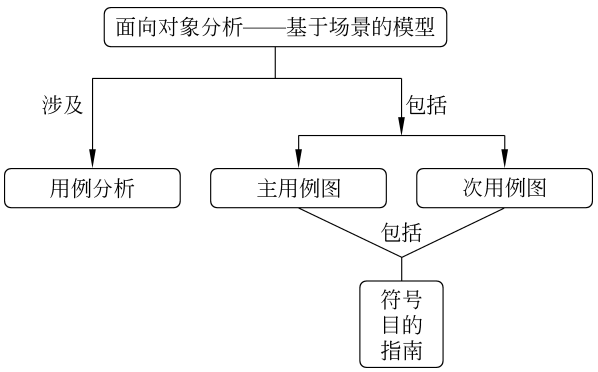
第 5 章　面向对象分析——基于场景的模型

概念图



学习目标

在学完本章之后，读者将能够：

- 理解一个好的分析模型所具有的特点。
- 了解 UML 用例图在面向对象分析中的作用。
- 理解用例图的作用和重要性。
- 理解绘制用例图所用的元素和符号。
- 识别一个给定应用程序域的参与者、用例、泛化及关联关系。
- 开发一个用例模型来刻画系统的功能需求。

通过 UML 对问题域的基于面向对象的理解进行图解表示，以突出的方式鼓励和支持分析阶段。UML 有许多图表来反映模型的所有方面，因此提供了对问题的共同且唯一正确的理解。以下部分描述了帮助理解问题域的两个最基本的图，即用例图和类图。

5.1　用 例 分 析

5.1.1　用例

A 先生最近买了一部手机。在购买时，他见到了各种各样的型号。这导致他对于选择哪一款有很大的困惑。他问自己他想用手机做什么。他是否只用于拨打电话，还是使用它来进行频繁通信？他是否对 QWERTY 模式感到满意，是否可以操控现有的设置？他想要

数码相机设备吗？如果是这样，他可以解决哪方面问题？他会粗暴地还是极其小心地使用手机？对于后一种情况，他能否负担得起触摸屏模式？他是否喜欢在手机上使用互联网，尤其是访问社交网站？他是否喜欢使用音乐播放器？他是否有兴趣在手机上收听 FM 广播？在这种情况下，他是否想用一套内置天线，或者是一个需要外部天线的设备来收听 FM 广播？他是否喜欢基于多媒体流的应用程序？他主要是对手机银行感兴趣，还是喜欢使用任何基于 GPRS 的应用程序？因此他不断地问自己这些问题，希望根据这些问题的答案来购买手机，让自己能够满意。

因此，当我们并非一时冲动而进行购物时，都会经历类似的过程。这就是所谓的“用例分析”。我们提出许多问题去了解产品或系统的使用方式，最后我们只会花钱购买那些完全符合我们需求的产品。这里的主要动机是收集和了解这些需求。

这种过程在分析阶段非常关键，因为这可以指导系统的设计和开发。“用例”被定义为帮助分析人员与用户一起确定系统使用情况的元素。相关用例的集合是根据用户想要完成的内容来完整地描绘整个系统的性质和功能。

用例可以被理解为有关系统使用场景的集合。每个场景都描述一个事件序列。每个序列由人/其他系统/硬件组件发起，或者仅仅在某个时间点到达之后。这些元素中的每一个都被命名，它们的语义在 UML 中被明确规定。以下各节将详细介绍这些内容。

5.1.2　用例的重要性

用例是激励潜在用户表达他们打算如何使用系统的一个很好的工具。可以看出，传统的系统开发过程没有提供一个很好的激励平台，大多数时候用户只有在被分析人员询问或采访时才会表达自己所需要的输入。

