第7章 图

本章内容：

- 图、视图和模型
- 对系统的不同视图建模
- 对不同的抽象层次建模
- 对复杂视图建模
- 组织图和其他制品

当对一些事物建模时，为了更好地理解正在开发的系统，要对现实世界进行简化。使用UML，用类、接口、协作、构件、节点、依赖、泛化和关联等基本构造块建造模型。【在第1章中讨论建模。】

图是观察这些构造块的方式。图是一组元素的图形表示，经常把大多数图表示成顶点（事物）和弧（关系）的连通图。用图从不同的角度对系统进行可视化。因为没有哪个复杂的系统能仅从一个角度理解其全局，所以UML定义多种图，使你能分别独立地注重于系统的不同方面。

优秀的图使得正在开发的系统易于理解和处理。选择一组正确的图来对系统进行建模，能促使你对系统提出正确的问题，并有助于表明决策的含义。

7.1 入门

当你与建筑师一起设计一座房屋时，要从3个事物开始：需求的列表（如“需要有3个卧室的房屋”，“支出要少于*x*元”）；一些取自别的房屋，描述了一些关键特征（如“带有环行楼梯的入口的照片”）的简单草图或照片；一些关于风格的一般想法（如“我喜欢带有加利福尼亚海岸情调的法国乡村样式”）。建筑师的工作是把这些不完整的、不断变化的且可能是矛盾的需求转换成设计。

为了做到这些，建筑师可能要从一份基本的楼面布置蓝图开始。该制品为你和建筑师提供了将最终的房屋可视化的交流媒介，它描述了细节并把决策文档化。每一次审查都可能要做一些改变，如移动墙体、重新安排房间和放置门窗。早期这些蓝图会经常变化。随着设计的成熟，你越来越满意已做出的设计，这个设计最好地满足了所有的形式、功能、时间和金钱上的约束，这些蓝图将稳定下来，可用于构造房屋。即使在建造房屋的过程中，可能还要对某些图进行改变，也可能要创建一些新图。

进一步，你想要看到的将是房屋的一些视图，而不仅仅是楼面的布置图。例如，你可能想看从不同的侧面展示房屋的立体视图。为了使所付出的工作有意义，当你开始详述细节时，

建筑师需要制订电器方案、加热和通风方案以及上下水方案。如果你的设计要求某些不寻常的特征（如地下室无支撑的长跨距）或者你认为有一个特征对你来说很重要（如为了把家庭影院放在壁炉的附近，要考虑对壁炉的放置），则你和建筑师就要勾画一些强调这些细节问题的草图。

绘制一些图，从不同的角度对系统进行可视化，这种做法并不限于建筑业。在各种要创造复杂系统的工程学科中 —— 从民用工程到航空工程、造船业、制造业和软件业 —— 都使用这种方法。

在软件方面，对软件体系结构进行可视化、详述、构造和文档化，有5种最重要的互补视图：用况图、设计图、进程图、实现图和实施图。每一种视图都包含结构建模（对静态事物建模）和行为建模（对动态事物建模）。这些不同的视图一起捕获了系统的最重要的决策。每个视图都分别使你注重于系统的一个方面，从而使你能清楚地思考设计决策。【在第2章中讨论体系结构的5种视图。】

当用UML从任一角度观察软件系统时，要用图去组织感兴趣的元素。 UML定义了9种图，可以混合和组配它们来汇集成各种视图。例如，系统的实现视图的静态方面可以用构件图可视化，同一个实现视图的动态方面可以用交互图可视化。类似地，系统数据库的静态方面可以用类图可视化，动态方面可以用协作图可视化。【在第31章中讨论对系统的体系结构建模。】

当然，你可以不限于这9种图。在UML中，定义这9种图是因为它们表示了被观察元素的最常见的组合。为了适应项目或组织的需要，可以创建你自己所需种类的图，从而以不同的方式对UML元素进行观察。

你将以两种基本的方式使用UML的图：详述用于构造可执行系统的模型（正向工程）和从可执行系统部分地重新构造模型（逆向工程）。正像建筑师一样，无论哪一种方式，你都趋于增量（一次制作一部分）和迭代（重复地进行设计一点、建造一点的过程）地创建图。【在附录C中总结增量和迭代过程。】

