实战OO：交互建模

徐锋／文

引言

在上一期中，我们在用例描述、域模型的基础上，通过Robustness分析工具，更进一步地理解了每一个用例的处理流程。而且在域模型的基础上，通过引入与设计相关的边界对象、控制对象，充实了域模型中类的属性与方法，进一步逼近了解决方案，也就是有效地跨出设计的第一步。

但是由于Robustness分析所固有的灵活性，得到的结果并不是很严密，还不足以对编码工作提供足够完整的信息，而且还会带来一定的歧义性。因此我们需要更进一步，更加细致地、结合代码实现地进行详细设计，这也就是交互建模所要解决的问题。

在许多UML的书籍中，都有关于交互建模的介绍，但许多初学者都会感到从用例模型演化到交互模型相当困难，笔者也不例外地经历过这样的困扰，最后还是在Robustness分析的帮助下走出了困境。而本篇的主要内容就是告诉大家，如何从Robustness分析的结果中更好地生成交互模型，从而完成重要的详细设计工作。

浅析交互模型

在前面的文章中，我们已经说过，在面向对象的视角里，整个系统是由一系列的对象，以及对象之间的交互与协作构成的。在域建模阶段我们一起找到了系统的最核心的业务类，则在用例建模阶段我们又从使用者的角度对系统进行了梳理，并通过Robustness对使用者的使用场景（一个用例的实例）进行的具体的分析，从而理解了系统需要做什么，也找到了更多的与解决方案相关的设计类。但我们并没有完整地捕获出这些类的行为、责任以及它们之间的交互，而这些正是系统运行的机制。

而交互建模，正是要通过寻找对象之间的交互关系，从而进行“行为分配”。正如Ivar Jacobson所说的的：“只有在所有的用例为所有事件进程建立了交互模型之后，才可以确定已经发现系统所需的每个对象所扮演的角色，以及它们的责任。”

在UML规范中，交互模型包括两种不同的表现形式：

1）一种是强调顺序的**顺序图**（Sequence diagram），读者可以从中可以很容易地看出事件发生的次序；

2）另一种是强调组织的**协作图**（Collaboration diagram），它通过使用布局图指明了各个对象之间是如何静态相连的。

由于使用这两种不同的表现形式构建出来的模型是等价的，因此它们之间是十分容易相互转换的，如果你使用了Rational Rose进行建模的话，那么你就可以更加深刻地体会到这一点。根据我的实际使用经验，建议大家首选顺序图，为每个用例绘制一个相应的顺序图，除非用例实在是很简单之外。而只在需要描述一个关键的交互过程时，选用协作图。紧接下来，我们就结合我们的实例例子一起构建交互模型。

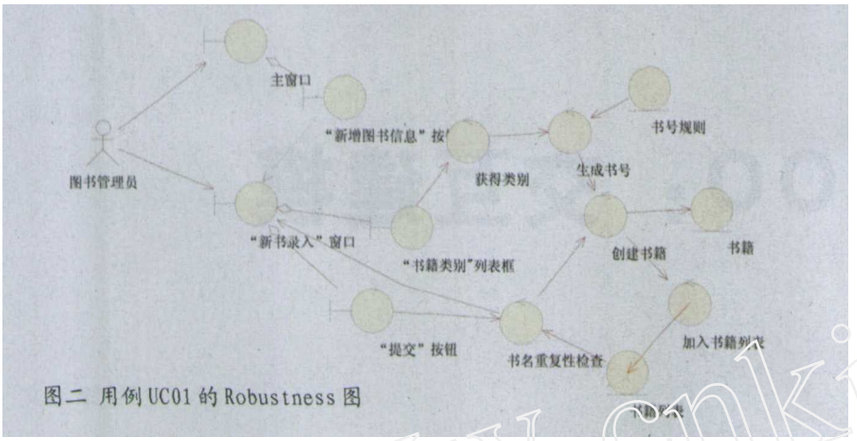
构建交互模型

我们前面提到，交互模型也是针对每个用例而言的，是在用例描述和Robustness分析的结果的基础上进行的。因此，我们还将以用例UC01为蓝本，来说明如何构建交互模型。我的习惯是将用例描述中的基本事件流与扩展事件流部分，以及Robustness分析的结果打印出来，以便在设计时参考。而且这也方便了设计时团队成员之间的交流，可以获得较好的效果，建议读者采用。

为了方便读者，在此将UC01的事件流描述及Robustness图（图二）再罗列如图一所示。

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
5a）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；
5a1）图书管理员选择取消输入，则结束用例，不做存储建档工作；
5a2）图书管理员选择修改书名后，转到5）