这里的关键原则是用户要参与系统分析和设计的过程，尤其是早期阶段。这大大提高了系统能够被正确构建以满足用户需求的可能性，而不会使用户无法理解和使用系统。

5.2　主 用 例 图

主用例图是系统的功能和需求以及连接系统外部接口的图形概述，显示了参与者及其与用例之间的关系，表明了设计的特点。此外，主用例图是使用 UML 来设计新系统的第一步，并在分析、实现和文档化阶段解释了系统的需求。它还引出了系统预期完成的整体功能。

1. 场景

首先我们将了解场景的含义，场景是用例的一个子集，用于清楚地了解该用例。场景是表示行为的一系列动作。基本上，使用场景来说明在系统中发生的交互行为或仅仅是用例实例的执行。描述场景的主要动机是为了在基于场景的需求启发中使用，这是一个提出与描述性故事有关的问题以确定设计需求的过程。例如，考虑以下“孩子手机监控软件”中的场景：

孩子被发现在他/她规定的学习时间内访问 Facebook——一个社交网站，且超过了两个小时。此时，会有一个警告提示发送给父母的手机账户，然后父母通过在学习时间内锁定

孩子的手机来限制其对 Facebook 的访问。

上述场景具体描述了当孩子通过浏览社交网站，浪费了他/她的学习时间时会发生的情况。

使用基于场景的需求启发，需要询问涉及的利益相关者希望系统完成的各种任务。他们会被问及想象中使用的系统是什么样子。然后这些关于系统的问题陈述会被映射到相应的系统规范中，该规范被表示为一组参与者和用例，这些内容将在以下章节中讨论。业务分析人员团队会与客户一起列举一整套可能的场景，这些场景以简单的自然语言（而不是使用任何正式的符号）记录在用户术语中。事实上，系统中预期执行的每一个任务都应该在这套场景中给出。场景对于启发、验证和记录需求非常有用。基于场景的方法有助于将用户/利益相关者的视图与未来系统的功能视图进行连接和映射，以便正在开发的预期系统可以满足其用户的预期要求。因此，基于场景的方法在行业内被大量地使用。

一般情况下，几个相关的场景会一同出现在一个用例中。例如，考虑以下两种场景：

孩子被发现在手机上使用不合适的应用程序。这个使用操作将会被记录下来，并向父母的手机发送警告消息。然后父母会查看该应用程序的性质，发现这个应用程序的确影响不好，会使孩子沉迷于它，并且污染孩子的心灵。所以父母阻止了该应用程序。现在，孩子就永远不能在手机上使用这个应用程序了。

孩子被发现在手机上使用不合适的应用程序。这个使用操作将会被记录下来，并向父母的手机发送警告消息。然后父母会查看该应用程序的性质，发现其非常有趣，唯一的负面因素是它一直在占用孩子的时间。所以父母通过锁定来管理该应用程序。在闲暇时间，孩子才可以访问并使用它。

这两种场景都有一个共同的用户目标，就是控制手机中不需要的应用程序。第一种场景是最简单的，应用程序是有害的，因此它被阻止。第二种场景多了一些行为——如果孩子真的想要使用这个应用程序，并且它本性不坏，只是会消耗时间和精力。当然，这两个相关的可选择的场景被构造在同一个用例中（如图 5.5 所示）。

2. 用例事件流

用例图有助于简化系统的可视化。不过仍然需要用文本描述用例的事务顺序，以便更好地了解用例中真正发生的情况。在这部分中，将介绍用例事件流，其描述了系统的行为。事件流是基于系统应该做什么的，而不是系统如何做。

几个不同的模板可用于记录用例事件流。这些模板的标准结构可能在版本与版本之间略有不同。其实质是事件流应该表达任何用例的过程。模板样例如图 5.1 所示。

X<名称>用例的事件流
X.1前置条件。在该用例开始之前，(在另一个用例中)需要发生哪些事情?系统在该用例之前处于什么状态?
X.2主流。主流是一系列声明的步骤。
X.3子流。一部分子流是主流划分后的模块，另一些子流提高了文档的可读性。
X.4替代流。替代流定义了可以中断正常流的异常行为。通常替代流表示在错误条件下要做的事情。要确定替代流，可以问自己，对于主流和子流中的每个操作，“哪些行为可能会出错”。
注意：X是每个用例特有的标识符。

