第3章 用 例 图

本章学习目标

(1) 理解用例模型的基本概念。

(2) 掌握识别参与者、用例的方法。

(3) 掌握用例模型中各种关系的分析。

(4) 掌握用例描述的分析与编写。

(5) 熟悉 UML 中用例建模的过程。

(6) 了解 UML 中用例建模的注意事项。

本章首先介绍参与者、用例的概念,以及它们之间的关系;再重点介绍如何识别参与者和用例,如何组织用例描述的内容;然后介绍 UML 的用例建模及注意事项;最后通过一个具体案例重点介绍用例建模的过程。

3.1 参 与 者

3.1.1 参与者的概念

参与者(Actor)是指系统以外的、需要使用系统或与系统交互的外部实体,包括人、设备、外部系统等。在国内,Actor 有很多不同的译名,包括参与者、活动者、执行者、行动者。在本书中,采用参与者这个译名。

【例 3.1】 在一个银行业务系统中,可能会有以下参与者。

(1) 客户:在银行业务系统中办理了账户的居民。他们通过银行业务系统进行金融交易。

(2) 管理人员:负责银行业务系统具体操作事务的办事人员。他们为客户办理存款、取款等金融交易。

(3) 厂商:进行转账支付的企业。厂商通过银行业务系统进行对公金融交易。

(4) Mail 系统:负责银行业务系统向个人发送电子邮件的软件系统。

实际上,参与者是一个集合概念。一个具体的外部实体仅代表同一参与者的一个实例。例如,在银行业务系统中,办理了账户的客户可能是张三,也可能是李四。因此,张三或李四都是“客户”参与者。此外,一个具体的外部实体可以充当多种不同的角色。例如,张三既可以是客户,也可以是银行的管理人员。因此,在用例建模中,不需要指向具体的参与者实例,而是指向集合特征的参与者。

在 UML 规范中,使用人形图标或带有版型标记的类图标来表示参与者。图 3.1 给出了参与者的 3 种表示形式。一般用人形图标表示的参与者是人,用类图标表示的参与者是设备或外部系统。

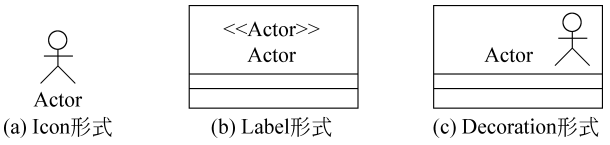

图 3.1　参与者的 3 种表示形式

【例 3.2】 在一个大学图书管理系统中，可能会有以下参与者。

（1）学生：在大学图书管理系统中办理了借书证的读者。他们通过大学图书管理系统进行书籍借还操作，以及查询书籍操作。

（2）教师：在大学图书管理系统中办理了借书证的读者。他们通过大学图书管理系统进行书籍借还操作，以及查询书籍操作。

（3）图书管理员：负责大学图书管理系统具体的书籍借还操作。学生或教师的书籍借还要通过图书管理员之手来完成。

（4）系统管理员：负责大学图书管理系统的用户维护和数据维护等操作。

一个参与者可以执行多个用例，一个用例也可以由多个参与者使用。需要注意的是，参与者实际上并不是系统的一部分，尽管他们会在用例模型中出现。参与者的内部实现是与系统无关的。

3.1.2　参与者之间的关系

由于参与者实质上也是类，用例图中的参与者之间有时会出现泛化的关系，这种泛化关系和类图中类之间的泛化关系是相似的。泛化关系的含义是把某些参与者的共同行为提取出来表示成通用行为，并描述成超类。参与者之间的泛化（Generalization）关系表示一个一般性的参与者（称为父参与者）与另一个更为特殊的参与者（称为子参与者）之间的联系。子参与者继承了父参与者的行为和含义，还可以增加自己特有的行为和含义，子参与者可以出现在父参与者能出现的任何位置上。

在 UML 规范中，泛化关系用带空心三角形箭头的实线表示，箭头指向父参与者。图 3.2 给出了参与者之间的泛化关系。

在下面的例子中，例 3.3 给出参与者之间在“含义”层次上的泛化关系，例 3.4 给出参与者之间在“行为”层次上的泛化关系。

【例 3.3】 在一个银行业务系统中，图 3.3 给出银行账户中部分角色之间的关系。“账户”参与者是父参与者，它描述了账户的基本特征（例如，账号、姓名、联系电话、住址等）。由于客户申请账户的方式不同：一类是通过柜台填表办理开户手续；另一类是通过网络申请办理开户手续，这类方式需要网络登录密码等信息。显然，两类客户具有相同的基本特征，但也有区别。因此，这两类参与者可以通过泛化关系来定义。

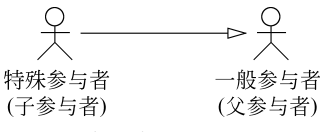

图 3.2　参与者之间的泛化关系

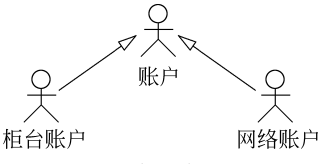

图 3.3　银行账户参与者之间的泛化关系

【例 3.4】 在一个网上购物系统中，一定会遇到用户权限问题。普通用户只能浏览网站上陈列的商品，而注册会员除了普通用户具有的操作之外，还有权限进行在线购物。其用例图如图 3.4 所示。

从图 3.4 中可以发现注册会员是一种特殊的普通用户，他们拥有普通用户所拥有的全部权限，此外他们还拥有自己独有的权限。因此，可进一步在普通用户和注册会员之间建立一个泛化关系，如图 3.5 所示。普通用户是父参与者，注册会员是子参与者。通过泛化关系，有效地减少了用例图中通信关联的个数，简化用例模型，便于人们理解。

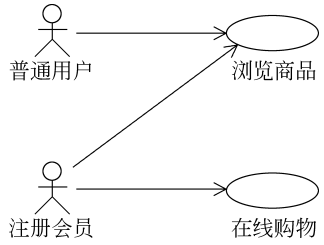

图 3.4 购物网站用户的用例图

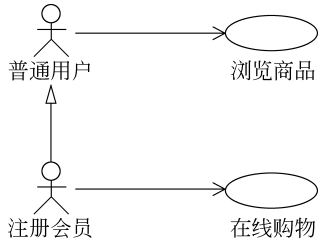

图 3.5 泛化后的购物网站用户的用例图

3.1.3 参与者和用例之间的关系

参与者和用例之间存在着一定的关系，这种关系属于关联关系(Association)，又称为通信关联(Communication Association)。这种关系表明了哪个参与者与用例通信。

在 UML 规范中，用例图中的参与者和用例之间的关联关系用带箭头或不带箭头的实线表示，箭头表示在这一关系中哪一方是对话的主动发起者，箭头所指方是对话的被动接受者。在图 3.4 中，“普通用户”参与者主动调用“浏览商品”用例，浏览网站上陈列的商品。所以，箭头指向“浏览商品”用例。

