第1章　面向对象概述

1.1　消费和经济

1.1.1　人类消费行为

鸡蛋营养又好吃，当我们想吃鸡蛋的时候，有两个方法，第一是养只小母鸡，花时间和精力养大它，让母鸡下蛋吃。第二是去市场买鸡蛋吃，这两种方法都是合理的，是不同的历史时期的两种常态，但成本是不一样的。

我们来设想一下传统的养鸡方法：当老太太发现有只母鸡抱窝时，找一个筐，铺上棉絮和稻草，做成一鸡巢，放在安全的地方；再找十几个新鲜的鸡蛋，把老母鸡抱过来，经过大约21天，由老母鸡完成小鸡的孵化；接下来，老母鸡教会小鸡觅食的技巧，小鸡学会独立觅食直到长大，长大后的小母鸡可以下蛋，于是有了鸡蛋吃。

老太太做的事情，只是利用了鸡的成长过程：如果不用新鲜的鸡蛋，可能孵不出小鸡；若鸡蛋不够十几个，则孵化效率太低；若鸡蛋数量超过十几个，母鸡照顾不过来，孵化率也会变低；筐子是为了保持温度和湿度，并方便母鸡觅食和返回；安全的地方则是为了保护老母鸡不被黄鼠狼等天敌吃掉。

小鸡孵化出来，则面临着一系列磨难：被小孩不小心踩死、喝雨水拉肚子而死、被小动物伤害而死、得鸡瘟而死等。自然成长的鸡，存活率并不高。我们可以想象，以这样效率和人工成本，现代人很难用养鸡卖鸡蛋的钱养活自己的。

1.1.2　设计、重用和省钱

在中国，现代化大型鸡场的规模都在10万只以上，本质上和老太太做的事是一样的，但要复杂得多，要解决很多老太太想不到的问题如：通风、光照、消毒、环境污染等。要管理好大型鸡场，有10个问题需要专业处理，这10个问题是：科学选择场址、合理划分区域、精选优良品种、科学建设鸡舍、强化人员管理、健全规章制度、注重档案管理、实施科学防疫、规范粪便处理和配备养殖设施。

以上10个问题的任何一个，又涉及更细的问题，以最后一个配置养殖设施为例：有条件的话，尽量使用先进的自动饮水系统、自动喂料系统、光照设备、储料罐或储料库、风机、湿帘降温机械通风系统等先进生产设施，以充分发挥家禽生产水平。同时健全防疫配套设施，有

围墙、绿化、沟河等防疫隔离带。配有预防鼠害、鸟害设施和专门的兽医室。

如何让养鸡场高效科学的运作，是一个系统化工程。有很多工作需要人去发现和解决，因此，梳理管理流程、明确人的职责是一个非常重要的问题，而这个工作要通过业务建模来完成。高效的流程不但能理顺工作，而且能找出一些工作量相对大，工作内容相对简单的部分，由机器来完成，从而相对地减少人的工作强度。业务建模不但可以找到未来要开发的计算机系统的需求，还会一直发现新问题，这是一个持续的改进过程，具体方法见第二章。

我们再分析这两种鸡蛋来源。首先看老太太养鸡的方法：如果老太太的鸡蛋自己吃，养鸡就是副业；老太太的主要任务是带带孙子，种种菜，做做家务；养鸡工作都可以在劳动之余完成，怡情而又受益；放养的鸡也可以自己找食吃，到了晚上，鸡可以飞到树枝上去过夜。在大多数人的生活半径为 20 公里的社会，这种方式自然是一种很好的模式；如果场景转为现代社会，为了让更多的人吃到鸡蛋，养鸡不再是副业，而是谋生的唯一手段，那么养鸡者就要考虑市场和效率。假定养鸡规模扩大到老太太的 1000 倍以上，则树枝栖息显然不行：要建鸡舍；自然觅食也没有保证，就要增加饲料；另外要避免鸡瘟等传染病、鸡蛋的存储和运输等一系列问题，这不仅是投入的固定成本大幅度提升的问题，而是一个复杂的管理难题和科学的设计问题。

大规模的养鸡场怎么降级成本呢？我们来分析一下，无论是土地选用、鸡舍构造、养鸡器具（如料桶、水管、喷雾器、铁锹、蛋筐）的设计、鸡苗培育、饲料配方、药品疫苗、照明，目前都有专业的做法，而这些做法都来自工作经验和科学技术发展的长期积累，因此才有了价格的降低；把人类积累的知识和经验合理的嵌入工作流程，是降低成本的根本原因。而这个思维的过程就是设计，由此我们可以得出这样的结论：

- 设计解决省钱的问题。
- 设计省钱的主要原因是重用了前人的知识和经验。

除了吃鸡蛋不需要养鸡，喝牛奶不需要养牛外，生活中通过设计和重用来降低成本的例子比比皆是。只要和商品打交道，就能看到设计和重用的应用。

善于从其他学科学习知识和经验的软件业，把设计和重用的理念用到了极致，可以说：软件是对人类解决问题逻辑的复用。

1.2 软件复用

1.2.1 软件和代码复用

机器语言是指一台计算机全部的指令集合，是第一代计算机语言。

电子计算机所使用的是由“0”和“1”组成的二进制数，二进制是计算机语言的基础。计算机发明之初，人们只能用计算机的语言去命令计算机完成特定任务，一句话，就是写出一串串由“0”和“1”组成的指令序列交由计算机执行，这种计算机能够认识的语言，就是机器语言。使用机器语言是十分痛苦的，特别是在程序有错需要修改时，更是如此。

机器程序就是一个个的二进制文件，由一条条的机器指令组成。机器指令是不可分

割的最小功能单元。由于不同类型的计算机的指令系统各不相同，因此在某一类型计算机上执行的程序，要想在另一类型计算机上执行，必须另编程序；实际上，另编程序存在大量的重复性工作，由于使用的是针对特定型号计算机的语言，运算效率却是所有语言中最高的。

为了减轻使用机器语言编程的痛苦，人们进行了一种有益的改进：用一些简洁的英文字母、符号串来替代特定指令的二进制串，比如，用“ADD”代表表示加法的 01 组合编码，“MOV”代表数据传递的 01 组合编码等等，这样一来，程序员不用记忆就可以很容易读懂并理解程序在干什么，纠错及维护变得方便了，这种程序设计语言就称为汇编语言，也称为第二代计算机语言。

计算机是不认识这些符号的，这就需要一个专门的程序，专门负责将这些符号翻译成二进制的机器语言，这种翻译程序被称为汇编程序。

汇编语言同样十分依赖于机器硬件，虽然移植性仍然不好，但效率却十分高。针对计算机特定硬件编制的汇编语言程序，能准确发挥计算机硬件的功能和特长，程序精炼而质量高，所以至今仍是一种常用而强有力的软件开发工具。

汇编语言和机器语言本质是相同的，都是直接对硬件操作，只不过指令采用了英文缩写的标识符，更容易识别和记忆。它同样需要编程者将每一步具体的操作用命令的形式写出来。

汇编程序的每一句指令只能对应实际操作过程中的一个很细微的动作如移动、自增等，因此汇编源程序一般比较冗长、复杂、容易出错，而且使用汇编语言编程需要有更多的计算机硬件知识，但汇编语言的优点也是显而易见的，用汇编语言所能完成操作的效率不是一般高级语言所能做到的，而且汇编语言程序经汇编生成的可执行文件不仅比较小，而且执行速度很快。

高级语言的“高级”二字，相对于汇编语言而言，它是较接近自然语言和数学公式的编程语言，基本脱离了机器的硬件系统，用人们更易理解的方式编写程序。编写的程序称之为源程序。高级语言并不是特指的某一种具体的语言，它包括很多编程语言，如流行的 Java，C，C++，C#，Pascal，Python，Lisp，Prolog，FoxPro，易语言，中文版的 C 语言“习语言”等，这些语言的语法、命令格式都不相同。

高级语言与计算机的硬件结构及指令系统无关，它有更强的表达能力，可方便地表示数据的运算和程序的控制结构，能更好地描述各种算法，而且容易学习掌握。但高级语言编译生成的程序代码一般比用汇编程序语言设计的程序代码要长，执行的速度也慢。所以汇编语言适合编写一些对速度和代码长度要求高的程序或直接控制硬件的程序。

高级语言程序“看不见”机器的硬件结构，不能用于编写直接访问机器硬件资源的系统软件或设备控制软件。为此，一些高级语言提供了与汇编语言之间的调用接口。用汇编语言编写的程序，可作为高级语言的一个外部过程或函数，利用堆栈来传递参数或参数的地址。

从机器语言到高级语言的抽象，带来的主要好处是：

高级语言接近算法语言，易学、易掌握、一般工程技术人员只要几周时间的培训就可以

胜任程序员的工作；高级语言为程序员提供了结构化程序设计的环境和工具，使得设计出来的程序可读性好、可维护性强、可靠性高；高级语言远离机器语言，与具体的计算机硬件关系不大，因而所写出来的程序可移植性好，重用率高；由于把繁杂琐碎的事务交给了编译程序去做，所以自动化程度高，开发周期短，且程序员得到解脱，可以集中时间和精力去从事对于他们来说更为重要的创造性劳动，以提高程序的质量。

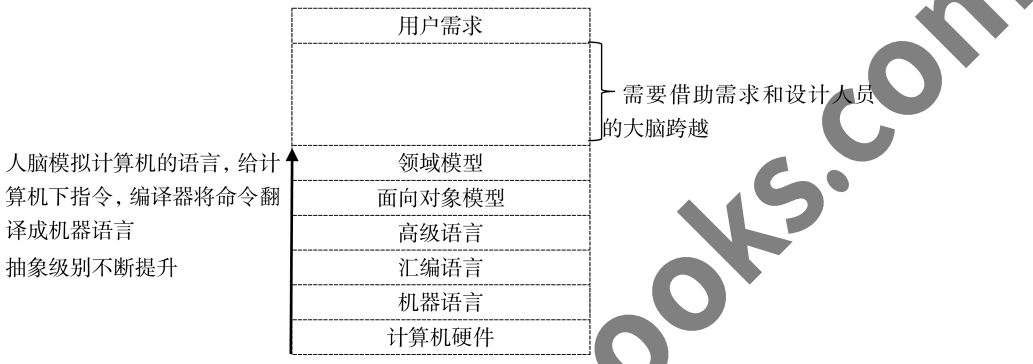

图 1.1 需求和代码重用

结构化高级语言的诞生，使程序员可以离开机器层次，在更抽象的层次上表达意图。由此诞生的三种重要控制结构，以及一些基本数据类型都很好地让程序员以接近问题本质的方式去思考和描述问题。

随着程序规模的不断扩大，20 世纪 60 年代末出现了软件危机。当时的程序设计模型，无法克服错误随着代码量的扩大而级数般扩大的问题，这时候诞生了一种新的思考程序设计的方式和模型——面向对象程序设计。由此也诞生了一批支持此技术的程序设计语言如 C++、Java，这些语言都以新的观点去看待问题，即问题就是由各种不同属性的对象以及对象之间的消息传递构成。面向对象语言由此必须支持新的程序设计技术，例如：数据隐藏、数据抽象、用户定义类型、继承、多态等。

