 supertype of phenomenon (现象的超类型), 46  
 Odell, Jim  
 business modeling (业务建模), 10  
 power type (强力类型), 25-26  
 structural modeling technique (结构化建模技术), 323  
 type diagrams (类型图), 314  
 One-way association (单向关联), 277  
 One-way pricing (单向定价), 186, 187  
 OO  
 association interface (关联接口), 275  
 common methods (公共方法), 263  
 computer system (计算机系统), 85  
 databases (数据库), use in information systems (用在信息系统中), 244-245  
 generalization (泛化), 281  
 implementing associations (实现关联), 274  
 separation of responsibilities (职责的划分), 210  
 techniques (技术), 3, 10  
 type conversion (类型转换), 265  
 visibility (可见性), 235  
 OOPSLA, 历史 (history of), 5  
 Operating scope (操作范围)  
 defined (定义的), 17, 31  
 generally (通常), 30  
 model (模型), 31  
 pattern (模式), 332  
 type (类型), 32  
 Operational level (操作级)  
 knowledge (知识), link to (连接), 50-51  
 measurement (测量), 42  
 planning patterns (计划模式), 165  
 posting rules (记入规则), 116-118  
 Operations in application facades (应用外观中的操作), 264-265  
 Option (期权)  
 barrier (关卡), 211  
 Black-Scholes analysis (Black-Scholes 分析), 201-202  
 call and put (看涨 (期权) 和看跌 (期权)), 202-204  
 combination (合并), 206  
 compound (复合), defined (定义的), 204  
 event diagram (事件图), 201  
 generally (通常), 197, 200  
 Harel state chart (Harel状态图), 201  
 hedging (套利), 177-178  
 longs and shorts (多头和空头), 202-204  
 pattern (模式), 339  
 structure of (结构), 202  
 subtyping (子类型化), 204-205  
 Organization hierarchy (组织层次)  
 structure with explicit level model (具有显式层次模型的结构), 20  
 modeling with (建模), 17  
 pattern (模式), 331  
 supertype model (超类型模型), 20  
 two hierarchies modeled (两层结构建模的), 21  
 Organization structure (组织结构)  
 pattern (模式), 331  
 pattern requirement (模式需求), 17  
 rule (规则), addition of (加法), 21-23  
 typed relationship (类型化的关系), 21-22  
 Outcome functions (输出函数), 157, 172-174, 338  
 Output (输出)  
 account (账目), defined (定义的), 117  
 Total Telecommunications example (TT集团的例子),  
 use in (用在), 135  
 Overload (重载), 323

## P

- Package (包), 322。参见Trading packages
- Package diagrams (包图), 328-330
- Parallel application (并行应用)
- domain model (领域模型), visibility of (可见性), 221-222
  - exception handling (异常处理), 223
  - generally (通常), 198, 216-217
  - pattern (模式), 340
  - run-time attribute (运行时属性), 219-221
  - supertype encompassing interface (包含接口的超类型), 218-219
  - type checking (类型检查), 218
- Parameterized method (参数化方法), use in individual instance method (用在个体实例方法中), 109-110
- Parent-component association (父构件关联), use in protocol (用在方案中), 166
- Partition (划分), immutable (不可变的), 322
- Party (团体)
- accounts (账目), use in (用在), 125
  - address book model (通讯簿模型), 18, 19
  - contract package (合同包), relationship between (之间的关系), 231-232
  - defined (定义的), 17
  - pattern (模式), 331
  - post subtype (职位子类型), 32
- Party type (团体类型)
- generalizations (泛化), 17, 27-28, 332
  - single inheritance hierarchy (单继承层次结构), 27-28
- Pattern (模式)。参见Association patterns; Design templates; Planning
- Alexander, Christopher (亚历山大·克里斯托夫), 5, 6
  - analysis (分析), 310
  - categories of (分类), 8
  - defined (定义的), 8
  - domains (领域), outside (外部的), 9-10
  - fixed format (固定格式), 6
  - frameworks and (框架和), 11-13
  - history of (历史), 4-5
  - literary form (描述格式), 6-7
  - meta-model (元模型), 298
  - naming of (命名), 7
  - origins of (来源), 8-9
  - parts of (部分), 6
  - planning (计划), 165
  - portfolio (合同夹), history of, (历史) 7
  - table of (表), 331-342
  - use of (使用), 11-13
- Pattern Language of Programming (PLoP) conference (程序设计的模式语言 (PLoP) 会议), 5, 310
- Phenomenon type (现象类型)
- categories (分类), mapping from (映射来源), 43-45
  - measurement and (测量和), 41-42
  - measurement protocol (测量方案), 67
  - observation concept as a supertype (作为超类型的观察概念), 46
  - qualitative phenomena used to describe (描述使用的定性现象), 58
- Phenomenon with range (带范围的现象)
- attribute (属性), 58, 78-79
  - function (功能), 58, 80-81
  - generally (通常), 58, 77-78
  - pattern (模式), 335
- Phone (电话), setting up new service (建立新服务), 在 in Total Telecommunications example (TT集团的例子中), 138-141。参见Calls, phone
- Plan (计划), 157, 162-164, 338
- Planned status (计划的状况), 69-71
- Planning (计划)
- abandoned actions (放弃的动作), 157, 160-161
  - completed actions (完成的动作), 157, 160-161
  - generally (通常), 157-158
  - graphs (图), 用作 (use as), 166-168
  - implemented actions (实现的动作), 157, 158-160
  - outcome function (输出函数), 157, 172-174