如果不想强调对话中的主动者与被动者，或者参与者和用例互为主动者与被动者，则可以使用不带箭头的实线来表示它们之间的关联关系。

3.2 用 例

3.2.1 用例的概念

用例(Use Case)这个概念是 Ivar Jacobson 在爱立信公司开发 AKE、AXE 系列系统时发明的，并在其博士论文 *Concepts for Modeling Large Realtime Systems*(1985 年)和 1992 年出版的论著 *Object-Oriented Software Engineering*：*A Use Case Driven Approach* 中做了详细论述。

自 Jacobson 的著作出版后，面向对象领域已广泛接纳了用例这一概念，并认为它是第二代面向对象技术的标志。在国内，Use Case 有很多不同的译名，包括用况、用案、用例。在本书中，采用用例这个译名。

目前对用例并没有一个被所有人接受的标准定义，不同的人对用例有不同的理解，不同的面向对象书籍中对用例的定义也是各种各样的。下面是两个比较有代表性的定义。

定义 1 用例是对一个活动者（Actor）使用系统的一项功能时进行交互过程的一个文字描述序列。

定义 2 用例是系统、子系统或类和外部的参与者（Actor）交互的动作序列的说明，包括可选的动作序列和会出现异常的动作序列。

用例（Use Case）是参与者（Actor）可以感受到的系统服务或功能单元。它定义了系统如何被参与者使用，描述了参与者为了使用系统所提供的某一完整功能而与系统之间发生的一段对话。

用例是代表系统中各个项目相关人员之间就系统的行为所达成的契约。软件的开发过程可以分为需求分析、设计、实现、测试等阶段，用例把所有这些都捆绑在一起，用例分析的结果也为预测系统的开发时间和预算提供依据，保证项目的顺利进行。因此，可以说软件开发过程是用例驱动的。在软件开发中采用用例驱动是 Jacobson 对软件界最重要的贡献之一。

在 UML 规范中，使用椭圆图标来表示用例。用例名往往用动宾结构或主谓结构命名（如果用英文命名，则往往是动宾结构，用例名中各单词之间可有空格，也可没有空格）。图 3.4 给出了“浏览商品”“在线购物”两个用例的表示形式。

【例 3.5】 在一个银行业务系统中，可能会有以下一些用例。

（1）浏览账户余额。

（2）列出交易内容。

（3）划拨资金。

（4）支付账款。

（5）登录。

（6）退出系统。

（7）编辑配置文件。

（8）买进证券。

（9）卖出证券。

【例 3.6】 在一个大学图书管理系统中，可能会有以下一些用例。

（1）借书。

（2）还书。

（3）预借。

（4）超期罚款。

（5）查找图书。

（6）图书维护（录入、修改、查询、删除）。

（7）借书证维护（录入、修改、查询、删除）。

（8）查看个人借书情况。

3.2.2 用例的特征

用例具有以下一些明显的特征。

（1）用例从使用系统的角度描述系统中的信息，即站在系统外部查看系统功能，而不考虑系统内部对该功能的具体实现方式。

(2) 用例描述了用户提出的一些可见的功能需求,对应一个具体的用户目标。使用用例可以促进与用户沟通,理解正确的需求,同时也可以用来划分系统与外部实体的界限,是面向对象系统设计的起点,是类、对象、操作的来源。

(3) 用例总是被参与者启动,并向参与者提供可识别的信息。

(4) 用例可大可小,但它必须是一个完整的描述。一个用例在系统设计和实现中往往会被分解成多个小用例。这些小用例的执行有先后之分,其中任何一个小用例的完成都不能代表整个用例的完成。只有当所有的小用例完成并最终产生了返回给参与者的结果,才能代表整个用例的完成。

(5) 用例在以后的开发过程中,可以进行独立的功能检测。

(6) 用例是对系统行为的动态描述,属于 UML 的动态建模部分。UML 中的建模机制包括静态建模和动态建模两部分,其中静态建模机制包括类图、对象图、构件图和部署图;动态建模机制包括用例图、顺序图、协作图、状态图和活动图。需要说明的是,有些书中把用例图归类到静态建模,但根据 Booch 归类,用例图属于动态建模部分。

(7) 在用例视图中,用例的设置代表了软件系统的功能划分。因此,将哪些动作组合为一个用例,将影响软件系统功能设置的合理性和方便性。

(8) 用例的实例是系统的一种实际使用方法,即系统的一次具体执行过程。例如,"张三在 ATM 机上插入银行卡并输入密码,在得到确认后,输入取款数字,系统收到信息后把人民币送出来。"这个场景可以认为是"取款"用例实例化的结果。

(9) 用例的动态执行过程可以用 UML 的交互作用来说明。可以用状态图、顺序图、协作图或非正式的文字描述来表示。

理论上可以把一个软件系统的所有用例都画出来,但实际开发过程中,进行用例分析时只需把那些重要的、交互过程复杂的用例找出来。

不要试图把所有的需求都以用例的方式表示出来。需求有两种基本形式:功能性需求和非功能性需求。用例描述的只是功能性方面的需求。那些用 UML 难以表示的需求很多是非功能性的需求。例如,开发项目中所涉及的术语表(Glossory)就很难用 UML 表示。对于这些需求往往是采用附加补充文档的形式来描述。

本质上,用例分析是一种功能分解(Functional Decomposition)的技术,并未使用到面向对象思想。因而有人认为用例分析只是面向对象分析与设计(Object-Oriented Analysis/Object-Oriented Design,OOA/OOD)的先导性工作,并非 OOA/OOD 过程的一部分,但也有人视其为 OOA/OOD 的一环。事实上,通过用例描述的参与者与系统一项功能的交互过程,从中可以发现类对象。不仅能够发现类对象的特征,也能发现类对象的行为。因此,无论怎样争论,用例是 UML 的一部分,确定一个系统的用例是开发 OO 系统的第一步。用例分析这步做得好,后续的类图分析、交互图分析等才有可能做得好,整个系统的开发才能顺利进行。

3.2.3 用例之间的关系

作为 UML 的结构事物,用例之间存在泛化(Generalization)关系、包含(Include)关系、扩展(Extend)关系。当然也可以利用 UML 的扩展机制自定义用例间的关系,如果要自定义用例间的关系,一般是利用 UML 中版型这种扩展机制。

1. 泛化关系

泛化(Generalization)代表一般与特殊的关系。在用例之间的泛化关系中，子用例继承了父用例的行为和含义，子用例也可以增加新的行为和含义或覆盖父用例中的行为和含义。父用例表示通用的行为序列。通过插入额外的步骤或定义步骤，子用例特化父用例。