图 5.1 用例事件流模板

图 5.2 所示是“应用程序阻塞器”用例的事件流。该示例使用图 5.1 的模板来构造事

件流。

UC5　“应用程序阻塞器”用例的事件流
5.1前置条件：
1.这是父母的行为。
2.孩子已经使用了一个应用程序很长时间了。
3.关于使用的历史被记录在审核中。
5.2主流：
当孩子使用了一个应用程序时，会引发父母的行为。父母会查看该应用程序的性质是会使孩子上瘾，变得粗俗、暴力，还是良性的、充满乐趣的。取决于这一点，父母会采取某些行动，如阻止它(不当的应用)或只是密码保护(有趣的应用)。
5.3子流：
[S1] 如果是不当的应用程序，如暴力、粗俗的游戏，则执行该子事件流：
当单击“阻止应用程序”按钮时，将显示应用程序阻止对话框。父母只需要选择须从该对话框中阻止的应用程序的类型、名称、阻止原因、开始时间等。在对话框中单击“确定”按钮后，应用程序将被阻止。孩子无法通过手机访问该应用程序。
[S2] 如果是不当的应用程序，如性质良好但有趣的游戏，则执行该子事件流：当单击“阻止应用程序”按钮时，将显示应用程序阻止对话框。父母只需要选择须通过该对话框进行密码保护的应用程序的类型、名称、允许使用的时间、锁定的密码、使用的最大时长以及锁定的原因、开始时间等。在对话框中单击“确定”按钮后，应用程序将被密码保护，孩子只能在一天中的特定时间内使用该应用程序且不能超过规定的时长，应用程序的密码必须从父母处获得。
5.4替代流：
[E1]如果应用程序运行正常且真实有效，则替代流不会发生任何事件。

图 5.2　“应用程序阻塞器”用例的事件流模板

5.3　次 用 例 图

如果当前正在开发的系统有很多功能需求，则这些需求可以分别进行管理。次用例图有助于解决这一问题。与关注系统主要活动的主用例相反，次用例图描述了系统中发生的次要活动。

在“孩子手机监控”这个应用程序中，支持用例“通话监控器”的所有用例如下所示：

① 父母注册应用程序

② 父母登录系统

③ 父母请求查看其孩子的通话历史

④ 父母浏览通话历史

⑤ 基于通话历史，以下行为可能会被触发

（a）不产生任何行为（真实可信的通话）

（b）阻止通话（异常的错误通话）

（c）在一天中的规定时间内被阻止（真实可信但不是很重要的通话）

⑥ 父母退出系统

由此可以看出，单个主用例中具有许多次要功能，从而可以使用次用例图进行详细描述。两种图表中使用的符号没有区别，只是传达的过程/活动的语义不同——主要活动（主用例图）/次要活动（次用例图）。

