# 用 例 分 析 实 践 体 会

## The Experience on Analysis of Use Case

吴　艳　金献珍　（浙江工业大学 之江学院 浙江杭州 310024）

**摘　要：** UML 中的用例是以“项目型态( Project – based )”开发的最有效利器，它表达的就是纯粹的信息系统局部功能观点的需求模型。有效表达系统功能的用例是项目开发成功的最重要的关键。根据自身实践的体会，提出了在用例图设计中应把握的几个设计准则，并对列出的几个比较常见的案例进行分析，提供了修正后的建议。

**关键字：** 用例模型　建模　案例　S. M. A. R. T

## 1　引言

要开发复杂的系统，开发者必须抽象出系统不同的视图，使用准确的表示法来构建模型，检验模型是否满足系统的需求，并逐步地给模型增加细节，最终将其转换并实现[1]。常见的模型有三类：(1) 类模型表示系统静态的、结构化的“数据”层面；(2) 状态模型表示系统时序的、行为的“控制”层面；(3) 交互模型表示独立对象的协作，是系统的“交互”层面。用例图、顺序图和活动图用来描述交互模型。用例描述系统和外部参与者之间交互的主要内容，每个用例都表示系统提供的、可以单独运行的一段功能。

用例模型(USE Case Model)易学难精，入门很快：只用几个简单的椭圆就能画出系统功能、几个小人就能表示系统的参与者以及仅用几个文字就能记录功能的需求描述。事实上，在许多案例中根本就是把用例模型当成是另一种需求记录的工具而已。就其原因，最大的问题在于写用例的需求分析人员摆脱不了传统思想的束缚，例如他们会从系统的业务流程(Business Process)的观点来分析用例，或者直接将系统内部的实作反映在用例图中。本文将根据用例设计中的一些标准，结合自身的实际体会，就几个经过简化的实际案例进行分析，并提出修正意见。

## 2　用例设计中的标准

用例是以“项目型态(Project – based)”开发的最有效利器，它表达的就是纯粹的信息系统局部功能观点的需求模型。要画出有效的用例图必需遵循 S. M. A. R. T 原则和用例模型的准则[2]。

### 2.1　S. M. A. R. T 原则

用例的命名法则，一定是“动词”+“名词”，充分表达系统的“What can system do for actor”的目标。用例的单位要符合“S. M. A. R. T”原则，即“Specific(具体的)”、“Measurable(可靠的)”、“Accurate(精确的)”、“Reachable(可达成的)”、“Time – limit(有时限的)”。更简单地说，每一个用例都是可以个别在某个时间点内通过系统能履行的功能来满足参与者使用系统的目的。

### 2.2　用例模型的准则

用例标识系统的功能，并根据用户的观点组织这些功能。下面是一些构建用例模型的指南[3]：

(1) 首先确定系统边界。必须识别参与者和用例。

(2) 关注参与者。每个参与者都应该有单一的、一致的目的。

(3) 提供给用户的用例必须有价值。用例应该给用户提供有价值的、完整的事务。

(4) 关联用例和参与者。每一个用例都应该至少有一个参与者，每个参与者都应该参与至少一个用例。

(5) 用例是非形式化的。用例应该以用户的角度来识别和组织系统功能。

(6) 用例可以结构化。对于大型系统，可以使用包含、扩展和泛化关系从较小的功能片断来构建用例。

### 2.3 用例设计要点

根据以上原则或准则,结合实践运作的经验,把握以下几个设计准则,有助于提高用例模型的有效性:

