

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

Coffeewoo 软件技术之路系列文集之一

用例分析篇

00 系统分析员之路

(姊妹篇 00 系统设计师之路尚在编写过程中, 敬请期待)

作者简介: coffeewoo

混迹于 IT 多年，做过 Coding, Presales, PM, SA, Architect。擅长于 UML、RUP、OOA、OOD、项目管理和 PMP，架构设计方面也有一些经验。有不少朋友来信问我的联系方式，可能邮件的确不能满足交流的需要吧。虽然我很希望有问题的朋友能在 blog 里留言，这样同样的问题可以 Share 给别人，现在还是决定公布一下俺的 MSN: coffeewoo@263.net

BTW: 白天是工作时间，我可能没办法及时回答一些问题，请谅解。

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

目录

COFFEEW00 软件技术之路系列文集之一	1
用例分析篇	1
00 系统分析员之路	1
目录	2
1 什么是用例	4
1.1 相关评论	7
2 用例的类型与粒度	10
2.1 用例的类型	10
2.2 用例的粒度	12
2.3 相关评论	14
3 涉众分析	19
3.1 什么是涉众	20
3.2 业主	21
3.3 业务提出者	21
3.4 业务管理者	22
3.5 业务执行者	22
3.6 第三方	23
3.7 承建方	23
3.8 相关的法律法规	23
3.9 用户	24
3.10 相关评论	24
4 业务建模一般步骤和方法	26
4.1 建模第一步	27
4.2 第二步	27
4.3 第三步	27
4.4 第四步	28
4.5 第五步	28
4.6 第六步	28
4.7 第七步	28
5 用户、业务用例和业务场景	30
5.1 用户	31
5.2 业务用例	31

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

5.2.1	用户视角:	32
5.2.2	业务视角:	33
5.3	业务场景	33
5.3.1	借阅图书业务过程:	34
5.4	相关评论	35
6	用例实现、用例场景和领域模型	38
6.1	用例实现	38
6.2	用例场景	40
6.3	领域模型	42
6.4	相关评论	45
7	用例规约的编写--业务规则和实体描述	49
7.1	业务规则	49
7.1.1	全局规则	49
7.1.2	交互规则	49
7.1.3	内幕规则	50
7.2	用例规约示例	53
7.3	业务实体描述示例	55
7.4	相关评论	55
8	编写完整的 UML 需求规格说明书	57
8.1	用例补充规约	57
8.2	需求规格说明书	60
9	问题和回复	62
10	APPENDIX	92
10.1	一个房屋中介业务建模的实例分析	92
10.2	需求规则说明书示例	101

1 什么是用例

我发现，在 OO 和 UML 几乎一统天下的今天，仍有很多系统分析员对 OO 和 UML 一知半解，甚至包括很多已经使用了很久 UML 的系统分析员。

于是打算写一个系列文章，将多年来的工作经验做一个总结。对初学者起个启蒙作用，也希望抛砖引玉，与各路大虾共同探讨，共同提高。

这个系列文章将以我对 OO 和系统分析的理解为主，从 UML 基础开始，阐述面向对象的需求分析方法，过程，并以 RUP 为例，阐述如何将 OO 过程与软件过程有机结合在一起，做一个真正 OO 应用。

好了，今天是第一篇。想得很远，真希望我能坚持下去，呵呵

用例是什么？其原始英文是 usecase，直译过来就成了用例。这也是一个比较贴切的叫法了，从字面的直接理解就是使用的例子。另一种比较流行的定义是用例就是与使用者(actor)交互的，并且给使用者提供可观测的有意义的结果的一系列活动的集合，这个定义还是比较费解的。笔者在众多应聘者中发现很多使用用例来做需求的系统分析员，有的已经使用了两年以上，但仍不能把握用例的本质，虽然他们号称精通 UML。

最具普遍意义的理解错误是认为用例就是功能的划分和描述，认为一个用例就是一个功能点。在这种理解下，用例变成了仅仅是较早前需求中功能框图的翻版，很多人用用例来划分子系统，功能模块和功能点。如果这样，用例根本没有存在的必要。有意思的是，造成这种理解错误的相当一部分原因却是因为对 OO 思想的理解不够深入，本质上说，把用例当成功能点的系统分析员脑子里还是面向过程的那一套思想，虽然他们在使用 OO 的工具，OO 的语言，号称在做面向对象的开发，但过程的影子还没有从他们脑子里彻底抹去。

如果用例不是功能的话，它是什么呢？从定义上说，能给使用者提供一个执行结果的活动，不就是功能吗？我的回答是：错！功能是计算机术语，它是用来描述计算机的，而非定义需求的术语。功能实际描述的是输入-->计算-->输出。

这让你想到了什么？DFD 图？这可是典型的面向过程分析模式。因此我说把用例当做功能点的分析员实际在做面向过程的分析。

而用例则不是计算机术语，UML 除了在计算机行业中应用，它也经常应用在其它行业中。用例是一种需求方法学，虽然软件危机和 OO 的发展促成了它的诞生并被完美的融合进了 OO 体系，形成了 UML，但它实际上并不是软件行业的专用品。如果非要从功能的角度解释，那么用例可以解释为一系列完成一个特定目标的“功能”的组合，针对不同的应用场景，这些“功能”体现不同的组合方式。实际上，把用例解释为某个参与者(actor)要做的一件事可能更为合适。这样的一件事有以下几个特征：

一、这件事是相对独立的。这意味着它不需要与其它用例交互而独自完成参与者的目的。也就是说这件事从“功能”上说是完备的。读者可能会想到，用例之间不是也有关联关系吗？比如扩展，比如实现，比如继承，它看上去并不是独立的嘛。关于这个问题，笔者会在后续的文章里详细说明。这里稍微解释一下，用例之间的关系是分析过程的产物，而且这种关系一般的产生在概念层用例阶段和系统层用例阶段。对于业务用例，这个特征是很明显的。

二、这件事的执行结果对参与者来说是可观测的和有意义的。例如，系统会监控参与者在系统里的操作，并在参与者删除数据之前备份。虽然它是系统的一个必需组成部分，但它在需求阶段却不应该作为用例出现。因为这是一个后台进程，对参与者来说是不可观测的，它应该在系统用例分析阶段定义。又比如说，登录系统是一个有效的用例，但输入密码却不是。这是因为登录系统对参与者是有意义的，这样他可以获得身份认证和授权，但输入密码却是没有意义的，输入完了呢？有什么结果吗？

三、这件事必须由一个参与者发起。不存在没有参与者的用例，用例不应该自动启动，也不应该主动启动另一个用例。用例总是由一个参与者发起，并且满足特征二。例如从 ATM 取钱是一个有效的用例，ATM 吐钞却不是。因为 ATM 是不会无缘无故吐钞的，否则，我从此天天守在 ATM 旁，生活无忧矣。

四、这件事必然是以动宾短语形式出现的。即，这件事必须有一个动作和动作的受体。例如，喝水是一个有效的用例，而“喝”和“水”却不是。虽然生活常识告诉我们，在没有水的情况下人是不会做出喝这个动作的，水也必然是喝进去的，而不是滑进去的，但是笔者所见的很多用例中类似“计算”，“统计”，“报表”，“输出”，“录入”之类的并不在少数。

除去以上的特征，笔者觉得用例的含义还要更深些。首先，用例的背后是一种需求方法论。其核心是以参与者为中心（区别于以计算机系统为中心），从参与者的角度来描述他要做的日常工作（区别于以业务流程描述的方式），并分析这些日常工作之间是如何交互的（区别于数据流的描述方式）。换句话说，用例分析的首要目标不是要弄清楚某项业务是如何一步一步完成的，而是要弄清楚有多少参与者？每个参与者都做什么？业务流程分析则是后续的工作了。其次，用例简直就是为 OO 而生的，其思想完美的符合 OO。用例分析方法试图找到问题领域内所有相对独立的参与者和事件，并把业务流程当成是这些参与者和事件之间的交互结果（在 UML 用活动图或序列图来描述）。因此，用例方法被吸纳到 OO 之后，UML 得以以完备的形式出现，用例成为了真正的 OO 核心。在 RUP 里，这种核心作用被发挥到极致，产生了用例驱动（usecase driven）的软件过程方法，在 RUP 里，软件生产的所有过程和产物都是围绕着用例形成的。

可以说，用例分析是 OO 的第一步。如果用例分析本身出了问题，对业务架构，软件架构的影响是很大的，将大大削弱 OO 的优势--复用、扩展。笔者认为软件复用可以分为三个层次，最低层次的复用是代码级复用，这是由 OO 语言特性提供支持的，例如继承，聚合，多态；较高层次的复用是组件级复用，这是由设计模式提供支持的，例如 Factory 模式，Builder 模式；最高层次的复用则是服务级复用，这在很大程度上是由应用服务器和通讯协议来提供支持的，例如最近炒得火热的 SOA（面向服务的应用）架构。用例分析的好坏也许对代码级和组件级的复用影响不太大，但对服务级的复用影响却是巨大的。笔者认为服务级复用是 OO 的最高境界，而结构良好的用例分析则是达到这一境界的基础。

闲话：

今天你 OO 了吗？

如果你的分析习惯是在调研需求时最先弄清楚有多少业务流程，先画出业务流程图，然后顺藤摸瓜，找出业务流程中每一步骤的参与部门或岗位，弄清楚在这一步参与者所做的事情和填写表单的结果，并关心用户是如何把这份表单传给到下一个环节的。那么很不幸，你还在做面向过程的事情。

如果你的分析习惯是在调研需求时最先弄清楚有多少部门，多少岗位，然后找到每一个岗位的业务代表，问他们类似的问题：你平时都做什么？这件事是谁交办的？做完了你需要通知或传达给谁吗？做这件事情你都需要填写些什么表格吗？.... 那么恭喜你，你已经 OO 啦！

1.1 相关评论

看完‘今天你 OO 了吗’，才知道。我是个完完全全的面过过程开发者...

非常感谢楼主。继续拜读其他文章...

cat 评论于：2006.10.31 11:26

不用灰心，其实 OO 和过程的转换只在一念之间，同一件事情你是从完整流程逻辑角度去看的？还是从人的角度去看的？流程还是要搞清楚的，无非是把流程看作是人做事以后交互的结果，而不是系统的本源，就这么一点差别：)

coffeewoo 评论于：2006.10.31 11:49

赞美之言就省略了，非常仔细的看了《什么是用例》一文。条分缕析，字字珠玑，看的直呼过瘾。

不过，在看“二、这件事的执行结果对参与者来说是可观测的和有意义的。例如，系统会监控参与者在系统里的操作，并在参与者删除数据之前备份。虽然它是系统的一个必需组成部分，但它在需求阶段却不应该作为用例出现。因为

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

这是一个后台进程，对参与者来说是不可观测的，它应该在系统用例分析阶段定义。…” 这段的时候，我一直疑惑着，系统备份删除前的数据，这个极其重要的功能需求，不在需求阶段列出来，何时列出来呢？我还没看到后续的文章，也许后续解决了，但是这个问题我不搞清楚，我会一直认为通过获取用例的方式，并不能将系统的所有功能都得到。很多用户不参与的事情，系统会悄悄的做了。用例不列出来，系统功能不就不完全了吗？

w8u 评论于: 2006.11.10 13:56

“系统备份删除前的数据”这样的语句其实很模糊。

有两种情况：

1. 语意：逻辑删除
2. 语意：客户提出的需求

假设是 1 的语意，那很简单，在业务用例阶段完全不需要出现“系统做什么”，只有在系统用例和之后的分析模型阶段再出现

假设是 2 的语意，那你就要考虑这样的语句本身对你的需求有什么样的影响？也即“备份删除前的数据”究竟是为了做什么？这也要分两种情况：

1. 客户假设说我是想为了保存以前的历史数据以备不时之需
2. 客户假设说我是可能想在某个报表中可以看到版本比较的信息

好，如果是 1 的需求，那很简单，你只是记录下这句客户的话，然后作为一个“非功能性需求”列在你最终生成的《需求规格说明书上》

如果是 2 的需求，那你就有必要在“备份删除前的数据”和“查看版本报表”这两个动作所处的某两个（或一个）业务用例中加上这样的“用例描述”：客户删除数据->生成历史数据->。。。->客户查看历史数据

rwyx 评论于: 2006.11.10 23:35

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

楼主能不能结合这个图书例子用你的风格讲解一下系统分析员和架构师的
区别。看了一些文章，总不得要领。感激涕零！

qcrsoft 评论于：2006.12.18 03:23

用例分析系列中所有的工作，都是系统分析员做的，架构师不做这些工作。
但这些工作的成果是架构师工作最主要的输入之一。

夸张一点说，系统分析员可以完全不懂计算机知识。他是领域专家，是行业顾问，或者有着对陌生问题领域敏锐的视角和极强的总结，归纳能力。能够把客户分散的，杂乱的，矛盾的需求整理成为完备的，自洽的，有弹性的结构，并且能够与客户共同探讨。

架构师是计算机专家，软件的行家里手。需要深入了解各种各样的软件结构，应用模式，中间件，服务器，甚至硬件知识...具备这么全面的知识目的是为接下来的开发工作定下基调。根据需求规模，应用环境要求，客户特殊要求，应用程序特性等，再根据公司的基本情况，来选择或决定技术路线，中间件，开发工具等。为开发定下基调，大的架构。小公司，中，小型项目我个人意见是根本用不到架构师这个角色的，顶多一个高级设计师把握整体框架就行了。架构师还要高于高级设计师，只有当一个项目或产品大到需要很多个开发组，产品由很多个可独立的组件组成的时候，架构师才有意义。

coffeewoo 评论于：2006.12.18 11:09

2 用例的类型与粒度

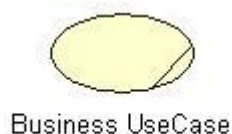
2.1 用例的类型

在正式讨论如何获取用例之前，笔者觉得有两个问题还是先解释清楚为好，这对正确获取用例有很大帮助。这两个问题也是初学者最为困惑，也是最难掌握的。一个是各种用例类型之间的区别和用法，另一个是用例的粒度。

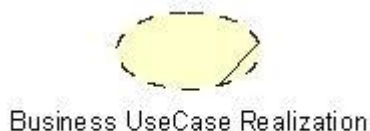
先说说用例类型的问题。

用例类型，有的资料翻译为版型，英文原文是 stereotype。在 Rose 中默认的类型有 business usecase，business usecase realization 和 use case realization 三种。相应的，就是指

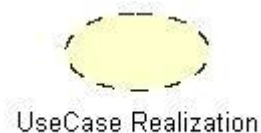
业务用例：



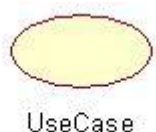
业务用例实现：



用例实现：



若不指定类型，则它就是通常意义上的 use case：



Rose 中定义了上述默认类型，但是也可以自定义用例类型。初学用 UML 建模的人常常在这些类型中迷失，搞不清楚这些类型是什么含义，什么时候该使用什么类型。简单说，需求分析中的各个阶段要描述和分析的目标不同，为表达不同的视角和分析目标，需要使用不同的用例类型。笔者的观点是，用例类型的区分是为了形式上的统一，但用例类型既然可以自定义，它就是一个很灵活的用法，不必墨守成规，大可在工作中根据实际情况定义适合自己项目特点和软件过程的用户类型。不过，默认的用例类型已经几乎可以满足需求分析的各个阶段，自定义的必要性并不大，并且 UML 的意义就是使用统一的形式描述需求，因此最好使用默认的类型。

说到需求分析阶段，那么需求分析都有些什么阶段呢？一般来说，需求分析要经过业务建模，用例分析和系统建模三个阶段才能完成需求工作，这三个阶段分别做什么笔者会在以后的文章的详细阐述，这里为了说明用例类型的含义和使用，先简单介绍一下。

1、业务建模的目标是通过用例模型的建立来描述用户需求，需求规格说明书通常在这个阶段产生。这个阶段通常使用业务用例和业务用例实现两种类型；

2、用例分析是系统分析员采用 OO 方法来分析业务用例的过程，这个阶段又称为概念模型阶段。这个阶段通常使用无类型的用例。用例分析是一个过渡过程，但笔者认为其非常重要，业务架构通常在这个阶段产生。

3、系统建模是将用户的业务需求转化为计算机实现的过程。这个阶段通常使用无类型的用例和用例实现两种类型。系统范围，项目计划，系统架构通常在这个阶段形成雏形（在系统分析阶段确定）。

所谓 business usecase，是用来描述用户原始需求的，它的含义是站在用户的角度，使用用户的业务术语来描述用户在其业务领域所做的事情。业务用例

命名，描述都必须采用纯业务语言，而不能出现计算机术语。因为业务模型是系统分析员与用户讨论需求，达到一致理解的基础，必须使用用户熟悉的，不会有歧义的专业术语以避免系统分析员与用户对同一事件的理解误差。business usecase realization 是达到需求可追溯要求的一个连接点，是用户在其业务场景中如何做这一件事的载体。

笔者自己在用例分析和系统建模阶段不区分用例类型，都使用无类型的 use case，但在这两个阶段，用例的含义还是有所差别的。用例分析阶段，用例主要是从业务模拟的概念上，从 OO 的视角来分解、组合业务用例，粗略的建立一个业务架构。而在系统建模阶段，用例主要是从计算机视角描述需求，规定开发范围，作为项目计划的依据，为系统设计做准备。usecase realization 的含义可从 business usecase realization 推知。

2.2 用例的粒度

粒度是令人迷惑的。比如在 ATM 取钱的场景中，取钱，读卡，验证账号，打印回执单等都是可能的用例，显然，取钱包含了后续的其它用例，取钱粒度更大一些，其它用例的粒度则要小一些。到底是一个大的用例合适还是分解成多个小用例合适呢？这个问题并没有一个标准的规则，笔者可以给大家分享的 experience 是根据阶段不同，使用不同的粒度。在业务建模阶段，用例的粒度以每个用例能够说明一件完整的事情为宜。即一个用例可以描述一项完整的业务流程。这将有助于明确需求范围。例如取钱，报装电话，借书等表达完整业务的用例，而不要细到验证密码，填写申请单，查找书目等业务中的一个步骤。在用例分析阶段，用例的的粒度以每个用例能描述一个完整的事件流为宜。可理解为一个用例描述一项完整业务中的一个步骤。需要注意的是，这个阶段需要采用 OO 方法，归纳，抽象业务用例中的概念模型。例如，宽带业务需求中有申请报装，申请迁移地址用例，在用例分析时，可归纳和分解为提供申请资料，受理业务，现场安装等多个业务流程中都会使用的概念用例。在系统建模阶段，用例视角是针对计算机的，

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

因此用例的粒度以一个用例能够描述操作者与计算机的一次完整交互为宜。例如，填写申请单，审核申请单，派发任务单等。可理解为一个操作界面，或一个页面流。在 RUP 中，项目计划要依据系统模型编写，因此另一个可参考的粒度是一个用例的开发工作量在一周左右为宜。

上述的粒度划分方法笔者是用相对比较具体化的一些依据来说明。实际上，用例粒度的划分依据（尤其是业务用例）最标准的方法是一个用例的粒度是否合适，是以该用例是否完成了参与者的某个目的为依据的。这个说法比较笼统，也比较难以掌握。举个例子，某人去图书馆，查询了书目，出示了借书证，图书管理员查询了该人以前借阅记录以确保没有未归还的书，最后借到了书。从这段话中能得出多少用例呢？请记住一点，用例分析是以参与者为中心的，因此用例的粒度以能完成参与者目的为依据。这样，实际上适合用例是：借书。只有一个，其它都只是完成这个过程——这里讨论的用例指的是业务用例——这个例子是比较明显的能够区分出参与者完整目的的，在很多情况下可能并没有那么明显，甚至会有冲突。读者可以从自己实际的情况去找出这种例子。以后的文章中笔者会做一些讨论。

但是上述的粒度选择并不是一个标准，只是在大多数情况下这样的粒度选择是比较合适的。笔者的意思也不是告诉读者上述哪一种是最好的，而只是把一些常用的比较，划分方法告诉读者，期望的是帮助读者在工作实践中自己总结出一套适合自己的方法来。现实情况中，一个大型系统和一个很小的系统用例粒度选择会有较大差异。这种差异是为了适应不同的需求范围。比如，针对一个 50 人年的大型项目应该选择更大的粒度，如果用例粒度选择过小，可能出现上百甚至几百个业务用例，造成的后果是需求因为过于细碎和太多而无法控制，较少的用例有助于把握需求范围，不容易遗漏。而针对一个 10 人月的小项目应该选择小一些的粒度，如果用例粒度选择过大，可能只有几个业务用例，造成的后果是需求因为过于模糊而容易忽略细节。一般来说，一个系统的业务用例定义在多于 10 个，少于 50 个之间，否则就应该考虑一下粒度选择是否合适了。

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

不论粒度如何选择，必须把握的原则是在同一个需求阶段，所有用例的粒度应该是同一个量级的。这应该很好理解，在描述一栋建筑时，我们总是把高度，层数，单元数等合在一起介绍，而把下水道位置，插座数量等合在一起介绍。如果你这样介绍一栋楼：这栋楼有 10 层，下水道在厨房东南角，预留了 15 个插座，共有 5 个单元，听众一定会觉得云山雾罩，很难在脑子里形成一个清晰的影像。

如果对上面两个问题读者还有疑惑，不用着急，在以后的文章中应该会逐步加深理解。

2.3 相关评论

Question: 由 pushboy 发布

“在业务建模阶段，用例的粒度以每个用例能够说明一件完整的事情为宜。即一个用例可以描述一项完整的业务流程。”

“在系统建模阶段，用例视角是针对计算机的，因此用例的粒度以一个用例能够描述操作者与计算机的一次完整交互为宜。”

那么，这样一个场景 —— 用户管理，有增删改查

这里，是把 用户管理 作为一个用例，还是把增删改查分别作为用例呢？

他们每一个都是一个完整的业务流程和一次完整交互，而且也都是一个 actor 发起的动作；怎么来确认呢？

我们的前提是一个普通的比如说财务系统，而不是一个用户管理系统

Answer: 这个问题很好，用例的粒度总是在这样左也可右也可之间难以决定。对这个问题来说，我的观点是业务用例应当用“管理用户”或“维护用户”作为合适的粒度，而增，删，改，查则在作为系统用例。我的理由是：

增删改查不能做为一个完整的业务来理解。作为一个管理业务，数据只有先增，才会有改，才会有删。增删改查结合起来才能完成 actor 的管理目的，只删或只增加都不是业务的全部。这篇文章后来我又补充了一条粒度的判断标准，可

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

以到 <http://coffeewoo.itpub.net/> 去看，是否是一项完整业务，要看 actor 的目标，而不是事情是否完整。这个例子中，actor 的目标是为了增加一个用户吗？是为了删除一个用户吗？都不是，而是为了管理用户，这个目标包括了用户这个实体的整个生命周期。

再讨论深一些，如果业务要求对用户除了增删改查，还有别的要求，例如权限升级，用户在组织机构中复杂的关系变更，用户与外部系统的交互.... 实际的情况可能会更多，那么，如果将每个都作为一个业务用例，很容易造成一个结果，这些原本与用户这个实体紧密关联，共同组成用户实体生命周期的业务，被割裂成多个独立的业务，因为定义了多个用例（请参看本人第一篇中的用例特征第一条）。要知道，在 RUP 中，用例驱动的含义是，一个用例就是一个分析单元，设计单元，开发单元，测试单元甚至部署单元。相信读者都知道，把紧密关联的业务分成多个独立部分去实施是高成本的，高风险的。

另外，为什么我要说在系统用例阶段要把增，删，改，查又分出来呢？原因在于，系统用例的目的是为了将 actor 的业务用计算机模拟出来。我们都知道，一般情况下，增，删，改，查对一个 actor 来说是不会同时发生的，每次 actor 只会完成其中的一个行为。分开来，有利于详细分析模拟这一行为的细节而不至于混淆。另一方面，对 WEB 应用来说，针对数据的增，删，改，查等，很容易形成所谓的“模板”，增加用户用这个模板，增加其它基础数据可能也用同一个模板，无非是操作的数据（实体）不同而已。因此，在很多情况下，这些模板是可以复用的。对这个例子来说，在系统用例阶段，我们可以用“管理用户” include “增加用户”来表示这个实现关系，同时，让“增加用户”，“增加 XX 数据”等等用例来继承自一个抽象出来的“增加数据”用例，这样，可复用的模板体现在“增加数据”用例上，而具体业务，则体现在“增加 XX 数据”上。实际上，这也是一种 OO 思想的体现。

本文是网络 ID 为 coffeewoo 的作者原创编写, 原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成, 并发给作者本人, 再由作者本人向外发布的。本文允许自由传播, 仅供个人学习之用, 禁止用于商业行为, 如出版物, 培训教材等。读者在使用本文时请尊重作者之著作权, 勿篡改或删除或改编本文。若有非个人的任何公司, 组织和商业机构想采用本文之全部或一部分, 请与作者联系, 勿私自使用。若出版商有意出版此文, 请与作者联系。谢谢合作。

只有一个查询是比较特殊的, 查询一般不一定只局限于一个 actor, 也不一定局限这个用例, 一般都是所谓的综合查询, 是可能跨用例的。所以根据实际情况, 查询可以作为一个业务用例出现。

感谢网友 pushboy 提出问题。原帖位于:

<http://www.itpub.net/507773.html>

coffeewoo 发表于: 2006.03.03 23:48

写得很好, 您的观点是十分有效的。

只是, 在用例阶段, 我觉得还是要将业务用例和系统用例的角色解释一下作为补充. 业务用例和系统用例的各个角色是不同的。

业务用例方面角色应该有: 业务角色 业务员工

系统用例方面角色应该有: 业务角色

什么概念? 业务用例的业务角色是指在企业外与企业交互的角色, 业务员工是指在企业内进行工作的角色. (因为业务用例仅仅是对企业内一个部门的业务做一个粗略的用例)

系统用例则没有业务员工, 只有业务角色, 业务角色是指对于整个系统来说与系统交互的人或其他子系统(因为系统用例是对整个系统的工作进行描述, 可以看作是需求完成后系统的工作结果)

其他完全同意作者的理论, 我的补充也仅仅是一点补充

rwyx 评论于: 2006.09.28 15:54

在业务用例模型中审核应该不是一个用例, 但这要看你做用例的企业是不是有一个部门是专门做审核的, 如果有一个部门的工作可以用"审核 XXX"来定义, 那就是一个用例. 否则应该以该部门的主体工作来包含"审核"的动作. 而到系统用例模型时才去将审核作为一个单独用例

rwyx 评论于: 2006.09.28 15:58

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

关于业务用例和系统用例，您的观点是对的。业务阶段和系统阶段的角色定义是不一样的，即使是业务阶段，也有 business work 和 business actor 之分。文章里我是有意忽略这种差别的。因为在实际工作中我觉得，搞那么多概念出来，除非是很专业的 UML 人员，否则不会了解这种差异，更不用说客户了。需求文档是要给客户看的，要给他们解释清楚 business work 和 business actor 很难，而这两者的虽然概念上不一样，但最后能映射到系统中的也就是角色。因此在实际工作中，我不再区分它们，而是只使用角色一个概念。