在UML规范中，泛化关系用带空心三角形箭头的实线表示，箭头指向父用例。

【例3.7】 在一个在线股票经纪人系统中，图3.6给出了交易行为中部分交易类型之间的关系。“做交易”用例是父用例，它描述了所有交易类型都会执行的步骤。例如，输入交易密码。“交易债券”用例、“交易股票”用例、“交易期权”用例都是子用例，各自包含了针对某种交易类型特别安排的额外步骤。例如，“交易期权”用例包含了期权的行权日期判定。

【例3.8】 在一个学校信息管理系统中，图3.7给出查找人行为中部分特殊身份查找之间的关系。“查找人”用例是父用例，它描述了所有身份人查找都会执行的步骤。例如，输入查找人姓名。“查找教师”用例、“查找学生”用例、“查找教辅人员”用例都是子用例，各自包含了针对某种身份人进行查找而特别安排的额外步骤。例如，“查找学生”用例包含学生身份的判定。

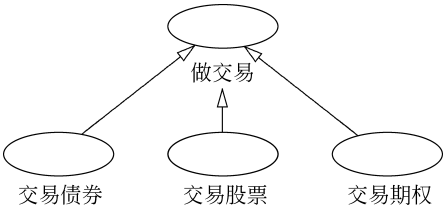

图3.6 交易用例之间的泛化关系

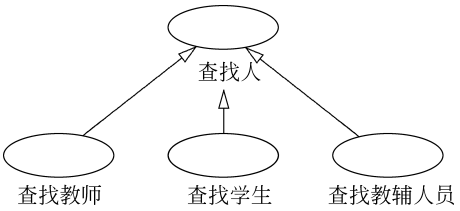

图3.7 查找人用例之间的泛化关系

例3.7给出的子用例包含了父用例没有的行为，反映的是子用例增加了新的行为。例3.8给出的子用例包含了父用例已有的行为，反映的是子用例更新了旧的行为。

当发现系统中有两个或者多个用例在行为、结构和目的方面存在共性时，就可以使用泛化关系。这时，可以用一个新的(通常也是抽象的)用例来描述这些共有部分；这个新的用例就是父用例。

2. 包含关系

包含(Include)关系指的是两个用例之间的关系，其中一个用例(称为基本用例)的行为包含了另一个用例(称为包含用例)的行为。

包含关系是依赖关系的版型，也就是说包含关系是比较特殊的依赖关系，它们比一般的依赖关系多了一些语义。

在UML规范中，包含关系用带箭头的虚线表示，箭头指向包含用例。同时，必须用<<include>>标记附加在虚线旁，作为特殊依赖关系的语义。

【例3.9】 在一个银行ATM系统中，有“存款”“取款”“账户余额查询”“转账”这4个用例，它们都要求储户必须先登录了ATM机。也就是说，它们都包含了储户登录系统的行为。因此，储户登录系统的行为是这些用例中相同的动作，可以将它提取出来，单独构成一个用例(作为包含用例)。图3.8给出了银行ATM系统中用例之间的包含关系。“存款”用例、“取款”用例、“账户余额查询”用例、“转账”用例都是基本用例，“登录”用例是包含用例。

由于将共同的储户登录系统的行为提取出来的，“存款”用例、“取款”用例、“账户余额查

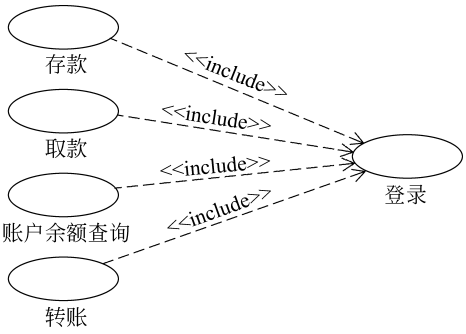

图 3.8 银行 ATM 系统中用例之间的包含关系

询”用例、“转账”用例都不再含有储户登录系统的行为。

【例 3.10】 在一个网上购物系统中,当注册会员在线购物时,网上购物系统需要对顾客的信用进行检查,检查输入的信用卡号是否有效,信用卡是否有足够的资金支付本次订购。可以将信用检查的行为提取出来,单独构成一个用例(作为包含用例)。图 3.9 给出了网上购物系统中用例之间的包含关系。“在线购物”用例是基本用例,“检查信用”用例是包含用例。

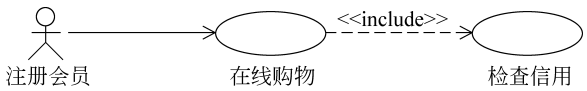

图 3.9 网上购物系统中用例之间的包含关系

在例 3.10 中,有没有必要将信用检查的行为提取出来,单独构成一个用例(作为包含用例),视具体情况而定。当信用检查的行为只发生在在线购物活动中时,可以不用提取出来。当信用检查的行为还发生在其他活动中时,应该提取出来,以便实现软件重用。

当发现系统中在两个或者多个用例中有某些相同的动作,则可以把这些相同的动作提取出来,单独构成一个用例(称为包含用例)。这样,当某个基本用例使用该包含用例时,就好像这个基本用例包含了包含用例的所有动作。具体地讲,基本用例在其内部说明的某一位置上显式地使用包含用例的行为的结果,包含用例作为包含它的基本用例的功能的一部分出现。基本用例仅仅依赖包含用例执行的结果,而不依赖包含用例的内部结构。

3. 扩展关系

扩展(Extend)关系的基本含义与包含关系类似,即一个用例(称为基本用例)的行为包含了另一个用例(称为扩展用例)的行为。但在扩展关系中,对于扩展用例(Extension Use Case)有更多的规则限制,即基本用例必须声明若干“扩展点”(Extension Point),而扩展用例只能在这些扩展点上增加新的行为和含义。

扩展关系也是依赖关系的版型,也就是说,扩展关系是特殊的依赖关系。

在 UML 规范中,扩展关系用带箭头的虚线表示,箭头指向基本用例。同时,必须用<<extend>>标记附加在虚线旁,作为特殊依赖关系的语义。

【例 3.11】 在一个图书借阅系统中,当读者还书时,如果借书时间超期,则需要交纳一定的滞纳金,作为罚款。图 3.10 给出了图书借阅系统中还书时用例之间的扩展关系。“还书”用例是基本用例,“罚款”用例是扩展用例。

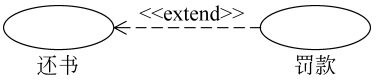

图 3.10　图书借阅系统中还书时用例之间的扩展关系

读者的借书时间超期时，会发生罚款动作。因此，在“还书”基本用例中指定了某种条件。当该条件为真时，将触发“罚款”扩展用例的执行，把扩展用例“罚款”的行为插入到由基本用例“还书”中的扩展点定义的位置。也就是说，“还书”用例可以单独存在，但是在一定的条件下，它的行为可以被另一个“罚款”用例的行为扩展。

