第16章 用　　况

本章内容：

- 用况、参与者、包含和延伸
- 对元素的行为建模
- 用协作来实现用况

没有任何一个系统是孤立存在的。每个有意义的系统都会与为了某些目的而使用该系统的人或自动的参与者进行交互，这些参与者期望系统以可预料的方式运行。一个用况代表一个系统或系统的一部分的行为，是对一组动作序列的描述，系统执行该动作序列（其中包括变体）来为参与者产生一个可观察的结果值。

你可以用用况来描述你正在开发的系统想要实现的行为，而不必说明这些行为如何实现。用况为开发者提供了一种途径以使他们与系统的最终用户和领域专家产生公共的理解。另外，用况还帮助我们在开发过程中验证体系结构，并随着系统的演化对系统进行校验。在实现系统时，这些用况是通过协作来实现的，协作中的元素共同工作以完成每一个用况。

结构良好的用况只表示系统或子系统的基本行为，并且既不过于一般，也不过于特殊。

16.1　入门

一个经过精心设计的房子，要比用几堵墙简单围起来再撑起一个屋顶来挡风避雨的房子强上百倍。当你和建筑师一起设计房子时，你会更仔细地考虑怎样使用房子。如果喜欢招待客人，你会考虑到把房子设计得更易于人们在房间里交谈，而避免导致过于集中的死角。当你考虑的是为家人准备饭菜时，你会考虑到把厨房设计得更易于储藏，更有效地摆放器具。甚至规划从车库到厨房的路径以便于卸货，也将影响你最终如何连通房间。如果你有一个大家庭，则还要考虑浴室的使用。及早地在设计中考虑到浴室的个数和设置，会极大地降低早晨家人都忙着上学和上班时所产生的拥挤。如果家中有十来岁的儿童，这个问题显得尤为重要。

推断你和你的家人将如何使用房子，是基于用况进行分析的一个例子。你要考虑使用房子的各种方式，而这些用况则将驱动建筑结构。许多家庭对房子有相同的使用情况，如吃饭、睡觉、哺育孩子和保存纪念品。每个家庭也都有自己特殊的使用情况或有这些基本使用情况的不同变种。比如，一个大家庭的需要与一个刚刚走出校门的单身成年人的需要是不同的。正是这些不同极大地影响着房子的最后形状。

创建这样的用况的一个关键因素是在创建中不详述用况是如何被实现的。举例来说，可以通过在用况中陈述用户与系统如何交互，来描述一个 ATM系统是怎样工作的，而不必知道 ATM

内部的任何细节。用况说明想要的行为，而不说明行为是如何被执行的。这样做的最大好处是让你（如最终用户和领域专家）与你的开发者（建造系统以满足你的需求的人）之间不必为细节所累。当然，将来肯定会考虑这些细节，但用况使你着眼于对你来说最具风险的问题。

在UML中，所有这些行为都可以建模为用况，而用况的描述可以独立于它们的实现。一个用况是一组活动序列（其中包括变体）的描述，系统执行这些动作序列来为参与者产生一个可观察的结果值。这个定义中有许多重要的部分。

一个用况描述一组序列，每一个序列表示系统外部的事物（系统的参与者）与系统本身（和它的关键抽象）的交互。这些行为实际上是系统级的功能，用来可视化、详述、构造和文档化在需求获取和分析过程中所希望的系统行为。一个用况描述了系统的一个完整的功能需求。例如，银行的一个重要的用况是处理贷款。【在第15章中讨论交互，在第16章中讨论需求。】

一个用况包含系统与参与者的交互。一个参与者表示的是用况的使用者在与这些用况进行交互时所扮演的角色的一个紧密的集合。参与者可以是人或自动的系统。例如，在对一个银行建模时，处理贷款所包含的是顾客与贷款员之间的交互。

一个用况可以有变体。在所有有趣的系统中，你将发现如下几种变体——作为其他用况的特化版本的用况，被包含在其他用况中作为其中一部分的用况，以及延伸其他核心用况的行为的用况。你可以按照这3种关系来组织一组用况，从而分解出其中的公共的、可复用的行为。例如对一个银行建模时，在处理贷款的基本用况中有许多变体，如大型抵押贷款与小型的商务贷款是不同的。然而，这些用况在某种程度上共享公共的行为，如审查顾客贷款资格的用况是处理任何一种贷款都有的一个行为。