7.2 术语和概念

系统（*system*）是一个为完成一定的目的而被组织起来的子系统的集合，这些子系统由一组可能来自不同观点的模型描述。*子系统*（*subsystem*）是一组元素的组合，其中的一些元素构成了其他被包含的元素所提供的行为规格说明。*模型*（*model*）是系统的语义闭合的抽象，这意味着它表示对现实的简化，这种简化既是完整的又是自相容的，这是为更好地理解系统而建立的。在体系结构的语境中，*视图*（*view*）是对系统模型的组织和结构的投影，注重于系统的一个方面。*图*（*diagram*）是一组元素的图形表示，通常表示成由顶点（事物）和弧（关系）组成的连通图。【在第31章中讨论系统、模型和视图。】

另一方面，系统表示了正在开发的事物，通过不同的模型从不同的角度对系统进行观察，并且以图的形式描述这些视图。

图正是对组成系统的元素的图形投影。例如，在一个社团人力资源系统的设计中，可能有几百个类。你总不能一开始就用一张包含所有的类和它们之间所有关系的大图，对系统的结构

和行为进行可视化。相反，要创建几张图，每张图注重于一个方面。例如，你可能发现一个类图，其中包含Person、Department和Office等类，这些类汇集在一起构成数据库模式。你可能在另一张图中看到这些类中的某些类以及另外的一些类，该图是表示由客户应用系统使用的API的。你也可能在交互图中看到一些同样的类，描述把一个 Person重新分配给一个新Department的事务的语义。

如本例所示，系统中的同一个事物（如类Person）可以在同一个图或不同的图中出现多次。在每种情况下，它都是同一个事物。每个图对组成系统的元素提供了一个视图。

对实际系统建模，无论问题域如何，你将会发现所建的图的种类都是一样的，这是因为它们把常见的视图表示成常见的模型。通常用下列4种图之一观察系统的静态部分。

1）类图
2）对象图
3）构件图
4）实施图

经常要用另外5种图观察系统的动态部分。

1）用况图
2）顺序图
3）协作图
4）状态图
5）活动图

UML定义了这9种图。

你创建的每一个图都很可能是这9种图之一，或偶尔是为你的项目或组织定义的另一种图。每个图都必须有一个在它的语境中唯一的名称，以便能够查阅到特定的图并能与其他的图相互区分。除了非常微小的系统外的任何系统，都要把图组织成包。【在第12章中讨论包。】

在UML的同一张图中，可以设计元素的任何组合。例如，可以在同一张图中显示类和对象（常见），或者可以在同一张图中显示类和构件（合法但不常见）。虽然没有什么能阻止你把不同种类的建模元素胡乱地放在同一张图中，但较为常见的是把种类大致相同的元素放于在同一张图中。事实上，UML定义的图是根据在图中最经常使用的元素来命名的。例如，如果要对一组类和它们之间的关系进行可视化，则要使用类图。类似地，如果要对一组构件进行可视化，则要使用构件图。

> **注释** 在实践中，你创建的所有图都是两维的，这意味着它们都是绘制在纸上、白板上、信封的背面或计算机显示器上的顶点和弧的平面图。 UML允许创建三维图，即有深度的图，允许在模型中“漫游”。一些虚拟现实研究组已经演示了这种UML的高级用法。

1. 结构图

现有的4种UML结构图可用于对系统的静态方面进行可视化、详述、构造和文档化。可以把系统的静态方面看作是对系统的相对稳定的骨架的表示。正如一所房屋的静态方面由墙、门、窗、管子、电线和通风管等事物的布局组成一样，软件系统的静态方面由类、接口、协作、构

件和节点等事物的布局组成。

UML的结构图大致上是围绕着建模时发现的几组主要事物来组织的。

1）类图　　类、接口和协作

2）对象图　对象

3）构件图　构件

4）实施图　节点

• 类图

类图（*class diagram*）显示了一组类、接口、协作以及它们之间的关系。在面向对象系统建模中类图是最常用的图。用类图说明系统的静态设计视图。包含主动类的类图用于表达系统的静态过程视图。【在第8章中讨论类图。】