扩展点是指用例中的一个或一组位置。在这样的位置上，可以插入其他用例的完整动作序列或其中的片段。在多数情况下，扩展关系有一个附加条件。只有当控制到达插入位置且条件为真时，才会发生扩展行为。在一个用例中，各扩展点的名字是唯一的。可以把扩展点列在用例图形符号中的一个题头为“扩展点”的分栏中，并以一种适当的方式（通常采用文本方式）给出扩展点的位置描述，作为基本用例中的标号。

【例 3.12】 在一个网上购物系统中，当注册会员下订单时，他可能需要临时查看一下商品目录。图 3.11 给出了带扩展点表示法的网上购物系统中下订单时用例之间的扩展关系。“下订单”用例是基本用例，“查看商品目录”用例是扩展用例。“附加请求”是用例“下订单”中的位置标号，“注册会员需要商品目录”是条件。

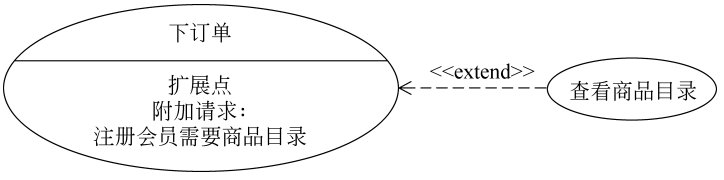

图 3.11　带扩展点表示法的用例之间的扩展关系

3.2.4　用例之间的泛化、包含、扩展关系的比较

一般来说，可以用 is a 和 has a 来判断使用哪种关系。泛化关系和扩展关系表示的是用例之间的 is a 关系，包含关系表示的是用例之间的 has a 关系。扩展关系和泛化关系相比，多了扩展点的概念，也就是说，一个扩展用例只能在基本用例的扩展点上进行扩展。

在扩展关系中，基本用例一定是一个 well formed 的用例，即是可以独立存在的用例。一个基本用例执行时，可以执行也可以不执行扩展部分。

在包含关系中，基本用例可能是也可能不是 well formed 的用例。在执行基本用例时，一定会执行包含用例（Inclusion Use Case）部分。

如果需要重复处理两个或多个用例时，可以考虑使用包含关系，实现一个基本用例对另一个用例的引用。

当处理正常行为的变形而且只是偶尔描述时，可以考虑只用泛化关系。

当处理正常行为的变形而且希望采用更多的控制方式时，可以在基本用例中设置扩展点，使用扩展关系。

扩展关系是 UML 中较难理解的一个概念，如果把扩展关系看作带有更多规则限制的泛化关系，则可以帮助理解。事实上，在 UML Specification l.1 版本以前，扩展关系就是用

泛化关系的版型表示的，在1.3版本后，扩展关系改为用依赖关系的版型表示。

【例3.13】 在一个网上购物系统中，当注册会员浏览网站时，他可能临时决定购买商品。当他决定购买商品后，就必须将商品放进购物车，然后下订单。图3.12给出了描述这种功能需求的用例图，其中既有扩展关系，又有包含关系。

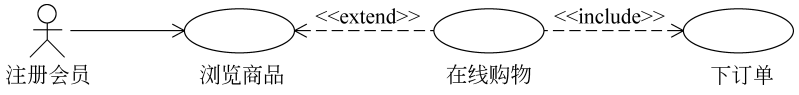

图3.12 客户网站购物的用例图(一)

如果网上购物系统的需求是这样的：注册会员既可以直接在线购物，又可以浏览商品后临时决定在线购物，则可以将图3.9和图3.12合二为一，得到图3.13所示的用例图。

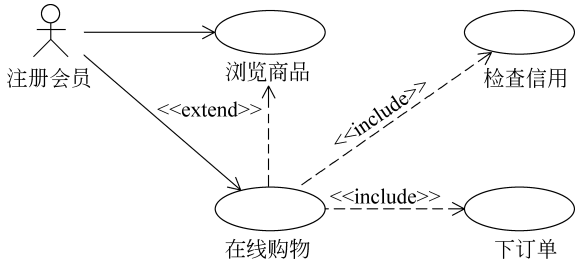

图3.13 客户网站购物的用例图(二)

通过例3.10和例3.13的文字描述以及图3.13的图形描述，可以清晰地反映出包含关系和扩展关系的异同。

相同点：它们都是基本用例行为的一部分。换句话说，最初的基本用例中的部分行为被提取出来，单独形成了一个用例。

不同点：在基本用例的每一次执行时，包含用例都一定会被执行，而扩展用例只是偶尔被执行。

总结一下UML中关系(Relationship)、关联(Association)、泛化(Generalization)、依赖(Dependency)这几个概念之间的区别和联系。

关系是模型元素之间具体的语义联系。关系可以分为关联、泛化、依赖、实现等几种。

关联是两个或多个类元(Classifier)之间的关系，它描述了类元的实例之间的联系。这里所说的类元是一种建模元素，常见的类元包括类(Class)、参与者(Actor)、组件(Component)、数据类型(Data Type)、接口(Interface)、结点(Node)、信号(Signal)、子系统(Subsystem)、用例(Use Case)等，其中类是最常见的类元。

泛化关系表示的是两个类元之间的关系。这两个类元中，一个相对通用，另一个相对特殊。相对特殊的类元的实例可以出现在相对通用的类元的实例能出现的任何地方。也就是说，相对特殊的类元在结构和行为上与相对通用的类元是一致的，但相对特殊的类元包含更多的信息。

依赖关系表示的是两个元素或元素集之间的一种关系，被依赖的元素称为目标元素，依赖元素称为源元素。当目标元素改变时，源元素也要做相应的改变。包含关系和扩展关系都属于依赖关系。

3.2.5 用例的实现

用例是与实现无关（Implementation-Independent）的关于系统功能的描述。在UML的用例图中，用例之间不存在实现关系。但是，为了表达描述契约用例的实现方式，UML规范中定义用协作（Collaboration）来说明对用例的实现，即用例与其协作之间存在实现关系。

协作是对由共同工作的类、接口和别的元素所组成的群体的命名，这组群体提供合作的行为。其实协作也可以用来说明操作的实现，但用得不多。除非是比较复杂的操作，大多数情况下，可以直接用活动图或代码说明操作的实现。

在UML规范中，用实线椭圆表示用例，用虚线椭圆表示协作，实现关系用带空心三角形箭头的虚线表示，箭头指向用例。

图3.14给出了用例与其协作之间的实现关系。Login用例共有两个实现：一个是简单的实现；另一个是带有安全验证功能的实现。

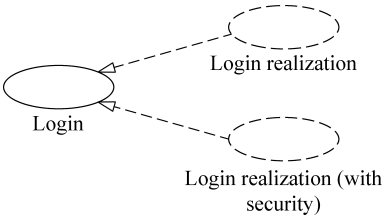

图3.14 用例及其实现

这里没有显示协作的内部结构和行为方面的内容。协作的内部由两部分组成：一部分是结构部分，如类、接口以及其他一些建模元素等；另一部分是说明类、接口以及其他建模元素如何协调工作的行为部分，如协作图（Collaboration Diagram）、顺序图（Sequence Diagram）、类图（Class Diagram）等。

