第 1 章　软件工程与 UML 概述

本章将对软件工程和 UML 进行简要的介绍，共分两节，每节介绍一个主题：软件工程概述、UML 概述。通过对本章的阅读，读者可以对软件和 UML 有一个总体的认识。

1.1　软件工程概述

1.1.1　软件工程的发展历史

从 20 世纪 60 年代中期到 70 年代中期，软件行业进入了一个大发展时期。这一时期软件作为一种产品开始被广泛使用，同时出现了所谓的软件公司。这一时期的软件开发方法仍然沿用早期的自由软件开发方式。但是随着软件规模的急剧膨胀，软件的需求日趋复杂，软件的性能要求相对变高，随之而来的软件维护难度也越来越大，开发的成本相应增加，导致失败的软件项目比比皆是，这样的一系列问题导致了“软件危机”。

1968 年，前北大西洋公约组织的科技委员会召集了一批一流的程序员、计算机科学家以及工业界人士共商对策，通过借鉴传统工业的成功作法，他们主张通过工程化的方法开发软件来解决软件危机，并冠以“软件工程”（Software Engineering）这一术语。30 余年来，尽管软件行业的一些毛病仍然无法根治，但软件行业的发展速度却超过了任何传统工业，并未出现真正的软件危机。如今软件工程已成了一门学科。

软件工程是一门建立在系统化、规范化、数量化等工程原则和方法上的，关于软件开发各个阶段的定义、任务和作用的工程学科。软件工程包括两方面内容：软件开发技术和软件项目管理。软件开发技术包括软件开发方法学、软件工具和软件工程环境；软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理和项目计划等。

1.1.2　软件工程的生命周期

软件开发是一套关于软件开发各阶段的定义、任务和作用的，建立在理论上的一门工程学科。它对解决“软件危机”，指导人们利用科学和有效的方法来开发软件，提高及保证软件开发的效率和质量起到了一定的作用。

经典的软件工程思想将软件开发分成以下 5 个阶段：需求捕获（Requirement Capture）阶段、系统分析与设计（System Analysis and Design）阶段、系统实现（System Implementation）阶段、测试（Testing）阶段和维护（Maintenance）阶段。

（1）需求捕获（Requirement Capture）阶段

需求捕获阶段就是通常所说的开始阶段。实际上，真正意义上的开始阶段要做的是选择合适的项目——立项阶段。其实，软件工程中的许多关于思想的描述都是通俗易懂的。立项阶段，顾名思义，就是从若干个可以选择的项目中选择一个最适合自己的项目的阶段。这个选择的过程是至关重要的，因为它将直接决定整个软件开发过程的成败。通常情况下，要考虑几个主要的因素：经济因素（经济成本、受益等）、技术因素（可行性、技术成本等）和管理因素（人员管理、资金运作等）。

在立项之后，才真正进入了软件开发阶段（当然，这里所说的是广义的软件开发，狭义的软件开发通常指的是编码）。需求捕获是整个开发过程的基础，也直接影响着后面的几个阶段的进展。纵观软件开发从早期纯粹的程序设计到软件工程思想的萌发产生和发展的全过程，不难发现，需求捕获的工作量在不断增加，其地位也随之不断提升。这一点可以从需求捕获在整个开发过程中所占的比例（无论是时间、人力，还是资金方面）不断地提高上可以看出。

（2）系统分析与设计（System Analysis and Design）阶段

系统分析与设计包括分析和设计两个阶段，而这两个阶段是相辅相成、不可分割的。通常情况下，这一阶段是在系统分析员的领导下完成的，系统分析员不仅要有深厚的计算机硬件与软件的专业知识，还要对相关业务有一定的了解。系统分析通常是与需求捕获同时进行，而系统设计一般是在系统分析之后进行的。

（3）系统实现（System Implementation）阶段

系统实现阶段也就是通常所说的编码（Coding）阶段。在软件工程思想出现之前，这基本上就是软件开发的全部内容，而在现代的软件工程中，编码阶段所占的比重正在逐渐缩小。

（4）测试（Testing）阶段

测试阶段的主要任务是通过各种测试思想、方法和工具，使软件的 Bug 降到最低。微软（Microsoft）宣称他们采用零 Bug 发布的思想确保软件的质量，也就是说只有当测试阶段达到没有 Bug 时他们才将产品发布。测试是一项很复杂的工程。