从以上语言的每一步发展，都涉及大量的软件复用和代码复用。

软件复用就是将已有的软件成分用于构造新的软件系统。可以被复用的软件成分一般称作可复用构件，无论对可复用构件原封不动地使用还是做适当的修改后再使用，只要是用来构造新软件，则都可称作复用。软件复用不仅仅是对程序的复用，它还包括对软件生产过程中任何活动所产生的制成品的复用，如项目计划、可行性报告、需求定义、分析模型、设计模型、详细说明、源程序、测试用例等。在一个系统中多次使用一个相同的软件成分的复用，称作共享；对一个软件进行修改，使它运行于新的软硬件平台的复用，称作软件移植。

目前，最有可能产生显著效益的复用是对软件生命周期中一些主要开发阶段的软件制品的复用，按抽象程度的高低，可以划分为如下的复用级别：代码复用、设计复用、分析复用、测试信息复用等，前面的简要计算机语言史其实就是代码复用的历史，在高级语言转化成机器语言的过程中，软件工具和编译程序做了大部分代码复用的工作，这些基础性的工作对程序员进行了隐藏。

平时所说的代码复用多站在程序员的角度来看，代码的复用包括目标代码和源代码的复用。其中目标代码的复用级别最低，历史也最久，当前大部分编程语言的运行支持系统都提供了连接(Link)、绑定(Binding)等功能来支持这种复用。源代码的复用级别略高于目标代码的复用，程序员在编程时把一些想复用的代码段复制到自己的程序中，但这样往往会产生一些新旧代码不匹配的错误。想大规模的实现源程序的复用只有依靠含有大量可复用构件的构件库。

1.2.2　设计复用

设计结果比源程序的抽象级别更高，因此它的复用受实现环境的影响较少，从而使可复用构件被复用的机会更多，并且所需的修改更少。这种复用有三种途径，第一种途径是从现有系统的设计结果中提取一些可复用的设计构件，并把这些构件应用于新系统的设计；第二种途径是把一个现有系统的全部设计文档在新的软硬件平台上重新实现，也就是把一个设计运用于多个具体的实现；第三种途径是独立于任何具体的应用，有计划地开发一些可复用的设计构件，如 DLL 文件(Dynamic Linkable Library，动态链接库)等。

1.2.3　分析复用

这是比设计结果更高级别的复用，可复用的分析构件是针对问题域的某些事物或某些问题的抽象程度更高的解法，受设计技术及实现条件的影响很少，所以可复用的机会更大。复用的途径也有三种，即从现有系统的分析结果中提取可复用构件用于新系统的分析；用一份完整的分析文档作输入产生针对不同软硬件平台和其他实现条件的多项设计；独立于具体应用，专门开发一些可复用的分析构件。

1.2.4　测试信息复用

主要包括测试用例的复用和测试过程信息的复用。前者是把一个软件的测试用例在新的软件测试中使用，或者在软件做出修改时在新的一轮测试中使用。后者是在测试过程中通过软件工具自动地记录测试的过程信息，包括测试员的每一个操作、输入参数、测试用例及运行环境等一切信息。这种复用的级别，不便和分析、设计、编程的复用级别作准确的比较，因为被复用的不是同一事物的不同抽象层次，而是另一种信息，但从这些信息的形态看，大体处于与程序代码相当的级别。

由于软件生产过程主要是正向过程，即大部分软件的生产过程是使软件产品从抽象级别较高的形态向抽象级别较低的形态演化，所以较高级别的复用容易带动较低级别的复用，因而复用的级别越高，可得到的回报也越大，因此分析结果和设计结果在目前很受重视。用户可购买生产商的分析件和设计件，自己设计或编程，掌握系统的剪裁、扩充、维护、演化等活动。

复用能带来较大的经济利益，从计算机诞生之日起，复用就一直是研发人员追求的目标之一。无论是结构化方法还是面向对象方法，设计中都引入了对复用的支持。

1.3 结构化方法及其弊端

结构化设计方法求解问题的基本策略是从功能的角度审视问题域。它将应用程序看成实现某些特定任务的功能模块，其中子过程是实现某项具体操作的底层功能模块。在每个功能模块中，用数据结构描述待处理数据的组织形式，用算法描述具体的操作过程。面对日趋复杂的应用系统，这种开发思路在下面几个方面逐渐暴露了一些弱点。

1.3.1 审视问题域的视角

在现实世界中，存在的客体是问题域中的主角，所谓客体是指客观存在的对象实体和主观抽象的概念，它是人类观察问题和解决问题的主要目标。例如，对于一个学校学生管理系统来说，无论是简单还是复杂，始终是围绕学生和老师这两个客体实施。在自然界，每个客体都具有一些属性和行为，例如学生有学号、姓名、性别等属性，以及上课、考试、做实验等行为。因此，每个个体都可以用属性和行为来描述。

通常人类观察问题的视角是这些客体，客体的属性反映客体在某一时刻的状态，客体的行为反映客体能从事的操作。这些操作附在客体之上并能用来设置、改变和获取客体的状态。任何问题域都有一系列的客体，因此解决问题的基本方式是让这些客体之间相互驱动、相互作用，最终使每个客体按照设计者的意愿改变其属性状态。

结构化设计方法所采用的设计思路不是将客体作为一个整体，而是将依附于客体之上的行为抽取出来，以功能为目标来设计构造应用系统。这种做法导致在进行程序设计的时候，不得不将客体所构成的现实世界映射到由功能模块组成的解空间中，这种变换过程，不仅增加了程序设计的复杂程度，而且背离了人们观察问题和解决问题的基本思路。另外，再仔细思考会发现，在任何一个问题域中，客体是稳定的，而行为是不稳定的。例如，不管是国家图书馆，还是学校图书馆，还是国际图书馆，都会含有图书这个客体，但管理图书的方法可能是截然不同的。结构化设计方法将审视问题的视角定位于不稳定的操作之上，并将描述客体的属性和行为分开，使得应用程序的日后维护和扩展相当困难，甚至一个微小的变动，都会波及整个系统。面对问题规模的日趋扩大、环境的日趋复杂、需求变化的日趋加快，将利用计算机解决问题的基本方法统一到人类解决问题的习惯方法之上，彻底改变软件设计方法与人类解决问题的常规方式相扭曲的现象迫在眉睫，这是提出面向对象的首要原因。

1.3.2 抽象级别

抽象是人类解决问题的基本法宝。良好的抽象策略可以控制问题的复杂程度，增强系统的通用性和可扩展性。抽象主要包括过程抽象和数据抽象。结构化设计方法应用的是过程抽象。所谓过程抽象是将问题域中具有明确功能定义的操作抽取出来，并将其作为一个实体看待。这种抽象级别对于软件系统结构的设计显得有些武断，并且稳定性差，导致很难准确无误地设计出系统的每一个操作环节。一旦某个客体属性的表示方式发生了变化，就有可能牵扯到已有系统的很多部分。而数据抽象是较过程抽象更高级别的抽象方式，将描

述客体的属性和行为绑定在一起，实现统一的抽象，从而达到对现实世界客体的真正模拟。

1.3.3　封装体

封装是指将现实世界中存在的某个客体的属性与行为绑定在一起，并放置在一个逻辑单元内。该逻辑单元负责将所描述的属性隐藏起来，外界对客体内部属性的所有访问只能通过提供的用户接口实现。这样做既可以实现对客体属性的保护作用，又可以提高软件系统的可维护性。只要用户接口不改变，任何封装体内部的改变都不会对软件系统的其他部分造成影响。结构化设计方法没有做到客体的整体封装，只是封装了各个功能模块，而每个功能模块可以随意地对没有保护能力的客体属性实施操作，并且由于描述属性的数据与行为被分割开来，所以一旦某个客体属性的表达方式发生了变化，或某个行为效果发生了改变，就有可能对整个系统产生影响。

1.3.4　可重用性

可重用性标识着软件产品的可复用能力，是衡量一个软件产品成功与否的重要标志。当今的软件开发行业，人们越来越追求开发更多的、更有通用性的可重用构件，从而使软件开发过程彻底改善，即从过去的语句级编写发展到现在的构件组装，从而提高软件开发效率，推动应用领域迅速扩展。然而，结构化程序设计方法的基本单位是模块，每个模块只是实现特定功能的过程描述，因此，它的可重用单位只能是模块。例如，在 C 语言编写程序时使用大量的标准函数。但对于今天的软件开发来说，这样的重用粒度显得微不足道，而且当参与操作的某些数据类型发生变化时，就不能够再使用那些函数了。因此，渴望更大力度的可重用构件是如今应用领域对软件开发提出的新需求。

上述弱点驱使人们寻求一种新的程序设计方法，以适应现代社会对软件开发的更高要求，面向对象由此产生。

1.4　面向对象方法对软件复用的支持

面向对象的思想已经涉及软件开发的各个方面。如：面向对象的分析（Object Oriented Analysis，OOA）、面向对象的设计（Object Oriented Design，OOD），以及我们经常说的面向对象编程（Object Oriented Programming，OOP）。

面向对象的分析根据抽象关键的问题域来分解系统。面向对象的设计是一种提供符号设计系统的可视化过程，它用非常接近实际领域术语的方法把系统构造成“现实世界”的对象。面向对象程序设计可以看作一种在程序中包含各种独立而又互相调用的对象的思想，这与传统的思想刚好相反：传统的程序设计主张将程序看作一系列函数的集合，或者直接就是一系列对电脑下达的指令。面向对象程序设计中的每一个对象都应该能够接收数据、处理数据并将数据传达给其他对象，因此它们都可以被看作一个小型的“机器”，即对象。

1.4.1 面向对象的基本概念

1. 对象

对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件。

2. 对象的状态和行为

对象具有状态，一个对象用数据值来描述它的状态。

对象还具有操作，用于改变对象的状态，对象及其操作就是对象的行为。

对象实现了数据和操作的结合，使数据和操作封装于对象的统一体中。

3. 类

具有相同特性（数据元素）和行为（功能）的对象的抽象就是类。因此，对象的抽象是类，类的具体化就是对象，也可以说类的实例是对象，类实际上就是一种数据类型。

类具有属性，它是对象的状态的抽象，用数据结构来描述类的属性。

类具有操作，它是对象的行为的抽象，用操作名和实现该操作的方法来描述。

4. 类的结构

在客观世界中有若干类，这些类之间有一定的结构关系。通常有两种主要的结构关系，即“一般-具体”结构关系，“整体-部分”结构关系。

“一般-具体”结构称为分类结构，是“is a”关系。“is a”表示的是属于的关系。例如：野马属于一种马，兔子属于一种动物（继承关系）。基于类继承或接口实现。

“整体-部分”结构称为组装结构，是一种“has a”关系。has-a 表示聚合、组合的包含关系。比如兔子包含有腿、头等组件。组合关系是整体与部分的关系，但部分不能离开整体而单独存在，如公司和部门是组合关系，没有公司就不存在部门；聚合关系也是整体与部分的关系，部分可以离开整体而单独存在，如车和轮胎是聚合的关系，轮胎离开车仍然可以存在。

