第 3 章

用例和用例图

3.1 用　　例

用例(use case)这个概念是 Jacobson 于 20 世纪 60～70 年代在爱立信公司开发 AKE、AXE 系列系统时提出的,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。

一些国内出版的书籍也有把用例翻译为用况、用案等。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所进行的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 中,用例用一个椭圆表示,用例名往往用动宾结构或主谓结构命名(如果用英文命名,往往是动宾结构)。图 3.1 表示的是用例的例子。

例 3.1　如图 3.2 所示,在字处理程序中,“置正文为黑体”是一个用例,“创建索引”也是一个用例。从这个例子可以看到,用例的粒度可大可小,有的用例可能很简单,如“置正文为黑体”这个用例就比较简单,很快就可以实现,但“创建索引”这个用例就比较复杂,实

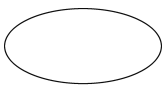
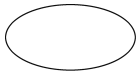


图 3.1　用例的例子 1



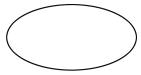


图 3.2　用例的例子 2

现起来可能要花费很长的时间。

例 3.2　在一个银行业务系统中，可能会有以下用例。

（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。

根据上面的例子，可以发现采用用例进行需求分析时的一些特点。

（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需求，对应一些具体的用户目标。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。

理论上，可以把一个软件系统的所有用例都画出来，但在实际开发过程中，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，这也是 UML 初学者易犯的一个错误。初学者对 UML 的一个普遍误解就是，认为用例可以表示所有的系统需求，因此千方百计地要用 UML 中的符号表示那些事实上很难用用例表示的需求。需求有两种基本形式：功能性需求和非功能性需求。那些用 UML 难以表示的需求很多是非功能性需求。例如，开发项目中所涉及的术语就很难用 UML 表示。对于这些需求往往是采用附加补充文档的形式描述的。

用例并不是系统的全部需求，用例描述的只是功能性方面的需求。在编写一个系统的需求说明时，应该根据特定的需求大纲来写，很多开发组织或个人提供了需求大纲供参考。例如，A.Cockburn 给出的需求大纲把需求分为以下 6 大部分。

（1）系统的目的和范围。

（2）系统中的术语表。

（3）用例。

（4）系统采用的技术。

（5）开发过程中的参加人员、业务规划、系统运行所依赖的条件、安全要求、文档要求等其他需求。

（6）法律、政治、组织机构等方面的问题。

从上面的需求大纲可以看到，用例只是所有需求中的一部分内容。

用例这种技术很容易使用，但也很容易误用。正确使用用例分析做好领域建模，以确保定义正确的需求，然后开发出正确的系统，是保证 OO 软件开发成功的基础。对于初学者来说，掌握用例的概念并不难，但要在具体的项目中灵活使用用例捕获用户的需求，并不是一件容易的事情，往往需要用户的经验、沟通能力和丰富的领域知识等。目前有很多介绍如何使用用例分析的书，其中 A.Cockburn 的 *Writing Effective Use Cases* 一书中，对如何确定一个系统的用例有深入的论述，书中总结并分析了一些使用用例时的常见错误。

本质上，用例分析是一种功能分解的技术，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。不管怎样，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，用例分析做得好，接着的交互图分析、类图分析等才有可能做得好，整个系统的开发才能顺利进行。

用例是与实现无关的关于系统功能的描述。在 UML 中，可以用协作说明对用例的实现。协作是对由共同工作的类、接口及其他元素组成的群体命名，这组群体提供合作的行为。图 3.3 是用例 Login 及其两个实现。

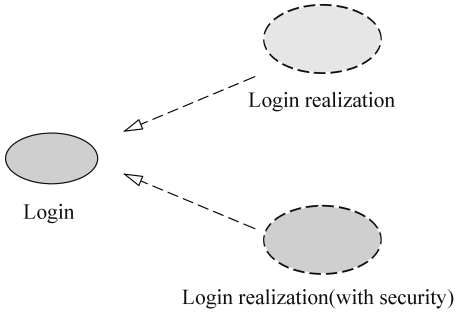


图 3.3　用例及其实现（简单实现以及带验证实现）

在 UML 中，协作用虚线椭圆表示。在图 3.3 中，对用例 Login 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。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，一个是结构部分，如类、接口及其他建模元素等；另一个是说明类、接口及其他建模元素如何协调工作的行为部分，如协作图、顺序图、类图等。

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

3.2　参　与　者

3.2.1　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东西，包括人、设备、

外部系统等。由于 UML 是最近几年才在国内流行起来的，所以很多译名并没有统一，如 Actor 就有很多不同的译名，包括参与者、活动者、执行者、行动者等。在本书中，采用参与者这个译名。