• 对象图

对象图（*object diagram*）显示了一组对象以及它们之间的关系。用对象图说明在类图中所发现的事物实例的数据结构和静态快照。对象图也像类图那样表达系统的静态设计视图或静态进程视图，但它是从现实或原型方面来透视的。【在第14章中讨论对象图。】

• 构件图

构件图（*component diagram*）显示了一组构件以及它们之间的关系。用构件图说明系统的静态实现视图。构件图与类图相关，通常把一个构件映射到一个或多个类、接口或协作。【在第29章中讨论构件图。】

• 实施图

实施图（*deployment diagram*）显示了一组节点以及它们之间的关系。用实施图说明体系结构的静态实施视图。实施图与构件图相关，通常一个节点包含一个或多个构件。【在第30章中讨论实施图。】

注释　这4种图有一些常见的变体，这要根据它们的主要内容来命名。例如，为了说明在结构上把系统分解成子系统，可以创建子系统图。子系统图就是一个类图，其中主要包含子系统。

2. 行为图

UML的5种行为图用于对系统的动态方面进行可视化、详述、构造和文档化。可以把系统的动态方面看作是对系统变化部分的表示。正像房屋的动态方面是由气流和人在房间中走动组成一样，软件系统的动态方面也是由诸如随时间变化的信息流和在网络上构件的物理运动之类的事物组成。

UML的行为图大致上是按照你能对系统的动态进行建模的几种主要方式来组织的。

1）用况图　　组织系统的行为

2）顺序图　　注重于消息的时间次序

3）协作图　　注重于收发消息的对象的结构组织

4）状态图　　注重于由事件驱动的系统的变化状态

5）活动图　　注重于从活动到活动的控制流

• 用况图

用况图（*use case diagram*）描述了一组用况和参与者（一种特殊的类）以及它们之间的关系。可以用用况图描述系统的静态用况视图。用况图对于系统行为的组织和建模特别重要。【在第17章中讨论用况图。】

下述的两种图以及最后的两种图在语义上是等价的，这意味着可以用一种行为图对系统的动态建模，然后再把它转化为另一种图而不丢失信息。这可使你思考系统语义的不同方面。例如，你可能要先创建说明消息时间次序的顺序图，然后把顺序图转化为协作图，以便开发参加协作的对象类之间的结构关系（也能把协作图转化为顺序图）。类似地，可能开始用状态图说明由事件驱动的系统响应，然后把它转化为注重于控制流的活动图（也能把活动图转化为状态图）。UML提供这些在语义上等价的图的原因是：对系统动态建模显然是困难的，经常需要同时从多个角度去攻克难题。

交互图（*interaction diagram*）是顺序图和协作图的统称。所有的顺序图和协作图都是交互图；交互图要么是顺序图，要么是协作图。

- 顺序图

顺序图（*sequence diagram*）是强调消息的时间次序的交互图。顺序图显示了一组对象和由这组对象发送和接收的消息。对象通常是类的已命名的或匿名的实例，但也可以表示协作、构件和节点等事物的实例。顺序图用于说明系统的动态视图。【在第18章中讨论顺序图。】

- 协作图

协作图（*collaboration diagram*）是强调收发消息的对象的结构组织的交互图。协作图显示了一组对象、这组对象间的链以及这组对象收发的消息。对象通常是类的已命令的或匿名的实例，但也可以表示协作、构件和节点等事物的实例。协作图用于说明系统的动态视图。【在第18章中讨论协作图。】

注释 顺序图和协作图是同构的，这意味着可以把一种图转换为另一种图而没有信息损失。

- 状态图

状态图（*statechart diagram*）显示了一个由状态、转换、事件和活动组成的状态机。用状态图说明系统的动态视图。状态图对接口、类或协作的行为建模是非常重要的。状态图强调一个对象按事件次序发生的行为，这对于反应型系统建模特别有用。【在第24章中讨论状态图。】

- 活动图

活动图（*activity diagram*）显示了系统中从活动到活动的流。活动图显示了一组活动，从活动到活动的顺序的或分支的流，以及发生动作的对象或动作所施加的对象。用活动图说明系统的动态视图。活动图对系统的功能建模是非常重要的。活动图强调对象之间的控制流。【活动图是状态图的一个特例，这要在第19章中讨论。】