关于审核的回答十分赞同您的意见。

coffeewoo 评论于：2006.09.28 18:24

提个关于过程控制的软件需求分析问题，对于业务管理者来说（一般是中层们），他们也是系统的用户，但他们参与系统的主要方式是对中层之下用户操作业务的结果进行查询分析，这样这些查询久的定义成用例。可这些查询用例之间或是和其他用例之间又没有什么交互，那我是不是需要把每个这样的查询用例当成一个单独的业务来绘制业务场景图，或是把这些查询弄成一个综合查询业务，但那样泳道好像太多，没有实际意义。疑惑。

boskin 评论于：2006.12.18 11:03

MIS 系统中唯一一个比较特殊的就是查询了。我在房屋中介实例分析那一篇中讨论过一部分。

对于查询这种用例，最特殊的地方就是 actor 目标是不明确的，查询的目的可以做很多事情。查询完了只看一看，或者删除一部分，或者打印出来，或者... 目的是很多的。显然如果按目的来分，就会有非常多的查询 XX 用例。对于查询，我的建议是将其作为正常用例的 extend。例如管理用户用例，要删除一个用例，显然应当先查询出来才能删除，所以这时查询用户只是管理用户用例的一个 extend。而扩展出来的用例是不单独视为一个用例的，换句话说可以不为它单独

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

分析。

另一方面，查询，除非特殊要求的，否则查询的过程都是一模一样的，输入条件，点查询，出结果。无非是查询的目标不同。所以即便要为查询专门分析，例如综合 查询，其重点也不在业务场景图上，而在查询用例所使用到的那些业务实体。UML 在说明功能性需求上是方便的，但在说明这种以数据为中心的内容上是不擅长的，什么条件出什么结果应当放在用例规约中去，以文字和表格说明为好。

coffeewoo 评论于: 2006.12.21 21:56

3 涉众分析

从这一篇开始，笔者将借助一个虚拟的实例来阐述获取用例的方法，以及如何判断用例获取是否完备，粒度选择是否合适。事实上，在做这些工作时，我们正在进行需求分析的第一个阶段，即业务建模阶段。借助这个例子，笔者同样会阐述业务建模到底应该做什么，做到什么地步才能说明业务需求已经完整，可以称为一份完整的需求规格说明书了。

一般来说，只有当以下工作都完成，才能说业务模型建立完成，它们是：

- I 发现和定义涉众
- I 画定业务边界
- I 获取用例
- I 绘制用例场景图
- I 绘制业务实体模型（领域模型）
- I 编制词汇表

下面笔者开始就一个事例来说明如何完成这些工作，这只是一个虚拟的事例，它的合理性和真实性请读者不必追究。

现在我们接到一个项目，是一个网上图书借阅系统，初步跟客户接触，网上图书馆的业务负责人这样告诉我：

我们原本是一个传统的图书馆，传统的借书方式要求读者亲自来到图书馆，这显得非常不方便，而且随着藏书的增加和读者群的增长，尤其而且大量的读者到图书馆，使得图书馆的场地不足，工作人员也不够了。所以想到借助网络，让读者通过网络借/还书，这样可以省掉大量的场地维护和工作人员成本支出，同时计算机可以方便的检索目录，让读者可以足不出户借到需要的书。为了把书送到借阅人手里，我们已经联系了 A 特快专递公司和 B 城市物流公司，初步达成协议，由他们往返借阅人和图书馆之间把图书送出和收回。读者在网上出示和验证借书卡，找到他们需要的书，提交申请，图书管理员确认后，就会通知物流公司

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

来取书，当读者拿到书之后，物流公司需要把读者的签单拿回来以证明读者已经拿到了书。当然这个过程中，读者是需要付费的。还书也是基本同样的过程。

(请读者注意这段需求描述。本系列文章后续的所有分析，讲解都基于这个需求的假设)

好了，通过这番谈话，我们已经基本上了解了系统目标。一些着急的系统分析员已经准备开始着手询问借书的流程，借阅人的资格认证问题了，甚至有的人已经凭借多年的开发经验在脑海中绘制出了一幅网页，考虑如何实现这个系统了。

笔者要说的是，请您千万不要着急往下走。因为我们得到的仅仅是一个由非计算机专业人员规划出的很粗略的构想，其可行性如何都尚未得到证实，在这样的基础下就开始细化，将来出现反复甚至失败的危险是很大的。

在了解了系统目标以后，系统分析员最先要做的事情不是去了解业务的细节，而是去发现与这个目标相关的人和物。英文把这种人和物称为 Stakeholder，在 Rose 中，这类模型的类型被定义为 Business Actor。有的资料翻译为干系人，笔者则更喜欢涉众这种翻译方法。这就谈到了业务建模的第一步：发现和定义涉众。

3.1 什么是涉众

涉众是与要建设的业务系统相关的一切人和事。首先要明确的一点是，涉众不等于用户，通常意思的 user 是指系统的使用者，这仅是涉众中的一部分。如何理解与业务系统相关的一切人和事？笔者可以给大家分享的的经验是通过以下大类去寻找：

3.2 业主

业主是系统建设的出资方，投资者，它不一定是业务方。比如可以假设这个图书馆的网络化建设是由一家国际风险投资机构投资的，它本身并不管理图书馆，它只是从资本上拥有这个系统并从借书收入中获得回报。

了解业主的期望是必须和重要的，业主的钱是这个项目存在的原因。若系统建设不符合业主的期望，撤回投资，那么再好的愿望也是空的。

一般来说，业主关心的是建设成本，建设周期以及建成后的效益。虽然这些看上去与系统需求没什么大的关系，但是，建设成本、建设周期将直接影响到你可以采用的技术，可以选用的软件架构，可以承受的系统范围。一个不能达到业主成本和周期要求的项目是一个失败的项目，同样，一个达到了业主成本和周期要求，但却没有赚到钱的项目仍然是一个失败的项目。

3.3 业务提出者

业务提出者是业务规则的制定者，一般是指业务方的高层人物，比如 CEO，高级经理等。他们制定业务规则，圈定业务范围，规划业务目标。他们的期望十分十分重要，实际上，系统建设正是业务提出者经营和管理意志的体现。他们的期望一般比较原则化和粗略化，但是却不能违反和误解，否则系统将有彻底失败的危险。业务提出者一般最关心系统建设能够带来的社会影响，效率改进和成本节约。换句话说，他们只关心统计意义而不关心具体细节，但是，如果建设完成的系统不能给出他们满意的统计结果，这必定是一个失败的项目。在系统建设过程的沟通中，他们的意志一般是极少妥协的，系统分析员不必太费心去试图说服他们接受一个与他们意志相左的方案。实际上，由于他们的期望是非常原则化和粗略的，因此留给了系统建设者很大的调整空间和规避风险的余地。

3.4 业务管理者

业务管理者是指实际管理和监督业务执行的人员，一般是指中层干部，起到将业务提出者的意志付诸实施，并监督底层员工工作的作用。他们的期望也很重要，一般也是系统的主要用户之一。他们关心系统将如何实现他们的管理职能，如何能方便的得知业务执行的结果，他们如何将指令下达，以及如何得到反馈。业务管理者的期望相对比较细节，是需求调研过程中最重要的信息来源。系统建设的好坏与业务管理者的关系最多，也是系统分析员最需要下功夫的。系统分析员必须要把业务管理者的思路，想法弄清楚，业务建模的结果也必须与业务管理者达成一致。在系统建设过程中，业务管理者的期望可以有所妥协，一个经验丰富的系统分析员可以给他们灌输合理的管理方式，提供可替代的管理方法，以规避导致高技术风险或高成本风险的不合理要求。

3.5 业务执行者

业务执行者是指底层的操作人员，是与将来的计算机直接交互最多的人员。他们最关心的内容是系统会给他们带来什么样的方便，会怎样的改变他们的工作模式。他们的需求最细节，系统的可用性，友好性，运行效率与他们关系最多。系统界面风格，操作方式，数据展现方式，录入方式，业务细节都需要从他们这里了解。他们将成为系统是否成功的试金石。Look and Feel，表单细节等是系统分析员与他们调研时需要多下功夫的地方。这类人员的期望灵活性最大，也最容易说服和妥协。同时，他们的期望又往往是不统一的，各种古怪的要求都有。他们的期望必须服从业务管理者的期望，因此，系统分析员需要从他们的各种期望中找出普遍意义，解决大部分人的问题，必要时可以依靠业务管理者来影响和消除不合理的期望。

3.6 第三方

第三方是指与这项业务而关联的，但并非业务方的其他人或事。比如在这个事例中，借阅人借书时需要交费，若交费是通过网上银行支付的，则网上银行就成为了网上借书系统的一个涉众。

第三方的期望对系统来说不起决定性意义，但会起到限制作用。最终在系统中，这种期望将体现为标准、协议和接口。

另一种典型的第三方是项目监理，系统分析员也必须弄清楚监理的期望。

3.7 承建方

承建方，也就是你的老板。老板的期望也是非常重要的。老板关心的是通过这个项目，能否赚到钱，是否能积累核心竞争力，是否能树立品牌，是否能开拓市场。老板的期望将很大的影响一个项目的运作模式，技术选择，架构建立和范围确定。比如，老板试图通过这个项目打开一个市场，树立起品牌，不惜成本，那么，系统分析员需要尽可能的深入挖掘潜在业务，建立扩展能力很强，但成本较高的业务架构，选择那些较新，但风险较高的技术。反之，如果老板只想通过这个项目赚更多的钱，系统分析员就需要引导业务方压缩业务范围，选择风险小的成熟技术，甚至不用考虑业务架构，考虑系统的可维护性，而较少考虑系统扩展能力。

一个业主满意但老板不满意的项目，恐怕也不是一个成功的项目吧？

3.8 相关的法律法规

相关的法律法规是一个很重要的，但也最容易被忽视的涉众。这里的法律法规，既指国家和地方法律法规，也指行业规范和标准。例如，这个借阅系统中要建立借阅人档案，就必须保障借阅人的隐私权；要与网上银行交易，必须遵守信息安全法等。若遇到业务方提出违反了法律法规的要求时，系统分析员要能给他

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

们指出来，说服无果的情况下要在合同里留下免责条款。否则一不小心惹上官司可是件郁闷的事。

另外，有时必须得遵守一些行业规范。例如本事例是网上借阅，网络需求决定了需要遵守 HTML 规范，才能保证借阅者能正常浏览网页。

3.9 用户

用户是一个抽象的概念，是指预期的系统使用者。用户可能包括上述的任何一种涉众。用户涉众模型建立的意义是，每一个用户将来都可能是系统中的一个角色，是实实在在参与系统的，需要编程实现。而上述的其它涉众，则有可能只是在需求阶段有用，最终并不与系统发生交互。在建模过程中，概念模型的建立和系统模型的建立都只从用户开始分析，而不再理会其它的涉众。在 Rose 中建模的时候，也只需要建立用户的模型，其它涉众则只需要体现在文档中即可。

这篇文章只能到此为止了，否则太长的话，读者该不耐烦了。只好在此分节。下一节笔者将一步步将涉众的期望导出，并得到需求范围的大致轮廓。

3.10 相关评论

请教咖啡兄了，类似调度这种需求怎么描述，比如数据库调度，1 天做一次备份。又比如我系统里需要每一分钟做一些数据分析处理，这种需求没有哪个用户来驱动，如何定义它的用户，我开始把定时器作为用户，但发现那已经是实现了，那可以把调度本身作为个用户吗，这样调度做的事情就可以方便的用用例来描述了。我这个系统可能会碰到多种类似情况，比如数据改变而做的事情。缺乏经验阿。

boskin 评论于: 2006.12.25 11:29

象这种情况定时器是 actor。

actor 的概念并不是人，而是站在系统边界外（系统边界根据分析需要是可以扩大缩小的）驱动系统的一切人，事，物，甚至规则。系统边界划定后，边界以内

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

的，只有用例。边界以外的，向系统主动发出动作的，是为 actor，被动的从系统获得消息的，是为接口。

[coffeewoo](#) 评论于: 2006.12.25 19:07

接着上面这个问题，actor 是属于系统边界之外的人、事、物，可是定时器应该是属于系统内部的一个物，我可不可以这么理解，认为计算机的时钟是 Actor，以一分钟时间间隔调度为例，Actor 就是 1 分钟、2 分钟等这些分钟点。

[boskin](#) 评论于: 2006.12.26 12:55

我倒不认为定时器是系统内部的一个物。如果是系统内部的，它必须向外提供服务，也就是系统外有人在驱动它，它是不主动执行的。从 UML 的观点来说，区分系统内外并不是计算机的内外，不是自己的代码和别人的代码，也不是已有的代码和将要开发的代码。而是主张者和服务者，或驱动者和执行者的差别。如果你把定义器认为是系统内部的，那么必须还有另一个系统外的东西来驱动它，哪怕只需要发出一次消息定时器就开始自动工作。就像，上帝轻轻推动了一下，世界从此开始按规律运行。

[coffeewoo](#) 评论于: 2006.12.26 13:37

4 业务建模一般步骤和方法

本篇开始之前先扯点闲话，商业应用系统开发经历了三个阶段：

第一个阶段以计算为中心，分析设计围绕程序的运行效率，算法优劣，存贮优化来进行。90 年代的大学课程讲的都是这些。

第二阶段以数据为中心，分析设计围绕数据流进行，以数据流程来模拟业务流程。这也就是所谓的面向过程的分析模式。

第三阶段以人为中心，分析设计围绕人的业务需求，使用要求，感受要求进行。这也就是现在的面象对象分析模式。

使用 OO 方法建立商业模型必须先定义涉众。商业系统无论多复杂，无论什么行业，其本质无非是人，事，物，规则。人是一切的中心，人做事，做事产生物，规则限制人事物。人驱动系统，事体现过程，物记录结果，规则则是控制。无论 OO 也好，UML 也好，复杂的表面下其实只是一个简单的规则，系统分析员弄明白有什么人，什么人做什么事，什么事产生什么物，中间有什么规则，再把人，事，物之间的关系定义出来，商业建模也就基本完成了。这时候可以说，系统分析员已经完全了解了用户需求，可以进入系统建模阶段了。

书归正传，上篇笔者归纳了一些典型的涉众类型及他们的普遍期望。接下来，就是要将他们这些期望定义出来。这个过程，就是业务用例获取的过程。笔者可以跟大家分享的经验是通过以下步骤进行，这些步骤并非唯一正确，对于经验不多的系统分析员来说，这些步骤很有指导意义。

笔者做了一个建模实例，有需要有读者请到笔者的 BLOG 资源中心下载，实例以上一篇所述网上图书馆需求为蓝本建立了业务用例模型，之后的概念模型、系统模型则抽取了其中的借阅过程作为例子。不记得了可以后头找找。

4.1 建模第一步

从涉众中找出用户。并定义这些用户之间的关系。在 ROSE 中，应该使用 business actor 类型。参考上一篇的需求描述，下载实例

4.2 第二步

找出每个用户要做的事，即业务用例，在 ROSE 中应使用 Business use case 类型。请参考《用例的类型与粒度》一文以帮助确定用例的粒度。笔者强烈建议为每一个 business actor 绘制一个业务用例图，这能很好的体现以人为中心的分析模式，并且不容易漏掉 business actor 需要做的事。至于以参与者为中心的视图容易漏掉某个业务用例的参与者的担心，可以在第四步中得到消除。下载实例

4.3 第三步

利用业务场景图帮助分析业务流程，在 ROSE 中，这个阶段最好使用活动图 Activity diagram。在这个阶段，业务场景图非常重要，在绘制过程中，系统分析员必须采用第一步中定义的用户名字作为泳道名，使用第二步中定义的业务用例名作为活动名来绘制。必须这么做的原因是，如果你无法把利用已经定义出来的 business actor 和 business use case 完备的描绘业务流程，那么一定是前面的定义出问题了，你需要回头审视是否 business actor 和 business use case 定义不完善或错误。如果不是所有的 business actor 和 business use case 都被用到，要么应该检查业务流程调研时漏了什么，要么应该检查是否定义了一些无用的 business actor 和 business use case。同时，绘制业务场景图也非常有助于选择合适的用例粒度并保持所有的用例都是同一粒度。下载实例

4.4 第四步

绘制用例场景图。与业务场景图不同的是，用例场景图只针对一个用例绘制该用例的执行过程。笔者仍然强烈推荐使用 activity diagram。在用例场景图的绘制中，必须使用第一步中定义的业务用户作为泳道。必须这么做的原因是，它能帮助你发现在定义业务用例图时的错误，比如是否漏掉了某个业务用例的潜在使用者。不是每个业务用例都需要绘制场景图，只有两三个步骤的业务用例是不必一定绘制业务用例图的，但仍然需要在业务用例规约文档中写明。下载实例

4.5 第五步

从第三步或第四步中绘制的活动图中找到每一步活动将使用到的或产生的结果。这是找到物的过程。找到后，应当建立这些物之间的关系。在 ROSE 中，这称为业务实体模型。应该使用 business entity 类型。下载实例

4.6 第六步

在上述过程中，随时补充词汇表 Glossary。将此过程中的所有业务词汇，专业词汇等一切在建模过程中使用到的需要解释的名词。这份文档将成为模型建立人与读者就模型达成一致理解的重要保证。

4.7 第七步

根据上一篇中提到的业主，老板等涉众的期望审视建立好的模型，确定业务范围，决定哪些业务用例在系统建设范围内。那些不打算纳入建设范围内的业务用例有两种情况，一种是该业务用例是被调用一方，那么应该把它改为 boundary 类型，意味着将来它是一个外部接口。另一种是该业务用例主动调用系统内业务用例，那么应该将它改为 business actor 类型。与普通 business actor 不同的是，由业务用例转换而成的 business actor 不是人，而通常是一个外部系统进程，因此应该在被调用的系统内业务用例与它之间增加一个 boundary 元素，意

意味着我们的系统将为这样一个外部进程提供一个接口。严格来说，那些需要纳入建设范围的 business use case 应当对应的生成一个 business use case realization，以后的设计工作将归纳到这些实现用例中。但笔者觉得这一步并非很关键的，实际中本人也经常省略这一步，而将协作图，象活动图，类交互图等直接在 business usecase 下说明。不过本实例中笔者还是按照正规方法来建模的。下载实例

需要说明的是，上述的步骤并非一次性完成的，在每一个步骤中都可能对以前步骤的调整。即使建模已经完成，当遇到变化或发现新问题，上述步骤应当从头到尾再执行一次。这也是 RUP 倡导的迭代开发模式。

经过以上的步骤，我们已经建立了一个完整的业务模型。但这决不是建模工作的全部，以上过程只说明了建立一个完整业务模型的过程，不能说这样就建立了一个很好的业务模型。因为上述的过程中并没有提及业务分析过程。分析过程全凭系统分析员的经验，对 OO 的理解和对行业业务的把握能力，对原始业务模型进行归纳，整理，抽象，重构，以建立一个更高效，合理，扩展性更强的模型。这个过程无法以步骤说明。或许以后笔者会专门针对模型分析写点东西。另外除了模型，还至少需要写业务架构文档、用例规约和补充用例规约三种文档。因为模型虽然可以较好的体现业务架构，但很不好表达业务规则和非业务需求，这些需要在文档中说明。例如用例的前置条件和后置条件就是一种业务规则。读者可以在 RUP 文档中找到这些文档的模板。

预告：下一篇笔者将讲述如何根据业务模型建立系统概念模型。这里先说一点，系统概念模型建立最主要依据的是第三步、第四步、第五步建立的业务/用例场景和业务实体模型。这也突显了场景和实体模型的重要程度。

5 用户、业务用例和业务场景

很久没有动笔了，这期间承蒙许多朋友的喜欢和鼓励，再不写点东西就对不住这些朋友了。

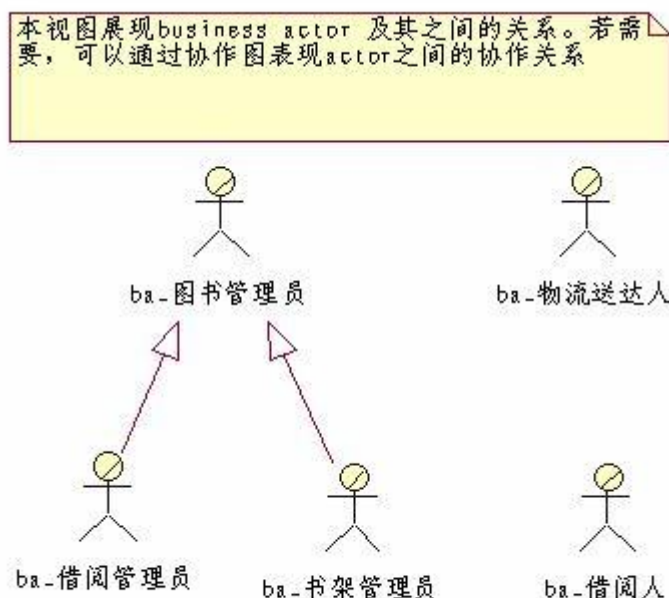
写点什么呢？按照原先的设想，应该开始动手写如何从业务用例转化到概念用例和系统用例，不过老实说这一步需要的是经验居多，而很难找出一个普适的步骤来。先暂时放一放吧，以后一定会写到的。上一篇讲到用例分析的一般步骤和方法，也给出了一个实例，不过没有做更进一步的说明，所以这一篇，笔者决定先罗嗦罗嗦之前的内容，说说业务建模中各种图的使用，以及它们对需求的贡献。

在说明实例之前，再重复一下的需求，并提醒读者下载实例，本文下面只会从实例中挑选很少一部分来说明，对照实例读者将能更好的理解。好了，让我们先回头看看需求吧，图书馆主任是这么说的：

我们原本是一个传统的图书馆，传统的借书方式要求读者亲自来到图书馆，这显得非常不方便，而且随着藏书的增加和读者群的增长，尤其而且大量的读者到图书馆，使得图书馆的场地不足，工作人员也不够了。所以想到借助网络，让读者通过网络借/还书，这样可以省掉大量的场地维护和工作人员成本支出，同时计算机可以方便的检索目录，让读者可以足不出户借到需要的书。为了把书送到借阅人手里，我们已经联系了 A 特快专递公司和 B 城市物流公司，初步达成协议，由他们往返借阅人和图书馆之间把图书送出和收回。读者在网上出示和验证借书卡，找到他们需要的书，提交申请，图书管理员确认后，就会通知物流公司来取书，当读者拿到书之后，物流公司需要把读者的签单拿回来以证明读者已经拿到了书。当然这个过程中，读者是需要付费的。还书也是基本同样的过程。

5.1 用户

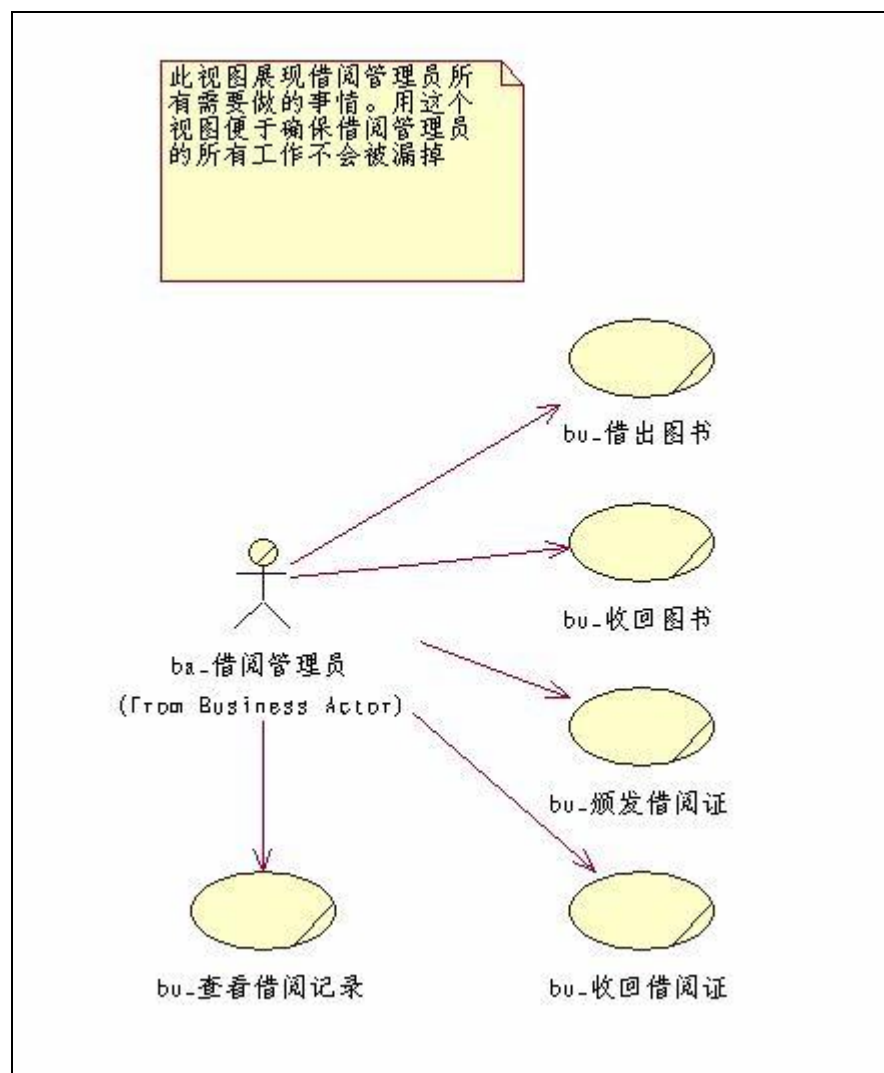
还记得上一篇是怎么说的吗？第一步是从涉众中找出用户。并定义这些用户之间的关系。这里的用户是指将与要建设的系统发生关系的那些涉众。通过下图表示。这张图里要绘出所有用户，以及它们之间的关系。需要说明的是，这里的用户指的是业务用户，并非将来系统中的“角色”，虽然将来它们可能就是。这张图的意义在于，清楚的表明将来的系统是为谁在服务，他们都是干什么的，有什么特点，有什么关系。对用例方法来说，这是最基本的出发点。用例，是以人为本的。