以下部分将讨论用例图的各种元素及其含义、符号表示，并给出示例来介绍其使用的主要目的。

5.4 用例图中使用的符号

以下是构成用例图的 6 个建模元素：系统、参与者、用例、关联关系、依赖关系和泛化关系。

① **系统**：它定义了系统相对于使用它的参与者（系统外部）和它必须执行的功能（系统内部）的边界。

② **参与者**：它定义了参与执行系统操作的人员、系统、硬件组件或设备的角色。

③ **用例**：它定义了系统的关键功能/特征。如果没有这些功能，系统将不能满足用户/参与者的需求。每个用例都表示了系统必须执行的目标。

④ **关系**：它定义了参与者和用例之间可能的交互关系。从这些关联中可以产生一组相关的场景，用作评估用例的分析、设计和实现时的测试用例。

（a）关联关系：它定义了参与者与用例之间通信的方式。

（b）依赖关系：它定义了两个用例之间可能的通信关系。

i. 扩展关系

ii. 包含关系

（c）泛化关系：它定义了系统的两个参与者或两个用例之间可能的关系，其中一个用例可以继承/派生另一个用例，以及增加或是重写另一个用例的属性。

i. 参与者之间的泛化关系

ii. 用例之间的泛化关系

下面给出关于使用的符号及其语义的详细描述。

5.4.1 系统

项目的首要任务是确定提议的应用程序的背景和范围。这个工作是通过回答许多问题来完成的，例如在你的架构中系统包含了多少模块、这个系统与其他系统之间的关系，以及这个系统的预期用户有哪些。这些细节可以在文档中提供。但是，俗话说，一张图片抵得过千言万语。这句话有助于解释系统符号的简单性，如图 5.3 所示用一个带有名称的矩形来表示系统。该系统符号简单地呈现了控制系统结构的所有元素的上下文。

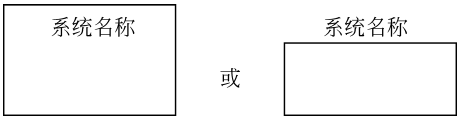


图 5.3 系统组件的用例图图标

要注意的一点是，这种系统图标很少使用。原因是这个符号太受限制了，并没有给图表增加任何实质信息。因此，在大多数工具中，我们只看到用例、参与者及其关系的图标。

回到封装的概念，封装强调使用一个对象，其中你只知道它的接口，而不了解它的内部实现。一个系统类似于一个对象，每个系统都有一个目的和一个接口。只要主要目的和接口保持不变，就可以增强或更改系统的内部实现，而不影响其他系统或对象。因此，定

义系统的主要任务是定义其目的和所需的接口。目的就是项目应用程序的目标。接口是系统外部的参与者与系统的功能（如用例）之间的通信机制。只要考虑到了这些，就可以建立系统内部行为的所有后续建模的背景。

5.4.2　参与者

系统被设计为由用户使用。广义上，用户意味着直接使用系统的人。实际上，用户也可以是处理信息的其他系统、硬件组件或设备。通常，用例图、人员、系统、硬件组件和设备都被称为“参与者”。参与者被定义为外部实体中与系统有关的角色。简而言之，参与者是一个角色，而不一定是一个特定的人或具体的系统。同一个人可以在同一系统中扮演不同的参与者，即他可以在不同的场景下在系统中扮演不同的角色。图 5.4 显示了“孩子手机监控”这个问题域中使用的参与者图标变量。

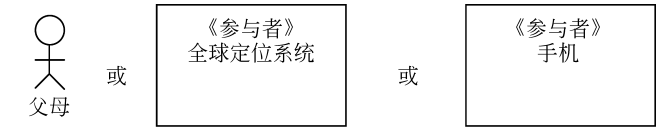


图 5.4　参与者组件的用例图图标

例如，参与者可以是向学生教授课程并指导研究学者的教授的角色。同样的教授可能会评估学生的答卷，在那里他担任“考官”的角色。同一个人因此扮演了两个角色。同样，不同的人也可以同时扮演同样的角色。例如，图书馆有很多借书的人。角色的使用可以指导分析人员关注如何使用系统，而不是关注公司现有的各种职位和职责。为了以更灵活的方式构建系统，同时向任何变化开放，每个人的任务必须与目前的职务分开考虑。

一个非常基本的问题就是，你如何识别潜在的参与者？你需要仔细观察系统的描述，注意人们在使用系统时扮演的角色。当许多人同时执行相同的功能时，尝试为他们在执行该特定功能时共同扮演的角色定义一个共同的名称。

这里有一个小提示，可以帮助你识别系统中涉及的参与者，那就是提出以下简单问题：

- 谁会使用系统的主要功能？
- 谁需要系统的支持以完成他们的日常任务？
- 谁会维护和管理系统并保持系统的持续运转？
- 系统需要控制哪些硬件设备？
- 系统需要和哪些其他系统进行交互？
  这可以分为发起联系的系统，以及与该系统交互的那些系统，例如包括其他计算机系统以及该系统操作的计算机中的其他应用程序。
- 谁对系统产生的结果（值）感兴趣？