在一个银行业务系统中，可能会有以下参与者。

客户：从系统获取信息并执行金融交易。

管理人员：开办系统的用户，获取并更新信息。

厂商：接受作为转账支付结果的资金。

E-mail 系统。

在一个教务管理系统中，可能会有以下参与者。

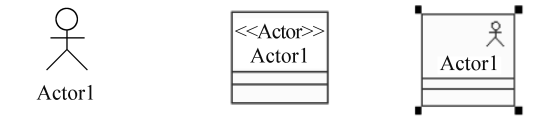
学生：在系统中完成个人信息的查询、修改、选课、成绩查询等功能。

教师：在系统中完成上传学生成绩、打印所授班级的学生成绩等功能。

管理员：在系统中完成学籍管理、成绩管理、考试管理、调停课管理、排课管理、教学管理等功能。

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。但需要注意的是，参与者实际上并不是系统的一部分，尽管在模型中会使用参与者。

在第 5 章讲到版型(Stereotype)的时候会提到，参与者实际上是一个版型化的类，其版型是<<Actor>>。图 3.4 是参与者的 3 种表示形式。



(a) Icon 形式　(b) Label 形式　(c) Decoration 形式

图 3.4　参与者的 3 种表示形式

可以用人形图标表示(Icon 形式)参与者，也可以用带有版型标记的类图表示(Label 形式)参与者。一般用人形图标表示的参与者是人，用类图表示的参与者是外部系统。另一种表示形式是修饰(Decoration)形式，这种表示形式兼有人形图标形式和版型标记形式的特征。

3.2.2　寻找和确定参与者

在获取用例之前，首先要确定参与者，在查找过程中，可以询问以下问题帮助确定参与者。

谁可以使用系统的主要功能？

谁有权改变系统的数据？

谁能够从系统获取数据？

谁负责支持和维护系统？

系统需要控制哪些外部资源或硬件设备？

系统需要与哪些外部系统交互？

谁对系统运行结果感兴趣？

一般说来，凡是直接使用系统的人都可以确定为参与者。另外，在对参与者的建模过程中，除了需要关注参与者的特征外，还应注意以下几点。

(1) 每个参与者应该有简短的描述，从业务角度描述参与者及其权利、功能是什么。

(2) 参与者可以具有属性和操作。

3.2.3 参与者之间的关系

由于参与者事实上就是类，因此，参与者之间也有继承关系(但在分析设计阶段，一般使用泛化这个词表示继承的意思)。参与者之间的泛化(Generalization)关系表示一个一般性的参与者(称作父参与者)与另一个更为特殊的参与者(称作子参与者)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 中，泛化关系用带三角形箭头的实线表示。如图 3.5 所示是参与者之间泛化关系的例子，其中，Commercial Customer 是子参与者，Customer 是父参与者。

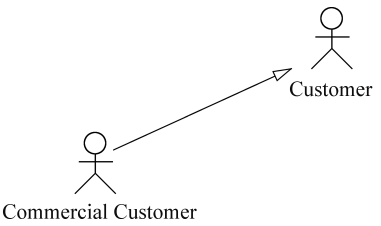


图 3.5 参与者之间的泛化关系

3.3 脚 本

脚本(Scenario)也被翻译为情景、场景、情节、剧本等。在 UML 中，脚本指贯穿用例的一条单一路径，用来显示用例中的某种特殊情况。

脚本是用例的实例(Instance)，如果与类和对象之间的关系做比较，则脚本与用例的关系相当于对象与类的关系。

每个用例都有一系列脚本，其中包括一个主要脚本，以及多个次要脚本。相对于主要脚本，次要脚本描述了执行路径中的异常或可选择的情况。

例 3.3 在“订货”这个用例中，包含着几个相关的脚本。一个是订货进行顺利的脚本，一个是相关货源不足的脚本，一个是涉及购货者的信用卡被拒的脚本，等等。这些脚本的组合构成了一个用例。

一个脚本用具体的文字描述表示，在 3.6 节讨论用例的表示时有具体的脚本描述的例子。

3.4 用例间的关系

用例除了与参与者有关联(Association)关系外，用例之间也存在一定的关系，如泛化关系、包含关系、扩展关系等。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 中的版型扩展机制。

3.4.1　泛化关系

泛化(Generalization)代表一般与特殊的关系。泛化是 OOA/OOD 中用得较多的术语。它的意思与面向对象程序设计语言中继承的概念类似,但在分析和设计阶段,用泛化这个术语更多一些。