准则 1 不要从业务流程观点描述用例之间的关系,不要用用例关系反映系统的实作。

准则 2 用例名必须符合命名法则。

准则 3 不要试图用用例表达模块化的分析思维,用例单位要符合 S. M. A. R. T 原则。

准则 4 不要混淆包含关系和扩展关系。

准则 5 不要混淆包含关系和泛化关系。

准则 6 开发系统的本身不要成为自己的参与者。

## 3 案例问题分析

下面通过几个经过简化的实际案例分析用例中存在的问题,并提供修改后的建议。

### 3.1 案例 1

图 1 所示的员工管理系统用例图中有三个问题:(1)问题 1 违背了准则 1,用例名应该是“验证密码”,而不是“密码验证”;(2)问题 2 违背了准则 1 和准则 4,用例不是表达业务流程的,不表示系统的实作,用例只专注与参与者与系统之间的交互过程,仅仅是表达某个时间点使用系统的局部功能观点。实际上,参与者(actor)使用“提出请假申请(apply for leave)”用例,多少就隐含了权限控制的考虑了。再者,行政人员可以“维护员工信息(maintain employee)”,但员工没有权限使用该用例;(3)问题 3 违背了准则 3,原因是采用了模块化的分析思维,将大功能逐层分解成小功能,利用用例图取代传统的模块化的表达。用例“维护信息(maintain information)”只是一个虚的、不可达成的用例,因此,该用例无需体现在用例图中。解决以上三个问题的修正图如图 2 所示。

