让历史告诉未来

——评 Ivar Jacobson 博士的《用例的昨天、今天和明天》

IT 之源 张恂

zhangxun2001@hotmail.com

2003.4 版本 A

概述

2003 年 3 月，著名的电子杂志《The Rational Edge》在 IBM 出巨资收购 Rational 之后的第一期发表了 UML 之父 Ivar Jacobson 博士撰写的《Use Cases——Yesterday, Today and Tomorrow》。这篇文章依次介绍了用例技术的起源、发展和演进过程，并且对大家在实践中困惑较多的用例关系、用例数量、用例与 UML 等问题作了深刻的阐释，同时提出了对扩展/包含用例的改进意见，最后还对用例未来的发展趋势作出了有趣的预测。

这次由用例的发明人亲自撰文、现身说法讲述用例的历史，实在难得。我怀着如获至宝的心情反复研读了原文。现在把我整理、学习这篇文章的体会记录下来，在介绍 Jacobson 博士观点的同时加入我个人的分析和评述，以飨广大读者。

一、用例的起源

1.1 萌芽期（1967-1986 年）

用例的出现源于 Ivar Jacobson 从 1967 年开始在爱立信公司所从事的近 20 多年对大量不同电话呼叫类型建模的工作。当时他把各种不同类型的电话呼叫情况（电话用户对电话交换系统的需求）称为 traffic case，完成所有呼叫则需要交换机具备各种不同的“功能”（function）或特性（feature）。

1986 年春天，在研究如何把 traffic case 映射到功能的过程中，Jacobson 突然产生了灵感，发明了“use case”这个术语。他发现可以利用一种类似继承的机制通过功能来表达 traffic case。当时他把这两者都叫做用例，而 traffic case 就是后来的具体（concrete）或真实用例，功能则变成了抽象用例。

在提交给 OOPSLA'86 的一篇会议论文中，Jocobson 首次提出了用例的概念，

可惜当时那篇文章由于特殊的原因没有被录取。在随后的更新版本中，Jacobson 又加入了许多用例建模的重要概念，这篇论文终于在 1987 年的 OOPSLA 大会被录取，用例正式诞生了。（迄今已有 15 年了！）

Jacobson 在发表的用例建模理论中提出，“**用例是由用户和系统在一次对话中执行的一个特殊事务序列。**”为此，Jacobson 还开发了一个独立的模型用于从外部视点描述系统，这就是我们熟知的用例模型，它提供了系统的黑盒视图，代表了系统的功能需求。当时的用例概念模型可以被无缝地翻译成一个新模型，它可以展示每个用例是如何通过明确的块（可能是一个子系统、类、或构件）实现的。当时序列图（sequence diagram）就已经被用来描述用例是如何通过软件块/构件之间的交互来实现的。对每个用例进行独立测试可以保证系统满足了用户的需求。可见，用例已经构成了当时整个开发活动的核心。

Jacobson 提到他早在 1969 年就发明了序列图，所以当他说出“这不足为怪，那时序列图展示其在实践中的价值已经差不多有 20 年了”时，实在令人惊讶——这意味着序列图到现在竟然已有 35 年的历史了！

1.2 成熟期（1987-1992 年）

从 1987 年到 1992 年的 5 年间，Jacobson 通过自己创办的公司 Objectory AB 展开了大量的用例实践，约有 20 多家涉及管理信息、国防（导航、测量、C3I）、电信（POTS、移动）等众多领域的公司纷纷把 Jacobson 发明的以用例内容为核心的 Objectory Process（对象工厂过程）应用于新产品开发。

此时，用例引入了“继承”关系。到了 1992 年（10 年前！）用例已经具备了大部分当代的语法和语义，并且已经对用例（也是像类一样的东西，即“类元”）、用例的一个实现和用例的一段描述进行了严格的区分。

