实战OO：用例建模

徐锋／文

引言

有些制作精细的“模型车”不管从外观还是内部结构上都与真车一模一样，但是却不能够像真车那样行驶，缺了什么呢？缺的是每个零件只是“神似”，而非“真是”，换一句话说就是处于静态状态下是相像的，但是无法动起来，无法实现这些零件本该实现的功能，这也就使得模型车无法真正地开起来。

在上一讲，我们一起为“个人图书管理系统”建立了概念模型。通过概念模型的建立，可以帮助开发团队更好地理解系统所涉及的问题领域，对要开发系统所相关的业务知识建立正确的理解。但是仅有概念模型（域模型）是不足以模拟系统的，因而我们到现在为止仅仅打造了一辆“模型车”而已，还不足以构件系统。

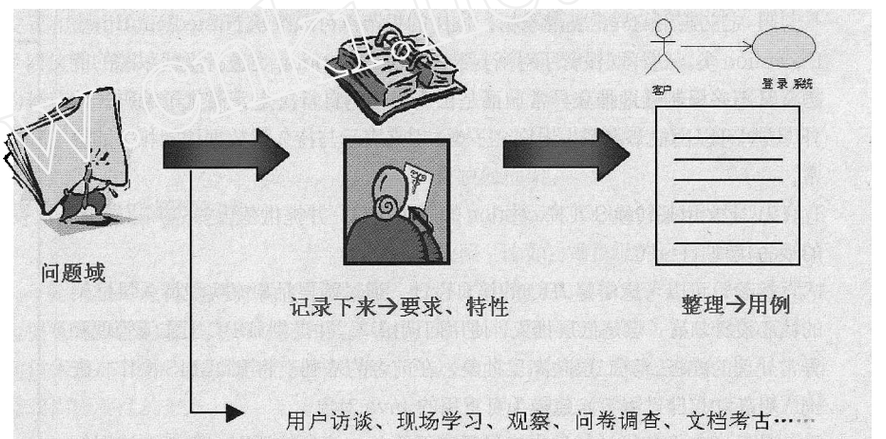
想让“模型车”开起来，最重要的就是建立反映系统行为的动态模型，也就是用例模型。在本章节中，我们就一起来为这辆“模型车”注入动力。

还是从需求开始

在讲解如何构建概念模型的时候，就提到过用例模型与概念模型这一动一静的两个模型都是源于需求的。用例在国外被广泛采用，在国内被大力吹捧，不过由于传入国内是伴随着UML一起，所以许多人就误把用例图当做用例模型，其实不然，用例模型不仅包括用例图，还有更为重要的用例描述。除了这个常见的误解之外，许多人还把用例分析技术当作一种分解技术（甚至有人将其套到结构化分解的思想下，画出的用例图就是一个功能分解的翻版），其实用例分析技术是一种合成技术。

接下来，我们就带着这两个新的观点，一起来看看用例应该是怎么产生的吧，见图一。

从图一中大家应该可以直观地感觉到，用例其实是应用于需求分析阶段的方法，而不是需求捕获阶段的方法。在该例子中，捕获到了如图二要实现的功能特性，我们将其做好相应的编号，以便进行跟踪管理。



图一 用例过程示意图

FEAT01.新增书籍信息
FEAT02.修改已有的书籍信息
FEAT03.书籍信息按计算机类、非计算机类分别建档
FEAT04.录入新书时能够自动按规则生成书号
FEAT05.计算机类与非计算机类书籍采用不同的书号规则
FEAT06.录入新书时如果重名将自动提示
FEAT07.按书名、作者、类别、出版社等关键字组合查询书籍
FEAT08.列出所有书籍信息
FEAT09.记录外借情况
FEAT10.外借状态能够自动反应在书籍信息中
FEAT11.按人、按书查询外借情况
FEAT12.列出所有的外借情况
FEAT13.按特定时间段统计购买金额、册数
FEAT14.所有查询、列表、统计功能应可以单独对计算机类或非计算机类进行

图二 功能特性一览表

接下来，我们就在这个功能特性一览表的基础上开始构建用例模型。

构建用例模型

用例是什么呢？Ivar Jacobson是这样描述的：“用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例定义一组用例实例。”

首先，我们从定义中得知用例是由一组用例实例组成的，用例实例也就是常说的“使用场景”，就是用户使用系统的一个实际的、特定的场景。其次，我们可以知道，用例应该给参与者带来可见的价值，这点很关键。最后，我们得知，用例是在系统中的。