在大多数情况下，一个用例由一个协作实现，这时可以不用在用例模型中显式指明这种实现关系。

3.3 用例描述

用例图（Use Case Diagram）是显示一组用例、参与者以及它们之间关系的图。这些关系可以包括用例之间的关系（泛化、包含、扩展）、参与者之间的关系（泛化）、参与者和用例之间的关系（关联）。

在用例图中，一个用例是用一个命名的椭圆表示的，但如果没有对这个用例的具体说明，那么还是不清楚该用例到底会完成什么功能。没有描述的用例就像是一本书的目录，人们只知道该目录标题，但并不知道该目录的具体内容是什么。所以说，仅用图形符号表示的用例本身并不能提供该用例所具备的全部信息，必须通过文本的方式描述该用例的完整功能。事实上，用例的描述才是用例的主要部分，是后续的交互图分析和类图分析必不可少的部分。

用例描述（Use Case Specification）实际上是一个关于参与者与系统如何交互的规范说明，该规范说明要清晰明了，没有歧义性。

由于用例描述了参与者和软件系统进行交互时，系统所执行的一系列的动作序列。因此，这些动作序列不但应包含正常使用的各种动作序列（称为主事件流），而且还应包含对非正常使用时软件系统的动作序列（称为子事件流）。所以，主事件流描述和子事件流描述是

用例描述的主要内容。需要注意的是,在表述用例描述时,仍然注重描述系统从外部看到的行为。用例描述的内容还应包括用例激活前的前置条件,说明如何启动用例,以及执行结束后的后置结果,说明在什么情况下用例才被认为是完成的。此外,在用例描述中除了表明主要步骤与顺序外,还应表明分支事件和异常情况。

一个比较完整的用例描述,可以包括表 3.1 所示的内容。

表 3.1 用例描述的内容

描 述 项	说 明
用例名称	动宾结构或主谓结构命名的用例名字
标识符[可选]	用例的唯一标识符,在文档的别处可以用标识符来引用该用例
用例描述	对用例作用和目的的简要描述,包括用例的最终任务是什么,想得到什么样的结果
参与者	与此用例相关联的参与者列表
优先级	说明对用例进行分析、设计、实现的紧迫程度
状态[可选]	用例的状态,通常为以下几种之一:进行中、等待审查、通过审查或未通过审查
前置条件	一个条件列表,这些条件必须在访问用例之前得到满足,包括哪个参与者或用例在怎样的情况下启动执行用例
后置条件	一个条件列表,这些条件将在用例完成以后得到满足,包括明确指出在什么情况下用例才能被看作完成,完成时要把什么结果值传递给参与者或系统
基本操作流程	描述用例中各项工作都正常进行时用例的工作方式,包括正常使用时各种动作序列,参与者和用例之间传递哪些消息,使用或修改了哪些实体
可选操作流程	描述变更工作方式、出现异常或发生错误的情况下所遵循的路径
特殊需求	描述用例的非功能性需求和设计约束
被泛化的用例	描述此用例所泛化的用例列表,即父用例列表,而此用例作为子用例
被包含的用例	描述此用例所包含的用例列表,即包含用例列表,而此用例作为基本用例
被扩展的用例	描述此用例所扩展的用例列表,即扩展用例列表,而此用例作为基本用例
修改历史记录[可选]	关于用例的修改时间、修改原因和修改人的详细信息
问题[可选]	与此用例开发相关的问题列表
决策[可选]	关键决策的列表,将这些决策记录下来以便维护时使用
频率[可选]	参与者访问此用例的频率

【例 3.14】 在银行 ATM 系统的 ATM 机上“登录”用例的一个简单的用例描述可采用下面的形式。

用例名称:登录。

用例描述:在储户银行卡有效且储户输入密码正确的情况下,为储户提供后续的服务。

参与者:储户。

前置条件:ATM 机正常工作。

后置条件:正常进入主界面。

主事件流如下。

（1）储户将银行卡插入 ATM 机，开始用例。

（2）ATM 机提示储户输入密码。

（3）储户输入密码。

（4）ATM 机确认密码是否有效。如果无效，则执行子事件流 a。如果与主机连接有问题，则执行异常事件流 e。

（5）如果输入的密码正确，进入主界面。ATM 机提供以下选项：存款、取款、查询、转账，用例结束。

子事件流 a 如下。

a1：提示储户输入的密码无效，请求再次输入。

a2：如果三次输入无效密码，则系统自动关闭，退出储户银行卡，用例结束。

异常事件流 e 如下。

e1：提示储户主机连接不上。

e2：系统自动关闭，退出储户银行卡，用例结束。

【例 3.15】 在银行 ATM 系统的 ATM 机上“取款”用例的一个简单的用例描述可采用下面的形式。

用例名称：取款。

用例描述：在储户账户有足够金额的情况下，为储户提供现金，并从储户账户中减去所取金额。

参与者：储户。

前置条件：储户正确登录系统。

后置条件：储户账户余额被调整。

主事件流如下。

（1）储户在主界面选择“取款”选项，开始用例。

（2）ATM 机提示储户输入欲取金额。

（3）储户输入欲取金额。

（4）ATM 机确认该储户账户是否有足够的金额。如果余额不够，则执行子事件流 b。如果与主机连接有问题，则执行异常事件流 e。

（5）ATM 机从储户账户中减去所取金额。

（6）ATM 机向储户提供要取的现金。

（7）ATM 机打印取款凭证。

（8）进入主界面。ATM 机提供以下选项：存款、取款、查询、转账，用例结束。

子事件流 b 如下。

b1：提示储户余额不够。

b2：返回主界面，等待储户重新选择选项。

异常事件流 e 如下。

e1：提示储户主机连接不上。

e2：系统自动关闭，退出储户银行卡，用例结束。

【例 3.16】 大学图书管理系统的“借书”用例的一个简单的用例描述可采用下面的形式。

用例名称：借书。

用例描述：在读者可以借书的情况下，为读者登记借书记录，并修改该读者可借的书本数。

参与者：图书管理员。

前置条件：图书管理员已经登录系统。

后置条件：生成借书记录，修改可借的书本数。

主事件流如下。

(1) 读者在柜台将欲借书籍交给图书管理员。

(2) 图书管理员执行借书操作，开始用例。

(3) 图书管理员输入读者借书证号。

(4) 系统确认该借书证是否还可以借书。如果已借书的本数达到可借上限，则执行子事件流 a。如果与数据库连接有问题，则执行异常事件流 e。

(5) 图书管理员输入图书流水号。

(6) 系统取当日日期作为借书日期，自动生成借书日期。系统创建新的借书记录，内容包括借书证号、图书流水号、借书日期。如果与数据库连接有问题，则执行异常事件流 e。