图一 用例UC01事件流描述

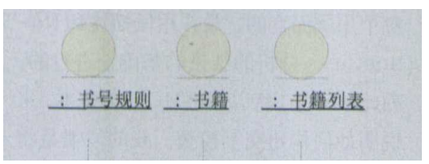


图二 用例UC01的Robustness图

引入实体对象

进行交互建模的第一步，就是将Robustness图中的实体对象找出来，然后如图三所列示，罗列出来，并且在每个对象下面加上一条垂直的虚线，用来表示对象的生命线。

细心的读者会发现，这些对象其实通常也是域模型（类图）中的某个类的一个实例。要注意的是，我们在每个对象的名称前面加上了“：”，根据UML的约定，“：”前面写类名，后面则写对象名。



图三 引入实体对象

由于交互图是用来描述运行状态时的情况，因此是对象，而非类。

引入边界对象和参与者

当我们将需要的实体对象都放置在图上之后，第二步就是将边界对象找出来，然后也罗列出来，要注意的是，由于通常使用者是通过边界类来访问系统的，因此消息的始发方向通常是边界类，因此我们需要将其放置在左边，如图四所示。

注：为了描述简单，我们在这里将无关大局的边界类“主窗口”省略，有兴趣的读者可以自行补充。

图四 引入边界对象与参与者

我们在交互模型中引入参与者的主要目的就是为了让绘制出来的顺序图能够更加清晰、直观，让读者更好地了解使用者与系统的交互关系。

引入控制对象

当我们将参与者、边界对象、实体对象一一摆好之外，就需要结合Robustness图中的控制对象，以及用例描述中的事件流来绘制对象之间的消息传递了。参照事件流描述，我们可以得到如下页图五所示的顺序图。

如下页图5所示的交互模型是如何得到的呢？当然是从用例中的事件流描述和Robustness图中导出来的。

1）首先，图书管理员（使用者）在主窗口上点击新增图书信息按钮—onMouseClick事件，系统将弹出新书录入窗口—ShowFrame方法；

2）然后图书管理员在新书录入窗口中选择图书的类别（通过书籍类别列表框来实现）；

3）图书管理员所选择的信息将传给书号规则—SendItem方法，书号规则将计算生成书号—GenerateNo方法，并返回给新书录入窗口—ReturnNo方法。

4）当图书管理员输入完新书信息后，按下提交按钮—onMouseClick事件，则将对书籍列表进行查询，看该书是否存在，即重复性判断—isExist方法；

5）如果没有重复，则创建书籍—Create方法，并存入书籍列表—Store方法。要注意的是，在Create方法前面，我们加上了约束“[No Exist]”，也就是表示分支情况。

交互建模之后

到此为止，我们就完成了用例UC01所对应的交互模型。不过，事件还没有结束，我们需要在这个成果的基础上进一步工作，将其发挥更大的作用。这些工作包括添加类的属性和方法、质量评审、引入基础类以及用设计模式进行优化，下面我们就一一作个简单的介绍。

添加个类的属性与方法

在构建交互模型时，我们将会发现类应该具有的方法，也会在设计时找到一些新的属性，而这些东西将进一步地完善我们的静态模型。

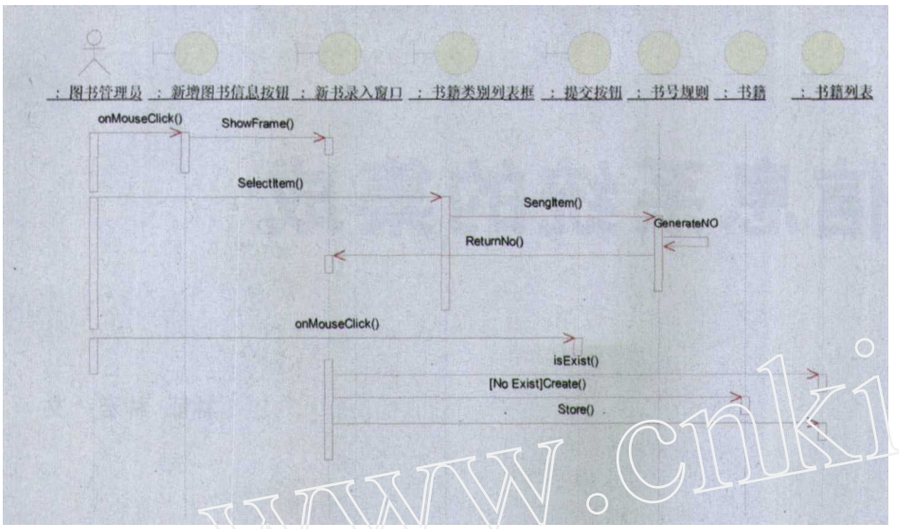
我们基于域模型的基础，结合Robustness分析、交互模型构建时所引入的设计类，画出相应的设计类图，并且将这里所找到的属性、方法补充在类图中去，这样我们将获得一个较完整的类模型。

引入基础类

在我们着手开发之前，还有一件很重要的事要去完成，那就是引入基础类。我们知道，不管你将使用什么开发工具进行代码编写，都将以各种库函数、框架作为开发基础。例如，.NET、J2EE、CORBA等框架，MFC、OWL、BDE、Swing、JDBC等库函数。