- plan (计划), 157, 162-164  
 proposed actions (建议的动作), 157, 158-160  
 protocol (方案), 157, 165-168  
 resource allocation (资源分配), 157, 168-172  
 start function (启动函数), 157, 172-174  
 suspension (挂起), 157, 161-162
- Pattern Language of Programming conference (PLoP) (程序设计的模式语言 (PLoP) 会议), 5, 310
- Pointers used in bidirectional implementation (在双向实现中使用的指针), 278-279
- Polymorphism (多态性)
- account entries operation (账目条目操作), use in (用在), 137
  - measurement protocol (测量方案), use in (用在), 58, 71
  - subtype of detail account (细目账目的子类型), 136
- Portfolio (合同夹)
- browser (浏览器), 217
  - defined (定义的), 180
  - dynamic (动态的), with filters (带过滤器), 181
  - filters (过滤器), 181-184
  - generally (通常), 175
  - pattern (模式), 338
  - persistent (永久的), 184
  - transient (瞬时的), 184
- Portland Pattern Repository (波特兰模式库), 310
- Post (职位),
- generally (通常), 17
  - party subtype (团体子类型), 32-33
  - pattern (模式), 332
- Posting account (记入账目), 141
- Posting-rule-based firing (基于记入规则的触发), 113-114
- Posting rule execution (记入规则的执行)
- account-based firing (基于账目的触发), 112-113
  - backward-chained firing (向后链式触发), 114
  - firing approaches (触发方法), comparison of (对比), 114-115
- generally (通常), 96  
 eager firing (急切触发), 111-112  
 pattern (模式), 336  
 posting-rule-based firing (基于记入规则的触发), 113-114
- Posting rules (记入规则)
- each-entry (每个条目), 143
  - generally (通常), 96, 104-105
  - many accounts (多个账户), 96, 116-118, 336
  - pattern (模式), 336
  - reversibility of (可逆性), 105
  - structure of (结构), 151-152
  - transactions (事务), abandoning of (禁止), 105
  - transform (转换), 146-147
- Power type (强力类型), 25
- Practical context of patterns (模式的实践上下文), 8
- Practice (实践), accounting (账务), 119-122, 337
- Prerequisite (前提), 329-330
- Presence (存在)
- category observation (分类观察), 46
  - observation concepts (观察概念), 47
- Presentation and application logic (表示层和应用逻辑层),
- application logic tier (应用逻辑层), 246-247
  - applications (应用), building of (建立), 245
  - client/servers environments (客户/服务器环境), 250-251
  - generally (通常), 240, 255-256
  - logic split (逻辑分割), 249-250
  - matrix (矩阵), building of (建立), 246
  - pattern (模式), 341
  - presentation tier (表示层), 246
  - risk report facade (风险报告外观), 247-248
  - risk report presentation (风险报告表示), 247
  - visibilities between domains (领域之间的可见性), 248-249
- Primary party (主要团体), 178
- Principle of conservation (守恒原理), 99
- Private type (私有类型), 227
- Process of observation (观察的过程)