一个用况要完成一些确定的工作。从一个给定的参与者的角度看，一个用况完成对一个参与者有价值的事情，如计算结果，并产生一个新的对象，或改变另一个对象的状态。在对银行建模的例子中，处理一笔贷款将导致交付一项已被批准的贷款，这表现为把一大笔钱交到顾客手中。

用况可以应用于整个系统，也可以应用于系统的一部分，包括子系统，甚至单个的类和接口。在各种情况下，这些用况不仅代表这些元素被期望的行为，而且还可把这些元素用作开发过程中测试用例的基础。应用于子系统的用况是回归测试的极好的来源；用于整个系统的用况，是集成测试和系统测试的极好的来源。如图 16-1所示，UML在图形上提供了用况和参与者的表示法。这种表示法允许你忽略用况的实现来可视化这个用况并可以在带有其他用况的语境中可视化这个用况。【在第31章中讨论子系统；在第4章和第9章中讨论类；在第11章中讨论接口。】

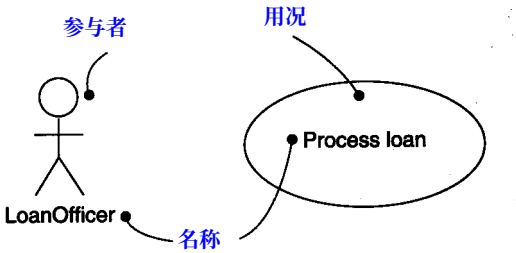

图16-1 参与者与用况

16.2 术语和概念

用况（use case）是对一组动作序列（其中包括它的变体）的描述，系统执行该动作序列来为参与者产生一个可观察的结果值。在图形上，一个用况用一个椭圆表示。【用况的表示法和协作图的表示法相似，这在第27章中讨论。】

1. 名称

每个用况都必须有一个区别于其他用况的名称。名称（*name*）是一个文字串。单独的名称叫做简单名（*simple name*）；在用况名前加上它所属于的包的名称的用况名叫做路径名（*path name*）。用况通常仅显示它的名称，如图 16-2所示。【一个用况的名称在包含它的包中应是唯一的，这在第12章中讨论。】

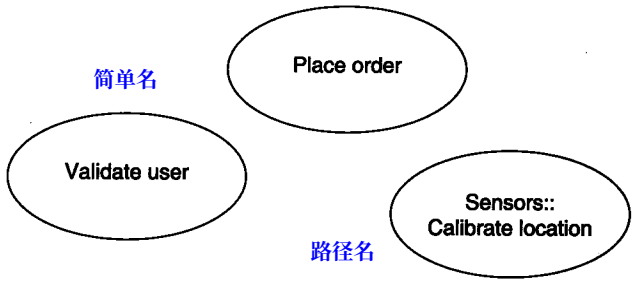

图16-2 简单名与路径名

注释 用况名可以是一个文字串，其中包括任意数目的字母、数字和大多数标点符号（冒号除外，它用来将类名与它所属于的包的名称分隔开来），并且可以连续多行。在实际应用中，用况的名称是简短的主动语态的动词短语，用来命名被建模的系统的某些行为。

2. 用况与参与者

一个参与者表示用况的使用者在与这些用况进行交互时所扮演的角色的一个紧密的集合。通常，一个参与者代表的角色有：人、硬件设备，或甚至另一个系统。例如，如果你在银行工作，你可能是一个贷款员（`LoanOfficer`）。如果你又在那里有私人储蓄，那么你同时也扮演一名顾客（`Customer`）的角色。所以，一个参与者的一个实例代表以一种特定的方式与系统进行的单独的交互。尽管在模型中使用参与者，但参与者实际上并不是系统的一部分。它们存在于系统之外。

如图16-3所示，参与者用人形图符表示。可以定义参与者的一般种类（如 `Customer`）并通过泛化关系将它们特殊化（如`Commercial Customer`）。【在第5章和第10章中讨论泛化。】