自然语义中，包含、属于往往含有继承、组合、聚合三种关系，严谨的计算机模型，需要对这三个关系做出区分。

5. 消息和方法

对象之间进行通信的结构叫作消息。在对象的操作中，当一个消息发送给某个对象时，消息包含接收对象去执行某种操作的信息。发送一条消息至少要包括说明接受消息的对象名、发送给该对象的消息名（即对象名、方法名）。一般还要对参数加以说明，参数可以是认识该消息的对象所知道的变量名，或者是所有对象都知道的全局变量名。

类中操作的实现过程叫作方法，一个方法有方法名、返回值、参数和方法体。

1.4.2 面向对象的特征

1. 对象唯一性

每个对象都有自身唯一的标识，通过这种标识，可找到相应的对象。在对象的整个生命期中，它的标识都不改变，不同的对象不能有相同的标识。

2. 抽象性

抽象性是指将具有一致的数据结构（属性）和行为（操作）的对象抽象成类。一个类就是

这样一种抽象，它反映了与应用有关的重要性质，而忽略其他一些无关内容。任何类的划分都是主观的，但必须与具体的应用有关。

3. 继承性

继承性是子类自动共享父类数据结构和方法的机制，这是类之间的一种关系。在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并加入若干新的内容。

继承性是面向对象程序设计语言不同于其他语言的最重要的特点，是其他语言所没有的。

在类层次中，子类只继承一个父类的数据结构和方法，则称为单重继承。

在类层次中，子类继承了多个父类的数据结构和方法，则称为多重继承。

Java、VB、NET、Objective-C 均仅支持单继承，注意在 C＋＋多重继承时，需小心二义性。

在软件开发中，类的继承性使所建立的软件具有开放性、可扩充性，这是信息组织与分类的行之有效的方法，它简化了对象、类的创建工作量，增加了代码的可重用性。

采用继承性，提供了类的规范的等级结构。通过类的继承关系，使公共的特性能够共享，提高了软件的重用性。

4. 多态性(多形性)

多态性是指相同的操作或函数、过程可作用于多种类型的对象上并获得不同的结果。不同的对象，收到同一消息可以产生不同的结果，这种现象称为多态性。

多态性允许每个对象以适合自身的方式去响应共同的消息。

多态性增强了软件的灵活性和重用性。

5. 封装性(信息隐藏)

封装性是保证软件部件具有优良的模块性的基础。

面向对象的类是封装良好的模块，类定义将其说明(用户可见的外部接口)与实现(用户不可见的内部实现)显式地分开，其内部实现按其具体定义的作用域提供保护。

对象是封装的最基本单位。封装防止了程序相互依赖而带来的变动影响。面向对象的封装比传统语言的封装更清晰、更有力。

6. 共享性(重用)

面向对象技术在不同级别上促进了共享。

同一类中的共享。同一类中的对象有着相同数据结构。这些对象之间是结构、行为特征的共享关系。

在同一应用中共享。在同一应用的类层次结构中，存在继承关系的各相似子类中，存在数据结构和行为的继承，使各相似子类共享共同的结构和行为。使用继承来实现代码的共享，这也是面向对象的主要优点之一。

在不同应用中共享。面向对象不仅允许在同一应用中共享信息，而且为未来目标的可重用设计准备了条件。通过类库这种机制和结构来实现不同应用中的信息共享。

1.4.3　面向对象与 RUP

软件开发过程是软件工程的要素之一，有效的软件开发过程可以提高软件开发团队的

生产效率，并能够提高软件质量、降低成本、减少开发风险。常见的软件过程模型有瀑布模型、原型模型、增量模型、演化模型、喷泉模型、V 模型、X 模型、敏捷过程等。

RUP(Rational Unified Process)即 Rational 统一软件过程，RUP 是一种用例驱动的、以架构为中心的、采用迭代增量方式开发的软件工程过程。它汲取了面向对象软件工程领域多年来的优秀研究成果，应用统一建模语言(UML)进行可视化建模，为面向对象的软件系统的开发提供了方法论的指导。RUP 模型如图 1.2 所示。

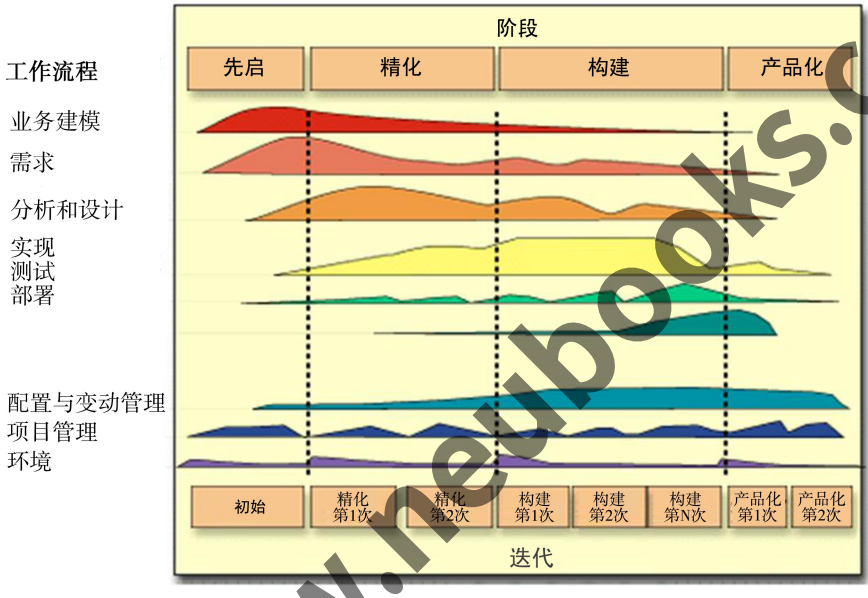

图 1.2 RUP 模型

RUP 中的软件生命周期在时间上被分解为四个顺序的阶段，分别是：先启阶段(Inception)、精化阶段(Elaboration)、构建阶段(Construction)和产品化阶段(Transition)。每个阶段结束于一个主要的里程碑(Major Milestones)；每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足。如果评估结果令人满意的话，可以允许项目进入下一个阶段。

1. 先启阶段

先启阶段的目标是为系统建立商业案例并确定项目的边界。为了达到该目的必须识别所有与系统交互的外部实体，在较高层次上定义交互的特性。本阶段具有非常重要的意义，在这个阶段中所关注的是整个项目进行中的业务和需求方面的主要风险。对于建立在原有系统基础上的开发项目来讲，先启阶段可能很短。先启阶段结束时是第一个重要的里程碑：生命周期目标(Lifecycle Objective)里程碑。生命周期目标里程碑评价项目基本的生存能力。

2. 精化阶段

精化阶段的目标是分析问题领域，建立健全的体系结构基础，编制项目计划，淘汰项目中最高风险的元素。为了达到该目的，必须在理解整个系统的基础上，对体系结构做出决策，包括其范围、主要功能和诸如性能等非功能需求。同时为项目建立支持环境，包括创建

开发案例、创建模板、准则并准备工具。精化阶段结束时第二个重要的里程碑：生命周期结构(Lifecycle Architecture)里程碑。生命周期结构里程碑为系统的结构建立了管理基准并使项目小组能够在构建阶段中进行衡量。此刻，要检验详细的系统目标和范围、结构的选择以及主要风险的解决方案。

3. 构建阶段

在构建阶段，所有剩余的构件和应用程序功能被开发并集成为产品，所有的功能被详细测试。从某种意义上说，构建阶段是一个制造过程，其重点放在管理资源及控制运作以优化成本、进度和质量。构建阶段结束时是第三个重要的里程碑：初始功能(Initial Operational)里程碑。初始功能里程碑决定了产品是否可以在测试环境中进行部署。此刻，要确定软件、环境、用户是否可以开始系统的运作。此时的产品版本也常被称为“beta”版。

4. 产品化阶段

产品化阶段的重点是确保软件对最终用户是可用的。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量的调整。在生命周期的这一点上，用户反馈应主要集中在产品调整、设置、安装和可用性问题，所有主要的结构问题应该已经在项目生命周期的早期阶段解决了。在交付阶段的终点是第四个里程碑：产品发布(Product Release)里程碑。此时，要确定目标是否实现，是否应该开始另一个开发周期。在一些情况下这个里程碑可能与下一个周期的初始阶段的结束重合。

RUP 中有 9 个核心工作流(Core Workflows)，分为 6 个核心过程工作流(Core Process Workflows)和 3 个核心支持工作流(Core Supporting Workflows)。尽管这 6 个核心过程工作流可能使人想起传统瀑布模型中的几个阶段，但应注意迭代过程中的阶段是完全不同的，这些工作流在整个生命周期中一次又一次被访问。9 个核心工作流在项目中轮流被使用，在每一次迭代中以不同的重点和强度重复。

1. 业务建模(Business Modeling)

业务建模工作流描述了如何为目标组织开发一个新的构想，并定义这个构想在商业用例模型和商业对象模型中组织的过程、角色和责任。

2. 需求(Requirements)

需求工作流的目标是描述系统应该做什么，并使开发人员和用户就这一描述达成共识。为了达到该目标，要对需要的功能和约束进行提取、组织、文档化；最重要的是理解系统所解决问题的定义和范围。

3. 分析和设计(Analysis & Design)

分析和设计工作流将需求转化成未来系统的设计，为系统开发一个健壮的结构并调整设计使其与实现环境相匹配，优化其性能。分析设计的结果是一个设计模型和一个可选的分析模型。设计模型是源代码的抽象，由设计类和一些描述组成。设计类被组织成具有良好接口的设计包(Package)和设计子系统(Subsystem)，而描述则体现了类的对象如何协同工作实现用例的功能。设计活动以体系结构设计为中心，体系结构由若干结构视图来表达，结构视图是整个设计的抽象和简化，该视图中省略了一些细节，使重要的特点体现得更加清晰。体系结构不仅仅是良好设计模型的承载媒介，而且在系统的开发中能提高被创建模型的质量。

4. 实现(Implementation)

实现工作流的目的包括以层次化的子系统形式定义代码的组织结构;以组件的形式(源文件、二进制文件、可执行文件)实现类和对象;将开发出的组件作为单元进行测试以及集成由单个开发者(或小组)所产生的结果,使其成为可执行的系统。

5. 测试(Test)

测试工作流要验证对象间的交互作用,验证软件中所有组件的正确集成,检验所有的需求已被正确的实现,识别并确认缺陷在软件部署之前被提出并处理。RUP 提出了迭代的方法,意味着在整个项目中进行测试,从而尽可能早地发现缺陷,从根本上降低了修改缺陷的成本。测试类似于三维模型,分别从可靠性、功能性和系统性能来进行。

6. 部署(Deployment)

部署工作流的目的是成功的生成版本并将软件分发给最终用户。部署工作流描述了那些与确保软件产品对最终用户具有可用性相关的活动,包括:软件打包、生成软件本身以外的产品、安装软件、为用户提供帮助。在有些情况下,还可能包括计划和进行 beta 测试版、移植现有的软件和数据以及正式验收。