- abstraction (抽象), 54-55  
 generally (通常), 36, 51  
 pattern (模式), 334  
 trigger rule (触发器规则), 51-54
- Product (产品)**  
 combination (组合), common (公共的), 209  
 generally (通常), 205-206  
 link to contracts (到合同的连接), 210  
 pattern (模式), 197-198, 339  
 spread (差价), 207  
 straddle (约期套购), 205-206  
 subtyping (子类型化), 208
- Projection (推理)**, 36, 49, 334
- Proposed action (建议的动作)**  
 generally (通常的), 157  
 pattern (模式), 337  
 planning (计划), 158-160  
 resource allocation (资源分配), 168
- Protection proxy (保护代理)**, 279
- Protocol (方案)**。参见Measurement protocol  
 defined (定义的), 46  
 graphs (图), use as (用作), 166-168  
 multiple source (多重来源), 67  
 multiple visibility (多重可见性), use in (应用于), 228  
 mutual visibilities (相互可见性), 232  
 pattern (模式), 333, 338  
 planning (计划), 157, 165-168  
 observation (观察), 35-36  
 value of (数值), 46-47
- Proxy (代理), protection (保护)**, 279
- Public type (公有类型)**, 227
- Put (看跌 (期权))**, in option contract (在期权合同中), 202-204
- Q**
- Qualitative (定性的)**  
 measurements (测量), 43  
 phenomena (现象), 58
- Quantity (数量)**  
 attribute (属性), 97-98  
 defined (定义的), 35, 37  
 fundamental type (基础类型), 324  
 measurements as attributes (作为属性的测量), 37  
 monetary values (金钱数值), 37-38  
 object-oriented analysis (面向对象分析), use in (用在), 38  
 pattern (模式), 332  
 unit (单位), purpose of (目标), in association name (在关联名称中), 36
- Quote (报价)**  
 abstract (抽象), 187, 188  
 generally (通常), 175  
 number subtype (数字子类型), 186  
 one-way pricing (单向定价), 186, 187  
 pattern (模式), 338  
 two-way pricing (双向定价), 185, 187, 188
- R**
- Range (范围)**。参见Phenomenon with range  
 corporate finance (公司财务), use in (用在), 76-77  
 defined (定义的), 58  
 function (功能), 58, 80-81  
 fundamental type (基础类型), 324  
 pattern (模式), 335
- Ratio (比率), conversion (转换)**, 35, 38-39, 332
- Rational Software's Unified Modeling Language (UML)** (Rational 软件公司的统一建模语言 (UML)), 313-314
- Real numbers (实数)**, fundamental type (基础类型), 324
- Record (记录), obje information (对象信息)**, 35
- Recursive association (递归关联)**, 322
- Registrar objects (登记表对象), use of (使用)**, 293-294
- Rejected observation (被否决的观察)**, 36, 48, 333
- Relational technique (关系技术)**, 4
- Replacement (替换), use in generalization implementation (用在泛化实现中)**, 286
- Resource allocation (资源分配)**, 157, 168-172, 338

Retrieval method (检索方法), 260-261, 262, 263

Reuse of object technology (对象技术的复用), 11

Reversibility of posting rules (记入规则的可逆性), 105

Rule (规则), use in organization structure (用在组织结构中), 21-23

Rumbaugh, 211, 298-299

Run-time attribute (运行时属性), 219-221

## S

Scenario (场景)

building of (建立), 191-196

caching policy (缓存策略), 196

cross-rate element (交叉汇率元素), 192-194

defined (定义的), 180

derived issue (派生的问题), 192

elements (元素), calculation of (计算), 192

elements (元素), referencing of (引用), 193

generally (通常), 39, 175

hard-coding (硬编码), 194

interactive formula builder (交互式的公式生成器), 194

interpreter (解释器), 194

multiple access levels to a package (到一个包的多重访问层次), 226-230

pattern (模式), 339

sourced element (起源元素), 192

strengths of (长处/优势), 189-190

timepoint (时间点), adding to quote (加到报价上), 188-189

Schema (模式)。参见Three-tier architecture

Selector (选择器), contract (合同), 182-184

Semantic statements (语义声明), 321-323

Sequence (顺序), dependency of (依赖性), 162

Set (集合)

collections (聚合), 273

proposed plan actions (提议的计划动作), 165

Shlaer, 211, 295

Short (空头)