注释 可用UML 提供的扩展机制对参与者构造型化，以便提供一个不同的图标，这个图标可能提供一个更好的可视化提示。【在第6章中讨论构造型。】

参与者仅通过关联与用况相连，一个参与者和一个用况之间的关联表示两者之间的通信，任何一方都可发送和接收消息。【在第5章和第10章中讨论关联关系；在第15章中讨论消息。】

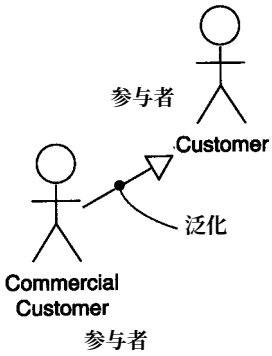

图16-3 参与者

3. 用况与事件流

用况描述的是一个系统（或一个子系统、类或接口）做什么（*what*），而不是说明怎么做（*how*）。在建模时，重要的是区分清楚外部与内部视图的界限。

可以通过用足够清晰的、外部人员很容易理解的文字描述一个事件流，来说明一个用况的行为。书写这个事件流时，应该包含用况何时开始和结束，用况何时和参与者交互，什么对象被交换，以及该行为的基本流和可选择的流。

例如，在ATM系统中，描述一个用况`ValidateUser`可以采用以下方式：

主事件流：在系统提示顾客输入PIN编号时，用况开始。顾客通过按键输入PIN号。顾客按“输入”按钮确认登录。系统校验这个PIN号是否有效。如果有效，系统承认这次登录，该用况结束。

异常事件流：顾客可以在任何时间通过按“取消”按钮取消一个事务，这样，该用况重新开始。顾客的账户未发生改变。

异常事件流：顾客可以在确认之前的任何时刻消除PIN号，并重新输入一个新的PIN号。

异常事件流：如果顾客输入了一个无效的PIN 号，用况重新开始。如果连续3次无效，系统将取消整个事务，并在60秒内阻止该顾客与ATM进行交易。

注释 可以用许多方式来说明一个用况的事件流，包括非形式化的结构化文字（如上例）、形式化的结构化文字（使用前置或后置条件）以及伪码。

4. 用况与脚本

通常，你会先用文字来描述一个用况的事件流。然而，随着你对系统需求的理解进一步精化，你也会想用交互图以图形化的方式来说明这些流。一般你将用一个顺序图来说明一个用况的主流，并用这种图的变体来说明该用况的异常流。【交互图包括顺序图和协作图，这在第18章中讨论。】

将主流与可选流分开描述是合适的，因为一个用况描述的是一组序列，而不只是一个单独的序列，将一个有趣的用况的全部细节仅用一个序列来表达是不可能的。例如，在一个人力资源系统中，有一个用况为`Hire employee`这种一般的业务功能可能会有许多变体。你可能会从另一家公司雇佣一个人（这是最普通的脚本）；你也可能从一个分部调配一个人到另一个

分部（常见于跨国公司中）；或者，你也可能雇佣一个外国人（此脚本包含它自身的特殊规则）。这些变体的任一个都可被表示在一个不同的序列中。

这个用况（Hire employee）实际上描述了一组序列，其中每个序列通过所有这些变体代表一个可能的流。每个序列被称作一个脚本。脚本是一个表示行为的特定动作序列。脚本对于用况，相当于实例对于类，也就是说一个脚本基本上是一个用况的一个实例。【在第13章中讨论实例。】

注释 从用况到脚本含有扩充的因素。一个比较复杂的系统可能会包含几十个表示行为的用况，而每个用况可能扩充为几十个脚本。对于每个用况，你都可以发现主要脚本（定义基本序列）和次要脚本（定义可选择序列）。

5. 用况与协作

用况捕获被开发的系统（或子系统、类或接口）想要实现的行为，而不必说明这些行为是怎样实现的。这是一个重要的分界线，因为系统分析（说明行为）应尽可能地不涉及实现问题（说明行为如何被完成）。然而，最后你还是要实现用况，并通过创建由一起工作以实现这个用况的行为的类和其他元素所构成的群体来实现用况。这组元素的群体，既包括静态的结构，也包括动态的结构，在UML中被建模为一个协作。【在第27章中讨论协作。】

