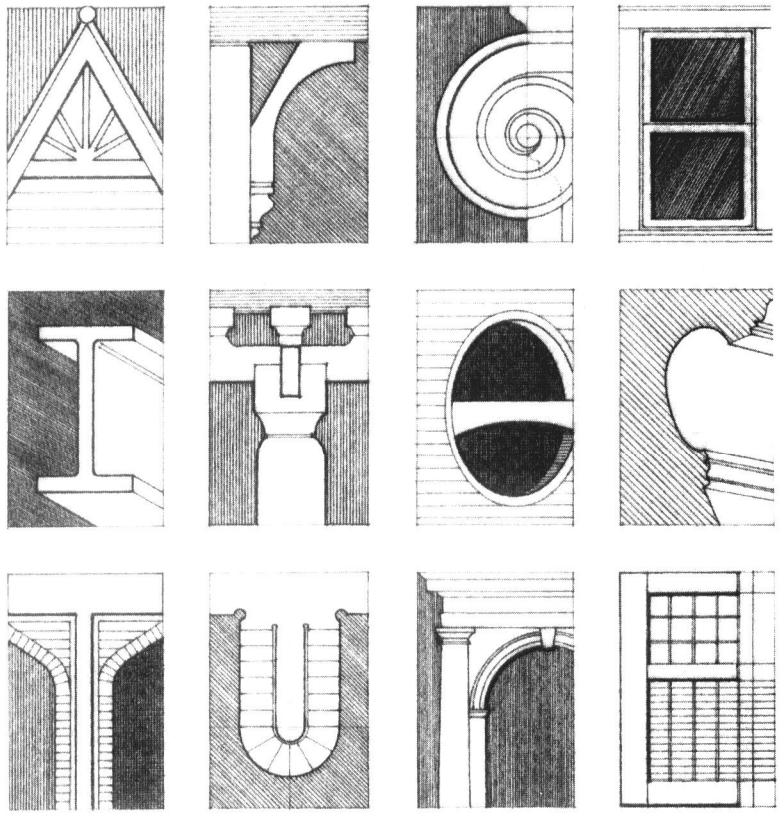
第一部分

入　门



第1章 为什么要建模

本章内容：

- 建模的重要性
- 建模的四个原理
- 软件系统的基本蓝图
- 面向对象的建模

成功的软件组织所发布的软件都应该是合格的，能够满足其用户的需要。如果一个软件组织能够及时并有远见地开发出这样的软件，并能够有效地利用人力和物力资源，那么这个软件组织就是可持续发展的。

在上段话里有一个重要含义：一个开发队伍的主要产品不应该是一堆漂亮的文档、世界级的会议、伟大的口号或几行获得普利策奖金的源代码，而应该是满足不断发展的用户和业务需要的优秀软件。其他的一切事情都是次要的。

不幸的是，很多软件组织把“次要的”和“不相关的”的含义搞混了。为了能得到满足特定功能的软件，你必须到用户中去，以一种科学的方式访问用户，这样才能得到你的系统的真实需求。为了能开发出具有持久品质保证的软件，你必须打好能适应变化的体系结构基础。为了能快速、有效地开发软件，并尽量减少软件废品和重复工作，你必须有合适的人员和合适的工具以及适当的工作重点。为了能一贯地、有预见性地做到这些，并使得在整个系统的生命期内花费合理，你必须有一个能适应业务和技术变化的合理的开发过程。

建模是开发优秀软件的所有活动中的核心部分，其目的是把所要设计的结构和系统的行为沟通起来，并对系统的体系结构进行可视化和控制。建模是为了更好地理解正在构造的系统，并经常提供简化和复用的机会。同时建模还可以管理风险。

1.1 建模的重要性

如果你想搭一个狗窝，你备好木料、钉子和一些基本工具（如锤子、锯和卷尺），就可以开始工作。从制定一点初步的计划到完成一个满足适当功能的狗窝，你可能不用别人帮助，在几个小时内就能够实现。只要狗窝够大且不太漏水，你的狗就可以安居。如果不制定一个计划，你总是可以返工，或是让你的狗受些委屈。