contracts (合同), use in (用在), 177-178

options (期权), use in (用在), 202-204

Simple interpreter (简单解释器), 67

Single classification (单分类), 319

Single delete (单个删除), 290

Single-valued mapping (单值映射)

association interface (关联接口), 275

category (分类), change to phenomenon type (变成现象类型), 44-45

pointers (指针), 278

structure (结构), use in implementation of (用在实现中), 137

type diagrams (类型图), use in (用在), 317

Singleton class (Singleton一类)

individual instance method (个体实例方法), 用在 (use in), 106-107

Total Telecommunications example (TT集团的例子), use in (用在), 145

type (类型), 322

Smalltalk

association interface (关联接口), 277

block method (块方法), 108

collections (聚合), 273

conceptual modeling (概念建模), use in (用在), 3

constraints (约束), implementation of (实现), 294

contracts (合同), use in (用在), 181

exception handling (异常处理), 223

information systems applications (信息系统应用), use in (用在), 244

object creation (对象创建), 289

quotes (报价), use in (用在), 186

Total Telecommunications example (TT集团的例子), use in (用在), 133

Software (软件)

implementation (实现), defined (定义的), 4

interface (接口), defined (定义的), 4

language (语言), 3

patterns (模式)。参见Pattern

protocol (方案), 166

- Source measurement protocol (源测量方案), 66  
 Split process rule (区分过程规则), 146  
 Spot contract (现货合同), 198  
 Spread (差价), defined (定义的), 207  
 Standard Template Library (STL) (标准模板库), 273  
 Star schema (星形模式)  
     defined (定义的), 60-61  
     focal event (焦点事件), 63  
 Start function (启动函数), 157, 172-174, 338  
 Status type (状态类型)  
     defined (定义的), 58  
     measurement protocol (测量方案), use in (用在), 69-72  
 Standard Template Library (STL) (标准模板库), 273  
 Storage schema (存储模式), 242  
 Storing entries (存储条目), 119  
 Straddle (约期套购), 205-206  
 State charts (状态图), conformance of (一致), 211-214  
 State diagrams (状态图), 327-328  
 Strategy pattern (策略模式), use in individual instance method (用在个体实例方法中), 107-108  
 Stretching of a facade (一个外观的伸展), 250-251  
 String (字符串)  
     fundamental type (基础类型), 324  
     identification scheme (标识方案), 88-89  
 Structural constraint (结构化约束), 170  
 Structural models (结构化模型), in Total Telecommunications example (在TT集团的例子中), 134-136  
 Structure (结构), implementation of (实现), in Total Telecommunications example (在TT集团的例子中), 137-138  
 Subclassing (子类化), 318  
 Subtype state machines (子类型状态机)  
     barrier option (关卡期权), 211  
     conformance (一致性), problems with using (使用的问题), 215-216  
     generally (通常), 198  
     pattern (模式), 339  
     state charts (状态图), conformance of (一致), 211-214  
     Subtyping (子类型化)  
         observation concept (观察概念), 35, 46, 333  
         packages (包), 233-234, 340  
         relationship between facades (外观之间的关系), 268  
     Summary account (汇总账目)  
         generally (通常), 96, 101-103  
         multiple (多重的), 127-129  
         pattern (模式), 336  
         posting rules for many accounts (多个账户的记入规则), use in (用在), 116-118  
     Superseding (替代), 85, 90-91  
     Supertype observation concept (超类型观察概念), 46  
     Suspension (挂起), 157, 161-162, 338  
     Symmetric property (对称属性), 125
- T
- Tax, calculation of (税, 计算), in Total Telecommunications example (在TT集团的例子中), 148-150  
 Telephone utility example (电话效用的例子). 参见Total Telecommunications (TT)  
 Templates (模板), design (设计), 137  
 Temporal resource (临时资源), 157  
 Tenor (期限), 197, 198-199  
 Text (文本), fundamental type (基础类型), 324  
 Three-schema architecture (三模式构架). 参见Three-tier architecture  
 Three-tier architecture (三层构架)  
     domain tier (领域层), location of (位于), 243-245, 255  
     generally (通常), 240, 242-243  
     pattern (模式), 341  
 Tilak chart (Tilak图), 64  
 Time (时间), fundamental type (基础类型), 324  
 Time period (时间段), fundamental type (基础类型), 324  
 Time reference (时间基准), fundamental type (基础类型), 324