5.2 业务用例

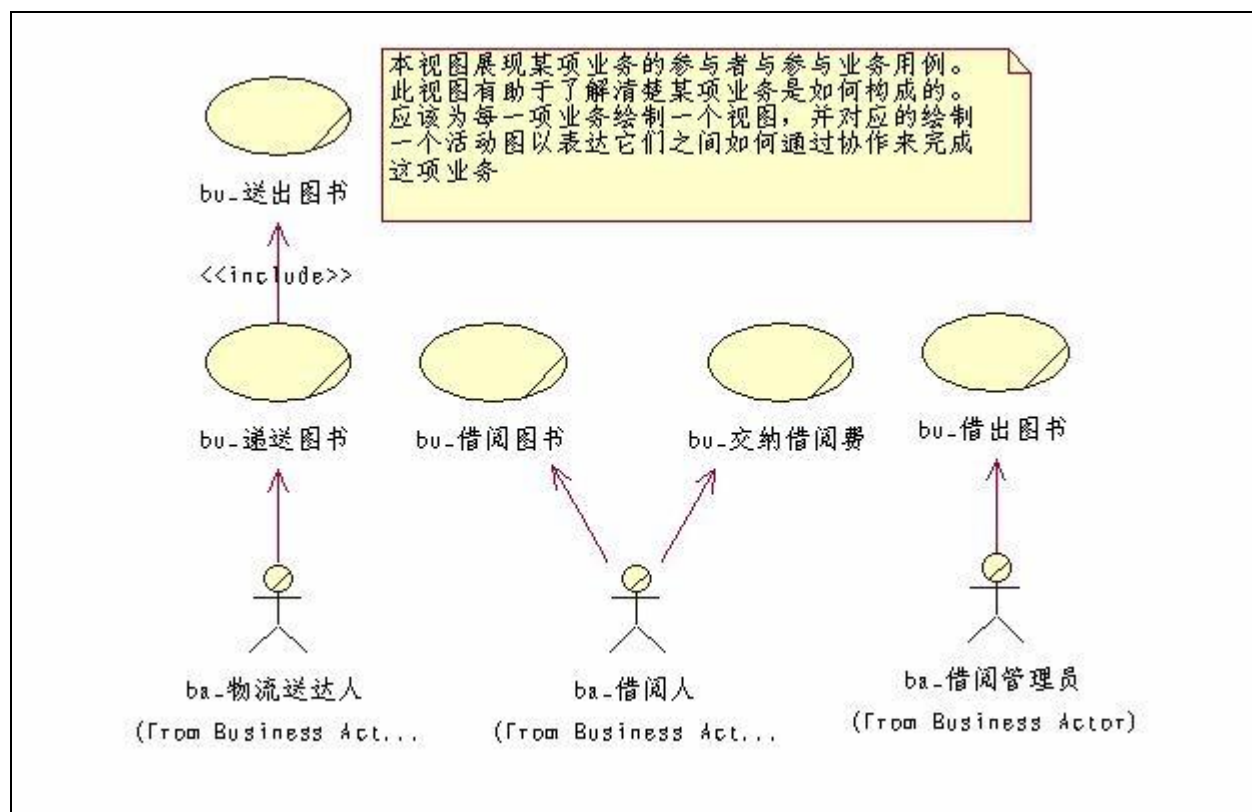
第二步，找出每个用户要做的事，即业务用例。业务用例来自于系统分析员对上一步中所有用户的访谈，总结和归纳（笔者正在考虑写一写系统分析员调研技巧方面的文章，会对访谈技巧，归纳方法有所描述）。笔者建议从每个用户的角度以及从每项业务的角度来绘制业务用例图，就象下面这样：

5.2.1 用户视角：



从用户视角查看每个用户在系统中将参与什么业务。这个视图的意义在于，调研对象一眼就能看出来，这个模型是否已经涵盖了他所有需要做的事。

5.2.2 业务视角：

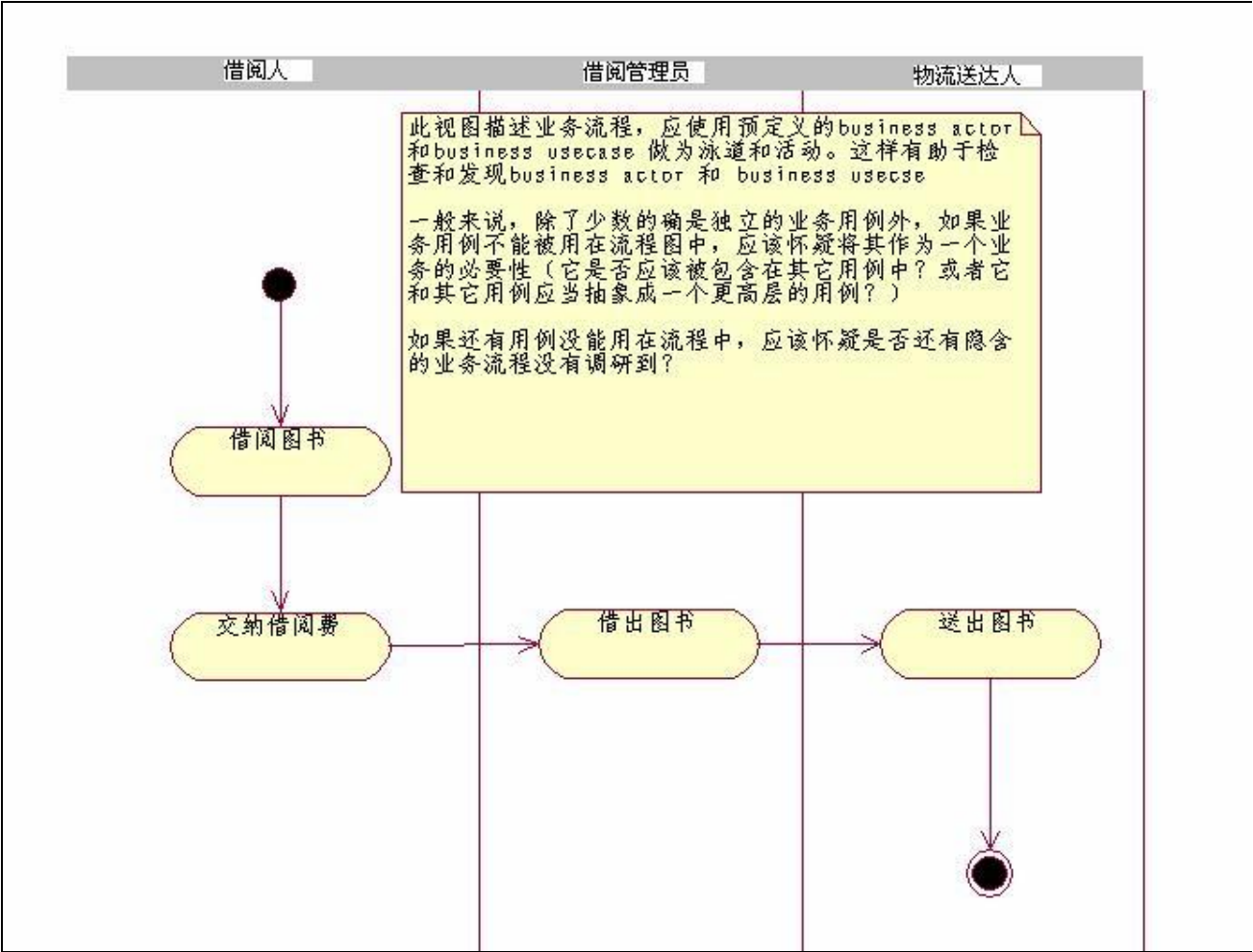


从业务的视角查看每项业务是由哪些用例和哪些角色参与完成的。这个视图的意义在于，需求研讨会上业务专家可以一眼看出这个模型是否已经能够完整的表达业务。

5.3 业务场景

第三步，针对每项业务视图，应该绘制业务场景图，用活动图详细描述这些用户、用例是如何交互来完成这项业务的。就象下面这样：

5.3.1 借阅图书业务过程:



业务场景图可能不止一个。同样一项业务，会有很多种不同的业务实现方式，例如借阅图书业务，就有可能第一次借书，又还书又借书，VIP 客户借书，借书时借证已经到期....等等许多种场景。对于这些场景来说，每一个都不能漏掉或省略，至少必须在文档中体现出来，除非已经很明确这个业务场景不包括在系统范围之内。这些业务场景图的意义在于，它已经绘制出了一份系统蓝图，将来的系统建设很大程度将依据这些场景图进行。

经过上面的三个步骤，我们得到了用户、业务用例以及业务场景。这三项工作成果已经形成了基本的需求框架，并圈定了业务范围。就笔者的工作习惯而言，在得到这三个成果后，就会暂停调研，而通过评审会，研讨会等形式充分论证这

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

些成果的正确性和完备性。求得业务专家，用户代表，开发方，项目经理等各方的一致认可，将其作为第一份基线。笔者很少在这个基线形成之前深入细化需求，因为需求过程，或说用例过程是一个自顶向向下的过程，RUP 中的每一个迭代实际上仍然是一个瀑布模型，大家都知道，如果上一步没有形成基线就进行下一步，对瀑布模型来说是无法控制的。

好了，这一篇就到此为止，下一篇继续讨论用例场景，用例文档等建模过程。

5.4 相关评论

ROSE 活动图的 Activity 都需要手工重新建立吗？

版主，我在学习您的举例过程中，有一个使用过程的疑问：就是在用 rose 建立活动图时候，Activity 都需要手工重新建立吗？我的意思是想通过拖拽“bu_借阅图书”到泳道里来形成 Activity，但总不成功。

通过您的文章开始入门的学习者 评论于：2006.10.19 09:18

当然不会成功的。BU 和 Activity 是完全两个不同的东西，从 UML 语义上说完全没有相关性，更不可能相等，所以不可能直接从 BU 变成 Activity。

你可能误解了我文章的意思，我说 Activity 名字要用 BU 的名字，是因为这样是一种很好的办法来检验和完善用例获取。你也可以用完全不同的名字，也可以完全不对应。只是这样做对需求过程没有好处。

coffeewoo 评论于：2006.10.19 10:07

有的系统的用户很少，比如一个游戏项目，也就玩家一个用户。

另外比如要做一个 mp3 解码库，那么用户就是 这个库的使用人。

这些项目，需求分析跟您讲的这些系统分析有很大区别，不知道能不能用 RUP 的方法，如何使用呢？

w8u 评论于：2006.11.13 16:23

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写, 原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成, 并发给作者本人, 再由作者本人向外发布的。本文允许自由传播, 仅供个人学习之用, 禁止用于商业行为, 如出版物, 培训教材等。读者在使用本文时请尊重作者之著作权, 勿篡改或删除或改编本文。若有非个人的任何公司, 组织和商业机构想采用本文之全部或一部分, 请与作者联系, 勿私自使用。若出版商有意出版此文, 请与作者联系。谢谢合作。

肯定是可以使用的, 但是视角要有所改变。在 UML 定义中, actor 并非是指人, 而是指某个边界之外的, 要与这个边界以内的事物进行交互的事物。因此关键是找到边界与 actor。

我觉得游戏的用户并不是玩家, 而是游戏控制器! 例如 CS, 一个动作就是一个独立的 acotr, 因为一个动作就是要从游戏引擎这个边界中获得一个结果。所以我认为所谓的用户是 a,w,s,d,alt,space... 等等, 每按下一个键等于一个 actor 发出它的要求。我没做过游戏, 只能瞎分析一下了:) 比如按下 w, 表示一个前进 actor 在发出动作, 它要做的事是分别向动作引擎, 声音引擎, 图形引擎发出请求, 由这些边界内的事物作出反应。如果按下鼠左键, 则向图形引擎, 枪械引擎, 弹道引擎等发出要求。

这个我纯属瞎说了, 懂游戏的别砸我, 呵呵

而解码器则纯属计算密集型程序了, 这种程序面向过程的方法更适合, 面向对象方法相反不适合了

coffeewoo 评论于: 2006.11.13 23:06

太谢谢了。茅塞顿开!

关于游戏 actor 的提示, 不管游戏是不是这么做的, 一下子突然打开了我的视界。原来 actor 还可以这么样。

mp3 解码库的东西, 用面向过程方法更好。也解开了我长时间的困惑。

谢谢了。

w8u 评论于: 2006.11.14 09:58

不知怎么就进到这儿来了, 觉得非常不错, 就把你的文章都下载打印出来, 认真的看了几遍, 非常谢谢。最近也在想用这种模式来分析设计, 但对于用户涉众的关系这块儿有个问题想问问, 图书管理员与借阅管理员和书架管理员是一个什么关系, 包含还是派生, 包含的话是不是说把图书管理员的业务划分成借阅管理员和书架管理员的业务, 如果是派生的话是不是在图书管理员那儿要定义些基

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

本的业务，而借阅管理员和书架管理员在继承这些业务的基础上，还有自己的业务。

boskin 评论于：2006.12.13 16:22

你的理解基本上是正确的。关于派生和包含的关系，也做 include 和 extend，或者就是 OO 关系中的泛化(继承)和聚合关系。这几种说法语义上是等价的。

派生是自顶向下的来看的，由小到大，由简单到复杂，由普遍到特例

聚合是从底向上来看的，是由小到大，由简单到复杂，由基本到全面。

coffeewoo 评论于：2006.12.13 22:28

向咖老大请教啊：

在第三篇中你说过业务场景图要把所有 business actor 和 business usecase 都用上，这章进一步说道“业务场景图可能不止一个”。是不是可以这么断定：可以画出 N 张业务场景图，每张场景图都包含了所有的 business actor 和 business usecase？

感激涕零！

qcrsoft 评论于：2006.12.18 18:16

No, no，不是一个场景要把所有用例都用上。而是说当完整的把表达用户需求的场景都画完以后，每个用例应该都能够至少一次的被用到。这样才能证明这个用例是有意义的，它是如何参与到业务场景中的。

coffeewoo 评论于：2006.12.18 18:52

6 用例实现、用例场景和领域模型

上一篇说到我们经过初步的业务分析，得到了用户、业务用例以及业务场景模型。这三项工作成果形成了基本的需求框架，并圈定了业务范围。这时应当做一份基线。

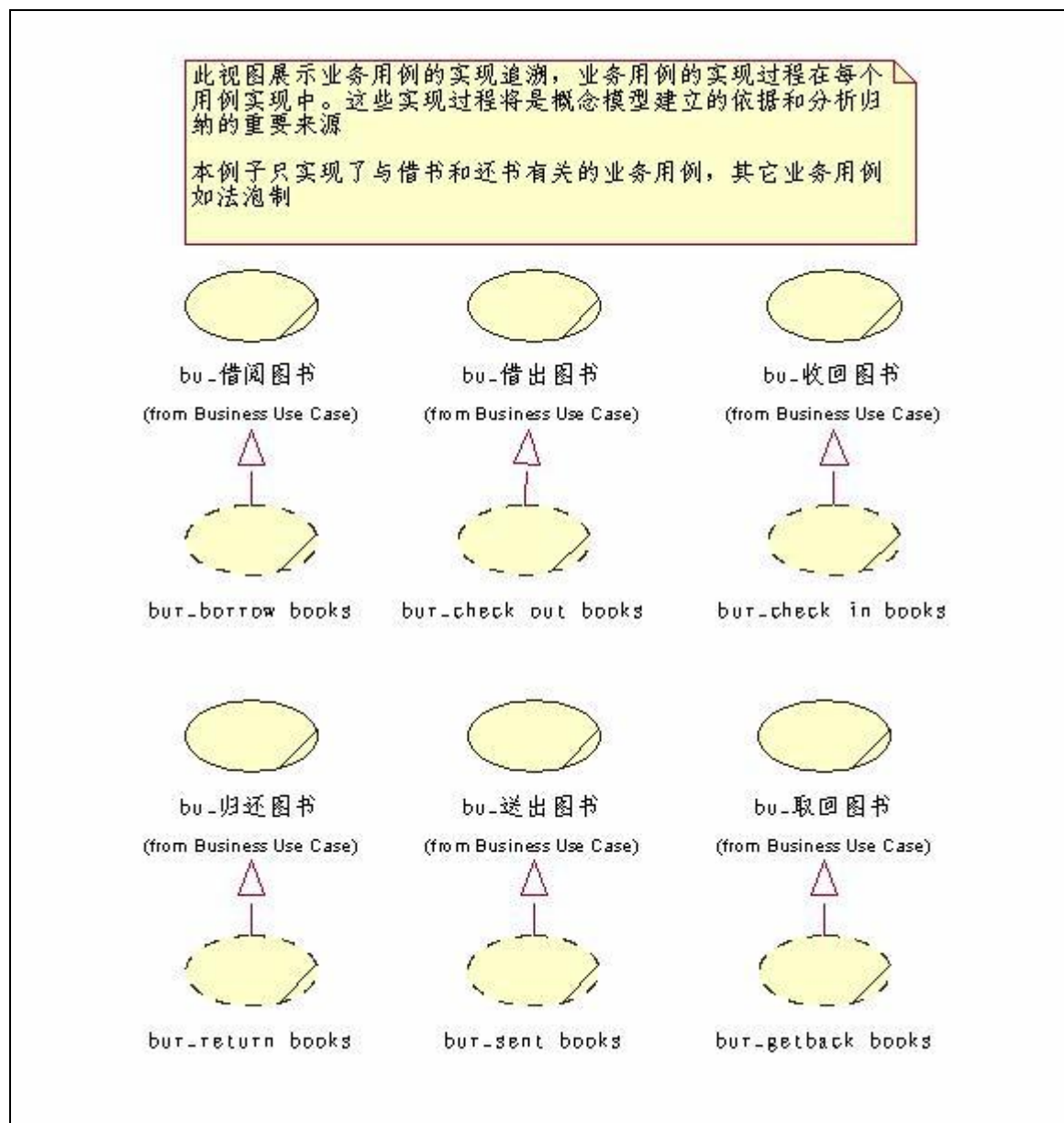
当然，第一份基线所包括的内容是非常粗的，要达到完整的需求说明还有更多工作要做。这一篇就来说说详细的需求过程和产出物，以及这些成果对需求的贡献。在开始之前，还是提醒读者下载实例，本文下面只会从实例中挑选很少一部分来说明，对照实例读者将能更好的理解。

上一篇确定了业务用例，以及业务场景。该场景只描述了业务框架，接下来要对业务用例进行场景分析。用例场景分析要用到三种视图，业务用例实现视图、业务用例场景、业务实体模型（领域模型），每个业务用例还应当写一份用例文档，也称为用例规约（UseCase Specification）。若有非功能性需求，例如性能要求，吞吐量要求等，还应当写一份补充用例规约。用例规约将在下一篇描述。

首先是业务用例实现视图。并非所有的业务用例都一定要最终在系统中实现，因此，这个视图的含义是表达由需求范围到系统范围的映射关系。这个视图没什么技巧，也可以省略，不过笔者建议不要省略。需求应当保持过程的连续和可追溯性，这是软件过程可控的重要保证。

6.1 用例实现

业务用例实现视图：



针对每个业务用例实现，应当对用例的实现过程进行场景模拟。上一篇是业务场景，而用例实现既然已经谈到“实现”，则应当将计算机包括进来，从人-机交互的视角来模拟业务场景。这是概念模型的一种，表达用户的实际业务在计算机环境下是如何实现的，给用户一个初步印象，告诉他们将来他们将怎样来做业务。请注意，虽然计算机已经参与需求描述，但是要尽量避免使用计算机术语，因为这时的文档仍然属于需求文档，是要与用户交流的，太多的计算机术语会大

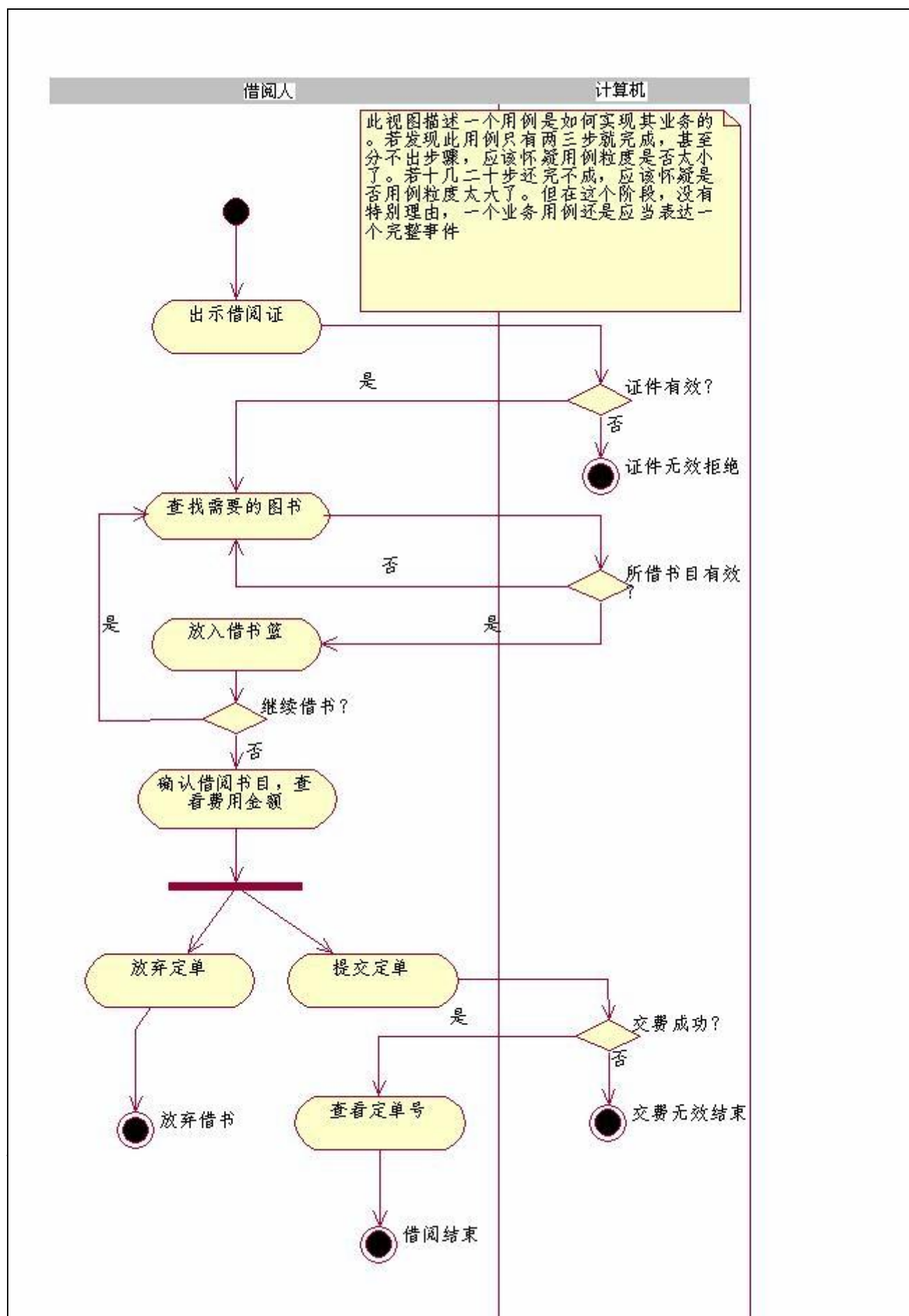
本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

大降低用户对需求的理解能力。霍金在写时间简史时曾经说过，在书中加入哪怕一个数学公式，都会让书的销量减半。业务用例场景是概念模型的一种，但不是概念模型的全部。概念模型本篇不打算讨论，简单说一下，概念模型主要包括业务架构和系统原型。

应当在业务用例实现里添加活动图用以描述用例场景，下图为示例，用活动图绘制。如果有多个场景，则应当绘制多个场景图。

6.2 用例场景

业务用例场景(借书过程):



用例场景有另一个重要意义，是帮助系统分析员发现和定义业务实体。业务实体一般来说就是调研时用户所提供的各类表单或报表，但在很多情况下，并非每一份表单就是一个业务实体，所有业务表单也不一定涵盖全了所有业务实体。很多系统分析员声称业务实体的发现过程是全凭经验的，到底有哪些业务实体，靠经验进行提取。笔者要说，经验固然重要，但经验有一个最大的缺陷---不能重复和验证。即，这些实体是怎么从业务中提取出来的？它们是怎样参与业务的？这些实体已经足够支持业务了吗？凭经验分析者无法通过文档将这个提取过程记录下来，而脑子里的东西是无法共享和传承的，越大的团队，越复杂的项目，尤其是横向结构的项目组结构下，这个缺陷越严重。很多人觉得用 UML 和 RUP 描述的需求总是一块块分离的，不知道是怎么出来的，觉得很乱，原因就在于此。实际上，RUP 做需求，每一步都是可验证和回溯的。用例实现视图是一个例子，这里也是一个例子。

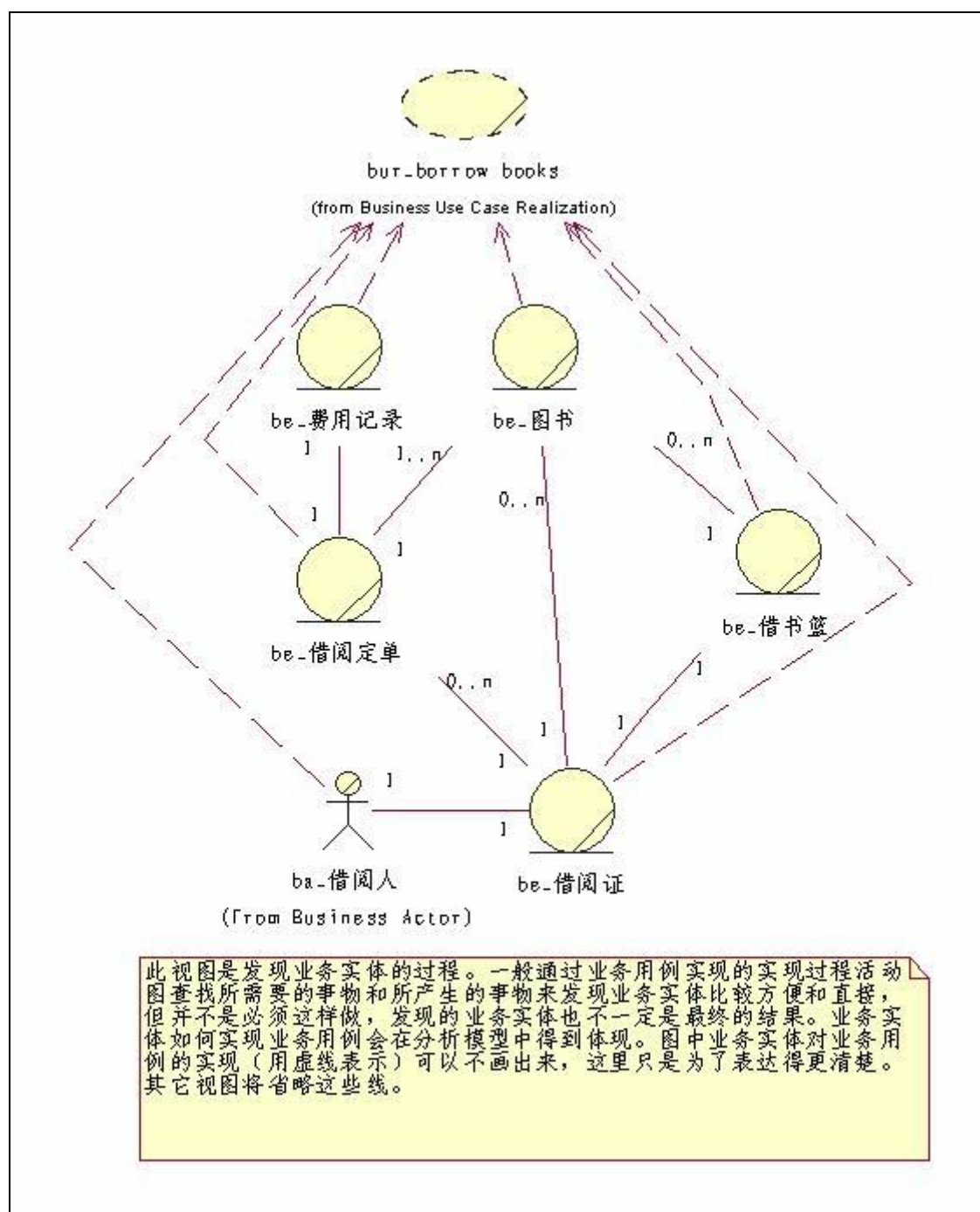
让我们看看上面的业务场景视图，每一个活动都有类似的命名：出示借阅证、查找需要的图书、放入借书栏.....看出什么来了吗？每个活动都是一个动作加上一个动作的受体。受体正是我们要寻找的业务实体，这些名词就是实体的来源。在需求阶段，系统分析员不要去考虑什么抽象，什么模式，别急，那是系统模型做的事情。抽象了，还弄一堆什么 Factory 模式，Builder 模式之类的出来，用户能看懂吗？别忘了我们正在做的是需求文档，是做给用户看的。