如果你想为你的家庭建造一所房子，你备好木料、钉子和一些基本工具，也能开始工作，但这将需要较多的时间，并且你家庭对于房子的需求肯定比狗对于狗窝的需求要多。在这种情况下，除非你曾经多次建造过房子，否则就需要事先制定出一些详细的计划，再开始动工，你

才能够成功。至少你应该绘制一些表明房子是什么样子的简图。如果你想建造一所能满足你家庭的需要并符合当地建筑规范的合格房屋，你就需要画一些建筑图，使你能想清楚房间的使用目的以及照明、取暖和水管装置等实际细节问题。作出这些计划后，你就能对这项工作所需的时间和物料作出合理的估计。你自己也可能建造出这样的房屋，但若有其他人协作（可能要将工程中的许多关键部分转包出去或购买预制的材料），效率就会高得多。只要你按计划行事，不超出时间和财务的预算，你的新房就可能非常令人满意。如果不制定计划，新房就不会完全令人满意。因此，最好在早期就制定计划，并谨慎地处理好所发生的变化。

如果你要建造一座高层办公大厦，若还是先备好木料、钉子和一些基本工具就开始工作，那将是非常愚蠢的。因为你所使用的资金可能是别人的，他们会对建筑物的规模、形状和风格作出要求。同时，他们经常会改变想法，甚至是在工程已经开工之后。由于失败的代价太高了，因此你必须要做大量的计划。负责建筑物设计和施工的组织机构是庞大的，你只是其中的一个组成部分。这个组织将需要各种各样的设计图和模型，以供各方相互沟通。只要你得到了合适的人员和工具，并对把建筑概念转换为实际建筑的过程进行积极的管理，你将会建成这座满足使用要求的大厦。如果你想继续从事建筑工作，那么你将一定要在使用要求和实际的建筑技术之间做好平衡，并且处理好组员们的休息问题，既不能把他们置于风险之中，也不能驱使他们过份辛苦地工作以至于精疲力尽。

奇怪的是，很多软件开发组织开始想建造一座大厦式的软件，而在动手处理时却好像他们正在仓促地造一个狗窝。

有时你是幸运的。如果你在恰当的时间有足够的合适人员，并且其他的事情都很如意，你可能（仅是可能）使你的团队推出一个令用户眼花缭乱的软件产品。然而，一般的情况是：你不能得到所有的合适人员（这样的人员经常供不应求），时间并不总是恰当的（昨天可能更好），其他的事情也并不尽如人意（常常由不得自己）。在因特网时代，对软件开发的要求正在日益增加，而开发团队却还是经常单纯地依靠他们唯一真正知道如何做好的一件事 —— 产生程序代码。英雄式的编程努力成为这一行业的传奇，人们似乎经常认为：更努力地工作是面对开发中出现的各种危机的正常反映。然而这未必能产生正确的程序代码，而且一些项目是非常巨大的，无论怎样延长工作时间，也不足以完成所需的工作。

如果你真正想建造一个相当于房子或大厦类的软件系统，问题可不是仅仅要写许多软件。事实上，关键是要编出正确的软件，并考虑怎样少写软件。要生产合格的软件就要有一套关于体系结构、过程和工具的规范。即使如此，很多项目开始看起来像狗窝，但随后发展得像大厦，原因很简单，它们是自己成就的牺牲品。如果对体系结构、过程或工具的规范没有作任何考虑，总有一天狗窝会膨胀成大厦，并会由于其自身的重量而倒塌。狗窝的倒塌可能会使你的狗恼怒，同理不成功的大厦将会对大厦的拥有者造成严重的影响。