基于上述问题，我们获得了“孩子手机监控”这个问题域中的参与者列表。

使用主要功能的用户：父母、孩子

维护和管理系统的用户：网络管理员、管理员（软件组件）

外部硬件系统：手机

软件系统：网络应用程序接口（软件组件）

交互的外部系统：全球定位系统（Global Positioning System, GPS）

5.4.3 用例

用例是定义系统执行所需功能的元素。如果没有这些功能，系统将毫无用处。一旦确定了参与者，就可以根据下列问题来确定用例：

- 参与者要求系统具备哪些功能？
- 系统是否存储信息？通过诸如创建、读取、更新或删除这些信息的操作来处理这些信息将涉及哪些参与者？
- 一旦系统的内部状态发生变化，系统是否需要提醒参与者？
- 系统需要注意哪些外部事件？通过哪些参与者，系统可以了解这些外部事件？

这些用例的名称主要是一个动词短语，表示预期系统必须完成的目标之一，例如查看地理位置、查看通话记录和阻止应用程序（见图 5.5）。虽然这些用例代表了一个支持的过程，但重点是放在总体目标上，而不是过程。

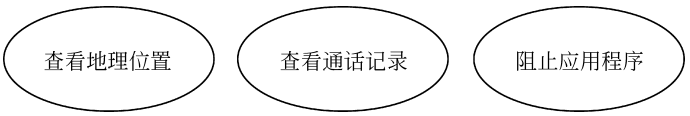


图 5.5 用例组件的用例图图标

当以这种方式定义用例时，系统被指定为一组需求，而不是解决策略。分析人员没有描述系统是如何执行的，而是指定了系统必须执行什么。每一个用例都只传达那些可见的且有意义的功能给使用系统该功能的有关参与者。这个事实有助于避免不正确的功能分解。功能分解就是将过程和任务分解成更小的子过程，直到充分描述系统的所有内部工作。经常遇到的系统开发的陷阱之一是预算超支，这是在每个任务的范围不受控制或模型被设计得太过全面的情况下发生的。

通过使用从客户和潜在用户那里收集的需求作为重要的信息来识别参与者和用例。在识别完成之后，将简要解释用例和参与者。客户必须验证并确保所有的用例和参与者都被识别且能共同满足客户的需求。在此步骤之后，执行用例的描述。这是一个迭代的开发过程，其中每个迭代是用例的一个子集被选择和细化的过程。

最后，审查完成的用例模型（包括用例的描述），然后客户和开发人员据此讨论系统应该做什么并达成一致意见。

本章结尾将列出所选择的应用程序的完整用例图，包括所有的组件及关系，以便更加清楚地了解用例图。

5.4.4 关系

之前的章节中已经定义了系统、参与者和用例等用例图中使用的基本元素。实际上，在实时操作中，用例与参与者或其他用例之间存在关联性。以下部分将描述这种关系。

1. 关联关系

它通过连接参与者和用例的简单线段来表示，如图 5.6 所示。基本上，关联关系意味着参与者与用例通信的过程。事实上，早期版本的 UML 规范称之为“与……通信”的关系。也许这是用例和参与者之间存在的唯一可能的关系。UML 规范还允许在关联线段的任一端使用方向箭头来表示通信的方向。尽管一些关联关系是单向的（例如，参与者向用例提供信息），但大多数关联关系是双向的（即参与者触发用例，用例为参与者提供所需的功能）。为了表示这些双向的关联，箭头可以放置在关联线段的两端，或者根本就不显示箭头。为了简化，大多数用户并不使用箭头表示。一个参与者可以与多个用例相关联。类似地，一个用例可能由多个参与者触发。该图没有提供关于哪个参与者（即“主要参与者”）触发用例的任何明显的线索。一般来说，主要参与者是使用系统服务的人员，辅助参与者是向系统提供服务的角色。

