实战OO：为问题域建模

徐锋／文

总序

“类是一个框，什么都往里装”，这就是很多程序员所编写的程序代码给我留下的印象。从这些代码中，我也终于找到了那些批判面向对象技术言论的根源：其实国内太多程序员一边在使用Java、C++等OO语言，而一边则还在沿用结构化开发方法的思想进行设计。而当这两种东西发生矛盾与冲突时，就将问题转而怪罪于面向对象技术。

而笔者认为，由于世界对软件系统的需求日益增长，变化也越来越快，软件开发技术的发展方向已悄悄地从“提升被开发系统的执行效率”转变成为“提升开发效率”。面向对象技术降低了解决方案域（计算机）与问题域之间的差别，提供了良好的复用机制，能够更加有效地提高软件开发效率，完全顺应了软件开发技术的发展方向。

在“实战OO”系列文章中，笔者将通过一个简单的个人图书管理系统，让大家亲身体验一下正确的面向对象分析与设计过程，从而帮助大家更好的驾驭OO技术，提升自己的“内力”。

注：为了使大家将精力集中于OO技术的应用，在本系列文章中不涉及项目计划、管理等内容。另外，如果大家被需求调研、分析、设计等词汇搞得一头雾水的话，那么就请记住：“需求调研是了解问题，分析是定义问题，设计则是解决问题。”

从需求开始

客户通常只能够或者是只会对其所需的软件系统提出一些要求，而且这些要求通常都相对比较模糊。甚至在笔者自己身上也不例外，首先只是整理出了一个如下的需求描述：

> 笔者是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但不能够删除记录。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时限进行统计。

尽管如此，我们还是可以从这样的需求描述中，对待开发的软件系统建立一个初步的认识。我们的工作就是从此开始，将通过一系列的分析活动，使得对用户的需求更加清晰，对需要解决的问题更加明确。具体来说，在分析阶段通常包括以下两个工作任务：建立一个反应问题域静态关系的概念模型，通常使用类图来表示；建立一个反应系统行为的动态模型，即用例模型。

建立概念模型的目的是帮助开发团队理解问题领域的各种概念、各种名词、以及它们之间的各种关系。而建立用例模型的目的则是帮助开发团队理解客户对系统的各种功能需求，这两方面的工作，将帮助开发团队定义问题，也是分析工作的主要内容。

建立概念模型

“问题域”是指一个包含现实世界事物与概念的领域，这些事物和概念与所设计的系统要解决的问题有关。而建立概念模型，又称为问题域建模、域建模，也就是找到代表那些事物与概念的“对象”。

接下来，我们就一起开始为“个人图书管理系统”建立一个概念模型：

发现类

发现类的方法有很多种，其中最广泛应用的莫过于“名词动词法”，下面我们就采用该方法开始问题域建模的第一步。

注：名词动词法其主要规则是从名词与名词短语中提取对象与属性；从动词与动词短语中提取操作与关联；而所有格短短语通常表明名词应该是属性而不是对象。

1）找到备选类

首先，我们可以逐字逐句地阅读上面那段需求描述，并将其中的所有名词及名词短语列出来，我们可以得到如下

的备选类列表：

笔者	人	家里	书籍	朋友	
个人图书管理系统		基本信息(书籍的)	计算机类	非计算机	
书名	作者	类别	出版社	关键字	功能
新书籍	系统	规则	书号	信息	记录
外借情况	外借情况列表		购买金额	册数	特定时限

图一 列出所有的名词及名词短语

2）决定候选类

很显然，并不是每一个备选类都是合适的候选类，有些名词对于要开发的系统来说并无关紧要，甚至是系统之外的；而有些名词表述的概念则相对较小，适合于某个候选类的属性。因此，我们需要对备选类进行一番筛选，将这些不合适的排除掉。

- “笔者”、“人”、“家里”很明显是系统外的概念，无须对其建模；
- 而“个人图书管理系统”、“系统”指的就是将要开发的系统，即系统本身，也无须对其进行建模；
- 很明显“书籍”是一个很重要的类，而“书名”、“作者”、“类别”、“出版社”、“书号”则都是用来描述书籍的基本信息的，因此应该作为“书籍”类的属性处理，而“规则”是指书号的生成规则，而书号则是书籍的一个属性，因此“规则”可以作为编写“书籍”类构造函数的指南；
- “基本信息”则是书名、作者、类别等描述书籍的基本信息统称，“关键字”则是代表其中之一，因此无需对其建模；
- “功能”、“新书籍”、“信息”、“记录”都是在描述需求时使用到的一些相关词语，并不是问题域的本质，因此先可以将其淘汰掉；
- “计算机类”、“非计算机类”是该系统中图书的两大分类，因此应该对其建模，并改名为“计算机类书籍”和“非计算机类书籍”，以减少歧义；
- “外借情况”则是用来表示一次借阅行为，应该成为一个候选类，多个外借情况将组成“外借情况列表”，而外借情况中一个很重要的角色是“朋友”——借阅主体。虽然到本系统中并不需要建立“朋友”的资料库，但考虑到可能会需要列出某个朋友的借阅情况，因此还是将其列为候选类。为了能够更好地表述，将“外借情况”改名为“借阅记录”，而将“外借情况列表”改名为“借阅记录列表”；
- “购买金额”、“册数”都是统计的结果，都是一个数字，因此不用将其建模，而“特定时限”则是统计的范围，也无需将其建模；不过从这里的分析中，我们可以发现，在该需求描述中隐藏着一个关键类——书籍列表，也就是执行统计的主体。