不成功的软件项目失败的原因各不相同，而所有成功的项目的成功原因在很多方面都是相似的。一个成功的软件组织有很多成功的因素，其中共同的一点就是对建模的采用。

建模是一项经过检验并被广为接受的工程技术。我们建立的房屋和大厦的建筑模型能帮助用户得到实际建筑物的印像。为了分析大风或地震对建筑物造成的影响，我们甚至可以建立数学模型。

建模不只是用于建筑业。如果不首先构造模型（从计算机模型到物理风洞模型以及与实物大小一样的原型），就装配新型的飞机或汽车，那简直是难以想像的。为了更好地了解系统并与他人交流思想，开发新型的电气设备（从微处理器到电话交换系统）都需要一定程度的建模。在电影业里，剧本是产品的核心，这也是建模的一种形式。在社会学、经济学和商业管理领域中，为了证实理论或用最小限度的风险和代价实验新的理论，我们也要建模。

那么，模型是什么？简单地说，

模型是对现实的简化。

模型提供了系统的蓝图。模型既可以包括详细的计划，也可以包括从很高的层次考虑系统的总体计划。一个好的模型包括那些有广泛影响的主要元素，而忽略那些与给定的抽象水平不相关的次要元素。每个系统都可以从不同的方面用不同的模型来描述，因而每个模型都是一个在语义上闭合的系统抽象。模型可以是结构性的，强调系统的组织；它也可以是行为性的，强调系统的动态方面。

我们为什么要建模？一个基本理由是：

我们建模是为了能够更好地理解我们正在开发的系统。

通过建模，要达到四个目的：

1）模型帮助我们按照实际情况或按照我们所需要的样式对系统进行可视化。

2）模型允许我们详细说明系统的结构或行为。

3）模型给出了一个指导我们构造系统的模板。

4）模型对我们作出的决策进行文档化。

【在第2章中讨论UML如何完成这四项事情。】

建模并不只是针对大的系统。甚至像狗窝那样的软件也能从一些建模中受益。然而，可以明确地讲，系统越大、越复杂，建模的重要性就越大，一个很简单的原因是：

因为我们不能完整地理解一个复杂的系统，所以我们要对它建模。

人对复杂问题的理解能力是有限的。通过建模，缩小所研究问题的范围，一次只着重研究它的一个方面。这就是Edsger Dijkstra几年前讲的“各个击破”的基本方法，即先把一个要解决的难题划分成一系列小问题，解决了这些小问题也就解决了这个难题。此外，通过建模可以增强人的智力。一个适当选择的模型可以使建模人员在较高的抽象层次上工作。

任何情况下都应该建模的说法也未必尽然。事实上，一些研究指出：大多数软件组织没有做正规的建模，即使做了也很少。按项目的复杂性划分一下建模的使用情况，你还会发现：项目越简单，采用正规建模的就越少。

这里强调的是“正规化”这个词。虽然很不正规，但实际上，开发者甚至对一些非常简单的项目也要做一些建模工作。开发者可能在一块黑板上或一小片纸上勾画出他的想法，以对部分系统进行可视化表示，或者开发组可能使用CRC卡片描述一个剧本或机械设计。使用任何一种这样的模型都没有什么错。如果它能行得通，就有理由使用它。然而，这些非正规的模型经常是太特别了，它没有提供一种容易让他人理解的共同语言。建筑业、电机工程业和数学建模都有通用的建模语言，在软件开发中使用一种共同的建模语言进行软件建模也能使开发组织获益匪浅。

每个项目都能从一些建模中受益。即使在可随意使用软件的领域里，由于可视化编程语言

的效率，有时扔掉不适合的软件是更有效的，建模能帮助开发组更好地对系统计划进行可视化，这有助于他们正确地实施工作，使开发工作进展得更快。如果根本不建模，项目越复杂，就越有可能失败或做错事情。有一个自然趋势：随着时间的推移，所有引人关注的实用系统都变得越来越复杂。虽然你今天可能认为不需要建模，但随着你的系统的演化，你会对这个决定感到后悔，但那时为时已晚。