- Timepoint (时间点), fundamental type (基础类型), 324
- Total Telecommunications (TT集团) (TT)
- account-based firing (基于账目的触发), implementation of (实现), 134, 143
  - accounting practice diagrams (账务实践图), 153-154
  - billing plan (收费方案), 133
  - calls (呼叫), setting up of (建立), 134, 142-143
  - framework (框架), 150, 152-153
  - generally (通常), 133-134
  - new phone service (新电话服务), 134, 138-141
  - posting rules (记入规则), 134, 151-152
  - separation of day and evening calls (区分白天和夜晚的呼叫), 134, 143-145
  - structural models (结构化模型), 134-136
  - structure (结构), implementation of (实现), 134, 137-138
  - tax (税), calculation of (计算), 134, 148-150
  - time (时间), charging for (按时间收费), 134, 145-148
- Trading (交易)
- contract (合同), 175, 176-180
  - portfolio (合同夹), 175, 180-184
  - quote (报价), 175, 185-188
  - scenario (场景), 175, 180, 188-196
- Trading packages (交易包)
- generally (通常), 225-226
  - multiple access levels (多重访问层次), 225, 226-230, 340
  - mutual visibility (相互可见性), 225, 230-233
  - private type (私有类型), 227
  - public type (公有类型), 227
  - subtyping packages (包的子类型化), 226, 233-234, 340
- Transaction (事务)。参见Multilegged transaction; Two-legged transaction
- abandoning of (抛弃), 105
  - generally (通常), 95-96, 98-99
  - pattern (模式), 336
- Transfer transaction (传输事务), 126
- Transform posting rule (转换记入规则), 146-147
- Transformation patterns (转换模式), defined (定义的), 271
- Transitivity property (传递属性), 125
- Trigger (触发/触发器)
- account (账目), 111-112
  - mapping (映射), 137
  - observation (观察), process of (过程), 51-54
- Total Telecommunications example (TT集团的例子), use in (用在), 135
- Two-dimensional history (二维历史), 298, 305-307
- Two-legged transaction (两腿事务)
- defined (定义的), 96
  - model (模型), 100
- Total Telecommunications example (TT集团的例子), use in (用在), 138, 139
- Two-tier architecture (两层构架), 240-242, 340
- Two-way pricing (双向定价), 185, 187, 188
- Type (类型)
- abstract (抽象), 322
  - associative (关联的), 297, 298-301, 342
  - checking (检查), 218, 276
  - conversion (转换), 265-267
  - instances (实例), 323
  - mapping (映射), 28
  - model (模型), 10。参见Design templates
  - overload (重载), 323
  - singleton, 322
- Type diagram (类型图)
- aggregation (聚集), 315-318
  - associations (关联), 315-318
  - attributes (属性), 315-318
  - class (类), 314-315
  - defined (定义的), 313
  - fundamental (基础的), 323-325
  - generalization (泛化), 318-321
  - rules and semantic statements (规则和语义声明), 321-

323

type (类型), 314-315

Typed relationship (类型化的关系), use in organization structure (用在组织结构中), 31-22

## U

Unidirectional associations (单向关联), 274-275, 278, 316

Unified Modeling Language (统一建模语言) (UML), 313-314

Uniqueness constraint (惟一性约束), use in identification scheme (用在标识方案中), 88-89

Unit (单位)

atomic (原子的), 39-41

compound (复合的), 35, 39-41, 332

conversion ratio used to convert (转换使用的转换率), 38-39

fundamental type (基础类型), 324

purpose of (目标), in association name (在关联名称中), 36

quantity combined with (与数量组合), 35

reference (引用), 40

Update method (更新方法), 261, 262, 263

## V

Validation method (确认方法), 261, 262, 263

Visibility (可见性)