通过以上的分析，我们就可以得到一个候选类列表：

书籍	计算机类书籍	非计算机类书籍
借阅记录	借阅记录列表	书籍列表

图二 候选类列表

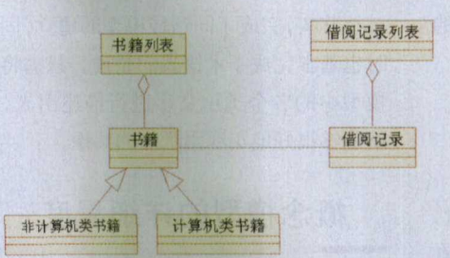
> 注：在使用“名词动词法”寻找类的时候，很多团队会在此耗费大量的时间，这样很容易迷失方向。其实我们在此主要的目的是对问题域建立概要的了解，无需太过咬文嚼字。

确定类之间的关联

通过上面的工作，我们从需求描述中找到了6个与问题域紧密相关的类，接下来首要的任务就是理清类之间的层次关系。

很明显，我们可以发现“计算机类书籍”、“非计算机类书籍”与“书籍”之间是继承关系；而“书籍列表”则是由多个“书籍”组成的，“借阅记录列表”是由多条“借阅记录”组成的。另外，我还可以发现“借阅记录”是与“书籍”关联的，离开“书籍”，“借阅记录”将失去意义。

为了反映和记录这些类之间的关联关系，就可以使用UML中的类图将其记录下来，如下图所示：



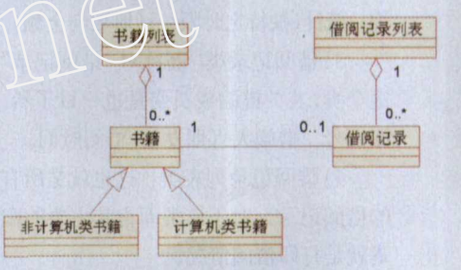
图三 最初的概念模型

但是，上图并无法将关联关系的细节信息传递出来。例如一本书可以有几条借阅记录？书籍列表指的是多少本书籍？这些问题需要进一步分析，并修改上面所列出的类图。

1）系统应用于个人藏书管理，因此每本书都是唯一的，没有副本。其要么被借出去，要么未被借出，因此对于每一本书籍来说，要么没有借阅记录，要么也只有一条借阅记录。

2）所有的书籍组成书籍列表，借阅记录列表则也是由所有的借阅记录组成的。

通过上面的分析，我们可以将得到的信息补充到类图上：



图四 加上关联描述的概念模型

这样，我们就对所有问题域中的各个类之间的层次结构关系、协作关系有了一个完整的了解与认识。而对于较大的系统而言，还可以在此基础上对一些关联度大的部分类合成一个包，以便更好的抽象系统。

> 注：例如，在本例中可以将“书籍列表”、“书籍”、“非计算机类书籍”、“计算机类书籍”合成一个包，而将“借阅记录”与“借阅记录列表”合成一个包。不过本例比较简单，类也相对较少，因此无需进行分解。

为类添加职责

当我们找到了反应问题域本质的主要概念类，而且还理清它们之间的协作关系之后，我们就可以为这些类添加其相应的职责。什么是类的职责呢？它包括以下两个主要内容：

- 类所维护的知识；
- 类能够执行的行为。

相信大家从上面的两句中，马上会

想到类的成员变量（也称为属性）和成员方法吧！是的，成员变量就是其所维护的知识，成员方法就是其能够执行的行为。

在本阶段，我们就是根据需求描述的内容，以及与客户的简单沟通将主要类的主要成员变量和成员方法标识出来，以便更好的理解问题域。

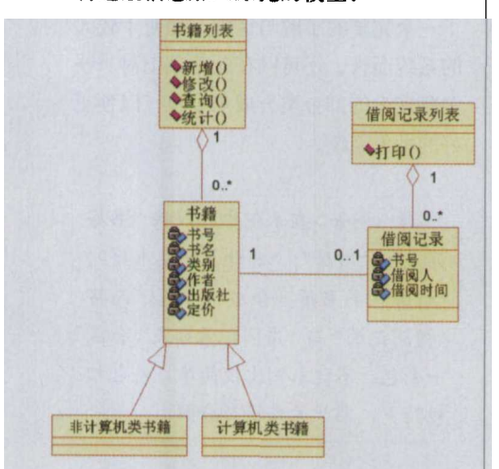
1）书籍类：从需求描述中，我们已经找到了描述书籍的几个关键成员变量，即书号、书名、类别、作者、出版社；同时从统计的需要中，我们可以得知“定价”也是一个关键的成员变量。