如图16-4所示，通过一个协作可以显式地说明一个用况的实现。因为在大多数情况下，一个给定的用况恰好是通过一个协作实现的，所以不必在模型中显式地对这个关系建模。【在第9章和第10章中讨论实现。】

注释 尽管你可以不在图中显式地可视化这种关系，但你使用的管理模型的工具可能会维护这种关系。

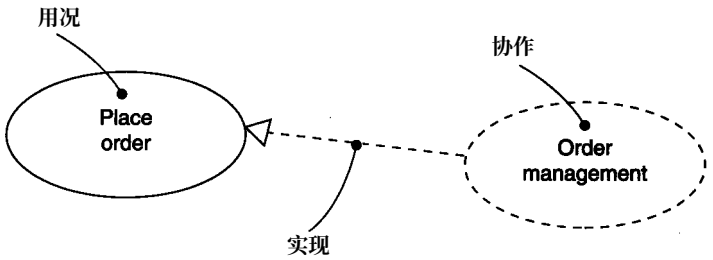

图16-4 用况与协作

注释 寻找结构良好的协作的最小集合，而这些协作能够满足系统的所有用况中说明的事件流，是一个系统的体系结构的焦点。【在第2章中讨论体系结构。】

6. 组织用况

可以采用将类组织到包中同样的方式，将用况组织到包中。【在第12章中讨论包。】

你也可以通过描述用况之间的泛化、包含和延伸关系来组织用况。为了分解公共的行为（通过从它所包含的其他用况中提取这样的行为）和为了分解变体（把这样的行为放入延伸它的其他用况中），你可以应用以上这些关系。

用况之间的泛化关系就像类之间的泛化关系一样。子用况继承父用况的行为和含义；子用况还可以增加或覆盖父用况的行为；子用况可以出现在父用况出现的任何位置（父和子均有具体的实例）。例如在一个银行系统中，有一个用况 ValidateUser，用来验证用户的合法性。它有两个特殊的子用况（Check Password和Retinal Scan），它们都有父用况ValidateUser那样的行为，并且可以出现在父用况出现的任何地方。它们还添加了它们自己的行为（前者检查文本密码，后者检查用户的唯一的视网膜模式）。如图16-5所示，用况间的泛化关系用一个带有大的空心箭头的实线表示，就像类之间泛化关系的表示法一样。【在第5章和第10章中讨论泛化。】

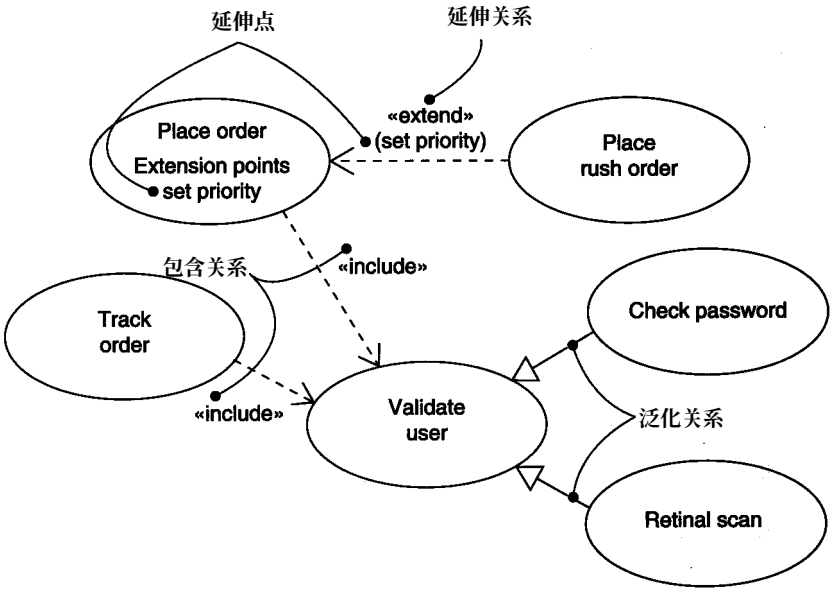

图16-5 泛化关系、包含关系和延伸关系

用况之间的包含关系表示基础用况在它内部说明的某一位置上显式地合并了另一个用况的行为。被包含的用况从不孤立存在，仅作为某些包含它的更大的基础用况的一部分出现。你可以把包含关系想象为基础用况从供应者用况中提取行为。