（5）维护（Maintenance）阶段

在软件工程思想出现之前，这一阶段是令所有与之相关的角色头疼的。可以说，软件工程思想很大程度上是为了解决软件维护的问题而提出的。因为，软件工程有 3 大目的——软件的可维护性、软件的可复用性和软件开发的自动化，可维护性就是其中之一，而且软件的可维护性是复用性和开发自动化的基础。在软件工程思想得到迅速发展的今天，虽然软件的可维护性有了很大的提高，但目前软件开发中所面临的最大的问题仍是维护问题。每年都有许多软件公司因为无法承担对其产品的高昂的维护成本而宣布破产。

值得注意的是，软件工程主要讲述软件开发的道理，基本上是软件实践者的成功经验和失败教训的总结。软件工程的观念、方法、策略和规范都是朴实无华的，一般人都能领会，关键在于运用。不可以把软件工程方法看成是“诸葛亮的锦囊妙计”——在出了问题后才打开看看，而应该事先掌握，预料将要出现的问题，控制每个实践环节，防患于未然。

1.2 建模的目的

在软件界有这么一条真理：一个开发团队首要关注的不应是漂亮的文档、世界级的会议、

响亮的口号或者华丽的源码，而是如何满足用户和项目的需要。

为了保证软件满足要求，开发组织必须深入到使用者中间了解对系统的真实需求；为了开发具有持久质量保证的软件，开发组织必须建立一个富有弹性的、稳固的结构基础；为了快速、高效地开发软件并使无用和重复开发最小化，开发组织必须具有精干的开发人员、正确的开发工具和合适的开发重点。为了实现以上要求，在对系统生存周期正确估计的基础上，开发组织必须具有能够适应商业和技术需求变化的健全的开发步骤。

建模是所有建造优质软件活动中的中心一环。本节主要介绍建模的优点、建模的重要性、建模中的 4 条原则和面向对象建模。

1.2.1 建模的重要性

1．一个揭示建模重要性的例子

如果你想给自己的爱犬盖个窝，开始的时候你的手头上有一堆木材、一些钉子、一把锤子、一把木锯和一把尺子。在开工之前只要稍微计划一下，你就可以几个小时之内，在没有任何人帮助的情况下盖好一座狗窝。只要它容得下你的爱犬、能遮风挡雨就可以了。就算差一点，只要你的狗不那么娇贵也是说得过去的。

如果你想为你的家庭建一座房子，开始的时候你的手头上也有一堆木材、一些钉子和一些基本的工具。但是这将要占用你很长的时间，因为你家庭成员的要求肯定要比你的狗高出很多。在这种情况下，除非你长期从事这项工作，否则最好在打地基之前好好地规划一下。首先，要为将要建造的房子设计一幅草图。如果想建造一座满足家庭需要的高质量的房屋，你需要画几张蓝图，考虑各个房间的用途以及照明取暖设备的布局。做好以上工作以后，你就可以对工时和工料做出合理的估计。尽管以人的能力可以独自盖一座房子，但是你会发现同其他人合作会更有效率，这包括请人帮忙或者买些半成品材料。只要坚持你的计划并且不超过时间和财力的限制，你的建造计划就成功了一多半。

如果你想建造一幢高档的写字楼，那么刚刚开始就准备好材料和工具是无比愚蠢的行为，因为你可能正在使用其他人的钱，而这些人将决定建筑物的大小、形状和样式。通常情况下，投资人甚至会在开工以后改变他们的想法，你需要做额外的计划，因为失败的代价巨大。你有可能只是很多个工作组之一，所以你的团队需要各种各样的图纸和模型以便同其他小组进行沟通。只要人员、工具配置得当，按照计划施工，你肯定会交付令人满意的工作。如果你想在建筑行业长久地干下去，你不得不在客户的需求和实际的建筑技术之间找到好的契合点。

2．软件的建模

许多软件开发组织总是像建造狗窝一样进行软件开发，而且他们还妄图开发出高质量的软件产品。这样的开发模式或许有些时候会奏效，有时候还可能开发出令用户赞叹的软件。但是，通常情况下都会失败。