(7) 系统退出借书操作，进入主界面，用例结束。

子事件流 a 如下。

a1：提示读者借书证已借书的本数达到可借上限。

a2：系统退出借书操作，进入主界面，用例结束。

异常事件流 e 如下。

e1：提示读者数据库连接不上。

e2：系统自动关闭，用例结束。

一个复杂用例主要体现在基本操作流程和可选操作流程的步骤和分支过多。此时，也可以采用“场景(或称脚本)”的技术来描述用例，而不是试图使用大量的分支和附属流来描述用例。

场景是指通过用例的一个特定路径，它能够通过用例的事件流(基本操作流程和可选操作流程)。场景的一个重要特征是它们无分支。因此，用例事件流中的每个可能的分支潜在产生一个独立场景。

用例事件流中的基本操作流程的主要路径构成主要场景，其他路径构成次要场景。一个用例仅有一个主要场景。

3.4 用例建模

用例模型主要应用在需求分析时使用。在用例模型中，人们把系统看成是实现各种用例的“黑盒子”，只关心该系统实现了哪些功能，并不关心内部的具体实现细节(例如，用例是由哪些类实现的，这些类有哪些属性和方法，方法的程序流程是怎样的，类之间有哪些通信)。建立用例模型，使开发人员在头脑中明确需要开发的系统的功能有哪些，同时方便用户和系统分析师进行沟通。

3.4.1 用例建模的步骤

用例建模是直接面向用户的，主要以需求陈述为基本依据，确定有关系统的边界、参与者、用例、通信关系等。

用例建模的基本步骤如下。

（1）找出系统外部的参与者和外部系统，确定系统的边界和范围。

（2）确定每一个参与者所期望的系统行为，即参与者对系统的基本业务需求。

（3）把这些系统行为作为基本用例。

（4）区分用例的优先次序。

（5）细化每个用例。使用泛化、包含、扩展等关系处理系统行为的公共或变更部分。

（6）编写每个用例的用例描述。

（7）绘制用例图。

（8）编写项目词汇表。

3.4.2 确定系统的边界

系统边界是指系统与系统之间的界限。系统可以认为是由一系列的相互作用的元素形成的具有特定功能的有机整体。不属于这个有机整体的部分可以认为是外部系统。因此，系统边界定义了由谁或什么（即参与者）来使用系统，系统能够为参与者提供什么特定服务。确定系统边界就是要定义好什么是系统的组成部分（边界内）和什么是系统的外部（边界外）。

用例图中的系统边界用来表示正在建模系统的边界。边界内表示系统的组成部分，边界外表示系统外部。在用例图中，系统边界用实线方框图形符号表示，同时附上系统的名称作为标签。用例绘制在方框里面（即边界里面），参与者绘制在方框外面（即边界外面），如图3.15所示。

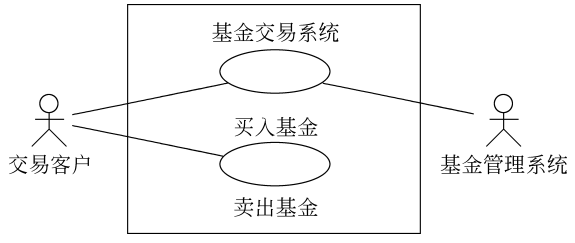

图3.15 仅完成基金买卖的“基金交易系统”的系统边界

通常从系统边界实际在何处的试探性分析开始用例建模。随着找出的参与者和用例，系统边界会变得越来越确定。

【例3.17】 在一个仅为交易客户提供买卖基金的基金交易系统中，它的参与者就是进行基金买卖的交易客户。交易客户能够操作的（看到的）系统功能有买入基金和卖出基金。因此，该系统中只有两个用例：买入基金、卖出基金。进一步分析发现，基金的品种应该存在于该系统中，否则交易客户无法进行基金的买卖。但是该系统中已发现的两个用例都不能完成基金品种的管理。因此，可以确定基金品种的录入应该在别的系统中完成。不妨把

完成基金品种录入的系统称为基金管理系统。显然,基金管理系统是另外一个参与者。由此确定的系统边界如图 3.15 所示。在这种情况下,需要开发的软件系统仅包含两个用例:买入基金、卖出基金。

【例 3.18】 在一个既提供基金买卖又提供基金品种录入的基金交易系统中,它的参与者之一还是交易客户,他能够操作的(看到的)系统功能有买入基金和卖出基金。另外,该系统中还应该包括基金品种管理(录入、修改、删除、查询)的功能,而这个功能应该是由基金公司员工操作的。所以,存在第 2 个参与者是基金公司员工。此时,例 3.17 中负责基金品种管理的基金管理系统不再是一个参与者了,它的功能要在基金交易系统中实现。由此确定的系统边界如图 3.16 所示。在这种情况下,需要开发的软件系统包含 3 个用例:买入基金、卖出基金、管理基金品种。

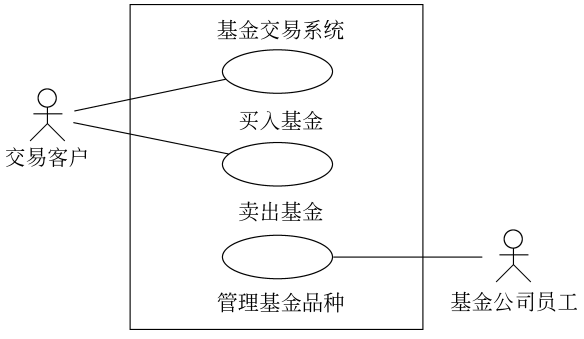

图 3.16 扩展了基金品种管理的"基金交易系统"的系统边界

系统边界决定了参与者。如果系统边界规划得不一样,系统的参与者就会发生很大的变化。相应地,系统的用例也会发生很大的变化。因此,只有弄清楚了系统边界,才能更好地确定系统的参与者和用例,才能保证后续工作的正确性。

3.4.3 确定参与者

为了识别参与者,需要考虑谁在使用系统,他们在与系统的交互中扮演什么角色。考虑特定的人和事物,能够找出与系统交互时人和事物所扮演的角色。然后,进行归纳,就可以得到待开发系统应当具有的参与者。

以下问题可以帮助识别参与者。

(1) 谁将使用系统的主要功能?

(2) 谁将需要系统的支持来完成他们的日常工作?

(3) 谁必须维护、管理和确保系统正常工作?

(4) 谁将给系统提供信息、使用信息和维护信息?

(5) 系统需要处理哪些硬件设备?

(6) 系统使用外部资源吗?

(7) 系统需要与其他系统交互吗?

(8) 谁对系统产生的结果感兴趣?