构建用例模型需要经历：识别参与者、合并需求获得用例、细化用例描述三个阶段。

1、识别参与者

参与者（Actor）是同系统交互的所有事物，该角色不仅可以由人承担，还可以是其它系统、硬件设备、甚至是时钟：

1）其它系统：当你的系统需要与其它系统交互时，如在开发ATM柜员机系统时，银行后台系统就是一个参与者；

2）硬件设备：如果你的系统需要与硬件设备交互时，如在开发IC卡门禁系统时，IC卡读写器就是一个参与者；

3）时钟：当你的系统需要定时触发时，时钟就是一个参与者，如在开发Foxmail中的“定时自动接收”功能时，就需要引入时钟作为参与者。

要注意的是，参与者一定在系统之外，不是系统的一部分。通常可以通过以下问题来整理思路：

- 谁使用这个系统？
- 谁安装这个系统？
- 谁启动这个系统？
- 谁维护这个系统？
- 谁关闭这个系统？
- 哪些其他的系统使用这个系统？
- 谁从这个系统获取信息？
- 谁为这个系统提供信息？
- 是否有事情自动在预计时间发生？

而在我们要开发的这个小型的“个人图书管理系统”中，参与者只有一个，那就是“图书管理员”。

2、合并需求获得用例

将参与者都找到之后，接下来就是仔细地检查参与者，为每一个参与者确定用例。而其中的依据主要可以来源于已经获取得到的“特征表”。

1）将特征分配给相应的参与者

首先，要将这些捕获到的特征，分配给与其相关的参与者，以便可以针对每一个参与者进行工作，而无遗漏。而在本例中，所有的特征就是与一个参与者（即图书管理员）相关的。

2）进行合并操作

在合并之前，我们首先还要明确为什么要合并，知道了合并的目的，也就会使得我们选择正确的合并操作。一个用例就是一个对参与者来说可见的价值结果，因此合并的根据就是使得其能够组合成为一个可见的价值结果。

由于本例相对较小，所以合并工作相对是比较简单的，如果是较大的项目则需要花费更多的时间，不过原理是一样的，主要是工作更加复杂一些而已。

合并后，将产生用例，而用例的命名应该注意采用“动词（短语）+名词（短语）”的形式，而且最好能够对其进行编号，这也是实现跟踪管理的重要技巧，通过编号可以将用户的需求落实到特定的用例中去。

最后，有阅读过用例相关资料、文献的读者可能会对本例中合并出来的用例有置疑的地方，请稍安勿躁，我将在后面为你解除这个疑问。

3）绘制成用例图

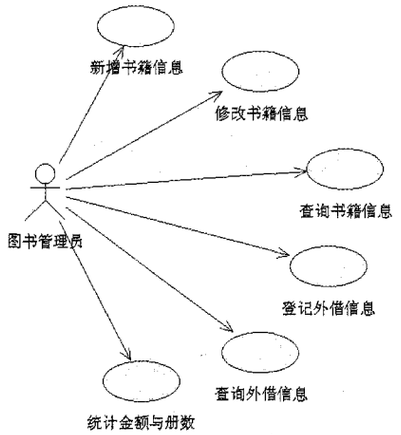
好了，最后我们就将识别到的参与者，以及合并生成的用例通过用例图的形式整理出来，以获得用例模型的框架，也算是得到一个中间的成果，如图三。

根据分析，修改书籍信息的操作是在查询书籍信息操作之后才发生的，并不需要单独地为其创建用户界面，因此

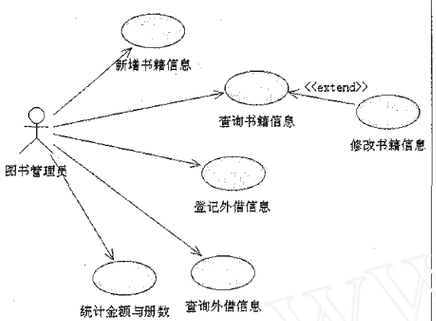
表一 合并生成用例

特征	用例
FEAT01.新增书籍信息 FEAT03.书籍信息按计算机类、非计算机类分别建档 FEAT04.录入新书时能够自动按规则生成书号 FEAT05.计算机类与非计算机类书籍采用不同的书号规则 FEAT06.录入新书时如果重名将自动提示	UC01.新增书籍信息
FEAT02.修改已有的书籍信息	UC02.修改书籍信息
FEAT07.按书名、作者、类别、出版社等关键字组合查询书籍 FEAT08.列出所有书籍信息 FEAT14.所有查询、列表、统计功能应可以单独对计算机类或非计算机类进行	UC03.查询书籍信息
FEAT09.记录外借情况 FEAT10.外借状态能够自动反应在书籍信息中	UC04.登记外借信息
FEAT11.按人、按书查询外借情况 FEAT12.列出所有的外借情况 FEAT14.所有查询、列表、统计功能应可以单独对计算机类或非计算机类进行	UC05.查询外借信息
FEAT13.按特定时间段统计购买金额、册数 FEAT14.所有查询、列表、统计功能应可以单独对计算机类或非计算机类进行	UC06.统计金额和册数