观察上面的用例场景，分析出现的名词，我们得到一个个业务实体，再根据场景分析这些业务实体之间的关系。实际上就是大家都熟悉的 ER 模型，但是与数据库建模的视角还是有所差别的。数据库 ER 模型要受到数据关系范式的限制，而业务实体 ER 模型则不必理会这种限制。只要与现实物体符合就 OK。好了，罗嗦了一大堆，我们终于得到了我们的成果。

6.3 领域模型

借阅图书过程业务实体视图：

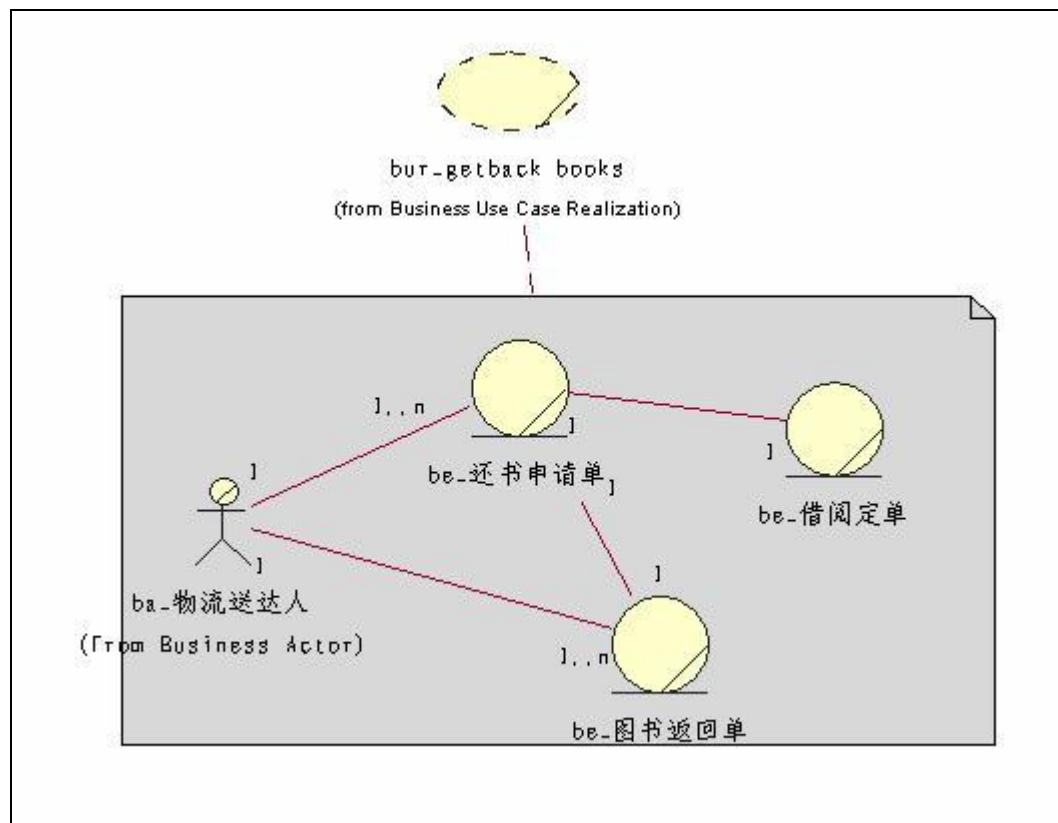
本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。



上图中画那么多虚线连接到业务用例实现是用来表示业务实体与业务用例实现之间的追溯关系的，这些线虽然麻烦，但是笔者强烈建议不要图省事。因为业务实体通过它们可以追溯到原始需求，再次重申，软件过程要可控，需求可追

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

溯是需要时时谨记的。当然，如果嫌麻烦，您也可以用下面的这种形式，是不是简洁得多呢？



经过以上的过程，我们得到了什么呢？往下看之前笔者建议您回想一下，总结一下。

第一、我们通用用例实现视图，从业务用例中找出了那些我们将在系统中实现的用例，并且记录了要在系统中实现的用例是如何映射到原始需求的。这提供了需求可追溯的验证。

第二、针对每个用例实现，我们引入了计算机，将实际的业务从人-机交互的角度模拟了执行过程。不仅得到了一个业务怎样在计算机环境下执行的概念模型，同时也给用户描述了他们将怎么和计算机交互以达到他们的目标。笔者提醒大家，用例场景非常非常的重要，后续工作就得靠它们了!! 绝对要认真对待，深入调研，不可漏掉一个场景，也不可模糊不清。

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

第三、通过对场景的分析，给了我们重要的线索去发现业务实体。而我们发现业务实体之后，又通过用例场景来验证这些实体是否支持了用例的实现。

现在请读者思考一下，如果记不清了，可以翻翻之前的文章。到现在为止，我们的需求是不是一步一步推出来的？从粗到细，从模糊到清晰，从原始需求到计算机的引入，是不是每一步都是可以追溯的呢？每一步的分析结果是不是都有方法来验证正确性和完备性呢？如果您之前迷惑于需求的各个阶段无法关联，也说不清分析结果是否是正确的，那么建议您再从头看看笔者目前已经完成的文章，找出这些线索来，相信您会对 UML 和 RUP 的理解提高一个层次的。

这篇文章该结束了。可是等等，到目前为止，虽然我们已经得到了不少产品，或可交付物，或成果，或 deliverable... 不管叫什么吧，我们已经做了很多工作了。不过作为需求来说，好象还缺点什么吧？对了，我们还缺少业务规则和业务实体的详细属性，这两个需求必不可少的内容，将在下一篇中讨论。敬请期待。

6.4 相关评论

这一个应该是分析模型吧，状态图类图时序图用例图只用其中一个就可以了

^_^

rwyx 评论于：2006.09.28 16:23

我觉得这不是分析模型，分析模型的特征是借助分析类，一般是边界，实体和控制类来“实现”用例的。这里用活动图只是一个场景模拟，表达人机交互的情况。还是应当属于需求范畴的。

coffeewoo 评论于：2006.09.28 18:17

没有真家伙，写不出来这么清晰的文章来。

让我回到了初中时，证明几何题的推理过程。每个步骤都有论据，都有原因，都有起源。

很佩服 楼主。

w8u 评论于：2006.11.13 15:06

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

呵呵，我花了几年时间思考这些问题，这些应该说是实践与思考的结果。我跟大家一样走过一段很痛苦的路，UML 概念很多，每个独立对象都有其定义，但就是没办法说清一个完整过程应该怎么把这些东西串起来。我写这些文章就是把我自己的思考表述出来，比掌握 UML 每个定义更重要的是知道它们应该怎样发挥作用，怎么用。软件工程知识的积累帮了我很大的忙，我发现只有当这些 UML 元素与软件工程有机结合起来才有意义。

coffeewoo 评论于: 2006. 11. 13 23: 16

吁，一口气看到这里，受益非浅，另外针对这篇文章，想听听 coffeewoo 对业务实体和领域模型这两个概念的见解

spiritj 评论于: 2006. 12. 12 17: 32

在我看来，业务实体模型基本上就等于领域模型了。虽然它们是两个名词。

领域模型的目的，是用一些事物，来表达或说建立起该问题领域，行业领域，业务领域... 的构成。是一种逻辑化了的结构模型。

任何一个领域，都有两个方面的描述，结构性的和功能性的。并且大部分时候结构决定功能。在 UML 中，功能性的描述是由用例来承担的。而结构性的描述则是由领域模型来承担的。实际上在 UML 中结构性的东西是由实体这一元素来代表的。所以我认为领域模型基本上就等于业务实体模型。只是在这时，业务实体之间的关系不能纯粹理解为是计算机逻辑关系，而是其代表的问题领域内事物的结构化描述。

coffeewoo 评论于: 2006. 12. 14 10: 13

我对领域模型一直搞不明白，看了你上面写的“业务实体模型(领域模型)”，立刻大爽，可 GOOGLE 了一下，又出来另一种解释，你瞧：

<http://www.itisedu.com/phrase/200604231350225.html>，在这里领域模型被解

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

释为业务对象模型，业务对象模型又被描述为“从业务角色内部的观点定义了业务用例”，又把我给看糊涂了。咖老兄能不能给拨开迷雾呀，太感激了

qcrsoft 评论于: 2006.12.18 18:36

你给的链接我看了，基本观点是一样的。不过是我所不喜欢的表述方式，晦涩难懂。

我的表述是功能+结构，所谓功能，也就是由用例表达的东西，也就是你给的文章中所说的“业务用例实现显示了协作的业务角色和业务实体如何执行某个工作流程。”中的工作流程。

所谓的结构，就是业务角色和业务实体。

业务角色也是一种特殊的业务实体。所以你看我例子中角色是和业务实体在同一张图里的。

工作流程由业务场景图表达了，就在本文中。而所谓的“从业务角色内部的观点定义了业务用例”，请看本文的最后一张图，可不就是从内部观点去定义业务用例的么？最后一张图完全可以解释为用例是由 XX 角色操作 XX 实体... 来完成的。与你所给文章中的观点并不冲突。“在 UML 中，功能性的描述是由用例来承担的。而结构性的描述则是由领域模型来承担的。”由于我把功能性的，动态部分划分到用例视图里了（实际上我觉得这样更合适），“所以我认为领域模型基本上就等于业务实体模型”。

这么解释不知清楚了否

coffeewoo 评论于: 2006.12.18 19:08

补充一点，我原话是这么说的，“任何一个领域，都有两个方面的描述，结构性的和功能性的”。因为我认为功能性的归到用例模型中更合适，所以剩下就只有结构性的，也就是实体模型了。应该这样来理解，功能性的被用例模型取代，不需再次描述，“所以我认为领域模型基本上就等于业务实体模型”。

coffeewoo 评论于: 2006.12.18 19:19

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

大多数的书和文章只有懂的人才看的明白，咖兄弟的文章是写给不懂的人看的。

qcrsoft 评论于: 2006.12.19 11:16

7 用例规约的编写--业务规则和实体描述

上一篇我们图形化建模的部分基本上完成了，得到了业务用例模型，这帮助我们获得了功能性需求。得到了业务场景和用例场景，这帮助我们获得了面对业务的执行过程描述和概念（逻辑）模型，让我们知道业务将如何的运作。得到了用例实现以及领域模型，这帮助我们得知哪些业务用例将在系统中实现，对应这些用例，哪些业务实体将会被包括进来，以及它们如何帮助业务实现。上一篇我们也留下了悬念，对于业务执行过程来说，除了以上的成果，我们还需要知道业务规则，以及业务实例的属性。即我们要如何做以及做什么。这一篇就来讨论这些内容。

7.1 业务规则

先说说业务规则。笔者习惯将业务规则分为三种。

7.1.1 全局规则

这种规则一般与所有用例都相关而不是与特定用例相关，例如 actor 要操作用例必须获得相应的授权，用例的操作与授权级别相关，或者用户在系统中的所有操作都要被记录下来等等。这类规则笔者习惯于，并且也建议将它们写到用例的补充规约里面去，因为它们一般与具体的业务功能性要求没有直接关系。有时候，这类规则也被写到软件架构文档中。关于用例补充规约以后再讨论。

7.1.2 交互规则

这种规则产生于用例场景当中，例如当提交一份定单时，哪些数据是必须填写的，用户身份是否合法。当然也包括一般理解上的业务流程流转规则等，例如金额大于一万元的定单被定为 VIP 定单进入特殊流程等。这类规则一般要写到用例规约中。交互规则实际上还有两个是比较特殊的，一个入口条件，也称为前置

条件，即 actor 满足什么条件才能启动用例；另一个是出口条件，也称为后置条件，即用例结束后会产生哪些后果。稍后参看示例。

7.1.3 内禀规则

所谓内禀规则是指业务实体本身具备的规则，并且不因为外部的交互而变化的规则。例如，每张定单至少要有一件商品，同一类商品数量不能大于 5 件等。同时也包括大家所熟悉的数据效验规则，例如身份证号必须是 15 或 18 位，邮编必须是 6 位等等。这类规则是业务实体的内在规则，因此应该写到领域模型文档中，稍后参看示例。

读者或许对把规则这么分类有不同的习惯和理解，可能会觉得麻烦。但是笔者这么做有充分的理由。首先，全局规则与具体用例无关，它实际是系统应该具备的特性，这些规则把它分出来目的就是为了让系统去负责这些特性的实现。读者可以结合实际考虑一下，通常授权，事务，备份等特性都是由系统框架去实现的，具体的功能中并不去实现它们。如果读者有过基于某个框架开发应用的经历的话一定会认同笔者的话。其次，交互规则是在用例场景当中产生的，它们规定了满足什么条件后业务将如何反应。通常，这部分规则是最复杂，也最不稳定，最容易变化的。大家所说的需求经常变更相信绝大部分就来自于此。因此将这些规则单独列出来并给予关注和管理是很有必要的。同时这部分规则通常在系统中以 Control 类去实现，MVC 模式如此，SOA 架构中的 BPEL 也是如此。如果需求无可避免的要变更，那么，将交互规则单独提取出来，并设计成为扩展性很强的结构就是一种有效的应对手段。第三，内禀规则与外部交互无关，不论谁，在什么情况下提交定单，必须有至少有一件商品；不论你在哪个国家，在什么公路上开车，刹车都是必不可少的。这种内在的性质让你想到了什么？面向对象的封装原则对吗？这类规则应该实现到你的实体类中去，不论你的业务实体是 EntityBean, JavaBean, POJO 还是 COM+，根据面向对象的封装原则，内禀的逻辑一定不要让它暴露到外部去。

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

这么分类以后，对软件成熟度比较高的组织来说，提供了很好的 Level Reference。如果你是架构师，应该关注的是全局规则；如果你是设计师，应该关注的是交互规则；如果你是程序员，应该关注的是内禀规则。

在这里笔者想说点题外话：)。笔者曾经看过一篇《架构师已死》的文章(很抱歉已经记不起出处和作者，如有冒犯之处请海涵)，作者的观点认为架构设计完成后，通常到最后改得面目全非，既然一开始不可能考虑到所有可能，何不如在开发过程中持续总结，重构，最后架构会自然而然出来的，如果这样，架构师有何用处呢？笔者认为这个说法是有一定道理的，不过要看软件组织架构和软件过程的定义，不可一概而论。《架》一文的作者是 XP 的 fans，对 XP 软件过程来说，本来就不需要架构师这个角色，甚至连设计都不需要，开发，测试，改进，重构...，当然，从一开始就没打算按照一个 stable 的方法去做，架构师也就没有存在的必要。架构师在 XP 中已死，不过在 RUP 中还活着：)。软件界一直对 XP 和 RUP 有着争论，笔者无意在本文中界入这个争执，只是话已经到此，就顺便表达一下笔者观点。XP 和 RUP 都是非常优秀的软件方法，只是它们各有各的适应范围。对于中小型公司和中小型软件来说，XP 是非常有效的管理方法，它能大大降低管理、开发成本和技术风险。不过要是对于大公司和大型项目来说，XP 就不适用了，这时 RUP 却非常适合。你能想象洛克希德·马丁公司用 XP 的方法来开发 F-35 战斗机会是一个什么情形吗？没有人清楚的知道将来的飞机整体是什么样，反正先造一架出来，摔了找找原因，改进改进，重构一下，再造一架...再摔了，没关系，咱们拥抱变更，再造就是了。呵呵，那 XP 什么情况下适用呢？如果你是一个杂货店的老板，不知道什么样的商品受欢迎，没关系，先各进一小批货，卖上一段时间，受欢迎的货品多进，不受欢迎的不进，跟顾客多交流，顾客喜欢什么商品咱就进什么，不断改进，最后一定会顾客盈门的。这时如果这个老板先做商业分析，客户关系分析，消费曲线分析...还没开业呢，估计就破产了。另外，RUP 和 XP 也不是非此即彼的关系，在造 F-35 的过程中，对整体飞机

来说，用 RUP 是适合的，具体到发动机的提供商，倒是大可 XP 一把，最终只要提供合格的发动机就 OK 了。

题外话说了不少，书归正传。业务规则分类以后，就应该按分类去调研了。

全局规则很难从用户处调研得来，通常这方面是用户的盲点。这主要是由有经验的系统分析员，或架构师以及客户方的 IT 部门(如果有的话)，从业务特点、应用环境、行业规定、法律规章等等方面去总结，再求得客户方的认可。

交互规则从用例场景而来，每一个场景，场景中每一个交互的过程可能都隐含着规则。这就需要与客户多讨论。请参考本系列文章的第 3 篇关于涉众的讨论，交互规则最主要的来源是业务提出者和业务管理者，最好不要去找业务执行者。

内禀规则是针对业务实体的，因此要对每个业务实体的属性进行罗列，并找出它们的规则。内禀规则最主要的来源是业务执行者，需求人员应该更多的去与他们交流。

现在来谈谈业务实体的属性。业务实体的属性在这个阶段应该用业务术语来描述，而非计算机术语。调研范围是上一篇模型中得到的领域模型中的每一个实体，而属性的原始来源是客户的各类实际表单，以及在交流过程中客户提出的各种要求。如果业务实体有状态，并且是比较复杂的，那么在建模的时候应该有一个状态图来说明。请参看本系统文章提供的建模实例中'图书'业务实体下面的状态图。请读者注意，在本文后面提供的例子中，业务实体描述看上去象是一张数据表，但它绝对不是数据表。它是对业务中实体属性的业务角度描述。系统分析不是做设计，脑子里不要有任何关于设计或实现方法的想法，这些想法会误导你做出不适合的决定。你的职责是通过一个合理的模型完整的将业务描述出来，并求得客户方的认可。任何替客户和架构师，设计师甚至程序员“着想”的方法其实都是不恰当的。

最后本文提供一个用例规约的例子，每个用例都应该写一份用例规约。本文提供的这个例子基本上来自 RUP 提供的模板，不过在实际工作中笔者做了些修

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

改，供读者参考。这个例子的蓝本还是本系统文章一直在使用的网上图书馆的实例。点这里下载用例规约例子，点这里下载建模实例。

到这里，需求过程差不多也该结束了。但是我们的确还有一些工作要做。如果读者所工作的组织是用 RUP 来做需求的，而客户方或者监理方未必会对这种需求文档表示满意。他们会以国标来要求你。同时，到目前为止，我们产生的成果都是一些分离的图形和文档，对于客户来说，这的确是不好的文档结构。难道客户的采购清单里还需要包括 Rational Rose，这样才能阅读你提供的需求文档吗？显然这是不可能的，所以，给用户提供的文档还是以传统的《需求规格说明书》为好。下一篇就讨论一下如何将我们的分析成本集成到《需求规格说明书》中。顺带讨论一下用例补充规约和系统原型的产生以及它的作用。敬请期待。

7.2 用例规约示例

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

用例名称	bu_借阅图书
用例描述	借阅人通过此用例向系统查询并提交借书请求
执行者	借阅人
前置条件	1. 借阅人借阅证件在有效期内 2. 借阅人没有逾期未归还的图书
后置条件	1. 创建借书定单 2. 更新借阅人借阅记录
主过程描述	1 用户用借阅证提供的帐号登录系统，计算机显示我的图书馆界面 2. 用户选择查询图书，计算机显示查询界面 3. 用户按书名、作者、出版社查询，计算机显示查询结果 4. 用户可单选或多选书本，并确认借阅。计算机显示确认借阅图书清单。 5. 用户选择确认借阅，计算机显示借阅定单及费用 6 用户选择提交定单，计算机显示提交结果和定单号 7. 计算机执行后置条件。用例结束
分支过程描述	2. 1. 1 用户选择查看原有定单，计算机执行 4; 4. 1. 1 用户可单选或多选书本，放入借书篮，计算机显示借书篮现有内容 4. 1. 2. 1. 1 用户选择继续借书，计算机执行 2; 4. 1. 2. 2. 1 用户选择提交借书篮，计算机执行 4 4. 2. 1 用户选择放弃，计算机执行 2; 6. 1. 1 用户选择保存定单，计算机保存并执行 1; 6. 2. 1 用户选择放弃，计算机执行 1;
异常过程描述	1. 1. 1 借阅证已过期，拒绝登录，用例结束 1. 2. 1 借阅人有逾期未归还书本，启动 bu_归还图书用例 5. 1. 1 用户余额不足，计算机显示余额和所需金额 5. 1. 2. 1. 1 用户选择续费，启动 bu_交纳借阅费用用例 5. 1. 2. 2. 1 用户选择放弃，计算机执行 1
业务规则	4. 至少选择一本, 至多选择三本
涉及的业务实体	Be_费用记录 Be_图书 Be_借书篮 Be_借阅定单 Be_借阅证

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

7.3 业务实体描述示例

实体名称	Be_图书		
实体描述	每本图书都经有上架，预定，借出，返回待查和下架几个状态，详细请参看图书状态图		
属性名称	类型	精度	说明(属性的业务含义及业务规则)
图书编号	字符	12	图书类别编号(3 位)+图书购入年份(4 位)+流水号(5)位
图书分类	字符	3	图书的分类
名称	字符	100	书本的封面名称
作者	字符	20	书籍的作者
出版社	字符	100	书籍标明的出版社
出版日期	日期		书籍标明的出版日期
版本信息	字符	100	书籍标明的出版社
简介	字符	1000	书籍的内容简介，上架时录入
状态	字符	1	书籍的状态，请参看图书状态图

7.4 相关评论

谢谢! 弱问一句, 不是计算机专业的, 但现在做需求分析, 需要看哪些书啊, 还有希望吗... 不要砸啊

hehe 评论于: 2006.09.06 11:40

可以负责的说, 计算机行业是专业技能门槛最低的行业之一, 问题只是很多公司靠证件招人搞得看似 IT 高深莫测。事实上, IT 行业所需的技能没什么了不起, 比较容易学。但是 IT 行业却是一个考脑子的行业, 新东西新概念新技术层出不穷, 脑子是比较累的。

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

其实纯粹的需求分析对计算机的专业知识要求并不高，你可以不会编写程序，可以不懂设计，也能做需求分析，只是往后发展会遇到障碍的，毕竟架构知识要来自于程序，设计，系统经验的积累。

需求分析是一门相对独立的技术，我本人这方面的书看得不是很多，主要经验都是来自工作当中。我记得有一本需求工程的书，讲的内容比较全，可以去找找。不过我建议，如果想在 IT 发展得更好，应该多了解一些程序，OO 方法，设计，软件工程等方面的知识，不要求精，只要达到扩展知识面的目的就好了。

coffeewoo 评论于: 2006.09.06 12:56

楼主，我看完 6、7 两篇后，对一些名词不大明白，比如概念模型、领域模型等，请问这些概念如何区分，或哪里有这些概念的入门介绍？谢谢

Nicole 评论于: 2006.10.20 10:19

通常，用 UML 做系统分析时，会有业务模型，概念模型和系统模型。它们用于不同的目的，采用不同的视角来描述将要做的系统。简单来说，业务模型是指，将客户的业务用图形化的形式描述出来，这里是没有计算机的，只有业务。而概念模型则是在业务模型的基础上，对某些业务的问题域用引入了计算机的概念，描述这些业务在计算机层次上是如何表述的。所谓领域模型就是概念模型的一种，说的是，针对用户的某个业务，我们如何用实体（也叫 domain），人，交互等来表达它，换句话说，我们用计算机概念去替换业务概念。

至于系统模型，就是纯计算机角度了。接口，实体，控制，类，模式... 总之，就是用纯计算机语言去解释上述的两个模型。

coffeewoo 评论于: 2006.10.20 10:44

8 编写完整的 UML 需求规格说明书

为了让我们的需求更完美，这一篇所要做的工作也是必不可少的。这一篇将要讨论到的内容包括：用例补充规约，系统原型，以及需求规格说明书

终于到了快结束的时候了，这将是用例分析系列的最后一篇，结果是得到需求规格说明书，以结束需求分析的过程。经过前面七篇的工作，我们从最初的业务用例获取入手，获得了业务用例模型，这是我们的业务范围；经过分析得到了业务场景，这是我们的业务蓝图；经过规划，得出用例实现视图，这是我们的系统范围；经过再次分析，得到了用例实现以及领域模型，包括用例规约，业务规则和业务数据，这是我们的概念模型。仅从需求所需的必要元素来说，我们基本上已经完成了需求分析的工作。诚如上一篇结尾所说，为了让我们的需求更完美，这一篇所要做的工作也是必不可少的。这一篇将要讨论到的内容包括：用例补充规约，系统原型，以及需求规格说明书

8.1 用例补充规约

先说说用例补充规约。

之前我们得到的用例规约是功能性的，它们是针对 Actor 完成目标业务所需的功能性 Feature 的描述。然而我们所面对的系统除了功能性的 Feature 之外，总有一些是与业务功能无关的，这部分需求就是补充规约要涉及的内容。

什么样的需求与业务功能无关呢？一般来说，就是系统需求，例如可靠性，可用性，扩展性，易用性等等。用户提出，系统必须提供 7*24 小时的服务，它应该有一定随业务变化而适应的能力，系统的界面应当简单易学，具备基础计算机知识的人可以不经培训就能使用它等等。这些需求与具体业务要求无关，哪怕一个不实现，系统也能 Run 起来。但是这些需求又是如此重要，它们是系统达到用户期望必不可少的。甚至在用户看来，某些补充规约要求比业务要求还重要，某个业务要求没做好，用户或许能宽容，如果你说系统不能提供 7*24 小时的服