1.2 建模原理

各种工程学科都有其丰富的建模使用历史。这些经验形成了建模的四项基本原理，分别叙述如下。

第一，选择要创建什么模型对如何动手解决问题和如何形成解决方案有着意义深远的影响。

换句话说，就是要好好地选择模型。正确的模型将清楚地阐明难以对付的开发问题，提供不能轻易地从别处获得的洞察力；错误的模型将误导你，使你把精力花在不相关的问题上。

暂时先把软件问题放在一边，假设你现在正试图解决量子物理学上的一个问题。诸如光子在时空中的相互作用问题，其中充满了令人惊奇的大量的数学知识。选择一个不同于微积分的模型，所有的复杂问题一下子就变得容易解决了。在这个领域中，这恰恰是 Feynmann图的价值，它提供了对非常复杂问题的图形表示。类似地，在一个完全不同的领域里，假设你正在建造一座新建筑，你会关心疾风对它的影响。如果你建立了一个物理模型，并拿到风洞中去实验，虽然小模型没有精确地反应出大的实物，但你也可以从中找出一些有趣的东西。因此，如果你正在建立一个数学模型，然后去模拟，你将知道一些不同的东西；与使用物理模型相比，你也可能得到更多的东西。通过严格的持续的实验，你将更信任你已经建模的系统，事实上，它在现实世界中将像你所期望的那样工作得很好。

对于软件而言，你所选择的模型将在很大程度上影响你对世界的看法。如果你以数据库开发者的观点建造一个系统，你将可能注意实体 -关系模型，该模型把行为放入触发器和存储过程中。如果你以结构化开发者的观点建造一个系统，你将可能得到以算法为中心的模型，即从处理到处理的数据流。如果你以面向对象开发者的观点建造一个系统，你将可能得到这样一个系统，它的体系结构以众多的类及交互模式（描述了类间的协同工作）为中心。对于一个给定的应用系统和开发氛围，使用上述的任何一种方法都可能是正确的。经验表明，在需求易变的系统中面向对象的方法表现得更为出众，甚至对使用大型数据库或计算单元的系统也是如此。尽管事实如此，但要强调一点，不同的方法将导致不同种类的系统，并且代价和受益也是不同的。

第二，每一种模型可以在不同的精度级别上表示。

如果你正在建造一座大厦，有时你需要从宏观上让投资者看到大厦的样子，感觉到大厦的总体效果。而有时你又需要认真考虑细节问题，例如，对复杂棘手的管道的铺设，或对少见的结构件的安置等。

对于软件模型也是如此。有时一个快速简洁且是可执行的用户接口模型正是你所需要的，而有时你必须耐着性子对付比特，例如，描述跨系统接口或解决网络瓶颈问题就是如此。在任何情况下，最好的模型应该是这样的：它可以让你根据观察的角色以及观察的原因选择它的详

细程度。分析人员或最终用户主要考虑“做什么”的问题；开发人员主要考虑“怎样做”的问题。这两类人员都要在不同的时间以不同的详细程度对系统进行可视化。

第三，最好的模型是与现实相联系的。

如果一所建筑的物理模型不能以与真实的建筑相同的方式作出反映，则它的价值是很有限的。一架飞机的数学模型，如果只是假定了理想条件和完美制造，可能会掩盖一些潜在的、致命的现实特征。最好是有一个能够清晰地联系实际的模型，而当联系很薄弱时能够精确地知道这些模型怎样与现实相脱离。所有的模型都对现实进行了简化，但有一点要记住，不能简化掉任何重要的细节。

对软件领域结构化分析的唯一致命弱点是在分析模型和系统设计模型之间没有基本的联系。随着时间的推移，这个不可填充的裂缝会使系统构思阶段和实施阶段出现不一致。在面向对象的系统中，可以把各个几乎独立的系统视图连结成一个完整的语义整体。