注释 图在本质上是静态制品（特别是把它们绘制在纸上、白板上或信封的背面时），用图说明一些固有的动态事物（系统的行为），明显存在着一些实际的限制。在计算机显示器上绘图，就有机会使行为图活动起来，从而能够模拟可执行的系统或者反应正执行的系统的实际行为。UML允许创建动态图，并且允许使用颜色和其他的可视提示去“运行”图。一些工具已经展示了UML的这种高级的用法。

7.3 普通建模技术

7.3.1 对系统的不同视图建模

当由不同的视图对系统建模时，实际上就是同时从多个维度构造系统。通过选择一组恰当的视图，你就设立了一个过程，该过程促使你对系统提出适当的问题并暴露出需要攻克的风险。如果选择视图的工作没有做好，或者以牺牲其他视图为代价只注重一个视图，就会冒掩盖问题以及延误解决问题（这里的问题是指那些最终会导致失败的问题）的风险。

为了由不同的视图对系统建模，要遵循如下的策略：

- 决定你需要哪些视图才能最好地表达系统的体系结构，并暴露项目的技术风险。早先描述的5种体系结构视图是一个好的开始点。
- 对这样的每种视图，决定需要创建哪些制品来捕获该视图的基本细节。在大多数情况下，这些制品由各种UML图组成。
- 作为你的过程策划的一部分，决定你将把哪些图置于某种形式或半形式的控制之下。对这些图要安排复审，并把它们作为项目的文档保存。
- 允许保留废弃的图。这些临时图仍然有助于探究决策的意图，也有助于用情况变化进行实验。

例如，对运行在单机上的一个简单的单片电路应用建模，可能仅需要以下几种图。

- 用况视图　　用况图
- 设计视图　　类图（用于结构建模）
  　　交互图（用于行为建模）
- 进程视图　　不需要
- 实现视图　　不需要
- 实施视图　　不需要

对于反应型系统或注重于进程流的系统，可能要包括状态图和活动图，分别用来对系统的行为建模。

类似地，对于客户/服务器系统，可能要包括构件图和实施图，用来对系统的物理细节建模。

如果是对复杂分布式系统建模，可能需要使用 UML的各种图，用于表达系统的体系结构以及项目的技术风险，这些图的列表如下，

- 用况视图　用况图
  　　活动图（对行为建模）
- 设计视图　类图（对结构建模）
  　　交互图（对行为建模）
  　　状态图（对行为建模）
- 进程视图　类图（对结构建模）
  　　交互图（对行为建模）
- 实现视图　构件图

- 实施视图 实施图

7.3.2 对不同的抽象层次建模

你不仅需要从不同的角度观察系统，而且你也将发现，与开发有关的人员也需要从这些角度观察系统，只是工作在不同的抽象层次上。例如，给定一组捕获了问题域词汇的类，编程人员可能需要其详细程度达到每个类的属性、操作和关系的详细视图。另一方面，浏览用况脚本的分析员和最终用户可能仅想对这些类做大概的了解。在这个语境中，编程人员工作在较低的抽象层次上，分析员和最终用户工作在较高的抽象层次上，但他们都是对同一个模型做工作。事实上，因为图只是组成模型的元素的图形表示，所以你可以对同一模型或不同的模型创建几种图，每种图掩盖或显露不同的元素集，并显示不同层次上的细节。

基本上有两种在不同的抽象层次上对系统建模的方法：通过针对同一模型，用不同层次上的细节描述图；或通过在不同的抽象层次上，用从一个模型跟踪到另一个模型的图来创建模型。

通过用不同层次上的细节描述图，在不同的抽象层次对系统建模，要遵循如下的策略：

- 考虑读者的需要，从给定的模型开始。
- 如果读者要用这个模型构造实现，就需要较低抽象层次的图，这意味着需要揭示许多细节。如果他们用这个模型向最终用户提供概念模型，就需要较高抽象层次的图，这意味着需要隐藏许多细节。
- 根据你在这个从低到高的抽象层次谱系中所处的位置，通过隐藏或揭示模型中的如下 4种事物而在恰当的抽象层次上创建图：
  1) 构造块和关系：把与图的内容无关的或读者不需要的构造块和关系隐藏起来。
  2) 修饰：仅显示对于理解意图是必要的构造块和关系的修饰。
  3) 流：在行为图的语境中，仅展开对于理解意图是必要的那些消息或转换。
  4) 构造型：在用于属性和操作等事物的分类列表的构造型的语境中，仅显示对于理解意图是必要的那些已被构造型化的项。