务，用户肯定是不能接受的。这些需求，就是要在 补充用例规约里面说明的内容。

笔者自己有个习惯，在上一篇也有所提及，就是习惯于把全局规则也写到补充用例规约里面。比如，用户提出，所有系统使用者在系统中的任何操作，都要能够记录下来；如果数据被更改，不论是 Modify, Add 还是 Delete，数据都要做一个备份；响应时间可能超过 1 分钟的功能，都要提供进度条等等... 这些全局规则在实际情况中，一般都是由系统框架，或某个中间件来完成的，它们是系统架构的一部分。因此这些需求虽然也是功能性的，但笔者认为将它们作为补充需求，与可靠性之类的系统需求一起，由较高层次的设计师或者是架构师来处理它们更合适。

那么补充用例规约到底是一个用例写一份还是全部集中在到一起呢？RUP 提供的模板是一个用例写一份，只要它们与该用例相关。笔者在实际工作中觉得集中在一份文档中似乎更合理。一是减轻很多的重复工作量，二是集中在一起更便于管理和验证。

至于补充规约的格式就没那么重要了，只要将用户提出的，或者用户未提出，但作为系统建设者知道系统要很好运行就必须去加入的那些特性，都一一写明白了，就 OK 了。当然，有时某个补充要求的确只与一个特定的用例有关，例如交纳借阅费，有一个可能的补充要求是保障安全性，包括数据安全，传输安全。其它用例则没有必要进入安全通道。这时，专门为交纳借阅费用例写一个补充规约也是合理并且推荐的方式。

再来说说系统原型。所谓系统原型根据目的不同，也分为好多种，本文无意深入探讨，大致说说，并只描述与需求过程相关的原型。例如，我们可能要使用一个全新的技术，为了验证其技术可行性，可能会开发一个小的原型，以掌握或证明我们能够使用这种技术，也证明这项技术能够支持后续的开发，这是一种验证性原型；我们有一个初步的想法，但不知往下能走多远，这个想法是否可行，也可以开发一个原型，这是一种探索型原型；我们要向别人说明某个产品，为了

形象化，也可以开发一个原型，以显式的向别人展示以加深理解，这是一种辅助原型...目的不同，原型也有多种。另一种分类方法是將原型分为抛弃型的和渐进型的，所谓抛弃，就是用完了就扔了，渐进型的，则是将来会在它基础上逐步完善，乃至形成最终系统。

我们在做需求时所需的原型主要是辅助性的，将用例场景转化成可操作的原型，如果是做 Web 系统，则基本上就是静态 html。第一，它能帮助系统分析员更好的与用户交流，同时让脑子糊涂的用户明白，哦，原来就是这样的啊.....第二，这个原型能帮助用户深化需求，凭空想象用户很难提出具体而清晰的需求，当面对一个可以操作的界面时，往往文思泉涌。第三，这个原型能帮助系统分析员验证需求分析的结果，如果将用例场景的文字描述转化成界面以后难以操作，那就得回头修改用例场景了。

那么需求的系统原型是抛弃型的还是渐进型的呢？不一定。有的组织有快速页面生成工具，能很快的将需求转化成界面，这些界面简单，不能满足开发要求，需求结束后往往被抛弃；有的组织为了在需求过程中将用户 Look and Feel 的需求也一并收集，会精心开发界面原型，这些界面就成为将来的开发基础。的确大部分组织是将 html 开发完成后，由程序员填入动态代码而形成 ASP, JSP 等动态网页的。

系统原型什么时候需要呢？虽然本系列文章到最后才来讨论它，但笔者的建议是从一开始就要开始原型的制作。很多人抱怨需求难做，用户讲不清又说不明，今天说的跟明天不一样，抱怨用户根本不懂计算机。有此抱怨是正常的，需求从来就不是容易的，但如果以这为借口而做不出好的需求，那就只能说明你不是一个合格的系统分析员了。用户如果都懂计算机，还要你做什么？还好意思拿着"高薪"，号称"高新技术人才"么？把用户想说又说不出，或者根本不想说的东西套出来，这就是系统分析员的职责。原型在这里将起到巨大的帮助作用，一个图形化的展示胜过千言万语，UML 的诞生也是这个原因。在需求的初始阶段，界面原型或许还开发不出来，但是，用 Word, Visio, Powpoint 画几个简单的图示不

困难吧？甚至在草稿纸上手工画画界面原型，也会对你跟用户的交流起到巨大的作用。

8.2 需求规格说明书

我们的所有准备工作都完成了，即将交出工作成果--需求规格说明书。有的读者会奇怪，之前我们做的工作不都是需求说明吗？怎么又来一个需求规格说明书？原因是这样，我们之前的工作的确都是需求说明，但是，这些需求说明是零散的，组织不好的。就拿笔者给大家提供的实例来说，读者在看的时候感觉如何？没有章节，没有提纲，也看不出作者的组织思路，要看明白一个需求，要点好几个图，展开好几层。对系统分析员来说不是什么问题，但对用户而言呢？你能指望他们满意这样的文档而让你验收通过吗？另一个原因是，现在系统建设一般都会按照国标来要求文档提供，例如 GB9385-88，尤其请了监理的用户更是如此。因此，写一份符合国标格式要求的需求文档是非常有必要的。

不必担心需求规格说明书会给你带来多大的工作量，其实所有的元素已经具备，需要做的工作不过是将这些元素组织到一起而已。笔者提供一个简单的例子以说明如何将他们组织起来。但这个例子并不是说明这是一个标准格式，你应当根据组织规范，用户要求来组织这些元素。笔者想说明的只是一个组织文档的思路和哪些元素是必须的以供参考。见附录。

最后需要说明的一点是，在这个例子里，分了用户需求和系统需求两个部分，这对应着业务用例和用例实现。用户需求不一定是系统需求，某些用户需求是不必实现到系统中的，例如本系列文章示例中的图递送过程，缺了它用户需求就不完整，但实际上这是一个人工过程，不需计算机的参与；同时用户需求也未必全部包含系统需求，例如用户需求中未提及事务处理，操作记录，但作为一个健壮的系统，这些需求又是必不可少的。

后记...

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

前后经过大半年，关于系统分析员用例分析的文章到这里就结束了。期间承蒙网友们的支持与鼓励，才走到今天。系统分析和 UML 是一个庞大的话题，短短的八篇文章仅能够揭起冰山一角。实践比理论学习能更快的成长。就笔者自己而言，若要论及理论知识，未必及得上科班出身的系统分析员们。只是实践多了，就有些经验积累下来，以至能与诸位分享。笔者相信，理论--实践--理论，永远是一条不二的成长途径。只要本系列文章对大家还有些帮助，就不枉这半年多的笔耕了。

笔者不寄望于能在这短短八篇文章里将所有知识和经验都讲到，也不保证有需要的读者一定能在这里找到答案。但笔者的 Blog 还在，也还没有从此收山的打算，只是这一系列文章到了该结束的时候了。若读者有疑问，有指教，都可以在我的 Blog 里留言，以武会友就是专门为大家准备的。笔者保证每条留言都会回复的。谢谢大家。

小小预告一下，用例分析系列结束了，接下来笔者将开始系统分析和设计的系列文章，就是通常所说的 OOD 过程，这是将需求转化为代码的中间重要阶段，所面对的读者是 OO 系统设计师。希望继续得到大家的支持与鼓励。敬请期待。

9 问题和回复

网友 ATC 提出的关于迭代的问题

ATC 网友的问题：

请教一下在实际的项目中是如何真正做到迭代的？

例如，任何的需求变更，都从重新分析业务模型开始？

谢谢

解答：

首先我说明一点，并不是任何的需求变更都一定要从重新分析业务模型开始的。这是对迭代的误解，如果什么需求变更都要迭代，那么工作量根本是无法承受的，迭代也会变得无法控制，谁知道需求什么时候变更？迭代并不是因为需求变更而来的。

我们知道 RUP 倡导迭代的软件过程，RUP 定义了四个阶段和九个核心 workflow，也知道 RUP 是可以裁减的。先澄清一个观点，RUP 中每一个迭代都可能贯穿这四个阶段和九个核心 workflow，但不是一定就会。

要实现迭代的软件过程要做以下一些事：

首先，要定义软件生命周期，即根据项目实际情况和你所处组织的情况，从 RUP 中裁减出适合本组织和本项目的软件过程。简单说就是规划出本项目要产生哪些可交付物，而可交付物决定了你要做哪些过程来产生它们。然后根据 RUP 定义出这些可交付物产生的流程，例如业务建模过程--->概念建模过程--->分析过程.....

其次，要有里程碑计划。即将把上面定义出来的可交付物归纳出来，形成某个阶段我们应该完成哪些可交付物。例如里程碑一要完成业务用例模型、概念模型、分析模型.....；里程碑二要完成界面原型.....

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

上面的工作是制定迭代计划的基础。一个迭代计划是说，在整个软件开发阶段中，根据实际情况，我们需要用几次反复来完成。而每一次反复，我们要完成哪些里程碑里的哪些可交付物。例如第一次迭代，我们要完成里程碑一里的业务用例模型、概念模型和里程碑二里的界面原型；第二次迭代我们要完成全部的里程碑一和全部的里程碑二；第三次迭代我们要完成.....而每一次的反复，我们都要重新检查和更新上一次反复的可交付物。

为什么制定迭代计划一定要先定义生命周期和里程碑呢？这是因为生命周期计划规定了每一次迭代要遵循的标准过程，即怎么做；里程碑计划规定了每个阶段交付哪些产品，即每一次迭代要做什么。迭代，是事先计划好的，不一定因为需求变更而变更，除非这个变更通过变更委员会评审决定后，才有可能调整迭代计划，事实上，如果迭代计划要调整，基本上整个软件计划可能都需要变更了。

所谓的迭代过程，就是在每一次反复的时候，按照生命周期计划里规定的实施流程一步步的，把每一个产生的可交付物根据新的需求，变更的需求，精化的要求，补充的内容再次完成一遍。

很多人混淆了迭代与变更管理，从形式上看，它们的确比较类似。但他们的目标和范围是不同的，或许可以类比为战略和战术的关系。在一个成熟的组织里，迭代是计划性的，不同的项目有不同的迭代次数和计划，而变更管理是管理性的，所有项目都遵循同样的管理流程。迭代是解决软件生命周期问题的，变更管理则是解决质量控制问题的。

不知道我的表述是否清楚了。很感谢你提这个问题，给我提供了一个想法，某天我会就 RUP 中软件过程是怎么实现的写点东西的。

[coffeewoo](#) 评论于: 2006.08.24 21:50

☞审核能作为一个业务用例吗？

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

在业务流程中，审核是经常的必须的环节，不知道在业务建模阶段，是否应该作为一个用例？如果是，则又导致用例非常多！麻烦解答一下！谢谢！

cloud 评论于：2006.09.28 09:31

re: cloud

不能。正如本系列 1 中所说，用例必须是有动作有受体的，审核只是一个行为，人们通过这个行为作用到某个或某类实体上才能获得一个可期待的结果。没有内容，审核什么呢？

coffeewoo 评论于：2006.09.28 13:05

谢谢回答！

我的意思主要是审核某种申请或文件，例如是审核物品申请，如样品申请、办公用品申请的审核等，

是否就可以算是一个用例呢？显然，这是存在受体的！也存在可观测的结果，如已审核后的申请表等。

如果把审核申请这一类也算是用例，可能会导致用例粒度太细。如果不包括审核 XX 申请这一类东西，似乎需求描述不太明确。

另外，在商业模型中，通常一个角色，如部门经理或总经理，大多会存在大量的申请需要审核，例如营销部门经理通常会审核退换货申请、发货申请、人员招聘申请等，而部门经理通常不是具体的申请流程的起点！

cloud 评论于：2006.09.28 18:12

re: cloud

关于这种情况，用例的确是会很细。我一般的处理办法是用一个审核文件作为用例。审核具体的内容则来继承它。从观点上说，子用例不作为业务用例的一员，因此用例数量得以减少。

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

当然减少用例数量并不是唯一的目的，能够这么做的一个根本前提是审核的过程是一样的，不论审核的是什么内容。实际上，这时候您在提取用例的时候，重点应当放在“如何”审核上，而不是审核“什么”上。如果审核过程是一样的或很小的差异，那么哪怕有一百种文件，也只需要一个审核文件用例。在系统建模阶段，才应当把具体文件和抽象文件的继承或扩展关系用实体类的形式表达出来。

coffeewoo 评论于：2006.09.28 18:32

非常感谢~ 很有启发！

cloud 评论于：2006.09.28 20:55

☞用例粒度的问题

coffeewoo, 非常感谢您的 OO 系统分析员之路系列！

这是难得的一个系统分析入门的好东东！

对我的触动和启发很多！非常感谢！

这里有些问题想请教一下，是关于用例粒度的！

你的文章提到：

“在业务建模阶段，用例的粒度以每个用例能够说明一件完整的事情为宜。

即一个用例可以描述一项完整的业务流程。”

”在用例分析阶段，用例的的粒度以每个用例能描述一个完整的事件流为宜。可理解为一个用例描述一项完整业务中的一个步骤。”

“在系统建模阶段，用例视角是针对计算机的，因此用例的粒度以一个用例能够描述操作者与计算机的一次完整交互为宜。”

因此,如上所述，在业务建模阶段，似乎一个用例对应一个业务流程，如 申请费用 就对应一个费用申请的业务流程。

假设费用申请的流程或者过程是： 业务员提出费用申请，部门经理审核申请，

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

财务主管审批。

显然，按照业务建模阶段 用例的粒度说明，业务员对应的用例是申请费用。是一个完整的业务流程

而部门经理对应的审核费用申请，财务主管的审核费用申请似乎也应该算是用例。

但是审核费用同时也是费用申请流程的一个步骤。也即审核费用的粒度应该与用例分析阶段的使用例的粒度一致。

因此，我想问的是：

业务建模阶段，费用申请这个由业务员启动的用例可以描述一个完整的业务流程。

那么，类似部门经理的审核费用这样业务应该如何业务模型中描述呢？因为审核费用是费用申请流程中的一个步骤，

若审核费用作为一个用例，它的粒度与业务员的费用申请的粒度应该是不一致的。

cloud 评论于：2006.09.29 00:04

☞re: 系统分析，业务建模，UML，RUP 相关

业务用例模型是针对组织机构的工作来说的，审核在业务用例模型中已经被包含在了“费用申请”（请注意用例命名的一致性，动宾或是主动）中。那么业务用例模型阶段就没有必要将它作为一个单独用例了。

除非你在划分组织机构的时候特别划分了一个“审核管理”的组织机构。但是相对来说这个可能性很低，因为除非你的企业本身是专门针对“审核”有业务往来的，否则“审核管理”的组织机构是不存在的。

因此系统用例模型的阶段再将审核费用 extend 生成费用为宜。

rwyx 评论于：2006.09.29 10:57

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写, 原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成, 并发给作者本人, 再由作者本人向外发布的。本文允许自由传播, 仅供个人学习之用, 禁止用于商业行为, 如出版物, 培训教材等。读者在使用本文时请尊重作者之著作权, 勿篡改或删除或改编本文。若有非个人的任何公司, 组织和商业机构想采用本文之全部或一部分, 请与作者联系, 勿私自使用。若出版商有意出版此文, 请与作者联系。谢谢合作。

re: cloud

😊 非常感谢 rwyx 替我回答, 完全赞同之。rwyx, 欢迎你常来, 若你也有 blog, 可以交换一下链接否?

另外我说一下我原文中的理解。其实在写原文的时候我也犹豫过, 因为在用例中引入“业务流程”这样的词汇很容易引人误解, 用例是基于人员视角的, 单个人员要做的事情显然不能构成一个“业务流程”。对于你所说的费用申请这个用例, 请这样去理解, 费用申请是谁启动的? 最终为谁服务的? 显然它不是为主管服务 也不是为部门经理服务的, 它是为业务员服务的, 它的执行的目的是为了给业务员一个期望的结果, 主管也好经理也是实际是参与这个用例并辅助达到那个期望的结果。我认为其实这正是用例定义的正解! 谁是主角? 谁是配角? 是一定要分清的。其次, 正如 rwyx 所说, 申请费用是一项业务, 但审批费用不是一项业务, 有哪个企业专为设置一个审批部门, 并且在没有申请情况下光靠审批就能办完一件事儿的?

因此, 还是回到用例是什么的最基本定义上来。业务阶段, 只有象申请费用这种有目标, 有结果, 为某人所启动, 并为之服务的, 才能作为用例。分析阶段, 是一个用逻辑角度解释如何服务的过程, 所以可以把经理, 主管之类的配角拉进来, 解释他们各在这个服务中贡献了什么。到系统阶段, 就要把计算机引入进来, 解释这些人员如何通过计算机一步步的来贡献于这个服务了。这是对我原文中所述的一个补充理解。

coffeewoo 评论于: 2006.09.29 11:14

re: 系统分析, 业务建模, UML, RUP 相关 [回复]

呵呵, 我的 blog 为 <http://rwyx.bokee.com>, 不过我的 blog 只能加 blog 为 bokee.com 的为友情 blog, 因此我很想将您的 blog 作为我的友情 blog, 只是无法实现.

也欢迎您常来我的 blog 看看, 一起交流进步

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

rwyx 评论于: 2006.09.29 13:16

☞thanks to rwxy and coffeewoo [回复]

谢谢二位的耐心解答！

cloud 评论于: 2006.09.29 15:10

☞re: 系统分析，业务建模，UML，RUP 相关 [回复]

很抱歉，我将您的 BLOG 加到了我的友情收藏中，之前的那个回复无效 😊

我的 BLOG: <http://rwyx.bokee.com>

期待您的光临指导

rwyx 评论于: 2006.09.29 18:11

☞用例规约的事件流描述中可以引入其它的 actor 吗？

大赞

这两天一直在咖啡小驻学习，对我这么一个初学者是莫大的帮助。

接着 cloud 的问题继续提问：

最近在做考勤系统的分析，也是遇到了 cloud 类似的苦恼。

针对“申请出差”这个用例，我是如下这样描述事件流的

- 1、员工提出出差申请
- 2、部门领导审批申请
- 3、人事专员审批申请

请问这样描述合适吗？应该怎样描述比较好，谢谢！

LittleD 评论于: 2006.09.30 11:23

☞re: LittleD

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

想先规范一下叫法，事件流不太适合在需求阶段，需求阶段用场景，或业务过程更合适一些。事件流这个叫法一般用到分析和系统阶段。

我觉得按你所说的描述业务过程没有不妥啊，你提供的信息较少，我不太能了解你的迷惑在什么地方。本系列文章的第 7 篇提供了一个用例规约的例子，你可以下载看看该如何描述一个用例场景。如果还有问题，请更详细一点说明你的疑惑，好为你解答。

coffeewoo 评论于: 2006.09.30 11:58

re: coffeewoo

非常感谢 coffeewoo 及时地解答我的问题。

可能是我刚才没有描述清楚我的问题，我再详细描述一下。

我现在想做一个公司考勤系统，用于员工签到签退，各种假勤和加班等的申请与管理，人力资源部制定排班计划等。

本来已经画了一个 rose 图，不过现在手头没有，那我就用文字描述一下：

目前我抽象了几个 actor：员工、人事专员、部门领导、公司主管

各个 actor 相关的 usecase 如下：

- 1、员工：签到、签退、申请加班、申请调班、申请休假、申请外出、查询个人排班、查询个人考勤
- 2、人事专员：排班、管理加班、管理调班、管理休假、管理外出、查看考勤统计报表
- 3、部门领导：管理加班、管理调班、管理休假、管理外出、查看考勤统计报表
- 4、公司主管：查看考勤统计报表

由于考勤系统中很多业务都是一个流程，所以我写的用例规约就是描述该用例对应的业务流程的步骤，以“申请出差”为例：

用例规约：申请出差

1. 用例名称

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

申请出差

1.1 简要说明

此用例描述员工通过考勤系统是如何进行申请出差操作的。

2. 事件流

计划外出的员工登录系统并选择“申请出差”后开始该用例。

2.1 基本流

1、员工提出出差申请

员工填写计划出差时间、地点和缘由

2、部门领导审批申请

系统提示部门领导审批申请，部门领导审批申请

3、人事专员审批申请

系统提示人事专员审批申请，人事专员审批申请

2.2 备选流

.....

3. 前置条件

员工已登陆系统并选择“申请出差”操作。

我对用例规约的理解是描述 actor 为了实现 usecase 的目的，actor 与业务系统交互的过程。

那么，我有几个问题：

1、用例规约中可以出现其它 actor 吗？比如说“申请出差”中的部门领导和人事专员

2、当 actor 发起一个 usecase 时，会产生一些信息，由系统主动传递给其它的 actor，这个需要出现在用例规约中吗？比如说“申请出差”中的系统提示部门领导审批申请

3、当一个 usecase 的业务场景中，需要其它 actor 的行为，需要在用例规约中出现吗？比如说“申请出差”中的部门领导审批申请

coffeewoo，不知道我这样是不是说清楚了，谢谢

LittleD 评论于：2006.09.30 13:39

re: 上面那个问题

coffeewoo，我还有一个想法就是引入 workflow 系统，不过想法还不成熟
每次申请时，都是员工提出申请，传递申请信息给考勤系统，然后考勤系统将申请信息传递给 workflow 系统，workflow 系统将审批结果传递给考勤系统，考勤系统将审批结果传递给员工

不过感觉这样把部门领导审批给遗漏了。

请 coffeewoo 指点一二，谢谢！

LittleD 评论于：2006.09.30 13:46

re: 系统分析，业务建模，UML，RUP 相关

想问一下 LittleD 您的 UC 是在什么阶段？业务用例还是系统用例？
业务用例的话太细化了，比如：签到、签退、申请加班、申请调班、申请休假、申请外出、查询个人排班、查询个人考勤
这些整个其实可以作为一个用例，就叫做“管理考勤”。
如果做在系统用例是可以的，但是您的前置条件又太少。
而且还有个很大的问题，从您对基本流的描述来看，您不是针对一个用例在描述。您是在描述全部的工作流程。这是不正确的，每个用例都有其自身的场景，您所描述的不是用例，这是常见的甬道图的工作流程。

另外我先回答一下您的问题：

1. 用例规约中可以出现其它 actor，多个 actor 对一个用例的使用是没问题的
2. 这样的信息是可以出现的，不过请注意是在系统用例的阶段才可以出现
3. 这个问题是粒度的问题了

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

我先代替 coffeewoo 回答一下哦，回答得不对不要怪我，呵呵

rwyx 评论于：2006.09.30 14:43

re: LittleD

😄 这回明白你迷惑在什么地方了。你跟 Cloud 一样，有点混淆业务用例和系统用例。你可以再回头看看我给 cloud 的回复。在这个阶段，所谓的 actor 是指的主角，也就是发起者，其它的配合者是不需要给他们建立用例的。你的用例基本上没什么问题。用例描述也很正确，关于你的疑问，我的回答是肯定的，别的 actor 一定是可以出现在这个描述中的，用例规约中还有一栏是指 actor，涉及到多少个，就可以填写多少个。

不要困惑于为什么在一个用例里出现了别的 actor 和别的疑似用例，其实这正是你要的结果。你现在是在做业务建模，目的只是说清楚业务如何动作，但下一阶段你必须将业务逻辑化成为系统模型，凭的是什么呢？从何入手呢？回头看这段用例描述，你已经有了很明显的系统用例了，员工角色做什么？提出申请出差，部门领导做什么？审批申请... 对不？再接下来，你再对提出申请出差时员工是怎么和计算机交互的，要填什么东西，计算机如何约束，讲清楚，从业务到逻辑的过程就完成了。并且是可以回溯验证的，对吗？其实用例分析就是这么一个自顶向下的简单过程，排除面对复杂业务时要进行业务架构和软件架构建立之外，是可以照猫画虎的。

在这个阶段你还用不着考虑工作流系统，这个工作已经是在设计阶段去考虑的了，现在就是说清楚业务就 OK。至于工作流，是一个很宽泛的概念，简单从状态机，复杂到分布式引擎。对于你这个例子来说，如果是一个实际系统的话，我个人认为没有必要用工作流，属于大炮打蚊子。哪怕是写死在代码里，我觉得相对也比较适合这个系统的规模。要一点灵活性的话我建议最多用一个状态机就足够了。最好的技术是最适合项目的技术，并且，越简单越好。

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

coffeewoo 评论于: 2006.09.30 14:45

re: rwyx

哈哈，总是被 rwyx 抢先😁

关于粒度的问题，就这个例子的规模来说，我倒觉得差不太多，因为我自己觉得用例的粒度在面对不同规模的系统来说有点调整还是有必要的。

比如说，我们可以用管理加班作为一个业务用例，而把申请加班，批准加班，查询加班等作为它的 include 或 extend，这是正解，这样就只会 有一个业务用例。但是就这个例子的规模来说，我们直接把申请加班，批准加班等作为用例也无不可，因为这样我们可以省掉后面概念模型的建立阶段而直接进入系统 用例。

还有，我怀疑 Li ttID 的管理加班用例有可能并非是审批一下这个简单，如果是这样简单，它的确只是管理加班的一个步骤，没必要单独作为用例。如果 管理加班并非审批，还包括浏览，查询，统计，分析，甚至与工资系统之类的话，它成为一个单独的用例也有其道理，关键还是看 actor 是如何看待这个用例 的。

最后提醒一下，rwyx 的说法更加符合 UML 的原始定义，推荐这么做。我自己是习惯了，一面对规模较小的系统，就会直接升级概念阶段用例到业务阶段来，这个做法对初学者可能不太好。

coffeewoo 评论于: 2006.09.30 15:00

re: rwyx&coffeewoo

非常感谢 rwyx 和 coffeewoo 的回答。

发现自己是有点把业务用例和系统用例搞混了，可能是没有经验的原因吧😁，希望以后不要搞混了。

to coffeewoo，我定义的管理加班的确只有审批，浏览、查询、统计等都归到查看考勤统计报表

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

之所以这样，我可能是有点仿 coffeewoo 给的那个图书馆的例子

在那个例子中，借阅管理员的所有用例（除了查看借阅记录）都应该是借阅人的操作驱动的，而不是借阅管理员自己发起的

这样我就有一个疑问：

在业务阶段，像“管理加班”和“颁发借阅证”这种不是由 actor 主动发起的事件，是不是应该作为用例出现？

如果不作为用例出现，那这种事件如何体现在业务建模阶段？

谢谢

LittleD 评论于：2006.10.08 10:10

re: Little

呵呵，总算没让那家伙抢先😁

恩，这么说吧，我给的那个例子严格来说是不正确的。因为本身例子的规模很小，而我自己的习惯是当项目规模太小时我就会把用例本身的粒度变小，实际上在那个例子中我已经在业务用例阶段用上了系统用例了，你看我的例子里实际上是没有画系统用例模型的。好象有点误导大家的样子😞

