第五章 UML 统一建模语言

5.1 UML 概述

软件工程领域在1995年至1997年取得了前所未有的进展,其成果超过软件工程领域过去 15 年来的成就总和。其中最重要的、具有划时代重大意义的成果之一就是统一建模语言— UML (Unified Modeling Language)的出现。在世界范围内,至少在近 10 年内，UML 将是面向对象技术领域内占主导地位的标准建模语言。UML 是软件界第一个统一的建模语言，已成为国际软件界广泛承认的标准，应用领域很广泛，可用于商业建模（Business Modeling）,软件开发建模的各个阶段，也可用于其它类型的系统。它是一种通用(General)建模语言，具有创建系统的静态结构和动态行为等多种结构(Construction)模型的能力，具有可扩展性和通用性，适合于多种、多变结构的建模。

UML 的价值在它综合并体现了世界上面向对象方法实践的最好经验，支持用例驱动(use-case driven),以架构为中心(architecture-centric)以及递增(incremental)和迭代(iterative)地进行软件开发。

5.1.1 UML 的形成从二十世纪八十年代初期开始，众多的方法学家都在尝试用不同的方法进行分析和设计，有少数几种方法开始在一些关键性的项目中中发挥作用。

到了二十世纪九十年代中期，出现了第二代面向对象方法，具有代表性的有 G.Booch 的面向对象的开发方法，P.Coad 和 E.Yourdon 的面向对象的分析（OOA）和面向对象的设计（OOD），J Rumbaugh 等人的对象建模技术（OMT）及 Jacobson 的面向对象的软件工程（OOSE）等。此时，面向对象的方法已经成为软件分析和设计方法的主流。

1994 年 10 月 J Rumbaugh 和 G Booch 共同合作把他们的 OMT 和 Booch 方法统一起来，到 1995 年成为“统一方法”（Unified Method）版本 0.8。随后，Ivar Jacobson 加入，并采用他的用例(User case)思想,到 1996 年，成为“统一建模语言”版本 0.9。

1997 年 1 月，UML 版本 1.0 被提交给 OMG 组织，作为软件建模语言标准的候选。其后的半年多时间里，一些重要的软件开发商和系统集成商都成为“UML 伙伴”，如 IBM,Mircrosoft,HP 等.1997 年 11 月 7 日被正式采纳作为业界标准。UML 的形成过程如图 5.1 所示。

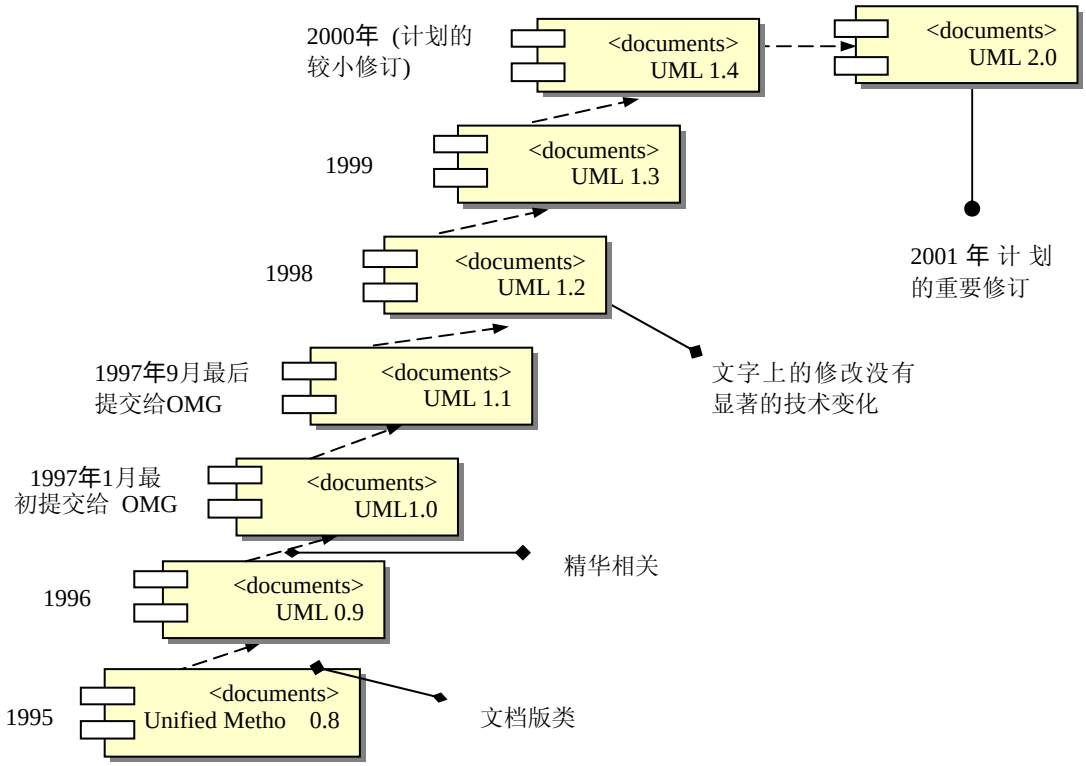


图5. 1

UML 的核心是建立系统的各类模型。模型是一个系统的完整的抽象，是人们对某个领域特定问题的求解及解决方案，对它们的理解和认识都蕴涵在模型中。

通常，开发一个计算机系统是为了解决某个领域特定问题，问题的求解过程，就是从领域问题到计算机系统的映射。

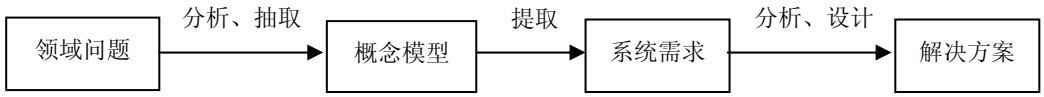


图 5.2 建模过程

5.1.2 UML 的主要内容

UML 是一种标准的图形化（可视化）的建模型语言，由元模型和图组成，图是 UML 的语法，元模型则是 UML 的语义，给出了图的含义。

1．UML 语义

UML 的语义通过元模型来精确定义。元模型为 UML 的所有元素在语法和语义上提供了简单、一致、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。

UML 支持各种类型的语义，如布尔、表达式、列表、阶、名字、坐标、这字符串和时间等，还允许用户自定义类型。

2．UML 表示法

UML 表示法定义了图形符号的表示，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。这些图形符号和文字所表达的是应用级的模型,在语义上它是 UML 元模型的实例。

UML 表示法分为通用表示和图形表示两部分组成；

(1) 通用表示包括

字符串 表示有关模型的信息。

名 字 表示模型元素。

标 号 赋予图形符号的字符串。

特定字串 赋予图形符号的特性。

类型表达式 声明属性变量及参数。

定 制 是一种用已有的模型元素来定义新模型元素的机制。(2) 图形表示

UML 由视图(views),图(Diagrams),模型元素(Model elements)和通用机制(general mechanism)等几个部分构成。

5.1.3 UML 的图形表示

UML 建模型语言的描述方式是以标准的图形表示为主的，是由视图、图、模型元素和通用机制构成的层次关系。

1．视图

一个系统应从不同的角度（视图）进行描述,一个视图由多个图（Diagrams）构成，它不是一个图表（Graph）,而是在某一个抽象层上，对系统的抽象表示。如果要为系统建立一个完整的模型图，需定义一定数量的视图，每个视图表示系统的一个特殊的方面。另外，视图还把建模语言和系统开发时选择的方法或过程连接起来。图 5.描述了常用的 UML 视图。

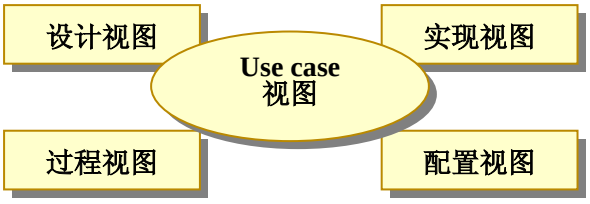


图 5.3 UML 的视图

⑴ Use case View　描述系统的外部特性、系统功能等。

⑵ Design View　描述系统设计特征，包括结构模型视图和行为模型视图，前者描述系统的静态结构(类图、对象图)，后者描述系统的动态行为(交互图、状态图、活动图)。

⑶ Process View　表示系统内部的控制机制。常用类图描述过程结构，用交互图描述过程行为。

⑷ Implementation View　表示系统的实现特征，常用构件图表示。

⑸ Deployment View　配置视图描述系统的物理配置特征。用配置图表示。

2．图

由各种图片（Graph）构成，用来描述一个视图的内容。UML 语言定义了 9 种不同的图的类型，把它们有机的结合起来就可以描述系统的所有视图。

标准建模语言 UML 的重要内容可以由下列五类图(共 9 种图形)来定义:

⑴ 用例图（Use Case diagram）　从用户角度描述系统功能,并指出各功能的操作者。

⑵ 静态图(Static diagram)　表示系统的静态结构。包括类图、对象图、包图。

⑶ 行为图(Behavior diagram)　描述系统的动态模型和组成对象间的交互关系。包括状态图、活动图。

⑷交互图(Interactive diagram)　描述对象间的交互关系。包括顺序图、合作图。

⑸实现图(Implementation diagram) 用于描述系统的物理实现。包括构件图、部件图。

3．模型元素

代表面向对象中的类，对象，关系和消息等概念，是构成图的最基本的常用的概念。一个模型元素可以用在多个不同的图中，无论怎样使用，它总是具有相同的含义和相同的符号表示。

4．通用机制

用于表示其他信息，比如注释，模型元素的语义等。另外，它还提供扩展机制，使用 UML 语言能够适应一个特殊的方法（或过程），或扩充至一个组织或用户。

5.1.4 UML 的特点

1. 统一标准

UML 统一了 Booch、OMT 和 OOSE 等方法中的基本概念，已成为 OMG 的正式标准，提供了标准的面向对象的模型元素的定义和表示。

2. 面向对象

UML 还吸取了面向对象技术领域中其他流派的长处。UML 符号表示考虑了各种方法的图形表示,删掉了大量易引起混乱的、多余的和极少使用的符号,也添加了一些新符号。

3. 可视化、表示能力强

系统的逻辑模型或实现模型都能用 UML 模型清晰的表示，可用于复杂软件系统的建模

4. 独立于过程

UML 是系统建模语言，独立于开发过程。

5. 易掌握、易用

由于 UML 的概念明确，建模表示法简洁明了，图形结构清晰，易于掌握使用。

5.2 通用模型元素

模型元素是 UML 构造系统各种模型的元素，是 UML 构建模型的基本单位。通用模型元素分为以下两类：

⑴ 基元素

是已由 UML 定义的模型元素。如：类、结点、构件、注释、关联、依赖和泛化等。

⑵ 构造型元素

在基元素的基础上构造的新的模型元素，是由基元素增加了新的定义而构成的，如扩展基元素的语义（不能扩展语法结构）。也允许用户自定义。构造型用括在双尖括号《》中的字符串表示。

目前 UML 提供了 40 多个预定义的构造型元素。如《使用》、《扩展》等。

5.2.1 模型元素

可以在图中使用的概念统称为模型元素。模型元素在图中用其相应的视图元素（符号）表示，常用的元素符号表示如下。利用视图元素可以把图象直观的表示出来，一个元素（符号）可以存在于多各不同类型的图中，但是具体以怎样的方式出现在哪种类型的图中要符合（依据）一定的规则。

给出了类、对象、结点、包和组件等部分模型元素的符号图例：

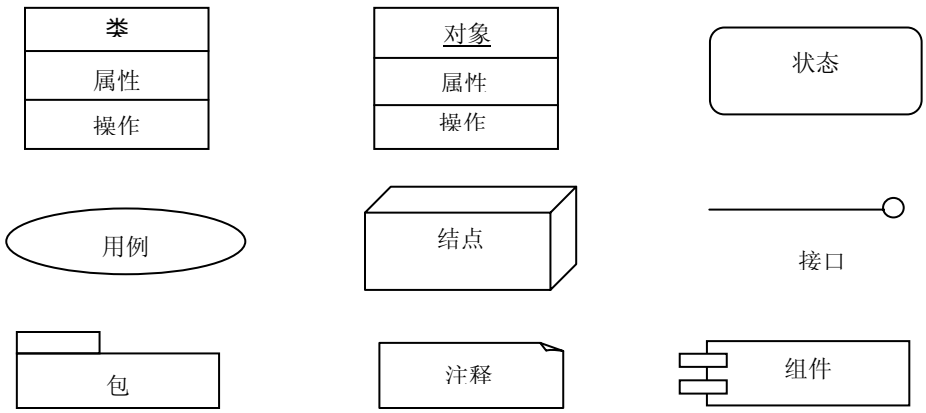


图 5.3 模型元素

模型元素与模型元素之间的连接关系也是模型元素，常见的关系有关联（association）、泛化（generalization）、依赖(dependency)和聚合(aggregation)，其中聚合是关联的一种特殊形式。这些关系的图示符号如图 5.4 所示。

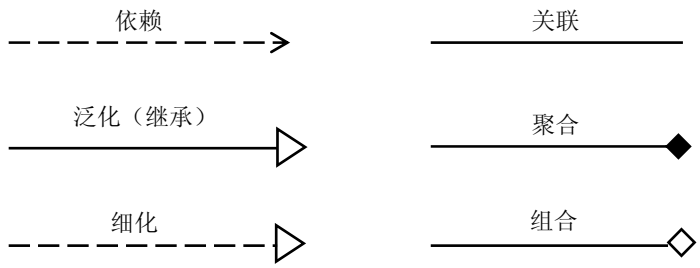


图5. 4 连接关系模型元素

关联：连接（connect）模型元素及链接(link)实例。
依赖：表示一个元素以某种方式依赖于另一种元素。
泛化：表示一般与特殊的关系，即“一般”元素是“特殊”关系的泛化。
聚合：表示整体与部分的关系。除了上述的模型元素外，模型元素还包括消息，动作和版类（stereotype）等。

5.2.2 约束

UML 中提供了一种简便、统一和一致的约束（constrraint）。是各种模型元素的一种语义条件或限制。一条约束只能应用于同一类的元素。约束的表示

如果约束应用于一种具有相应视图元素的模型元素，它可以出现在它所约束元素视图元素的旁边。

通常一个约束由一对花括号括起来（{constrraint}），花括号中为约束内容（图 5.5）。

如果一条约束涉及同一种类的多个元素，则要用虚线把所有受约束的元素框起来，并把该约束显示在旁边（如或约束）。如果在某个图中都要使用某个约束，可以在工具中声明该约束，也可以在图中边定义边使用。

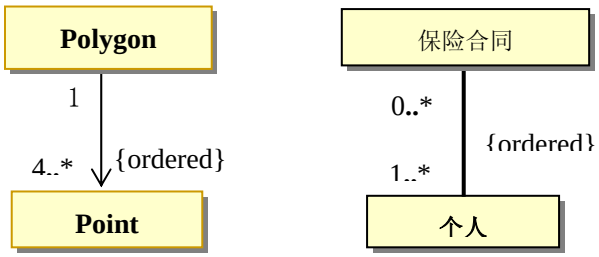


图5.5

通常约束分为对泛化的约束和对关联的约束几类：

1. 对泛化的约束

完整，不完整，不相交和覆盖是四种应用于泛化的约束，显示在大括号里，若有多个约束，用逗号隔开。如果没有共享，则用一条虚线通过所有继承线，并在虚线的旁边显示约束，如图 5.6 所示