为了表现其间的关系，可以适当地修改为如图四所示。



图三 最初的用例图



图四 修改后的用例图

千万不要以为到此，用例分析就结束了。这仅仅是一个好的开端，接下来的工作才是最为重要的一环，也是用例发挥作用的关键。

3、细化用例描述

接下来，我们以用例UC01“新增书籍信息”为例，说明如何细化用例描述。

1）搭框架

首先，我们根据特性表和前面的分析，先完成一个框架：

```
1.用例名称：
    新增书籍信息（UC01）
2.简要说明：
    录入新购书籍信息，并自动存储建档。
3.事件流：
    3.1 基本事件流
    3.2 扩展事件流
4.非功能需求
5.前置条件
    用户进入图书管理系统。
6.后置条件
    完成新书信息的存储建档。
7.扩展点
    无
8.优先级
    最高（满意度 5，不满意度 5）
```

图五 用例描述框架

注：以下是每个部分写作时的注意点

1）用例名称：应该与用例图相符，并写上其相应的编号；

2）简要说明：对该用例对参与者所传递的价值结果进行描述，应注意语言简要，使用用户能够阅读的自然语言。

3）前置条件：是执行用例之前必须存在的系统状态，这部分内容如果在现在不容易确定可以在后面再细化。

4）后置条件：用例执行完毕系统可能处于的一组状态，这部分内容如果在现在不容易确定也可以在后面再细化。

5）扩展点：如果包括扩展或包含用例，则写出扩展或包含用例名，并说明在什么情况下使用。而在本例中，用例图里没有相应的内容，因此可以直接写无。如果有，则应该在编写事件流的同时进行编写。

6）优先级：说明用户对该用例的期望值，可以为今后开发时制定先后顺序。可以采用满意度/不满意度指标进行说明，其中满意度的值为0-5，是指如果实现该功能，用户的满意程度；而不满意度的值也为0-5，是指如果不实现该功能，用户的不满意程度。

```
……
3．事件流：
    3.1 基本事件流
    1）图书管理员向系统发出“新增书籍信息”请求；
    2）系统要求图书管理员选择要新增的书籍是计算机类还是非计算机类；
    3）图书管理员做出选择后，显示相应界面，让图书管理员输入信息，并自动根
       据书号规则生成书号；
    4）图书管理员输入书籍的相关信息，包括：书名、作者、出版社、ISBN 号、开
       本、页数、定价、是否有CDROM；
    5）系统确认输入的信息中书名未有重名；
    6）系统将所输入的信息存储建档。
    3.2 扩展事件流
    5 a ）如果输入的书名有重名现象，则显示出重名的书籍，并要求图书管理员选
         择修改书名或取消输入；
    5 a 1 ）图书管理员选择取消输入，则结束用例，不做存储建档工作；
    5 a 2 ）图书管理员选择修改书名后，转到5 ）
4．非功能需求
    无特殊要求
……
```

图六 用例描述的血肉—事件流

对于任何一个用例，在分析阶段都应该将其框架用例描述建立起来。

2）填血肉

在这个阶段的主要工作就是将事件流进行细化，在实际的开发工作，要不要对一个用例进行细化、细化到什么程度主要根据你项目迭代的计划来决定。如图六。

在编写事件流的时候，应该注意：

1）使用简单的语法：主语明确，语义易于理解；

2）明确写出“谁控制球”：也就是在事件流描述中，让读者直观地了解是参与者在控制还是系统在控制；

3）从俯视的角度来编写：指出参与者的动作，以及系统的响应，也就是第三者的角度；

4）显示过程向前推移：也就是第一步都有前进的感受（例如，用户按下tab键做为一个事件就是不合适的）；

5）显示参与者的意图而非动作（光有动作，让人不容易直接从事件流中理解用例）；

6）包括“合理的活动集”（带数据的请求、系统确认、更改内部、返回结果）；

7）用“确认”而非“检查是否”，例如“系统确认所输入的信息中书名未有重名”；

8）可选择地提及时间限制；

另外，事件流的编写过程也是可以分阶段，迭代进行的，对于优先级高的用例花更多的时间，更加的细化；对优先级低的用例可以先简略地将主要事件流描述清楚留到以后。另外，对于一些事件流较为复杂的，可以在用例描述中引用顺序图、状态图、协作图等手段进行描述。

而在非功能需求小节中，主要对该用例所涉及的非功能性需求进行描述。由于其通常很难以在事件流中进行表述，因此单列为一小节进行阐述。这些需求通过包括法律法规、应用程序标准、质量属性（可用性、可靠性、性能、支持性等）、兼容性、可移植性，以及设计约束等方面的需求。在这些需求的描述方面，一定要注意使其可度量、可验证，否则就容易流于形式，形同摆设。