使用包含关系，通过把公共的行为放到它自己的一个用况中（该用况被一个基础用况所包含），可以避免多次描述同一事件流。包含关系本质上是一个委托的例子，你可以获得系统的一组职责并在一个地方获取它（即被包含的用况），然后，每当系统的所有其他部分（其他的用况）需要使用这些功能时就去包含这个新的职责聚合。

可以把包含关系表示成一个构造型为 include的依赖关系。为了说明在基础用况的事件流中包含另一个用况的行为的位置，只需简单地在书写 include后接着写想包含的用况的名称，例如下面的流Track order。【在第5章和第10章中讨论依赖关系；在第6章中讨论构造型。】

主事件流：获取和校验订单号。Include（Validate User）。对于这个序列中的每一部分，查询它的状态，然后将报告返回给使用者。

用况之间的延伸关系表示，基础用况在由延伸用况间接地说明的一个位置上，隐式地合并了另一个用况的行为。基础用况可以单独存在，但是在一定的条件下，它的行为可以被另一个用况的行为延伸。这个基础用况只是在一个被称作它的延伸点的确定点上被延伸。可以将延伸关系理解为延伸用况把行为放入基础用况中。

延伸关系用于对可能被用户看作是可选系统行为的用况的一部分建模。通过这种方式，可以把可选行为从必须行为中分离出来。使用延伸关系还可以描述一个只有在给定条件下执行的独立的子流。最后，还可以使用延伸关系对一些可在某一确定点被插入，并通过与参与者显式地交互而进行控制的流建模。

可以把延伸关系表示为一个构造型为 extend的依赖关系。在一个附加栏里列出基础用况的延伸点，这些延伸点其实就是在基础用况流中出现的标号。【在第5章和第10章中讨论依赖关系；在第6章中讨论构造型和附加栏。】例如，流Place Orde可以像下面这样读：

主事件流：include（Validatuese）。收集用户的订单项。（set priority）。呈交处理的订单。

在这个例子中，set priorit就是一个延伸点。一个用况可以有多个延伸点（可出现多次），这些延伸点通常被赋予名称。正常情况下，这个用况是在不考虑订单的优先级的情况下执行的。从另一方面讲，如果这是一个优先订单的实例，基础用况流将会按照上面的叙述执行。但在延伸点（set priority），延伸用况（Place rush orde）的行为将被执行，然后流将继续。如果有多个延伸点，延伸用况将会按顺序将它们简单地包含在流中。

注释 组织用况是通过提取公共行为（包含关系）和区分变体（延伸关系）来实现的，它是为系统创建一个简单、平衡和易于理解的用况集合的重要部分。

7. 其他特性

用况是一种类元，所以它可以像类一样具有属性和操作。可以把这些属性想象为用况中的对象，你需要描述其外部行为。类似地，可以把这些操作想象为系统的动作，你需要描述一个事件流。这些对象和操作可以被用在交互图中，用来描述用况的行为。【在第4章中讨论属性和操作。】

作为类元，你也可以为用况附加上状态机。状态机是描述用况所代表的行为的另一种方式。【在第21章中讨论状态机。】

16.3 普通建模技术

对元素的行为建模

使用用况的最常见的方式是对元素的行为建模，不论它是整个系统、一个子系统或是一个类。对这些事物的行为建模时，最重要的是关心元素做什么，而不是怎么做。【在第31章中讨论系统和子系统；在第4章和第9章中讨论类。】