当时的用例描述主要包含下列内容：

- 用例目的的简要叙述
- 控制流
- 基本流和可选流
- 子流程（可在同一个用例描述中的多个位置上被重用）
- 前置条件和后置条件

Jacobson 还与同事们创造了“用例驱动式开发”这一术语。整个过程简单说

就是，首先找出所有的用例并在需求文档中对其进行详述，然后对每个用例进行分析、设计，最后对所有的用例进行逐个测试。

在这一阶段，用例不仅仅是一种非常有效的需求技术，它还可以跟踪到分析、设计、实现和测试。对于每个用例，在分析和设计阶段都有一个协作（参与对象的视图）与其对应。每个用例都能产生一个测例（test case）集合。用例对于用户界面设计、编写用户手册也很重要。由于用例能够完美地捕捉业务过程，其应用甚至还扩展到了业务建模领域。

在获得了深入理论研究和充分实践经验的基础上，Ivar Jacobson 与其同事一起于 1992 年出版了《*Object-Oriented Software Engineering: A Use Case Driven Approach*》（Addison Wesley, 1992）一书，奠定了 OOSE 和用例方法的基础，这无疑是一本软件工程史上划时代的里程碑之作。

1.3 发展期（1992 年至今）

1992 年以后用例的发展史大家可能比较熟悉了。1995 年 Jacobson 加入了 Rational 公司，并与另两位 OO 大师 Grady Booch、James Rumbaugh 携手一同领导面向对象建模语言的统一进程，演绎了软件史上的一段佳话。用例和 OOSE 随之融入了 UML，并与 Objectory 过程、Rational 方法结合产生了如今闻名遐迩的瑞理统一过程（RUP）。RUP 的 3 个基本特征就是以用例为驱动，以架构为中心和迭代递增式开发[2]。

用例被广泛接受、迅速采纳的速度令 Jacobson 也感到惊讶：它发表后几乎立即就受到所有方法论学家的欢迎，并在全世界范围内被投入大量应用。这可能源于用例本质上是一个简单、显而易见的想法，它天然地与对象、面向对象思维吻合得很好。

二、用例的今天

在回顾辉煌的历史之后，Jacobson 博士不但对用例应用中一些常见的问题作出了澄清和说明，而且还提出了相应的改进建议。

2.1 如何控制用例的数量

今天 RUP 和 UML 中关于用例的定义主要来源于 1994 年，为了使软件人员准确地把握用例的粒度避免用例数量过多或过少，它强调用例必须为“ 特定的使用者（Actor）”提供“ 可度量的价值 ”。Jacobson 建议：支持单个业务过程的大型

系统所拥有的具体用例的数目通常不应该超过 20 个。

2.2 用例与 UML

如今 RUP、UML、用例已经完美地融合在一起，全球成千上万的客户和专家对其进行了充分的实践。

UML 把用例的“使用”（uses）关系改为一般化（generalization，又译泛化）关系，并且增加了《包含》关系。UML 旧版的使用关系实际上源自继承关系，这一点可能出乎许多读者的意料。

Jacobson 博士还向我们推荐了 2003 年刚出版的新书《用例建模》（Kurt Bittner 和 Ian Spence 著，Addison Wesley），他认为这是“ THE book on use cases ”（用例专著），并强烈推荐每一位软件工程和需求开发人员阅读。

2.3 用例的形式化

Jacobson 博士提醒我们，在使用用例的形式化技术时要小心。毕竟，用例模型是用来与客户和用户交流的。他指出，“ 利用数学对用例进行形式化从来都不是个问题 ”，他本人早在 1986 年就已经这么做了，而且计算机科学系的学生也都可以做到。

到现在，Jacobson 仍然不太情愿建议系统分析员用一种比简单文本更形式化的方式来描述用例。他建议，应避免用活动图或状态图来详述每个用例的内部状况（注：这其实属于分析任务），尽管用序列图或带有泳道的活动图来描述用例与使用者之间的交互是不错的。