如果你像盖房子或者盖写字楼一样开发软件，问题就不仅仅是写代码，而是怎么样写正确的代码和怎么样少写代码了。这就使得高质量的软件开发变成了一个结构、过程和工具相结合的问题。所以说，如果没有对结构、过程和工具加以考虑，所造成的失败是惨重的。每个失败的软件项目都有其特殊的原因，但是成功的项目在许多方面是相似的。软件组织获得成功的因素有很多，但是一个基本的因素就是对建模的使用。

3．模型的实质

那么模型究竟是什么？简而言之，模型是对现实的简化。

模型提供系统的蓝图，包含细节设计，也包含对系统的总体设计。一个好的模型包括重要的因素，而忽略不相干的细节。每一个系统可以从不同的方面使用不同的模型进行描述，因此每个模型都是对系统从语义上近似的抽象。模型可以是结构的、侧重于系统的组织，也可以是行为的、侧重于系统的动作。

4．建模的目标

建立模型可以帮助开发者更好地了解正在开发的系统。通过建模，要实现以下 4 个目标。

（1）便于开发人员展现系统。

（2）允许开发人员指定系统的结构或行为。

（3）提供指导开发人员构造系统的模板。

（4）记录开发人员的决策。

建模不是复杂系统的专利，小的软件开发也可以从建模中获益。但是，越庞大复杂的项目，建模的重要性越大。开发人员之所以在复杂的项目中建立模型，是因为没有模型的帮助，他们不可能完全地理解项目。

通过建模，人们可以每次将注意力集中在一点，这使得问题变得容易。这就是 Edsger Dijkstra 提出的“分而治之”的方法：通过将问题分割成一系列可以解决的、较小的问题来解决复杂问题。

5．通用建模语言的必要性

对比项目的复杂度会发现，越简单的项目，使用规范建模的可能性越小。实际上，即便是最小的项目，开发人员也要建立模型，虽然说很不规范。开发者可以在一块黑板或者一小片纸上概略地描述一下系统的某个部分，团队可以使用 CRC（类－责任－协作者模型）卡片来验证设计的可行性。这些模型本身没有任何错误，只要有用就尽可能地使用。但是这种不正规的模型通常情况下很难被其他开发者所共享，因为太有个性色彩了。正因为这样，通用建模语言的存在成为必然。

每个项目都可以从建模中受益。甚至在自由软件领域，模型可以帮助开发小组更好地规划系统设计，更快地开发。所有受人关注的有用的系统都有一个随着时间的推移越来越复杂的趋势，如果不建立模型，那么失败的可能性就和项目的复杂度成正比。

1.2.2 建模四原则

在工程学科中，对模型的使用有着悠久的历史，人们从中总结出了 4 条基本的建模原则。

（1）选择建立什么样的模型对如何发现和解决问题具有重要的影响。换句话说，就是认真选择模型。正确的模型有助于提高开发者的洞察力，指导开发者找到主要问题；而错误的模型会误导开发者将注意力集中在不相关的问题上。

（2）每个模型可以有多种表达方式。假设你正在建一幢高楼，有时你需要一张俯视图，以使参观者有一个直观的印象；有时你又需要认真考虑最低层的设计，例如铺设自来水管或者电线。

相同的情况也会在软件模型中出现。有时你想要一个快速简单的、可实行的用户接口模型；其他时候你又不得不进入底层与二进制数据打交道。无论如何，使用者的身份和使用的原因是

评判模型好坏的关键。分析者和最终的用户关心“是什么”，而开发者关心“怎么做”。所有的参与者都想在不同的时期、从不同的层次了解系统。

（3）最好的模型总是能够切合实际。一幢高楼的物理模型如果只有有限的几个数据，那么它不可能真实地反映现实的建筑；一架飞机的数学模型如果只考虑理想的飞行条件和良好的制造技术，那么很可能掩盖实际飞行中的致命缺陷。避免以上情况的最好办法就是让模型与现实紧密联系。所有的模型都是简化的现实，关键的问题是必须保证简化过程不会掩盖任何重要的细节。