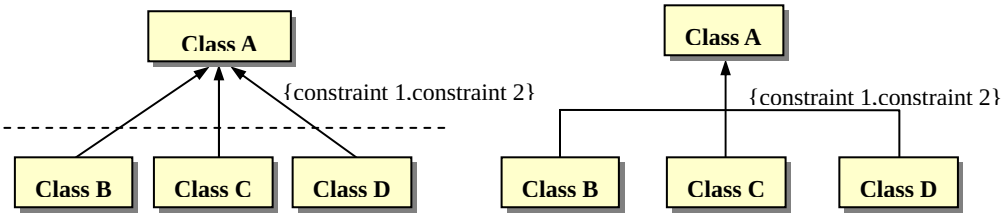


图5.6 对泛化的约束的两种表示方法

对泛化有以下常用的约束：

⑴. complete: 说明泛化中所有子元素都已在模型中说明，不允许再增加其它子元素。

⑵. disjoint: 父类对象不能有多于一个型的子对象。

⑶. incomplete: 说明不是泛化中所有子元素都已说明，允许再增加其它子元素。

⑷. overlapping :给定父类对象可有多于应该型的子对象，表示重载。

2. 关联的约束

对关联有以下常用的约束：

⑴. Implici :该关联只是概念性的，在对模型进行精化时不再用。

⑵. ordered: 具有多重性的关联一端的对象是有序的。

⑶. changeable: 关联对象之间的链(Link)是可变的（添加、修改、删除）。

⑷. addonly: 可在任意时刻增加新的链接。

⑸. Frozen :冻结已创建的对象，不能再添加、删除和修改它的链接。

⑹. xor: “或约束”，某时刻只有一个当前的关联实例。

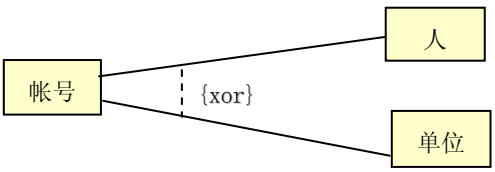


图5.7 对象类的xor关联

此外，还有对消息，链接角色和对象的约束和自定义约束；对消息，链接角色和对象的约束有九个约束应用于交互：全局的、局部的、参数、自我、投票、广播、创建、注销和临时。

自定义约束则是指用户可按照一定的规则自定义约束。

5.2.3 依赖关系

依赖关系描述的是两个模型元素（类，组合，用例等）之间的语义上的连接关系，其中一个模型元素是独立的，另一个模型元素是非独立的（或依赖的），它依赖于独立的模型元素，如果独立的模型元素发生改变，将会影响该模型元素的模型元素，比如，某个类中使用另一个类中的对象作为操作中的参数，则这二者之间就具有依赖关系。

图示具有依赖关系的两个模型元素时，用带箭头的虚线连接，箭头指向独立的类，箭头旁边还可带一个版类标签，具体说明依赖的种类。如图 5.8 表示类 A 依赖于类 B，其依赖关系为友元。

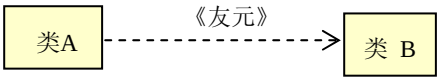


图5.8 依赖

依赖的形式可能是多样的，针对不同的依赖的形式，依赖关系有不同的变体(varieties)，常有以下依赖关系的变体：

（1）抽象(abstraction)：从一个对象中提取一些特性，并用类方法表示。

（2）绑定(binding)：为模板参数指定值，以定义一个新的模板元素。

（3）组合(combination)：对不同类或包进行性质相似融合。

（4）许可(permission)：允许另一个对象对本对象的访问。

（5）使用(usage)：声明使用一个模型元素需要用到已存在的另一个模型元素，这样才能正确实现使用者的功能(包括调用、实例化、参数、发送)。

（6）跟踪(trace)：声明不同模型中元素的之间的存在一些连接。

（7）访问或连接(access)：允许一个包访问另一个包的内容。

（8）调用(call)：声明一个类调用其他类的操作的方法。

（9）导出(derive)：声明一个实例可从另一个实例导出。

（10）友元(friend)：允许一个元素访问另一个元素，不管被访问的元素是否具有可见性。

（11）引入(import)：允许一个包访问另一个包的内容并被访问组成部分增加别名。

（12）实例(instantitate)：关于一个类的方法创建了另一个类的实例声明。

（13）参数(parameter)：一个操作和它参数之间的关系。

（14）实现(realize)：说明和其实之间的关系。

（15）精化(refine)：声明具有两个不同语义层次上的元素之间的映射。

（16）发送(send)：信号发送者和信号接收者之间的关系

5.2.4 细化

有两个元素 A 和 B，若 B 元素是 A 元素的详细描述，则称 B,A 元素之间的关系为 B 元素细化 A 元素。细化关系表示了元素之间更详细一层的关系描述。

细化与类的抽象层次有密切的关系，人们在构造模型时不可能一下就把模型完整，准确的构造出来，而是要经过逐步细化的过程，要经过逐步求精的过程。

在建立一个应用问题的类结构时，在系统分析中要先建立概念层次上的类图，用于描述应用域的概念，这种描述是初步的,不详细的描述,在进入系统设计时,要建立说明层次的类图,该类图描述了软件接口部分,它比概念层次的类图更详细;在进入系统实现时要建立实现层次的类图,描述类的实现，实现层次的类图比说明层次的类图更要详细。

两个元素细化的关系用两个元素之间的空心三角形箭头的虚线来表示,箭头的方向由细化了的元素指向被细化了的元素。如图 5.9 所示，类 B 是类 A 细化的结果。

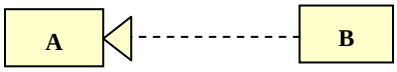


图5.9 细化

5.2.5 注释

注释用于对 UML 语言的元素或实体进行说明，解释和描述。通常用自然语言进行注释。

注释由注释体和注释连接组成。注释体的图符是一个矩形，其右上角翻下，矩形中表注要注释的内容。注释连接用虚线表示，它把注释体与被注释的元素连接起来。

注释的表示如图 5.10，“这是一个类”为注释体，对类“人员”进行注释。在 UML 的各种模型图中，凡是需要注释的元素或实体均可加注释。

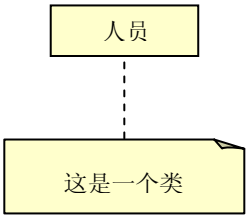


图5.10 注释

5.3 用例模型

在传统的和面向对象的开发过程中，常常通过典型的使用情景来了解需求，但这往往不够，也很难建立正式的需求文档。

1992 年由 Jacobson 提出了 Use case 的概念及可视化的表示方法—Use case 图，受到了 IT 界的欢迎，被广泛应用到了面向对象的系统分析中。用例驱动的系统分析与设计方法已成为面向对象的系统分析与设计方法的主流。

用例模型(Use case model)描述的是外部执行者(Actor)所理解的系统功能。用例模型主

要用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。

首先，它描述了待开发系统的功能需求，其次，它将系统看作黑盒，从外部执行者的角度来理解系统，它驱动了需求分析之后各阶段的开发工作，不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统，从而影响到开发工作的各个阶段和 UML 的各个模型。

5.3.1 用例图

用例模型由若干个用例图构成，用例图中主要描述执行者和用例之间的关系。在 UML 中构成用例图的主要元素是用例和执行者及其它们之间的联系。创建用例模型的工作包括：定义系统、确定执行者和用例、描述用例、定义用例间的关系、确认模型。

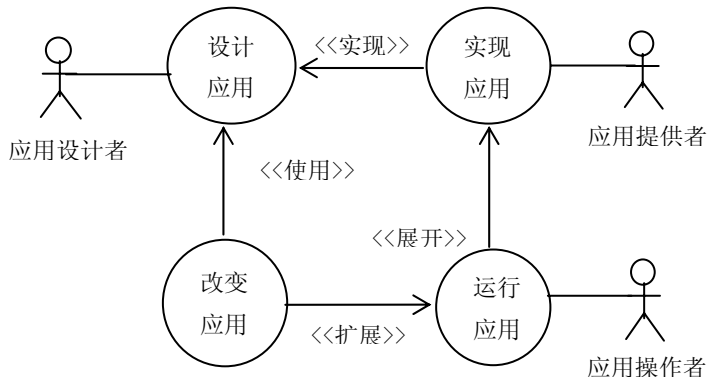


图5. 11 应用生命周期用例图

5.3.2 确定执行者

用例图中包含执行者、用例和连接等三种模型元素。执行者(Actor)是指用户在系统中所扮演的角色。执行者在用例图中是用类似人的图形来表示，但执行者未必是人。例如，执行者也可以是一个外界系统，该外界系统可能需要从当前系统中获取信息，与当前系统进行交互。

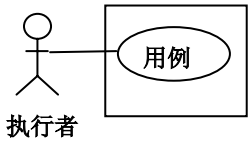


图 5.12 描述了自动售货机系统用例图，执行者有“顾客”、“供货人”和“收银员”。矩形框表示系统的边界，用于划分系统的功能范围。如何确定执行者是画用例图首要问题，首先要与系统的用户进行广泛而深入的交流，明确系统的主要功能，以及使用系统的用户责任等。

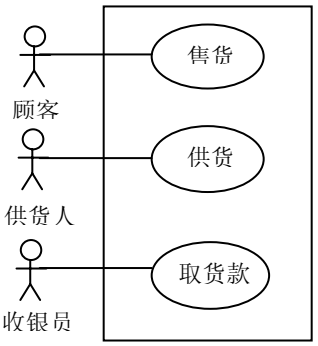


图5. 12 自动售货机系统用例图

可以通过回答以下问题来确定：

1．谁使用系统的主要功能（主执行者）？

2．谁需要从系统获得对日常工作的支持和服务？

3．需要谁维护管理系统的日常运行（副执行者）？

4．系统需要控制哪些硬件设备？

5．系统需要与其它哪些系统交互？

6．谁需要使用系统产生的结果（值）？

识别出的角色都应该用文字形式或角色描述模板来作进一步的描述，描述模板的格式如图 5.13 所示。

例如，有一个自动取款机 ATM（Auto Trade Machine）系统，为储户提供 24 小时的服务，储户需要提款时，必须将银行信用卡插入 ATM 机，并输入正确的口令后才能取款。

通过回答上述问题来识别角色：

问：1．谁使用 ATM 系统的主要功能（提款）？

答：储户。

问：２．谁需要从 ATM 系统获得对日常工作的支持和服务？

答：出纳员？（不肯定）

问：3．需要谁维护管理系统的日常运行？

答：银行工作人员、系统工程师。

问：4．ATM 系统需要控制哪些硬件设备？

答：银行信用卡。

问：5．系统需要与其它哪些系统交互？

答：不清楚。

问：6．谁需要使用 ATM 系统产生的结果？

答：银行会计、储户。

通过回答以上问题，得到可能的角色有：储户、出纳员、银行工作人员、系统工程师、银行信用卡、银行会计。

对于问题 1、4、5、6 的回答没有什么问题。问题 2 的答案“出纳员”，经过分析，与 ATM 系统的关系不大，并没有因为 ATM 的的存在而减轻出纳员的工作量，即并非需要从 ATM 系统获得对日常工作的支持和服务。问题 3 所确定的“银行工作人员”主要是维护 ATM 的信息及 ATM 的一般故障的维修，而“系统工程师”则是掌握和熟悉 ATM 技术及系统配置的工程技术人员，事实上对系统的日常维护工作，只需要银行工作人员就行了。

根据以上分析，最后确定 ATM 系统的执行者为：储户、银行工作人员、银行信用卡、银行会计。图 5. 14 给出了角色“储户”的描述模板。

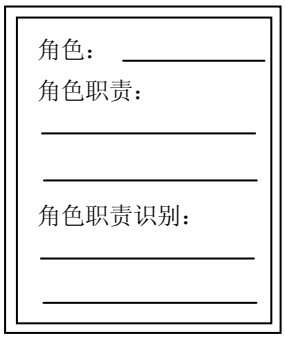


图5.13 角色描述模板

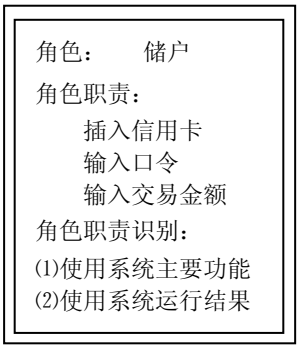


图5.14 储户角色描述模板

5.3.3 确定用例

本质上讲，一个用例(use case)是用户与计算机之间的一次典型交互作用。在 UML 中,用例被定义成系统执行的一系列动作（功能），即 Use Case 是对系统用户需求的描述，表达了系统的功能和所提供的服务。UML 中的用例用椭圆形表示，用例的名字写在椭圆的内部或下方，用例位于系统边界的内部，角色与用例之间的关联关系（或通信关联关系）用一条直线表示。

1．用例的特征

概括地说,用例具有以下特征:

⑴用例捕获某些用户可见的需求，实现一个具体的用户目标。

⑵用例由执行者激活，即用例总是由执行者启动的。并提供确切的值给执行者。

⑶用例可大可小,但它必须是对一个具体的用户目标实现的完整描述。

如图 5.12 中确定的用例有：“售货”、“供货”和“取货款”。用例图还可进一步细化。

2．识别用例

识别用例的方法有多种，其基本的出发点都可以从系统的功能考虑，可以根据以上特征确定用例，也可以通过回答以下问题来确定用例：

⑴与系统实现有关的主要问题是什么？

⑵系统需要哪些输入/输出？这些输入/输出从何而来？到哪里去？

⑶执行者需要系统提供哪些功能？

⑷执行者是否需要对系统中的信息进行读、创建、修改、删除或存储？如果首先确定了系统的角色，也可以通过角色来识别用例。

例如：在 2.2.3 节所给出的实例—医院病房监护系统系统，通过分析确定了系统有以下角色：值班护士，医生，病人，标准病症信号库。

通过回答问题来识别用例：

问：⑴与系统实现有关的主要问题是什么？

答：中央监护，将病人的病症信号与标准的病症信号库里的病症信号的正常值进行比

较，当病症出现异常时系统自动报警，自动更新病历并打印病情报告。

问：⑵．系统需要哪些输入/输出？这些输入/输出从何而来？到哪里去？

答：输入：

① 病症信号，由“病症监护”，将从病人采集到的病症信号，实时地传送到中央监护系统。

② 病症信号的正常值，“中央监护”将病人的病症信号值与标准的病症信号库里的病症的正常值进行比较。

输出：

① 报警信号，当病症出现异常时“中央监护”系统自动报警。

② 病情报告，根据医生要求或病症信号异常时，“病情报告管理”自动打印病情报告。

问：⑶执行者需要系统提供哪些功能？

答：医生和值班护士需要查看病情报告、病历并进行打印。

问：⑷执行者是否需要对系统中的信息进行读、创建、修改、删除或存储？

答：病人通过“病症监护”产生如血压、脉搏、体温等病症信号。

通过回答问题，得到系统用例：

中央监护，病症监护，提供标准病症信号，病历管理，病情报告管理。

用例确定后，还应该用文字或者用例描述模板进行描述。下面是对医院病房监护系统系统的文字描述。

1．中央监护

a 分解信号　将从病症监护器传送来的组合病症信号分解为系统可以处理的信号。

b 比较信号　将病人的病症信号与标准信号比较 。

c 报警　如果病症信号发生异常（即高于峰值），发出报警信号。

d 数据格式化　将处理后的数据格式化以便写入病历库 。

2．病症监护

e 信号采集　采集病人的病症信号。

f 模数转化　将采集来的模拟信号转化为数字信号。

g 信号数据组合　将采集到的脉搏，血压等信号数据组合为一组信号数据。

h 采样频率改变　根据病人的情况改变监视器采样频率。

3．提供标准病症信号　　i（此用例不分解）

4．病历管理

j 生成病历

k 查看病历

l 更新病历

m 打印病历

5．病情报告

n 显示病情报告　在显示器上显示病情

o 打印病情报告　在打印机打印病情报告

5.3.4 用例之间的关系

用例（Use Case）除了与执行者有联系外，用例之间也存在一定的联系，用例之间通常有扩展、使用、组合三种关系，使用（sue）和扩展（extend）,它们都是继承关系的不同形式，分别用《sue》和《extend》表示。组合则是把相关的用例打成包，当作一个整体看待。

使用关系是一种泛化关系，当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。图 5.15 的自动售货机系统中“供货”与“取货款”这两个用例的开始动作都是打开机器，而它们最后的动作都是关闭机器。因此，将开始动作抽象为“打开机器”用例，将最后的动作抽象为“关闭机器”用例，“供货”与“取货款”用例在执行时必须使用这两个用例。

2．扩展关系

扩展也是一种泛化关系，即向一个用例中加入一些新的动作后构成了另外一个用例，这两个用例之间的关系就是扩展关系。后者是继承前者的一些行为得来的，通常把前者称为基本用例，而把后者称为扩展用例。基本用例通常是一个独立的用例，一个扩展用例是对基本用例在对某些“扩展点”的功能的增加。被扩展的用例是一般用例，扩展的用例是特殊用例。如在图 5.15 中，“售货”是一个基本用例，定义的是售罐装饮料，而用例“售散装饮料”则是继承了“售货”的一般功能的基础上进行修改，因此是“售货”的扩展。