generally (通常), 234-235

multiple (多重的), 227-228

mutual (相互的), 230-233, 340

package diagrams (包图), 329-330

subtyping (子类型化), 136, 233-234

Visitor pattern approach (访问者模式方法), 218

Visual Basic, software components (Visual Basic, 软件件), 11

## W

Whole value (完整的数值), quantity as (数量作为), 37

Wirfs-Brock solution (Wirfs-Brock解决方案), 227

World Wide Web site (万维网站点), 309-310

2772600

# 软件工程技术丛书书目

丛书编号	英文书名	中文书名	作者
I	Object-Oriented and Classical Software Engineering, 5E	面向对象与传统软件工程(原书第5版)	Stephen R. Schach
I	Object-Oriented Software Engineering	面向对象软件工程	Timothy C. Lethbridge
I	Software Engineering: A Practitioner's Approach, 5E	软件工程:实践者的研究方法(原书第5版)	Roger S. Pressman
I	Software Engineering, 6E	软件工程(原书第6版)	Ian Sommerville
I	Software Engineering with Java	软件工程:Java语言实现	Stephen R. Schach
I	Project-Based Software Engineering: An Object-Oriented Approach	基于项目的软件工程:面向对象研究方法	Evelyn Stiller
I	Software Engineering Economics	软件工程经济学	Barry W. Boehm
I	Software Cost Estimation With Cocoma II	用Cocoma II模型进行软件成本估算	Barry W. Boehm
I	Object-Oriented Software Construction, 2E	面向对象的软件结构(原书第2版)	Bertrand Meyer
I	Software for Use: A Practical Guide to The Models and Methods of Usage-Centered Design	面向使用的软件设计	Larry L. Constantine
I.1.1	Software Process Improvement: Practical Guidelines for Business Success	软件过程改进	Sami Zahran
I.1.1	Making Process Improvement Work	软件过程改进简明实践	Neil S. Potter
I.1.2	The Road to the Unified Software Development Process	统一软件开发之路	Ivar Jacobson
I.1.2	The Unified Software Development Process	统一软件开发过程	Jacobson/Booch/Rumbaugh
I.1.2	The Rational Unified Process: An Introduction , 2E	Rational统一过程引论(原书第2版)	Philippe Kruchten
I.1.2	UML and The Unified Process: Practical Object-Oriented Analysis & Design	UML和统一过程:实用面向对象的分析和设计	Jim Arlow
I.1.2	The Unified Process Inception Phase	统一过程初始阶段	Scott Ambler
I.1.2	The Unified Process Elaboration Phase	统一过程细化阶段	Scott Ambler
I.1.2	The Unified Process Construction Phase	统一过程构造阶段	Scott Ambler
I.1.2	The Unified Process Transition & Production Phase	统一过程移交和生产阶段	Scott Ambler
I.1.3	Managing Global Software Projects	全球化软件项目管理	Gopalaswamy Ramesh
I.1.3	Software Project Management: A Unified Framework	软件项目管理:一个统一的框架	Walker Royce
I.1.3	How to Run Successful Projects III: The Silver Bullet	成功的软件项目管理:银弹方案(原书第3版)	Fergus O'Connell
I.1.3	Successful Software Development, 2E	成功的软件开发(原书第2版)	Scott E. Donaldson
I.1.3	Six Sigma Software Development	六西格码软件开发	Christine B. Taynor
I.1.3	IT Project Management: On Track from Start to Finish	实用IT项目管理:从开始到结束的历程	Joseph Phillips
I.1.3	Successful IT Project Delivery: Learning the lessons of Project Failure	IT项目成功交付的秘诀	David Yardley
I.1.3	Software Project Management, 3E	软件项目管理(原书第3版)	Bob Hughes
I.1.3	Architecture-Centric Software Project Management	软件项目管理实用指南:以体系结构为中心	Daniel J. Paulish
I.1.3	Mentoring Object Technology Projects	对象技术项目管理	Richard T. Due
I.1.3	Virtual Project Management	虚拟项目管理	Paul E. McMahon
I.1.3	AntiPatterns and Patterns in Software Configuration Management	软件配置管理中的模式与反模式	William J. Brown
I.1.4	Handbook of Software Quality Assurance, 3E	软件质量保证(原书第3版)	Gordon G. Schulmeyer
I.1.4	Software Reliability Engineering.	软件可靠性工程	John Musa
I.1.4	Implementing ISO 9001:2000 The Journey from Conformance to Performance	2000版ISO 9001标准实施指南:从符合性到业绩改进	Tom Taormina
I.1.4	CMMI Distilled: A Practical Introduction to Integrated Process Improvement	CMMI精粹:集成化过程改进实用导论	Dennis M. Ahern