在确定参与者时,要注意以下问题。

(1) 参与者对于系统而言总是外部的。

（2）参与者直接同系统交互。

（3）参与者表示人或事物同系统发生交互时所扮演的角色，而不是特定的人或事物。

（4）一个人或事物在与系统发生关系时，同时或不同时扮演多种角色。

（5）一个参与者可以包含多个不同的具体用户。

3.4.4 确定用例

识别用例的最好方法是从参与者列表开始，然后考虑每个参与者如何使用系统，需要系统提供什么样的服务。使用这个策略，能够获得一组候选用例。当识别用例时，也可能找出一些新的参与者，这是好事。此外，还要弄清楚系统的问题域、业务流程、系统的功能需求。

以下问题可以帮助识别用例。

（1）参与者要向系统请求什么功能？

（2）每个参与者的特定任务是什么？

（3）参与者需要读取、创建、撤销、修改或存储系统的某些信息吗？

（4）是否任何一个参与者都要向系统通知有关突发性的、外部的改变？或者必须通知参与者关于系统中发生的事件？

（5）这些事件代表了哪些功能？

（6）系统需要哪些输入输出？

（7）是否所有的功能需求都被用例使用了？

在确定用例时，要注意以下问题。

（1）每个用例至少应该涉及一个参与者。

（2）如果存在不与参与者进行交互的用例，则应该检查是否遗漏了该用例的参与者。如果确实没有与参与者进行交互，则可考虑将其并入其他用例中。

（3）每个参与者也必须至少涉及一个用例。

（4）如果存在不与用例进行交互的参与者，则应该考虑该参与者是如何与系统发生联系的，或者由参与者确定一个新的用例，或者该参与者是一个多余的模型元素。

3.4.5 区分用例的优先次序

识别出基本用例后，需要确定它们的优先次序，以便弄清楚哪些用例是最关键的，哪些用例是最艰巨的，哪些用例是最复杂的，哪些用例是必须在其他用例之前完成的，从而对用例做出高层与低层、主要与次要、基本与详细的区分；便于规划好后续的系统功能的分析与设计阶段的任务展开的先后次序，确定出哪个用例可以为其他用例所重用，从长远的角度节省时间。

3.4.6 细化每个用例

分析基本用例要完成的功能，将基本用例中具有一定独立性的功能，特别是具有公共行为特征的功能分解出来，将其作为包含用例，并供基本用例使用。分析基本用例功能以外的其他功能，将其作为扩展用例，并供基本用例进行功能扩展。

对用例进行细化，有助于弄清楚用例的行为中哪些是手工操作的，哪些是可以编程实现的。软件系统的开发落实在可以编程实现的功能上（即用例的行为）。

细化用例本质上就是对原有用例进行分解，其结果会导致用例个数的增加和每个用例的动作步骤的简化。但是，不是用例个数越多、每个用例的动作步骤越简单越好。因为过多的用例个数会增加用例间的通信关系，从而增加系统的耦合复杂性和通信开支。

3.4.7 编写每个用例的用例描述

确定了系统需要多少用例后，需要对每个用例编写其用例描述，详细说明参与者和用例进行交互时，用例所执行的一系列的动作序列。其中包括主事件流，子事件流，前置条件，后置条件，异常情况处理，所泛化、包含、扩展的用例等。

3.4.8 绘制用例图

使用 OOA 相关的绘图工具绘制用例图。一个用例模型由若干个用例图构成。

用例图的主要作用是描述参与者和用例之间的关系。简单的系统中只需要有一个用例图就可以把所有的关系都描述清楚。复杂的系统中可以有多个用例图。

3.4.9 编写项目词汇表

项目词汇表是最重要的项目制品之一。每个业务领域都有它本身独一无二的语言，需求工程和分析的主要目的是理解和捕获那种语言。词汇表提供了主要业务术语和定义的字典。它应该被项目中的所有人员理解，包括所有的利益相关人。

除了定义关键术语之外，项目词汇表必须解决同音异义词和同义异音词的问题。

1. 同音异义词

同音异义词是指相同的单词对不同的人表示不同的事物。各方其实是在说不同的语言，而他们都相信他们正在说同一种语言，这就产生了困难的通信问题。解决这个问题的方法是为术语挑选一种意思，并且可能为其他同音异义词引入新术语。

2. 同义异音词

同义异音词是指相同事物的不同的词。作为面向对象分析师，必须挑选其中之一（看起来使用最广泛的词）并且坚持使用它。其他变体必须完全从用例模型中剔除掉。这是因为，如果允许使用同义异音词，结果会产生或多或少做相同事情但是具有不同的名称的两个类。并且，如果允许在特定基础上使用同义异音词，那么能够发现，这些词将随着时间推移而逐渐发生分歧。

在项目词汇表中，应该记录推荐术语并且列出处在定义之中的同义异音词。这可能涉及鼓励一些业务利益相关人习惯不同的术语。通常，使利益相关人改变他们的语言用法是艰巨的任务，但要有毅力，坚持就会成功。

UML 没有为项目词汇表设置任何标准。使它尽可能简单和简要是很好的实践。采用像字典字母表那样的形式，对单词和定义进行排序。简单的基于文本的文档就可以了，但是大的项目可能需要基于在线 HTML 或 XML 的词汇表或者是一个简单的数据库。

表 3.2 给出了网上购物系统的一个项目词汇表的部分示例。作为一种风格，如果没有同义异音词或者同音异义词，就写上 None，而不是忽略它们或者保持空白。这表明已经考虑过这个问题。

表 3.2　网上购物系统的项目词汇表

术　　语	定　　义
商品目录	Clear View Training 当前提供出售的所有商品的列表 同音异义词：None 同义异音词：None
检出（Checkout）	超市（顾客为购物车中的商品付款的地方）中真实世界检出的电子模拟 同音异义词：None 同义异音词：None
Clear View Training	致力于销售书籍和 CD 的有限公司 同音异义词：None 同义异音词：CVT
信用卡（Credit Card）	诸如 VISA 或者 MasterCard，用于支付商品的金融交易卡 同音异义词：None 同义异音词：卡（Card）
顾客	从 Clear View Training 购买商品或服务的当事人（指自然人或法人） 同音异义词：None 同义异音词：None

与项目词汇表相关的一个问题是：项目词汇表中的术语和定义也可以用在 UML 模型中，但要确保这两种文档保持同步。

3.5　用例建模中常见的问题分析

用例模型是捕获需求的有力工具，但不恰当地使用用例模型也会引起许多问题。

3.5.1　用例的设计原则

为了避免用例的使用误区，应该遵循以下基本原则。

1. 需求和用例的关系

用例能够表达需求，因此不必将用例转换成其他形式。如果编写恰当，用例可以准确地对系统必须要做什么进行详细描述。但用例不详细描述外部接口、数据格式、业务规则和复杂公式。用例只是需要收集所有需求中的一部分，虽然这部分是非常重要的，但毕竟只是一部分。