2.4 用例关系

Jacobson 博士对在用例关系的应用中出现的问题进行了澄清，并对扩展/包含抽象用例的处理方式提出了改进。

他认为，一个用例模型通常包含四种用例：

- 具体用例（可被实例化，抽象用例不能）
- 一般化用例（支持用例的重用）
- 扩展用例（向已存在或假定存在的用例添加新的行为而不改变它）
- 包含用例（通过增加行为改变其他用例）

2.4.1 一般化

一般化用例都是抽象用例，它们是对其他具体用例或抽象用例的一般化。一

般化用例（父用例）与子用例必须类型相同以保证可替换性。这遵从了 OO 继承的基本原理：由于子与父具有相同的类型，所以可以在使用父实例的任何地方用子实例来代替。

2.4.2 扩展

扩展用例的目的比较特殊，它们的作用是在不改变某个已存在（或假定存在）的用例的前提下为之增添新行为。这些附加的行为可能是必需的，也可能是可选的。

使用扩展的一个潜在问题是创建过深的扩展依赖层次，因此 Jacobson 博士建议永远不要扩展一个扩展（片段）。

对于在描述用例的时候，什么时候用扩展，什么时候用可选路径，Jacobson 建议：只有当扩展用例与被扩展用例完全分离（即它本身是一个独立的具体用例或者是其他用例需要的一个小片段）时，才使用扩展关系。基用例自身必须是完整的，它的正确执行不需要扩展。否则，就应该用可选路径来描述附加行为。

扩展实际上并不仅仅对用例建模有用，它的出现甚至可以追溯到 1978 年，当时 Jacobson 还在爱立信工作，后来他在 OOPSLA'86 的论文中对扩展作了介绍，而直到 1991 年开发人员才接受这一概念。有趣的是，爱立信公司甚至还为此申请了专利，用于对 C++、操作系统、计算机体系结构的扩展。当时的基础设施团队还利用这些方法显著降低了开发成本。

2.4.3 包含

用例有两种重用方式。Jacobson 在 1987 年引入用例的继承关系，并在 1992 年把它的名称改为“ uses ”，也就是后来 UML 的“ 一般化 ”关系。

第 2 种重用方式，即从用例的描述中提取出公共（共享）的事件流，就是现在的《包含》关系。Jacobson 博士多次强调，他很不情愿引入包含这样一种关系，包含关系使用不当容易诱使人们进行攻能分解，从而导致对用例的误用。因此他们过去不允许开发人员对用例公共的片段建模，而是用文本对象（text object）来代替。这是一些包含文本段落的可重用对象，可以被多个位置同时引用，很容易保证同步。

Jacobson 说，“ 事实上，今天一些人误用了用例，把它们用来描述功能（注：指功能分解式的分析）而不是对象，反过来又指责用例概念存在问题 ”，这真是

令人哭笑不得。前些年我在网上也看到不少贴子说用例不是 OO 的，甚至还有国外某些“ 专家 ”文章佐证，这令我很纳闷。我们怎么不动动脑子呢？让直觉说话吧——用例明明出自 OO 大师之手，怎么反倒变成不是 OO 的了？

2.4.4 扩展与包含

扩展和包含用例本质上其实非常相似，它们的主要区别在于用例实例中断基用例、执行附加用例的方式。

扩展和包含用例都于基用例相联。在基用例的执行过程中，可能在某种条件下基用例的执行流被中断，转而执行扩展或包含用例（在 UML 中统称为附加用例）的流。当附加用例流执行完毕，控制将返回到基用流原来被中断的那个位置恢复执行。

扩展用例通过引用扩展点（extension point）建立与基用例的联系，扩展点指明了在基用例中的扩展位置。以前在 OOSE 和 Objectory Process 中，扩展点属于扩展用例。在制定 UML1.1 时，James Rumbaugh 建议扩展点应该属于被扩展用例。这是出于封装的考虑：扩展用例不应看到基用例的细节——它只能看见扩展点；因为如果扩展点改变了，只有被扩展用例应该知道新的位置。

