第5章 定义结构与连接

前面讨论了如何找出对象类及其属性与服务，本章将把眼光从各个单独的对象转移到对象以外，分析和认识各个对象类之间的关系，以建立 OOA 基本模型的关系层。只有定义和描述了对象类之间的关系，它们才能构成一个整体的、有机的系统模型。

对结构与连接的分析还将启发分析员进一步完善对象层和特征层，包括：发现一些原先未曾认识的类；重新考虑某些对象的分类；对某些类进行调整；以及，对某些类的属性和服务进行增删或调整其位置。所以在前两章讨论发现对象及其属性与服务时都曾谈到结构与连接的影响，对象类与外部的关系有以下几种。

(1) 继承关系(即对象类之间的一般—特殊关系)，用一般—特殊结构表示。

(2) 整体—部分关系(即对象之间的组成关系)，用整体—部分结构表示。

(3) 对象之间的静态联系(即通对象属性反映的联系)，用实例连接表示。

(4) 对象之间的动态联系(即对象行为之间的依赖关系)，用消息连接表示。

表示上述关系的两种结构和两种连接将构成 OOA 基本模型(类图)的关系层。如何定义这些结构与连接是本章讨论的主要内容。下面将分别讨论一般—特殊结构、整体—部分结构、实例连接和消息连接 4 种关系的概念、表示法及分析活动。

5.1 整体—部分结构

5.1.1 整体—部分结构及其用途

整体—部分关系反映了对象之间的构成关系，也称为聚集关系，用于描述系统中各类对象之间的组成关系，通过它可以看出某个类的对象以另外一些类的对象作为其组成部分。

如果对象 a 是对象 b 的一个组成部分，则 b 为 a 的整体对象，a 为 b 的部分对象。并把 b 和 a 之间的关系称作整体—部分关系(又可称为“has-a”关系)。按照这个定义，整体—部分关系是对象实例之间的一种关系，其定义域是由系统中对象实例集合构成的笛卡儿积。但是 OOA 主要是针对类来讨论问题的，而不是把类的每个具体的对象实例之间的组成情况都一一地表示出来。这样看，给出一个类与类之间的整体—部分关系的定义，似乎更为有用。如果对象 a 是对象 b 的一个组成部分，那么 b 的类定义引用 a 的类定义，即 A 的类定义是 B 的类定义的一个(直接或间接的)组成部分。

做了上述解释之后，可以谈论类与类之间的整体—部分关系，它指一个类定义引用另一个类定义。也可理解为一个类的对象实例，以另一个类的对象实例作为其组成部分。给出整体—部分结构的定义如下：整体—部分结构是把一组具有整体—部分关系的类组织在一起的结构。它是一个以类为结点，以整体—部分关系为边的连通有向图。

整体—部分结构体现了 OO 方法的聚合原则，它和体现分类及继承原则的一般—特殊结构同样重要，是 OOA 表示复杂事物的另一个重要手段。

首先，整体—部分结构可以清晰地表达事物之间的组成情况。客观世界中，事物之间的组成关系（即整体—部分关系）是大量的，表现为多种形式的，例如，物理上的整体事物和它的一个部分；组织机构与它的下级组织或部门；空间上的包容关系，如教室和桌椅；抽象团体与成员，如班级与学生；事物的整体部分；具体事物和它的某个抽象方面等。

在客观世界中，事物之间的整体—部分关系处处可见。平常认识这些事物时并不是孤立地看待它们。比如认识一张桌子，并不是将其看作“一个桌面和四个桌脚”，也不是把整体和部分放在同一级别来看待，说“这是一张桌子、一个桌面和四个桌脚”。按日常的思维方式，一般是这样来看：“这是一张桌子，它是由一个桌面和四个桌脚构成的”。整体—部分结构正是按这样的认识方法，确切地反映了事物之间的组成情况，包括一些较复杂的情况，因此，整体—部分结构可使 OOA 模型清晰地表达问题域中事物的复杂构成关系。对软件开发者而言，整体—部分结构是一种用途更为广泛的系统构造技巧。运用它可以在许多情况下简化系统的描述，并解决一些其他表达方式不容易解决的问题。

在 OOA 中表示整体—部分结构的方法是在整体对象的类和部分对象的类之间画出整体—部分结构连接符号，同时在整体对象的类中增设一个表示部分对象的属性。这个属性既然设在整体对象之中，它所表示的对象当然就是整体对象的一个组成部分了。在用 OOPL 编程时，有两种实现整体—部分结构的方式，一种方式是把整体对象中的这个属性变量定义成指向部分对象的指针，或定义成部分对象的对象标识，运行时动态创建部分对象，并使整体对象中的指针或对象标识指向它，如图 5-1(a)所示。另一种方式是用部分对象的类作为数据类型，静态地声明整体对象中这个代表部分对象的属性变量。这样，部分对象就被嵌入到整体对象的属性空间中，形成嵌套对，如图 5-1(b)所示。

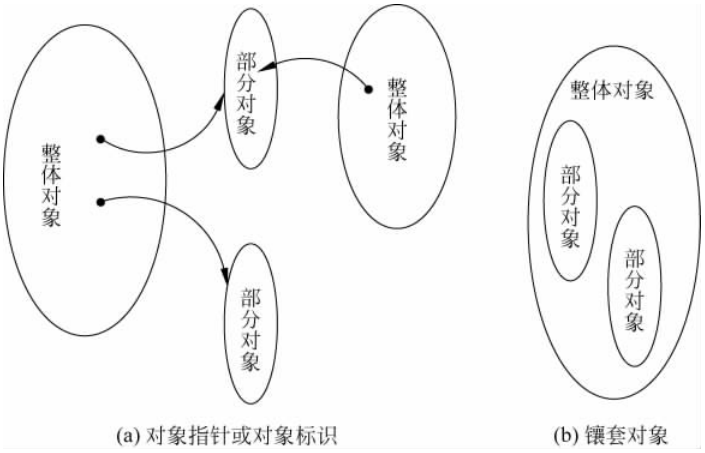

图 5-1　整体—部分结构

这两种实现方式都能表达对象之间的整体—部分关系，但效果有所不同。采用对象指针或对象标识方式时，整体对象和部分对象是各自独立创建的。刚刚创建一个整体对象时，其指向部分对象的属性值为空。只有创建了部分对象，并为该属性赋值，整体对象才开始拥有这个部分。而且，在运行中可以动态地切断或恢复这种关系。部分对象的数据空间与整体对象相分离，只需在整体对象中留一个指针或对象标识的位置。这种方式适合于表达松散的、动态变化的整体—部分关系，并可表达一个部分对象能够同时属于多个整体的情况。采用嵌套对象方式时，一个部分对象成为整体对象不可分割的一部分，其数据空间包含在整体对象之中，从生存时间看，它与整体对象同生同灭。这种方式适合于表达紧密的、固定不变的整体—部分关系，如汽车和发动机。

在 OOA 的详细说明中可以指明这是哪种方式的整体—部分结构，在对整体对象中代表部分对象的属性进行详细说明时，指出它的数据类型。如果是用指针或对象标识，则是前一种方式；如果是用部分对象的类直接作为其数据类型，则是后一种方式。

5.1.2　表示法

如图 5-2 所示是表示组合关系的图形符号。图中上部是一个整体对象，下部是组成该组合关系的方向：从三角形顶角引出的线指向整体对角，从三角形标记连接。三角形标记表明组合关系的方向：从三角形顶角引出的线指向整体对象，从三角形底边中点画出的线连到部分对象。通常，把整体对象画在图的上部而把部分对象放在下部，这样安排有助于模型的理解。

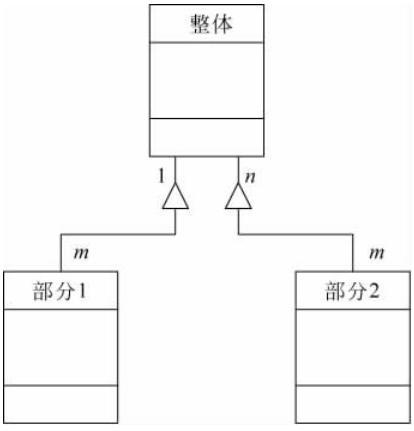

图 5-2　整体—部分结构表示

组合关系具有的最重要的性质是传递性，也就是说，如果 A 是 B 的一部分，B 是 C 的一部分，则 A 也是 C 的一部分。当组合关系有多个层次时，可以用一棵简单的聚集树来表示它。聚集树是多级组合关系的一种简化表示形式。整体—部分结构连接符两端所标的数字或字母用于表明该结构中对象实例的“多重性”。这里，所谓多重性是指，位于连接符一端的一个对象实例要求另一端多少个对象实例与自己进行整体—部分组合。

5.1.3　如何发现整体—部分结构

整体—部分结构可以清晰地表达问题域中事物之间的组成关系，同时它又是一种用途更为广泛的系统构造手段。这里讨论如何从问题域发现整体—部分结构。

问题域中事物之间的组成关系表现为多种方式。从多种方式考虑事物之间的组成情况是发现整体—部分结构的基本策略。考虑以下几个方面。

1. 组织机构和它的下级组织及部分

在当今社会的政治、经济、科技、文化、工业、商业等各个领域都有各种形式的组织机构，

并且常常需要在系统中进行管理。整体—部分结构可以很好地表达它们之间的组成关系。例如，一个公司有若干子公司和市场部、产品部、公关部等部门，而它的某些子公司也需要设立这些部门，可由如图 5-2 所示的整体—部分结构表达。

2. 物理上的整体事物和它的组织部分

这种整体—部分关系是大量存在的，也是最容易发现的。大到星系、星球，小到基本粒子，宇宙万物处处存在这种整体—部分关系。如果一组存在这种关系的整体对象和部分对象都需要在系统中表示，并且需要表明它们之间的组成关系，则应建立整体—部分结构。

3. 组织与成员

组织与它的成员也是一种整体—部分关系。例如，班级与学生、办公室与办事员等。这种整体—部分关系也可推广到人员以外的其他事物，故又称之为“集合与成员”的关系。

4. 抽象事物的整体与部分

对象的作用，本来就包括对问题域中某些抽象事物的再抽象，所以整体—部分结构也应表达这种事物之间的组成关系。推而广之，人类的脑力劳动所产生的事物中，凡是要在系统中用对象表示的，均在考虑之列。

5. 一种事物在空间上包容其他事物

例如，工厂中有管理人员、工人和机器，可把工厂看作整体对象，而把管理人员、工人和机器看作部分对象。

6. 具体事物和它的某个抽象方面

在有些情况下，往往需要把具体事物的某个抽象方面独立出来作为一个部分对象来表达。例如，把人员的基本情况（如姓名）用“人员”对象描述，而把他的工作职责、身份、立功受奖事迹等（假如都需要有较多的信息来表示）独立出来，用一些部分对象来表示并与“人员”对象构成整体—部分结构。