我们首先要根据应用的需要选择相应的框架，然后再根据具体的局部需要还选择相应的库函数。例如：

- 如果我们需要进行数据库操作，我们将可能使用ODBC、JDBC、ADO、BDE等数据库访问引擎中的一种；
- 如果我们需要分布式地处理，就可能要从DCOM、CORBA、EJB、Web Services中选择一种合适的技术；
- 当我们需要进行网络操作时，我们可以从MFC中选择一个Socket的实现，也可能是从SUN提供的网络包中选



图五 用例UC01的交互模型

择相应的类；

❑ 当我们需要进行用户界面设计时，我们可能使用MFC中提供的相关类，也可能使用Java中的AWT或Swing包来实现。

我们需要将这些包中，将要使用的类引入，然后从中派生（使用继承）应用系统所需要的类。如果你使用Rose进行类建模，那么你就可以很方便地引入这些基础类，因为Rose都将这些基础类做好了。

从这里的描述中，大家也应该看出一个学习开发知识的要点，即应该花足够多的时间来了解各种基础框架、库函数的功能与特性，以便在设计时做出最优的选择；另外，还应该对这些基础框架、库函数的类结构有一个清晰的了解，这样就可以最有效地找到基础类，最高效地使用。

质量评审

当我们通过引入基础类之后，将获得一个较完整的类模型，接下来我们就需要运用面向对象设计的一些基本原理，对其进行质量评审。评审的要点在于以下几个方面：

❑ 低耦合：耦合性是指两个类之间的连接强度，耦合性越低，说明类之间的独立性越高，相应的系统的灵活性也越高。

❑ 高内聚：内聚性则是指一个类的属性与方法高度地集成，内聚性越高，说明类的设计越合理，系统的稳定性也越高。

❑ 效率：低耦合与高内聚都是一个相对的概念，衡量的要点在于解决方案的执行效率是否满足系统的需求。

❑ 完整性：类的完整性是指在任何环境下都可以重复使用，完整的类也就意味着其具有较高的内聚性，也就意味着它与其它类之间的耦合较低。

❑ 简单性：每一个类越简单，出错的可能性越小，系统的灵活性和可维护性也越好。而把类当作一个框，什么都往里装的代码风格，就是一个具有“坏味道”的代码，需要重构它！

在质量评审是，你可以采用一些例如OCP原则、SRP原则、DIP原则、LSP原则等等经典的面向对象设计原则来衡量。关于这方面的知识，可以参考Robert Martin倾注9年心血而成的《敏捷软件开发》一书，笔者在《CSDN开发高手》上连载的“大话Design”系列文章中也有针对性地介绍了这些知识。

用设计模式进行优化

如果你在质量评审中发现了问题，那么你可以使用两种武器，那就是设计模式与重构。它们都将帮助你使代码更加的高质量，重构技术侧重于代码结构的重新整合，而设计模式则是通过引入新的设计类，还提高代码的可维护性、灵活性。

例如，在本例中，如果我们希望系统能够很方便地移植到不同的数据库环境中，就可以使用Facade模式或者是Proxy模式来实现；如果我们希望程序以AWT或Swing两种风格，那么Factory系列模式将帮助你达到这点。关于设计模式的更多内容，你可以从GOF的《设计模式》、阎宏的《JAVA与模式》、Martin的《敏捷软件开发》进一步研究。

不过，有一点希望大家能够明白，设计模式是在详细设计阶段中应用的，并不能够很好地与分析阶段的工作有机地结合起来。

表一 模式应用领域一览表

模式类别	应用阶段
设计模式	详细设计阶段
体系结构模式	高层设计阶段
分析模式	分析阶段

近几年，模式的研究进展很快，除了上表中所列出之外，还涌现出了一些项目管理模式、排错模式，以及总结错误经验教训的反模式。因此，正确地理解模式所应用的范围、适用的阶段是很重要的。毕竟模式就是一种经验的复用，它虽然不能够代替一切，但却能够帮助你以更短的时间、更好地完成任务。

结言

总之，交互建模是详细设计阶段的重要工具，当我们完成了交互模型之后，我们就会发现所有的类跃然纸上，而且这些类所需的属性和方法（即行为）也被清晰地找到，还清楚地掌握了类与类之间的交互，然后通过引入基础类、利用设计模式优化，将会使得紧接下来的代码编写工作将变得更加清晰。

不幸的是，由于篇幅的限制，我们从用例建模开始，只对其中的一个用例进行了分析，完成了用例描述，也仅对一个用例进行了Robustness分析（初步设计）、构建交互模型（详细设计）。因此，我想大家也不知不觉地走进了细节，也许让您感到“只见树木，不见森林”了。不过没关系，我将在下一期中再次从更宏观地角度帮助大家整理一下思绪，然后再继续进发。