在泛化关系中,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。

如图 3.6 所示是用例之间泛化关系的例子。

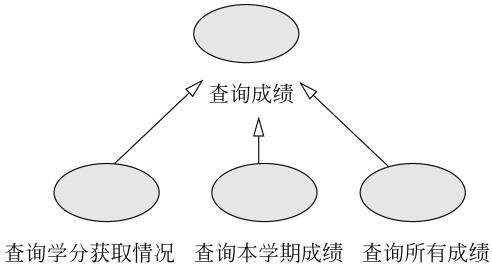


图 3.6　用例之间的泛化关系

在图 3.6 的例子中,父用例是查询成绩,子用例有查询学分获取情况、查询本学期成绩和查询所有成绩三个。若父用例的名字用的是斜体字体,表示该父用例是抽象父用例。

3.4.2　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(称作基本用例)的行为包含另一个用例(称作包含用例)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一些语义。如图 3.7 所示是包含关系的例子,其中,用例学籍管理是基本用例,用例班级设置、休学管理和退学管理是包含用例。

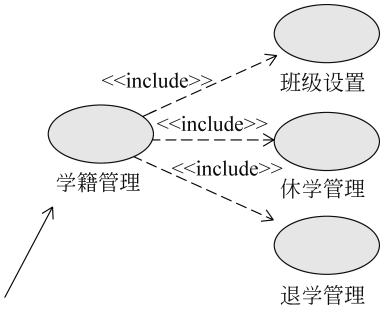


图 3.7　用例之间的包含关系

在包含关系中,箭头的方向是从基本用例到包含用例,也就是说,基本用例是依赖于包含用例的。

需要说明的是，在 UML 1.1 版本的规范说明中，用例之间是使用(Uses)和扩展(Extends)这两种关系，且这两种关系都是泛化关系的版型。但在 UML 1.3 以后版本的规范说明中，用例之间是包含(Include)和扩展(Extend)这两种关系。UML 1.1 版本中的 Uses 关系已被取消，且 UML 1.3 版本中，Include 和 Extend 都是依赖关系的版型，而不是泛化关系的版型。

3.4.3 扩展关系

扩展(Extend)关系的基本含义与泛化关系类似。但在扩展关系中，对于扩展用例有更多的规则限制，即基本用例必须声明若干“扩展点”，而扩展用例只能在这些扩展点上增加新的行为和含义。与包含关系一样，扩展关系也是依赖关系的版型，也就是说，包含关系是特殊的依赖关系。

如图 3.8 所示是扩展关系的例子。

与包含关系不同的是，在扩展关系中，箭头的方向是从扩展用例到基本用例，也就是说，扩展用例是依赖于基本用例的。

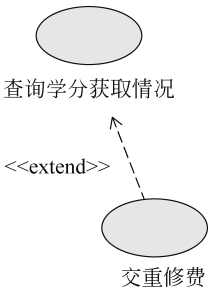


图 3.8 用例之间的扩展关系

3.4.4 用例的泛化、包含、扩展关系的比较

一般来说，可以用“is a”和“has a”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，只有当满足了扩展点的要求时才执行扩展用例。

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

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型而且只是偶尔描述时，可以考虑只用泛化关系。

当描述正常行为的变型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 1.1 版本以前，扩展关系就是用泛化关系的版型表示的，在 UML 1.3 版本以后，扩展关系改为用依赖关系的版型表示。

表 3.1 是参与者、用例之间关系的总结。

表 3.1　参与者和用例的关系

关系类型	说　明	表示符号
关联(Association)	Actor 和 Use Case 的关系	———
泛化(Generalization)	Actor 之间或 Use Case 的关系	———▷
包含(Include)	Use Case 之间的关系	<<include>> ……>
扩展(Extend)	Use Case 之间的关系	<<extend>> ……>

在这里总结一下 UML 中关系(Relationship)、关联(Association)、泛化(Generalization)和依赖(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、依赖等几种,另外还有一种关系是实现,表 3.1 中没有提到。

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)、节点(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,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,被依赖的元素称作目标元素,依赖元素称作源元素。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。

3.5　用　例　图

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。在 UML 中,一个用例模型由若干个用例图描述。如图 3.9 所示是在 Rational Rose 2003 中画出的金融贸易系统的用例图。

在该例子中,有 Trading Manager、Accounting System、Trader、SalesPerson 等参与者。其中,Accounting System 这个参与者是一个外部系统,用例有 Set Limits、Update Accounts、Analyze Risk、Price Deal、Limits Exceeded 等。

需要说明的是,Rose 中并没有实现 UML 1.5 规范说明中的所有建模符号,如 3.4.3 节提到的扩展点在 Rose 中就不能直接画出来,但这并不妨碍 Rose 成为市场领先的支持 UML 的工具。