5.1.4 审查与筛选

按 5.1.3 节所介绍的策略可以发现许多候选的整体—部分结构，但仍然需要进行审查与筛选，可以从以下几方面考虑其是否需要。

1. 是不是系统责任的需要

仅当结构中的整体对象和部分对象都是系统责任需要的，并且二者之间的整体—部分关系也是系统责任要求表达的，才有必要建立这个结构，否则就不应该保留。即使现实中的整体对象和部分对象都是系统责任要求描述的，也仍要考虑，经过抽象之后体现这两类对象之间整体—部分关系的信息是否真正需要在系统中保持。从实现考虑，如果准备建立一个紧密的整体—部分结构（用嵌套对象），则意味着，整体对象中将包括部分对象的全部信息；如果准备建立一个松散的整体—部分结构（用对象指针或对象标识），则意味着整体对象将

保持部分对象的踪迹。如果系统责任没有这种要求，就不要建立这种结构，否则只能增加复杂性并造成浪费。

2. 是否属于问题域

整体—部分结构中的整体对象和部分对象都应该属于当前的问题域，否则就不需要这个结构。例如，在一个学校的教务管理系统中，尽管教师和他们的家庭可组成整体—部分结构，但家庭不属于问题域，所以不应该保留这个结构和这个无用的对象。

3. 是否有明显的整体—部分关系

如果两个对象之间不能明显地分出谁是部分、谁是整体，则不应该用整体—部分结构表示。例如，系统中要表示教师与学科之间的关系，但是教师和学科不能分出哪个是整体，哪个是部分。对这种情况不应采用整体—部分结构，而应该用实例连接，实例连接将在后面介绍。

4. 部分对象是否有一个以上的属性

如果部分对象只有一个属性，应考虑把它取消，合并到整体对象中去，变为整体对象的一个属性即可。例如，“发动机”作为“汽车”的部分对象，如果只有一个属性“马力”，则可合并到“汽车”对象中，在“汽车”对象中增设一个“马力”属性。这样做是为了使系统简化。

5.1.5　简化对象的定义

任何一种好的软件开发方法或软件评估标准都不希望系统的组成单位过于庞大，因为这样的系统单位将难以理解并增加隐藏错误的机会。

实际系统中，某些对象的内部构成可能是相当复杂的，其表现为：对象含大量的属性和服务，某些属性是较为复杂的数据结构，或服务中包含较为复杂的功能。这种对象的类定义将会很庞大。

在 OOA 模型中，如果某些对象的定义过于复杂，应想办法把它简化，通常的方法是：在一个复杂对象的内部进行“再分析”，看它的某些属性与服务是不是描述了该对象的某个独立部分。如果是，则用它们组成一个部分对象，从整体对象的类定义中分离出来，建立整体—部分结构。

5.1.6　支持软件复用

在整体—部分结构中通过组装而支持软件复用是 OO 方法颇受重视的优点之一。在以下两种情况下都可以运用整体—部分结构而实现或支持复用。

一种情况是在两个或更多的对象类中都有一组属性和服务描述这些对象的一个相同的组成部分。把它们分离出来作为部分对象，建立整体—部分结构，这些属性和服务就被多个类复用，从而简化了它们的描述。例如，在一个机械加工厂的生产管理系统中，各种机床、起重机和电动送料车等对象类中都有一组属性和服务，描述这些对象中所装配的电动机。有

些 OO 方法的入门者可能首先想到运用继承来解决上述问题：把电动机作为一般类，使机床、起重机和送料车通过继承而拥有它的属性和服务。这固然也可以解决问题，但对问题域的映射却远不如使用整体—部分结构。因为把机床、起重机和送料车等对象看作一种特殊的电动机，道理上很难说得通，而说它们都装有一台电动机则非常合理。为了复用，可把这组属性和服务分离出来，建立如图 5-3 所示的结构。

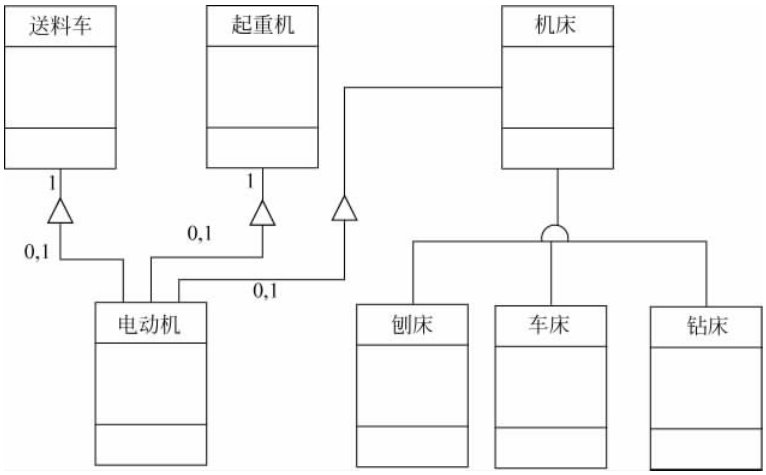

图 5-3 整体—部分结构描述复用

另一种情况是系统中已经定义了某类对象，在定义其他对象时，发现其中一组属性和服务与这个已定义的对象是相同的。那就不必再重复地定义这些属性与服务，只需建立它与前一类对象之间的整体—部分结构。

还可以考虑通过整体—部分结构提取可复用构件，以支持领域范围的复用问题。如果一个对象类中有一组属性和服务描述了该对象的一个独立部分，即使从本系统看并不需要这样一个部分对象，只要它是一种在本领域经常使用的对象，就可把它从整体中分离出来作为部分对象，使它的类定义成为一个应用范围较广的可复用构件。

5.1.7 整体—部分结构的进一步运用

整体—部分结构是一种表达能力很强的系统构造手段，还可以把它作为一种改进系统构造和解决某些棘手问题的有效手段。

1. 表示动态变化的对象特征

OO 方法作为一种新技术，在实际应用中常常会遇到一些新问题，所谓“新问题”可能只是方法的应用者在自己以往的实践中未经历的，或者在他们读过的文献中未曾提及的，而不一定是这种方法现有的概念与技术不能解决的。对象的某些属性与服务在系统的运行和演化过程中发生动态的变化便是此类问题之一。最近几年曾多次遇到国内一些研究者以解决这一问题为动机试图创造一些新的 OO 概念，并开发相应的机制来支持这些新概念。对此，专家曾给予的建议是：为解决这种问题而创造新概念的理由是不充分的。运用 OO 方法的整体—部分结构可以简单而自然地解决这一问题。

可说明这个问题的一个典型例子是某些系统中的“人员”对象。在系统运行中随着时间的推移人员身份在发生变化，于是在人员对象中，除一些描述人员基本情况的属性（如姓名、出生年月）和服务（如上下班打卡）保持不变之外，描述其身份的属性和服务则需随着这个人的身份变化而变化。这种变化，不是通过属性值的变化或服务在执行时的某些微小差别所能表达的；它可能需要一组从语法到语义根本不同的新的属性和服务。例如，作为营业员时需要“班组”“班次”“柜台号”属性；当上会计师之后就不需要这些了，需要的是“注册会计师号”等属性和“收入”“支出”等服务。这就是所谓的“动态变化的对象特征”。

用一个类定义的对象，其属性与服务，从数量到名称、语义都将保持不变，永远与创建这个对象的类相符。多态性也不能解决这个问题，它只能使不同类的对象中相同命名的属性与服务有不同的语义，而不能使同一个类的对象随着时间的变化而改变自己的属性的服务。

一种大体上可行的解决办法是针对各种可能的特征变化，预先定义多个类，当对象的特征需要变化时，则删除旧的对象而用另一个类创建一个新对象。这种解决办法有三种缺陷：一是让对象反复地从一个类中删除而加入另一个类，将为数据管理带来困难；二是对象有相当一些不需要变化的属性，随着旧对象的删除和新对象的建立，需要把旧对象中的有用信息复制到新对象中；三是概念上不太自然，一个对象仅仅因为一些特征发生了变化，就要让它经历一番死而复生的阴阳轮回，显得太有点儿小题大做了。

其实，一种比较简单的解决办法是用整体—部分结构来解决这一问题。首先分析一个对象哪方面的特征变化需要由一些动态变化的属性与服务来描述。把这些属性与服务分离出来组成一个部分对象，并与整体对象组成松散的整体—部分结构。系统在运行中动态地产生新的部分对象，以代替旧的部分对象。例如，在上述例子中“人员”对象中动态变化的属性与服务是为了描述人员身份的变化，把它们独立出来，组成一个“身份”对象，作为一种松散的结构，实现时可在“人员”对象中用指针指向表示自己当前身份的部分对象。当身份发生变化时，创建一个新的“身份”对象实例并使“人员”对象的指针指向新的身份。

这种解决办法没有引进任何新的概念，而且前面提到的解决办法的缺点它都避免了。首先，那些不需要动态变化的属性与服务都在整体中稳定地保持着，不必受部分特征变化的牵连而频繁地删除和重建，这为实现时的数据管理避免了很多麻烦。需要动态变化的属性与服务被组成部分对象，在系统中进行独立的处理，并可方便灵活地与整体对象拼接。此外，它在概念上显得很自然、很容易理解——对象不需要改换自己所属的类，只需要更换自己的一个发生了变化的组成部分。最重要的一点是，这种解决方法在实现上没有任何困难，它不需要任何特殊的支持机制。

2. 表示数量不定的组成部分

一个对象中若含有某种数量不定而内容相同的组成部分，则会给实现带来困难。例如“书”这种对象，含有描述其每一章的属性，假如每一章都用三个属性来描述，一本书可能只有几章，也可能多达几十章；那么，“书”的对象类应怎样定义？定义几组描述其各章的属性？太少了对于某些对象可能不够用，按多的数量定义又使大部分对象造成空间浪费。在这种情况下可用整体—部分结构解决，如图 5-4 所示。

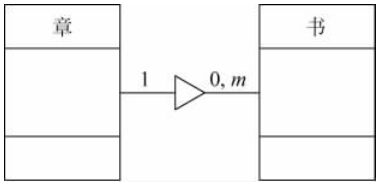

图 5-4　表示数量不定的部分

5.1.8 调整对象层和属性层

对于每个整体—部分关系，整体对象中要增加一个属性来表明它的部分对象。在该属性的详细说明中要给出这个属性的数据类型。如果是紧密的结构，用部分对象的类作为其数据类型；如果是松散的结构，用对象指针或对象标识作为其数据类型。

定义整体—部分结构的活动可能发现一些新的对象类，或者从整体对象的类定义中分割出一些部分对象的类定义，应把它们加入到对象层中，并给出它们的详细说明。引起整体对象的属性与服务应该被划分出去作为部分对象。对此，类符号及其详细说明都要做相应的修改。

5.2 一般—特殊结构

5.2.1 一般—特殊结构及其用途

一般—特殊结构是由一组具有一般—特殊关系（继承关系）的类所组成的结构。以两种方式给出了一般类和特殊类的定义，从类的特征来看：如果类 A 具有类 B 的全部属性和全部服务，而且具有自己特有的某些属性或服务，则 A 叫作 B 的特殊类，B 叫作 A 的一般类。从类集合的元素来看：如果类 A 的全部对象都是类 B 的对象，而且类 B 中存在不属于类 A 的对象，则 A 是 B 的特殊类，B 是 A 的一般类。这两种定义本质上说是等价的。