按正规的做法的话，借阅管理员那些的确不是业务用例，它们只是借阅人用例的扩展或包含。在系统用例阶段才应该被体现出来的。

coffeewoo 评论于：2006.10.08 10:36

re: coffeewoo

感谢！

问一个比较基本比较教条的问题：

业务、概念和系统阶段分别应该产出哪些内容和图？

LittleD 评论于：2006.10.08 10:54

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

re: LittleD

这个问题按教条的话我也说不全，去查 RUP 的教材里讲得是最全的，其文档之多挺吓人的。我说的有点脱离教条了，是按自己的经验去讲的。所以建议还是去看看 RUP 的原始教材，我写的时候是没有参考的，因此会和教材不太一样。RUP 里，图是辅助的，取决于需求情况，需要什么图就用什么图。总之呢，业务就是明确需求范围，概念就是用例分析，形成关键业务的框架和解决方案，系统就是决定开发范围，形成软件架构的用例视图。至于要用的什么图正如上说，是不一定的，取决于要解决什么问题。

coffeewoo 评论于: 2006.10.08 11:11

re: 系统分析，业务建模，UML，RUP 相关 [回复]

今天真是失策呀，没有比 coffeewoo 快，不过这里是他的领地，也得给他点面子是吧^_^

关于“业务、概念和系统阶段分别应该产出哪些内容和图？”这个问题确实很难回答，因为每个公司对其需求规格说明书的要求是不一样的。

通常来说，业务用例虽然常见的是用例图，但是对每个用例的说明却有很多很多，前置条件、后置条件、基本流、可选流等等，而且在业务用例前做得规范一点还要做组织机构图、角色说明等等工作。再做得具体一点，可以产生一份专门和客户以交流的甬道图，当然这些并没有规定你是必须去做的工作。

系统用例则相对更细节，它的画法相信 LittleD 已经明白了，不过，我猜测 LittleD 可能对每个系统用例的描述有困惑。其实在系统用例来说可以有两种描述，一种是利用用例实现图，将实体和实体间的交互在用例实现图中表述出来，然后配合文档的说明，另一种是纯文档形式，纯文档形式就和业务用例的说明差不多了，基本要包括“前置条件、后置条件、基本流、可选流等”这些内容，此外，纯文档形式还必须抽取业务对象（一般这个算是分析模型的一种）。

就如同 coffeewoo 所说的“业务就是明确需求范围，概念就是用例分析，形成关

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

键业务的框架和解决方案，系统就是决定开发范围”，抓住这个主要概念就可以了。

评论比较短可能说得不够清楚，有什么问题还请包涵

rwyx 评论于: 2006.10.08 17:07

re: rwyx&coffeewoo

非常感谢二位的回答。

我又有几个问题：

- 1、如果在业务用例阶段，有一个 actor A1 和相关联的 usecase UC1, 如果在系统用例阶段，UC1 可能会扩展或包含一个 usecase UC2, 请问 UC2 是必须与 actor A 相关联还是可以与其它的 actor B 相关联？
- 2、其实这个问题也与第一个问题有关：用例规约是在什么阶段写，是在业务阶段？系统阶段？还是都有可能？
- 3、是不是所有的用例都应该写用例规约？比如说第一个问题中的 UC1 包含 UC2, 那么这两个用例是不是都需要写用例规约？
- 4、用例规约与时序图有啥关联吗？

呵呵，又提了一些基本而且教条的问题🤔

LittleD 评论于: 2006.10.08 17:38

re: LittleD

1, 用例永远是从 actor 的角度去看的，谁主动用它，谁从中获得一个期望的结果，谁与它关联，其它 actor 都与它没关系。例如你借书，管理员要检查你以前 有没有没还的书，检查是借书用例的一个 include，但是检查是管理员需要的，对你来说检查不检查不是目的，借书才是目的，所以你只和借书关联，管理员 只和检查关联。

2, 都有可能，如果你有充分的项目时间很高的要项目要求，当然要写。

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写, 原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成, 并发给作者本人, 再由作者本人向外发布的。本文允许自由传播, 仅供个人学习之用, 禁止用于商业行为, 如出版物, 培训教材等。读者在使用本文时请尊重作者之著作权, 勿篡改或删除或改编本文。若有非个人的任何公司, 组织和商业机构想采用本文之全部或一部分, 请与作者联系, 勿私自使用。若出版商有意出版此文, 请与作者联系。谢谢合作。

3, 是的, 每个用例都要写。但是业务阶段和系统阶段用例说明是不一样的, 业务阶段说的是业务流, 系统阶段说的是事件流。业务阶段的业务流基本上是 actor 之间如何交互而完成一个业务用例目标, 事件流则基本上是单个 actor 如何与计算机交互而完成系统用例的目标。而每个系统用例的完成最终达成了 业务用例的完成。

4, 没啥必然关系。用例规约要说明流程性的过程, 这个过程用文字写可以, 画成图也可以, 两者都提供更好。

coffeewoo 评论于: 2006. 10. 08 18: 03

re: 系统分析, 业务建模, UML, RUP 相关 [回复]

我 只想回答第一个问题, 可以的, 两个 actor 同时对一个用例起作用是没问题的, 关键在于如果你这么画了, 那么在描述时就有必要在两个 actor 工作的这个 用例场景中分别详细描述, 因为归根结底, 它属于两个不同场景的工作. 用两个用例或许更清晰, 不过你画成了一个, 这说明你抽象了, 但是在描述时还要描述清晰的.

而且在系统用例阶段, 并不是人才是 actor, 连一个子系统也可能是 actor, 所以, 你既然已经得到了一个子系统, 那么一个用例也可能指向另一个子系统.

正好说到了子系统的概念, 我想问一下 coffeewoo, 除了使用常见的 UC 矩阵 (use/creat) 方法来划分外, 还有其他比较好的方法吗, 或者你一般是怎么分的?(拍脑瓜, 凭经验的方法除外)^_^

rwyx 评论于: 2006. 10. 09 15: 27

re: coffeewoo [回复]

谢谢, 明白了许多。

以后可能还有很多问题向 coffeewoo 和 rwyx 请教

有一个问题就是

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

在 coffeewoo 上面的回答中，提到检查和借书两个用例，我怎么感觉检查这个用例也不是主动用它，而是由读者的借书用例驱动

还有一个小问题：

正如 coffeewoo 提到业务阶段用例规约是描述业务流，基本上是 actor 之间如何交互完成一个业务用例目标，如果一个 UC1 只与一个 actor A 相关联，即只有 A 主动用 UC1，那么在 UC1 的用例规约中，actor 栏是不是也可以出现其它的 actor B，C 之类的，因为完成 UC1 的业务目标需要 actor A 与 B，C 进行交互。

LittleD 评论于：2006.10.09 15:30

re: LittleD

你的疑问实际上是因为把借书和检查作为同一个粒度的用例去想，所以出了问题。我说的是，检查是借书的一个 include，也就是说检查本身不是一个业务用例，只是为了表达清楚，而画出来示意的。对于我那个例子，实际上粒度已经细到系统用例级别了，好象误导了不少人：（实在失误，虽然我自己很清楚...

re: rwyx

希望你说的子系统不是指的 UI 展现出来的那些功能的组合。很多人认为那就是子系统或功能模块，不过我不那么认为，我认为那是只是 UI 的设计而已。把哪些功能放在一起无非是方便使用，符合习惯罢了。UI 设计无教条可询，用户的意见最重要。

我所理解的子系统实际上是由系统内部各组件的依赖关系形成的。划分子系统依据于分析模型的结果。我的做法是先把分析模型做出来，然后尝试将分析模型中的对象放入不同的包（里的确有经验的成份，并不是一个个瞎试的，最初的依据还是来自业务用例，把一个业务用例当成一个包，在此基础上再改进）。这时会发现一些有趣的内容，比如，某个 control 类有好几个包都需要，比如，某个包中的某个 Entity 类被多达七八个其它包所引用，比如，某个 bandage 类要与分散在七八个包里的 control 类打交道...为了解决这些问题，尝试将分析类移到别的包，合并一些包，分拣出公有元素形成新包，也就是所谓的 LIB 等等。

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写, 原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成, 并发给作者本人, 再由作者本人向外发布的。本文允许自由传播, 仅供个人学习之用, 禁止用于商业行为, 如出版物, 培训教材等。读者在使用本文时请尊重作者之著作权, 勿篡改或删除或改编本文。若有非个人的任何公司, 组织和商业机构想采用本文之全部或一部分, 请与作者联系, 勿私自使用。若出版商有意出版此文, 请与作者联系。谢谢合作。

最理想的情况, 那些分析类的所有依赖都在局限一个包里, 或只与 LIB 有关, 或仅通过一个 bandage 与其它包交互.... 到这时就形成了子系统的雏形了, 剩下的工作, 就是参考 UI 的要求, 决定将哪些包合成一个更高层次的包, 这个包包含了 UI 要求, 由于基础来源于业务用例, 基本上也会符合业务习惯要求, 这个包就是子系统, 取个合适的名字就 OK 了。

总结一下, 大部分子系统划分是自顶向下的方法, 我用的是自底向上的方法。如果构成底部组件级别的包已经耦合度很低了, 再用它们来组合子系统就自由得多, 尽量参考 UI 要求就是了。

coffeewoo 评论于: 2006. 10. 09 16: 19

re: coffeewoo

不得不说的是, 你的这个做法是有效的, 不过并没有解决我的提问。

为什么呢?呵呵

因为你这个做法其实就 UC 矩阵的做发呀, 只不过 UC 矩阵用的是矩阵中 Create 和 use 来归纳工作和实体, 然后手动将 U/C 两类数据进行不断移行, 形成 U/C 交集点得到各个子系统。

而你的做法, 虽然没有这么做, 但你是将控制类和边界类与实体类进行分析合并, 这个就等于是默默得在将 use 动作和 create 动作在不断的移行. 到达 user 和 create 的交集. (coffeewoo 式做法?)

rwyx 评论于: 2006. 10. 09 16: 46

re: rwyx [回复]

我理解与 UC 是两回事的, UC 其实是不 OO 的, 首先在 UC 中的实体和工作是怎么出来的? 如果没有这两者, 矩阵就没办法排。从用例中如何推导 UC 呢? 其次, UC 关注点是 Create, use, 而我的方法关注点在依赖, 根本依赖关系除了 create, use, 更多的是 OO 中的耦合关系, 引用, 继承, 聚合, 这些关系都不是 create 和 use 关系, 也就是它们是静态的, 天然的关系。在 UC 矩阵中, 是哪怕

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

子模块或功能点被排得非常完美，UC 矩阵却不能告诉我们，被分在不同子系统的那两个模块的藕合度如何。因为在一个 OO 的世界里，create 等交互关系并不能代表对象的静态关系。

coffeewoo 评论于：2006.10.09 17:59

re: 系统分析，业务建模，UML，RUP 相关 [回复]

受教，也许是我错了，不过我还是觉得使用 UC 矩阵是划分子系统的好方案，这点不管在 OO 还是非 OO 的项目中，而且，我觉得划分子系统最好是在越前面越好，甚至可以指导 UC 的包的划分。因为这么做的话在系统用例的时候就可以利用子系统作为角色来引导整个系统用例了。

另外 control 类和 bandage 类通常是在描述用例实现时才划分的，这个阶段属于的是逻辑模型中的分析模型阶段，和系统用例视图是有区别的。

系统用例关注的是系统做些什么，而逻辑模型则关注怎样做。

但是子系统的出现却是在系统用例阶段就有了。那么系统用例阶段的子系统该如何出现呢？

rwyx 评论于：2006.10.10 12:18

re: rwyx

恩，你说的这点我很赞同，的确子系统越早划分越对系统建设有指导意义。我想我和你的分歧是对子系统的定义不同。我的观点比较怪，我不认为子系统应该是功能性 的，这是 UC 关注的点，我认为是内在逻辑性的，所以只有在内部逻辑得以明确，也就是分析模型出来之后，才可能决定子系统。但之前用例的确要分包，甚至这时候组件模型和部署模型都可能可以明确了。那依据什么呢？依据业务模块，这是一个自然的，符合用户习惯的，组织架构的，业务形态的，部署环境的一个自然划分。我觉得这样就足够了，因为业务用例阶段所有文档都是与用户有关的，没必要在这时引入 UC 来搞得用户迷惑，分出一些他们不习惯

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

的层次结构来。用他们最自然的划分方式来分包，分组是最好的。用户认为的这种子系统实际就是对功能模块的某种组织方式，我在这方面不太下功夫，用户认为合适就行，UI 是他们说了算的。而我认为的子系统是内部逻辑组织的优化结果，是系统架构的一部分。而这部分是没必要对用户说明的。

在 OO 的前提下，为什么我不认为子系统是功能性的呢？就是说不赞同从功能的角度来划分子系统，其实也就是我反对用 UC 图。原因在于，如果我们要基于 OO 的分析方法，而又用了从功能角度来划分子系统，还以它来指导，会出问题的。比如某个功能 F1 从 UC 的观点来说应该归属于 S1 子系统，但是，从领域模型来看，它的依赖却大部分在 S2 子系统，这种矛盾我想无法避免。因为 UC 是非 OO 的，而最终实现却是 OO 的。其实上 IBM 当初提出 UC 方法的 70 年代，正是过程化设计，瀑布模型盛行的年代，我觉得对现在来说已经不大适用了。就从业务用例来说，怎么可能把一个业务用例逐层分解呢？从 UML 观点来看，一个业务用例已经是个原子结构了。

那我的方法又如何避免这个矛盾呢？其实我的想法来源于 MVC 模式和 Facade 模式，将用户理解的子系统，这种过程化的，非 OO 的，功能的罗列，视为 V，将用我的方法划分出的内部子系统当做 M，而引入一个 Facade 模式，一个 BusinessLogic 层，作为 C。以达到的目的是将内部逻辑的耦合度降到最小，而又满足非 OO 的，功能性的展现要求。

coffeewoo 评论于：2006.10.10 14:18

再补充一点

在 UC 提出的时候，分布式计算，应用系统整合的需求是非常少的，而现在呢？在一个分布式计算的环境下，UC 该如何做？比如一个企业，从网站接收定单，转交生产计划部门，财务部门，市场部门，给到生产线，转到质量部门，维护部门，物流中心....假设这些部门的应用都分布在异构的环境下，UC 能够提供这些信息吗？如果把分布的两个模块划到了一个子系统中，又该如何去实

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

现呢？

UC 的目标是分析功能和数据的关系。如果按照 UC 的观点，也就是过程化的观点，我们一定会得出一个网上定单生产子系统，把上面的部门都包括进来，因为所有部门都在 use 这个定单，可能还会 create 一部分定单中的信息，它们都是以定单数据为中心的。实际上可能吗？过程化的方法划分出的子系统含义是，这个网上定单生产子系统是可以独立于计划部门的其它子系统，财务部门的其它子系统... 等等运行的，然而 OO 方法划分出的子系统的含义是说，计划部门的系统可以独立于网上定单系统运行...，它们的某个交互场景能够实现网上定单生产这一业务场景。它们有着质的不同。在我看来，用 UC 方法划分出的这个网上定单生产子系统是用户理解的，过程化的，基于 UI 的，实际上等价于某个业务场景，而不是我所理解的那些以 actor 为中心的，业务用例为基础的，考虑了内部依赖关系的，考虑了部署模型和组件模型的，可能被称为定单中心，财务中心，计划中心等的那些子系统。

我的观点已经表达清楚了吧？

coffeewoo 评论于：2006.10.10 14:54

re: 系统分析，业务建模，UML，RUP 相关 [回复]

你的观点很清楚，也很明确，看起来我们最大的分歧点的确是在于子系统的定义，你的子系统的定义是为设计服务的，而我的子系统的定义则是为系统用例服务做指导的。

另外，你用 MVC 模式和门面模式来阐述你的观点，还真是第一次

见到有人这么说啊^_^

以武会友的话，最后应该说一句“佩服”

rwyx 评论于：2006.10.10 15:27

客气了

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

我满脑子都是架构，节点，组件，部署这些东西，划分子系统已经 N 久没考虑过了，子系统这个词儿好象我只有在做 UI 设计和项目计划里会用到。看来我受 UML 和 OO 毒害之深已经无可救药了....

coffeewoo 评论于: 2006.10.10 16:46

☞能推荐关于需求分析、商业业务建模方面的经典教材吗？

谢谢！

uhoo2005 评论于: 2006.10.12 18:36

☞re: uhoo2005

🙏很抱歉啦，我自己是没怎么系统的看过书的，我看的東西就是 RUP 自己的附带的文档。写这些东西的知识都是来自自己的实践，其实有些东西和标准的 UML 有所差别，已经有人批评我误人子弟了:) 我记得有一本讲需求工程的书还是不错的，我大概浏览过一下，书名记得不是很清了，你找找看。

coffeewoo 评论于: 2006.10.12 22:31

☞请教个问题

我是个学生，😊这几天一直在 coffeewoo 看这些文章，深深体会到您理解的深刻，这几天收获很多，对我以后的继续学习打下了很好的基础。谢谢您把经验与我们分享。在学习中，我遇到了问题。想问问您。问题：先创建了 business usecase，在业务建模之后，是否还要创建 use case，我觉得两者好像没什么差别。是不是可以直接拷贝过来，然后改变一下类型就可

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

以了？

怎么从业务用例过渡到系统用例？

小学生 评论于：2006.10.13 01:34

re: 小学生

business usecase 与 use case 差别是相当大的，概念上完全不同。BU 是业务视角，从业务主角 actor 的角度来看，这个系统能给他提供什么价值，通过他做一件事（业务用例）来表达，人->做事->获得价值。BU 是完全业务语言的，不加入计算机的理解和设计。

uc 是系统视角的，从操作员的角度看，他在系统中做什么事（系统用例）来完成他的职责和获得他要的结果。系统用例来自对业务用例的分析和抽象。换句话说 uc 要完成 bu 描绘出的那些业务要求。

另一方面，bu 的结果是确定业务范围，而 uc 的结果是确定开发范围。你可以看看本帖中 cloud 和 littlD 的问题，他们对 bu 和 uc 也有疑问，上面有解答。

coffeewoo 评论于：2006.10.13 10:09

还有些不理解

谢谢您的回答。

那我能不能做这样的理解，您给的那个例子中，业务过程视图部分是属于业务视角方面的，而用例实现是属于系统用例方面的，因为用例实现是有加入计算机进行理解与设计了；

如果是这样的话，是不是要把用例实现的部分放到系统用例模型的文件夹中来。我有点搞蒙了。

小学生 评论于：2006.10.14 09:59

re: 小学生

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

我例子中的用例实现是属于系统用例方面的。

但是实际上业务用例也会有业务用例实现，不过我例子里没做，很多时候也不是必要的，你可以暂时不理睬业务用例实现，先认为用例实现就是属于系统视角的。关于文件夹你先不要迷惑，找个机会我要重新整理一次那个例子，现在还有些毛病。你的理解是正确的。

coffeewoo 评论于: 2006.10.14 21:42

☺..... [回复]

恩, 谢谢! 😊

我把 Coffeewoo 推荐给了同学朋友，他们现在都在学习。

我们也一直在进行着讨论，期待您的文章能再写下去，给我们以指导。有您作为明灯，我们不会觉得前进困难。

小学生 评论于: 2006.10.14 23:11

☺re: 系统分析，业务建模，UML，RUP 相关 [回复]

关于用例实现，我想说明一下，用例实现通常是被作为分析视图来给出的，因为它想告诉的是系统究竟怎样完成这个用例，而用例实现的每一个用例都是系统用例。

所以具体做的时候就应该在逻辑视图下建立一个分析视图，然后把系统用例的每一个用例拉进来做用例实现。

还要说明的是用例实现和用例描述是两个概念，用例实现更倾向于用例中对象的合作（这个对象的概念是业务对象），而用例描述则专注于对用例本身的描述。前者已经说过了是属于分析模型，后者则属于系统用例部分的工作

rwyx 评论于: 2006.10.15 00:00

☺求助

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

现在学习 UML 有点晕了

请问 coffeewoo 和 rwyx:

同一个用例可能有多个 actor 发起吗?

比如说加班业务:

员工申请加班，直接主管审批，然后部门助理可以设置加班，人事专员也可以设置加班

这个时候是不是可以把部门助理和人事专员合并?

但是部门助理可以查看部门考勤情况，而人事专员可以设置考勤参数、查看公司考勤情况等（部门助理不可以）

请问这种情况下，可以合并两个 actor 吗?

这两天和同事讨论了很久都没有结果

duxi angyun 评论于: 2006.10.24 15:43

re: duxi angyun [回复]

同一个用例完全可以由多个 actor 发起。

你的问题同样是因为业务用例和系统用例混淆导致的。加班业务是由员工发起的，并不是由主管发起。部门助理也可视为发起加班业务的 actor。

对于业务来说，员工，主管，助理等等共同奉献于这一业务，他们具体做什么在业务用例的角度上不需要考虑。

至于他们做的事情有不同，这是在系统用例的层次上考虑的。系统用例的 actor 并非用户的岗位概念，而是系统角色的概念。人事专员和部门助理是两个岗位，你所谓的合并，实际上只是他们都是同一个系统角色而已。而业务用例层次上的 actor 最后会被映射到系统用例层次上的角色，也就是说，你合并的是角色，而不是部门助理和人事专员。

coffeewoo 评论于: 2006.10.24 16:06

re: 系统分析，业务建模，UML，RUP 相关 [回复]

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

多谢 coffeewoo 的回答

为什么部门助理也可视为发起加班业务的 actor，那人事助理是不是也可以？

应该说只有员工申请加班，其他的 actor 才会共同奉献于加班业务

duxi angyun 评论于：2006.10.24 16:42

re: duxi angyun

当然人事助理也可以。是否是用例的发起人要看这个人是否会主动去启动这个用例并且从中得到他要的一个完整的业务结果。员工，助理都会主动启动，并从中得到他们要的完整结果，员工得到加班批准，助理得到加班参数管理的结果（注意，这件事是可以独立于其它事件存在的，没有申请没有审批也可以设置加班参数，这是与审批的差别）。

比如说查看加班记录，actor 要通过这个用例获得加班情况信息，所以他会主动启动用例，并且从用例中得到查询的结果，他的完整愿望实现了。即使没有人申请，没有人审批，他一样得到了自己想要的结果，无非是无记录而已。它是相对独立的。

如果这件事是相对独立的，从业务用例的角度说，你也可以把它视为一个单独的业务用例，只是要注意与其它业务用例的粒度相匹配就行了。从你的例子来说，把设置加班参数和查询加班记录看作独立的业务用例是可以的。如果这样分开，也就不存在你第一个问题中的迷惑。根据我说的这些，你可以思考一下为什么主管审批不能作为单独的业务用例而我却说设置参数和查询记录可以。请参考我的文章第一篇中用例的四大特征。

coffeewoo 评论于：2006.10.24 18:15

re: 系统分析，业务建模，UML，RUP 相关 [回复]

呵呵，我觉得吧，coffeewoo 还是把系统用例和业务用例分出来吧，不然大家都会问这个问题的哦。

rwyx 评论于：2006.10.25 00:17

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

另外，回复 duxi angyun 一下，当你觉得几个 actor 多用例都有动作的话，那我建议你不要考虑对多个 actor 都明确分为“员工，主管，助理”，而是可以抽象一个 actor，因为这个在设计人员设计时他会考虑某个 actor 有某些权限，而不会以职位来定死这个 actor 做什么。

也就是说权限、职能、角色三者的关系：

一个角色多个权限

一个角色一种职能（纯业务角度上讲，理论上这个需求只有在客户明确要求的情况下才应该实现）

一种职能多个权限（这个是考虑默认职能和权限的关系，但最终还是角色和权限的关系）

rwyx 评论于：2006.10.25 00:24

re: rwyx

是的哦😓 抽空吧，好多得重写一次了，累。

coffeewoo 评论于：2006.10.25 10:21

求助...

coffeewoo，感谢您的文章和您的例子。

不在偶在学习您的例子中，有一个疑问：如何一个用例可以有多个用户发起，那么在泳道图中如何表示啊？

是不是要抽象出一个用户来啊？

zqc 评论于：2006.11.01 16:29

re: zqc

这个问题要分两种情况考虑

1，用例的确是由多个用户发起的，但是这些用户的目的各不相同，每一个目的实际就是一个不同的用例场景。例如我们有一个维护论坛注册 ID 的用例。对于网友 这个 actor，他启动这个用例的目的是注册一个新 ID，对于论坛总管理员，他启动这个用例的目的是给某个 ID 授予管理某个版块的权利，对于版块管理员，他启动这个用例的目的是禁止或删除某个违反了规定的 ID。看，虽然他们都会主动启动这个用例，但是却是三个不同的用例场景，换句话说，你需要画三个泳道图。在用例图里也可以表现这个区别，例如，维护论坛注册 ID 是一个业务用例，它有三个扩展用例，分别是注册 ID，授权版主和管理 ID。就这个例子来说，假如用户的需求导致注册 ID，授权版主和管理 ID 这三个子用例的差异很大，你也可以直接将这些子用例升级成业务用例而不再需要维护论坛注册 ID 这个用例，这时就解构成了一个 actor 启动一个用例。当然，使用维护论坛注册 ID 这个用例也会有它的道理，其潜在含义便是，不论是网友，总管理员还是版主，他们使用这个用例时都会遵循同样或相似的过程，操作同一个业务实体。到底是分开还是合并，取决于具体用户需求。

2，用例的确是由多个用户发起，并且这些用户的目的是一样的。如果出现这种情况，那一定是你分析涉众的时候抽象不够。还是上面的例子，我们假设总管理员也可以做版主的管理 ID 的工作，那么总管理员和版主就都会主动启动维护论坛注册 ID 用例，并且有时，总管理员的目的和版主是一样的。在这种情况下，版主能做的事被包含于总管理员，在 UML 里，这种概念很常见，叫什么？泛化，对吗？因此，可以抽象出一个管理员的抽象 actor，于总管理员和版主都是管理员的泛化。或者，可以把总管理员作为版主的泛化，也可以把版主作为总管理员的泛化，都不错，选择不同的方案会影响到实现时的难易和复杂程度，但肯定都能实现。我的建议是，把最最经常出现的，使用量最大的，最具有普遍意义的那一个作为抽象，而把稍为特殊的那一个作为泛化。就这个例子来说，到底是总管理员普遍还是版主普遍？显然禁止或删除违反规定的工作要比授权一个版主经

本文是网络 ID 为 coffeewoo 的作者原创编写, 原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成, 并发给作者本人, 再由作者本人向外发布的。本文允许自由传播, 仅供个人学习之用, 禁止用于商业行为, 如出版物, 培训教材等。读者在使用本文时请尊重作者之著作权, 勿篡改或删除或改编本文。若有非个人的任何公司, 组织和商业机构想采用本文之全部或一部分, 请与作者联系, 勿私自使用。若出版商有意出版此文, 请与作者联系。谢谢合作。

常得多, 因此, 把版主作为基础抽象的 actor, 而把总管理员作为一个特殊的版主就是最合理的。假设 总管理员和版主的工作一样重要并且很难分出谁比谁普遍, 那么抽象一个新的管理员 actor 可能就是更好的方案了。在这种情况下, 泳道图只需要画那一个抽象 的 actor, 然后在文档里说明泛化后的 actor 的差别。当然如果有时间, 把每个 actor 都画一次(然有些步骤会是重复的)会更清楚。

coffeewoo 评论于: 2006.11.01 17:11

re: 系统分析, 业务建模, UML, RUP 相关

多谢 coffeewoo 的解答心中疑问.