2. 需求应该有层次地组织起来

对于复杂的需求采用层次分解法是有效的、简洁的方法。为了便于理解需求，应将需求层次化。系统的高层需求一般用不超过 12 个用例表示出来。在接下来的层次中，用例的数量不应超过当前用例的 5～10 倍。过多的用例对于开发人员、用户没有任何帮助，而且会影响人们对需求的正确理解。用例最佳的开发方式也是迭代式和递增式。在不同的开发阶段，可将用例工具用来描述不同层次的问题。例如，可以将用例划分为业务用例、系统用例、组件用例等。

3. 不要从用例直接推论出设计

如果从用例直接推论出设计，“用例开发”仅仅成为功能分解的一个借口。用例应该描

述参与者使用系统所遵循的顺序，但用例绝不说明系统内部采用什么步骤来响应参与者的刺激。用例终止于系统接口的边界。软件系统架构的确定和创建通常不仅仅根据功能需求，而且要考虑标准化和系统的时间、空间、可靠性和可分布性等方面的要求。

3.5.2 用例模型的复杂度

一般小型的系统，其用例模型中包含的参与者和用例不会太多，一个用例图就可以容纳所有的参与者，所有的参与者和用例也可以并存于同一个层次结构中。对于较复杂的大中型系统，用例模型中的参与者和用例会大大增加，人们需要一些方法来有效地管理由于规模上升而造成的复杂度。

1. 用例包

包是 UML 中最常用的管理模型复杂度的机制，包也是 UML 中语义最简单的一种模型元素，它就是一种容器，在包中可以容纳其他任意的模型元素（包括其他的包）。在用例模型中，可以用构造型<<Use Case>>来扩展标准 UML 包的语义，这种新的包称为用例包（Use Case Package），用于分类管理用例模型中的模型元素。

可以根据参与者和用例的特性来对它们进行分类，并将它们分别置于不同的用例包管理之下。例如，对于一个大型的企业管理信息系统，就可以根据参与者和用例的内容将它们分别归于人力资源、财务、采购、销售、客户服务等用例包之下。这样就将整个用例模型划分为两个层次：在第一个层次将系统分为五部分；在第二个层次可分别表示每一用例包内部的参与者和用例。

一个用例模型需要有多少个用例包取决于要如何管理用例模型的复杂度（包括参与者和用例的个数，以及它们之间的相互关系）。UML 中的包类似于文件系统中的目录，文件数量少的时候不需要额外的目录，文件数量一多就需要有多个目录来分类管理。对于同样一组文件，不同的人会创建不同的目录结构来进行管理，关键是要保证在目录结构下每一个文件都要易于访问。同样的道理存在于用例建模之中，如何创建用例包及用例包的个数取决于不同的系统和系统开发人员，但要保证整个用例模型易于理解。

2. 用例的粒度

系统需要有多少个用例？这是很多人在用例建模时会产生的疑惑。描述同一个系统，不同的人会产生不同的用例模型。

例如，对于各种系统中常见的“维护用户”用例，它里面包含了添加用户、修改用户、删除用户等操作，这些操作在该用例的事件流中可以表述成为基本流的子事件流。“维护用户”用例的主事件流由 3 个子事件流构成：添加用户子事件流；修改用户子事件流；删除用户子事件流。在这种情况下，只有一个用例。不过，其动作序列描述较为冗长。

也可以根据“维护用户”用例中的具体操作把它抽象成 3 个用例：“添加用户”用例、“修改用户”用例和“删除用户”用例。它所表示的系统需求和单个用例的模型是完全一样的。在这种情况下，共有 3 个用例。每个用例的动作序列较为简洁。

如何确定用例的粒度呢？最好将用例模型的规模控制在几十个用例左右，这样比较容易管理用例模型的复杂度。Ivar Jacobson 认为对于一个 10 人年的项目，他需要大约 20 个用例。而 Martin Fowler 认为他需要大约 100 个用例。

在用例个数大致确定的条件下，人们很容易确定用例粒度的大小。对于较复杂的系统，

需要控制用例模型这一级的复杂度，所以可以将复杂度适当地移往每一个用例的内部，也就是让一个用例包含较多的需求信息量。对于比较简单的系统，则可以将复杂度适度地暴露在用例模型这一级。也就是说，可以将较复杂的用例分解成为多个用例。

用例的粒度不但决定了用例模型级的复杂度，而且也决定了每一个用例内部的复杂度。系统分析师和开发人员应该根据每个系统的具体情况来把握各个层次的复杂度，在尽可能保证整个用例模型的易理解性的前提下决定用例的大小和数目。

3. 用例图

用例图的主要作用是描述参与者和用例之间的通信关系，简单的系统中只需要有一个用例图就可以把所有的关系都描述清楚。复杂的系统中可以有多个用例图，例如，每个用例包都可以有一个独立的用例图来描述该用例包中所有参与者和用例的关系。

在一个用例模型中，如果参与者和用例之间存在着多对多的关系，并且其间的关系比较复杂，那么在同一个用例图中表述所有的参与者和用例就显得不够清晰，这时可创建多个用例图来分别表示各种关系。

如果想要强调某一个参与者和多个用例的关系，就可以以该参与者为中心，用一个用例图表述出该参与者和多个用例之间的关系。在这类用例图中，强调的是该参与者会使用系统所提供的哪些服务。

如果想要强调某一个用例和多个参与者之间的关系，就可以以该用例为中心，用一个用例图表述出该用例和多个参与者之间的关系。在这类用例图中，强调的是该用例会涉及哪些参与者，或者说该用例所表示的系统服务有哪些使用者。

3.5.3 用例模型的调整

用例模型建成之后，需要对用例模型进行调整，看是否可以进一步简化用例模型、提高重用程度、增加模型的可维护性。主要可以从以下检查点入手。

（1）用例之间是否相互独立？如果两个用例总是以同样的顺序被激活，可能需要将它们合并为一个用例。

（2）多个用例之间是否有非常相似的行为或事件流？如果有，可以考虑将它们合并为一个用例。

（3）用例事件流的一部分是否已被构建为另一个用例？如果是，可以让该用例包含另一用例。

（4）是否应该将一个用例的事件流插入另一个用例的事件流中？如果是，利用与另一个用例的扩展关系来建立此模型。

3.5.4 用例模型的检查

用例模型完成之后，还需要对用例模型进行检查，看一下是否有遗漏或错误之处。主要可以从以下几个方面来进行检查。

1. 功能需求的完备性

现有的用例模型是否完整地描述了系统功能，这也是系统分析师和开发人员判断用例建模工作是否结束的标志。如果发现还有系统功能没有被记录在现有的用例模型中，那么就需要抽象一些新的用例来记录这些需求，或是将它们归纳在一些现有的用例之中。