特殊类之所以称为“特殊”，是因为它具有独特的属性与服务，一般类的某些对象不符合这些条件，使特殊类成为一个较为特殊的概念；而一般类之所以“一般”，是因为它的属性与服务具有一般性，这个类以及它的所有特殊类的对象都应该具有这些属性与服务，所有这些对象都属于一般类，使一般类成为一个较为广泛的概念。一般类的特征集合则是特殊类集合的真子集；而特殊类的对象实例集合是一般类对象实例集合的真子集，如图 5-5 所示。

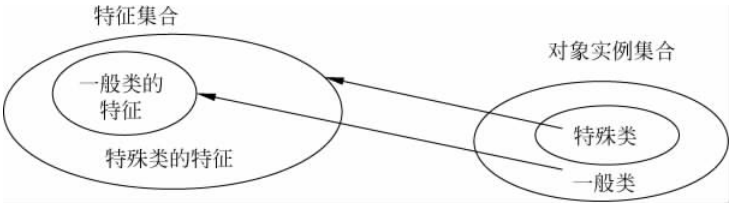

图 5-5 对象与特征关系

一般类与特殊类之间的关系叫作一般—特殊关系，又称为“is-a-kind-of”关系。一般—特殊结构的定义是：一般—特殊结构是把一组有一般—特殊关系的类组织在一起而得到的结构，它是一个以类为结点，以一般—特殊关系为边的连通有向图。

这种关系是传递的，即若类 A 继承类 B，类 B 继承类 C，则 A 也继承了 C 的全部属性与服务。继承分为单继承和多继承。如果每个特殊类只直接地继承一个一般类，则这种继承称为单继承；如果一个特殊类直接地继承两个以上的一般类，则称为多继承。

如果在一般—特殊结构中仅存在单继承，则它是一个以最上层一般类为根的树，称作层次结构；如果结构中存在多继承，则它是一个半序的连通有向图，称作网格结构。

一般—特殊结构是问题域的事物之间客观存在的一种关系。在 OOA 模型中建立一般—特殊结构，是为了使系统模型更清晰地映射问题域中事物的分类关系。它把具有一般—特殊关系的类组织在一起，可以简化对复杂系统的认识。它清楚地表达了一般类和特殊类之间的关系，使人们对系统的认识和描述更接近日常思维中对一般概念和特殊概念的处理方式。

OOA 中通过一种继承机制实现这一功能，只需指明一个类是另一个类的特殊类，继承机制将保证特殊类自动地拥有一般类的全部属性与服务。一般—特殊结构可简化类的定义——对象的共同特征仅在一般类中给出，特殊类通过继承而拥有这些特征，从而不必再重复地加以定义。

OOA 模型中的一般—特殊结构将为编程阶段用一种 OOPL 实现类之间的继承提供依据，它不但简化了 OOA 文档，最终还将导致一个结构清晰的、较简练的程序。而且，当软件维护人员从程序追溯到 OOA 文档时，他们将看到一个如实地映射问题域中对象分类关系的结构。

整体—部分结构和一般—特殊结构是两种迥然不同的结构，一个用于描述对象之间的组成关系，一个用于描述对象类之间的继承关系。OOA 同时需要这两种结构以解决不同的问题。

两种结构在概念上的差别是很明显的，一个体现了“is-a-kind-of”关系，一个体现了“has-a”关系。概念上如此不同的两种结构，在有些情况下，二者之间却是可以互相变通的，或者说，它们可以达到殊途同归的效果。这是由于，一般—特殊结构是使特殊类通过继承而拥有一般类的特征，整体—部分结构是使整体对象通过组装而拥有部分对象的特征。尽管途径不同，着眼点不同，结果却是一样的：一些对象拥有另一些对象的特征。说法不同，实质内容是一样的。在 OOA 中选用哪种结构，要看其实际问题用哪种结构表达最为自然。

从实现的角度看，整体—部分结构对编程语言的要求远不像一般—特殊结构那样严格。有些 OO 语言不支持多继承，非 OO 的语言连单继承也不支持，但几乎任何一种当前流行的编程语言都可以实现整体—部分结构。可以运用整体—部分结构将多继承转化为单继承或无继承，从而使模型与编程结果能够更好地对应。这也是对两种结构的变通使用。

5.2.2 表示法

一般—特殊结构的表示法，是用一般—特殊结构连接符来连接该结构中的每个类，如图 5-6 所示。其中，图 5-6(a)是一般—特殊结构连接符，从圆弧引出的连线连接到一般类，从直线分出的连线连接到每个特殊类。图 5-6(b)是一个完整的一般—特殊结构，它包括结构中的每个类。

5.2.3 如何发现一般—特殊结构

本节主要介绍如何从不同的角度努力发现可能有用的一般—特殊结构。后面将对发现的结构进行审查、调整和简化，处理异常情况，最后建立合适的结构。为了发现一般—特殊结构，有以下可供采用的策略。

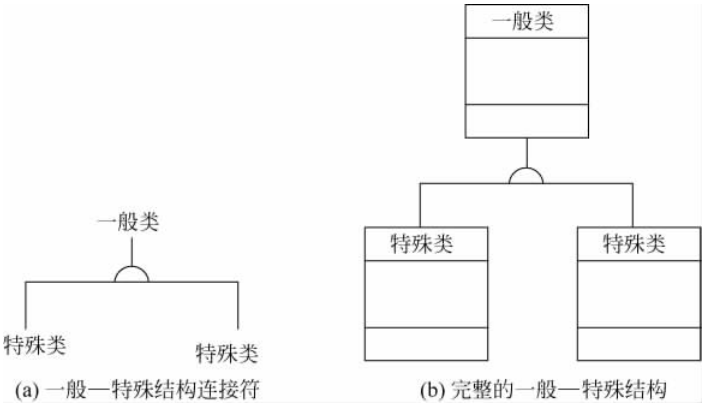

图 5-6 一般—特殊结构

1. 按常识考虑事物的分类

按自己的常识,从各种不同的角度考虑问题域中事物的分类,可以形成一些建立一般—特殊结构的初步设想,从而启发自己发现一些确实需要的一般—特殊结构。

2. 学习问题域的分类学知识

分析员应该花一番功夫学习一点儿与当前问题域有关的分类学知识。分类是一门学问,在许多行业和领域已经形成了一套科学的分类方法。例如,动物分类学、植物分类学、图书分类法等都已经成为一门学科。问题域现行的分类方法往往比较正确地反映了事物的特征、类别以及各种概念的一般性与特殊性,学习这些知识,将对认识对象及其特征、定义对象类、建立一般—特殊结构有很大的帮助。

3. 按照一般—特殊结构的定义分析

按照一般—特殊结构的两种定义,可引导从两种不同的思路去发现一般—特殊结构。一种思路是把每个类看作一个对象集合,分析这些集合之间的包含关系。如果一个类是另一个类的子集,则它们应组织到同一个一般—特殊结构中。另一种思路是看一个类是不是具有另一个类的全部特征,当发现一个类中定义的属性与服务全部在另一个类中重新出现时,应考虑建立一般—特殊结构,把后者作为前者的特殊类,以简化其定义。两种思路的最终结果是相同的,但可以作为两种不同的手段互为补充。

4. 考察类的属性与服务

对系统中的每个类,从以下两方面考察它们的属性与服务:一方面看一个类的属性或服务是否适合这个类的全部对象。如果某些属性或服务只能适合该类的一部分对象,说明应该从这个类中划分出一些特殊类,建立一般—特殊结构。这是一个“自顶而下”地从一般类发现特殊类并形成结构的策略;另一方面检查是否有两个(或以上的)类含有一些共同的属性和服务。考虑若将这些共同的属性与服务提取出来,能否构成一个在概念上包含原先那些类的一般类,组成一个一般—特殊结构。

5. 考虑领域范围内的复用

为了加强 OOA 结果对本领域多个系统的可复用性，应考虑在更高的水平上运用一般—特殊结构，使本系统的开发能贡献一些可复用性更强的类构件。假如从本系统看，在你的 OOA 模型中这个类的定义已经很合理了，可是考虑到它在同一个领域的可复用性，则存在不足。

5.2.4　审查与调整

找到了许多候选的一般—特殊结构之后，要对它们逐个加以审查，从而取消那些不合适的结构或对它进行调整与修改。通过以下几个问题进行审查。

1. 是否符合分类学的常识

一般—特殊结构中各个类之间的关系应该符合分类学的常识和人类的日常思维方式，如果违背了这些，就会产生一种怪异的、有悖常理的“结构”。造成这种问题的原因是在建立结构时只注意到属性与服务的继承，而没有注意与问题域的实际事物之间分类关系的对应。检查这种错误的方法是用“is-a-kind-of”关系来衡量每一对一般类与特殊类。

2. 系统责任是否需要这样的分类

在一个候选的一般—特殊结构中，特殊类与特殊类以及一般类与特殊类之间，虽然从概念上讲是有所区别的，但是系统责任却未必要求做出这样的区别。例如，一般类“开发人员”和特殊类“设计人员”及“编码人员”之间，在概念上是有所不同的。但在一个具体系统中，若系统责任没有对设计人员和编码人员的特殊要求，则不要建立这个结构，只要“开发人员”这个类就行了。

3. 问题域是否需要这样的分类

无论是按分类学的知识还是按常识找到的结构，都不一定是问题域真正需要的，考虑分类时必须从问题域出发。

4. 是否构成了继承关系

有时，会出现这样一种情况：虽然按常识某些类之间应该是一种一般—特殊关系，但在系统中经过抽象之后所得到的类却没有什么可以继承的属性与服务。正确的做法是，保持这两个类之间的互不相干，不要为之建立一般—特殊结构。因为在系统中类之间没有继承，就失去了一般—特殊结构的意义。

通过以上的审查，删除或修改了不合适的一般—特殊结构，使准备在系统中建立的结构都是合理正确的。

5.2.5　多继承及多态性问题

按照问题域和系统实际要求，如果类之间的关系是多继承的，则应该建立多继承的一般—特殊结构。OOA 模型应该如实地映射问题域，所以在 OOA 中暂不考虑实现条件，按

问题域和系统责任的原貌建立结构，到 OOD 时再根据具体的实现条件做必要的调整。

可以从两个思路来发现多继承的结构。一是看特殊类是不是被包含于两个(或以上)一般类的交集；二是看一个特殊类是否同时需要两个(或以上)一般类的属性与服务。两种思路出发的角度不同，结果却是一致的。图 5-7 是一个多继承结构的例子。