（4）孤立的模型是不完整的。任何好的系统都是由一些几乎独立的模型拼凑出来的。就像建造一幢房子一样，没有一张设计图可以包括所有的细节。至少楼层平面图、电线设计图、取暖设备设计图和管道设计图是需要的。而这里所说的“几乎独立”是指每个模型可以分开来建立和研究，但是他们之间依然相互联系。就像盖房子一样，电线设计图可以独立存在，但是在楼层平面图甚至是管道图中仍然可以看到电线的存在。

1.2.3 面向对象建模

全世界的工程师建造了多种多样的模型，每一种模型建立的方式都是不同的，而且都有其侧重点。

在软件业中，建立模型的方法多种多样，两种最常用的方法是：基于算法方法建模和面向对象建模。

传统的软件开发采用基于算法的方法。在这种方法中，主要的模块是程序或者函数，这使得开发人员将注意力集中在控制流和将庞大的算法拆分成各个小块上。虽然说这种方法本身并没有错误，但是随着需求的变化和系统的增长，运用这种方法建立起来的系统很难维护。

现代的软件开发采用面向对象的方法。在这种方法中，主要的模块是对象或者类。对象通常是从问题字典或者方法字典中抽象出来的，类是对一组具有共同特点的对象的描述。每一个对象都有自己的标识、状态和行为。

比如考虑一个包含界面、中间层和数据库的简单的订货系统。在用户界面层上，有一些具体的对象，例如按钮、菜单以及对话框。在数据库中，也有一些具体的对象，例如包含客户、产品和订单信息的表。在中间层，存在如事务或交易的规则和客户、产品、订单等问题实体的高层视图。面向对象方法之所以是现在软件开发的主流，原因非常简单，因为它已经被证实在任何情况下都能很好的建模。而且，大多数现代的编程语言、操作系统和编程工具都是不同形式的面向对象的体现。

1.3 UML 概述

1.3.1 UML 的历史

面向对象的分析与设计（OOA&OOD）方法的发展在 20 世纪 80 年代末至 90 年代中出现了一个高潮，UML（Unified Modeling Language，统一建模语言）是这个高潮的产物。它不仅

统一了 Booch、Rumbaugh 和 Jacobson 的表示方法，而且在此基础上有了进一步的发展，并最终统一为大众所接受的标准建模语言。

公认的面向对象建模语言出现于 20 世纪 70 年代中期。从 1989～1994 年，其数量从不到 10 种增加到了 50 多种。在众多的建模语言中，语言的创造者努力宣传自己的产品，并在实践中不断完善。但是，使用面向对象方法的用户并不了解不同建模语言的优缺点及相互之间的差异，因而很难根据应用特点选择合适的建模语言，于是爆发了一场“方法大战”。20 世纪 90 年代中期，一批新方法出现了，其中最引人注目的是 Booch 1993、OOSE 和 OMT-2 等。

Booch 是面向对象方法最早的倡导者之一，他提出了面向对象软件工程的概念。1991 年，他将以前面向 Ada 的工作扩展到整个面向对象设计领域。Booch 1993 比较适合于系统的设计和构造。Rumbaugh 等人提出了面向对象的建模技术（OMT）方法，采用了面向对象的概念，并引入各种独立于语言的表示符。这种方法用对象模型、动态模型、功能模型和用例模型共同完成对整个系统的建模，所定义的概念和符号可用于软件开发的分析、设计和实现的全过程，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换。OMT-2 特别适用于分析和描述以数据为中心的信息系统。Jacobson 于 1994 年提出了 OOSE 方法，其最大特点是面向用例（Use Case），并在用例的描述中引入了外部角色的概念。用例的概念是精确描述需求的重要武器，但用例贯穿于整个开发过程，包括对系统的测试和验证。OOSE 比较适合支持商业工程和需求分析。此外，还有 Coad/Yourdon 方法，即著名的 OOA/OOD，它是最早的面向对象的分析和设计方法之一，该方法简单、易学，适合于面向对象技术的初学者使用，但由于该方法在处理能力方面的局限，目前已很少使用。