这种方法的主要优点是总可以从公共的语义库进行建模。这种方法的主要缺点是在一个抽象层次上的图发生变化可能会影响到不同抽象层次上的图。【在第15章中讨论消息，在第21章中讨论转换，在第6章中讨论构造型。】

通过创建不同的抽象层次上的模型，在不同的抽象层次上对系统建模，要遵循如下策略：

- 考虑读者的需要，决定每个读者要观察的抽象层次，在各层次上形成独立的模型。
- 一般来说，用简单的抽象在高的抽象层次开发模型，用详细的抽象在低的抽象层次开发模型。在不同模型的相关元素间建立跟踪依赖。【在第31章中讨论跟踪依赖。】
- 实际上，如果遵循体系结构的 5种视图，在不同的抽象层次上对系统建模时，存在 4种普通情景：
  1) 用况及其实现：用况模型中的用况要跟踪到设计模型中的协作。
  2) 协作及其实现：协作要跟踪到共同工作以完成协作的一组类。
  3) 构件及其设计：实现模型中的构件要跟踪到设计模型中的元素。
  4) 节点及其构件：实施模型中的节点要跟踪到实现模型中的构件。

【在第16章中讨论用况；在第27章中讨论协作；在第25章中讨论构件；在第26章中讨论节点。】

这种方法的主要优点是不同抽象层次上的图保持较松散的耦合。这意味着在一个模型中的变化对其他模型的直接影响较少。这种方法的主要缺点是必须花费资源以保持这些模型以及它们的图同步变化。特别是当模型在软件开发生命周期的不同阶段并存时（例如决定把分析模型与设计模型分别维护时），尤其如此。

例如，假设你在对一个Web商务系统建模，这样的系统的主要用况之一是订货。假如你是分析员或最终用户，你可能要在较高的抽象层次上创建一些显示订货动作的交互图，如图 7-1所示。【在第18章中讨论交互图。】