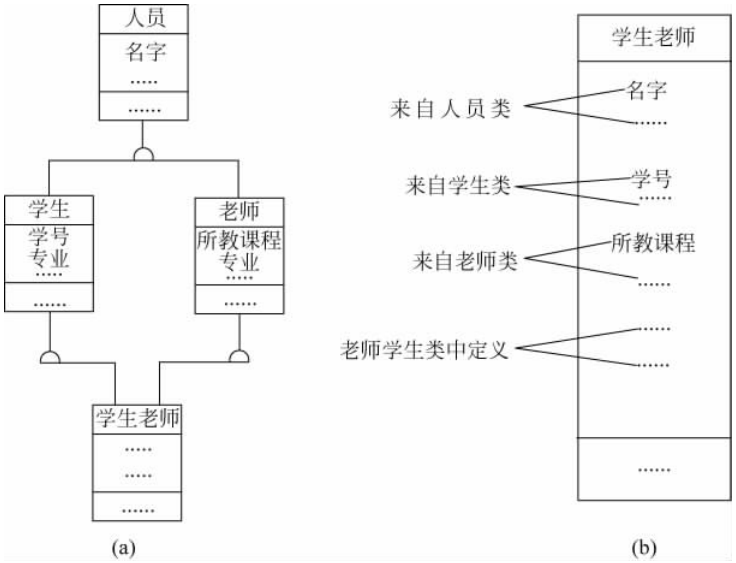

图 5-7 多继承

在这个例子中，“学生老师”类同时继承了“学生”和“老师”两个类的属性与服务。它实际拥有的属性和服务如图 5-7(a)所示；通过两个直接的一般类从更上层的一般类“人员”继承的姓名等属性，应该只有一份，不应该出现两次；从“学生”类和“老师”类继承的属性，各自占有一部分空间，互不重叠；其余是在本类定义的特殊属性。服务的继承情况与此类似。

注意：“学生老师”类从“学生”类和“老师”类都继承了一个名为“专业”的属性，但二者有不同的含义。

“学生老师”的对象类中，从“老师”类继承的是该生作为一名老师在工作中从事的专业，从“学生”类继承的“专业”则是该生作为一名学生攻读学位的专业。在这种情况下，就会出现命名冲突的现象，这将使系统无所适从。

分析员最好能在 OOA 阶段避免命名冲突问题，解决的办法是：从多继承的特殊类开始，向上检查它的每一条继承路径，不同继承路径上的属性与服务的命名不要重复，如有重复则加以修改。

例如，把图 5-7(a)中的“学生”类和“老师”类的“专业”属性分别改为“攻读专业”和“从事专业”。

有时，还需要在一般—特殊结构中表达对象的多态性。这里多态性是指在一般—特殊结构的各个类中名字相同的属性及服务具有不同的语义。例如，一般类“多边形”定义的“边数”“边长”“顶点数据”属性和“绘图”服务，将被它的两个特殊类继承。但希望有如下的多态性。

(1) “多边形”类。

边数：指出该多边形的边数。

边长：指出该图形 4 个边的长度。

顶点数据：由每个顶点的坐标构成的数组。

绘图：用直线连接每两个相邻的顶点。

(2)“正方形”类。

边数：取消该属性。

边长：指出该正方形边的长度。

顶点数据：由每个顶点的坐标构成的数组。

绘图：用直线连接每两个相邻的顶点。

(3)“长方形”类。

边数：取消该属性。

边长：指出长方形的长和宽。

顶点数据：由每个顶点的坐标构成的数组。

绘图：用直线连接每两个相邻的顶点。

这样的多态性应该在 OOA 模型中表示出来，并在详细说明中分别给出不同的定义，以便为多态性的实现提供依据。首先，在特殊类中把语义与一般类不同或者拒绝继承的属性和服务重新写出来。如果是语义不同，则在属性名或服务名之前加“*”符号；如果是拒绝继承，则在它前边加“×”符号；其他与一般类的原样继承相同。符合上述例子要求的多态性表达如图 5-8 所示。

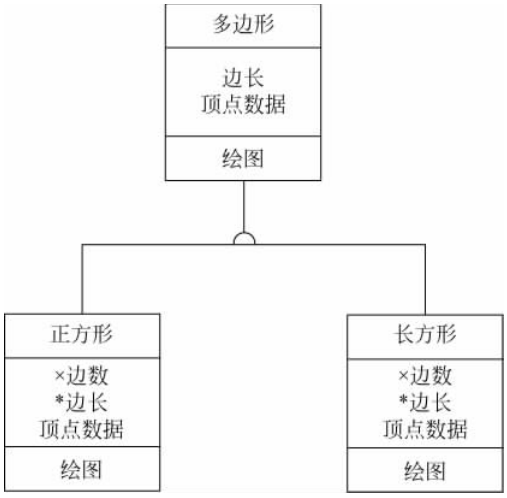

图 5-8　多态性表达

如果在特殊类中出现了与一般类中名字相同的属性或服务，而又未做任何多态性标记，这将被认为是一个错误。模型一致性检查应能发现这种错误，此时分析员应该检查自己的本意到底是什么。如果是忘记了加“*”和“×”符号，就加上；如果是照一般类的原样继承，则在特殊类中去掉它，如果是两个不相干的属性或服务，只是名字重复了，则为其中一个更换名字。

最后，提醒初学 OO 方法的读者注意以下两个容易混淆的概念。

(1) 重命名(Rename)和重载(Overload)是两个不同的概念。重载是实现多态性的方

法之一，它修改继承来的属性或服务的内容而不更改其名字。重命名是解决多继承带来的命名冲突问题的方法之一，它更改属性或服务的名字而不修改其内容。

(2) 多继承(Multiple Inheritance)和多态性(Polymorphism)也是截然不同的两个概念，它们之间没有任何必然的联系。

5.2.6 一般—特殊结构的简化

一般—特殊结构把问题域中具有一般—特殊关系的事物组织在一起，在一般类中集中地定义对象的共同特征，通过继承简化特殊类的定义。然而，如果不加节制地建立一般—特殊结构，也会带来一些不利的影响，表现为两种现象：一是建立过深的继承层次，增加了系统的理解难度和处理开销；二是从一般类划分出太多的特殊类，使系统中类的设置太多，增加了系统的复杂性。所以对一般—特殊结构的运用要适度。重点检查以下几种情况。

一种情况是某些特殊类之间的差别可以由一般类的某个属性值来体现，而且除此之外没有更多的不同。例如，在某一系统中需要区别人员的性别和国籍，但是这些人员对象除了性别和国籍不同之外在其他方面都没有什么不同。此时，如果按分类学的知识建立一个一般—特殊结构，系统中就要增设大量的特殊类，这没有太大必要。简化的办法是取消这些特殊类。

第二种可以简化的情况是：特殊类没有自己的特殊的属性与服务。但是在一个软件系统中，每个类都是对现实事物的一种抽象描述。

抽象意味着忽略某些特征，如果体现特殊类与一般类差别的那些特征都被忽略了，系统中的一般—特殊结构就出现了这种异常情况——特殊类除了从一般类继承下来的属性与服务之外，自己没有任何特殊的属性与服务。

第三种可以简化的情况是：一个一般类之下只有其他的特殊类，并且这个一般类没有可创建的对象实例。在这种情况下，这个一般类的其他用途就是向仅有的一个特殊类提供一些被继承的属性与服务。此时可以取消这个一般类，同时把它的属性与服务放到特殊类中。这种简化策略不但可减少类的数量，而且可有效地压缩类的继承层次。有些经验不足的开发者在认识到继承的好处之后往往过分地运用继承，使系统中的类形成很深的继承层次。

通常，系统中的一般类应符合下述条件之一才有存在的价值，如果不符合下述任何条件，则应考虑简化。

(1) 需要用它创建对象实例。

(2) 它有两个或两个以上的特殊类。

(3) 它的存在有助于软件复用。

5.2.7 调整对象层和特征层

定义一般—特殊结构的活动将使分析员对系统中的对象类及其特征有更深入的认识。在很多情况下，随着结构的建立需要对类图的对象层和特征层做某些修改，包括增加、删除、合并或分开某些类，以及对某些属性与服务增、删或把它们移入其他类。对此，分析员应随时返回到对象层和特征层，做出必要的修改。为了用一般—特殊结构连接符连接结构中的每个类，并达到整齐且美观的效果，对类的位置也要做必要的调整。在 OOA 工具的支持

下，上述修改与调整都是很容易的。

5.3　实例连接

本节讨论对象之间的另一种关系——实例连接。首先介绍实例连接的基本概念、用途及表示法，然后讨论几种复杂的情况。

5.3.1　简单的实例连接

实例连接又称为链，它表达了对象之间的静态关系。静态联系是指最终可通过对象属性来表示的一个对象对另一个对象的依赖关系。这种关系在现实中是大量存在的，并与系统责任有关。如果这些关系是系统责任要求表达的，或者为实现系统责任目标提供了某些必要的信息，则 OOA 应该把它们表示出来，即在以上每两类对象之间建立实例连接。

实例连接是对象实例之间的一种二元关系，在实现之后的关系中它将落实到每一对具有这种关系的对象实例之间，但是在 OOA 中没有必要做如此具体的表示，只需在具有这种实例连接关系的对象类之间统一地给出这种关系的定义。

1. 表示法

本节中将讨论实例连接中的一种最简单的情况，即两类对象之间不带属性的实例连接，其表示法如图 5-9(a)所示：在具有实例连接关系的类之间画一条连接线把它们连接起来；连接线的旁边给出表明其意义的连接名；在连接线的两端用数字标明其多重性。图 5-9(b)概括了因两端的多重性不同而形成的三种情况：一对一的连接，一对多的连接和多对多的连接。实例连接线每一端所标的数可以是一个固定的数、一个不定的数、一对固定的或不固定的数，线的一端所标的数表明本端的一个对象将和另一端几个对象建立连接，即它是本端对另一端的要求。

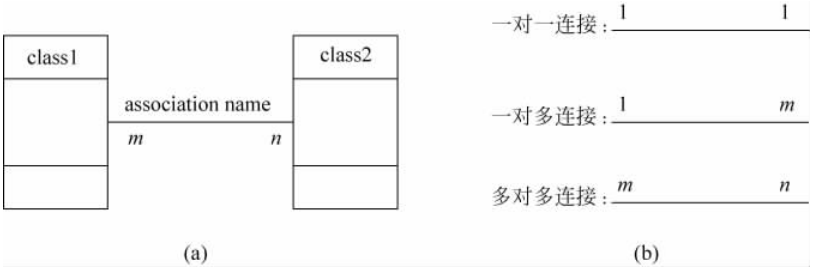

图 5-9　对象之间实例连接

2. 实现方式

分析员在工程实践中应该独立于具体的实现条件去建立 OOA 模型，但也应该适当地了解这种方法提供的 OOA 概念可以用什么技术来实现。这种知识背景将有助于分析员更恰当地运用各种 OOA 概念。

实例连接一般可用对象指针来实现。即在被连接的两个类中选择其中一个，在它的对象中设立一个指针类型的属性，用于指向另一个类中与它有连接关系的对象实例。这种属性一般只要在一个类的对象中设立就够了。若连接的某一端标注的多重性是固定的，且数量较少，则在这一端的对象中设立指针对实现较为有利。

3. 实例连接与整体—部分结构的异同

实例连接与整体—部分结构有某些相似之处，又有一些差别。在概念上，它们都是对象实例间的一种静态关系，并且都是通过对象的属性来体现的。但它们的差别是，实例连接中的对象之间没有这种语义，即分不出谁是整体、谁是部分；整体—部分结构中的对象在实现世界中含有明显的“has-a”语义。在实现上，实例连接绝不能用紧密的整体—部分结构所用的嵌套对象来实现，但它和松散的整体—部分结构实现方法是有一定相似之处的。