在参与者参与了相似类型的多个用例的情况下，可以在用例末端用多重性因子表示关联关系，如图 5.7 所示。这种多重性参与的具体性质取决于当前的用例。

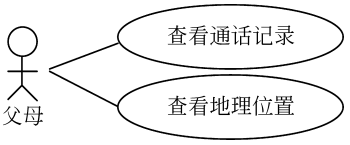


图 5.6　父母参与者与两个用例的关联关系

图 5.7　GPS 参与者涉及多次访问用例

基本上，用例多重性意味着一个参与者触发了多个用例：

① 并行（并发）

② 在不同的时间点

③ 在时间上互斥

类似地，当用例涉及许多参与者实例时，多重性因子被放置在图中参与者的末端，如图 5.8 所示。

参与者的多重性可能意味着

① 特定的用例可能涉及两个独立参与者的同时（并发）的行为（例如，在发射核导弹的情况下）。

② 可能需要参与者之间互补和连续的行为（例如，一个参与者开始做某事，而另一个人停止操作）。

这里的关注点是确定参与者需要获取哪些用例，而非其他。这些连接线段是设计系统接口和后续建模工作的核心。

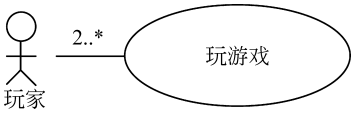


图 5.8　两个或更多的玩家参与者要求触发“玩游戏”用例

2. 依赖关系

① 扩展关系：扩展关系定义了一个“定向关系”，指定了如何以及何时将一个常用的

补充扩展用例的行为插入扩展用例的行为中。

被扩展的用例与扩展用例无关，具有自己的语义。扩展关系由扩展用例所有。同一扩展用例可以扩展多个用例，扩展用例本身也可以被扩展。

被扩展用例中可以定义一个或多个扩展点。

扩展关系被表示为带箭头的虚线，其中箭头从扩展用例指向被扩展（基础）用例。箭头具有关键字<<extend>>作为其标签。如图 5.9 所示。

② 包含关系：包含关系也在需要时定义两个用例之间的定向关系，将被包含用例中非可选的行为添加到包含（基础）用例的行为中。包含用例只有添加了被包含用例，才是完整的、有意义的。

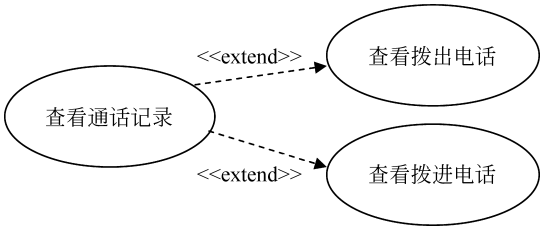


图 5.9 “查看通话记录”用例本身是有意义的。它可以扩展为“查看拨出电话”和“查看拨进电话”两个用例

包含关系在以下两种情况下有帮助。

- 当两个或多个用例的行为中存在相同的部分时
- 当通过将大的用例分解成若干用例来简化时

被包含的用例对于包含关系来说是必需的，而不是可选的。被包含的用例的执行类似于编程中的函数调用或宏命令。在执行包含用例之前，必须已经在包含用例的某个位置上执行被包含用例的所有行为。

如图 5.10 所示，用例之间的包含关系由带箭头的虚线表示，从包含（基本）用例指向被包含（公共部分）用例。箭头具有关键字<<include>>作为其标签。当两个或多个用例有一些共同的行为时，这个共同的部分可以被提取到一个单独的用例中，被多个基用例所包含。