UML 规范说明中并不使用颜色作为图形语义的区分标记,但建模人员可以在 Rose 中给某些图符加上填充颜色,以强调某一部分的模型,或希望引起使用者的特别注意。但

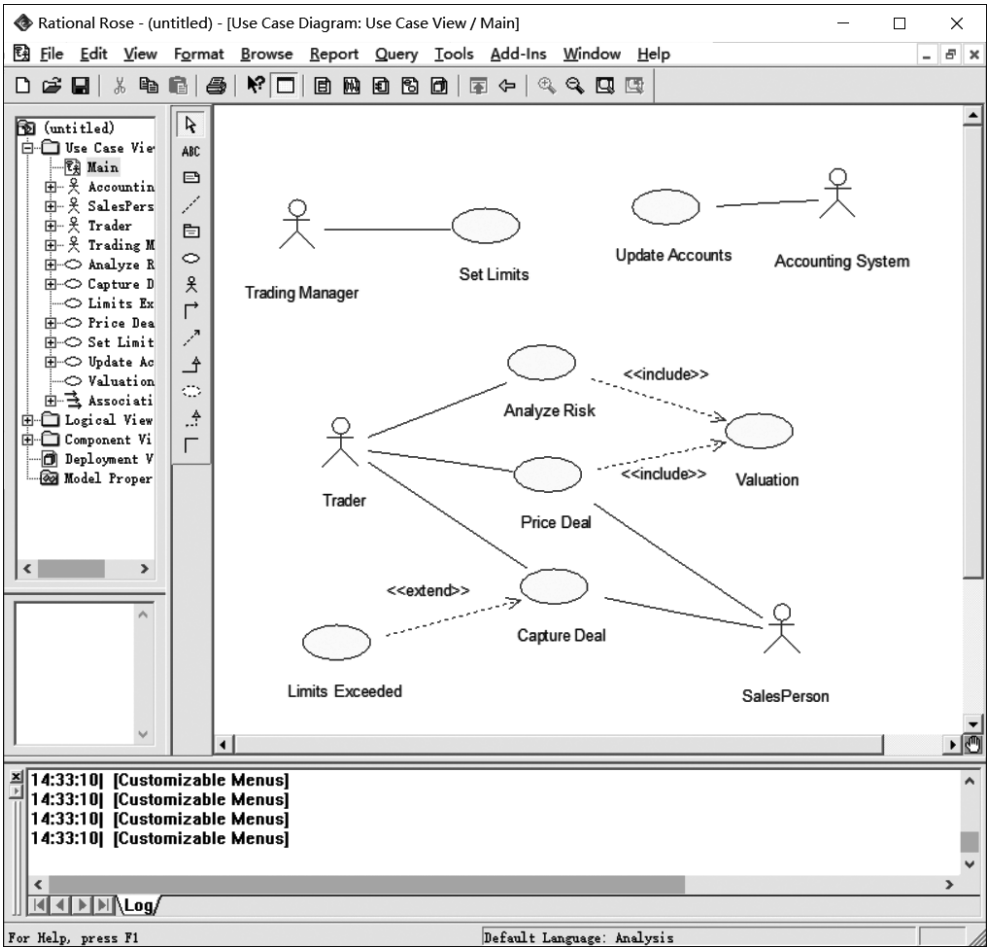


图 3.9　金融贸易系统的用例图

在语义上，使用填充颜色和不使用填充颜色的模型是一样的。

3.6　用例的描述

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，我们只知道该目录标题，并不知道该目录的具体内容是什么。对于 UML 初学者，一个很容易忽视的问题就是缺少用例的描述或用例的描述不完整，往往只是用一个椭圆表示用例。在 3.1 节给出的用例定义中也可以看到，用例是一个“文字描述序列”，是“动作序列的说明”。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。

一般来说，用例采用自然语言描述参与者与系统进行交互时双方的行为，不追求形式化的语言表达。因为用例最终是给开发人员、用户、项目经理、测试人员等不同类型的人

员看的，如果采用形式化描述，对大部分人来说会很难理解。

用例的描述应该包含哪些内容，并没有一个统一的标准，不同的开发机构可能会有不同的要求，但一般应包括以下内容。

(1) 用例的目标。

(2) 用例是怎么启动的？

(3) 参与者和用例之间的消息是如何传送的？

(4) 用例中除了主路径外，其他路径是什么？

(5) 用例结束后的系统状态。

(6) 其他需要描述的内容。

总之，描述用例时的原则是尽可能写得“充分”，而不是追求写得形式化、完整或漂亮。

作为 OOA 文档的一个组成部分，用例的描述应该有一定的规范格式，但目前并没有一个统一的标准。在统一的标准出现之前，人们可以采纳适合于自己的用例描述格式，但不管怎样，在一个开发机构内部应该采用统一的格式。表 3.2 是参考了一些不同的开发机构和 UML 使用者的经验后总结的用例描述格式，可以供初学 UML 者参考。具体使用时可用表格的形式表示，也可以不使用表格形式。