5.3.2 复杂的实例连接及其表示

对象之间的静态联系不能那么简单。在某些情况下，仅指出两类对象的实例之间有或者没有某种联系是不够的，应用系统可能要求给出更多的信息。例如，在要求指出某个教师为某个学生指导毕业论文的同时，还要求给出论文题目、答辩时间、成绩等信息。又如，在要求指出两个城市之间有无航线的同时，对于有航线的情况要求表明航线距离和每周班次。对于这样的系统要求，OOA 有两种解决问题的方法：一种方法是采用复杂的实例连接（或链）的概念，扩充实例连接的表达能力，使它可以带有属性，甚至可带有操作（服务）；另一种方法是采用较纯的、形式单一的 OO 概念，用普通的对象来描述这种复杂性。以下分别讨论这两种方法并做出本书的选择。

1. 采用复杂的实例连接概念

参照 OMT 的概念与表示法，可以允许实例连接带有一组属性，这些属性通过一个关联（Association）来描述。这种方法的思路是，对象之间的关系，有时并不是通过一个由对象实例构成的二元组就能充分表达的，还需要附加一些属性信息。所以，OOA 方法就扩充实例连接的概念，加强它的表达能力，并引入关联的概念及表示法，用以描述实例连接的属性。进一步按这种思路考虑可以看到，实际应用中有时不仅要求实例连接带有一些属性信息，还可能要求给出一些操作。一个关联可能要记录一些信息，可以引入一个关联类来记录。关联类通过一根虚线与关联连接，其表示法如图 5-10 所示。

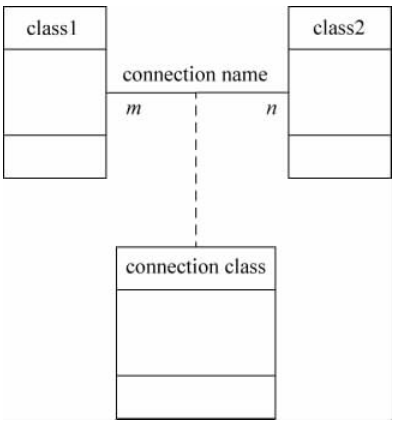

图 5-10 复杂的实例连接

图 5-10 中的关联，既含有属性，又含有操作，和普通的对象类构成颇为接近，所以在 OMT 中把它作为一种对象类来看待，称为“作为类的关系”，并把实例连接看作它的对象实例。

对于只带有属性，而不带操作的实例连接可供考虑的实现方式有以下几种。

（1）在实例连接一端的对象中设立一组属性，其

中一个属性是指向实例连接另一端对象的指针；其余的属性是在关联中列出的连接属性。这种方法的问题是：连接属性在建模时放置在关联符号中，而实现中被搬到连接一端的类中，使模型与程序不能很好地对应。此外，对于多对多的实例连接，容易造成空间浪费。

（2）根据关联定义一个结构数据类型，其中两个域变量是分别指向两端对象的指针，其余域变量是连接属性。用这个数据类型定义的结构变量作为一个具体的实例连接。这种方法可解决多对多的连接问题，但关联所对应的是一种非 OO 的成分。

（3）根据关联定义一个结构数据类型，其中一个域变量是指向一端对象的指针，其余域变量是连接属性。用这个数据类型定义是另一端对象的一个表示连接的属性，于是，该属性中就包括一个对象指针和所有的连接属性。这种实现方法在一定程度上改善了程序与模型的对应。但多对多的连接问题仍未解决，而且关联所对应的也是一种非 OO 的成分。

（4）把关联用一个类来实现，用这个类的对象代表两端对象之间一个具体的实例连接。它的属性包括所有的连接属性和指向两端对象的指针。这种方法使程序具有更强的 OO 风格，也解决了多对多的连接问题。

对于既带有属性，又带有操作的实例连接，由于它带有操作，所以最合理的实现方式就是用一个类来实现一个关联，用这个类的对象代表两端对象之间的一个具体的实例连接。它的属性与服务分别是实例连接要求描述的属性与操作，并可在属性部分安排指向两端对象的指针。

无论是仅带有属性的实例连接，还是既带有属性又带有服务的实例连接，在编程时用一个类来实现它时，模型中两个类之间复杂的实例连接关系在程序中变为以一个中间类为过渡的简单的而又规则的关系，它在语法上就和程序中其他的没有什么区别，完全被作为一个普普通通的类来处理。可见，解决这一问题不需要增添新的概念或者把原有的概念复杂化；可以运用原有的面向对象概念，以纯 OO 的风格来实现这种连接。相形之下，OOA 模型中的这种表示却与纯 OO 的风格有相当的差距。尽管可以把关联称作一个类，但它与普通的类有许多不同：它只能依附于其他类之间的实例连接而存在，并且难以用一种整齐划一的方式表示它与外部的关系。

2. 用对象表示实例连接的复杂性

若两类对象之间的联系带有某些复杂的信息，这说明它们之间存在着某种事物（可能是抽象事物）。这种方法所依据的原理是，对象不仅可用于表示有形的事物也可用于表示无形的事物。当两类对象之间的实例连接比较复杂时（带有一些属性或操作），说明在它们之间存在某种尚未用对象加以描述的事物。按前面的方法，把它的描述信息附加到实例连接之后表示出来就造成了实例连接的复杂性。而当更充分地运用对象的观点，用对象来表示这种事物之后，所有的复杂信息都被包含在这种新增的对象之内，所有实例连接就变得简单了。原先一个复杂的实例连接变为两端的对象与新增的对象之间的简单实例连接；原先的两类对象之间的直接联系变为通过新增的对象的间接联系。

这种方法使 OOA 模型中的每个对象类在形式上都是一致的。类之间所有的实例连接形式也都是一致的，并且是简单的。这种方法也很好地解决了实现多对多实例连接的困难：由于新增的对象类可以根据两端对象类的要求创建数量足够的对象实例，所以从它引出的实例连接线的多重性总是“1”。因此，即使一个多对多的实例连接不带有属性和操作，也可

以用这种增加中间对象的方法来化解它的多重性。另外，对象类以及它们的对外关系都不需要做任何特殊处理。这使得 OOA 模型具有一种更规范的 OO 风格，并且向后续阶段的开发人员提供了更确定的语义信息。

前面给出的方法也有它的可取之处，它比较符合人们考虑问题由浅入深的思维过程。分析员首先考虑到类图中两个对象类之间存在某种联系，于是画出它们之间的连接线；进而看到，这种连接带有一些属性和操作，于是在连接线之上增加一个关联符号来描述这些属性与操作。这种思维过程是比较自然的。把它作为 OOA 过程的一种中间表示方法，利用它引导分析员认识和表示连接的属性与操作，但最终把它转化为纯 OO 的表示形式，如图 5-11 所示。

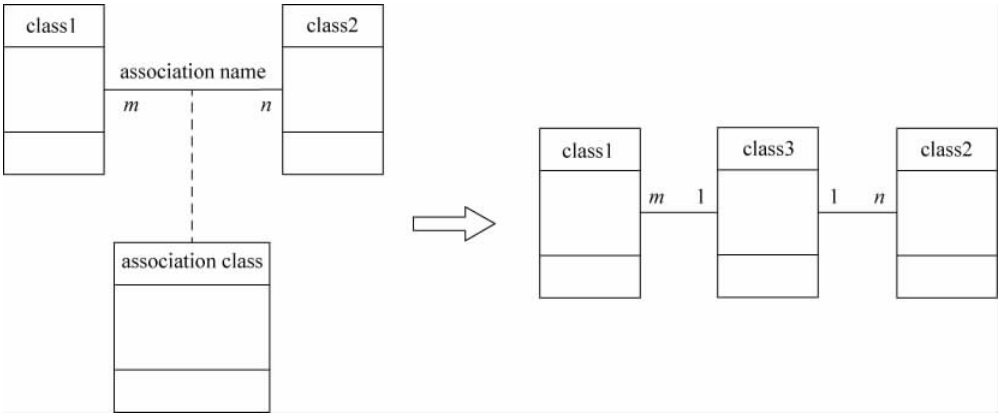

图 5-11　关系转换

5.3.3　三元关联问题

J. Rumbaugh 等在 OMT 中注意到三类对象之间的联系问题，并称之为三元关联（Ternary Association）。例如，在“人员”“项目”和“语言”三类对象之间可能需要表达这样的关系：“某人员使用某种语言从事某个项目”。于是他们使用了一种三元关系的表示符号，如图 5-12 所示。

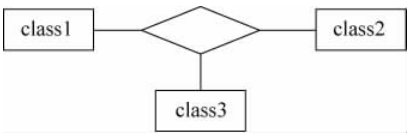

图 5-12　三元关联

其实，事物之间的关系三元也不是最多的，还可以举出四元、五元甚至更多元关联的例子，如果 OOA 方法要创造这么多表示符号来表示这些关系，就显得太杂乱了。最好的办法还是通过充分运用对象的概念来解决多元关联的表示问题。这样的表示法也解决关联带有属性或操作的问题，并同时化解了多对多的连接。

5.3.4　如何建立实例连接

在 OOA 模型中建立实例连接包括下述分析活动。

1. 认识对象之间的静态联系

首先从问题域和系统责任考虑，各类对象之间是否存在着某种静态联系。然后，重点从

系统责任考虑，这种联系是否需要在系统中加以表示，即这种联系是否提供了某些与系统责任有关的信息。有时虽然从问题域的现实情况来看，对象之间也发生联系，但若系统责任不要求表示这些信息，则不必建立其实例连接。

2. 认识实例连接的属性与操作

对于考虑中的每一种实例连接，进一步分析它是否应该带有某些属性和操作。就是说，是否含有一些仅凭一个简单的实例连接不能充分表达的信息。例如，在用户工作站的例子中，是否需要给出优先级、使用权限等属性信息和开始对话的操作？如果需要，则可以先在实例连接线上附加一个关联符号来表示这些属性与操作，然后把它们转换为纯 OO 的表示，方法是：分析这些属性与操作可以用一种什么对象来表示，增设这个对象类，并分别建立它与原有的两个类之间的简单实例连接。

3. 分析实例连接的多重性

对于每个实例连接，从连接线的每一端看本端的一个对象可能与另一端的几个对象发生连接，把结果标注到连接线的本端。

4. 异常情况处理

1）多对多实例连接的处理

用增加对象类的办法解决带有属性或操作的实例连接问题或解决多元关联问题时，其中多对多的实例连接问题也同时得到了解决。现在剩下的问题是，一些二元的、不带属性与操作的实例连接是多对多的。多对多的实例连接对实现所带来的麻烦是，无论在连接线哪一端的对象设立指向另一端对象的指针，数量都是不固定的，对此，实现时只有两种选择，一是采用较复杂的数据结构（例如把对象指针组织成链表），二是预留充分多的空间。虽然不带属性的实例连接在对象属性中只占若干指针的位置，空间浪费问题不太严重，但若 OOA 模型能够避免这种情况，无疑将为实现带来很大的方便。解决此问题的方法是，在多对多实例连接两端的对象类之间，插进一个对象类，并在它和两端的对象类之间分别建立一对多的实例连接。