第四，单个模型是不充分的。对每个重要的系统最好用一组几乎独立的模型去处理。

如果你正在建造一所建筑物，你会发现没有任何一套单项设计图能够描述该建筑的所有细节。至少你需要基础计划、电梯计划、电气计划、供热计划和水管装置计划。

在这里的重要短语是“几乎独立的”。在这个语境中，它意味着各种模型能够被分别进行研究和构造，但它们仍然是相互联系的。如同搞建筑一样，你能够单独地研究电气计划，但你也能看到它与之对照的基础计划，甚至它与水管装置计划中的管子排布的相互影响。

面向对象的软件系统也如此。为了理解系统的体系结构，你需要几个互补和连锁的视图：用况视图（揭示系统的需求）、设计视图（捕获问题空间和解空间里的词汇）、进程视图（对系统的进程和线程的分布建模）、实现视图（描述系统的物理实现）和实施视图（着重于系统的工程方面的组织）。每一种视图都可能有结构方面和行为方面。这些视图一起从整体上描绘了软件蓝图。【在第2章中讨论这5种视图。】

根据系统的性质，一些模型可能比另一些模型要重要。例如，对于数据密集型系统，表达静态设计视图的模型将占主导地位。对于图形用户接口密集型系统，静态和动态用况视图就显得相当重要。在实时系统中，动态进程视图尤为重要。在分布式系统中，例如Web密集型的应用，实现模型和实施模型是最重要的。

1.3 面向对象的建模

土木工程师建造了很多种模型。最普通的是这样的结构模型，它能帮助人们可视化系统，并能详细地说明系统的各部分以及各部分之间的相互关系。依据业务或工程中所着重关心的内容，工程师也可以建立动态模型，例如，为了帮助研究地震时的结构行为，就可以建立一种动态模型。各种模型的组织是不同的，各自都有所侧重之处。

对于软件，有几种建模的方法。最普通的两种方法是从算法的角度建模和从面向对象的角度建模。

传统的软件开发是从算法的角度进行建模，所有的软件都用过程或函数作为其主要构造块。这种观点导致开发人员把精力集中在控制流程和对大的算法进行分解上。除了用这种方法建立

的模型是脆弱的之外，采用这种方法没有其他本质上的害处。当需求发生变化以及系统增长时，用这种方法建造的系统就会变得难以维护。

现代的软件开发采用面向对象的角度进行建模，所有软件系统都用对象或类作为其主要构造块。简单地讲，通常要从问题空间或解空间的词汇中找出对象；类是对具有共同性质的一组对象的描述。每一个对象都有标识（你能够对它命名，以区别于其他对象）、状态（通常有一些数据与它相联系）和行为（使你能对该对象做某些事，它也能为其他对象做某些事）。

例如，可考虑把一个简单的计账系统的体系结构分成三层：用户接口层、中间件层和数据库层。在用户层，你将找出具体的对象，如按钮、菜单和对话框。在数据库层，你将找出具体的对象，如从问题域中找出描述实体的表，它包含顾客、产品和订单项。在中间件层，你将找出诸如交易、商业规则等对象，以及更高层次上的问题实体，如顾客、产品和订单。

可以肯定地说，面向对象方法是软件开发方法的主流部分，其原因很简单，因为事实已经证明，它适合于在各种问题域中建造各种规模程度和复杂度的系统。此外，当前的大多数程序语言、操作系统和工具在一定的方式上都是面向对象的，并给出更多按对象来观察世界的理由。面向对象的开发为使用构件技术（如 Java Beans或 COM+）装配系统提供了概念基础。

选择按面向对象的方式观察世界，会产生一系列的问题：什么是优秀的面向对象的体系结构？项目会创造出什么样的制品？谁创造它们？怎样度量它们？

对面向对象系统进行可视化、详述、构造和文档化正是统一建模语言（ UML）的目的。【在第2章中讨论这些问题。】