表 3.2　不同的用例描述格式

描　述　项	说　　明
用例名称	表明用户的意图或用例的用途，如“划拨资金”
标识符[可选]	唯一标识符，如“UC1701”，在文档的别处可以用此标识符引用这个用例
用例描述	概述用例的几句话
参与者	与此用例相关的参与者列表
优先级	一个有序的排列，1 代表优先级最高
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，如果其中包含条件，则这些条件必须在访问用例之前得到满足
后置条件	一个条件列表，如果其中包含条件，则这些条件将在用例完成以后得到满足
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
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
被泛化的用例	此用例所泛化的用例列表
被包含的用例	此用例所包含的用例列表
被扩展的用例	此用例所扩展的用例列表
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]	与此用例的开发相关的问题列表
决策[可选]	关键决策记录的列表，以便维护时使用
频率[可选]	参与者访问此用例的频率，如用户是每日访问一次还是每月访问一次

表 3.3 是对用例“管理教学班”的描述。与表 3.2 的模板相比，少了问题、决策、频率 3 个描述项。

表 3.3 用例“管理教学班”的描述

用例名称	管理教学班
标识符	UC1701
用例描述	通过管理教学班功能，编辑制定教学班信息，包括查询和匹配教学班，合并教学班，拆分教学班操作
参与者	教务任务管理员
优先级	1
状态	
前置条件	课程、专业、自然班级、教学进程等信息可用
后置条件	
基本操作流程	1. 打开教学班 [教务任务管理员]：要求打开所有教学班信息。 [系统]：得到教学班信息，返回给用户所有教学班信息。 2. 查询所有匹配的教学班 [教务任务管理员]：要求系统查询和指定教学班匹配的所有教学班，输入指定的教学班信息。 [系统]：按照指定的教学班信息遍历所有教学班，返回并显示给用户所有匹配的教学班结果。 3. 选择及合并目标教学班 [教务任务管理员]：请求合并教学班到目标教学班。 [系统]：系统显示所有待合并教学班。 [教务任务管理员]：选择需要合并的教学班。 [系统]：合并指定的教学班，如果合并失败进入 3a。 4. 拆分教学班 [教务任务管理员]：要求拆分指定的教学班。 [系统]：显示要求用户确认教学班操作窗口。 [教务任务管理员]：如果选择确认拆分操作则继续，否则进入 4a。 [系统]：拆分指定的教学班，如果拆分失败则进入 4b
可选操作流程	备选事件序列： 3a：提示用户合并失败，返回选择及合并目标教学班。 4a：返回拆分教学班。 4b：提示用户拆分失败，返回拆分教学班
被泛化的用例	
被包含的用例	
被拓展的用例	
修改历史记录	

用例描述虽然看起来简单，但事实上它是捕获用户需求的关键一步。很多 UML 初学者虽然也能给出用例的描述，但描述中往往存在很多错误或不恰当的地方，在描述用例

时易犯的错误主要有：

(1) 只描述系统的行为，没有描述参与者的行为。

(2) 只描述参与者的行为，没有描述系统的行为。

(3) 在用例描述中就设定对用户界面的设计的要求。

(4) 描述过于冗长。

例 3.4　下面是一个用例描述的片断。

用例：Withdraw Cash

参与者：Customer

主事件流：

(1) 储户插入 ATM 卡，并输入密码。

(2) 储户按 Withdraw 键，并输入取款数目。

(3) 储户取走现金、ATM 卡并拿走收据。

(4) 储户离开。

上述描述中存在的问题是只描述了参与者的动作序列，而没有描述系统的行为，改进的描述如下。

用例：Withdraw Cash

参与者：Customer

主事件流：

(1) 通过读卡机，储户插入 ATM 卡。

(2) ATM 系统从卡上读取银行 ID、账号、加密密码，并用主银行系统验证银行 ID 和账号。

(3) 储户输入密码，ATM 系统根据上面读出的卡上加密密码，对密码进行验证。

(4) 储户按 Fastcash 键，并输入取款数量，取款数量应该是 5 美元的倍数。

(5) ATM 系统通知主银行系统，传递储户账号和取款数量，并接收返回的确认信息和储户账户余额。

(6) ATM 系统输出现金、ATM 卡和显示账户余额的收据。

(7) ATM 系统记录事务到日志文件。

例 3.5　下面是一个用例描述的片断。

用例：Withdraw Cash

参与者：Customer

主事件流：

(1) ATM 系统获得 ATM 卡和密码。

(2) 设置事务类型为 Withdrawal。

(3) ATM 系统获取要提取的现金数目。

(4) 验证账户上是否有足够储蓄金额。

(5) 输出现金、数据和 ATM 卡。

(6) 系统复位。

上述描述中存在的问题是只描述了 ATM 系统的行为而没有描述参与者的行为，这

样的描述很难理解、验证和修改，改进的描述同例 3.4。

例 3.6 下面是一个用例描述的片断。

用例：Buy Something

参与者：Customer

主事件流：

(1) 系统显示 ID and Password 窗口。

(2) 顾客输入 ID 和密码，然后按 OK 键。

(3) 系统验证顾客 ID 和密码，并显示 Personal Information 窗口。

(4) 顾客输入姓名、街道地址、城市、邮政编码、电话号码、然后按 OK 键。

(5) 系统验证用户是否为老顾客。

(6) 系统显示可以卖的商品列表。

(7) 顾客在准备购买的商品图片上单击，并在图片旁边输入要购买的数量。选购商品后按 Done 键。

(8) 系统通过库存系统验证要购买的商品是否有足够库存。

……(后续描述省略)