您回答的好快呀! 感动中 ing...

偶刚开始学 uml, 刚学着使用 Rose, 再次感谢您的导航.

以后可能很多疑问请教您哦!

zqc 评论于: 2006.11.01 17:31

求助

在做分布式系统需求时, 使用 uml 业务建模, 是按服务器、客户端分开描述需求, 还是一起描述

zd 评论于: 2006.11.10 16:57

re: zd

UML 是通过多种视图来表达需求的, 用例视图表达功能性需求, 逻辑视图表达计算机针对业务需求的逻辑描述, 构件视图表达组件关系, 还有一个部署视图表达计算机逻辑构件的部署情况。

对你的问题, 我的意见是用例视图要一起描述, 不论是否分布式, 业务需求是整体的; 逻辑视图应当表达出软件层次结构来, 构件视图描述那些可以分离的, 独立部署的逻辑部件, 最后用部署视图表达分布式结构。

coffeewoo 评论于: 2006.11.10 22:26

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

re: 系统分析，业务建模，UML，RUP 相关 [回复]

不知怎么就进这儿来了，觉得非常不错，就把你的文章都下载打印出来，认真的看了几遍，非常谢谢。最近也在想用这种模式来分析设计，但对于用户涉众的关系这块儿有个问题想问问，图书管理员与借阅管理员和书架管理员是一个什么关系，包含还是派生，包含的话是不是说把图书管理员的业务划分成借阅管理员和书架管理员的业务，那是不是也可以不定义图书管理员这个涉众，如果是派生的话是不是在图书管理员那儿要定义些基本的业务，而借阅管理员和书架管理员在继承这些业务的基础上，还有自己的业务。

boskin 评论于: 2006.12.13 16:41

re: boskin

已经在 00 系统分析员之路--用例分析系列(5)--用户、业务用例和业务场景 中回复。请移步

coffeewoo 评论于: 2006.12.13 22:29

10 Appendix

10.1 一个房屋中介业务建模的实例分析

一位名叫 Mi dhael Yan 的朋友给我发来一封信，信中谈到这样一个问题。我觉得很有代表性，因此公开发布到 BLOG 上。这位朋友的问题是这样的：

一个租房中介准备提供一个网上中介服务系统，主要包括以下服务：

给求租者发布求租信息，寻找房屋信息

给出租者注册一个店面，在小店里发布出租房信息，也支持寻找求租信息

使用该服务必须注册一个用户

对房屋有收藏和评论的需要

我和几个朋友初步探讨了一下，在业务建模阶段出现了争执

我的分析：

一个求租者业务角色

一个出租者业务角色

发布求租信息业务用例

找房屋信息业务用例

注册店面业务用例

发布出租房屋信息业务用例

注册用户业务用例

朋友的分析：

一个求租者业务角色

一个出租者业务角色

发布信息业务用例（注册店面业务用例也被合并进来了）

查询信息业务用例

注册用户业务用例

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

另外一个朋友的分析更简单：

客人业务角色

发布信息业务用例

查询信息业务用例

请您给出您的见解，谢谢！

非常有幸拜读你的文章，收益甚多，谢谢！

有几点问题，希望指正！

1、关于你的网上借书范例

对于你把图书管理员这样的业务工人定义成了业务角色有点不解

2、我模拟了一个网上中介系统的范例，遇到了一些两难问题，请教

一个租房中介准备提供一个网上中介服务系统，主要包括以下服务：

给求租者发布求租信息，寻找房屋信息

给出租者注册一个店面，在小店里发布出租房信息，也支持寻找求租信息

使用该服务必须注册一个用户

对房屋有收藏和评论的需要

我和几个朋友初步探讨了一下，在业务建模阶段出现了争执

我的分析：

一个求租者业务角色

一个出租者业务角色

发布求租信息业务用例

找房屋信息业务用例

注册店面业务用例

发布出租房屋信息业务用例

注册用户业务用例

朋友的分析：

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

一个求租者业务角色

一个出租者业务角色

发布信息业务用例（注册店面业务用例也被合并进来了）

查询信息业务用例

注册用户业务用例

另外一个朋友的分析更简单：

客人业务角色

发布信息业务用例

查询信息业务用例

请您给出您的见解，谢谢！

合适的话也希望把这个范例单独在您的 BLOG 上发布出来，供大家一起探讨，谢谢！

这个讨论很有代表性，把它贴出来：)

我对第一个问题是这样看的，在我平时工作中有意忽略 business actor, actor, business worker, worker 这样的区别。因为我觉得，虽然在 UML 概念上它们是不同的，这样定义有其道理。但是这种概念的差异太过于学术化。在实际工作中，大家都熟悉岗位，角色这样的概念，甚至用户对岗位，角色这样的定义都有非常好的认识。但对于不熟悉 UML 的人来说，如果试图去向他们解释什么是 worker 什么是 actor，什么是 business actor... 我认为这是件费力不讨好的事情，我曾经试过，很难让人理解这么些小人图到底有什么差别。做一个业务模型的目的是让所有相关人等看得明白看得懂，而不是是否符合 UML 的规定。我用 UML 的一个观点是适合的采用，不适合的修改甚至放弃。我承认 UML 的定义是有道理的，但我不认为在实际工作中这样做会带来好处。在我们说明需求的时候，如果就是不区分 actor 和 worker，我们就会说不清需求了吗？我

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

相信不会，相反的，如果我们用岗位这个概念来做业务模型，用角色这个概念来做系统模型，那么对所有相关人等都会是很好很容易理解的。所以实际上，在我做业务建模的时候，虽然用了 UML 的元素，但实际我的概念是岗位、角色，我认为这两个概念足以支持业务分析，并且容易理解，而抛弃了 UML 拗口复杂的定义。这在实际工作中给我带来了很方便。

第二个问题，总的来说，我支持你的分析。我的理由是：

首先分为求租者和寻租者我是支持的，定位很准，而客人业务角色呢，我猜想你这位朋友带了抽象的思想在里面，实际上他的意思是不论求租者和寻租者在将来的 WEB 应用程序看来都是通过同一个界面登录的，可能也是通过同一个界面管理的。所以从 ID 的角度说，他们是一样的。但我不支持在这个时候带着实现的思想，而要从业务角色的目的上去区分它们是否应该合并。显然，求租和寻租两者的目的是完全不同的，在业务角度说，他们没有任何可以合并的理由。至于到了系统用例阶段，由于他们可以共享同样的登录模块，同样的 ID 管理，抽象出一个客人角色是合理的，但在业务阶段这个抽象是不合适的。同一个参与者使用同样用例却抱着不同的目的，这违反用例的基本定义。

其次，在业务用例的获取方面，可以参考我在第一篇文章里讲到的用例的四个特征，你的业务用例定义符合得很好。对于你第一个朋友的定义，发布信息这个业务用例，同时包含进了注册店面，我不大赞同。原因在于，业务用例带有“原子”特性，也就是说一个业务用例应该能够不依赖于别的业务用例而完成一个参与者的目的。如果按你第一个朋友的定义，会有这样一个结果，寻租者启动同一个用例，然后去做一件事情，而这件事情与用例中另一件事完全无关。例如

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

这样一个场景，某个出租者，注册了店面，成功后就离开了，显然，他并没有做发布信息的事，发布信息并没有在这个场景中 发挥任何作用，反之也一样。既然这两件事情可以独立存在，就应当独立成用例。我猜想你朋友的意思是，要发布信息，就要先注册店面，有无店面是发布信息的一个前置条件，因此把它包括进来。如果是这么想的，有其道理，但就这个事例来说我还是觉得不合适，我提出这几个问题：是否要发布信息就必须有房间？是否每发布一次信息都要注册一次房间？一个 ID 是否能够注册多个房间？如果房间有时限规定失效了以后怎么办？如果客人想注销房间呢？想想看，这些问题都是针对注册 房间的，如果注册房间是发布信息用例的一部分，结果是什么呢？为了发布一条信息而已，客人被要求做那么多准备工作，开发人员写了 N 多 if-else。可其实呢，以上的问题都不直接涉及到发布信息的流程啊，方法啊这些。可见，把两个原本独立的目标（一个客人上来，可以只发布信息，也可以只注册房间，未必都要做）合并，会造成混乱。如果要问，我提的那几个针对房间的问题，的确会影响到是否可以发布信息啊，还有信息发布到哪里啊？没错啊，是否可以发布是发布信息的前置条件，发布后被放到哪里是发布信息的后置条件，它们都是在发布信息之外的，换句话说，注册房间和发布信息之间有一些业务规则需要定义，仅此而已。这和我们不需要在每个用例里都去包括注册用户用例是一个道理，虽然没有正确的 ID 注册和登录不能做任何事情，其他用例也不必去包含注册 ID 的用例，因为客人 的确可以注册完成以后什么事都不做。注册 ID 用例和其它业务用例有关系是因为业务规则，而不是因为客人的目的。第二个朋友的我就不细说了，原因大致同上，另一方面，由于我不赞同业务角色的获取，当然从它得出的业务用例我也不赞同。

有一点是值得讨论的，就是查询用例。这是一个比较特殊的东西，因为查询的目标可以很明确，也可以很模糊。比如，我就只是查查看而已，看完了什么也不做；也可能是因为我要租房，所以查询；还可以因为是我查完后修改我已经

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

发布的信息...类似这样的不同目标还有很多，显然我们不可能每一个目标都去定义一个查询 XXX 业务用例。同时，一个查询可能是跨越多个业务用例的，比如把求租、寻租和用户信息集中在一个查询结果里面。在这一点上我也不能断定你专门目标的用例：找房屋信息业务用例，和你朋友通用目标的用例：查询信息业务用例哪一个是正确的，事实上都是正确的，也都有道理，取决于客户需求中针对查询的要求，到底是专门目的，还是模糊目的。

coffeewoo 发表于:2006.11.08 13:52:

谢谢你对我提出的案例做分析！

你文中提到的岗位和角色能否举例描述一下，也算是学习下你的实际经验

当然，我个人还是保留对 UML 的意见，RUP、UML 毕竟是许多历史经验的总结，我一直认为是自身的认知程度还不够，还需要不段的理论、实践并与同行交流总结再回去认知才能升华

我现在对 uml 已经琢磨到了分析阶段，你的设计师之路的第一篇已拜读，关于实体、边界、控制产生了新的困解，特别是你拿它与事、物、规则做对照后，困解更大了

现在的问题是：

1、实体=M?、边界=V?、控制=C?

特别是当你把边界与事做对照后，这种困惑就更大

图书管理员往系统里填写借阅人的基本资料，系统展示的界面形式(web or form)也就是 V，和事是什么关系？

2、可能也是对 MVC 的理解问题

业务逻辑存在与实体 OR 控制中？还是都存在？在某些文章中甚至都有人对 UML、OO 提出了反对论

一个实体修改自身属性，同时在数据库也做了同步，这个同步操作是实体自己做还是控制类来完成？

等你《。。。设计之路》写完，还将继续探讨下中介的例子啊：)

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

[michaelyan](#) 评论于: 2006.11.08 19:17

岗位，角色是客户喜闻乐见的形式。他们对自己的岗位职责最清楚，所以如果画出一个名字等于他岗位的 actor，他立刻明白是在说他呢。比如办公室主任，副总经理，总经理，谁看了都明白。而按 UML 的定义，一个主角是代表有共同特征的一批涉众的。这个定义就隐含了一个“抽象”概念在里面。假设前三者都能做审批文件的事情，在 UML 的定义下，自然而然会产生一个类似“文件审批者”这样的主角去完成“文件审批”这样一个业务用例。我的想法就是，这个抽象概念必要吗？“文件审批者”这样一个由系统分析员“创造”出来的东西，虽然非常符合 UML 的思想，但是用户，开发人员，测试人员，能够有很好的理解吗？如果用岗位来代替，实际上放弃了“抽象”的概念，对于审批文件这个业务用例来说，大不了三个岗位都指向它，就 OK 了。

抽象概念什么时候要？在做系统用例时，用角色这个已经用了很多年的词儿来替换 actor。这个时候倒是有可能“文件审批者”这个抽象会出现，不过由于已经到了系统用例阶段，客户可以不参与，所以抽象就没那么大问题了。

其实我称之为岗位，角色，UML 叫 business actor, actor, 还有 worker，实际上做出来的差别是非常小的，甚至很多时候是重合的。所以我更觉得没必要去给用户解释什么是主角什么是业务主角，直接问他，这个岗位做什么。

关于 MVC 模式和人事物规则，和分析类的关系我在以后的文章里会讲到的。

[coffeewoo](#) 评论于: 2006.11.08 20:15

就回答一个最简单的问题：

业务用例阶段请不要做过多抽象，因为你是想描述需求，而不是在做分析

系统用例阶段请进行有必要的抽象，因为你是想描述人与系统间针对需求的交互，所以你在做分析

分析模型阶段请靠近代码级的抽象，因为你是想描述系统究竟是怎样完成人所提出的需求的，所以你在做分析和设计的过渡（这时候依靠的就是业务对象之间所进行的交互）

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

设计模型阶段请进行代码级的抽象，你会发现许多 CLASS(业务对象、业务逻辑)，于是你也会使用常见的时序图来表述

其实抓住这些重点，你就会发现在任何一个阶段你都有你的目标，做起事情来就清晰多了。

MVC 属于构架体系的模式，如果理解为实体=M、边界=V、控制=C 没有问题，问题的关键在于对于一个用例实现你在设计时怎样将这三者清晰的分离，尤其针对的是 BS 结构。

rwyx 评论于: 2006.11.10 01:13

totally agree with you!

另外，文章一时半会儿写不完，估计大家对我把边界和事做对照很迷惑，这里先小解释一下 😊

一件事情，在一个准备做它的人看来，它是什么样儿的？一个做事人对一件事的理解不是这件事的内部概念，而是怎么去做，可以做什么，什么不能做。就比如开车我相信不会有人问什么是开车啊？而会问，怎么开车啊？于是你的解释不会是：开，是开车的开，车是开车的车，而会是：怎么把方向，怎么踩离合，怎么给油，对吧？

好了，不再多说，留点空间给大家考虑，就我上面的小解释，想想为什么我会把事和边界做对照 😊

coffeewoo 评论于: 2006.11.10 10:56

对 uml 的研究又前进了一点...

关于边界，举个例子，不知是否恰当？

比如你要去看电视，你看到电视机的所有表面，比如按钮、有限电视插口等都属于边界，而各个按钮可以干什么，是控制器，按钮按下以后会发生什么事情，首先，如果你看过说明书，那么你已经知道每个按钮做什么事情，说明书也应该属于边界，如果没有看过说明书，按钮按下后，发生的事情可以通过发生的结果而得知，比如关闭按钮，按下后，屏幕（也属于边界）什么也看不到，内部发

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

生什么事情不用管它，对于电视机设计开发人员，他们知道内部的某些对象进行了交互，而 后实现该按钮需要完成的事情

[michaelyan](#) 评论于: 2006.11.11 10:26

😄 恭喜，其实这不仅是对 uml 的理解，而是对 OO 方法的理解。从接口的角度去理解事物，任何事物都需要有一个边界，各负其责，边界两边的对象只通过边界交互，双方绝对不会去试图存取对方的内部。这是 OO 思想的真正精髓，真正理解了这一点，什么设计模式，什么 SOA，什么 AOP 之类的都只是一种技巧和应用了。

其实在 OO 世界里，只有两点是最重要的，不论是在需求，分析设计，实现还是观察系统，只要随时能从这两点去思考就是一个真正的 OO 思想者了，这两点就是：边界，职责。最好的抽象，就是找出最适当的边界，并且职责分工明确。

举个有意思的小例子

我在什么是用例一文中说用例就是一件事情。正好和这篇当中的事是对应起来的。用例图象什么样子？一个个鸡蛋？呵呵，很形象啊，actor 看用例时只能看到蛋壳，蛋壳就是边界，就是这一件事的边界，你对鸡蛋的所有理解都是来自这个 shell。如果你试图了解壳以内的世界，打破了边界，恩，的确看到了，不过很快 就后悔了，鸡蛋被破坏了，一滩粘粘乎乎的蛋清弄脏了手，很难收拾。糟糕的设计就象一堆破了壳的鸡蛋，一片混乱

[coffeewoo](#) 评论于: 2006.11.11 13:26

呵呵，比喻的很形象

设计和定义好的边界非常重要，职责是 ENGINEER 需要完成的事情，控制器也被边界给包容了，所以在设计和定义时必须非常的小心

[michaelyan](#) 评论于: 2006.11.11 14:50

10.2 需求规则说明书示例

网上图书借阅管理系统 需求规格说明书(示例)

声明：本示例是网络 ID 为 coffeewoo 的作者为配合其系列文章< 00 系统分析员之路—用例分析系列>所提供之辅助阅读材料。仅供学习和交流之用，请勿用于任何形式的商业用途。违反此约定导致的法律后果由使用者自行承担，coffeewoo 不承担任何责任，并保留维护自身利益的权利

2006-9-10

2006 年 09 月 10 日

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

修订记录

序号	时间	修订内容	修订人
1	2006 年 09 月 10 日	创建文档	coffeewoo

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

目录

1	引言	1
1.1	概述	1
1.2	背景	1
1.3	定义	1
1.4	参考资料	1
2	任务概述	1
2.1	目标	1
2.2	用户的特点	2
2.3	假定和约束	2
3	需求规定	2
3.1	对功能的规定	2
3.1.1	用户需求	2
3.1.1.1	组织机构和角色	2
3.1.1.2	业务概览	4
3.1.1.3	业务场景	5
3.1.2	系统需求	6
3.1.2.1	概览	6
3.1.2.2	系统需求规定	7
3.1.2.3	数据分析	12
3.2	补充规定	14
3.2.1	精度	15
3.2.2	时间特性要求	15
3.2.3	灵活性	15
3.2.4	界面要求	15
3.2.5	可靠性	15
3.2.6	可用性	15
3.2.7	可维护性	15
3.3	输入输出要求	15
3.3.1	安全性	15
3.3.2	输入输出模式	15
3.4	故障处理要求	15
3.5	其他专门要求	15
4	运行环境规定	15

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写，原始出处为 <http://coffeewoo.itpub.net>。本文档乃热心的读者自行由作者 BLOG 下载整理而成，并发给作者本人，再由作者本人向外发布的。本文允许自由传播，仅供个人学习之用，禁止用于商业行为，如出版物，培训教材等。读者在使用本文时请尊重作者之著作权，勿篡改或删除或改编本文。若有非个人的任何公司，组织和商业机构想采用本文之全部或一部分，请与作者联系，勿私自使用。若出版商有意出版此文，请与作者联系。谢谢合作。

4.1	设备	15
4.2	支持软件	16
4.3	接口	16
4.4	控制	16

引言

概述

说明文档目的, 针对的目标读者, 文档内容, 文档组织结构等

背景

说明项目提出的背景, 应用环境, 应用范围, 目标人群等

定义

列举文档中所用到的专业名词, 所使用的术语含义

参考资料

列举文档所引用到的资料, 例如行业规范, 法律规章, 用户的岗位手册, 工作流程等

任务概述

目标

说明系统建设目标, 针对背景, 系统要解决的问题

用户的特点

说明系统目标人群的特点,使用习惯,使用场景,使用频度,以及人群的计算机水平等

并以下表列出系统的使用者,以及使用者所代表涉众(参看系列文章的第 3 部分)

使用者名称	说明	代表的涉众

假定和约束

说明针对系统使用 and 开发,以及目标人群的假定和约束,例如用户支持数量,系统运行环境等

需求规定

对功能的规定

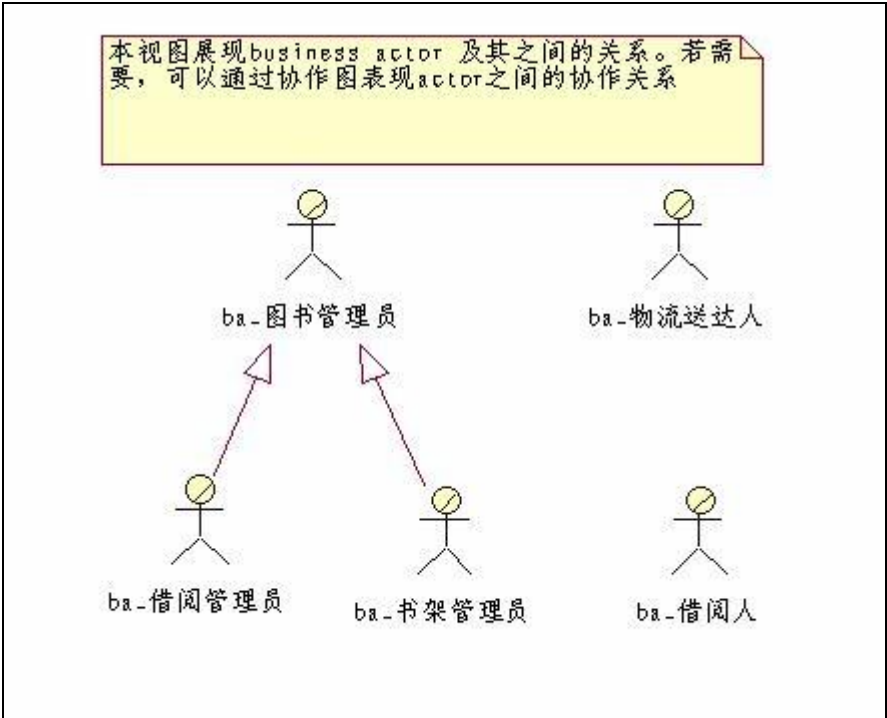
用户需求

组织机构和角色

说明系统角色及它们组织机构中所处的位置。将用例分析结果的 Actor 视图拷贝到此,并用表格逐一说明。

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

角色视图:

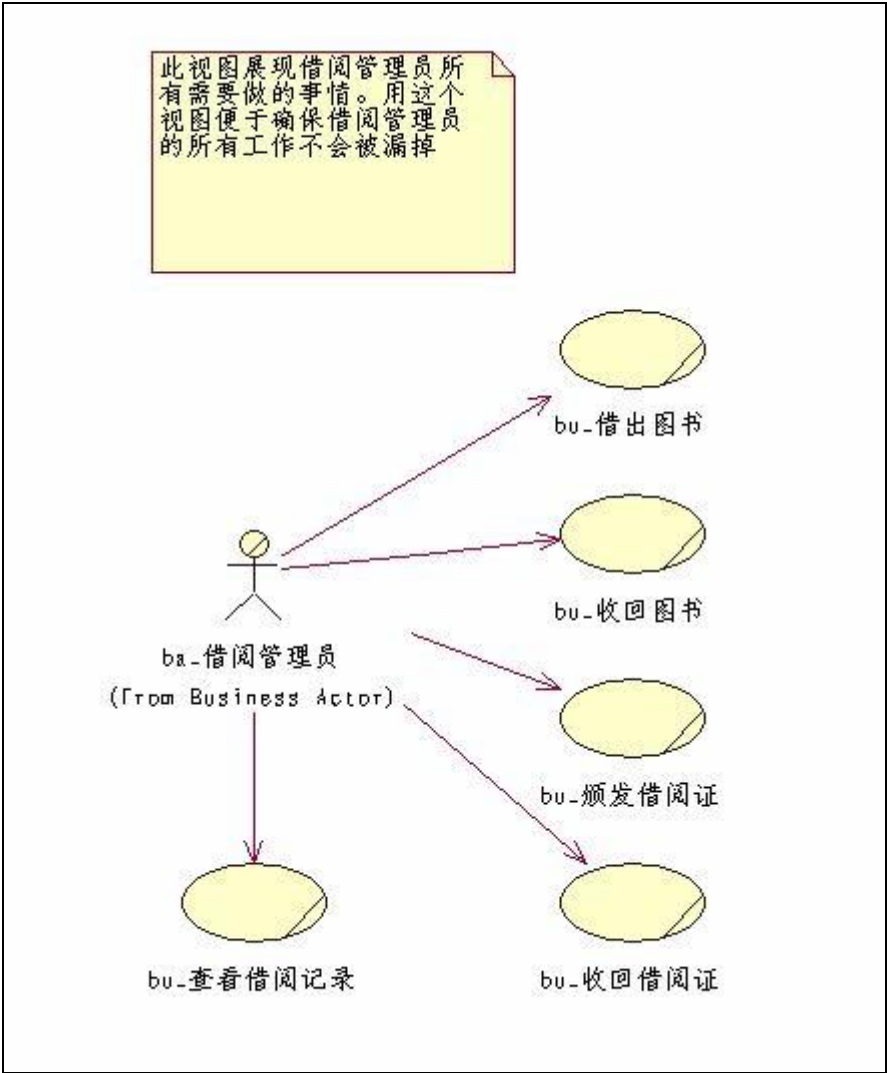


角色说明:

角色名称	说明
	说明角色代表的用户, 其岗位职责, 在组织机构中所处位置

再将业务用例模型中的 Actor 视角视图拷贝至此, 逐一说明角色如何参与业务, 参与哪些业务

例如: 借阅管理员参与业务:



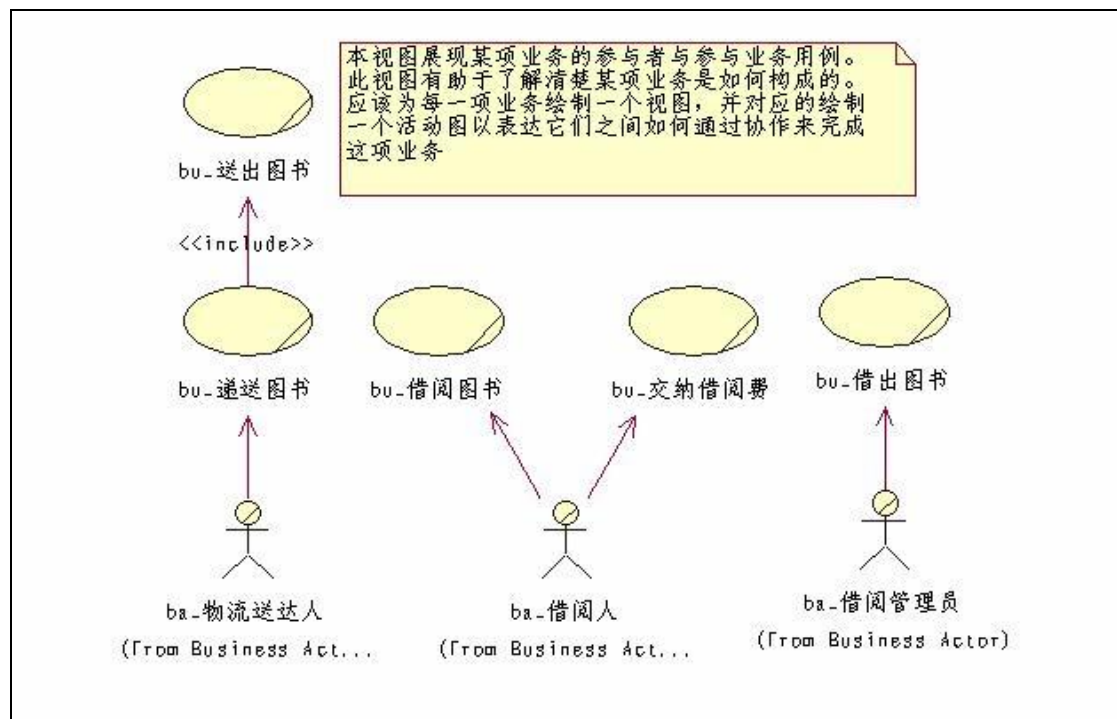
说明:

业务概览

将业务用例模型的业务视角视图一一拷贝至此,逐一说明

例如: 借书业务

本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或一部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。



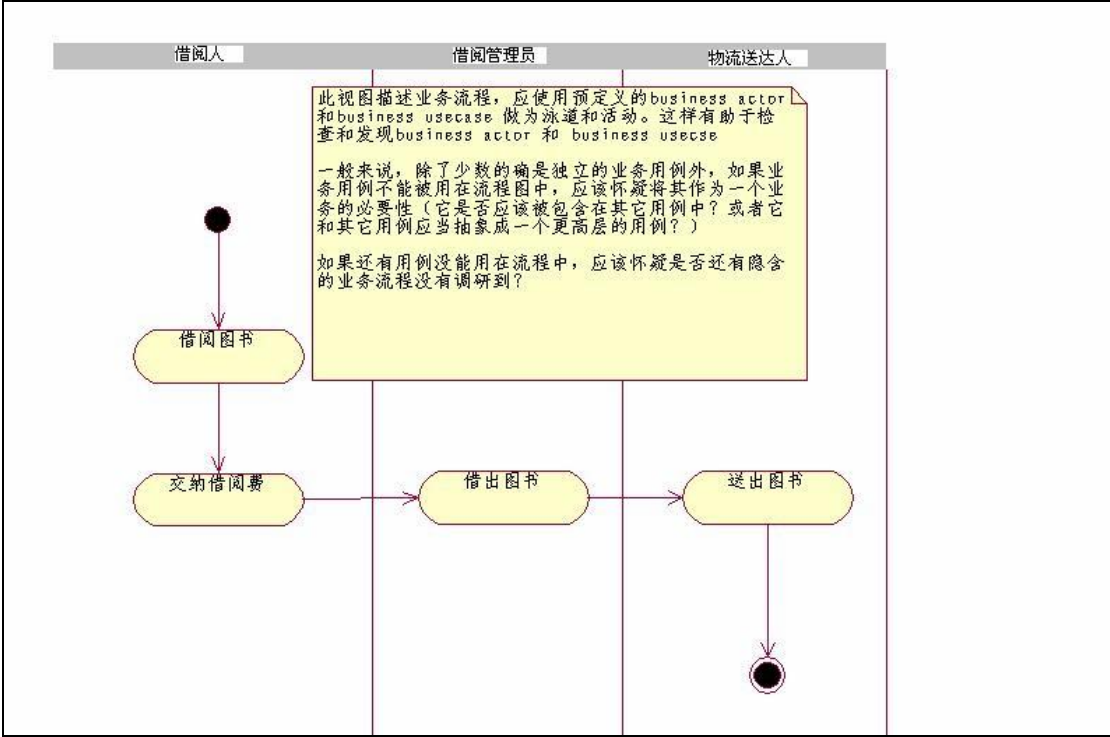
借书业务说明:.....