7. 配置和变更管理(Configuration & Change Management)

配置和变更管理工作流描绘了如何在多个成员组成的项目中控制大量的产物。配置和变更管理工作流提供了准则来管理演化系统中的多个变体,跟踪软件创建过程中的版本。工作流描述了如何管理并行开发、分布式开发、如何自动化创建工程。同时也阐述了对产品修改原因、时间、人员保持审计记录。

8. 项目管理(Project Management)

软件项目管理平衡各种可能产生冲突的目标、管理风险,克服各种约束并成功交付使用户满意的产品。其目标包括:为项目的管理提供框架,为计划、人员配备、执行和监控项目提供实用的准则,为管理风险提供框架等。

9. 环境(Environment)

环境工作流的目的是向软件开发组织提供软件开发环境,包括过程和工具。环境工作流集中于配置项目过程中所需要的活动,同样也支持开发项目规范的活动,提供了逐步的指导手册并介绍了如何在组织中实现过程。

RUP 是一个通用的过程模板,包含了很多开发指南、软件制品、开发过程所涉及的角色说明,由于它非常庞大,所以对具体的开发机构和项目,用 RUP 时还要做裁剪,也就是要对 RUP 进行配置。RUP 就像一个元过程,通过对 RUP 进行裁剪可以得到很多不同的开发过程,这些软件开发过程可以看作 RUP 的具体实例。RUP 裁剪可以分为以下几步:

(1)确定本项目需要哪些工作流。RUP 的 9 个核心工作流并不总是需要的,可以取舍。

(2)确定每个工作流需要哪些制品。

(3)确定 4 个阶段之间如何演进。确定阶段间演进要以风险控制为原则,决定每个阶段要那些工作流,每个工作流执行到什么程度,制品有哪些,每个制品完成到什么程度。

(4)确定每个阶段内的迭代计划。规划 RUP 的 4 个阶段中每次迭代开发的内容。

(5)规划工作流内部结构。工作流涉及角色、活动及制品,它的复杂程度与项目规模及角色多少有关。规划工作流的内部结构,通常用活动图的形式给出。

RUP 中的每个阶段可以进一步分解为迭代。一个迭代是一个完整的开发循环，产生一个可执行的产品版本，是最终产品的一个子集，它增量式地发展，从一个迭代过程到另一个迭代过程直到成为最终的系统。

1.5　UML

1.5.1　UML 简介

UML(Unified Modeling Language，统一建模语言)是面向对象开发中一种通用的图形化建模语言，它定义良好、易于表达、功能强大且普遍适用。UML 的出现既统一了 Booch、OMT、OOSE，以及其他方法，又统一了面向对象方法中使用的符号，并且在提出后不久就被 OMG 接纳为其标准之一。改变了数十种面向对象的建模语言相互独立且各有千秋的局面，使得面向对象的分析技术有了空前发展。它本身成为现代软件工程环境中对象分析和设计的重要工具，被视为面向对象技术的重要成果之一。

1.5.2　UML2.0 的组成

UML 提供了多种图形来可视化的描述模型，同一个模型元素可能会出现在多个图中，一张图也对应多个图形元素，最基本的图形元素叫事物，链接事物的元素叫关系，各种关系构成的图就是 UML 的模型，下面对事物、关系和图做一个简单的介绍。

事物包括：结构事物，行为事物，组织事物和辅助事物。

(1)结构事物包括 7 种，分别是类(Class)、接口(Interface)、协作(Collaboration)、用例(Use Case)、主动类(Active Class)、构件(Components)、节点(Nodes)，其中开发前期用类、接口和用例较多，构件和节点在后期使用，协作用得较少，主动类用在一些特殊的建模中。

(2)行为事物用来代表时间和空间上的动作。主要分为两种：交互和状态机。交互的消息通过画带箭头的直线表示。状态机表示对象的一个或者多个取值的迁移。用圆角矩形表示。第二章用得比较多。

(3)组织事物也称分组事物，在 UML 中，组织事物只有一种，那就是包。包是元素分组的机制。包的符号就像我们计算机中的文件夹。

(4)辅助事物就是注释事物。这一类中只有注释(Notes)。注释是 UML 模型中的解释部分。符号就是一个折起一角的矩形(折了角的一页纸)。

事物以图标的形式显示在建模工具的工具条里，几乎所有的工具，只要鼠标在工具条的对应图标上停放一会，就会显示出名字来。

在 UML 图中，常见以下几种关系：泛化(Generalization)、实现(Realization)、关联(Association)、聚合(Aggregation)、组合(Composition)和依赖(Dependency)

泛化关系就是前面讲的“is a”关系，也叫继承关系。用带三角箭头的实线表示，箭头指

向父类。图 1.5 中,动物和鸟,鸟和大雁、鸭、企鹅之间的关系都是泛化关系箭头指向父类。

聚合和组合关系就是前面讲的“has a”关系,它们的区别是,聚合关系中的“部分”可以离开整体而单独存在,组合关系中的“部分”不可以离开整体而单独存在。聚合关系用带空心菱形的实心线表示,菱形指向整体。组合关系用带实心菱形的实心线表示,菱形指向整体。图 1.5 中,雁群和大雁的关系为聚合关系,鸟和翅膀的关系为组合关系。

下面详细讲一下其他几种关系。

实现关系是一种接口与类的关系,表示类是接口所有特征和行为的实现,用带三角箭头的虚线实现,箭头指向接口。对接口而言,里面也包含了泛化的成分,通常用 Implement 关键字。图 1.5 中,飞翔和大雁,讲话和唐老鸭之间都是实现关系。

关联关系是一种拥有的关系,它指一个类知道另一个类的属性和方法;如:老师与学生,丈夫与妻子关联可以是双向的,也可以是单向的。双向的关联可以有两个箭头或者没有箭头,单向的关联有一个箭头。如果是单向关系,用带普通箭头的实心线表示,箭头指向被拥有者。图 1.5 中,企鹅和气候之间就是一种单向的关联关系。

依赖关系是一种使用的关系,即一个类的实现需要另一个类的协助,所以要尽量不使用双向的互相依赖。依赖关系用带箭头的虚线表示,箭头指向被使用者。在图 1.5 中,动物依赖水和氧气。在使用中,依赖着往往是使用者的局部变量、方法的参数或者对静态方法的调用等,如图 1.3 所示。

图 1.3 依赖关系

泛化、实现、组合、聚合可以认为是一种强关联关系,从以上分析中也可以看出,这六种关系的强弱顺序为:泛化=实现>组合>聚合>关联>依赖。前五种很容易区分,而关联和依赖就容易混淆,它们的区别是:依赖没有关联程度强烈,对于两个类之间的依赖,只是保持在方法上,例如一个类的方法中引用了某个类的变量;关联中彼此把对方作为自己的一个属性,耦合度更高。保持的时间更长些,毕竟是作为一个属性存在,若 A 依赖于 B,实例化 A 的时候,就会有 B。而依赖只是单纯的用到对方方法时才存在。

图 1.4 是 UML2.5 包含的图,注意图之间的继承关系,这个图是一个工具箱,看起来复杂,一次建模中,并不需要画出所有的图,事实上,大多数开发,只用到其中的几张图,学会了,只需要根据待开发特点选用需要的即可。

UML(语言)与 UML 建模(UML Modeling)是两个不同的概念。正确地学 UML 等于 UML 建模,不应该只学语言、模型的表示方法,还应该学建模的方法和技术;学会了 UML 建模,自然就会 UML 了。14 种图中,最常用的简单介绍如图 1.5 所示。