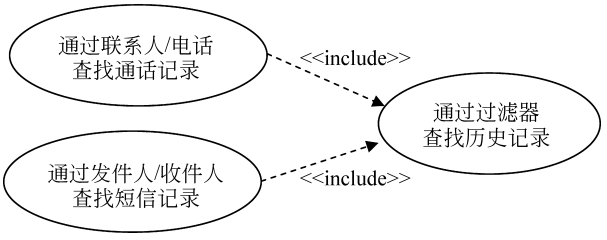


图 5.10 “查找通话记录”和“查找短信记录”用例都要求（包含）“通过过滤器查找历史记录”用例

3. 泛化关系

有时我们遇到不同的参与者以相同的方式使用同一系统的情况。例如，母亲和父亲可能都会登录和退出“孩子手机监控软件”这个系统。在这种情况下，用例对用户提供的功

能是类似的。例如，他们都会查看通话记录、查看短信记录、阻止某些不需要的应用程序、阻塞时间和定期监视孩子的联系人。所有这些用例都由所有这些参与者使用。

泛化是一种概念，可以帮助我们表示这些类型的场景，在面向对象编程、分析和设计中代表继承。继承是指一个对象在创建时可以访问除了自己的类之外的另一个类的所有属性，并且这些属性具有该对象自身定义的属性值。通常在用例图中，泛化经常被用于描述参与者和用例，也常表示“是一个”关系。

① 参与者之间的泛化关系：参与者之间的泛化关系是指将用例中更具体的参与者连接到更为一般化的参与者，表明这些具体参与者的实例可以替代一般参与者的实例。这些具体的参与者拥有用例提供的相同功能，而无论其身份如何。参与者之间的泛化关系用一条实线路径表示，从更具体的参与者出发，指向更为一般化的参与者，并在路径的末端（即更一般化的参与者）加上一个空心的三角形。

图 5.12 用参与者之间的泛化关系细化了图 5.11。图中，参与者“父母”触发了“登录”和“退出”两个用例。父亲、母亲都属于参与者“父母”。

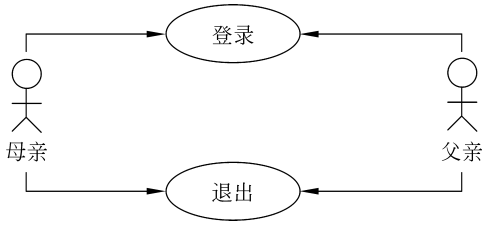


图 5.11　相似参与者

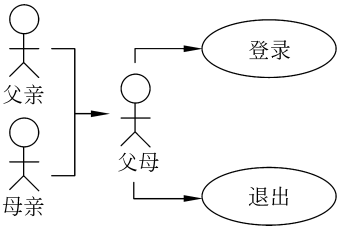


图 5.12　参与者之间的泛化关系

② 用例之间的泛化关系：在图 5.13 中，父母需要查看孩子的地理位置，这里通过两种方式查看：父母通过手机上的应用程序组件向服务器发送一个短信请求，服务器告知孩子的地理位置；或者父母登录网络应用程序界面，通过在线表单提交请求，以查看孩子的地理位置。因此，用例涉及父母对孩子地理位置的查看，地理位置状态信息的收集、存储和操作等所有的过程。可以使用用例之间的泛化关系来表示以上的场景，即抽取和重用多个用例中的相同行为。

用例之间泛化关系的表示是从更具体的用例指向更为一般化的用例，表明这些更具体的用例获取或继承了更一般化的用例，包括其过程、参与者、行为序列和扩展点，具体用例的实例也可以替代一般化用例的实例。用例之间的泛化关系也显示为一条实线路径，由更具体的用例指向更为一般化的用例，并在路径的末端（即更一般化的用例）加上一个空心的三角形。

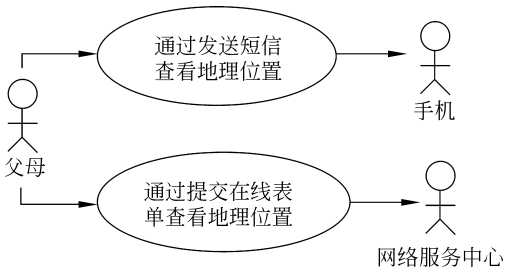


图 5.13　相似用例