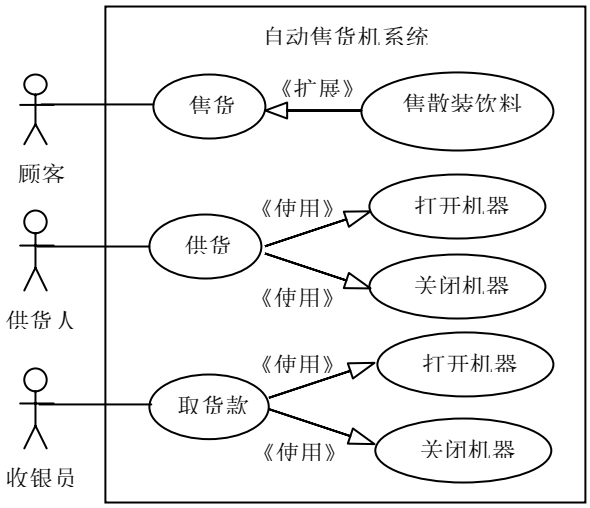


图 5.15 含有使用和扩展关系的用例图

用例模型是获取需求，规划和控制项目迭代过程的基本工具。在建立用例模型时还要考虑用例的数目；用例数量大则每个用例小，小的用例易执行实施方案，但用例数量过多则使用例图显得过于繁杂，因此要根据系统大小，适当选择用例数目。

5.3.5 用例图实例

例 1 画出金融贸易系统的 Use Case 图。

按照获取执行者的方法，确定了 4 种执行者：贸易经理、营销人员、销售人员和记帐系统。

在该系统所确定的用例中，由贸易经理确定“设置边界”，所谓边界，是指金融贸易的范围。基本的用例是“进行交易”，用例“交易估价”与“风险分析”都要使用公共的“评价”动作，所以将“评价”作为一个独立的用例，

在交易过程中，可能会出现超越交易范围或超过交易量，这时，允许对用例“进行交易”进行修改，即用例“超越边界”是用例是“进行交易”的扩展。

虽然大多数执行者是人，而图5.16中执行者“记帐系统”是一个外部系统，它需要与用例“更新帐目”进行交互。“交易估价”与“风险分析”都要对交易进行评价，因此，“评价”用例要被“交易估价”与“风险分析”使用。

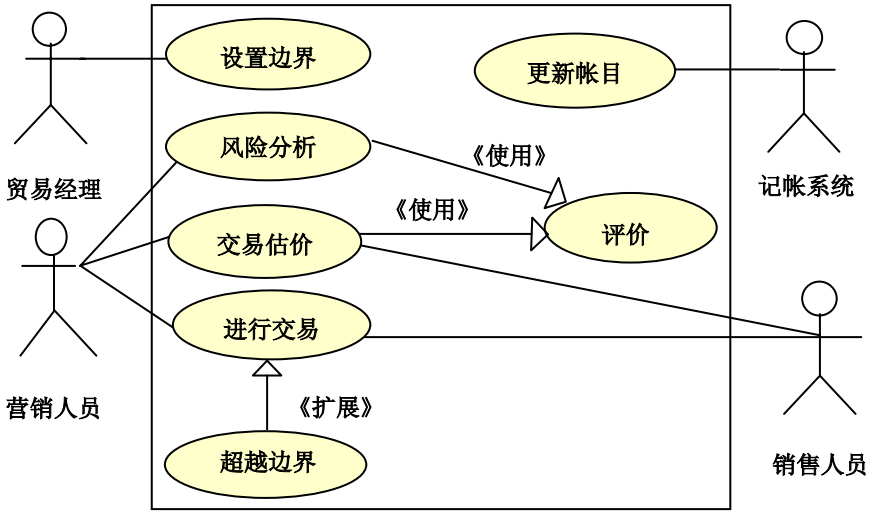


图5.16 金融贸易系统用例图

例 2 建立项目与资源管理系统（PRMS）的 Use case 图系统的主要功能是：项目管理，资源管理和系统管理。项目管理包括项目的增加、删除、更新。资源管理包括对资源和技能的添加、删除和更新。系统管理包括系统的启动和关闭，数据的存储和备份等功能。

一．建立用例图

1．分析确定系统的执行者(角色)

执行者是对系统外的对象描述，是用户作用于系统的一个角色，它有自己的目标，通过与系统的交互来实现，交互包括信息交换和与系统的协同。

执行者可以是人，也可以是一个外部系统。通过回答 5.3.2 的问题来，确定本系统的 4 类角色：项目管理员、资源管理员、系统管理员、备份数据系统。

2．确定用例

Use case 是对系统的用户需求（主要是功能需求）的描述，Use case 描述了系统的功能和和所提供的服务。过回答 5.3.3 的问题，确定本系统用例是：项目管理，资源管理和系统管理。

3．画出用例模型概图

Use case 图是系统的一个功能模型，在绘制 Use case 图时，需要认真考虑它的粒度和抽象层次。按照抽象层次，Use case 图可以划分为系统层（最高层）、子系统层和对象类层（最低层），系统层的 Use case 图也称为用例模型概图，描述了系统的全部功能和服务。图 5.17 是 PRMS 高层 Use Case 图,描述了该系统的一个最基本的模型。

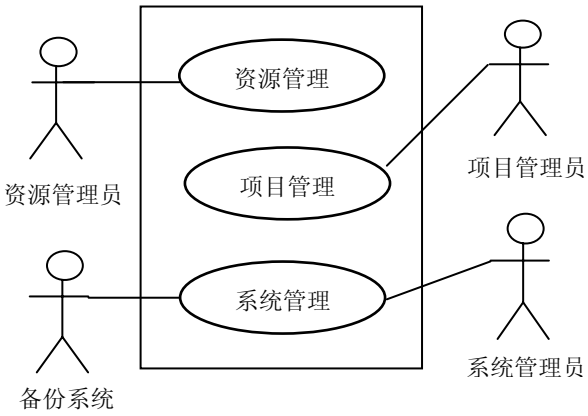


图5.17 PRMS高层Use Case图

4. 分解高层的用例图

对高层的用例图进行分解,进一步描述其子系统的功能及服务，并将执行者分配到各层次的 Use case 图中，子系统层又可以自顶而下不断精化，抽象出不同层次的 Use Case 图。图 5.18 是资源管理子系统的 Use Case 图，图 5.19、图 5.20 分别是项目管理 Use Case 图和系统管理的 Use Case 图。

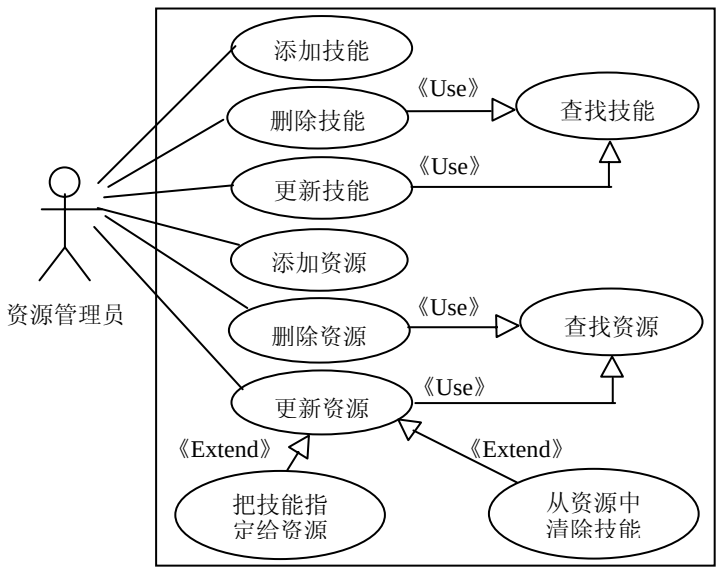


图 5.18 资源管理子系统的 Use Case 图

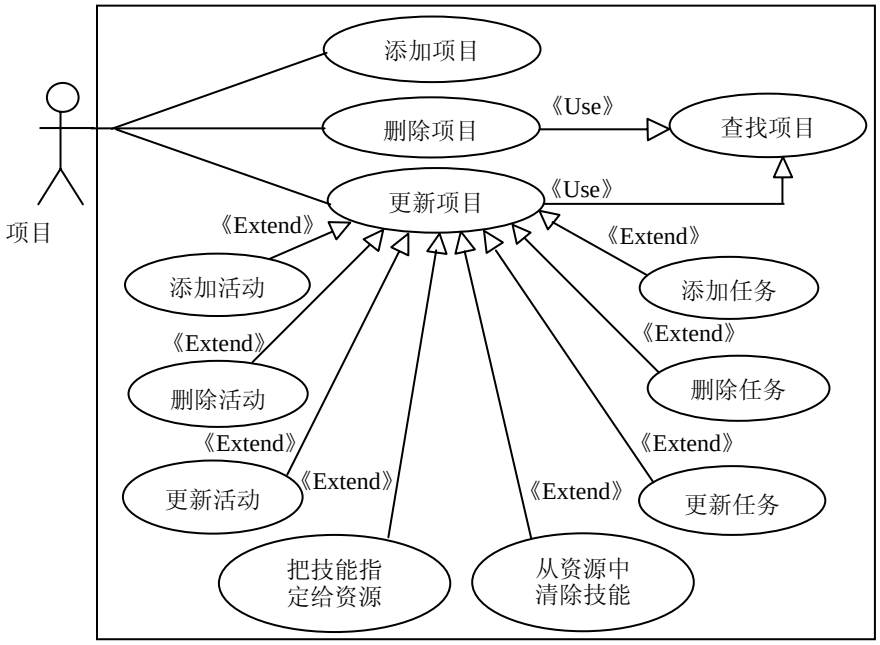


图5.19 项目管理Use Case图

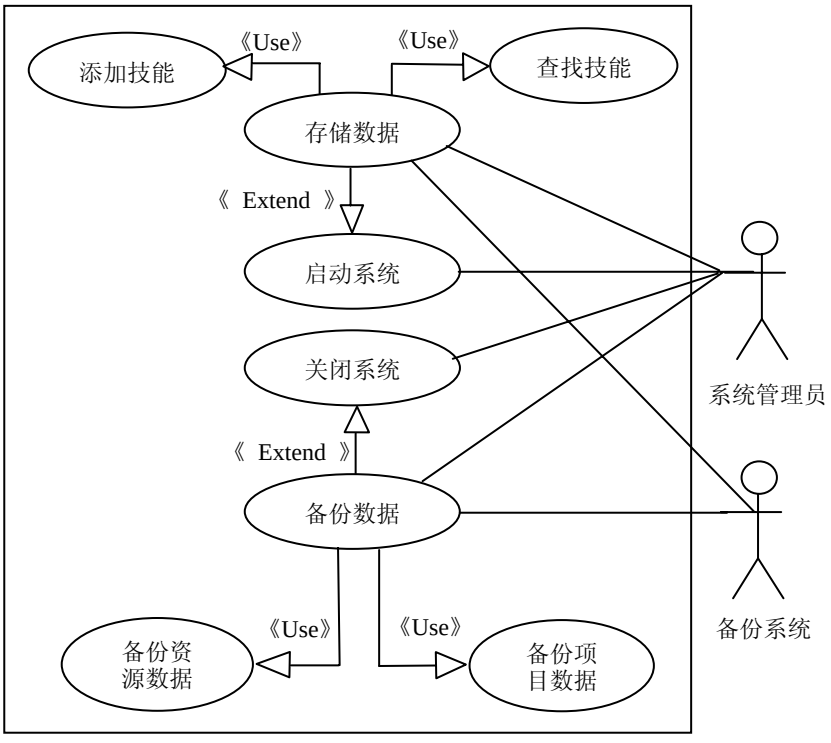


图5.20 系统管理Use Case图

二．执行者与用例的进一步描述

还应画出相应的执行者描述模板及用例描述模板。如图 5.21 为一个医院监护系统的两个角色描述模板。对用例也可用类似的模板来描述，或者用文字进行描述。

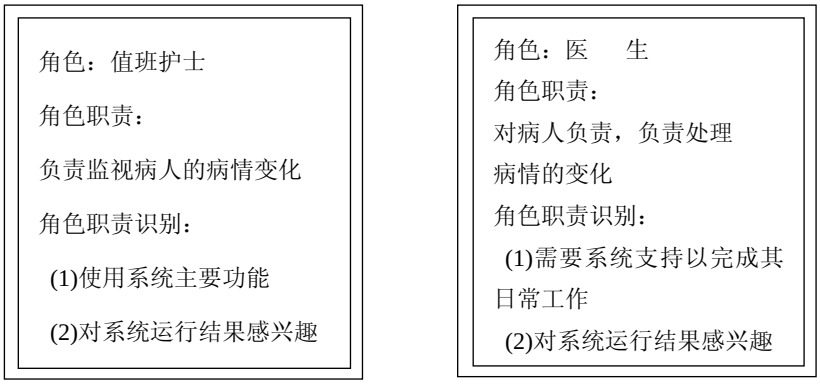


图 5.21 角色描述模板

5.4 建立静态模型

任何建模语言都以静态建模机制为基础,标准建模语言 UML 也不例外。所谓静态建模是指对象之间通过属性互相联系，而这些关系不随时间而转移。

类和对象的建模，是 UML 建模的基础。我们认为,熟练掌握基本概念、区分不同抽象层次以及在实践中灵活运用，是三条最值得注意的建模基本原则。

UML 的静态建模机制包括：

用例图(Use case diagram)

类图(Class diagram)

对象图(Object diagram)

包图(Package diagram)

构件图(Component diagram)

配置图(Deployment diagram)

用例图在 5.3 节已经讨论过，本节主要讨论类图、对象图和包图，构件图和配置图将在 5.6 节进行讨论。

5.4.1 类图与对象图

1．类图

类是所有面向对象的开发方法中最重要的基本概念。它是面向对象的开发方法的基础，可以说 UML 的基本任务就要识别系统所必需的类，并分析类之间的联系，并以此为基础，建立系统的其它模型。

类是面向对象模型的最基本的模型元素，图 5.22（a）是对类描述的图式。分为长式和短式。长式由类名、属性及操作三部分组成；类及类型名均用英文大写字母开头，属性及操作名为小写字母开头。属性的常见类型有：Char，Boolean，Double，Float，Integer,Object,Short,String 等。

类图(Class diagram)由系统中使用的类以及它们之间的关系组成，是描述系统的一种图式，类图是构建其它图的基础，是面向对象方法的核心。

2．对象图

与类密切相关的另外一个概念是对象，对象是类的实例(instance)，对象描述的图式如图 5.22（b）所示，对象的图式亦分长式和短式。用对象图(Object Diagram)来描述系统中对象及对象之间的联系。

对象图(Object diagram)是类图的变体，两者之间的差别在于对象图表示的是类图的一个实例。它及时具体的反映了系统执行到某处时，系统的工作状况。

对象图中使用的图示符号与类图几乎完全相同，只不过对象图中的对象名加了下划线，而且对象名后面可接以冒号和类名。

对象图也可用在协作图中作为其一个组成部分，用来反映一组对象之间动态协作关系。如图 5 .23(a)为一对象图实例。

UML 中的类图与对象图表达了对象模型的静态结构，能够有效地建立专业领域的计算机系统的对象模型。

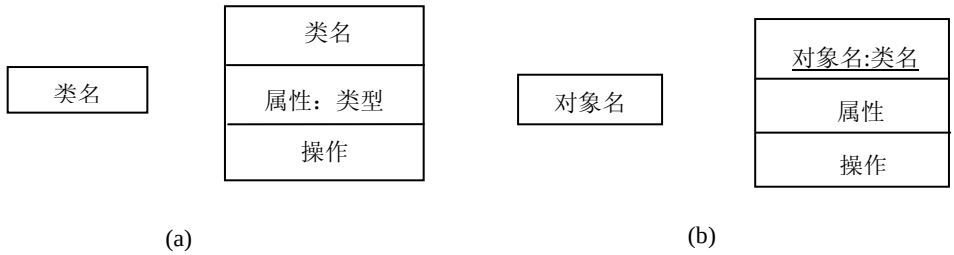


图5.22 类与对象的图式

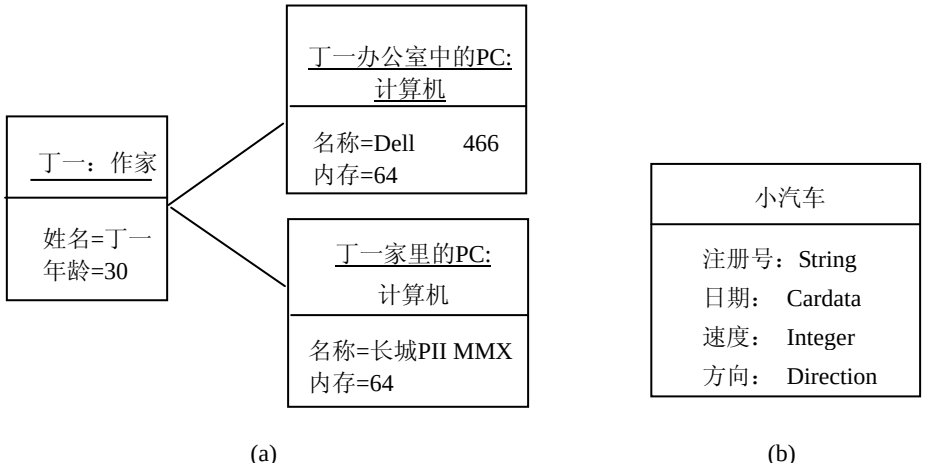


图 5.23 对象图

5.4.2 类的识别

在分析阶段，类的识别通常由分析员在分析问题域的基础上来完成。类的识别是面向对象方法的一个难点，但又是建模的关键。常用的方法有：

1．名词识别法

该方法的关键是识别问题域中的实体，由于问题域中的实体通常是以名词或者名词短语来描述的，通过对系统简要描述的分析，在提出实体对应名词的基础上识别类。

名词识别法的步骤如下：

⑴按照指定语言，对系统进行描述。描述过程应与领域专家共同合作完成，并遵循问题域中的概念和命名。

⑵从系统中标识名词、代词、名词短语。其中，单数名词（代词）可以标识为对象，而复数名词则可标识为类。

⑶识别确定（取、舍）类。并非所有列出的名词、代词、名词短语都是类，应根据一定的原则进行识别确定。例 1：确定银行网络系统 ATM(Auto Trade Machine)的类，