上述描述中存在的问题是对用户界面的描述过于详细。对于需求文档来说，详细的用户描述对获取需求并无帮助。改进的描述如下。

用例：Buy Something

参与者：Customer

主事件流：

(1) 顾客使用 ID 和密码进入系统。

(2) 系统验证顾客身份。

(3) 顾客提供姓名、地址、电话号码。

(4) 系统验证顾客是否为老顾客。

(5) 顾客选择要购买的商品和数量。

(6) 系统通过库存系统验证要购买的商品是否有足够库存。

……(后续描述省略)

例 3.7 下面是一个用例描述的片断。

用例：Buy Something

参与者：Customer

主事件流：

(1) 顾客使用 ID 和密码进入系统。

(2) 系统验证顾客身份。

(3) 顾客提供姓名。

(4) 顾客提供地址。

(5) 顾客提供电话号码。

(6) 顾客选取商品。

(7) 顾客确定商品的数量。

(8) 系统验证顾客是否为老顾客。

(9) 系统打开到库存系统的连接。

(10) 系统通过库存系统请求当前库存量。

(11) 库存系统返回当前库存量。

(12) 系统验证购买商品的数量是否少于库存量。

……(后续描述省略)

上述描述中存在的问题是对用例描述过于冗长。可以采用更为简洁的描述方式,如合并类似的数据项(步骤(3)～(5)),提供抽象的高层描述(步骤(9)～(12))等。改进的描述如下。

用例: Buy Something

参与者: Customer

主事件流:

(1) 顾客使用 ID 和密码进入系统。

(2) 系统验证顾客身份。

(3) 顾客提供个人信息(包括姓名、地址、电话号码),选择购买的商品及数量。

(4) 系统验证顾客是否为老顾客。

(5) 系统使用库存系统验证要购买的商品数量是否少于库存量。

……(后续描述省略)

3.7　寻找用例的方法

用例分析步骤可以按下面的顺序进行。

(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。

(3) 把这些系统行为命名为用例。

(4) 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。

(5) 编制每一个用例的脚本。

(6) 绘制用例图。

(7) 区分主事件流和异常情况的事件流,如果需要,可以把表示异常情况的事件流作为单独的用例处理。

(8) 细化用例图,解决用例间的重复与冲突问题。

当然,上述这个顺序并不是固定的,可以根据需要进行一些调整。

采用用例分析法捕获用户的需求,其中一个比较困难的工作是确定系统应该包含哪些用例,以及如何有效地发现这些用例。事实上,在做用例分析时,并没有一个固定的方式或方法来发现用例,而且对同一个系统,往往会同时存在多种解决方案,但其中某些方案会比另一些方案好。与设计和实现阶段相比,需求分析阶段更多的还是依赖于分析人员的个人经验和领域知识。例如,如果某分析人员以前做过类似的系统分析和开发工作,那么以后再做类似的工作时就比较容易,但如果是针对一个全新的领域,往往会觉得很难

入手，这时就需要领域专家的帮助。

下面的这些启发性原则可以帮助分析人员发现用例。

与用户交互。寻找用例的一个途径就是和系统的潜在用户会面、交谈。有可能不同的用户对系统的描述会是完全不同的，即使是同一个用户，他对系统的描述也可能是模糊的、不一致的，这时就需要分析员做出判断和抉择。

把自己当作参与者，与设想中的系统进行交互。问一些问题，例如，系统交互的目的是什么？需要向系统输入什么信息？希望由系统进行什么处理并从它那里得到何种结果？等等，都有助于发现用例。