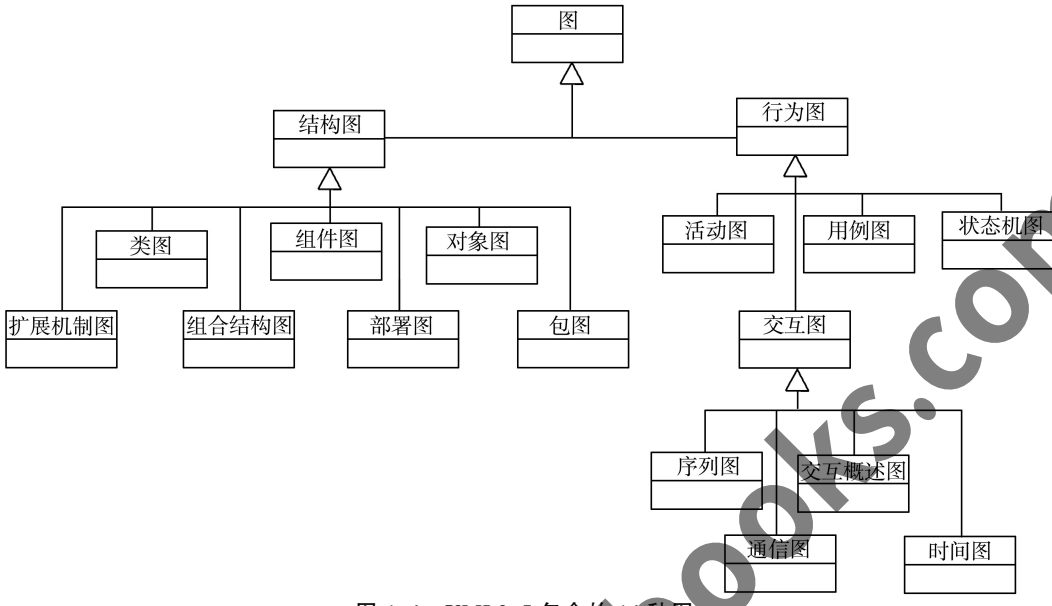

图 1.4　UML2.5 包含的 14 种图

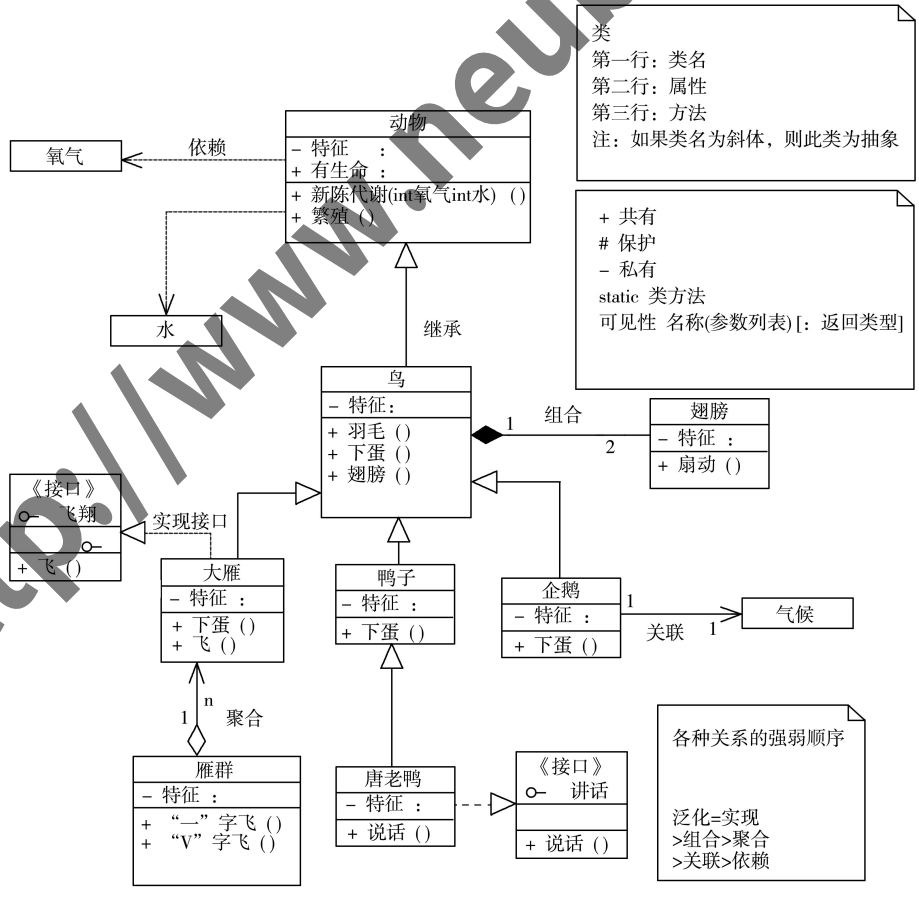

图 1.5　类图的 UML 语法

用例图描述角色(执行者:业务执行者和系统执行者)与用例之间的连接关系。以系统用例为例:说明的是谁要使用系统,以及他(她、它)们使用该系统可以做些什么。一个用例图包含了多个模型元素,如系统、参与者和用例,并且显示了这些元素之间的各种关系,如泛化、关联和依赖。第二章将详细介绍用例图的画法。

类图是描述系统中的类(接口是一种特殊的类)以及各个类之间的关系的静态视图,类图让我们在正确编写代码以前对系统有一个全面的认识。图 1.5 是一个典型的类图,这个图中体现了类图中的各种元素和关系。正如前面所描述:虚线箭头指向依赖;实线箭头指向关联;虚线三角指向接口;实线三角指向父类;空心菱形能分离而独立存在,是聚合;实心菱形精密关联不可分,是组合。

对象图表示某个类图的一个实例。类图和对象图的基本概念是相似的,对象图也代表了一个系统的静态视图,但这种静态视图是系统在某一时刻的一个快照,如属性取了某个值。

包图用于描述系统的分层结构,由包和类组成,表示类与包、包与包之间的关系。

活动图记录了单个操作或方法(活动)的逻辑,或单个业务流程的逻辑。活动图是结构化开发中流程图和数据流程图(DFD)的扩展。活动图强调的是从活动到活动的控制流。

状态图描述对象可能的状态,以及事件发生时状态的转移。状态图告知一个对象可以拥有的状态,以及事件(如消息的接收、时间的流逝、错误发生、条件改变等)如何影响这些状态。状态图是对类图的补充。

序列图(也叫顺序图)用来显示组织内部各系统是如何协作来完成一个用例,也可以用来展示对象之间是如何进行交互。第二章对序列图有详细的介绍。

通信图(协作图)和序列图等价,描述的是对象和对象之间的关系,即一个类操作的实现。简而言之就是,对象和对象之间的调用关系,体现的是一种组织关系。如果强调时间和顺序,则使用序列图;如果强调上下级关系,则选择通信图。

组件图描述代码构件的物理结构以及各种构件之间的依赖关系。由构件标记符和构件之间的关系构成。在组件图中,构件是软件单个组成部分,它可以是一个文件、产品、可执行文件和脚本等。

部署图用来建模系统的物理部署。部署图用于表示一组物理结点的集合及结点间的相互关系,比如,一部分组件部署于服务器、另一些部署于客户端、客户端具体部署于哪个街道等。

本教材的重点在于面向对象的分析和设计,因此,在第二章,我们围绕业务建模到系统需求来展开,用到的图有用例图、序列图、状态机图,从建模的严肃性上来说,我们不建议用活动图。

1.5.3 UML 建模工具

当前市场上基于 UML 可视化建模的工具很多,例如有 IBM 的 Rational Rose,Microsoft 的 Visio,Sybase 的 Power Designer,还有 EA、PlayCase、CA BPWin、CA ERWin 等。各工具有不同的定位、能力和市场策略,每一种工具都不同程度地实现了标准的不同子集。市面上的工具基本上都能提供规范所定义的主要功能,但不同产品甚至同一产品的不同版本,在具体的功能实现上总存在一些差异,表现出各自的特性而具有不同的适用面。

ROSE 是直接为 UML 发展而诞生的设计工具，它的出现就是为了对 UML 建模的支持，ROSE 一开始没有对数据库端建模的支持，但是在现在的版本中已经加入数据库建模的功能。ROSE 对开发过程中的各种语义、模块、对象以及流程、状态等描述比较好，能够从各个视角来分析和设计，使软件的开发蓝图更清晰，内部结构更加明朗（仅仅对掌握 UML 的开发人员来说，并非对客户了解系统的功能和流程等），对系统的代码框架生成有很好的支持。但对数据库的开发管理和数据库端的迭代不是很好。

PowerDesigner 刚开始是一种数据库建模工具。直到 7.0 版才开始了对面向对象的开发的支持，后来又引入了对 UML 的支持。由于 PowerDesigner 最初侧重于数据库建模，所以它对数据库建模的支持很好，支持了能够看到的 90%左右的数据库，对 UML 的建模使用到的各种图的支持比较滞后，但是在最近得到加强。

从以上描述可以看到，两者所走的路线不同，Rose 出道时，走的是 UML 面向对象建模，而后再向数据库建模发展，而 PowerDesigner 则反其道而行之，它先是一个纯粹的数据库建模工具，后来才向面向对象建模，业务逻辑建模及需求分析建模发展。现在的 Rose 和 PowerDesigner 既可以进行数据库建模，也可以进行面向对象建模，只是存在支持上的偏重而已。

Visio 是一种画图工具，能够用来描述各种图形（从电路图到房屋结构图，UML 仅仅支持其中很少的一部分）。它跟微软的 Office 产品能够很好兼容。能够把图形直接复制或者内嵌到 Word 文档中。更多的是支持微软的产品如 VB，VC＋＋，MS SQL Server 等的代码生成。它对图形语义的描述比较方便，对软件的迭代开发支持较弱。

另外，Enterprise Architect 也是用于软件系统的设计与开发、企业业务过程建模以及更广泛建模的可视化平台。Enterprise Architect 基于 UML 2.0 规范，是一款不断进步和完善的工具，它覆盖了开发周期的所有方面，提供了从初始设计阶段到系统部署、维护、测试以及修改控制的全程可跟踪性。

对初学者来说，学会 UML 建模是重要的，不同工具在各种细节上会有所区别，我们甚至不需要软件工具，而用一只笔直接在纸上建模。画图很简单，只需要知道怎么画就可以，但建模直接面对现实问题，知道画什么非常难，需要用非常严谨的态度去面对，正如不会因为用树枝在地上推导公式而随意使用数学符号，也不会因为纸张不好就可以乱画乐谱。对数学来说，符号后面展示的是共识，对乐谱来说，后面是基本乐理。专业建模，需要严格的训练才能做到。

习　题

一、填空题

1. 软件是对人类解决问题逻辑的_____。

2. _____是指一台计算机全部的指令集合，是第一代计算机语言。

3. 代码的复用包括_____和_____ 的复用。

4. 设计复用有三种途径，第一种途径是从现有系统的设计结果中_____一些可复用

的设计构件，并把这些构件应用于新系统的设计；第二种途径是把一个现有系统的全部设计文档在新的软硬件平台上_____，也就是把一个设计运用于多个具体的实现；第三种途径是独立于任何具体的应用，有计划地开发一些可复用的_____。

5. 过程抽象是将问题域中具有明确功能定义的_____抽取出来，并将其作为一个实体看待。

6. 结构化设计方法没有做到客体的整体封装，只是封装了各个_____，而每个功能模块可以随意地对没有保护能力的_____实施操作，并且由于描述属性的数据与行为被分割开来，所以一旦某个客体属性的表达方式发生了变化，或某个行为效果发生了改变，就有可能对整个系统产生影响。

7. 公司和部门是_____关系，没有公司就不存在部门。

8. 车和轮胎是_____关系，轮胎离开车仍然可以存在。

9. RUP 中的软件生命周期在时间上被分解为四个顺序的阶段，分别是：_____、_____、_____和产品化阶段(Transition)。

10. 在 UML 图中，常见以下几种关系：_____，_____，关联(Association)，聚合(Aggregation)，_____，依赖(Dependency)。

二、简答题

1. 面向对象的特征有哪些？

2. 简述 RUP 的需求工作流的主要任务。

3. 状态机和类的关系是什么，请简单说明。

第 2 章　需求建模

在计算机发展的初期，软件规模不大，软件开发所关注的是代码编写，需求分析很少受到重视。后来软件开发引入了生命周期的概念，需求分析成为其第一阶段。随着软件系统规模的扩大，需求分析与定义在整个软件开发与维护过程中越来越重要，直接关系到软件的成功与否。人们逐渐认识到需求分析活动不再仅限于软件开发的最初阶段，它贯穿于系统开发的整个生命周期。

2.1　需求-设计前的质量

2.1.1　错误成本的放大

《软件工程经济学》的作者 Barry W. Beohm 通过对 63 个软件开发项目的研究，得出表 2.1 的统计结果：为了修正一个错误，生命周期的不同阶段所要付出的成本。

表 2.1　　错误成本放大比例

发现错误的阶段	成本倍数
需求阶段	1
设计阶段	3～6
编码阶段	10
开发测试阶段	15～40
应用测试阶段	30～70
实际运行阶段	40～1000

这些数字还是很保守的，因为 Beohm 研究的是已经完成的项目，也就是说这些数据中还没有包括至少 1/3 的没有完成的项目，而这些夭折的项目很大程度上都应该“归功于”需求分析。

几十年来的经验表明，那些错误的、失败的或灾难性的项目让投资者付出了巨大的代价，而这些项目中的大部分都起源于含混的需求。很多理想主义者在潜意识中认为，这个世界上存在着完全确定的东西；当他们筹建一个项目的时候，就把项目要求看成了确实存在的标准。当我们接下他们提出的项目，项目要求就成了我们获得的需求陈述；不幸的是，这一需求陈述在绝大多数情况下是含混的。

陈述需求中的含混性包括缺少的需求、含混的词语、无意中引入的假设等。这些含混性往往造成巨大的成本损失。所以需求的探索是为了消除含混性而探索。需求探索过程也应该是一个渐进的、逐步明确问题的过程。就像是一次“历险”。为了避免含混性,我们要随时提醒自己:

(1)含混性需要成本;含混性发现得越早越好,解决成本越低。

(2)任何需求分析的开始之处,就是发现问题的真正所在。“真正”的意思,就是说:很多时候,问题看起来是一回事情,其实是另一回事情。

(3)是什么问题?问题是谁的问题?来自哪儿?