⑴系统简要描述

银行网络系统包括人工出纳和分行共享的自动出纳机；各分理处用自己的计算机处理业务（保存帐户、处理事务等）；各分理处与出纳站通过网络通信；出纳站录入帐户和事务数据；自动出纳机与分行计算机通信；自动出纳机与用户接口，接受现金卡；发放现金；打印收据；分行计算机与拨款分理处结帐。

要求系统正确处理同一帐户的并发访问；网络费用平均摊派给各分理处。

⑵类的识别

采用名词识别法：检查问题陈述中的所有名词，得到初始类：

软件	银行网络	分行计算机	系统	分行	出纳站
出纳员	分理处	分理处计算机	自动出纳机	帐户	帐户数据
现金卡	事务	事务数据	用户	顾客	现金
收据	访问	费用	安全措施	记录保管	

⑶根据下述原则进一步确定类：

① 去掉冗余类：如两个类表述同一信息，应保留最具有描述能力的类，如“用户”与“顾客”是重复的描述，由于“顾客”更具有描述性，故保留它，删除“用户”。

②去掉不相干的类：删除与问题无关或关系不大的类，如“费用”。

③ 删除模糊的类：有些初始类边界定义不确切，或范围太广，应该删除。如“系统”、“安全措施”、“记录保管”、“银行网络”。

④ 删除那些性质独立性不强的，而应该是类“属性”的候选类：如“帐户数据”、“收据”、“现金”、“事务数据”。⑤ 所描述的操作不适宜作为对象类，并被其自身所操纵，所描述的只是实现过程中的暂时的对象，应删去。如“软件”，“访问”。

最后确定的类为：

分行计算机　　分行　　出纳站　　出纳员　　分理处　　分理处计算机

自动出纳机　　帐户　　现金卡　　事务　　顾客

2．系统实体识别法

该方法不关心系统的运作流程及实体之间的通信状态，而只考虑系统中的人员、组织、地点、表格、报告等实体，经过分析将他们识别为类（或对象）。

被标识的实体有：系统需要存储、分析、处理的信息实体、系统内部需要处理的设备、与系统交互的外部系统、系统相关人员、系统的组织实体等。

下面举例说明系统实体识别法的应用：

例 2 有一个购物超市，顾客可在货架上自由挑选商品，由收款机收款，收款机读取商品上的条型码标签，并计算商品价格。收款机应保留所有交易的记录，以备帐务复查及汇总使用。

通过分析问题的陈述，确定以下几类实体：

⑴ 信息实体：交易记录、商品、税务信息、销售记录、货存记录。

⑵ 设备：收款机、条型码扫描器。

⑶ 交互系统：信用卡付款系统。

⑷ 人员职责：收款员、顾客、会计、经理。

⑸ 系统的组织实体：本例不考虑。

以上列出的实体，都可以直接识别为类。

3． 从用例中识别类

用例图本质上是一种系统的描述形式，可以根据用例的描述来识别类。通过用例识别类的方法与实体识别法很相似，只不过实体识别法是针对整个系统考虑，而用例识别法是分别对每一个用例进行识别，因此，用例识别法可能会识别出使用实体识别法未识别出来的类。

针对每个用例，可通过回答以下问题来识别类：

⑴ 对用例描述中出现了哪些实体？或者用例的完成需要哪些实体的合作？

⑵ 用例在执行过程中会产生并存储了哪些信息？

⑶ 用例要求与之关联的角色应该向该用例输入什么信息？

⑷ 用例向与之关联的角色输出什么信息？

⑸ 用例需要对哪些硬设备进行操作？

用例识别法的应用实例，请参考《软件工程实践》的第 2.3 节会议管理系统。

4．利用分解与抽象技术

无论使用哪种方法确定类时，还常使用分解和抽象两类技术：

⑴分解技术

通过上述方法，可以得到一系列反映问题域的类，但往往有的类还未被识别出来，有的“小类”可能被包含在“大类”中。实际上，所谓“大类”常以整体类和组合类的形式

出现，所以分解技术是对整体类和组合类进行分解的技术，可控制单个类的规模。通过分析对已标识出来的“大类”进行分解，得到新的类。例如已经识别了“汽车”类，可以通过分解技术得到“卡车”类、“小轿车”类、“客车”类等。

但在使用分解技术时一定要注意，分解出来的类一定要是系统所需要的相关的类，否则，分解就没有意义。

⑵抽象技术

如果在所识别的类中，存在着一些具有相似性的类，所谓相似性是指在信息和动作上的相似性。例如“汽车”类与“摩托车”类之间的相似性是都有“发动机”。

根据这些类的相似性建立抽象类，并建立抽象类与这些类之间的继承关系。如可以建立抽象类“机动车”类，而“汽车”类与“摩托车”类都是通过继承关系而得到的子类。

抽象类实现了系统内部的重用，很好地控制了复杂性，并为所有子类定义了一个公共的界面，使设计局部化，提高系统的可修改性和可维护性。

抽象技术时相似性不强时，要慎重考虑是否需要建立抽象类。抽象技术的掌握难度较大，需要有较多的实践经验。

总之，类的识别是 UML 建模的基础和关键，但又是比较难于掌握的步骤，只有通过对实际系统的分析和设计，逐步加深理解。

5.4.3 属性与操作识别

(1)属性(attribute)

属性用来描述类的特征，表示需要处理的数据。

属性定义：

visibility attribute-name : type = initial-value {property-string}

即： 可见性 属性名：类型=缺省值{约束特性}

其中：可见性(visibility)表示该属性对类外的元素是否可见。

分为：

public（+） 公有的，即模型中的任何类都可以访问该属性，用“+”号表示。

private（-） 私有的，表示不能被别的类访问，用“-”号表示。

protected（#） 受保护的，表示该属性只能被该类及其子类访问，用“#”号表示。

如果可见性未申明，表示其可见性不确定。

(2) 操作

对数据的具体处理方法的描述则放在操作部分，操作说明了该类能做些什么工作。操作通常称为函数，它是类的一个组成部分，只能作用于该类的对象上。操作定义：

visibility operating-name(parameter-list): return-type {property string}

即：可见性 操作名（参数表）；返回类型{约束特性}

其中：可见性同上。

参数表：参数名：类型，…Parameter-name :type =default-value

返回类型：表示操作返回的结果类型。

5.4.3 类之间的关系

在 UML 中，类之间的关系通常有关联(association)、聚集(aggregation)、泛化(generalization)、依赖(depending)和细化(refinement)。

1．关联

关联是两个或多个类之间的一个关系，链（link）是关联的具体体现。分为：

⑴常规关联

常见的关联是连接类之间的一条直线段，线段上标注关联的名字，可用实心的三角形表示关联名所指的方向。如图 5.24 描述了“公司”类和“员工”类之间的雇佣关系，此外，还用重数来描述这两个类之间连接的数量关系。

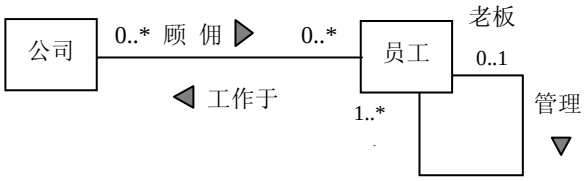


图 5.24 雇佣关联

重数的表示通常有：

- ● 表示“多个”，表示零或多个。
- ○ 表示“可选”，表示“0或者1”。

也可在连线上标注数字表示重数：

“1” — 表示只有1个；

“1+” — 表示1个或多个；

“0..*”或者“*”— 表示零或者多个；

“1..*” — 表示1或者多个；

“3 ..5” —表示 3个到5个之间

“2，4，15” — 表示2个，4个或15个

重数的缺省值为1。

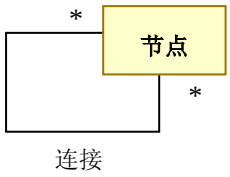


图5.25 递归关联

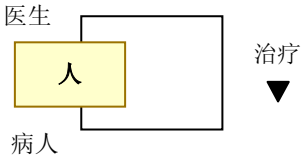


图5.26 带有职责的递归关联

此外，UML 中还允许一个类与自身关联，称为递归关联(Recursive association)，如图 5.24 中员工。图 5.25 是对递归关联的一般描述，图 5.26 则描述了具有职责的递归关联，即医生对病人进行治疗。

⑵多元关联

关联有二元关联(binary)、三元关联(ternary)及多元关联(higher order)。两个类之间的关联称为二元关联（图 5.27），三个类之间的关联称为三元关联（图 5.28）。对多元关联的描述也可用重数及角色。多元关联之间用大菱形连接。

图5.27 二元关联

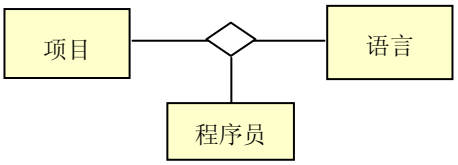


图5.28 三元关联

图 5.29 描述了一个人（Person）与嗜好（Hobby）的关联，一个人可以有多种嗜好。图 5.30 是一个具有重数的三元关联，重数 0 ••2 表示每个人(Person)在指定的年度(Year)，最多可以参加两个委员会（Committee），而重数 3 •• 5 表示每个委员会由 3 到 5 个委员组成，重数 1 •• 4 表示在一个委员会中，一个人的任期不超过 4 年。

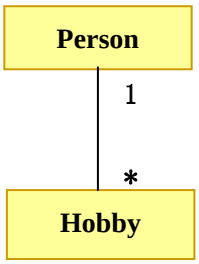


图 5.29 人与嗜好的关联

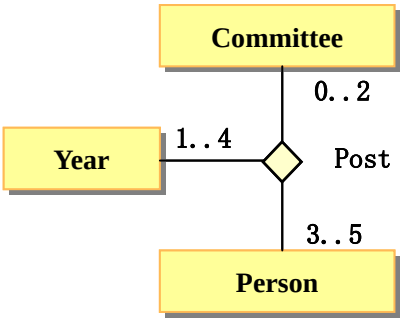
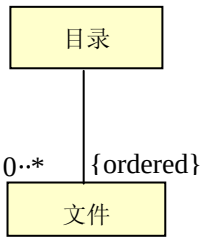


图 5.30 具有重数的三元关联的关联

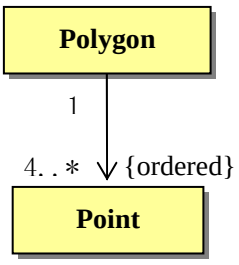
⑶有序关联

在关联的“多”端标注{ordered}指明这些对象是有序的(图 5.31)。关联可用箭头表示该关联使用的方向(单向或双向)，又称为导引或导航(navigation)。

图 5.31 (a) 描述一个目录下可以有多个有序的文件，而一个文件只属于一个目录。图 5.31 (b)表示多边形的多个顶点的有序关联。



(a)链接之间有明确的顺序



(b)单向关联

图5.31 有序关联

⑷**受限关联**(qualified association)

使用限定词对该关联的另一端的对象进行明确的标识和鉴别（图 5.32）。

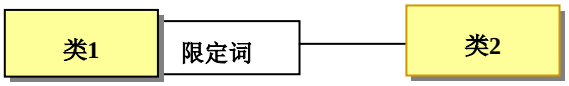


图5.32 受限关联

⑸**或关联**

如图 5.33 所示，描述了一个签定保险合同的类图，由于人和公司不能拥有同一份合同，因此描述为或的关系，用虚线连接两个关联并标注{or}。

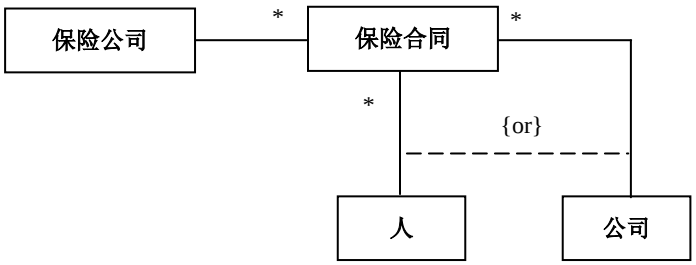


图5.33 或关联

⑹**关联类**

图 5.34 描述了一个授权的授权

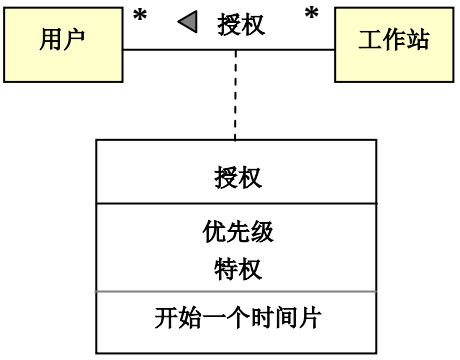


图 5.34 关联类

2．聚集(aggregation)关系聚集是一种特殊的关联，它指出类间的“整体-部分”关系。又分为：

⑴共享聚集(shared aggregation)

其“部分”对象可以是任意“整体”对象的一部分。当“整体”端的重数不是 1 时，称聚集是共享的。在“整体”端用一个小菱形表示共享聚集，如图 5.35 所示。

图 5.35 共享聚集

⑵组合聚集(composition aggregation)

其“整体”(重数为 0、1)拥有它的“部分”。部分仅属于同一对象，或者说整体与部分必须同时存在。如图 5.36 所示，显然组合聚集的“整体”与“部分”之间的关系更加紧密。组合聚集有三种描述形式，另外两种如图 5.37 所示。

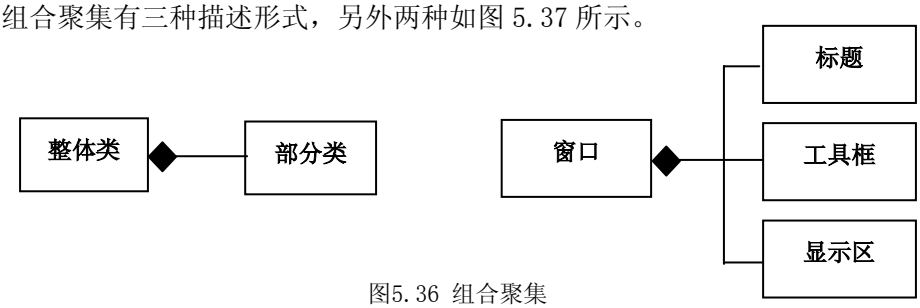


图5.36 组合聚集

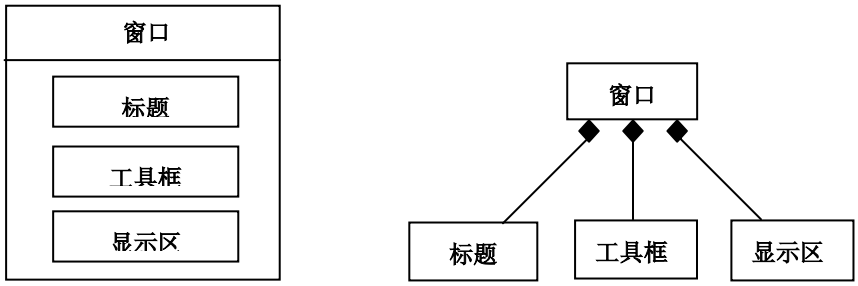


图5.37 组合聚集的描述形式

3．泛化关系

在 UML 中泛化关系指出类之间的“一般与特殊关系”，它是通用元素与具体元素之间的一种分类关系，通常即是继承关系(图 5.38)。一般类描述了多个具体类的共性，一般类又称为父类，通过特化得到子类。泛化关系用一个三角形表示，三角形的尖对着一般类，父类与子类之间可构成类的分层结构，图 5.39 是一个分层继承类图的实例。