业务场景

将业务场景图拷贝至此,逐一说明业务是如何进行的

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或一部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。



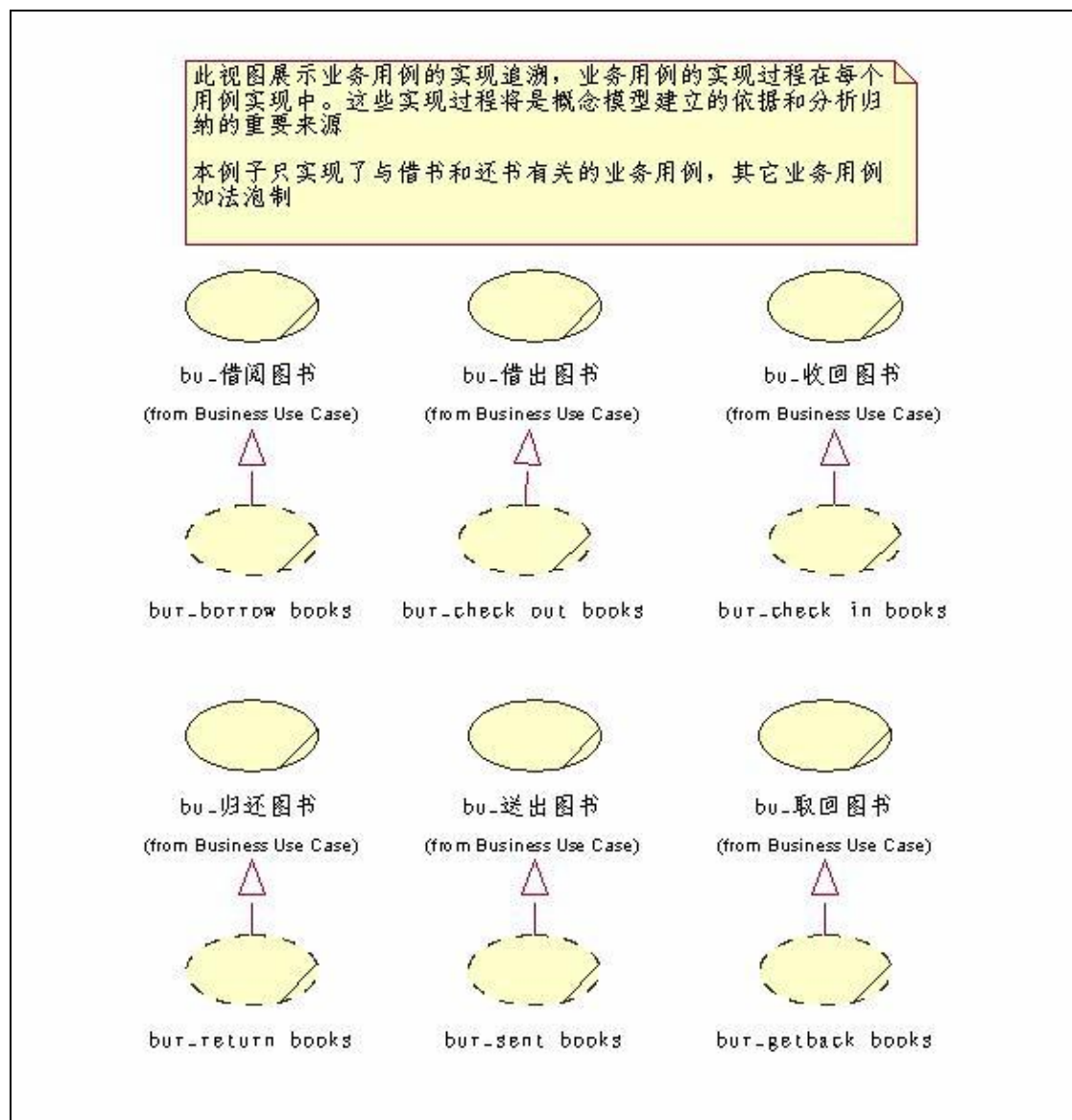
业务场景说明.....

系统需求

概览

将用例实现视图拷贝到此,并进行说明

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^



系统实现或不实现的用例, 范围及描述

系统需求规定

针对每一个用例实现, 拷贝用例规约和用例场景至此, 同时需要拷贝用例实现针对领域模型。

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。

借阅图书

业务说明

拷贝每个用例规约至此

用例名称	bu_借阅图书
实现名称	Bur_borrow books
用例描述	借阅人通过此用例向系统查询并提交借书请求
执行者	借阅人
前置条件	3. 借阅人借阅证件在有效期内 4. 借阅人没有逾期未归还的图书
后置条件	3. 创建借书定单 4. 更新借阅人借阅记录
主过程描述	1 用户用借阅证提供的帐号登录系统, 计算机显示我的图书馆界面 2. 用户选择查询图书, 计算机显示查询界面 3. 用户按书名、作者、出版社查询, 计算机显示查询结果 4. 用户可单选或多选书本, 并确认借阅。计算机显示确认借阅图书清单。 5. 用户选择确认借阅, 计算机显示借阅定单及费用 6 用户选择提交定单, 计算机显示提交结果和定单号 7. 计算机执行后置条件。用例结束
分支过程描述	2. 1. 1 用户选择查看原有定单, 计算机执行 4; 4. 1. 1 用户可单选或多选书本, 放入借书篮, 计算机显示借书篮现有内容 4. 1. 2. 1. 1 用户选择继续借书, 计算机执行 2; 4. 1. 2. 2. 1 用户选择提交借书篮, 计算机执行 4 4. 2. 1 用户选择放弃, 计算机执行 2; 6. 1. 1 用户选择保存定单, 计算机保存并执行 1; 6. 2. 1 用户选择放弃, 计算机执行 1;
异常过程描述	1. 1. 1 借阅证已过期, 拒绝登录, 用例结束 1. 2. 1 借阅人有逾期未归还书本, 启动 bu_归还图书用例 5. 1. 1 用户余额不足, 计算机显示余额和所需金额 5. 1. 2. 1. 1 用户选择续费, 启动 bu_交纳借阅费用用例 5. 1. 2. 2. 1 用户选择放弃, 计算机执行 1
业务规则	4. 至少选择一本, 至多选择三本

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

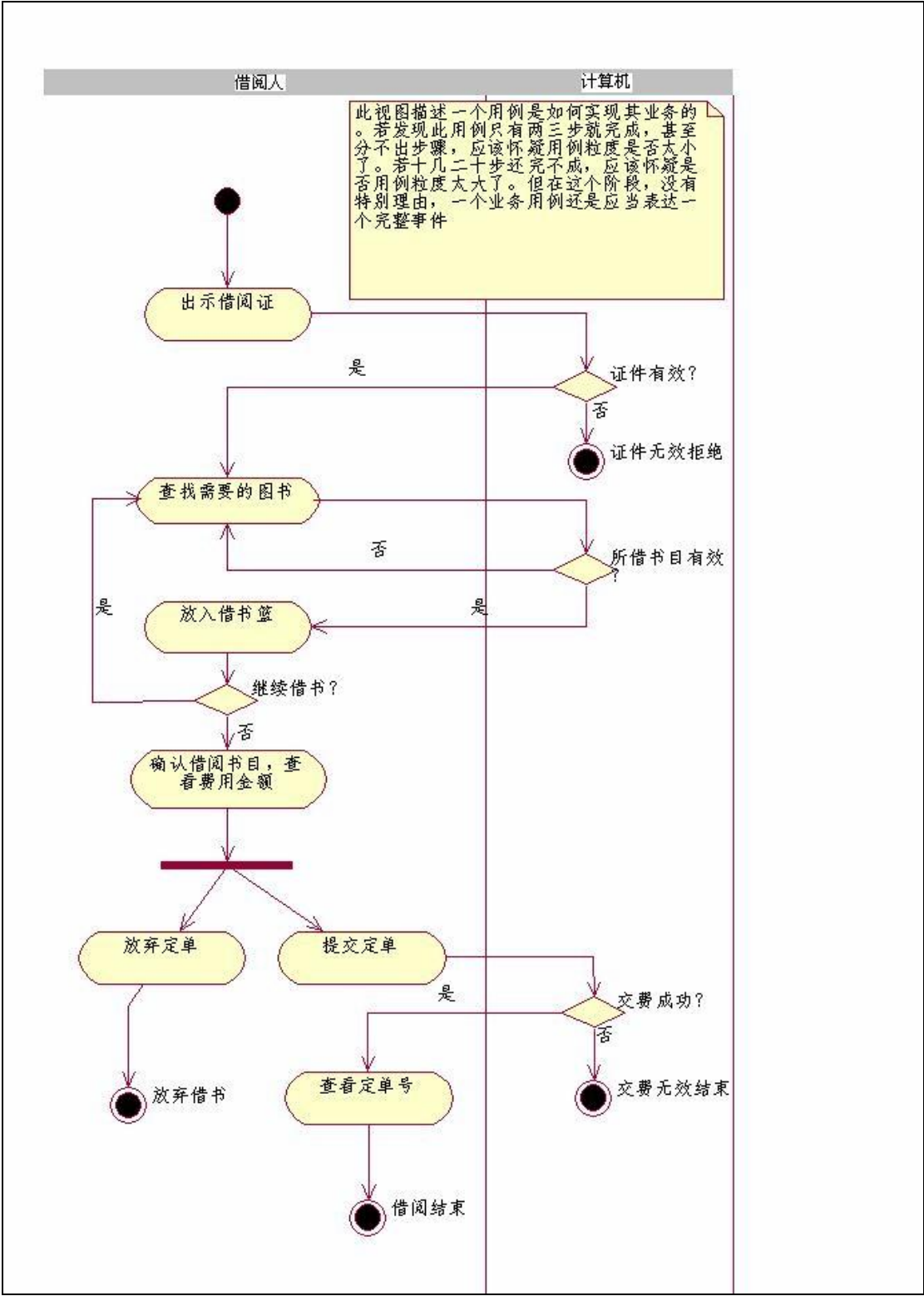
本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或一部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。

涉及的业务实体	Be_费用记录, Be_图书, Be_借书篮, Be_借阅定单, Be_借阅证
补充说明	

业务场景分析

拷贝每个用例场景至此,并说明之

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

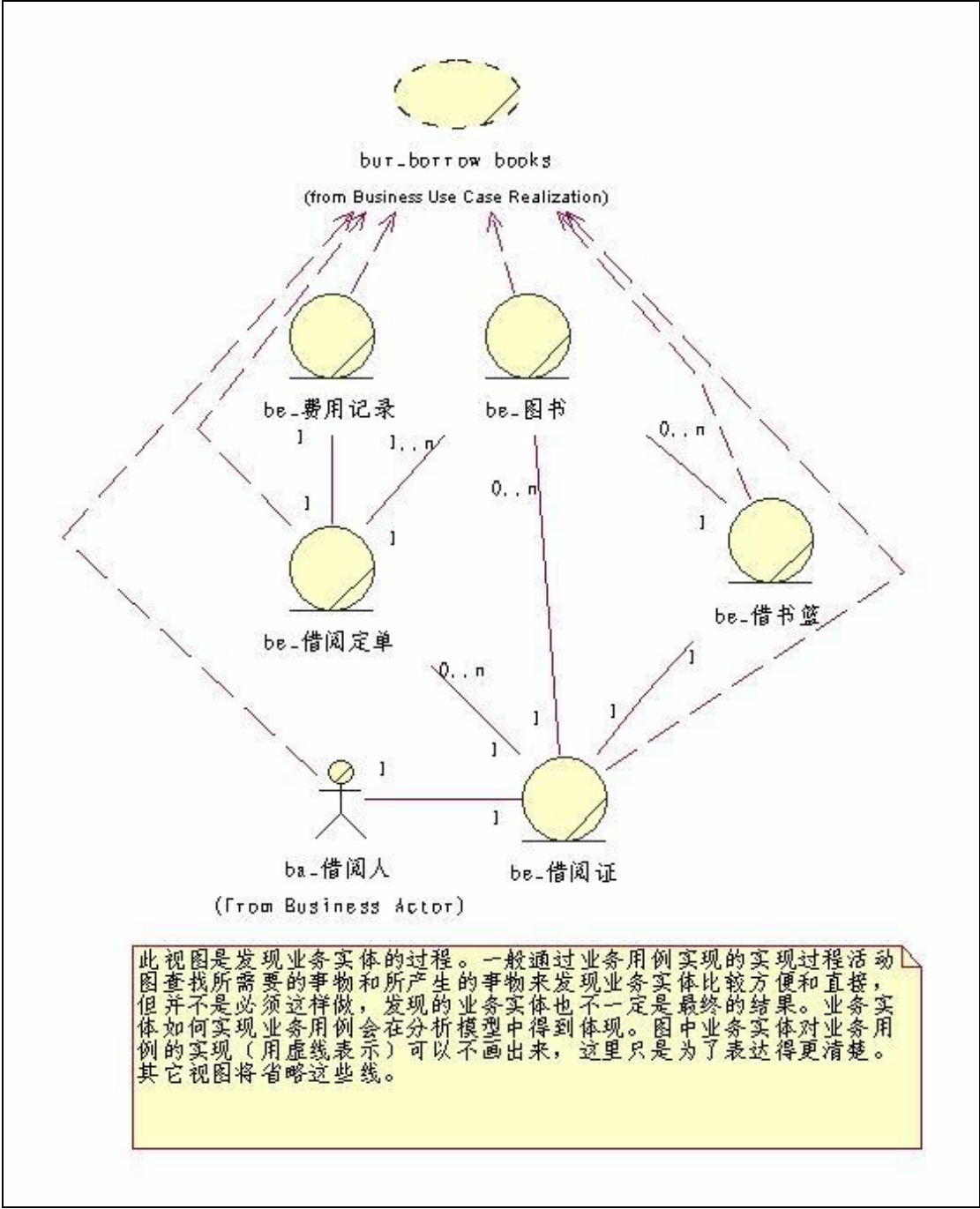


业务场景说明.....

作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载, 敬请注明, 谢谢 ^_^

业务实体分析

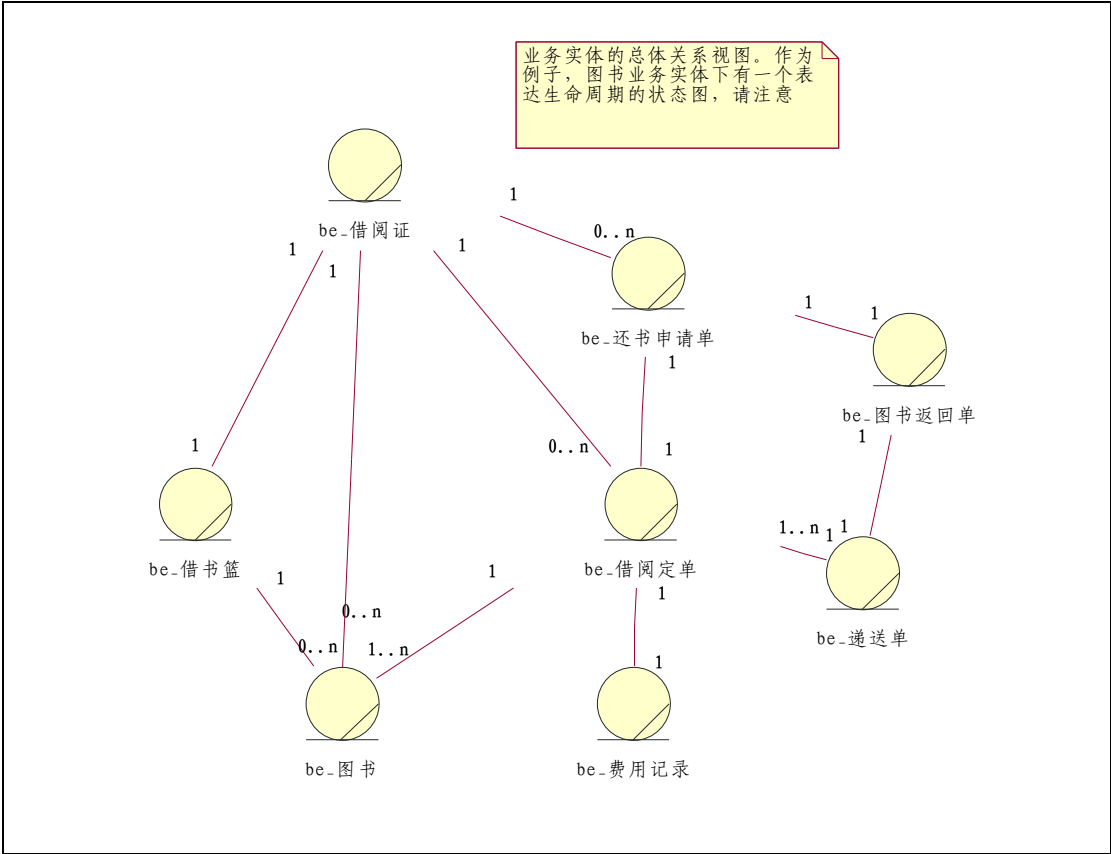
拷贝对应的业务实体视图至此,逐一说明其如何参与业务



数据分析

概览

将总体的业务实体视图拷贝到此,并作说明



实体之间关系说明.....

图书

针对每一个实体,说明其详细情况,将领域模型说明表格拷贝至此

实体名称	Be_图书
实体描述	每本图书都经有上架,预定,借出,返回待查和下架几个状态,详细请参看图书状态图

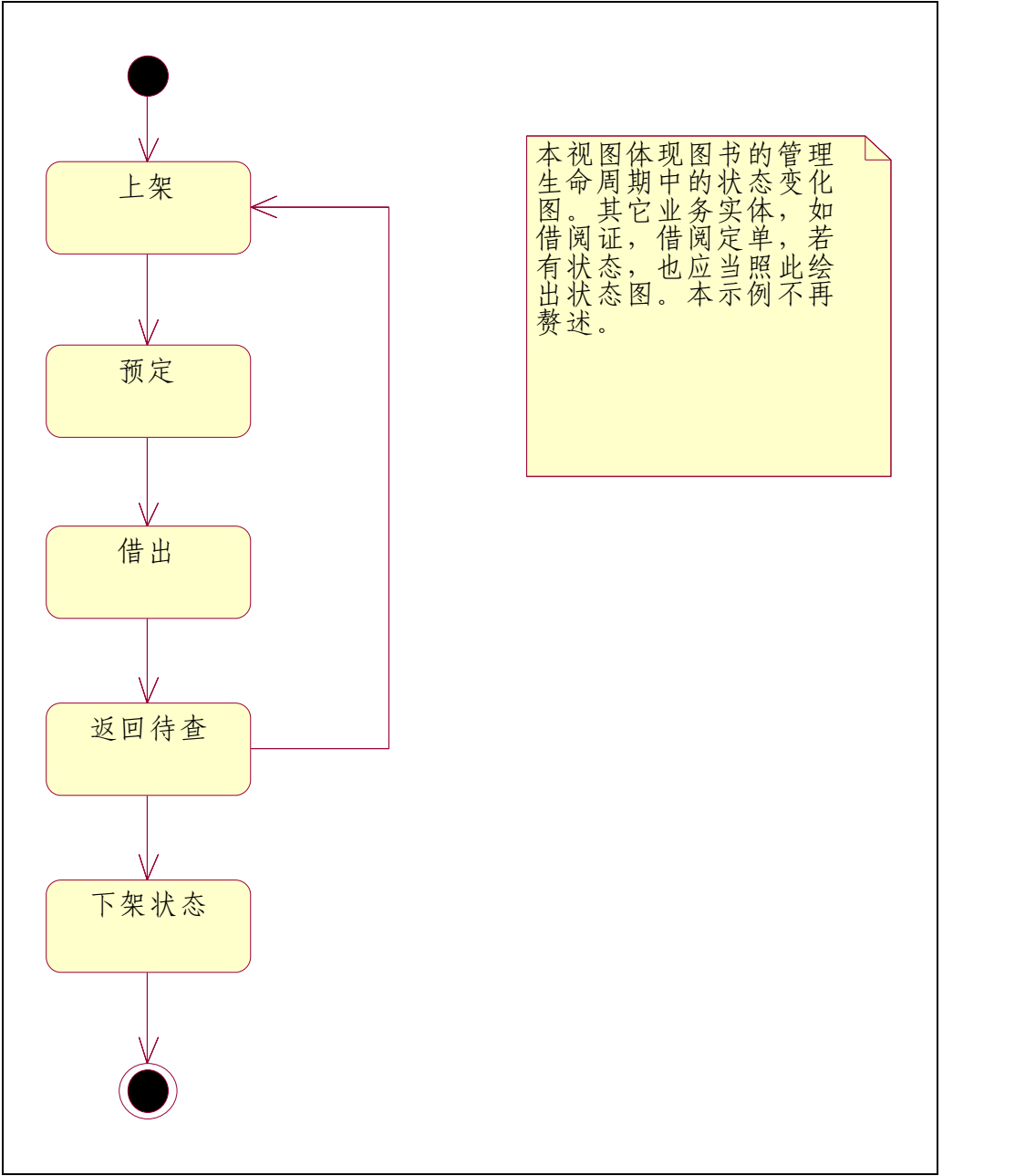
作者 coffeewoo 欢迎您访问文章原始出处: <http://coffeewoo.itpub.net>, 阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net, 我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载,敬请注明,谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或一部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。

属性名称	类型	精度	说明(属性的业务含义及业务规则)
图书编号	字符	12	图书类别编号(3 位)+图书购入年份(4 位)+流水号(5)位
图书分类	字符	3	图书的分类
名称	字符	100	书本的封面名称
作者	字符	20	书籍的作者
出版社	字符	100	书籍标明的出版社
出版日期	日期		书籍标明的出版日期
版本信息	字符	100	书籍标明的出版社
简介	字符	1000	书籍的内容简介,上架时录入
状态	字符	1	书籍的状态,请参看图书状态图

若有针对实体状态图或其它视图,也拷贝至此

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^



补充规定

在此章节中描述用例补充规约中的相关内容，有多少写多少，没有的可以不用写

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或一部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。

精度

时间特性要求

灵活性

界面要求

可靠性

可用性

可维护性

输入输出要求

安全性

输入输出模式

故障处理要求

其他专门要求

运行环境规定

设备

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^

本文是网络 ID 为 coffeewoo 的作者原创编写,原始出处为 <http://coffeewoo.itpub.net>。本文档乃从原始 BLOG 下载整理而成,不包括最新的进展。本文允许自由传播,仅供个人学习之用,禁止用于商业行为,如用于出版物,作为培训教材等。读者在使用本文时请尊重作者之著作权,勿篡改或删除或改编本文。若有非个人的任何公司,组织和商业机构想采用本文之全部或一部分,请与作者联系,勿私自使用。若有版商有意出版本文也请联系作者,谢谢合作。

支持软件

接口

控制

作者 coffeewoo 欢迎您访问文章原始出处：<http://coffeewoo.itpub.net>，阅读中有任何问题可以在 BLOG 上留言或发邮件到 coffeewoo@263.net，我将尽力为您解答。具有代表性的问题我会以 Q & A 的形式收录到对应的文章之后。希望本系列文章对您有帮助。欢迎转载，敬请注明，谢谢 ^_^