2）多元关联的处理

如果系统中存在着多元关联，可在多元关联的汇集点增设一个对象类，使之转化为二元的实例连接。策略是反复地陈述这种多元关系，分析这种陈述是在描述一种什么事物。通过这样的处理，可把多元关联用简单的二元实例连接清晰地表示出来。

5. 命名与定位

经过以上的处理，使 OOA 模型中最终使用的表示符号只是简单的实例连接符号，没有关联联系和多元连接符号。这样的实例连接语义大部分是能一目了然的。当一条实例连接线的语义不那么清楚时，可以给它取一个。命名一般可用动词和动宾结构。这样的命名暗示实例连接实际上是有向的，但是为了表示法的简单和处理的灵活性，没有采用有向的连接线作为表示符号，也不强调命名暗示了什么方向。

实例连接的定位问题是指：当连接线的某一端是一个一般—特殊结构时，要考虑连接线画到结构中的哪个类符号上。原则是，如果这个实例连接适应结构中的每一个类的对象，

则画到一般类上，如果只适应其中某些特殊类，则画到相应的特殊类上。

5.3.5 对象层、特征层的增补及实例连接说明

在建立实例连接的过程中可能增加一些新的对象类。特别是在分析复杂的实例连接、多元关联及解决多对多的问题时，都要求增加一些新的类，要把这些新增的类补充到类图的对象层中，并建立它们的类描述模板。

对于每一个实例连接，应该在它某一端所连接的对象类中增加相应的属性，它的类型应该被说明为指向另一端对象的指针。在这个类的描述模板中，要给出这个属性的详细说明，特别是要说明它所代表的实例连接有什么实际意义，把 OOA 模型中仅靠一条连接线和实例连接命名不能详尽表达的内容都确切地表达出来，但详细说明要求准确无误地加以说明。

5.4 消息连接

本节介绍如何分析和认识对象之间在行为上的依赖关系，并通过消息连接来表示这种关系，从而使 OOA 模型最终成为一个有机的整体。

5.4.1 消息的定义

在现实生活中，消息(Message)这个词指的是人或其他事物之间传递的一种信息，在一般的软件系统中，消息这个术语较多地应用于进程之间的通信。广义地理解，一个软件成分向其他软件成分发出的控制信息或数据信息，都可称为消息。一个消息应具有发送者和接收者共同约定的语法和语义，接收者在收到消息之后，将按照其要求做出某种反应。

在 OO 方法中，按封装的要求消息是对象之间在行为上的其他联系方式，即：对象以外的成分不能直接地存取该对象的属性，只能向这个对象发送消息，由该对象的一个服务对收到的消息做出响应，完成发送者要求做的事。消息的定义是：消息是面向对象发出的服务请求。消息连接描述对象之间的动态联系，即：若一个对象在执行自己的服务时，需要请求另一个对象为它完成某个服务。消息连接是有向的，从消息发送者指向消息接收者。

在这里，对顺序系统(没有并发执行的多个任务)和并发系统要分开进行讨论，以下对这两种情况分别加以讨论。

5.4.2 顺序系统中的消息

顺序系统中的一切操作都是顺序执行的。它的 OOA 模型只有一个主动对象(并且这个主动对象中只有一个主动服务)，其余的对象都是被动对象。实现之后的系统在运行时，将只对应一个处理机调度单位(进程或线程)。如图 5-13 所示，系统从其他的主动对象 A 的主动服务 a 开始运行，当它需要其他对象(被动对象)的某个服务为它完成某项工作时，就向

它发一个消息，控制点转移到接收消息的对象服务，使这个服务开始执行。接收消息的服务根据消息所表达的要求完成相应的工作后，控制点返回到发送消息的对象服务，并(在必要时)带回消息的处理结果。图 5-13 中向被动对象 B 发消息，使 B 的服务 b 执行；b 在执行时先后向 C、D 两个被动对象发消息，其中 D 的服务 d1，在执行时又向 D 的另一个服务 d2 发消息。

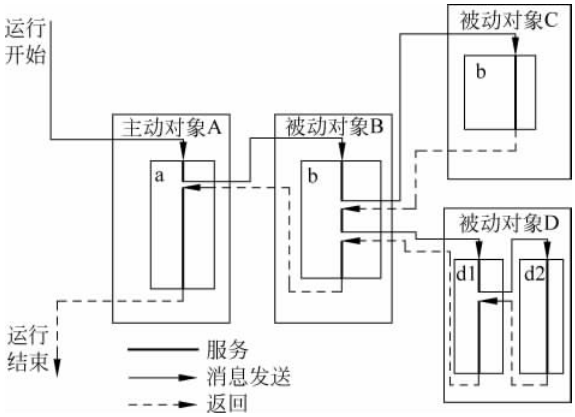

图 5-13　顺序系统中的消息

每个接收消息的服务执行完之后都返回到发送点，最后返回到 a，当 a 执行完时系统运行结束。此时，发送者将继续执行在这个消息之后的其他操作。所以被动对象的服务都是在消息的驱动下才能执行的。当它们执行时如果需要其他的对象服务为它们完成某项工作，也同样向它们发出消息，一切过程都和以上的叙述相同。

在顺序系统中，对象之间的消息具有下述特点。

(1) 每个消息都是向对象发出的一个服务请求，它必定引起接收者一个服务的执行。

(2) 每个消息的发送与接收都是同时进行的，即消息都是同步的。

(3) 除了主动对象其他的主动服务之外，其他对象服务只有在接收到消息时才开始执行。

(4) 消息是从正在执行的服务中发出的。消息发出之后，发送者暂停执行位于消息发送点之后的其他操作，将控制点转移到接收者，直到接收者执行完相应的服务之后才返回到发送消息的服务，继续执行这个消息之后的其他操作。也就是说，所有的操作都是串行的。

因此，在顺序系统中，消息是面向对象发出的服务请求是合适的。在语法上，一个消息的描述应包括以下内容。

(1) 消息名，即接收消息的服务名。

(2) 接收消息的服务要求的输入参数，即入口参数。

(3) 接收消息的服务提供的输出参数，即返回参数。

在语义上，一个消息应包括下述信息。

(1) 发送者，这是通过消息发送点的位置隐含表明的。

(2) 接收者，是由消息名表达的。

(3) 其他需传送的信息，通过入口参数和返回参数表示。

5.4.3 并发系统中的消息

并发系统是有多个任务并发执行的系统。它的 OOA 模型含有多个主动对象和若干被动对象。系统实现之后，这些主动服务将对应一些并发执行的处理机高度单位，这里采用“控制线程”这个术语。在并发系统中，将有多个控制线程并发地执行，每个控制线程是由一系列顺序执行的操作所构成的活动序列。

图 5-14 是一个并发系统的例子。

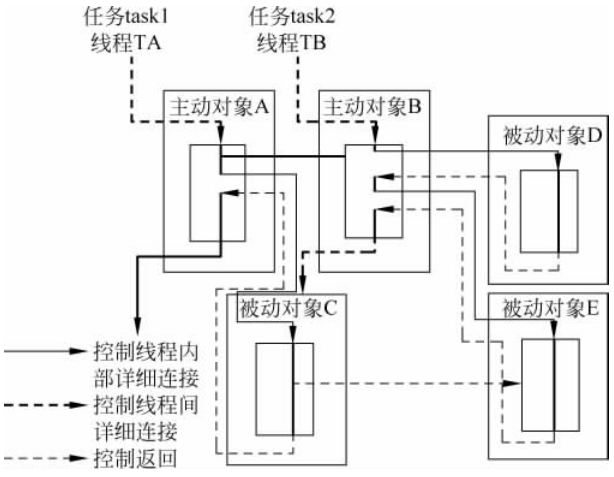

图 5-14 并发系统中的消息

假设系统中有两个需要并发执行的任务 Task1 和 Task2，分别用主动对象 A 和 B 描述这两个任务。其中，A 在执行时将顺序地使用被动对象 C 提供的服务；B 在执行时顺序地使用被动对象 D 和 E 提供的服务。系统运行时，将有两个控制线程 Ta 和 Tb 并发地执行。但在 Ta 中，对象 A 和 C 的服务是顺序执行的，在 Tb 中对象 B、D 和 E 的服务也是顺序执行的。可以看出，并发系统中的消息分为两种情况：一种是发生在一个控制线程内部的消息，这种消息和顺序系统中的消息是完全相同的；另一种情况是两个或多个控制线程之间的消息，例如，在控制线程 Ta 执行到主动对象 A 的服务时，向执行控制线程 Tb 的主动对象 B 发出的消息，或者 Ta 执行到被动对象 C 的服务时向 Tb 中的被动对象 E 发出的消息。

两个或多个控制线程之间的消息与顺序系统中的消息有许多不同，具体表现在以下几个方面。

1. 消息的多种用途

在并发执行的控制线程之间传送的消息，有以下几种不同的用途。

(1) 向接收者发出一个服务请求。

(2) 向接收者提交一些数据。

(3) 向接收者发布一个通知或事件信息。

(4) 向接收者传递一个同步控制信号。

从以上的各种用途来看，考虑到并发系统中的情况，消息在完全用对象表示的系统中可定义为：消息是对象之间在一次交互中所传送的通信信息。

这些信息可能是一些提交给接收者阅读和理解的数据信息，例如向接收者提交一些数据或者向接收者发布一个通知或事件信息；也可能是某种直接起到控制作用的控制信息，例如向接收者传递一个同步控制信号；或者两者兼而有之，比如说向接收者发出一个服务请求。这里强调“一次交互”，是为了表明消息是一个原子的信息通信单位。在多次交互中传送的一批信息，不能看作一条消息，而应看作多条消息。构成通信信息的一个元素也不能称为一条消息，因为它不能形成完整的语义。

2. 消息的同步与异步

不同控制线程之间的消息可分为同步(Synchronous)消息和异步(Asynchronous)消息，二者的区别在于发送消息与接收消息的动作是否同时发生。

同步消息的定义是：仅当发送者要发送一个消息而且接收者已做好接收这个消息的准备时才能传送的消息称为同步消息。无论发送者还是接收者，如发现对方未做好准备都必须等待。

异步消息的定义是：发送者不管接收者是否做好接收准备都可以发送的消息称为异步消息。

还有人提出了另外两种消息：一种是阻断(Balking)消息，另一种是限时(Timeout)消息。

阻断消息的定义是：与同步消息相同，只是当接收者未做好接收准备时发送者放弃发送消息的操作。

限时消息的定义是：与同步消息相同，只是当接收者未做好接收准备时发送者将只等待一个限定的时间。

阻断消息和限时消息都属于同步消息，它们都必须在接收者做好接收准备时才能发送。如果接收者未做好接收准备，无论是立即放弃本次发送操作还是等待一定的时间再放弃，都意味着发送者将恢复执行发送点之后的操作。如果它一定要把这个消息发出去，则在设计策略上可反复地做发送尝试，直到接收者能够接收时才真正发送，所以说它们都属于同步消息。