概括起来，首先，面对众多的建模语言，用户由于没有能力区别不同语言之间的差别，因此很难找到一种比较适合其应用特点的语言；其次，众多的建模语言实际上各有千秋；最后，虽然不同的建模语言大多雷同，但仍存在某些细微的差别，极大地妨碍了用户之间的交流。因此在客观上，有必要在精心比较不同的建模语言优缺点及总结面向对象技术应用实践的基础上，组织联合设计小组，根据应用需求，取其精华，去其糟粕，求同存异，统一建模语言。

1994 年 10 月，Grady Booch 和 Jim Rumbaugh 首先将 Booch 93 和 OMT-2 统一起来，并于 1995 年 10 月发布了第一个公开版本，称之为统一方法 UM 0.8（Unitied Method）。1995 年秋，OOSE 的创始人 Jacobson 加盟到这一工作中。经过 Booch、Rumbaugh 和 Jacobson 3 人的共同努力，于 1996 年 6 月和 10 月分别发布了两个新的版本，即 UML 0.9 和 UML 0.91，并将 UM 重新命名为 UML（Unified Modeling Language）。UML 的开发者倡议并成立了 UML 成员协会，以完善、加强和促进 UML 的定义工作。当时的成员有 DEC、HP、I-Logix、Itellicorp、IBM、ICON Computing、MCI Systemhouse、Microsoft、Oracle、Rational Software、TI 以及 Unisys。UML 成员协会对 UML 1.0 及 UML 2.0 的定义和发布起了重要的促进作用。

1.3.2 UML 包含的内容

首先，UML 融合了 Booch、OMT 和 OOSE 方法中的基本概念，而且这些基本概念与其他面向对象技术中的基本概念大多相同，因而，UML 必然成为这些方法以及其他方法的使用者乐于采用的一种简单一致的建模语言；其次，UML 不是上述方法的简单汇合，而是在这些方法的基础上广泛征求意见，集众家之长，几经修改而完成的，UML 扩展了现有方法的应用范

围；最后，UML 是标准的建模语言，而不是标准的开发过程。

作为一种建模语言，UML 的定义包括 UML 语义和 UML 表示法两个部分。

（1）UML 语义

描述基于 UML 的精确元模型定义。元模型为 UML 的所有元素在语法和语义上提供了简单、一致和通用的定义性说明，使开发者能在语义上取得一致，消除了因人而异的表达方法所造成的影响。此外 UML 还支持对元模型的扩展定义。

（2）UML 表示法

定义 UML 符号的表示法，为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。这些图形符号和文本所表达的是应用级的模型，在语义上它是 UML 元模型的实例。

1.3.3 UML 的定义

UML 是一种面向对象的建模语言。它的主要作用是帮助用户对软件系统进行面向对象的描述和建模（建模是通过将用户的业务需求映射为代码，保证代码满足这些需求，并能方便地回溯需求的过程）；它可以描述这个软件从需求分析直到实现和测试的开发全过程。UML 通过建立各种联系，如类与类之间的关系、类/对象怎样相互配合实现系统的行为状态等（这些都称为模型元素），来组建整个结构模型。UML 提供了各种图形，比如用例图、类图、时序图、协作图和状态图等，来把这些模型元素及其关系可视化，让人们可以清楚容易地理解模型，可以从多个视角来考察模型，从而更加全面地了解模型，这样同一个模型元素可能会出现在多个 UML 图中，不过都保持相同的意义和符号。

1．UML 的组成

UML 由视图（View）、图（Diagram）、模型元素（Model Element）和通用机制（General Mechanism）等几个部分组成。

视图（View）是表达系统的某一方面特征的 UML 建模元素的子集；视图并不是图，它是由一个或多个图组成的对系统某个角度的抽象。在建立一个系统模型时，通过定义多个反映系统不同方面的视图，才能对系统做出完整、精确的描述。

图（Diagram）是模型元素集的图形表示，通常是由弧（关系）和顶点（其他模型元素）相互连接构成的。UML 通常提供 9 种基本的图，把这几种基本图结合起来就可以描述系统的所有视图。

模型元素（Model Element）代表面向对象中的类、对象、接口、消息和关系等概念。UML 中的模型元素包括事物和事物之间的联系，事物之间的关系能够把事物联系在一起，组成有意义的结构模型。常见的联系包括关联关系、依赖关系、泛化关系、实现关系和聚合关系。同一个模型元素可以在几个不同的 UML 图中使用，不过同一个模型元素在任何图中都保持相同的意义和符号。