以这种方式把用况应用于元素是重要的。这其中主要有以下三个原因：第一，通过用况对元素的行为建模，可以向领域专家提供一种说明元素的外部视图的方法，达到开发者足以利用它来构造其内部视图的程度。用况为领域专家、最终用户和开发者提供了一个相互交流的论坛。第二，用况为开发者提供了一种认识和理解元素的方法。系统、子系统或类可能会很复杂，充满了操作和其他部分。通过说明一个元素的用况，可以帮助这些元素的使用者根据他们将如何使用这些元素而直接地认识它们。如果没有这些用况，用户将不得不亲自搞清楚怎样使用这些元素。用况可以让一个元素的作者就应该如何使用这个元素，交流他或她的意图。第三，用况是开发期间随着演化而测试每个元素的基础。通过不断地测试每个元素和它的用况，可以不间断地校验它的实现。这些用况不仅为回归测试提供了依据，而且每当向一个元素中加入新的用况时，还可以迫使我们重新考虑这个计划的实现，以确保这个元素易于修改。否则，必须恰当地调整体系结构。

对元素的行为建模，要遵循如下策略：

- 识别与这个元素交互的参与者。候选参与者包括需要一定行为来执行任务的小组，或者需要直接或间接地完成这个元素功能的小组。
- 通过识别一般的或较特殊的角色来组织参与者。
- 对于每个参与者，考虑它与这个元素进行交互的主要方式，还要考虑改变这个元素的状态或它的环境的交互，或者要考虑涉及对某些事件响应的交互。
- 考虑每个参与者与这个元素进行交互的异常的方式。
- 把这些行为组织为用况，应用包含关系和延伸关系分解公共行为，并区分异常的行为。

例如，一个零售系统将与订货的顾客进行交互。然后，系统将装运订货，并通告顾客付帐。如图16-6所示，可以通过把这些行为声明为用况（Place order、Track order、Ship

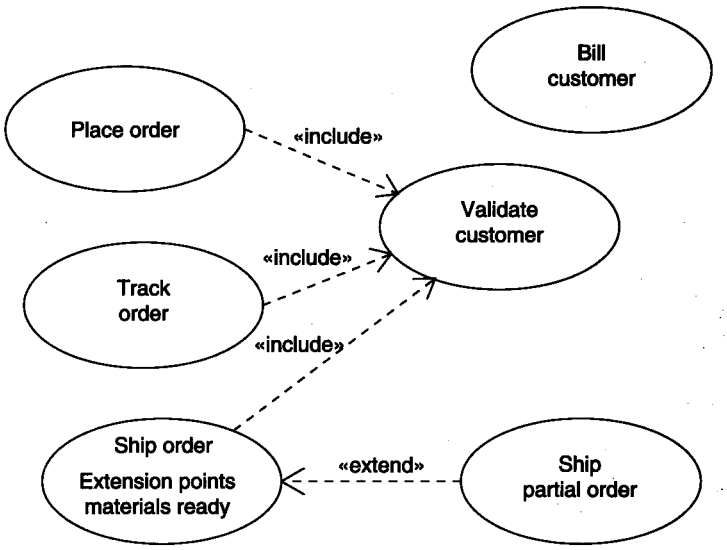

图16-6 对元素的行为建模

order和Bill customer）来对系统的行为建模。公共的行为（Validate customer）可以被分解出来，变体（Ship partial order）也被区分开来。每个这样的用况都包含有对行为的规格说明，既可以通过文字、状态机，也可以通过交互来表示。

随着模型越来越大，你会发现有许多用况应该簇集在一起形成一些概念和语义上相关的组。在UML中，用包来对这些类簇建模。【在第12章中讨论包。】

16.4 提示和技巧

在UML中对用况建模时，每个用况应该表示系统或部分系统的可区分和可标识的行为。一个结构良好的用况，应满足如下的要求：

- 为系统和部分系统中单个的、可标识的和合理的原子行为命名。
- 通过从它所包含的其他用况中提取公共的行为，来分解出公共行为。
- 通过把这些行为放入延伸它的其他用况中，来分解出变体。
- 清晰地描述事件流，足以使局外人轻而易举地理解。
- 是通过说明该用况的正常和变体语义的最小脚本集来描述的。

在UML中绘制一个用况时，要遵循如下的策略：

- 仅仅显示那些对理解系统或部分系统的行为来说重要的用况。
- 仅显示那些与这些用况有关的参与者。