确定用例和确定参与者不能截然分开。

Jacobson 作为用例方法的提出者，也提出一些原则帮助发现用例，例如通过回答下列问题帮助发现用例。

参与者的主要任务是什么？

参与者需要了解系统的什么信息？需要修改系统的什么信息？

参与者是否需要把系统外部的变化通知系统？

参与者是否希望系统把异常情况的变化通知自己？

随着分析人员开发经验的不断积累，对于如何寻找用例会逐渐形成自己的一套方法，也可以通过与其他人的交流提高自己的分析水平。

3.8 用例图建模实例

随着高校校园网的建设和 Internet 技术的引进，基于校园网和 Internet 的应用系统的开发正在蓬勃发展。教务管理是高校教学管理的一项重要工作，现代化的高校教务管理需要现代化的信息管理系统支持。新世纪背景下，高校教育体制进行了大规模的改革，招生人数逐年增加，教学计划不断更新。在高校日常管理中，教务管理无疑是核心工作，重中之重。其管理模式的科学化与规范化，管理手段的信息化与自动化对于学校的总体发展产生深远的影响，由于管理内容过多，烦琐，处理的过程也非常复杂，并且随着学校人员的增加，教务管理系统的信息量大幅上升，因此往往很难及时准确地掌握教务信息的运作状态，这使得高校教务管理的工作量大幅度增加。另外，随着教育改革的不断深化，教学管理模式也在发生变化，例如，实施学分制、学生自主选课等。这一切都有赖于计算机网络技术和数据库技术的支持，在这样的形势下建立和完善一个集成化的教务管理系统势在必行。

目前，国内高校都开发了自己基于校园网的教务管理系统。由于其教务管理模式不尽相同，不同学校的实际教务管理情况各有自己的特点，因而各高校需要针对自己的教务管理模式和特点建立自己的教务管理系统。本设计是基于某高校的教务管理模式开发的基于校园网的教务管理系统。这样一个系统不仅可以降低工作量、提高办公效率，而且使分散的教务信息得到集中处理，对减轻教务工作负担、提高教务管理水平、实现教务管理的现代化具有重要意义。

建立系统用例模型如下。

1. 确定系统模型的参与者

仔细分析教务管理系统问题描述。在 UML 中，角色代表位于系统之外和系统进行交互的一类对象，本系统中创建主要的角色有以下三类。

(1) 管理员：管理员在教学管理系统中对全体学生进行用户登录、学籍管理、选课管理、教学管理和成绩管理，并且对教师进行登录管理、教学管理和成绩管理。教务处工作人员处理日常的系统维护，例如，维护和及时更新学生，教师信息以及安排选课等。(本章为了简化系统，将教务工作人员和教务系统管理人员统一作为管理员角色。)

(2) 教师：教师根据教务系统的选课安排进行教学，将学生的考试成绩录入此系统。

(3) 学生：学生能够在教务管理系统更改个人信息、进行选课、查询已选课程和考试成绩。

2. 识别用例

用例是系统外部参与者与系统在交互过程中需要完成的任务，识别用例最好的方法就是从分析系统的参与者开始，考虑每一类参与者需要使用系统的哪些功能，如何使用系统，根据教务管理系统的运行流程提取的参与者信息，确定系统分为以下几个用例。