2）书籍列表类：书籍列表就是全部的藏书列表，对于该类而言其主要的成员方法是新增、修改、查询（按关键字查询）、统计（按特定时限统计册数与金额）。

3）借阅记录类：而针对“借阅记录”这个类，其关键的成员变量也一目了然，即书号、借阅人（朋友）、借阅时间。

4）借阅记录列表类：这也就是所有的借阅记录，对于该类而言其主要的职责就是打印借阅情况。

通过上面的分析，我们对这些概念类的了解更加深入，可以重新修改类图，将这些信息加入原先的模型：



图五 加上职责描述的概念模型

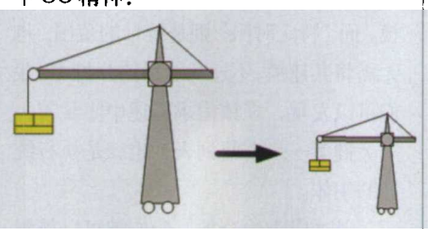
这样我们就可以对该系统所涉及的问题领域有一个完整的理解了。

概念模型的意义

看到这里，大家可能会有一个感觉，怎么一上来就画类图了？难道这不是直接从需求过渡到设计了吗？其实并不是这样。接下来我就从两方面来说明一下这个问题。

面向对象的视角

首先，我们结合下图来再次领会一下OO精神：



图六 这就是OO的思想

在结构化的理论基础下，我们会将应用分解成为一个个功能模块、子功能模块、功能接口等，它完全与现实问题域的东西没有具体的联系。而从上图我们可以看出，使用OO的思想所建立的系统模型就是对问题域的完整、直接的映射。也就是从现实世界中抽象出一个模型，然后在计算机中实现出来。

因此，首先从现实的问题域中找到最有代表性的概念对象，并将其整理成为类，是一件很有意义的事，也是体现了OO思想的关键活动。

开发团队的需要

长期以来，软件需求的问题通常的根源是开发团队无法对客户所处的问题域有深入的了解，只从解决领域的角度思考。而概念模型的建立，其主要的作用是帮助开发团队能够对问题域有一个全貌式的了解，这必将能够缓解这一矛盾。

问题域模型虽然使用了设计时常用的类图，但此时其捕捉的类是反馈问题域本质内容的信息，因此还是属于分析问题、了解问题的活动。

其实，完成了问题域模型的建立，同时也基本完成了术语表的整理。只需将模型中的各个类用文字进行描述出来，就能够很好地实现团队内的共识。

概念模型的详细程度

那么，概念模型到底应该详细到什么程度呢？有些OO的书中建议只列出类以及类之间的主要关联关系，不要对关联关系进行描述，更不要体现类的职责；而有些OO的书则认为应该将这些东西都列出来。其实干巴巴地讨论这个问题是没有任何意义的，我认为：

1）概念模型的目的是让开发团队对问题域建立一个全貌式的了解，并作为今后进一步深化建立基础，因此不妨取中庸之道，将需求描述中所谈到的主要内容都反馈到概念模型中去，而无需顾及其是关联关系还是职责描述；

2）概念模型是在开发过程中生产出来的第一个系统的静态模型，随着开发活动的推进，该模型将会随之加入新内容、改去旧内容，逐渐完善，演变完整。

总之，模型不是我们要生产的目标产物，而且过程中的一个辅助工作，只要能够利用其帮助团队更好的开发，详细也罢、简约也罢，都是好模型。

结言

谈到这里，相信大家对如何建立概念模型，为什么要建立概念模型有了一个认识。通过概念模型的建立，开发团队将会对要开发的系统所涉及的问题域有一个全貌性的了解。所以接下来，则应该根据需求描述、概念模型开始进行“用例分析”，建立用例模型，从而将需求整理归纳成为指导全开发过程的“软件需求规格说明书”。

如果说概念模型是“帮助开发团队了解客户所处的世界”，那么用例模型则是“帮助开发团队明白客户想解决什么问题”。在有些OO的教程中认为应该先建立用例模型，再建立概念模型（域模型）；有的则相反。其实我认为顺序并不重要，在实际的开发过程中，这两个模型将会呈现迭代发展的状态，互相启发、互相补充。

在下一篇文章中我们将一起为“个人图书管理系统”建立用例模型，再后面的文章则将基于概念模型、用例模型这一静一动两个模型，进行设计、编码。