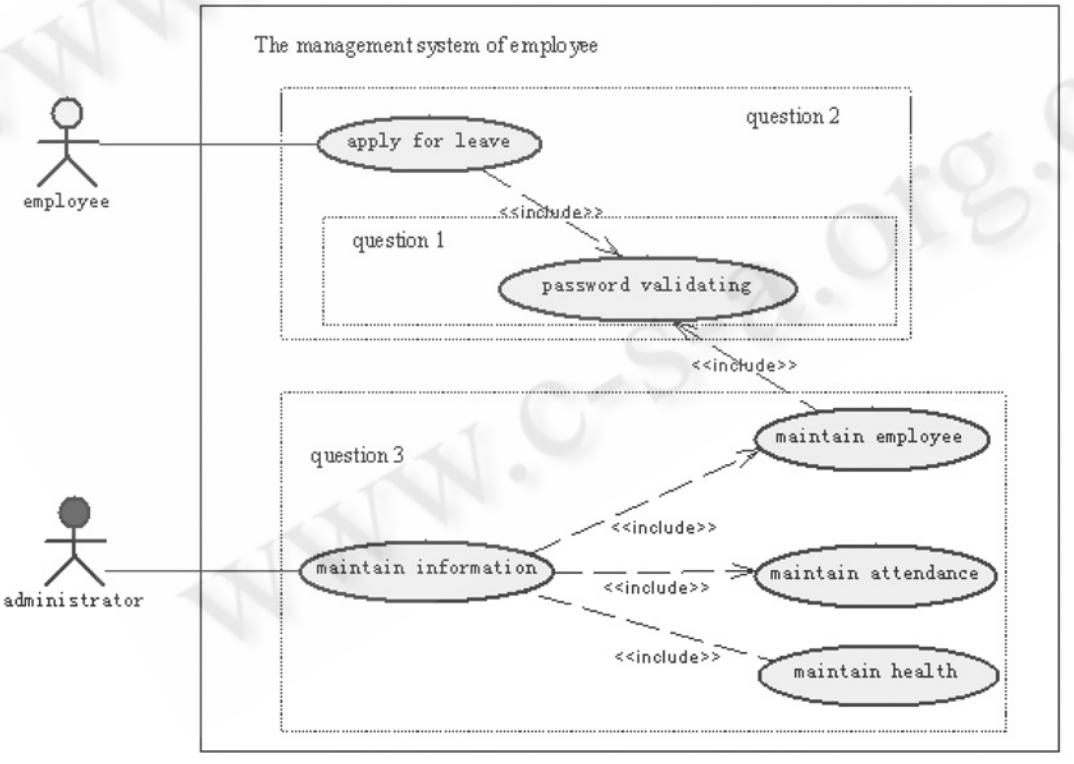


图 1 用例图中存在的 3 个问题

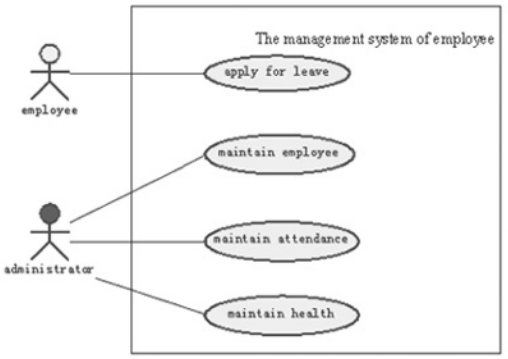


图 2　图 1 所示用例图的修正

### 3.2 案例 2

图 3 显示了股票经纪人系统的用例图。图 3 中的问题是滥用泛化、包含和扩展关系：(1) 问题 1 违背了准则 4 和准则 5。“ 安全会话(secure session) ”用例不是‘ 做交易(make trade) ”用例的变体，前者并非特化后者，只是共享后者的行为片断，因此，应该使用包含关系。“ 管理帐号(manage account) ”用例是‘ 安全会话 ”用例的一个共享行为，而不是在‘ 安全会话 ”用例的某个控制条件为真时才启动一个‘ 安全会话 ”用例，因此，在此应使用安全关系；(2) 问题 2 违背了准则 4。在账户资金不足时，“ 交易债券(trade bonds) ”用例才会启动用例‘ 保证金交易(margin trading) ”，支持贷款购买股票，因此，应该使用扩展关系。解决以上两个问题的修正图如图 4 所示。

### 3.3 案例 3

当需求分析人员发现系统需要在某个时间点‘ 自动 ”执行某个用例时，会以为系统本身就是参与者。

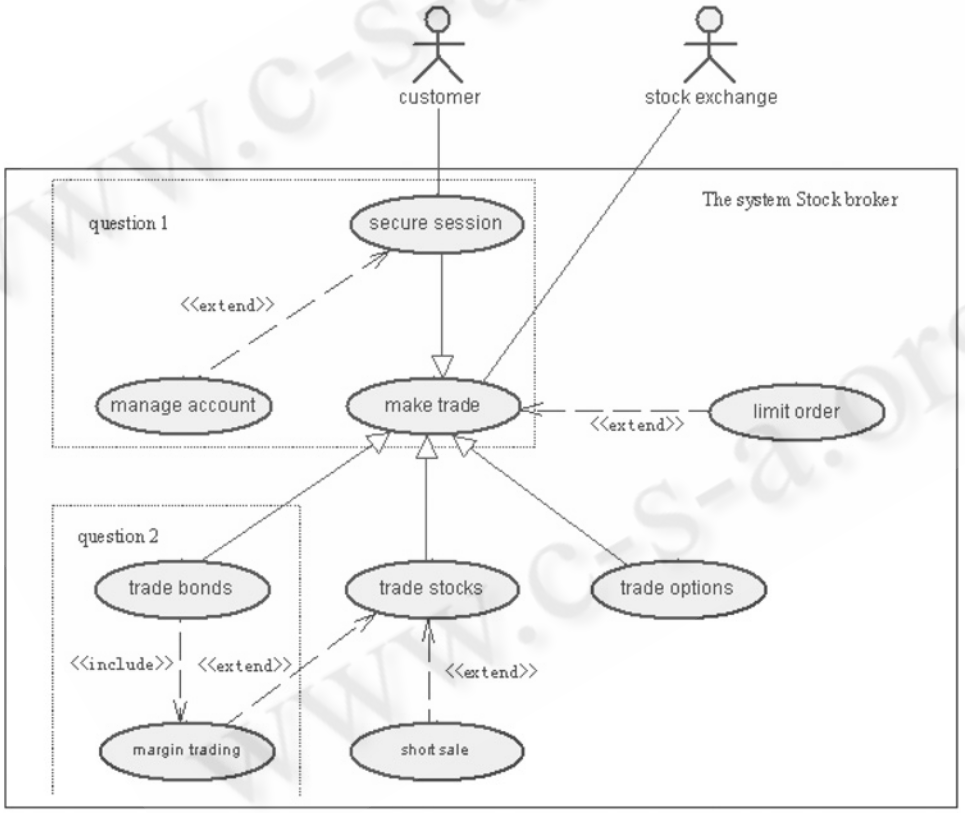


图3　用例图中存在的2个问题

ERP 的系统作业，如结账、批处理等相关作业，就会常遇到这样的案例。图 5 所示的用例图乍看起来没什么问题，销售子系统(sale subsystem) 和财务管理子系统