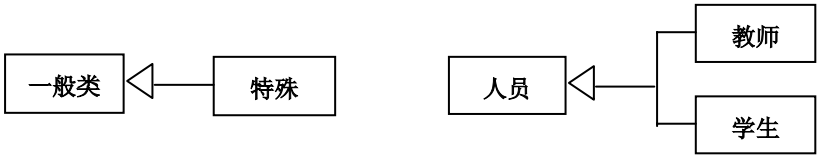


图5.38 泛化

图 5.39 中一般类为“图形”，按照维数分为 0 维、1 维、2 维。在“图形”类及 1 维、2 维类旁标注的{abstract}，表示该类为抽象类。

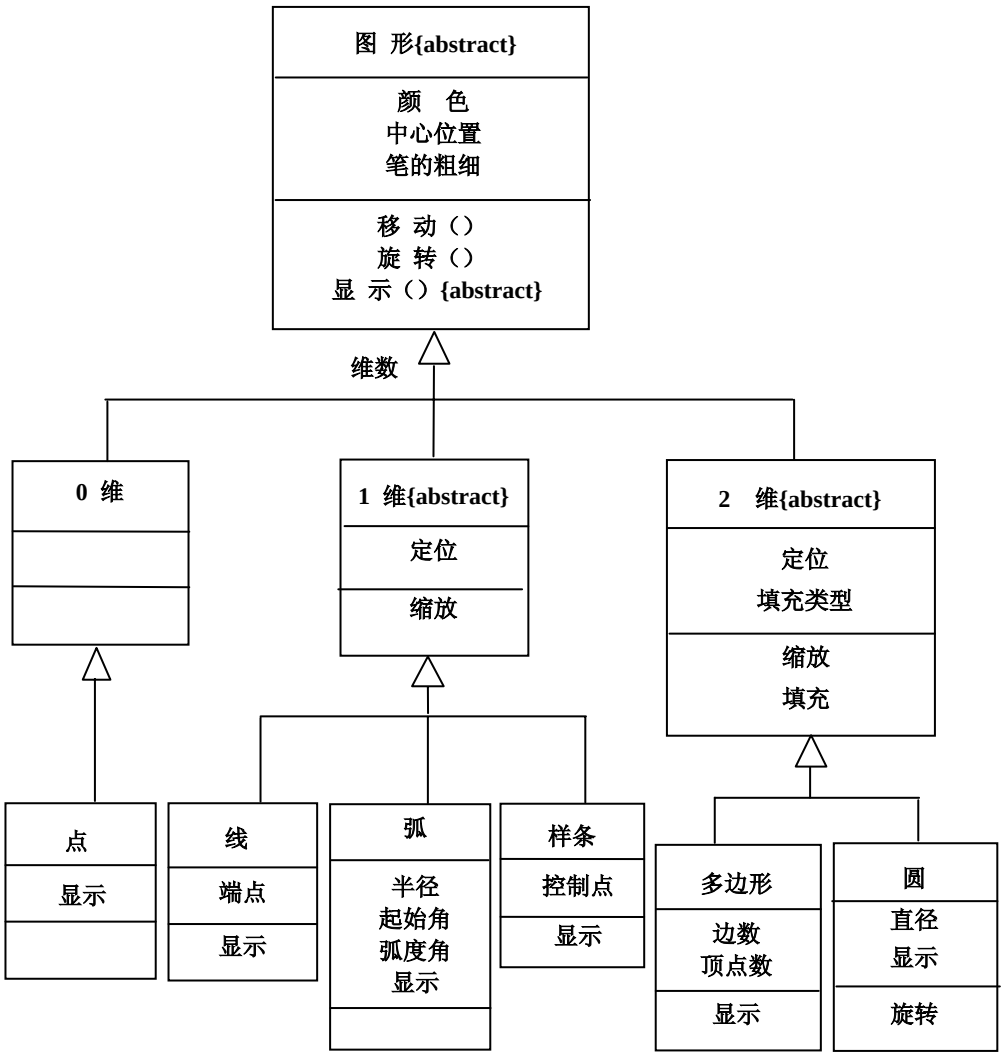


图5.39 泛化关系

抽象类 指没有实例的类，定义一些抽象的操作，即不提供实现方法的操作，只提供操作的特征，并附以{abstract}。

在泛化关系中，还有几类特殊的泛化关系：

重叠泛化 在继承树中，若存在某种具有公共父类的多重继承，称为是交叠的，并标注{overlapping}，否则是不交的{disjoint}，如图 5.40 中的“水陆两栖车”类。

完全泛化 一般类特化出它所有的子类，称为完全泛化，记为{complete}（图 5.41）。

不完全泛化 即未特化出它所有的子类，称为是不完全泛化的，表示为{incomplete}。

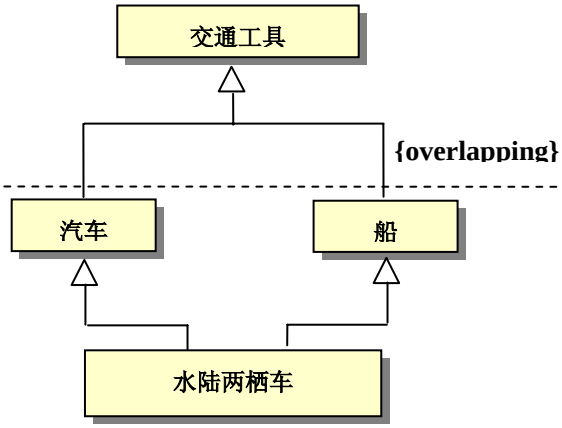


图5.40 重叠泛化

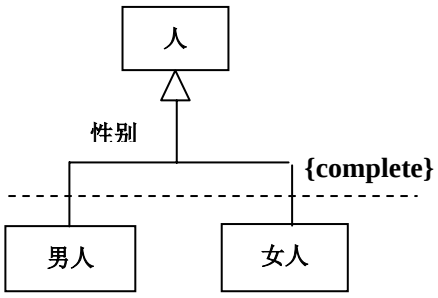


图 5.41 完全泛化

图5.42描述了一个Person对象与交通工具之间有关联“驾驶”，当Person对象使用交通工具的drive操作时，具体结果取决于所操作的对象，如果是汽车对象，则drive对应启动轮子转动，若是轮船对象，则drive对应启动螺旋桨。这种在子类中重新定义父类的某些操作的技术，称为多态性。

此外，在泛化关系中还可采用识别名称（discriminator）来指明泛化中一般化到具体化的主要依据。因此，交通工具与汽车和轮船的泛化关系中，识别名称为驱动方式（propulsion）。

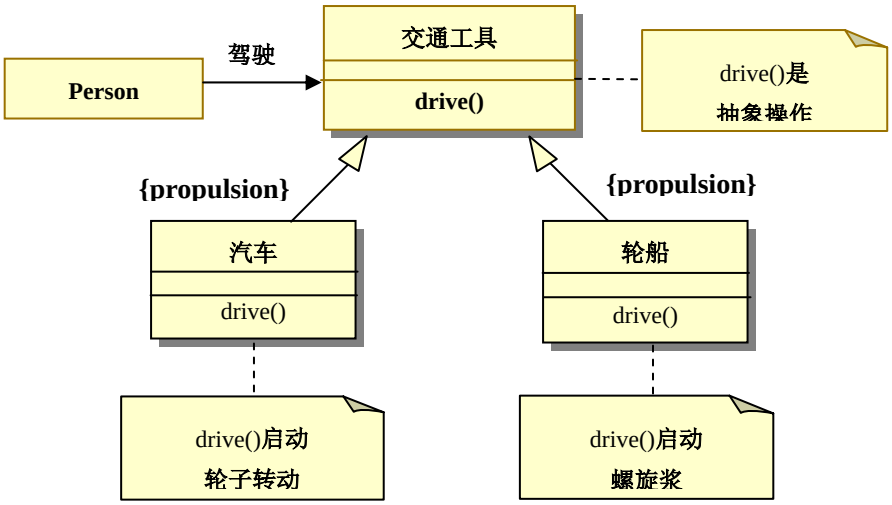


图 5.42 泛化中的多态性带识别名称的泛化

图 5.43 是一个关于订单的类图，其中，订单类（Order）与订单行类（OrderLine）之间有 1 对多的关联，角色 LineItem（行项目）表示订单行是订单的一个行项目。客户类

（Custome）与团体客户类（Corporate Customer）和个人客户类（Personal Customer）之间是继承关系。图中还描述了职员类（Employee）、产品类（Product）与其它类之间的关联。

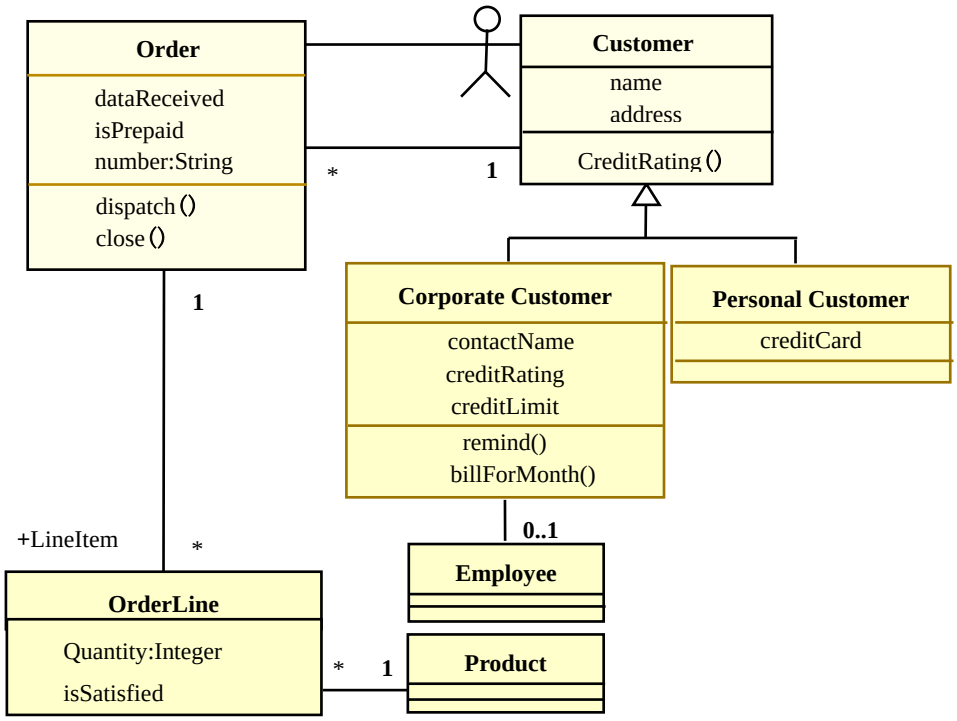


图 5.43 类图

4．类图的抽象层次和细化(Refinement)关系

需要注意的是,虽然在软件开发的不同阶段都使用类图,但这些类图表示了不同层次的抽象。在需求分析阶段,类图是研究领域的概念;在设计阶段,类图描述类与类之间的接口;而在实现阶段,类图描述软件系统中类的实现。

按照 Steve Cook 和 John Dianiels 的观点,类图分为三个层次:概念层 (Conceptual)、说明层(Specification) 、实现层(Implementation)。

概念层(Conceptual)的类图描述应用领域中的概念。实现它们的类可以从这些概念中得出，但两者并没有直接的映射关系。事实上，一个概念模型应独立于实现它的软件和程序设计语言。

说明层(Specification)类图描述软件的接口部分,而不是软件的实现部分。面向对象开发方法非常重视区别接口与实现之间的差异,但在实际应用中却常常忽略这一差异。这主要是因为 00 语言中类的概念将接口与实现合在了一起。大多数方法由于受到语言的影响,也仿效了这一做法。现在这种情况正在发生变化。可以用一个类型(Type)描述一个接口，这个接口可能因为实现环境、运行特性或者用户的不同而具有多种实现。只有在实现层(Implementation)才真正有类的概念,并且揭示软件的实现部分。这可能是大多数人最

常用的类图,但在很多时候,说明层的类图更易于开发者之间的相互理解和交流。理解以上层次对于画类图和读懂类图都是至关重要的。但是由于各层次之间没有一个清晰的界限,所以大多数建模者在画图时没能对其加以区分。画图时,要从一个清晰的层次观念出发;而读图时,则要弄清它是根据哪种层次观念来绘制的。

需要说明的是,这个观点同样也适合于其他任何模型,只是在类图中显得更为突出。更好地描述了类图的抽象层次和细化(Refinement)关系。

5.4.4 包图

包(Package)是一种组合机制，包由关系密切的一组模型元素构成，包还可以由其它包嵌套构成，包即是将许多类集合成一个更高层次的单位,形成一个高内聚、低耦合的类的集合，UML 中把这种分组机制称为包。引入包是为了降低系统的复杂性，包图是维护和控制系统总体结构的重要建模工具。包的描述如图 5.44 所示。

构成包的模型元素称为包的内容，包通常用于对模型的组织管理，因此有时又将包称为子系统（subsystem）。包拥有自己的模型元素，包与包之间不能共用一个相同的模型元素，包的实例没有任何语义（含义），仅在模型执行期间包才有意义。

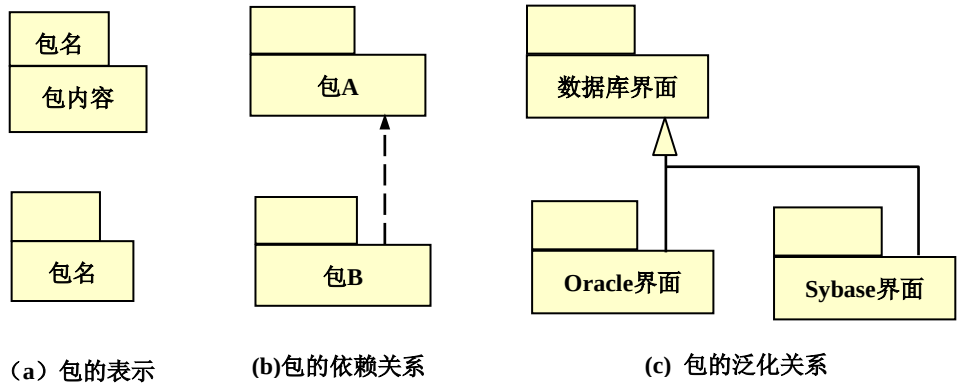


图5.44 包图的元素

包的内容：可以是类的列表,也可以是另一个包图,还可以是一个类图。包之间的关系有依赖和泛化（继承）。

依赖关系：两个包中的任意两个类存在依赖关系，则包之间存在依赖关系。包之间的依赖关系，最常用的是输入依赖关系《Import 》、《Access 》，两者之间区别是后者不把目标包内容加到源包的名字空间。图 5.45 表示了包及包之间的依赖关系。