2.5 两类扩展/包含用例

1）作为具体用例的扩展/包含

具体用例既可以扩展基用例，也可以被基用例包含。这些用例可以与使用者交互、实例化并提供价值。

2）作为用例片段的扩展/包含

这种类型的用例数量较多，但一般每个用例都很小。这些用例是抽象的，也就是说不能被独立地实例化，它们通常是其他具体或一般化用例所需要的。

Jacobson 博士专门举了一个银行系统的例子（如图 1）。在这张图中有两个具体用例：“ 执行事务 ”和“ 检查事务失败 ”（管理用例）。每当一个事务失败时，银行要记录这一事件以便其他用例（如“ 检查事务失败 ”）能够获得这一信息。

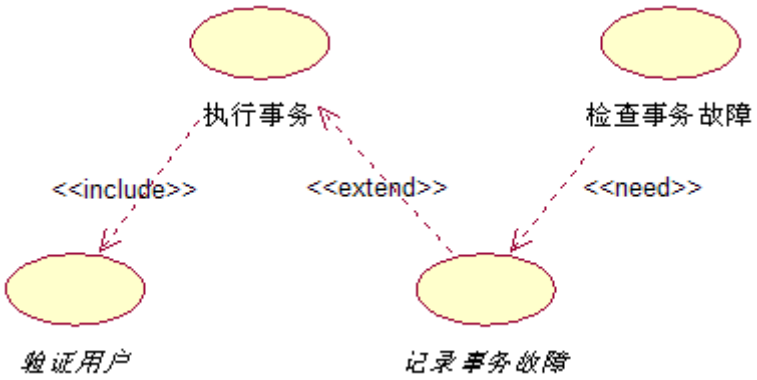

图 1、具体用例与抽象用例（片段）

“ 记录事务失败 ”与执行事务本身无关，它只是为了登记其他用例需要的信息，所以它是基用例“ 执行事务 ”的扩展。“ 验证用户 ”是一个被包含的用例片段。“ 验证用户 ”和“ 记录事务失败 ”都是抽象的（故用例名称用斜体字表示），他们不能直接被使用者实例化。

值得注意的是，Jacobson 博士强调，用例片段（抽象用例）——无论扩展片段还是包含片段——都不是“ 真正的 ”用例。没有对真正的用例和不那么重要的用例片段作出明确的区分，实际上是过去的一个小缺陷，很容易在使用中造成混淆。于是他提出通过补充少量图形元素对 UML 改进的想法。

既然片段不是用例，那么就不应该用用例的语法来表示它。在 UML 中，片段应该是一种带有图标的类元，能提示我们它是一种细小的东西。为此 Jacobson 向我们推荐了一种可能的新表示法，采用“ 圆点 ”或类似的小图标来表示扩展/包含片段。

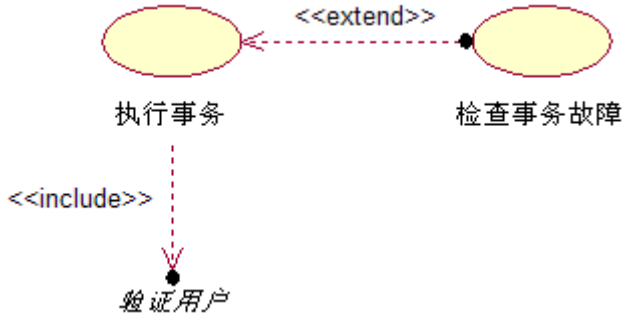

图 2、片段表示法

在图 2 中，“ 记录事务故障 ”缩成了一个匿名的圆点（表示扩展片段），并贴附在具体用例“ 检查事务故障 ”上面，通过《扩展》关系指向“ 执行事务 ”。这

表示：当“ 执行事务 ”用例激活扩展点时，由小圆点代表的扩展将开始执行。“ 检查事务故障 ” 用例将利用（需要）“ 记录事务故障 ” 这一扩展来完成其任务，而后者并不是前者的一部分。注意，我们不能直接让“ 检查事务故障 ” 扩展“ 执行事务 ”，这在语义上是不通的。所以，需要用在具体用例“ 检查事务故障 ” 上附加扩展片段的方法来处理。