丛书编号	英文书名	中文书名	作者
1.4.1	Use Cases: Requirements in Context	用例:通过背景环境获取需求(原书第2版)	Daryl Kulak
1.4.1	Practical Software Requirements	实用软件需求	Benjamin L. Kovitz
1.4.2	Pattern-Oriented Software Architecture, Vol I: A System of Patterns	面向模式的软件体系结构 卷1:模式系统	Frank Buschmann
1.4.2	Pattern-Oriented Software Architecture, Vol II: Patterns for Concurrent and Networked Objects	面向模式的软件体系结构 卷2:用于并发和网络化对象的模式	Douglas Schmidt
1.4.2	Server Component Patterns	面向模式的软件体系结构 卷3:服务器组件模式——EJB描述的组件结构	Markus Volter
1.4.2	DesignPatterns: Elements of Reusable Object-Oriented Software	设计模式:可复用面向对象软件的基础	Gamma/Helm/Johnson /Vlissides
1.4.2	Patterns of Enterprise Application Architecture	企业级应用体系结构模式	Martin Fowler
1.4.2	Software Architecture: Organizational Principles and Patterns	软件架构:组织原则与模式	David Dikel
1.4.2	The UML Profile for Framework Architectures	框架体系结构的UML档案	Marcus Fontoura
1.4.2	Software Architect's Profession: An Introduction	软件架构师职业导读	Marc Sewell
1.4.2	Aspect-Oriented Programming With AspectJ	面向方面编程	Ivan Kiselev
1.4.2	The Art of Software Architecture	软件体系结构的艺术	Stephen T. Albin
1.4.2	Object-Oriented Reengineering Patterns	面向对象的再工程模式	Serge Demeyer
1.4.3	Building J2EE Applications With The Rational Unified Process	用RUP构建J2EE应用程序	Peter Eeles
1.4.3	Programming from Specifications	从规范出发的程序设计	Carroll Morgan
1.4.4	Testing IT: An Off-the-Shelf Software Testing Process	实用软件测试过程	John Watkins
1.4.4	Lessons Learned in Software Testing	软件测试经验与教训	Cem Kaner
1.4.4	Testing Computer Software: The Bestselling Software Testing Book Of All Time, 2E	计算机软件测试(原书第2版)	Cem Kaner
1.4.4	Software Testing in the Real World: Improving the Process	软件测试过程改进	Edward Kit
1.4.4	Effective Methods for Software Testing, 2E	软件测试的有效方法(原书第2版)	William E. Perry
1.4.4	Beta Testing for Better Software	软件Beta测试	Michael R. Fine
1.4.4	A Practical Guide to Testing Object-Oriented Software	面向对象的软件测试	John D. McGregor
1.4.4	Managing the Testing Process, 2E	软件测试过程管理(原书第2版)	Rex Black
1.4.4	Software Testing: A Craftsman's Approach, 2E	软件测试(原书第2版)	Paul C. Jorgensen
1.4.4	Just Enough Software Test Automation	软件测试自动化	Daniel J. Mosley
1.4.4	The Craft of Software Testing: Subsystem Testing, Including Object-Based and Object-Oriented Testing	软件子系统测试	Brian Marick
1.4.4	The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests	Web测试指南	Lydia Ash
1.5	Java Tools for Extreme Programming: Mastering Open Source Tools, Including Ant, JUnit, and Cactus	Java极限编程	Richard Hightower
1.5	Agile Software Development Ecosystems	敏捷软件开发生态系统	Tom DeMarco
1.5	Agile Modeling: Effective Practices For eXtreme Programming and The Unified Process	敏捷建模:极限编程和统一过程的有效实践	Scott W. Ambler
1.5	A Practical Guide to Feature-Driven Development	特征驱动开发方法:原理与实践	Steve R. Palmer
1.5	Pair Programming Illuminated	结对编程技术	Laurie Williams
1.5	Agile Management for Software Engineering	软件工程的敏捷管理	David Anderson