(4)如何在特定的层面上,考虑解决问题?

(5)解决问题能达到什么目标或好处?对谁有利?对谁不利?

(6)客户真的需要解决问题吗?

市面上的关于解决方案或需求分析的书,有些是拿一个表格或流程去问客户。这是工作形式,但不是思考。

另外,什么样的需求才算需求?应该向谁,以何种方式提出问题,才能得到关于需求的真实反馈?对于需求的描述是不是足够清晰?如何在探索需求的整个过程中避免含混性带来的影响?在讨论需求的会议上如何最大限度地发挥团队合作的能力?有人说产品经理的核心能力就是问出正确的问题,那么上面这些问题大概就是在探索需求的时候产品经理必须提出的问题了。

2.1.2 从卖的视角看需求

第一章我们说过,设计通过重用前人的劳动,可以提高劳动效率进而达到节省成本的目标,也就是说,设计是从做的角度看问题,设计可以省钱。

软件开发的目的,自然是在设计中集成更多的业务逻辑和劳动成果,但是需求,却是满足用户的需要,满足用户需要的最佳考核点就是产品有没有人买,好不好卖,卖的量大不大。排除违法因素,站在好卖的角度,给愿意掏钱购买产品的人提供软件服务,帮他们解决问题,就可以更精准的找出我们的服务对象,进而挖掘出具体的需求。

为了搞清楚利润、需求和设计和关系,潘家宇先生在他的《软件方法》一书中,提出了一个公式,可以帮助我们理清楚这个问题:

$$利润=需求-设计$$

这个公式把软件开发工作拉入了整个经济活动,直接让软件变成了经济生态链中的一个环节。有了这个公式,需求和设计阶段的工作目标就很清晰,需求工作致力于“提升销售”,设计工作致力于“降低成本”,二者不能互相替代,而是相辅相成。低成本生产的产品,如果不好卖,无法取得丰厚的利润,就不能算成功。系统好卖,如果生产成本太高,同样赚不了钱,生产的动力就会打折扣。好的软件产品,在法律、道德的约束范围内,一定是取得需求和成本的平衡。

理清了需求是帮用户解决问题,获得需求的方式就非常灵活,下面的方法都是我们可选取的形式:

(1)面谈和问卷调查:面谈是获取软件需求的最有用的方法之一。面谈需准备面谈对象

和面谈的问题。

面谈对象是与系统相关的有代表性涉众，选取涉众可以从下面的问题着手：

● 谁为系统付费，购买系统？谁使用系统、谁会受到系统结果的影响，谁来监管该系统？谁来维护系统？是否有专门的维护团队？

● 确定访谈对象的背景：姓名、年龄、部门所处的职位、目前的工作范围，目前碰到哪些问题，这些问题会对工作、生活产生什么影响？

● 对象环境的背景：计算机应用水平；目前是否有相同的系统在使用？使用该系统碰到哪些问题？目前如何处理工作？对培训等有什么要求？

● 重复问题，取得面谈者对问题的认同。

● 分析问题：问题产生的原因是什么？在什么情况下会有该问题？目前的解决方案是什么，效果如何？客户期待的解决方案是什么？

● 解决方案的可行性分析。

● 非功能性需求：性能和稳定性方面的要求。

● 对当前的访谈结果的认同，确认后期有问题可继续联系。

● 总结出当前优先级最高的三个问题。

调查问卷无法取代面谈在需求获取阶段的作用，问卷调查的问题和答案具有一定的引导性，在某种程度上会影响结果。

(2)研讨会：把各个岗位的人员召集在一起，通过头脑风暴，讨论问题或解决方案。

(3)场景法：涉众描述的业务流程可能由于某些原因会遗漏掉重要的信息，需求分析人员可申请参与到他们具体的工作，观察、体验业务操作过程。需求分析员在观察业务操作过程时，可根据实际的情况提问并详细记录，记录业务操作员操作过程，操作过程中碰到的难题，可获取真实的材料和理解整个业务。

(4)文档分析：阅读现有产品文档有利于了解当前系统情况，从中也可以了解业务流程，对操作员反映的系统问题有着更深层次的理解。

(5)快速原型：用软件原型方法反复确认需求。

2.1.3　建模工作流与思考边界

以上讲解的很多需求访谈方法，可以粗略的了解应用的现状。实际上。软件实施后，有些人工的工作，会被软件取代的，因此，在访谈的基础上，对业务建模，才是最重要的一步。

潘加宇先生《软件方法》一书中，对 RUP 的四个工作流的思考边界重新进行了定义，四个工作流的描述如下：

(1)业务建模——描述组织内部各系统(人脑系统、电脑系统……)如何协作，使得组织可以为其他组织提供有价值的服务。新系统只不过是组织为了对外提供更好的服务，对自己的内部重新设计而购买的一个零件。组织引进一个软件系统，和招聘一名新员工没有本质区别。如果能学会通过业务建模去推导新系统的需求，而不是拍脑袋得出需求，假的“需求变更”会大大减少。

(2)需求——描述为了解决组织的问题，系统必须具有的表现——功能和性能。这项技能的意义在于强迫我们从“卖”的角度思考哪些是涉众(Stakeholder)在意的、不能改变的契

约,哪些不是,严防“做”污染“卖”。需求工作流的结果——需求规约是“卖”和“做”的衔接点。

(3)分析——提炼为了满足功能需求,系统需要封装的核心领域机制。可运行的系统需要封装各个领域的知识,其中只有一个领域(核心域)的知识是系统能在市场上生存的理由。对核心域做研究,可以帮助我们获得基于核心域的复用。

(4)设计——为了满足质量需求和设计约束,核心领域机制如何映射到选定平台上实现。

如果把制作新系统的愿景加上,业务建模和需求的工作将包括下面的步骤:愿景、业务用例、业务序列图和业务序列图的改进,系统用例图、系统用例规约,下面将逐一讲述。

2.2 愿景

愿景就是发自你内心最深沉、最真挚的愿望。有了它,人就会兴奋、激动,产生出创造性张力。任何伟大新事物的出现,都是从一个愿景开始的;任何人的进步,也是从愿景开始的。每当心中有了一个动人的愿景,就会创造一段动人的经历。

软件愿景,也是如此,它是需要开发软件的组织老大,对新开发系统的期望。在老大看来,引进这个系统的目的是什么?更通俗地说:待开发的软件系统最应该卖给谁?对他有什么好处?这是一个看似很简单的问题,但回答起来并不是想象的那么容易。

缺乏清晰和共享的愿景,开发人员可能会在错误的方向上努力,做得越多,浪费越多。在开发中,总能看到这样的现象,有些功能与愿景无关,但技术是新的,技术人员很感兴趣,于是花费大量的时间去做,不但拖延工期,也浪费了资源。愿景往往是组织老大的期望,普通的技术人员限于职责和眼界,并不太熟悉,因此也不会太重视。

另外,有些项目也意识到愿景的重要,但对愿景的具体组成并不清晰,写出的愿景往往是一堆大话和套话,因此,有必要对愿景做一详细的讨论。

2.2.1 定位目标组织和老大

目标组织,指待开发系统将改进流程的组织,它可以是一个机构,也可以是一个人群。比如:软件的引入要改进的是学校教务处管理的流程,组织就是教务处,软件引入后要改进的是财务处的工作流程,那么,组织就是财务处。

组织委托软件开发,委托者就是“客户”,在权衡系统的各种需求时,意见最重要的、最有“地位”的涉众就是老大。

为什么要说“老大”,不直接说“客户”呢?因为“客户”指的是一个组织或人群,不是具体的某个人。我们需要具体到老大——客户中针对此系统最有发言权的人。

看起来,找老大,是个很容易的事情,但细想一下,问题没那么简单。

作为一个软件公司,如果我们给一家大医院开发医院管理信息系统,对接的是医院的信息科,那么信息科的主任往往是我们能接触的最有权威的人,那他是不是老大呢?

医院信息系统的上线,“看病”这件事情不变,但看病的流程可能发生改变,而这个改变

涉及医院管理的方方面面，涉及医院各部门流程和人员职位的变更，因此，信息科主任显然是无法完成的，老大选院长似乎更合适。

找老大的要点 1：系统改善哪个组织的流程？老大就是该组织的负责人。

如果老大定义为院长，但有些情况下，院长受更高领导的约束，受法律的约束，难道税务局领导，卫计委领导，省长该是系统的老大？显然不是。

找老大的要点 2：系统好坏的度量指标藏在他的大脑里吗？

改革受到的约束条件很多，老大是系统最优先照顾其利益的那个人，如果他不满意，系统无法完成交付，因此，虽然院长的上级很多，但他们并不为系统负责，因此，老大选院长更合适。

目前的很多软件，并不是为一个具体的组织而开发，而是服务于某个人群，如 QQ、滴滴，那这部分的开发对象和老大怎么确定呢？是 QQ 公司和滴滴公司的老总吗？

找老大的要点 3：老大是目标组织的代表。

要点三说明：QQ 的老大可以选择喜欢使用 QQ 的人群，滴滴的老大可以选择最乐意选择滴滴作为交通工具的司机和乘客，是不是和要点 2 冲突呢？我们深入的思考就会发现，如果选这两个公司的老总为老大，服务的目标可能是老总满意，但是我们知道，这两个公司是靠他们的“用户”来产生利润的，如果“用户”不买账，公司无法盈利，老总自然也当不长。

同样，医院的例子中，选择患者为老大，或许对医院的长期发展来说更好。

在一个官本位的社会里，一切由院长说了算，院长就是老大；在一个以盈利为目的的医院里，或许一个大牌医生，可以吸引很多的患者前来就医，医生的地位很高，那么，医生可以做老大；在一个成熟的社会中，职业道德、医生的技术和诊疗水平均可以得到保障，那么，医院可以以自己的服务水平选定特定的病人群体作为服务对象，那么病人就是老大。

定位目标组织和老大，可以强迫开发人员做更深刻的思考，从而抓住重点，抛弃一些无用的需求。

“愿景”是一个广泛的概念，由组织领导者与组织成员共同形成，是对未来情景的意象描绘，不但可以引导与激励组织成员，而且在不确定和不稳定环境中，方向性的长程导向。愿景把组织活动聚焦在一个核心焦点的目标状态上，使组织及其成员在面对混沌状态或阻力时能有所坚持，借由愿景，可有效增加组织生产力，达成组织目标。

愿景受组织领导者及组织成员的价值观和组织宗旨等影响，是一种对组织及个人未来发展预期达成未来意象的想法，它会引导或影响组织及其成员的行动和行为。

借助以上对愿景的理解，我们可以总结：待开发软件系统的愿景，实际上就是该系统最重要涉众的利益。涉众指受系统影响的各种人。也就是直接或间接地影响到系统研发和使用的人，包括因系统的使用而受益和受损的人。常见的涉众有：

客户：为达到其公司的业务目的而投资项目或购买产品；

用户：直接或间接与产品打交道，是客户的一部分；

需求分析员：负责编写需求并传递给开发团队；

开发人员：设计、实现和维护产品；

测试人员：确定产品的行为是否与预期的一致；

文档编制人员：负责编写用户手册，培训资料和系统帮助；

项目经理:制定项目计划并带领开发人员获得成功;