泛化关系：使用继承中通用和特例的概念来说明通用包和专用包之间的关系。例如专用包必须符合通用包的界面,与类继承关系类似。

和类一样包也有可见性，利用可见性控制外部包对包的内容的存取方式，UML 中定义了四种可见性：私有，公有，保护和实现。包的缺省值为公有的。

包也可以有接口，接口与包之间用实线相连，接口通常由包的一个或多个类实现，如图 5.46 所示。

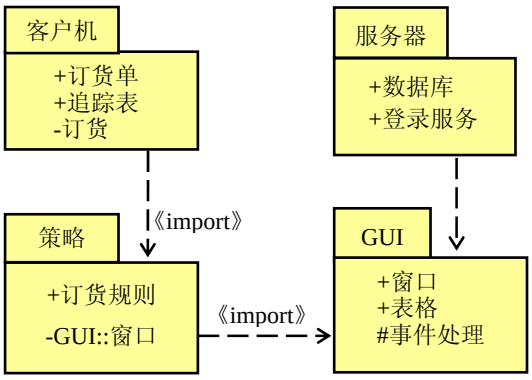


图 5.45 包之间的依赖关系

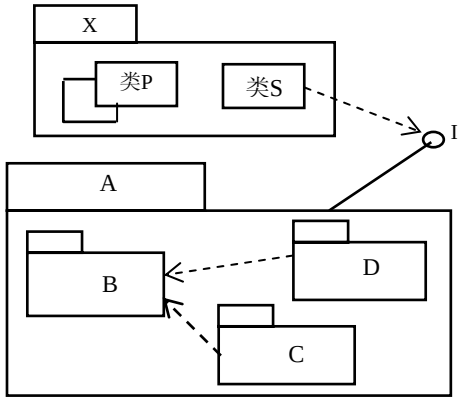


图5.46 包之间的接口

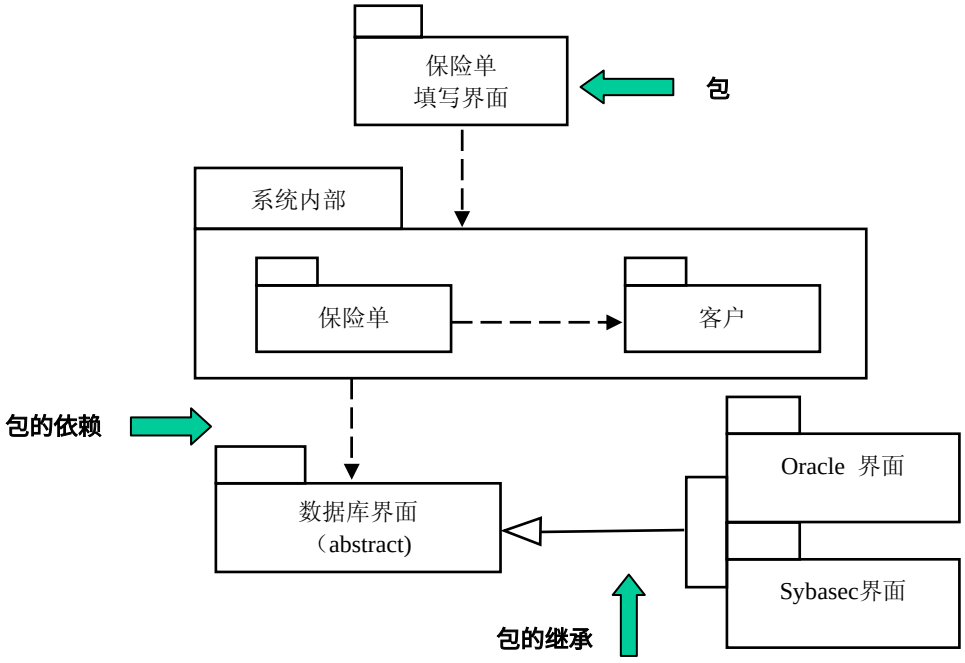


图5.47 保险信息系统的包图

5.5 建立动态模型

动态模型主要是描述系统的动态行为和控制结构。动态行为包括系统中对象生存期内可能的状态以及事件发生时状态的转移，还包括对象之间动态合作关系，显示对象之间的交互过程以及交互顺序，同时描述了为满足用例要求所进行的活动以及活动间的约束关系。

动态模型包括 4 类图：状态图、活动图、顺序图、合作图。

状态图(state diagram)：状态图用来描述对象、子系统、系统的生命周期。

活动图(activity diagram)：着重描述操作实现中完成的工作以及用例实例或对象中的活动，活动图是状态图的一个变种。

顺序图(sequence diagram)：是一种交互图，主要描述对象之间的动态合作关系以及合作过程中的行为次序，常用来描述一个用例的行为。

合作图(collaboration diagram)：用于描述相互合作的对象间的交互关系，它描述的交互关系是对象间的消息连接关系。

5.5.1 消息

在动态模型中,对象间的交互是通过对象间消息的传递来完成的。对象通过相互间的通信(消息传递)进行合作，并在其生命周期中根据通信的结果不断改变自身的状态。在 UML 中,消息的图形表示是用带有箭头的线段将消息的发送者和接收者联系起来（图 5.48）,箭头的类型表示消息的类型。

1．简单消息(simple)

表示简单的控制流，描述控制如何从一个对象传递到另一个对象，但不描述通信的细节。

2．同步消息(synchronous)

是一种嵌套的控制流，用操作调用实现，操作的调用是一种典型的同步消息。操作的执行者要到消息相应操作执行完并回送一个简单消息后，再继续执行。

3．异步消息(asynchronous)

表示异步控制流，消息的发送者在消息发送后，不用等待消息的消息的处理和返回即可继续执行。异步消息主要用于描述实时系统中的并发行为。

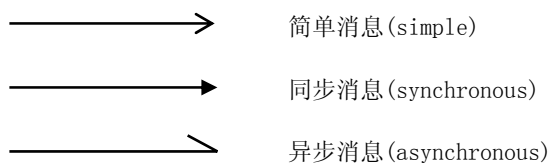


图 5.48 消息的类型

5.5.2 状态图

状态图(State Diagram)用来描述一个特定对象的所有可能的状态及其引起状态转移的事件。一个状态图包括一系列的状态以及状态之间的转移。

1．状态

所有对象都具有状态,状态是对象执行了一系列活动的结果。当某个事件发生后,对象的状态将发生变化。状态图中定义的状态有:

初态—状态图的起始点,一个状态图只能有一个初态。

终态—是状态图的终点。而终态则可以有多个。

中间状态—可包括三个区域:名字域、状态变量与活动域。

复合状态—可以进一步细化的状态称作复合状态。

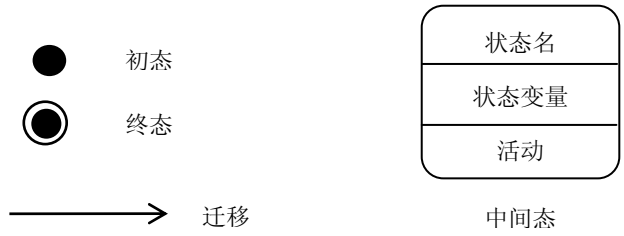


图 5.49 对象的状态

在中间状态中，状态变量表示状态图所显示的类的属性。活动则列出了在该状态时要执行的事件和动作，即响应事件的内部动作或活动的列表，定义为：

事件名 (参数表[条件])/动作表达式

通常有 3 个标准事件，而且都无参数：

entry 事件　用于指明进入该状态时的特定动作。

Exit 事件　用于指明退出该状态时的特定动作。

do 事件　　用于指明在该状态中时执行的动作。

图 5.50 描述 login 状态

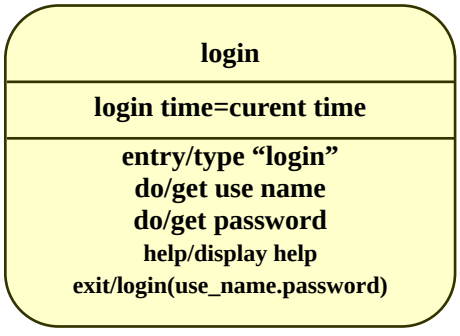


图5.50 login 状态

2．状态迁移

一个对象的状态变迁称为状态迁移。通常是由事件触发的，事件是激发状态迁移的条件或操作。在 UML 中，有 4 类事件：

⑴某条件变为真；表示状态迁移的上的警戒条件。

⑵收到来自外部对象的信号（signal）表示为状态迁移上的事件特征，也称为消息。

⑶收到来自外部对象的某个操作中的一个调用，表示为状态迁移上的事件特征，也称为消息。

⑷状态迁移上的时间表达式。在状态图中，一般应标出触发转移的事件表达式。如果转移上未标明事件,则表示在源状态的内部活动执行完毕后自动触发转移。

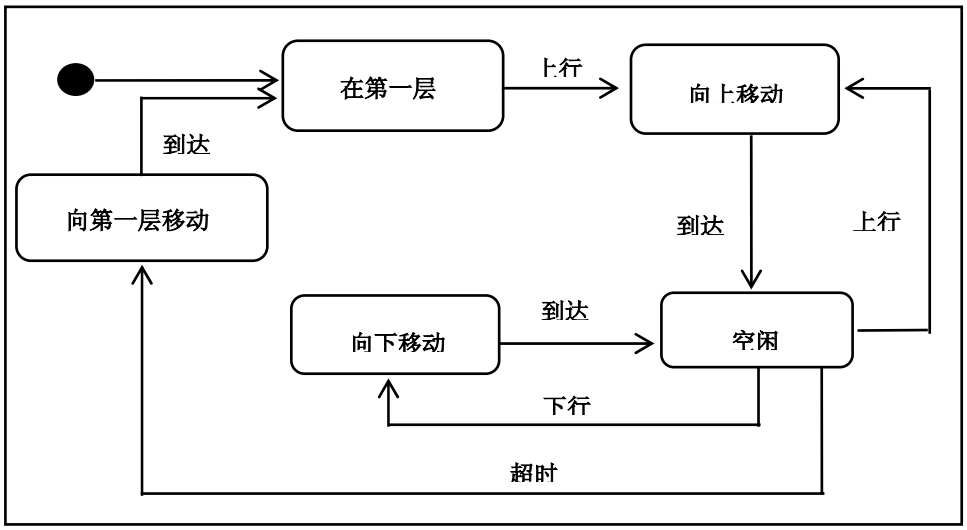


图 5.51 电梯状态图

图 5.51 描述了电梯升降的状态图。电梯升降有五个状态，三个循环圈；“空闲”状态与“向下移动”状态之间，“空闲”状态与“向上移动”状态之间，以及大循环。循环圈越多，表明对象的控制逻辑越复杂。“超时”是指电梯的空闲状态超过某个规定的时间值。

3．嵌套的状态图

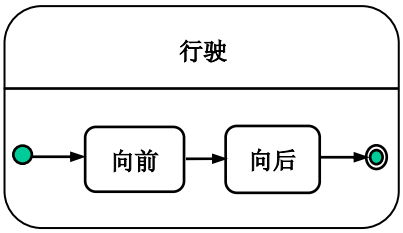


图 5.52 或关系的子状态

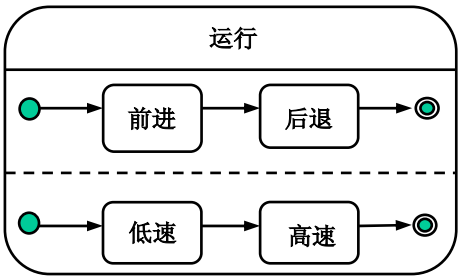


图5.53 与子状态及或子状

状态图可能有嵌套的子状态图，且子状态图可以是另一个状态图。子状态又可分为两

种：“与”子状态和“或”子状态。“行驶”状态有两个或关系的子状态：“向前”或者“向后”。而图 5.53 中，“前进”与“后退”，“低速”与“高速”分别是两对或关系的子状态，而虚线上、下又分别构成与关系的子状态。

4．**细化的状态表示**

UML给出了电梯升降的细化状态表示(图 5．54)。对系统中对象状态的状态变量和活动作了进一步的描述，状态变量time,初值为零，活动为do/increase timer。

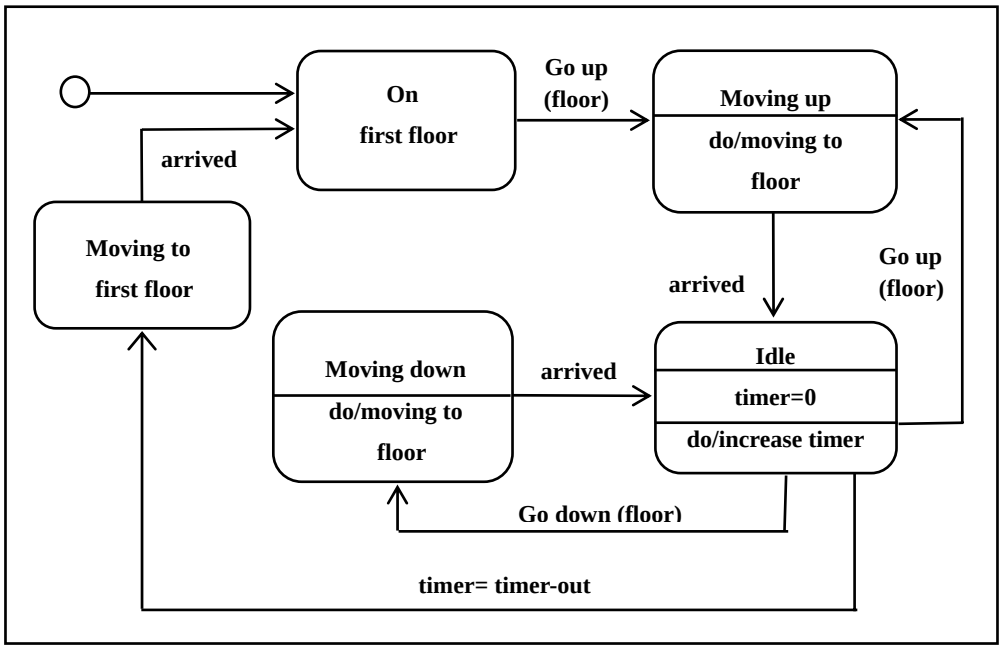


图5.54 细化电梯状态图

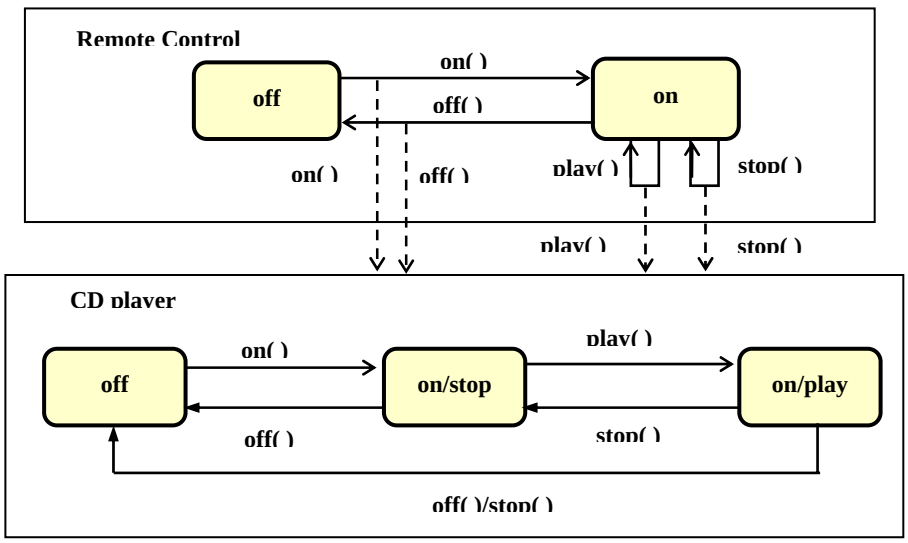


图 5.55 消息发送状态图

状态图之间的消息发送

状态图之间可以发送消息，用虚箭头表示，这时状态图必须画在矩形框中，如图 5.55 所示。

5.5.3 顺序图

顺序图(Sequence Diagram)用来描述对象之间动态的交互关系,着重体现对象间消息传递的时间顺序。

1. 概述

顺序图存在两个轴:水平轴表示不同的对象,垂直轴表示时间。顺序图中的对象用矩形框表示,并标有对象名和类名。垂直虚线是对象的生命线,用于表示在某段时间内对象是存在的。

对象间的通信通过在对象的生命线之间的消息来表示，消息的箭头类型指明消息的类型，分为简单消息(simple)、同步消息(synchronous)和异步消息(asynchronous)。

说明信息用于说明消息发送的时间，动作执行的情况，定义两个消息之间的时间限制，定义一些约束信息等。消息可以是信号，操作调用或其它，消息可以有序号，还可有条件。

简单消息(simple)：表示消息类型不确定或与类型无关,或是一同步消息的返回消息。

同步消息(synchronous)：表示发送对象必须等待接收对象完成消息处理后，才能继续执行。

异步消息(asynchronous)：表示发送对象在消息发送后，不必等待消息处理后，可立即继续执行。

消息延迟：用倾斜箭头表示。

消息串：包括消息和控制信号，控制信息位于信息串的前部。

控制信息 { 条件控制信息　如：[x ＞ 0]
　　重复控制信息　如：＊ [I ＝1..n]