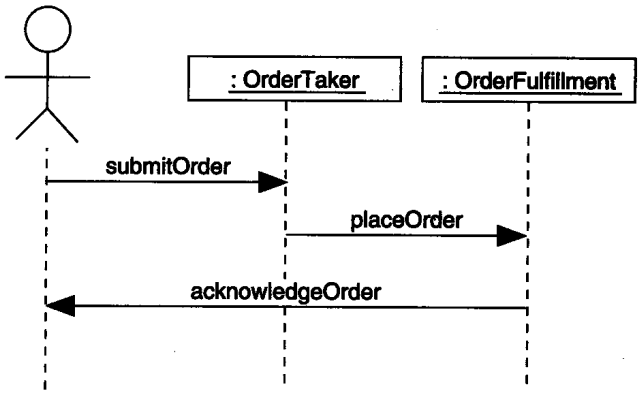

图7-1 在较高的抽象层次上的交互图

另一方面，负责实现这个脚本的编程人员必须在这个图的基础上进行构造，在这个交互图上要扩展一定的消息，并增加其他的扮演者，如图 7-2所示。

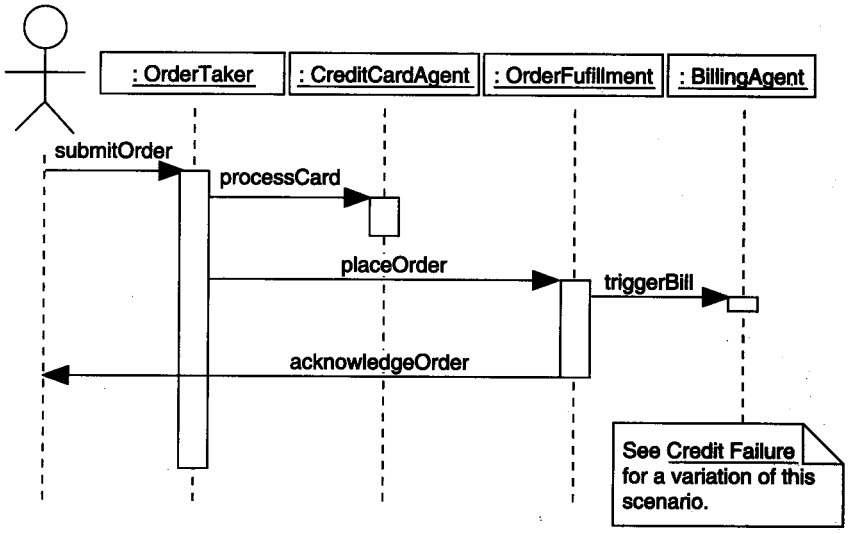

图7-2 在较低的抽象层次上的交互图

这两个图都针对同一个模型工作，但分别处于不同的细节层次。具有很多像这样的图是合理的，尤其是如果能用工具在各个图之间方便地导航，效果会更好。

7.3.3 对复杂视图建模

无论怎样分解模型，有时还会发现有必要创建大而复杂的图。例如，要分析一个由100多个抽象组成的数据库的完整模式，研究一个能显示所有的类及其关联的图，那确实是有价值的。这样做能够看到协作的公共模式。如果你是在较高的抽象层次上通过省略一些细节来显示这个模型，就会损失洞察这些细节的必要信息。

为了对复杂视图建模，要遵循如下的策略：

- 首先，要确信不存在任何有意义的方法，能用于在较高的抽象层次上表达这种信息，或许要省略一部分图，并抑制其他部分的细节。
- 如果已经隐藏了能够隐藏的细节，但图仍然是复杂的，就考虑把一些元素组织到一些包或者层次较高的协作中，然后仅把这些包或协作画在图中。【在第12章中讨论包；在第27章中讨论协作。】
- 如果图仍然是复杂的，就用注解和色彩作为可视化提示，以将用户的注意力引导到你所强调的重点上。
- 如果图还是复杂的，把它全部打印出来，挂在适宜的大墙上。虽然丧失了图的联机形式所带来交互性，但能够退后一步研究它的公共模式。

7.4 提示和技巧

当创建图时，要遵循如下的策略：

- 记住在UML中图的目的不是为了绘制漂亮的图画，而是为了进行可视化、详述、构造和文档化。图是一种用于最终部署可执行系统的手段。
- 不是所有的图都是值得保存的。通过对模型中的元素提出问题，考虑绘制一些草图，并用这些草图去思考正在构造的系统。很多这样的图达到其目的后就要被丢弃（但创建它们时所依据的语义仍然作为模型的一部分）。
- 避免产生无关的或冗余的图。这些图会使得模型混乱。
- 在每个图中只显示足以表达特定问题的细节。无关的信息会使读者把握不住你想要表达的要点。
- 另一方面，除非确实需要在很高的抽象层次上表达事物，否则不要使图过于简化。过分简化会隐藏对理解模型来说是重要的细节。
- 在系统中的结构图和行为图之间保持平衡。很少有哪个系统是完全静态的或完全动态的。
- 不要使图过大（大于若干打印页的图是很难浏览的），也不要使图过小（可考虑把几个小图合并成较大的图）。
- 给每个图一个能清楚地表达其意图的有意义的名称。

- 要对图进行组织。根据视图把它们组织到包中。
- 不要为图的格式所困扰。用工具来帮助工作。

一个结构良好的图，应满足如下的要求：

- 注重表达系统视图的一个方面。
- 仅包含对于理解这个方面所必须的元素。
- 提供的细节与它的抽象层次一致（仅显示对于理解这个方面所必须的修饰）。
- 不过分简化，以免读者误解重要的语义。

当绘制图时，要遵循如下的策略：

- 给图一个表达其用途的名称。
- 安排图中的元素，以尽量减少线段的交叉。
- 在空间上组织图的元素，以使得在语义上接近的元素在物理位置上也接近。
- 用注释和色彩作为可视化提示，以把注意力引导到图的重要特征上。