在图 2 中“ 验证用户 ” 是一个包含片段，它可以是可重用的用例描述或文本对象，并与包含它的用例“ 执行事务 ” 分离。如果有多个用例“ 需要 ” 某个扩展片段，那么也可以把它独立地表示成一个类元。所以在这点上，扩展/包含片段在语义上是相似的。

当然，扩展/包含片段也存在重要的区别：一个包含片段（如“ 验证身份 ”）将由执行真实用例（即包含它的用例，如“ 执行事务 ”）的用例实例来执行它；而一个扩展片段（如“ 记录事务故障 ”）将由需要它的用例实例（如“ 检查事务故障 ”）之外的用例（如“ 执行事务 ”）来执行。

三、用例的未来

在文章的最后部份，Jacobson 博士向我们介绍了用例技术的发展趋势：

1) 可以通过 UML 的构造型机制对不同的用例类型进行细分以方便开发人员使用，比如添加管理用例、运营用例、辅助用例等等。
2) 进一步澄清模式与用例的关系。在模式与描述特定问题的一般化的、可重用的用例之间存在一种有意思的联系。许多设计模式实际上是实现可重用用例的“ 模板 ”。
3) 在人机交互（HCI）领域中运用用例。HCI 是一门科学，用例能够在集成软件开发与 HCI 的实践中扮演个更加重要的角色。
4) 利用用例进行成本估算。早在 1994 年 Magus Christerson 就在 Gustav Karner 硕士论文[1]的基础上提出了基于用例点的项目估算方法，这些经验如今仍具有指导意义。
5) 开始重用用例。这是一个很大的话题。**重用业务软件应该从理解用例开始**——既包括业务用例，也包括软件用例。1997 年 Jacobson 与 Martin Griss、Patrik Jonsson 发表的专著《*Software Reuse：Architecture, Process and Organization for Business Success*》（Addison Wesley Longman）对此专

门作了深入地探讨。

6) 让用例情节（scenario）成为一等“公民”。识别、列举用例情节并说明它们之间的依赖是有帮助的。比方说，可以根据情节来制定、跟踪项目迭代计划，建立用例情节到测例的跟踪链。

7) 建立用例与 Aspect 的密切联系。Jacobson 博士认为 AOP（Aspect-Oriented Programming）是当今最激动人心的技术新进展之一，他深信 AOP 将为用例带来戏剧性的变化。实际上，扩展用例的目的——在不改变已有系统的情况下添加新的行为——与 aspect 非常相似。用例实现可以实现为 aspect，扩展用例也可以实现为 aspect，扩展点则与 AOP 中的联结点语义上近似。有了 AOP，我们就可以从用例设计直接过渡到用例编程，再到用例测试，显著减少构件方面的工作。事实上，用例就像横切构件的模块，在这些模块上的工作将由新型编程环境（AOP 语言=OOP 语言+aspects）来支持。AOP 使得我们可以无缝地实现用例，两者的结合将极大地改进如今软件开发的方式。

8) 使用例成为运行时实体。最令人兴奋的用例展望是——将来用例在运行时环境中也有对应物。能够在操作系统中识别执行用例（实例）将带来很多重要的特性，比如可以更加增量化的方式改变已安装的软件（每次 1 个用例实例），用更小的步骤（终断的用例实例）重启软件系统。