当收到消息时,接收对象立即开始执行活动,即对象被激活了，通过在对象生命线上显示一个细长矩形框来表示激活。

3. 顺序图的形式

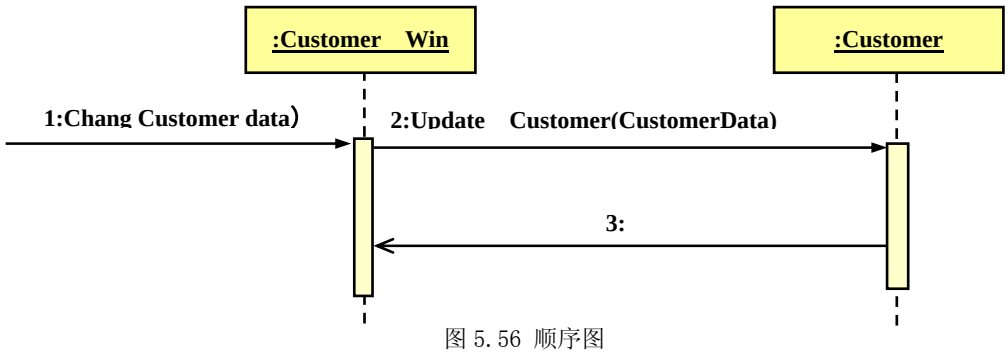


图 5.56 顺序图

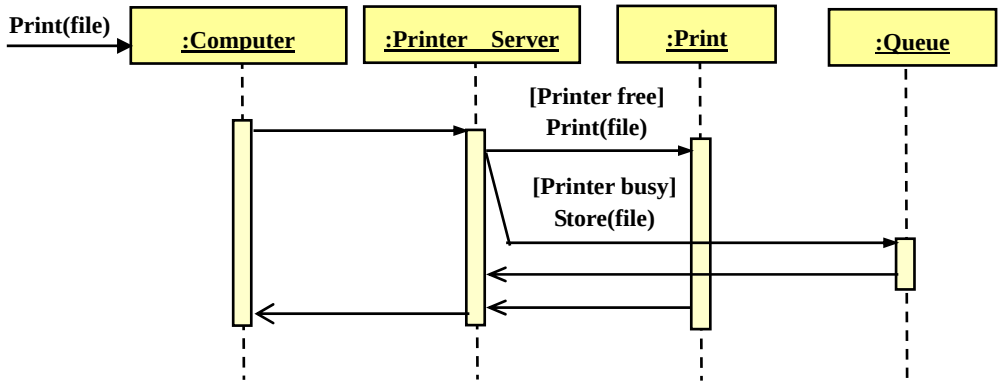


图 5.57 带分支的顺序图

有两种使用顺序图的方式：一般格式和实例格式。实例格式详细描述一次可能的交互，没有任何条件和分支或循环，它仅仅显示选定情节（场景）的交互（图 5.56）。而一般格式则描述所有的情节。因此，可能包括了分支，条件和循环。

图 5.57 和图 5.58 分别描述了带分支的顺序图及有循环标记的顺序图。5.59 是一个打电话的顺序图。

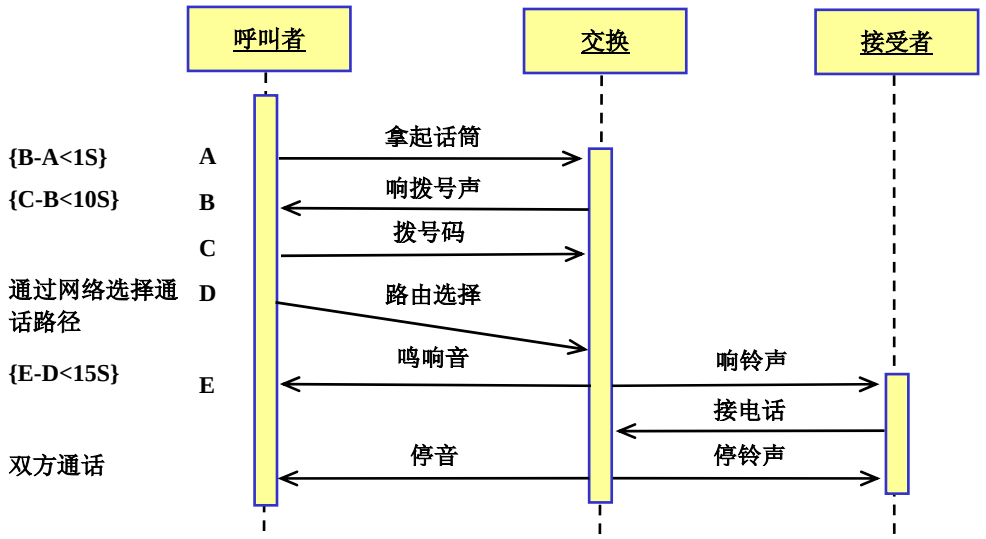


图5.59 打电话的顺序图

在打电话的顺序图中，有呼叫者、交换、接受者三个对象，对象之间传送消息。左边的 A、B、C、D、E 表示消息发送和接收的时刻，花括号中的信息表示时间限制。这些都是说明信息。

4. 创建对象与对象的消亡

在顺序图中，还可以描述一个对象通过发送一条消息来创建另一个对象。当对象消亡(destroying)时，用符号×表示。

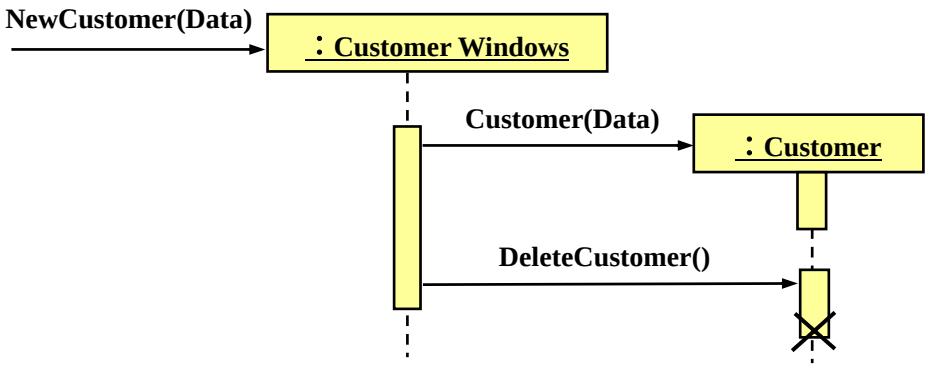


图 5.60 创建或删除对象

5.5.4 合作图

合作图(Collaboration Diagram),也称为协作图，用于描述相互合作的对象间的交互关系和链接(Link)关系。虽然顺序图和合作图都用来描述对象间的交互关系,但侧重点不一样。顺序图着重体现交互的时间顺序,合作图则着重体现交互对象间的静态链接关系。

图 5.61 是一个打印文件的合作图，图中有 3 个对象：“Computer”、“PrinterServer”和“Printer”。由操作者向对象“Computer”发出打印文件的消息，当打印机空闲时，对象“Computer”向“PrinterServer”对象发 1：打印消息，“PrinterServer”再向对象“Printer”发消息 1.1。

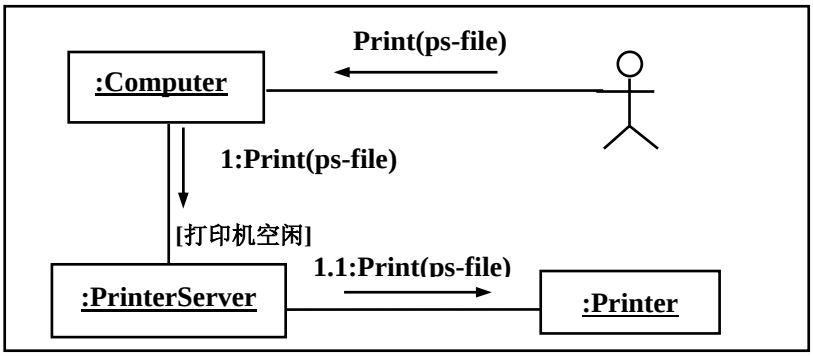


图 5.61 合作图

1．合作图中的模型元素

⑴对象合作图中对象的外观与顺序图中的一样。如果一个对象在消息的交互中被创建,则可在对象名称之后标以{new}。类似地,如果一个对象在交互期间被删除,则可在对象名称之后标以{destroy}。

⑵链接(Link)

链接用于表示对象间的各种关系，包括组成关系的链接(Composition Link)、聚集关系的链接(Aggregation Link)、限定关系的链接(Qualified Link)以及导航链接(Navigation Link)等，如图 5.62 所示。各种链接关系与类图中的定义相同。对于链接还可以加上“角色”与“约束”，在链角色上附加的约束有 global(全局)，local(局部)，parameter(参数)，self(自身)。

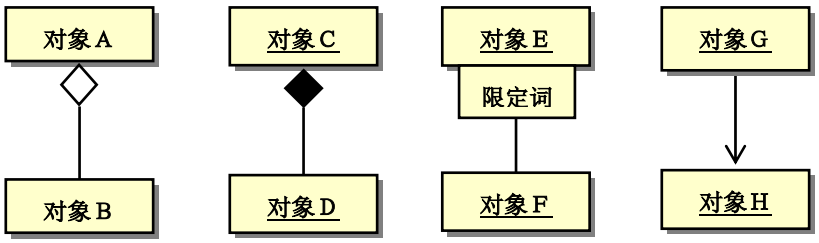


图5.62 各种关系的链接

⑶消息

在对象之间的静态链接关系上可标注消息，消息类型有简单消息，同步消息和异步消息三种。用标号表示消息执行的顺序。消息定义的格式如下：

消息类型　标号 控制信息：返回值：=消息名 参数表其中，

标号有 3 种：

顺序执行：按整数大小执行。1，2…嵌套执行：标号中带小数点。1.1，1.2，1.3，…

并行执行：标号中带小写字母。1.1.1a，1.1.1b，…

在控制器控制下进行布线，找出左端点 r0 和右端点 r1，创建对象“直线”，并在窗口显示出来。

控制信息 { 条件控制信息　如：[x ＞ y]
　　重复控制信息　如：＊[I ＝1..n]

合作图应用举例

图 5.63 在控制器控制下进行布线，每次布线，先要定位两个端点，即找出左端点 r0 和右端点 r1，再以 r0 和 r1 为参数，创建“直线”对象，并将其在窗口显示出来。嵌套的消息标号，表示其消息发送次序。如图 5.63 中。学习消息发送的次序是标号 1，再是 1.1，而 1.1.1a 和 1.1.1b 并行执行，接着再是 1.1.2，1.1.3，最后是 1.1.3.1。

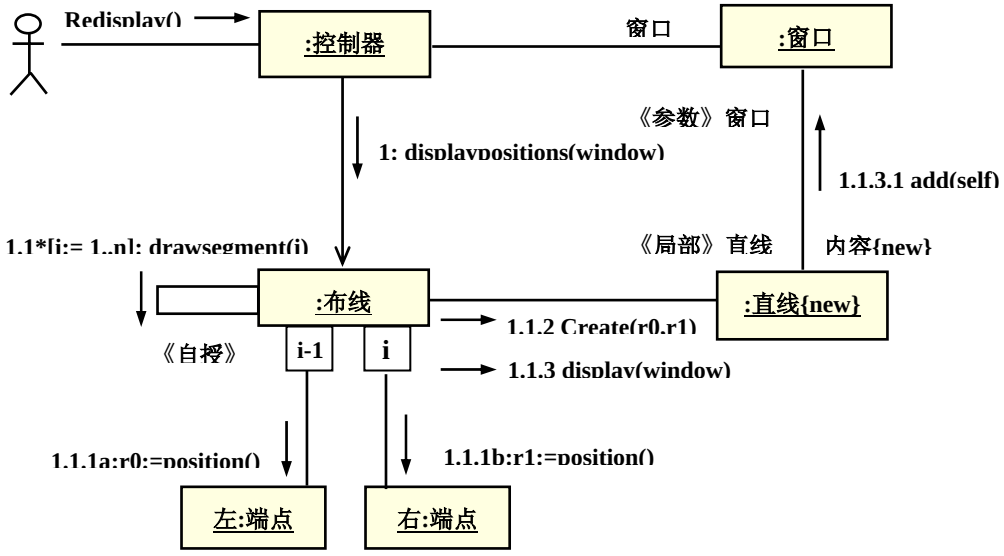


图 5.63 电路设计的合作图

图 5.64 是一个统计销售结果的合作图，请读者自己分析对象之间的关系及消息的发送过程。

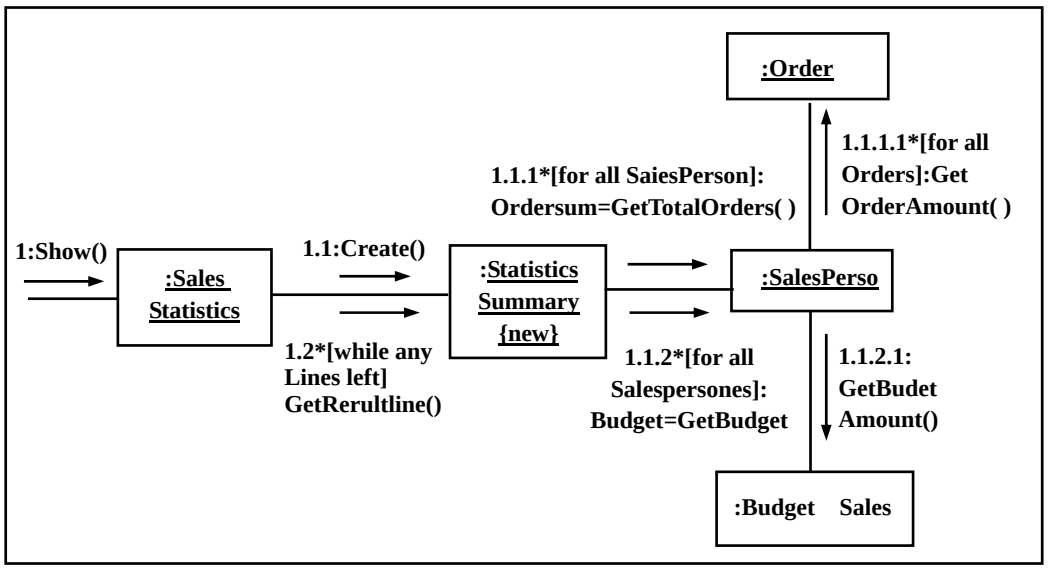


图5.64 统计销售结果的合作图

5.5.5 活动图

活动图(Activity Diagram)是由状态图变化而来的,它们各自用于不同的目的。状态图着重描述了对象的状态变化以及触发状态变化的事件，交互模型（顺序图和合作图）则描述对象之间的动态交互行为。但是，从系统任务的观点看系统时，我们发现它是由一系列的有序活动组成的，用例图虽然也是从活动的角度描述系统任务，但是却无法描述系统任务中的并发活动，为此引入活动图。

活动图描述了系统中各种活动的执行的顺序，刻化一个方法中所要进行的各项活动的执行流程。活动图的应用非常广泛,它既可用来描述操作(类的方法)的行为,也可以描述用例和对象内部的工作过程,并可用于表示并行过程。活动图显示动作及其结果，着重描述操作实现中完成的工作以及用例或对象内部的活动。在状态图中状态的变迁通常需要事件的触发，而活动图中一个活动结束后将立即进入下一个活动。

1．活动图的构成

构成活动图的模型元素有：活动、转移、对象、信号、泳道等。其中活动是活动图的核心概念。

⑴活动

活动是构成活动图的核心元素，是具有内部动作的状态，由隐含的事件触发活动的转移。活动的解释依赖于作图的目的和抽象层次，在概念层描述中，活动表示要完成的一些任务；在说明层和实现层中，活动表示类中的方法。

活动用圆角框表示，标注活动名。活动图还有其它的图符(图 5.59)：初态、终态、判断、同步。

在活动图中使用一个菱形表示判断（decision）,表达条件关系,是一种特殊的活动。判断标志可以有多个输入和输出转移,但在活动的运作中仅触发其中的一个输出转移。

同步也是一种特殊的活动，同步线描述了活动之间的同步关系。

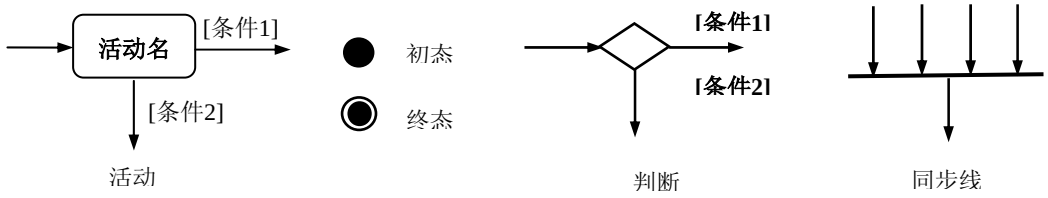


图 5.65　活动图的图符

⑵转移

转移描述活动之间的关系，描述由于隐含事件引起的活动变迁，即转移可以连接各活动及特殊活动（初态、终态、判断、同步线）。

转移用带箭头的直线表示，可标注执行该转移的条件，无标注表示顺序执行。

⑶泳道

活动图描述了要执行的活动和顺序，但并没有描述这些活动是由谁来完成的，泳道（swimlane）进一步描述完成活动的对象，并聚合一组活动，因此泳道也是一种分组机制。

将一张活动图划分为若干个纵向矩形区域，每个矩形区域称为一个泳道，包括了若干活动，在泳道顶部帮助标注的是完成这些活动的对象，图 5.66 描述了一个顾客购物的过程。