通用机制（General Mechanism）用于表示其他信息，比如注释、模型元素的语义等。另外，UML 还提供扩展机制（Extension Mechanism），使 UML 能够适应一个特殊的方法/过程、组织或用户。

UML 是用来描述模型的，通过模型来描述系统的结构或静态特征，以及行为或动态特征。为方便起见，用视图来划分系统各个方面，每一个视图描述系统某一方面的特征。这样一

个完整的系统模型就由许多视图来共同描述。

UML 中的视图大致可以分为如下 5 种。

（1）用例视图（Use Case View），强调从用户的角度看到的或需要的系统功能，是被称为“参与者”的外部用户所能观察到的系统功能的模型图。

（2）逻辑视图（Logical View），展现系统的静态或结构组成及特征，也称为结构模型视图（Structural Model View）或静态视图（Static View）。

（3）并发视图（Concurrency View），体现了系统的动态或行为特征，也称为行为模型视图（Behavioral Model View）或动态视图（Dynamic View）。

（4）组件视图（Component View），体现了系统实现的结构和行为特征，也称为实现模型视图（Implementation Model View）。

（5）配置视图（Deployment View），体现了系统实现环境的结构和行为特征，也称为环境模型视图（Environment Model View）或物理视图（Physical View）。

视图是由图组成的，UML 提供了 9 种不同的图。

（1）用例图（Use Case Diagram），描述系统功能。

（2）类图（Class Diagram），描述系统的静态结构。

（3）对象图（Object Diagram），描述系统在某个时刻的静态结构。

（4）时序图（Sequence Diagram），按时间顺序描述系统元素间的交互。

（5）协作图（Collaboration Diagram），按照时间和空间顺序描述系统元素间的交互和它们之间的关系。

（6）状态图（State Diagram），描述了系统元素的状态条件和响应。

（7）活动图（Activity Diagram），描述了系统元素的活动。

（8）组件图（Component Diagram），描述了实现系统的元素的组织。

（9）配置图（Deployment Diagram），描述了环境元素的配置，并把实现系统的元素映射到配置上。

提示：在 UML 2.0 中，将会提供 13 种图，本书将会在附录 A 中介绍。

2．UML 的建模机制

UML 有两套建模机制：静态建模机制和动态建模机制。静态建模机制包括用例图、类图、对象图、包、组件图和配置图。动态建模机制包括消息、状态图、时序图、协作图、活动图。

对于本节中的诸多概念，读者暂时只需了解即可，在后面的章节中将会结合实例进行详细的介绍。

1.3.4 UML 的应用领域

UML 的目标是以面向对象图的方式来描述任何类型的系统。其中最常用的是建立软件系统的模型，但它同样可以用于描述非软件领域的系统，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。

总之，UML 是一个通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行建模。此外，UML 适用于系统开发过程中从需求规格描述到系统完成后测试的不同阶段。

在需求分析阶段，可以用用例来捕获用户需求。通过用例建模，描述对系统感兴趣的外部角色及其对系统（用例）的功能要求。分析阶段主要关心问题域中的主要概念（如抽象、类和对象等）和机制，需要识别这些类以及它们相互间的关系，并用 UML 类图来描述。为实现用例，类之间需要协作，这可以用 UML 动态模型来描述。在分析阶段，只对问题域的对象（现实世界的概念）建模，而不考虑定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。这些技术细节将在设计阶段引入，设计阶段将为构造阶段提供更详细的规格说明。

编程（构造）是一个独立的阶段，其任务是用面向对象编程语言将来自设计阶段的类转换成实际的代码。在用 UML 建立分析和设计模型时，应尽量避免考虑把模型转换成某种特定的编程语言。因为在早期阶段，模型仅仅是理解和分析系统结构的工具，过早考虑编码问题十分不利于建立简单、正确的模型。

UML 模型还可作为测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试小组使用不同的 UML 图作为测试依据：单元测试使用类图和类规格说明；集成测试使用部件图和协作图；系统测试使用用例图来验证系统的行为；验收测试由用户进行，以验证系统测试的结果是否满足在需求捕获阶段所确定的需求。