法律人员:确保产品符合所有相关法规;

生产人员:制造包含该软件的产品;

开发方领导:关心产品能给公司带来的利益;

市场营销、技术支持及其他和产品和客户打交道的人员。

一般来说,涉众会做但不一定会定义系统,这就需要系统分析人员去挖掘和定义系统的功能等。

愿景不是问系统有什么功能,而是回答买了这个系统,对组织有什么好处。老大的确定也是利益博弈的结果,确定不同的老大,会对系统的走向产生影响。有些时候,我们很难确定出真正的老大,通过我们的整理,把老二确定为老大,也要远远好过我们不去思考这个问题。

在《软件方法》一书中,潘加宇先生提出了一个"投币法",来确定和检验谁是老大和系统的愿景。方法如下:在最短时间内(比如身上绑了定时炸弹),如果你推销不出你的软件产品,你的生命会面临威胁,这个时候,你准备给"谁"推销你的软件产品?推销时选择"说什么"?问题的答案中,前来购买产品的人就是老大,所说的话就是愿景。当然,父母为了爱你,买下这个产品不算,因为父母关心的是你,而不是系统。购买,是一个广义的概念,不只是金钱,还有时间和声誉。这个设定的目的是,我们可以不被无聊的非重点问题所困扰。

2.2.2 度量愿景——提炼改进目标

确定了组织和老大,老大脑子中对系统的期望指标就呼之欲出了。改进目标是"系统改善组织行为的指标",而不是"系统能做某事"。以教务考勤系统为例,见表 2.2。

表 2.2　　教务考勤系统愿景示例

系统:教务考勤系统
老大:成都东软学院教务部王部长
目标(度量指标)
* 提高考勤数据的上报效率和准确性
* 为教学管理者了解出勤率提供准确的依据
* 减少搜集考勤数据的工作量

首先,报考勤,是人工可以完成的工作,因此,功能不能作为改进的目标;另外,愿景的改进目标是思考在组织的视角下这条需求的意义,性能需求可针对此指标做出细化,新系统可以针对这个愿景,对组织的流程进行改进;第三,组织的改进,是新系统带来的,因此,系统做不到的事情,不应该列入。新系统的引入,对组织的局部目标有优化,因此,对整体目标也有一定的促进,但组织的大目标,并不是系统的目标。

愿景的设置中,组织、老大和度量指标细微的差别,可能导致软件产品风格的较大差异,尤其在组织是一个群体,而不是一个机构的情况下。我们用三个比较相近的、大家比较熟悉的系统做比较,见表 2.3、表 2.4 和表 2.5。

表 2.3　　知乎社区的愿景

系统：知乎社区
老大：知识用户群（知识猎奇者和生产者）里的领域知识大 V
目标（度量指标）
* 创造增量的信息（利用更新的服务和更高效的工具，帮助用户把更多好的内容生产出来）
* 提出好问题，并给出好答案
* 避免内容水化

知乎想要创造的是增量的信息，让用户在彼此问答互动中，产生更多原本存在于他们脑子里的，但不存在于网页中的知识、经验与见解。而这些增量信息，往往也是其他一些人非常想知道的。

虽然互联网上已经有了像谷歌这么优秀的搜索引擎公司，但是用户通过搜索得到的都是已经存在的信息（即存量信息）。搜索引擎只能收录已经产生的网页，但是大量存在于人们脑子里的知识、经验与见解并没有被发布到网上。

基于组织（知识猎奇者和生产者）、老大（知识大 V）与愿景（创造增量的信息），配合“知乎”网站，运营公司产生了以下的策略：

（1）通过问答的形式创造出具体的场景，然后请有能力的人来回答，不断创造出新的信息（即增量信息）。

（2）在知乎早期，为了严把质量关，知乎团队采取了“邀请＋认证”这样相对封闭的制度。

（3）激励优秀的答题用户给出更多好的答案，就必须保证他们过往的好回答不被后来的内容淹没。当然，也不能让很多质量不高的回答一直灌水，拉低社区的内容水准。为此，知乎的运营团队开发了投票和举报功能，其他的用户可以对优秀的答案进行点赞，把其顶的高高的；对质量不高的答案投反对票，把其踩下去；对不良答案进行举报，让算法自动将其折叠，甚至是删除。另外，对好的回答，整理编辑即可出书，甚至于寻找出版社出版发表，那都是非常容易做到的事。进一步激发了大 V 回答问题的动力。

（4）作为一个知识型的社区，知乎平台上必定会有大量的书单推荐内容，进一步加大了知乎和出版社的联系。

（5）通过精编的浓缩知识，去击穿互联网上的层层传播壁垒，为知乎的品牌建设做好广告服务。

表 2.4　　谷歌的愿景

系统：谷歌
老大：知识用户群里的关键字用户
目标（度量指标）
* 以最简洁最快速的方式展示存量信息

谷歌搜索里的信息都是存量信息，存在于已经被创造出来的网页中。搜索引擎通过广泛的收录网页，并且通过不断优化算法为用户呈现他们最想要看到的内容。以“信息＋按钮”的方式和用户交互。

表 2.5 百度社区交互愿景

系统:百度社区交互
老大:知识用户群里的非关键字用户
目标(度量指标)
* 为百度完善用户搜索出来的信息结构
* 提升用户的滞留率与参与度
* 植入广告

百度知道是依附于搜索引擎的附生工具,目的是为百度完善用户搜出来的信息结构。为了激励用户提出更多的问题、给出更多的回答,百度会通过积分等形式激励用户,但不会对问题和答案本身做质量把控。

结果就是用户为了得到积分,会不断往百度知道里灌水。大量重复的问题,大量毫无质量的回答,甚至直接就是从别处复制粘贴过来的答案,最终的结果就是百度知道并没有达成创造增量信息的使命,而只是在存量信息里去搅动搬转。至于那些根本文不对题的回答,则更是在损坏百度知道的用户体验。

有人认为,在热点汇聚的基础上,再强化一下时效性、原创性,将使百度新闻极有竞争力。另外一点,从互动性看,贴吧、空间积聚的人气和互动氛围,也让百度新闻的可参与性具备了先天优势,由此,可以形成百度新闻的特色,那就是可参与性。

事实上,数据、信息和知识是递进关系。按照中科院的陆汝钤院士提出的观点:“知识是结构化的信息”。因此从愿景开始,百度知道就不可能创造知识,它的运营逻辑就掐断了这条路。并不像有的人想象的一样,只要百度知道在知乎没起来之前及早转型,就能阻击知乎,自己成为中文互联网的最大知识社区。

正如网上有人评价的那样:百度知道是快餐,知乎是私家厨房。愿景的不同,产生了不同的效果,或许刚开始的时候,用户群是相同的,但随着时间的推进,这种差异就越来越明显。

愿景和业务建模就是为了确定最重要的需求是哪些。不同的人去挖掘,得出的答案是不同的,确定老大,并不是说其他涉众不重要,老大是系统最先照顾利益的人。

2.3 业务用例图

选定了组织,就可以进一步地对组织的行为进行研究,业务建模的目的是从组织的角度来定位系统应该提供的价值。

新系统的引入,是为了改进组织现有的效率或降低组织运营的成本,在新系统问世前,组织往往有一套老的处理问题的办法,如:在未使用财务系统之前,组织通过大量的财务人员,用算盘等工具,来计算工资、计算利润和成本等,有了财务系统,计算的效率更高,准确率更好。从这个角度来看,新系统其实和一个雇员差不多,只不过,这个雇员有个特点,他效率

高，但处理模糊需求的灵活性差，需要严格地定义和其他员工之间交互的细节。

因此说：要改进组织的价值，不一定要开发软件，有时换人（也就是说，不是更换软件系统，而是更换人肉系统）更管用。和一套软件系统的价格相比，也许雇用高级员工花费更多。在印度，高种姓家族雇佣很多的工人来洗衣，只是因为他们比洗衣机更便宜。

在建模中，开发人员有时会有意无意把自己的系统想得太重要。因此，建模要克服的第一个障碍是，不要夸大系统的价值，把新系统看成一个新员工即可。这个新员工可以高效处理一些老员工不喜欢做，或容易出错，或强度大的工作。这样的转化，可以让新系统成为整个组织业务的一个零件。可以方便地把一些人工的职责，迁移给新系统（工具）。因此，在建模过程中，我们也可以把新系统看成一个企业员工，新系统的企业员工之间的关系，可以看作同事关系，员工和员工的关系，在建模的角度看，实际上只有职责的差异。

可以说，整个人类历史就是一部工具（新系统）不断取代人工的历史。如果说，早期的替代更多地以体力取代（用弓箭代替投石头捕杀猎物）为主的话，近 40 年，这种替代的加速度不但变大，并且以取代难于培训的熟练技能进而引起生活方式的改变为主。如电报员：能记住复杂的电报码，能用耳朵分别出数字信号并与文字互换，能发出电报码的高级职位，很快被电话取代，电报员的技能现在变成了计算机的基本技能。

以近百年为例，淘汰的职业或岗位有：更夫（钟表）、马力（蒸汽机）、汽车火车飞机（车夫和船夫）、扳道岔的工人（自动控制）、交警（红绿灯）、电话接线员（程控交换机）、传呼小姐（双向通话的小灵通和手机）、信件邮递体系（电子邮件）、领航员（GPS）等，都是人的职能被机器替换的例子。系统建模，则是这一个过程更加的专业化。

有了以上的认知，我们建模的目标会更加的清晰。在需求工程中困扰工程师的需求变更问题，更多的是对建模的认知不清晰。

业务建模工作流，可以从两个视角研究组织，从外部看，组织是一个价值的集合，待建模组织给企业其他组织提供价值，用业务用例图来表示；从组织内部看，组织是实现价值的实体，是系统（人和机器）的集合，可以用序列图来展示。本节我们先讲业务用例图的画法。

以银行为例，画业务用例图的步骤如下：

首先，确定组织，并标记出组织的边界。银行的边界如图 2.1 所示。

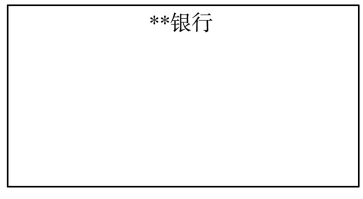

图 2.1　银行的边界

第二，设想有一个摄像机，在该组织的边界进行观察，所有进入该组织办事的其他企业、组织和人群，就是该组织的业务执行者。进入，可以是人进入，也可以是电话、网络等现代工具。银行的业务执行者如图 2.2 所示。