3. 发送者对消息处理结果的不同期待方式

(1) 发送者在发出消息之后等待，直到得到处理结果才继续原先的工作。

(2) 发送者不等待处理结果，发出消息之后立刻继续执行，只是在以后的某个执行点或某种时机查看消息的处理结果。

(3) 发送消息后，发送者既不等待，也不再关心其处理结果。

4. 接收者对消息的不同响应方式

(1) 在消息产生之前，处理这个消息的进程或线程并不存在；仅当发送者发这个消息时才立刻创建一个进程或线程来响应这个消息，完成它所要求的服务。

(2) 处理消息的进程或线程已经存在，并且与发送者同步地接收消息。接到消息时立即处理。

(3) 某些消息所指出的接收者可能并不关心这种消息,它可能不做任何响应。

(4) 接收者异地接收和处理消息,即在消息发出之后的某个时刻才接收和处理该消息。

5. 消息的接收者是否其他

根据消息的接收者是否其他,可以区分为两种情况:一种情况是消息定向地发送给其他的接收者,称作定向消息;另一种情况是把消息发送给某个范围内所有可能的接收者,称作广播消息。

并发系统中消息的多样性一方面起因于需求的多样性,另一方面起因于丰富多彩的设计技巧和日益增多的实现支持技术。OOA 应该只注重于考虑需求问题,而不必过多地关心设计与实现细节。所以,并发系统中的消息尽管有这么多情况,但并不是所有的细节差异都需要在 OOA 中认识和表达,OOA 应该认识和表达有关消息的那些问题,以及辨别那些不同情况。

5.4.4 消息对 OOA 的意义

在用面向对象方法构造的系统中,消息体现了对象行为之间的依赖关系。它是实现对象之间的动态联系,使系统成为一个能活动的整体,并使各个部分能够协调工作的关键因素。如同人体的神经,有了它,整个机体才能活动。

对顺序系统而言,OOA 通过对消息的分析而建立对象之间的动态联系。消息体现了过程抽象的原则:在一个对象的服务中通过消息而引用其他对象的服务。OOA 通过定义消息而把所有的对象服务贯穿在一起,在系统实现之后,它们将在一个控制线程中顺序地执行。

在并发系统中有多个任务并发地执行。开发这种系统的难点在于:这些任务所包含的操作在执行时没有固定的时序关系,容易发生与时间有关的错误,而且这些错误在调试时往往是难以再现的。因此,OOA 对并发系统的行为分析中最关键的问题是把所有的对象服务组织到一些彼此并发执行,但内部不再存在并发问题的控制线程中。在每个控制线程内部,按与顺序系统相同的方式定义对象之间的消息,同时把解决并发问题的注意力集中于分析各个控制线程之间的消息通信。

5.4.5 OOA 对消息的表示——消息连接

通过前面的讨论可以看到,顺序系统中的消息是比较简单的,并发系统中的消息则有许多不同的情况。OOA 对消息的考虑,重点是系统责任对消息的不同要求。作为一种既适应顺序系统,又适应并发系统的 OOA 方法,应该识别和表示的主要问题如下。

(1) 对象之间是否存在某种消息。

(2) 这种消息是同一个控制线程内部的还是不同控制线程之间的。

(3) 每一种消息的发出者和接收者。

(4) 消息是同步的还是异步的。

(5) 发送者是否等待消息的处理结果。

这些问题是按其重要性排列的。尽管已经忽略了消息的许多细节,但要把它们都在

OOA 模型中表示出来仍然太烦琐。因为这需要引进许多不同的消息表示符号，并在类图上对存在多种消息的类符号之间画多连接线或者附带多种标记。这将增加模型的复杂性，并影响其清晰度。

增强 OOA 模型的表达能力和保持模型的简明性是矛盾的两个方面，并不是使用的表示符号越多，对问题辨别得越细微，方法就越好。必须承认以下两个事实：第一，分析只是软件工程中的一个环节，分析员只能认识和描述系统开发过程中的一部分问题，其余问题还将由设计人员和编程人员继续进行认识和描述；第二，分析文档中的图形化表示，只是对分析员所得到的认识给出一种简明、直观的表示。因此，讨论一种方法的表达能力，以及它的语义确定性等问题，只能着眼于分析员应该认识而且能够认识的问题范围，不能期望它对系统开发中的一切问题都能进行详细的、确切的表示。它的作用是让模型的阅读者能快速地捕捉系统构造中的主要问题，引导和帮助人们的形象思维，而不是把分析员所能认识的问题毫无遗漏地都在模型图中表示出来。

目前，国际上几种影响较大的 OOA 方法，大部分只是在模型中用一条带箭头的线表示两类对象之间有消息传送，没有更详细地表示是否有多种消息，每一种消息由哪个服务发出，哪个服务接收，以及消息的同步与异步等问题。这些问题都只在详细说明中表明。只有 G. Booch 方法在模型图中采用了较详细的表示法：标明每种消息将由接收者的哪个服务来处理，并引入了 5 种表示符号以区别消息是简单的、同步的、阻断的、限时的还是异步的。

在本节开头列出的 5 个问题中前两个问题是最需要在模型中表达的。其中，确定对象之间是否存在某种消息是最重要的，因为模型应该表示出哪些类的对象之间存在行为上的依赖关系。确定该消息是同一个控制线程内部的还是不同控制线程之间的也很重要，因为把同一个控制线程内部的消息和不同控制线程之间的消息用不同的连接线加以区别，可以使源于不同主动对象的每一条执行路线都能清晰地呈现给阅读者，避免互相交叉和混淆。区别这一点实际上比区别消息的同步与异步问题更为重要。后面三个问题也是 OOA 应该认识和表示的。其中，说明消息由发送者的哪个服务发出以及由接收者的哪个服务处理，对于设计、实现和测试都是有用的信息；消息的同步与异步，以及发送者是否等待处理结果对设计和编程也很有用。但这三个问题对于从总体上理解系统的行为依赖关系不像前两点那么重要，所以不需要在 OOA 模型中表示，只需要在详细说明中给出确切的说明。

所以，这里使用了两种消息连接符号来表示对象之间的消息传送关系，如图 5-15 所示。其中，图 5-15(a)表示同一个控制线程内部的消息连接；图 5-15(b)表示不同控制线程之间的消息连接。

图 5-15　控制线程内部的消息连接

消息连接的定义是：消息连接是 OOA 模型中对对象之间行为依赖关系的表示，即若类 A 的对象在它的服务执行时需要向类 B 的对象发送消息，则称存在着从 A 到 B 的消息连接。

“消息连接”在概念上和“消息”有所不同。一个类的对象可能向另一个类的对象发送多种消息，但在 OOA/OOD 模型中至多使用两条消息连接线（带箭头的实线或虚线），它们表示了从第一个类向第二个类发送的全部消息。

两个类的对象之间可能互相发消息，例如在类 A 和类 B 之间，既有从 A 发送给 B 的消息，也有从 B 发送给 A 的消息。此时，可以使用两个方向相反的消息连接符号，但是更简单的方法是使用一个双向的消息连接符号，如图 5-16 所示。

图 5-16　两个方向相反的消息连接

在同个对象内部的不同服务之间也可能需要传送消息。这种情况是很常见的，如果在每个存在这种情况的类符号上都画一条从它发出又指向它自己的连接线，就显得太杂乱。OOA 模型的主要作用是表示不同类之间的关系，内部的消息连接关系一般不需要在模型图上表示。不过，如果这种消息是发生在不同控制线程之间的，则在模型图中显式地表示出来是有好处的。

5.5　如何建立消息连接

本节介绍如何在 OOA 模型中建立消息连接，首先讨论如何建立每个控制线程内部的消息连接；其次介绍如何建立各个控制线程之间的消息连接；最后讨论对象分布对消息的影响。

5.5.1　建立控制线程内部的消息连接

此活动的基本策略是“服务模拟”和“执行路线追踪”，其具体做法是从类图中每个主动对象的主动服务开始，做下述工作。

（1）模拟当前对象的执行，考虑：为了完成当前的工作，需要请求其他对象提供什么服务。每当发现了一种新的请求，就是发现了一种新的消息。

（2）分析该消息的发送者与接收者在执行时是否属于同一个控制线程。可从以下几个不同的角度去判断。

① 按问题域的情况和系统责任的要求应该顺序地执行还是并发地执行。

② 从发送者的执行到接收者的执行是否引起了控制线程的切换。

③ 接收者是否只有通过当前这种消息的触发才能执行。

（3）在当前服务的详细说明中指出由它发出的每一种消息的接收者，从当前服务所在的类向所有接收消息的对象类画出消息连接线。

（4）沿着控制线程内部的每一条消息追踪到接收该消息的对象服务，重复进行以上的工作。

当从每个主动对象服务开始的这种服务模拟和执行路线追踪都进行完毕时，对全系统中的对象类做一次检查，看是不是每个类的每个服务都曾经到达并模拟执行过。如果某个

服务从未到达，则有两种可能：一种可能是遗漏了向这个服务发出的消息，另一种可能是这个服务是多余的。找出在何处发生了遗漏，加以补充；确实无用的服务应该删除。

5.5.2 建立控制线程之间的消息连接

此项工作仅仅在并发系统的分析中需要。5.5.1节介绍的策略在找出各个控制线程内部消息的同时，可能也发出了一些控制线程之间的消息，但可能很不完全。所以，需要进行更全面的分析。

由于已经找出了各个控制线程内部的消息，因而可使分析员以这些源于主动对象的控制线程作为并发执行单位，对整个系统的动态执行情况进行全局的观察，从而发现这些控制线程之间需要哪些消息。

对每个控制线程主要考虑以下几个问题。

(1) 该控制线程在执行时，是否需要请求其他控制线程中的对象为它提供某种服务？这种请求由哪个对象发出？由哪个对象中的服务进行处理？

(2) 该控制线程在执行时，是否要向其他控制线程中的对象提供或索取某些数据？

(3) 各个控制线程的并发执行，是否需要传递一些同步控制信号？

(4) 它在执行时是否将产生某些对其他控制线程的执行有影响的事件？

(5) 一个控制线程将在何种条件下中止执行？在它中止之后将在何种条件下由其他控制线程唤醒？用什么办法唤醒？

(6) 这个消息由一个控制线程中的哪个对象服务发出？由另一个控制线程中的哪一个对象服务来处理？

根据对上述问题的思考与回答，在相应的类符号之间画出用虚线箭头表示的消息连接符。进一步分析，消息应该是同步的还是异步的，以及发送者是否等待消息的处理结果，分别在发送者和接收者的类描述模板中针对有关的服务做该消息的详细说明。