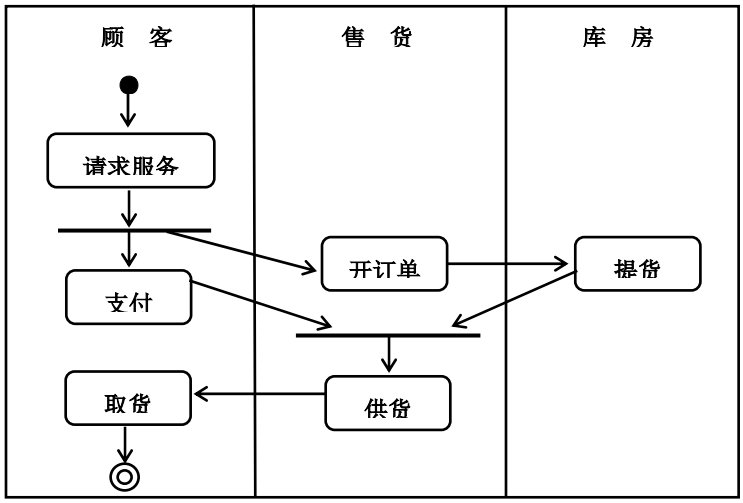


图 5.66 泳道

⑷对象流

活动图中可以出现对象，对象作为活动的输入 / 输出，用虚箭头表示。如图 5.67 中测量活动所产生的结果输出给对象“测量值”，再由该对象将值传送给显示活动。

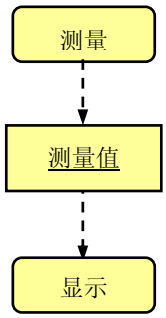


图 5.67 对象流

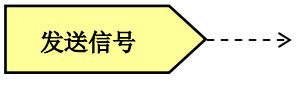


图 5.68 控制图符

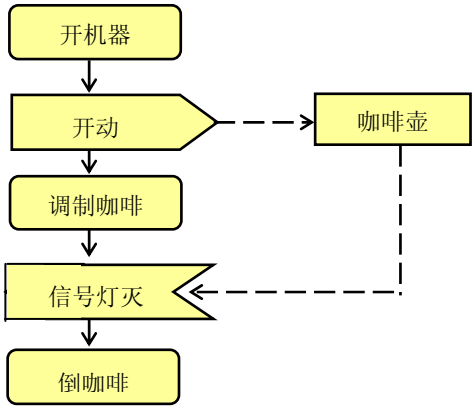


图5.69 控制图符举例

⑸控制图符

活动图中可发送和接收信号,分别用发送和接收图符表示(图 5.68)，发送符号对应于与转移联系在一起的发送短句。接收符号也同转移联系在一起。图 5.69 描述了一个调制咖啡的过程，将“开动”信号发送到对象“咖啡壶”，当调制咖啡完成后，将接收来自对象“咖啡壶”的“信号灯灭”信号。

2．活动图举例

活动图中只有一个起点一个终点，表示方式和状态图一样，泳道被用来组合活动，通常根据活动的功能来组合。图 5.64 中 PS 文件即 Postscript file。

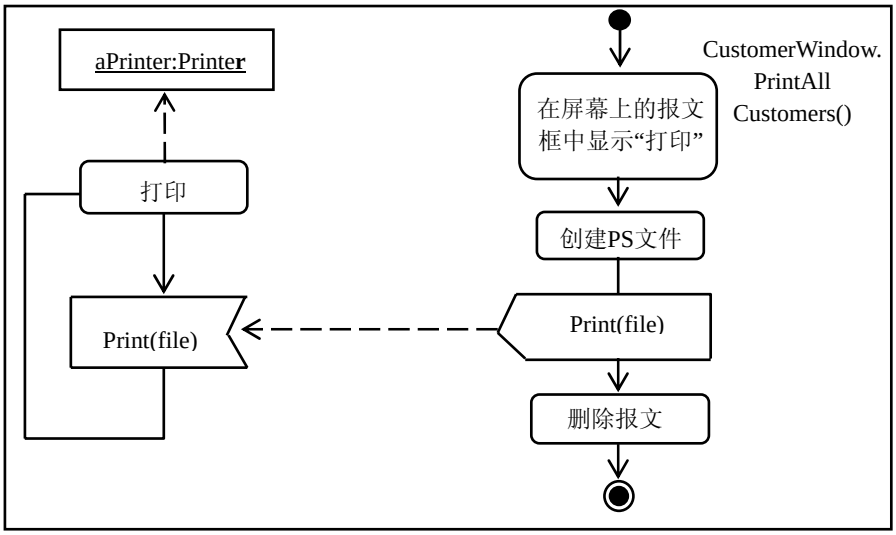


图 5.70 处理报文的活动图

5.6 实现模型

实现模型描述了系统实现时的一些特性，又称为物理体系结构模型。包括源代码的静态结构和运行时刻的实现结构。实现模型包括构件图和配置图

5.6.1 构件图

构件图(Component diagram)又称为组件图，显示代码本身的逻辑结构，它描述系统中存在的软构件以及它们之间的依赖关系。构件图的元素有构件，依赖关系和界面。

构件（Component）是系统的物理可替换的单位，代表系统的一个物理组件及其联系，表达的是系统代码本身的结构。构件的描述如图 5.71 所示，构件图符是一个矩形框。
构件可以看作包与类对应的物理代码模块，逻辑上与包，类对应，实际上是一个文件，构件的名称和类的名称的命名法则很是相似，分为简单构件与扩充构件。可以有下列几种类型的构件：

1．源代码构件（Source Component）

源代码构件是实现一个或者多个类的源代码文件。可在构件上标注以下符号：

《file》:表示包含源代码的文件。

《page》:表示 WEB 页。

《document》:表示文档，而不是可编译的代码。

图 5.71 简单构件

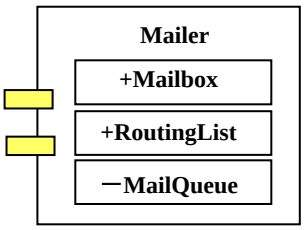


图 5.72 扩充构件

2．二进制构件(Component)

是一个目标代码文件，或者是编译一个或者多个源代码构件生成的静态库文件或动态库文件。

3．可执行构件(Executable Component)

即是在ＣＰＵ上运行的一个可执行文件。

构件对外提供的可见操作和属性称为构件的界面。界面的图符是一个小圆圈。用一条连线将构件与圆圈连起来。构件之间的依赖关系是指结构之间在编译，连接或执行时的依赖关系。用虚线箭头表示。

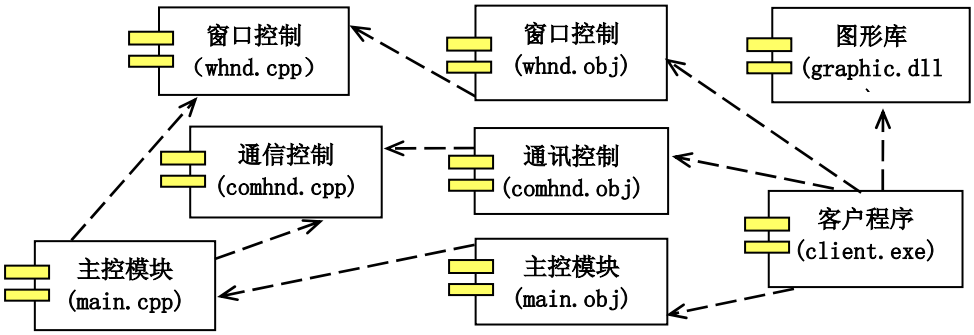


图5.73 构件图实例

图 5.73 和图 5.74 是构件图实例。构件的依赖关系又分为：开发期的依赖和调用依赖。开发期的依赖(Development －time Dependency)是指在编译阶段和连接阶段，构件之间的依赖关系。调用依赖(Call Dependency)是指一个构件调用或使用另外一个构件服务。

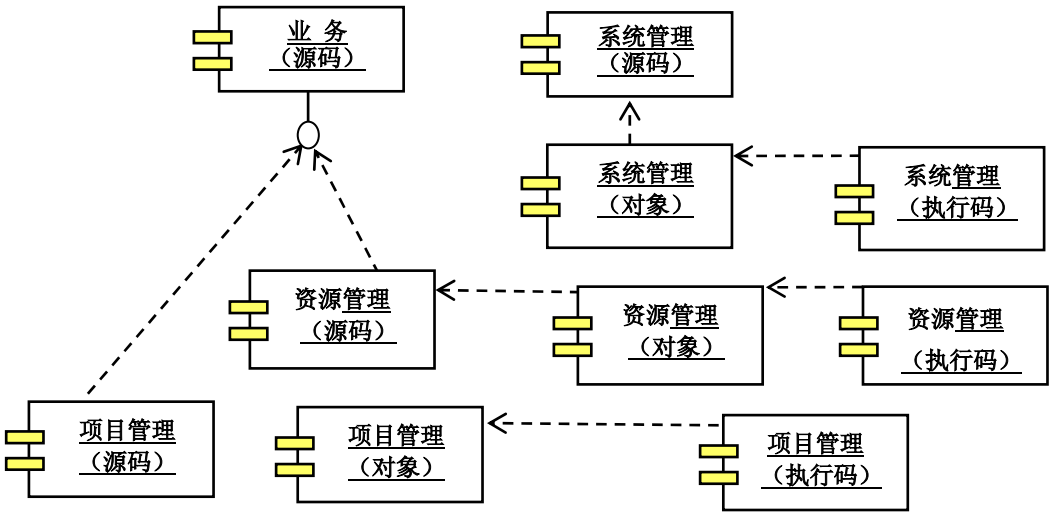


图 5.74 构件图实例

5.6.2 配置图

配置图(Deployment diagram) 描述了系统中硬件和软件的物理配置情况和系统体系结构，显示系统运行时刻的结构。配置图的元素有结点和连接，配置图中的简单结点是指实际的物理设备以及在该结点上运行构件或对象。配置图还描述结点之间的连接以及通信类型。

1．结点与连接

配置图中的结点代表计算机资源，通常是某种硬件，如服务器、客户机或其它硬件设备，结点包括在其上运行的软构件及对象，结点的图符是一个立方体。结点应标注名字

配置图各结点之间进行交互的通信路径称为连接，连接表示系统中的结点之间的联系，用结点之间的的连线表示连接，在连接的连线上要标注通信类型。图 5.75 描述了一个保险系统的配置图，配置图中“客户 PC”结点和“保险服务器”结点是由通信路径按照 TCP/IP 协议连接的。

2．构件与接口

软构件代表可执行的物理代码模块，如一个可执行程序。图 5.75 中，结点“保险服务器”包含了“保险系统”、“保险系统配置”和“保险数据库”三个构件。

在面向对象的方法中，并不是类和对象等元素的所有属性及操作对外都可见，它们对外提供的可见操作和属性称为接口，用一个联结小圆圈的线段表示。在保险系统的配置图中的“保险系统”构件，提供了一个称为“配置”的接口，图中还显示了构件之间的依赖关系；即“保险系统配置”构件通过接口依赖于“保险系统” 构件。

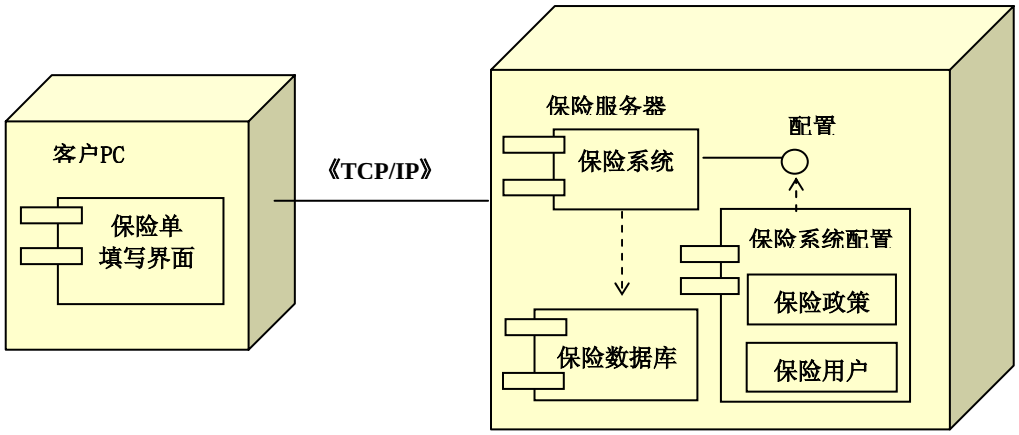


图5.75保险系统的配置图

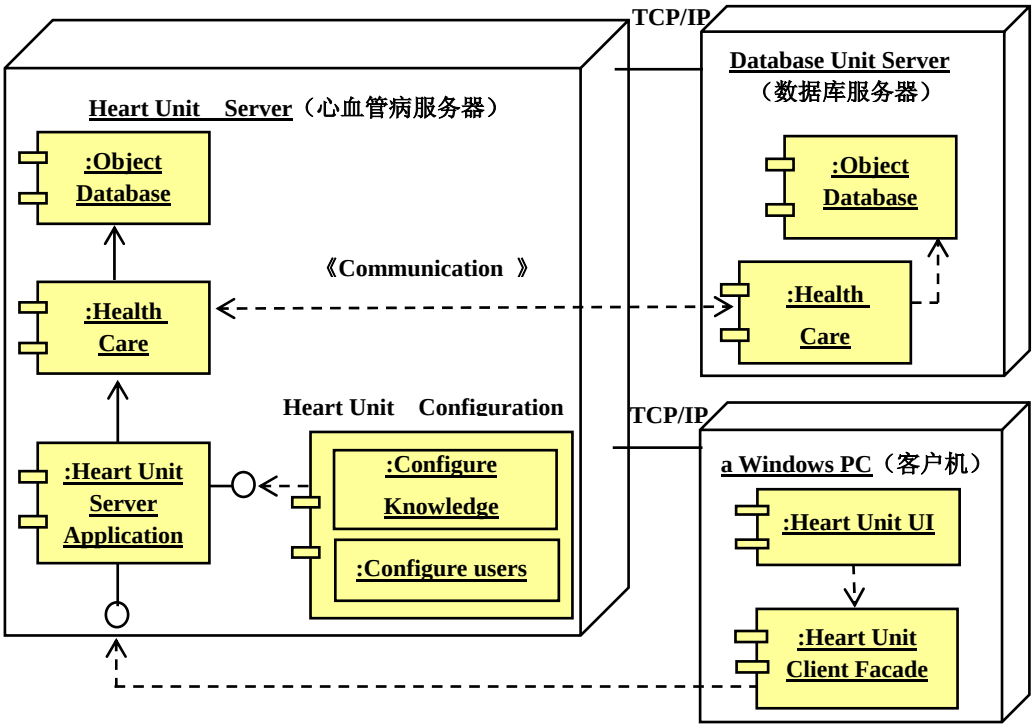


图5.76 医院诊疗系统的配置图

图 5.76 是描述医院诊疗系统的配置图，该图是一个对象配置图，结点“数据库服务器”、“客户机”和“心血管病服务器”都是可视的某个结点的实例，“心血管病服务器”通过 TCP/IP 与结点“数据库服务器”和“客户机”连接。结点“数据库服务器”包括两

个构件：“Object Database”（对象数据库）和“Health Care”（心血管病领域）。在结点“客户机”中的构件“Heart Uni Client Façade”（心血管病客户）通过接口依赖于“心血管病服务器”结点的中的构件“Heart Unit Server Application”（心血管病应用程序）。即心血管病客户可以通过网络获得心血管病服务器的服务，在网上看病。

并不是所有的系统都需要建立配置图，一个单机系统只需建立包图或构件图就行了。配置图主要用于在网络环境下运行的分布式系统或嵌入式系统的建模。

配置图可以显示计算机结点的拓扑结构和通信路径，结点上执行的软构件，软构件包含的逻辑单元等，特别对于分布式系统，配置图可以清楚的描述系统中硬件设备的配置，通信以及在各硬件设备上各种软构件和对象的配置。因此，配置图是描述任何基于计算机的应用系统的物理配置或逻辑配置的有力工具。