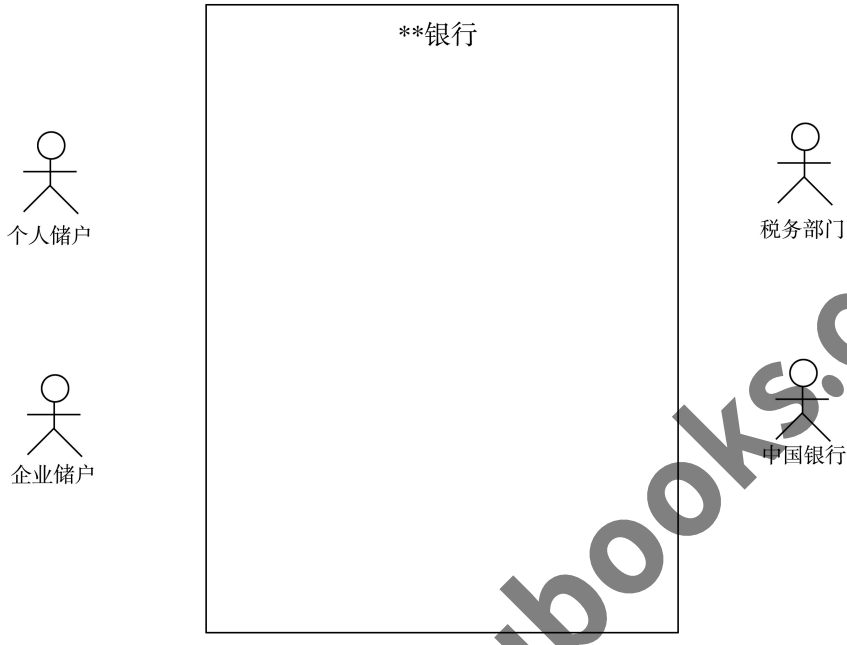

图 2.2 银行的业务执行者

第三,跟踪业务执行者,观察他们在该组织内寻求的价值所在。该价值就是待建模组织的业务用例,我们一般用动宾词组给业务用例命名。银行的业务用例如图 2.3 所示。

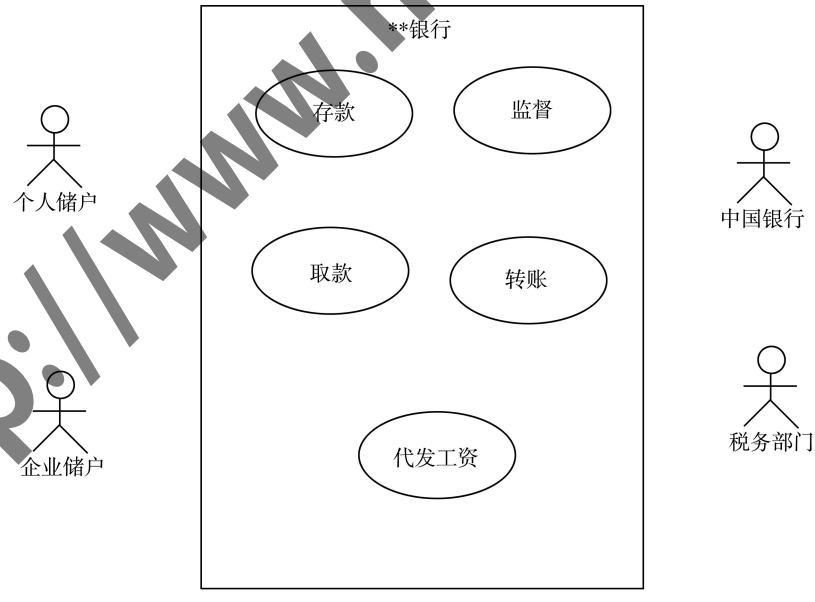

图 2.3 银行的业务用例

第四,画出执行者和用例之间的关系,执行者分主执行者和辅助执行者,主执行者是用例的启动方,箭头从主执行者指向用例;辅助执行者是辅助完成一个用例的一方,箭头从用例指向辅助执行者,无论是主执行者还是辅助执行者,都在组织之外。如图 2.4 所示,代发工资用例,由企业储户发起,但需要个人储户和税务部门的协助才能完成。

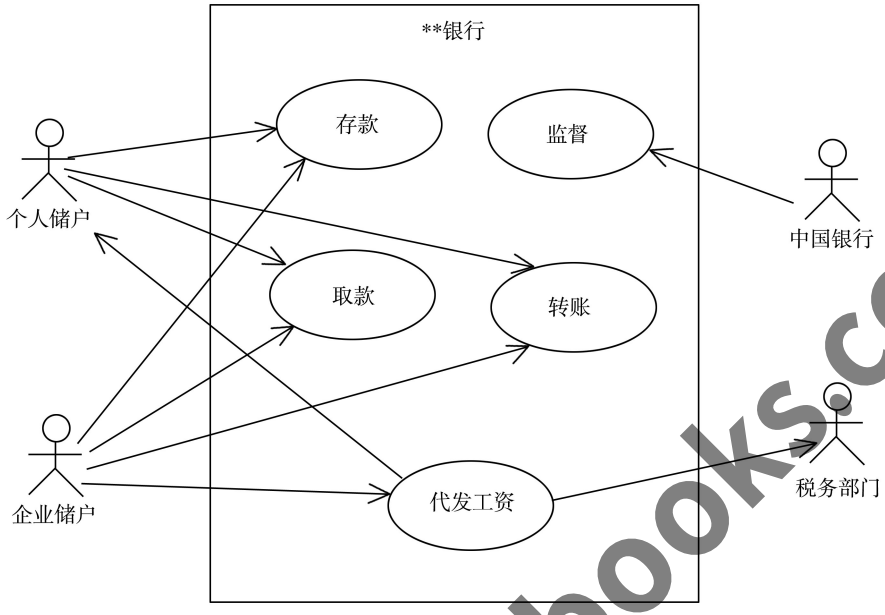

图 2.4　执行者和用例之间的关系

第五步，优化，如果企业用户和个人用户的存取款流程完全一样，可以用泛化关系来优化，如图 2.5 所示；如果不一样，可以当作不同的业务用例，如图 2.6 所示。

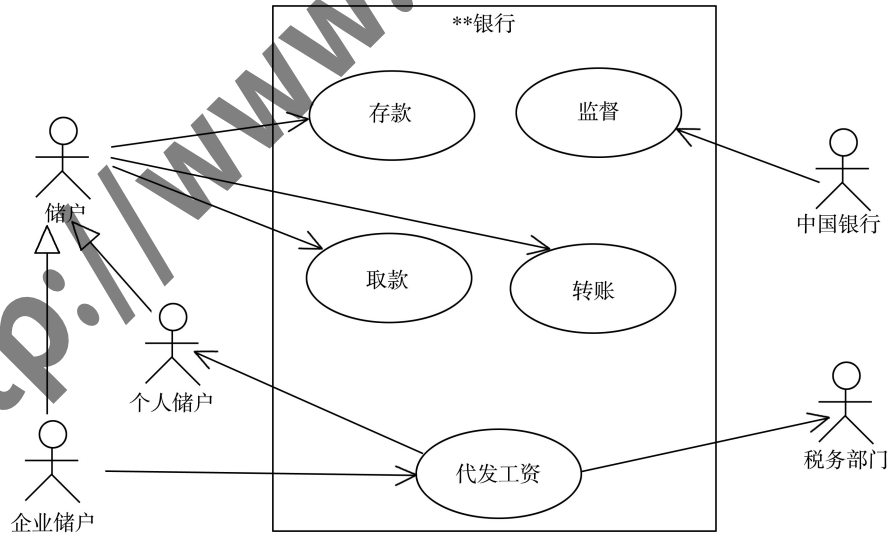

图 2.5　企业储户、个人储户和储户的泛化关系

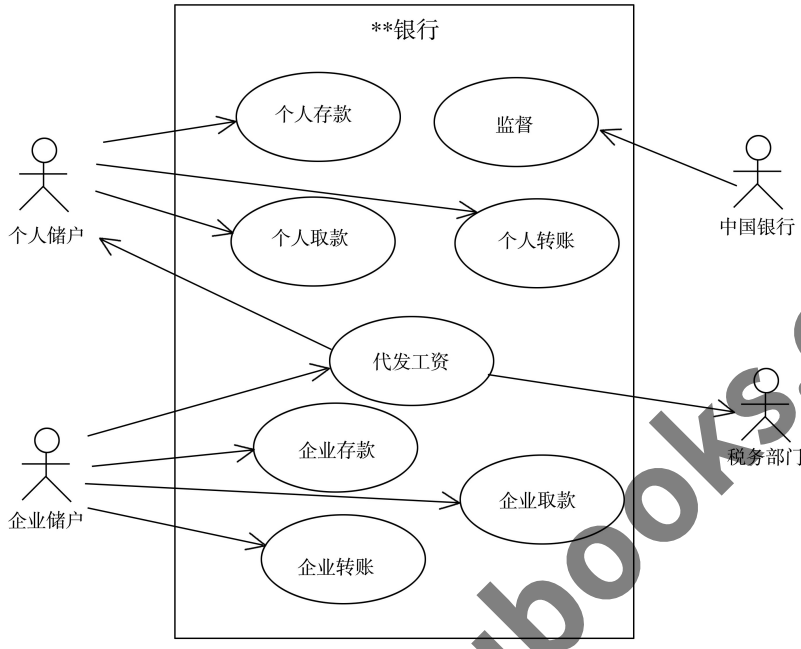

图 2.6　个人储户和企业储户不同的业务用例

以上的步骤，是一个基本的步骤，需要记住的是：组织价值的提供者，最权威的是组织本身，作为需求分析师，我们的职责是帮助组织挖掘自己的价值，而不是闭门造车。如图 2.1～图 2.6 所示，企业储户看重的银行价值，可能是贷款，而不是存款。因此，企业贷款应该是一个重要的用例。另外，组织价值的发现，并不是一个简单的事情，往往只有组织的上层才能看清楚。省略掉个人储户，图 2.7 展示的企业储户的业务用例更合理。

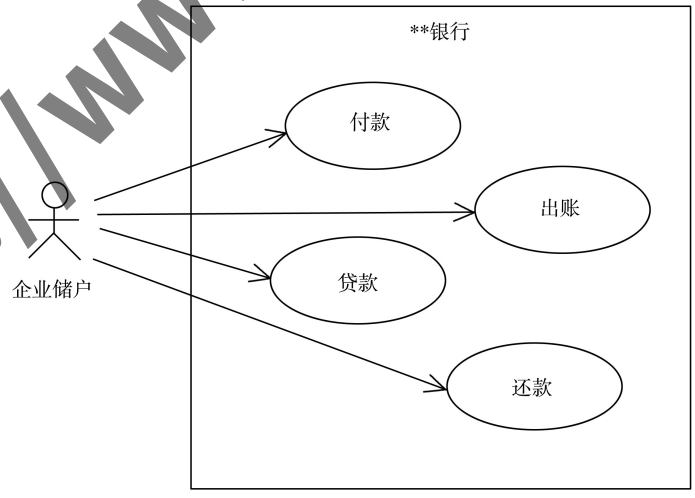

图 2.7　企业储户的价值

事实上，作为银行，对企业的价值不仅如此，账户信息查询、投资理财、信用证业务、国债、协定存款、代理汇兑等都是卖点，在此不再赘述。

一个完整的业务用例，必须独立地实现一次价值。