3）补缺漏

在填血肉阶段要注意加强与用户的沟通，写完后需要与客户进行验证，然后不断地进行补缺漏，以保证用例描述完整、清晰、正确。

限于篇幅，在此就无法对每个用例一一写出其描述，有兴趣的读者可以自行以其它用例为例，进行用例细化描述。

用例的粒度

用例作为一种有效的需求分析技术，近几年来被软件开发业界广泛采用和认同。虽然用例的形式比较简单，规则也不复杂，但正是由于这种自由性，要得心应手地灵活应用和发挥并不是一件很容易的事。其中最大的一个不容易把握的地方，就是用例的粒度，也就是多大才算是一个好的用例。

1、思辨“四轮马车”

在前面，我们通过合并特征获得了用例，在那里就留下了一个疑问。这个疑问其实就与用例的粒度相关。那就是笔者合并生成的用例中包括了“新增书籍信息”、“修改书籍信息”、“查询书籍信息”，这三个刚好是违反了一个大名鼎鼎的错误——四轮马车！在新增、修改、查询、删除四个操作中，就引入了三个，很多大师都建议将其归结为一个——“管理书籍信息”。

那么，笔者又为什么要犯这个明知故犯的错误呢？其实，在大量的应用中，都会涉及到新增、修改、查询、删除的动作，因此如果在分析时把这些东西全都整理为一个用例，就会使得用例过多，复杂度太大，模型不够抽象。而其实在具体的处理中，还是会将其作为子用例看待，用扩展的方式来描述出来。而在本例中，系统相对简单，这几个功能将其独立出来并没有什么影响，而且这几个功能属于系统的重要核心功能，因此笔者认为这样处理并无不妥。当然这么说，并不讲四轮马车错误的总结不对，四轮马车的本意应该是指对非核心实体无须过度展开，如图书馆管理系统中的“管理会员信息”功能就不应该过度展开；另一方面如果系统较大，也会使得用例的数量过多，大大提高了复杂度。

其实，我也想表达出的一种观点是，用例的粒度其实是一个“度”的问题，而根据中国传统的中庸之道，度无绝对，也就是说，找不到一个绝对值来说明到什么程度是对的，什么程度是错的。因此，大家完全可以不要为此所困，而且应该根据自己的需要来决定。不过，其中有一个很重要的东西，那就是不管用例的粒度大还是小，都需要符合“可见的价值结果”这一原则，否则就将违背了用例的思想，就无法获得用例所带来的益处。

例如，“财务管理”，或故意为了符合用例的命名规则，而改成类似“管理财务信息”的名称，这作为一个用例，其实是违背了用例的思想，因为它无法符合“可见的价值结果”的原则，它太大了，这样使用用例的人其实还在用“功能分解”的思路在理解系统。

再如：“输入支付信息”，这作为一个用例，分析后就会发现它只是一个步骤，并不能够传达“可见的价值结果”，它太小了，这是一个过度使用用例的例子。

2、如何整理用例的层次

在实践中，经常看到实践者忍不住将用例分成几个层次，先找到一些像“财务管理”这样的所谓的大用例，然后在后面用include或extend关系引入所谓的小用例，建立所谓的层次结构。其实这样的做法并不正确，应该通过包来表现用例层次，如果用例太多，就应该归类整理到一个个包里。下面就针对本例进行整理，当然该系统其实是无需进行这一步的，这里只是帮助大家理解：

1）书籍管理：包括“新增书籍信息”、“修改书籍信息”、“查询书籍信息”；

2）外借管理：包括“登记外借信息”、“查询外借信息”；

3）数据统计：包括“统计金额和册数”。

如果你使用Rational Rose绘制模型的话，可以先画三个包，然后在三个包中分别绘制出相应的子图即可。如果你是使用其它工具，如Visio、纸和笔，那么你可以在用例图上将属于一个包的用例框在一起，并在框上写上包的名字即可。

小结

本节中我们讲解如何从捕获而来的需求细节，整合成为用例，然后通过一个用例，说明了如何进行用例描述，相信对大家理解和运用用例分析技术有一定的帮助。心动的读者，请一定结合自己的工作进行实践，才能够有效地运用自如。

用例模型也是随着项目开发的进度而不断演化的，我们现在有了一动一静两个模型，接下来就可以开始朝着设计方向挺进了，下一讲再见。

作者介绍：徐锋，高级程序员，系统分析员，CSAI顾问。自1992年开始被电脑神奇的魅力深深吸引，一发不可收拾。十余年来，涉猎面广泛，只叹自己仍博而不精。现致力于需求工程、系统分析与设计、过程改进等领域的研究与探索。