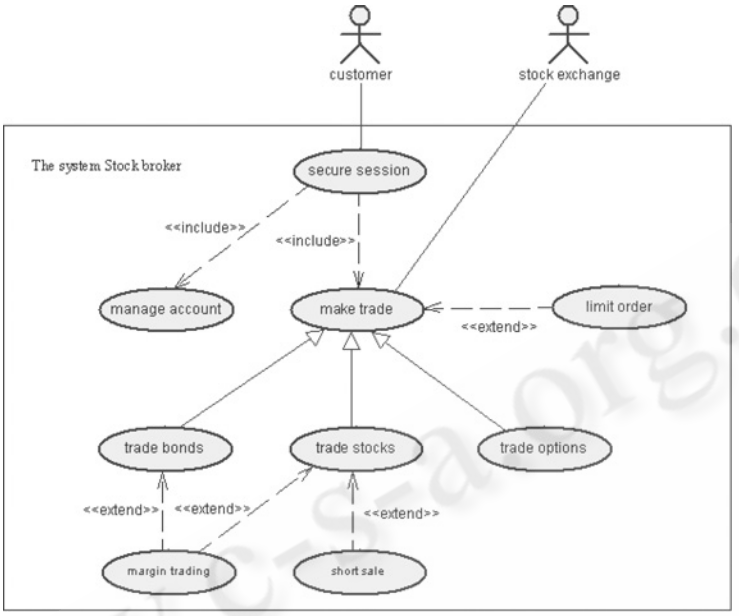


图 4 图 3 所示用例图的修正

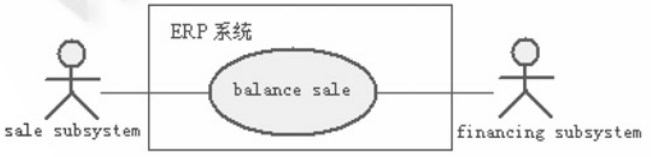


图 5 问题 :子系统定义不明确

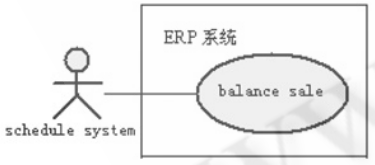


图 6 修正 业务逻辑的子系统不能成为 ERP 系统的外部参与者

(financing subsystem) 应该是两个独立的系统,即 ERP 系统必须通过接口(interface)来呼叫“ 财务管理子系统 ”所提供的 APIs,而不能直接连接该子系统的私有数据库。但实际上,系统的开发范围仍在 ERP 系统的范畴内,并没有界定这些子系统之间是通过接口的信息沟通传递的。明确了子系统的定义,这个用例图就违背了准则 6,即系统本身成了系统的参与者。会触发系统用例的参与者其实是外部系统,例如,“ Timer ”、“ 排程系统(schedule system)”等。同样的,也没有所谓的“ 销售子系统 ”来呼叫 ERP 系统,正如图 6 中所示的,参与者实际是排程系统。

## 4 结论

要顺应短线的项目开发,快速实现用例并马上导出系统的实作是最有效的手段。然后,再加上两个配套的措施,以确保项目的质量 :(1) 以用例为单位,撰写验证功能的测试代码 ;(2) 分析类别,确定实体框架,尽量隔离用户界面(UI)与数据库的变动。画用例时,要由简入繁,逐步增加用例的精度,不要刚开始就陷入繁杂的细节,这样才不至于将过多的精力投入到错误的设计及描述上。不断通过实践和推敲,才能体会到用例的奥妙和精髓。

## 参考文献

1 Michael B, James R. 车皓阳, 杨眉译. UML 面向对象建模与设计( 第二版 )[ M ]. 北京: 人民邮电出版社,2006.

2 James R, Ivar J, Grady B. 姚淑珍, 唐发根译. UML 参考手册[ M ]. 北京: 机械工业出版社,2001.

3 Grady B, James R, Ivar J. 邵维忠, 麻志毅, 马浩海, 刘辉译. UML 用户指南[ M ]. 北京: 人民邮电出版社,2006.