5.5.3 对象分布问题及其消息的影响

在面向对象的软件开发中，系统功能分布与数据分布将通过对象的分布而体现。在OOA阶段不能最终确定如何把系统中的对象分布到联网的各台处理机上。因为这涉及选用什么机器及软硬件配置的问题，而这些问题需要在设计时根据具体的实现条件最终确定。但是分布问题又不纯粹是设计时的问题，它在很大程度上也属于系统需求。实际上，很多系统在项目可行性论证或需求报告中就已经提出了一些对系统功能及数据的分布要求。这些来自用户或经过专家论证的系统分布要求未必涉及很多具体实现条件，无论用什么方案来实现，都应该予以满足。所以对分布问题应该从两个方面来看，一方面是设计决策问题，另一方面是系统需求提出的分布要求。OOA应该只考虑后一方面的问题，前一方面的问题则应推迟到OOD中考虑。

分布问题对OOA的影响如下。

(1) 同一台处理机上的对象之间的消息通信既可能是一个控制线程内部的,也可能是不同控制线程之间的。分布在不同处理机上的对象之间的消息通信只能是不同控制线程之间的。它们在实现时所依赖的消息通信机制也有所不同。

(2) 在每个子处理机上分布的一组对象中,至少应有一个对象是主动对象;所有的被动对象都是在位于本机的主动对象驱动下运行的。如果这一条不满足,应考虑增加主动对象,或把其中的某些被动对象改为主动对象。

根据系统需要中已经明确的分布要求,分析员可把 OOA 模型中的对象进行初步的分组。设想每一组对象是分布在一台处理机上的,从而确定,哪些消息属于本机的通信,哪些消息属于不同处理机之间的通信。在未考虑对象分布问题之前,可能把某两个对象之间的消息看作一个控制线程内部的消息,对接收者的服务请求如同一个顺序执行的函数调用。

但是,如果这两个对象分布到不同的处理机上,接收者就不能以这种方式提供服务,它必须与发送者属于不同的控制线程。那么,需要考虑接收者是在哪个主动对象的驱动下对外响应请求并提供服务的。如果缺少这样的主动对象,则考虑是把某个被动对象改为主动对象还是增加一个主动对象,然后分为以下三种情况定义对象之间的消息。

(1) 本地机上同一个控制线程内部的消息。

(2) 本地机上不同控制线程之间的消息。

(3) 异地机上不同控制线程之间的消息。

对象分布问题一般不能在 OOA 阶段完全确定,要在 OOD 中按实现时的软硬件配置进一步考虑分布问题。

5.6 消息的详细说明

消息的详细说明包括对接收者和发送者两方面的说明。

在接收者的类描述模板中对每个服务做如下说明。

(1) 说明由这个服务接收和处理的每一种消息,规定消息的格式及内容。

(2) 说明本服务是顺序执行的还是并发执行的。

(3) 有时还要说明消息是同步的还是异步的。

在发送者的类描述模板中对每个发送消息的服务做如下说明。

(1) 指出这个服务在执行时可能发出的每一种消息,给出接收者的类名和处理该消息的服务名。

(2) 说明接收者是与本服务顺序执行的还是并发执行的。

(3) 有时还要说明该消息是同步的还是异步的,以及发送者是否等待该消息的处理结果。

(4) 如果服务流程图是比较详细的,则应画出在什么位置上发送什么消息。

5.7 电梯控制系统部分关系结构

现在对前两章使用的电梯控制系统进行定义结构与连接活动，建立其 OOA 模型的关系层。

5.7.1 一般—特殊关系

可以发现电梯控制系统中的到达事件(ARRIVAL EVENT)、目的地事件(DESTINATION EVENT)和召唤事件(SUMMONS EVENT)都可以归类为事件类，故创造出一个电梯事件(ELEVATOR EVENT)作为这三个类的一般类。按照本章前面的方法，可得到以下的一般—特殊关系。

(1) 到达事件是电梯事件的特殊类。

(2) 目的地事件是电梯事件的特殊类。

(3) 召唤事件是电梯事件的特殊类。

5.7.2 整体—部分关系

在电梯控制系统中可以发现以下整体—部分关系。

(1) 电梯马达(ELEVATOR MOTOR)是电梯(ELEVATOR)的部分类，它们之间是一对一关系。

(2) 超载传感器(OVERWEIGHT SENSOR)是电梯(ELEVATOR)的部分类，它们之间是一对一关系。

(3) 到达面板(ARRIVAL PANEL)是电梯(ELEVATOR)的部分类，它们之间是一对一关系。

(4) 目的地面板(DESTINATION PANEL)是电梯(ELEVATOR)的部分类，它们之间是一对一关系。

(5) 楼层(FLOOR)是到达面板(ARRIVAL PANEL)的部分类，它们之间是一对一关系。

(6) 召唤面板(SUMMONS PANEL)是楼层(FLOOR)的部分类，它们之间是一对一关系。

5.7.3 连接

在电梯控制系统中可以发现以下实例连接。

(1) 电梯(ELEVATOR)与楼层(FLOOR)之间的实例连接。

(2) 目的地面板(DESTINATION PANEL)与目的地事件(DESTINATION EVENT)之间的实例连接。

5.7.4 电梯控制系统的关系层

根据以上的分析，将得到的结构与连接在 OOA 模型中画出来，就是模型的关系层，最终形成了如图 5-17 所示的整个类图。

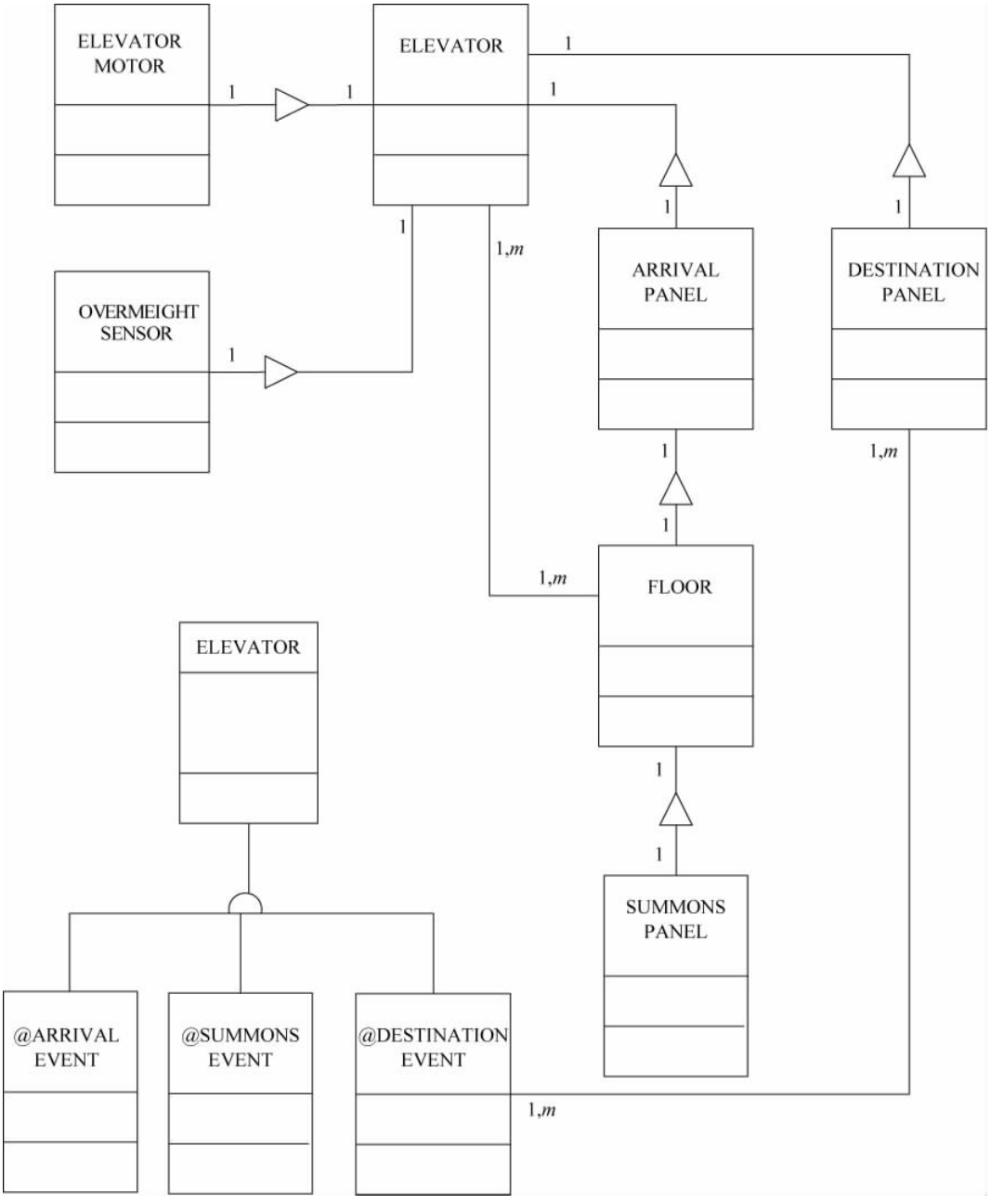

图 5-17 电梯控制系统的关系层

5.8 结构与连接实验

5.8.1 实验问题域概述

用户需求见 1.5 节。

5.8.2 实验 5

1. 实验目的

(1) 熟悉运用对象、类的结构及其分析方法。

(2) 掌握发现对象、类结构的方法。

(3) 学习正确地运用发现对象、类结构。

2. 实验环境

(1) 计算机一台,互联网环境。

(2) 绘图工具、文字编辑等工具软件。

3. 实验内容

根据图书管理系统用户描述,用发现分析对象、类结构的分析方法,认真分析每个对象的服务与属性,结合用户需求,分析得出对象、类间的结构关系。将图书管理系统中的一般—特殊结构、整体—部分结构、实例连接、消息连接至少各完成一个描述。

4. 实验步骤

(1) 准备好实验环境的机器(计算机)和互联网。

(2) 在机器上安装必要的软件平台(语言、绘图、文字编辑等)。

(3) 熟练掌握工具。

(4) 认真阅读题目,理解用户需求。用发现分析对象、类结构的分析方法,认真分析每个对象的服务与属性,结合用户需求,分析得出对象、类间的结构关系。

(5) 对所得图书馆系统的对象属性和行为进行分析、归纳,调整关系结构。

(6) 结束。

5. 实验报告要求

(1) 整理实验结果。

(2) 分析实验结果。阐述图书管理系统中的一般—特殊结构、整体—部分结构、实例连接、消息连接的分析过程。

(3) 小结实验心得体会。

小　　结

本章主要介绍了面向对象中的4种结构与连接，包括一般—特殊结构、整体—部分结构、实例连接以及消息连接。

综合练习

一、填空题

1. 对象类与外部的关系包括_____、_____、_____、_____。
2. 一般类的定义：_____。

二、选择题

1. 对象之间的静态联系用(　　)表示。

A. 一般—特殊结构　　B. 整体—部分结构

C. 实例连接　　D. 消息连接

2. 对象之间的动态联系用(　　)表示。

A. 一般—特殊结构　　B. 整体—部分结构

C. 实例连接　　D. 消息连接

三、简答题

1. 什么叫整体—部分关系？
2. 用图示表示整体—部分结构。
3. 列举出几种情况下运用整体—部分结构而实现或支持复用。
4. 画出一般类和特殊类的关系图。
5. 画图说明一般—特殊结构的表示法。