(1) 学生参与者用例：

- 用户登录
- 成绩查询
- 选课管理

……

(2) 教师参与者用例：

- 用户登录
- 成绩管理
- 教学管理

……

(3) 管理员参与者用例：

- 用户登录
- 学籍管理
- 选课管理
- 成绩管理
- 教学管理

……

3. 建立用例图模型

建立三个用例图模型，如图 3.10～图 3.12 所示。

用例描述略。

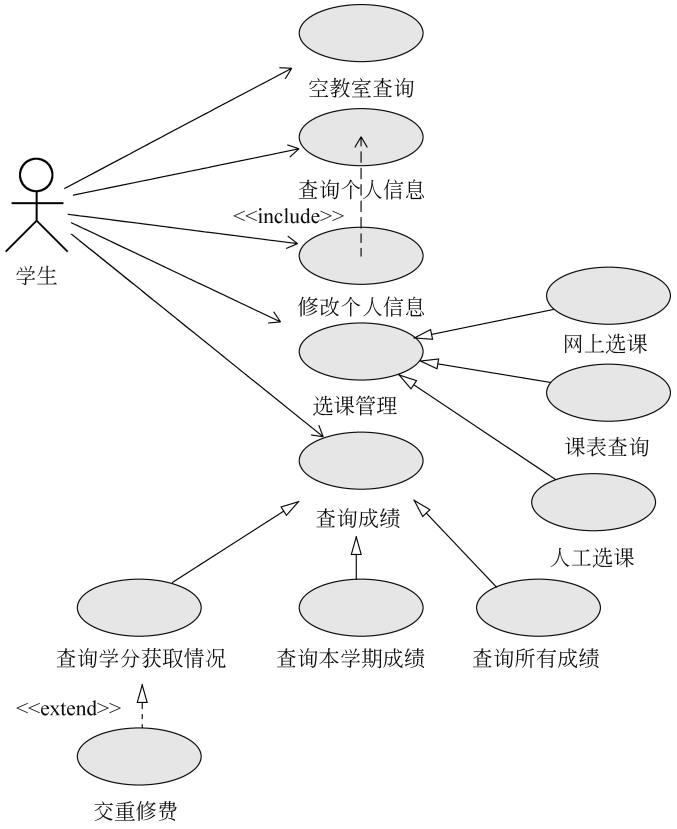


图 3.10　学生角色用例图

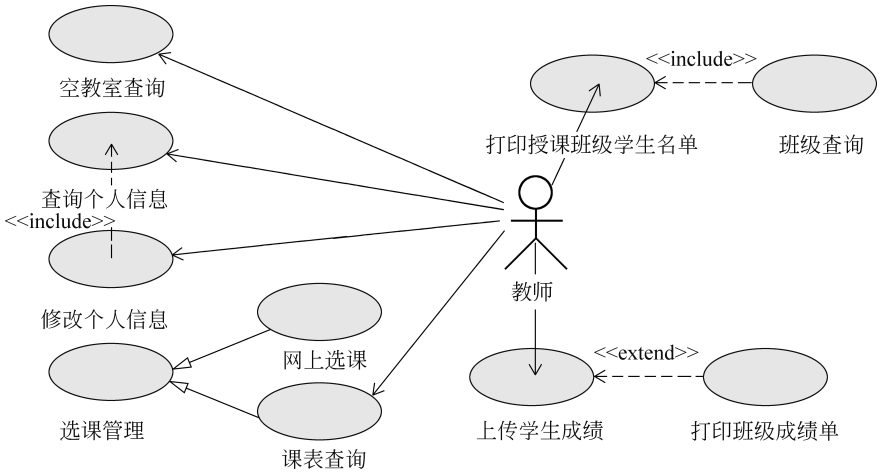


图 3.11　教师角色用例图

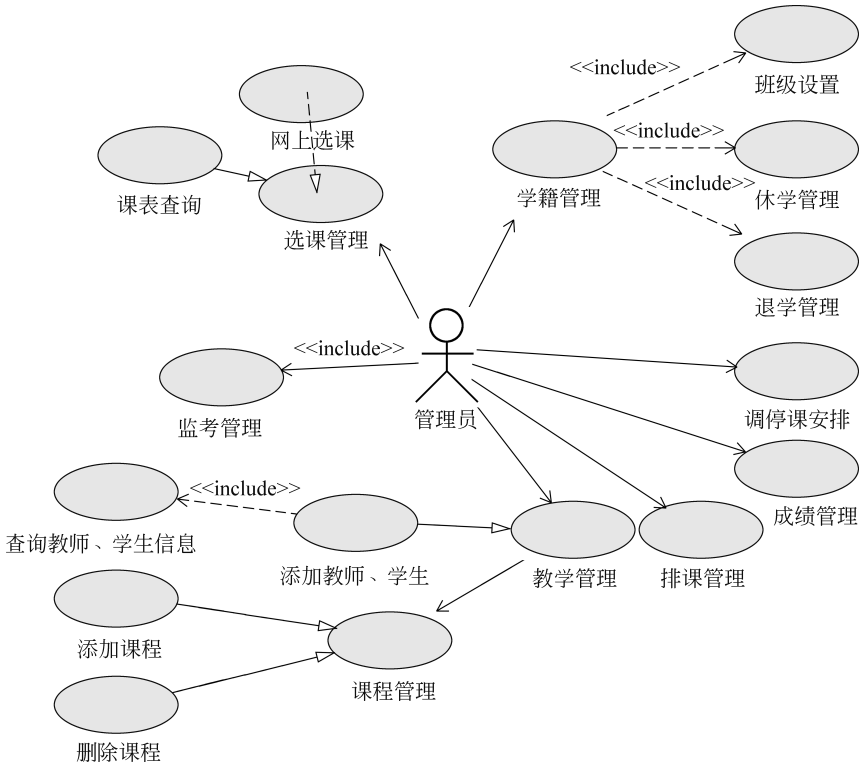


图 3.12　管理员角色用例图

小　　结

1. 用例是 Ivar Jacobson 发明的概念，用例驱动的软件开发方法已得到广泛的认同。

2. 用例是系统、子系统或类和外部的参与者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。

3. 用例命名往往采用动宾结构或主谓结构。

4. 系统需求一般分为功能性需求和非功能性需求两部分，用例只涉及功能性方面的需求。

5. 用例之间可以有泛化关系、包含关系、扩展关系等。

6. 脚本是用例的实例。

7. 参与者是指系统以外的、需要使用系统或与系统交互的东西，包括人、设备、外部系统等。

8. 参与者之间可以有泛化关系。

9. 用例的描述是用例的主要部分。

10. 用例的描述格式没有一个统一的标准，不同的开发机构可以采用自认为合适的格式。

11. 用例分析结果的好坏与分析人员的个人经验和领域知识有很大的关系。