丛书编号	英文书名	中文书名	作者
1.1.4	CMM Implementation Guide	CMM实施与软件过程改进	Kim Caputo
1.1.4	Implementing the Capability Maturity Model	CMM实施指南	James R.Persse
1.1.4	Object-Oriented Defect Management of Software	面向对象的软件缺陷管理	Houman Younessi
1.1.4	Metrics and Models in Software Quality Engineering	软件质量工程:度量与模型	Stephen H. Kan
1.1.4	Performance Solutions	软件性能工程	Connie U. Smith
1.1.4	Peer Reviews in Software: A Practical Guide	软件同级评审	Karl E. Wiegers
1.1.5	Practical Software Measurement	实用软件度量	John McGarry
1.1.5	Software Metrics: A Rigorous and Practical Approach, 2E	软件度量(原书第2版)	Norman E. Fenton
1.1.5	Software Assessments, Benchmarks, and Best Practices	软件评估、基准测试与最佳实践	Capers Jones
1.2.1	The Object Primer: The Application Developer's Guide to Object Orientation and the UML, 2E	面向对象软件开发教程(原书第2版)	Scott W. Ambler
1.2.1	UML and C++: A Practical Guide to Object-Oriented Development, 2E	C++面向对象开发(原书第2版)	Richard C.Lee
1.2.1	Object-Oriented Methods: Principles & Practices, 3E	面向对象方法:原理与实践(原书第3版)	Ian Graham
1.2.1	Principles of Object-Oriented Software Development, 2E	面向对象软件开发原理(原书第2版)	Anton Eliëns
1.2.1	Object Solutions: Managing the Object-Oriented Project	面向对象项目的解决方案	Grady Booch
1.2.1	An Introduction To Object-Oriented Programming, 3E	面向对象编程导论(原书第3版)	Timothy Budd
1.2.1	The Unified Modeling Language User Guide	UML用户指南	Booch/Rumbaugh/Jacobson
1.2.1	The Unified Modeling Language Reference Manual	UML参考手册	Rumbaugh/Jacobson/Booch
1.2.1	Applying UML and patterns: An Introduction to Object-Oriented Analysis and Design, 1E	UML和模式应用(原书第1版)	Craig Larman
1.2.1	Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process , 2E	UML与模式应用(原书第2版)	Craig Larman
1.2.1	Object-Oriented Analysis and Design with Applications, 2E	面向对象分析与设计(原书第2版)	Grady Booch
1.2.1	Business Modeling with UML: Business Patterns at work	UML业务建模	Hans-Erik Eriksson
1.2.2	Software Reuse: Architecture, Process and Organization for Business Success	软件复用:结构、过程和组成	Ivar Jacobson
1.2.2	Software Reuse Techniques: Adding Reuse to the Systems Development Process	软件复用技术:在系统开发过程中考虑复用	Carma McClure
1.2.2	Practical Software Reuse: Strategies for Introducing Reuse Concepts in Your Organization	软件复用实践	Donald J. Reifer
1.2.2	Large-Scale Component-Based Development	大规模基于构件的软件开发	Alan W. Brown
1.2.2	Component-based Product Line Engineering with UML	基于构件的产品线工程:UML方法	Colin Atkinson
1.2.2	Business Component Factory	商业构件工厂	Peter Herzum
1.4.1	Object-Oriented Analysis & Design	面向对象的分析与设计	Andrew Haigh
1.4.1	Analysis Patterns: Reusable Object Models	分析模式:可复用的对象模型	Martin Fowler
1.4.1	Requirements Analysis and System Design: Developing Information Systems with UML	需求分析与系统设计	Leeszek A. Maciaszek
1.4.1	Systems Analysis and Design in a Changing World	系统分析与设计	John W. Satzinger
1.4.1	Advanced Use Case Modeling, Vol I: Software Systems	高级用例建模 卷I:软件系统	Frank Armour
1.4.1	Requirements Engineering: A Good Practice Guide	需求工程	Ian Sommerville
1.4.1	Software Requirements and Estimation	软件需求与估算	Swapan Kishore
1.4.1	Effective Requirements Practices	有效需求实践	Ralph R. Young
1.4.1	Applying Use Cases: A Practical Guide, 2E	用例分析技术(原书第2版)	Geri Schneider
1.4.1	Managing Software Requirements	软件需求管理:统一方法	Dean Leffingwell
1.4.1	Writing Effective Use Cases	编写有效用例	Alistair Cockburn
1.4.1	Problem Frames Analyzing and Structuring Software Development Problems	Problem Frames Analyzing and Structuring Software Development Problems	Michael Jackson