四、体会和结语

读完全文，我有以下几点体会和看法与大家分享：

1）我们怎么强调用例的重要性可能都不为过。老式的结构化功能分析其实并不能满足当今复杂商业软件开发的需要，用例分析不同于功能分解，也就是说很多情况下简单的功能描述是不够的，**我们必须规范、确切地给出功能“使用的例子”，这才是真正的功能需求**。用例桥接客户需要和软件功能的特殊地位和作用使得它不仅仅能用于需求的捕捉和提炼，而且还构成了系统分析、设计、测试乃至整个项目管理的基础。**事实上用例已经成为当代软件开发活动的轴心**。

2）我们知道 Alistair Cockburn 是用例另一支流派的代表人物，他在基于用户目标的用例需求分析方面开展了大量实践，并且也获得了全球范围内开发者的一致好评和认同。可是我发现 Rational 公司和 Jacobson 博士本人有关用例的大量文献很少提到 Cockburn 的名字，这一现象让我感到有些奇怪，我也隐约地

感觉到 Jacobson 博士和 Cockburn 在对用例、UML 的看法上存在一些分歧（虽然并无本质上的不同）。事实上，我们不难发现，Cockburn 与 Rational、UML 的用例方法在一些细节上存在着明显的差别。作为一名中国的用例和 UML 实践者，我认为面临这个现实问题，我们有必要在具体实践中将 Cockburn、Jacobson 以及 UML 标准的方法巧妙地结合起来，汲取二者的精华，从中提炼出统一而实用的用例方法。

3）Jacobson 的文章再次说明了一个道理。坦率地讲，用例、UML、OOAD 和 RUP 基本上都不能算什么“新技术”，它们至少都有 10 年以上的发展史。我们总是不断听到业内人士评论、抱怨软件技术发展太快，新技术层出不穷，跟也跟不上，从而得出推论：既然它们是新技术，那么肯定都还不成熟，我们不如观望之或干脆拒绝之，还是用老方法保险。事实到底如何呢？我怀疑还有一种极大的可能性——大量所谓的软件“新”技术其实并不新，**不过是“新”近传进来或者我们刚刚知道罢了**，人家早已经用了 5 年、10 年甚至更长时间了；况且很多看似时髦的新技术背后其实都有着非常深厚的技术渊源和理论基础，我们看到、听到的“新技术”不过是些肤浅的表象。长期的技术传播落后、信息交流不畅和安于现状、自欺欺人的心态阻碍了我们虚心踏实、坚持不懈地学习基础知识，知己知彼、与时俱进，赶上国外先进的软件技术发展水平，与国际接轨。现在我们其实是在“补课”，需要补 CMM 的课（CMM 差不多也是别人 10 多年前的成果），也需要补 UML、用例的课。所以，大家应该协同努力尽早改变这一不合理的现象。

总之，Jacobson 博士的《用例的昨天、今天和明天》无疑是一篇宝贵的学习资料，它带给我们很多的启迪和感悟。它能使我们对用例相关概念的来龙去脉有更为清晰、深刻和完整地了解，便于我们在今后的工作中更好地掌握和应用这一先进技术。相信大家在读了之后也会有相同的感受。

参考文献

[1]《用例分析技术》（第 2 版），G. 施奈德等著，姚淑珍等译，机械工业出版社，2002.8

[2]《统一软件开发过程》，I. 亚各申等著，周伯生等译，机械工业出版社，2002.1

张恂：东南大学工学硕士（软件），OO 软件工程传道者，软件咨询顾问。长期从事软件工程与面向对象技术的应用开发、管理、咨询和推广工作，具有 8 年软件开发的丰富经验和扎实

的理论基础。曾先后担任国内著名通信企业大型移动系统研发的软件项目经理、某民营软件公司 CTO。在国内通信软件界较早地引入了 UML、模式、统一过程、配置管理等先进的软件工程理念和方法。2002 年 9 月发表国内第 1 本软件组织模式译著《软件架构：组织原则与模式》。研究兴趣包括 OOAD*UML、需求工程、软件架构、软件模式、敏捷（统一）过程、软件项目管理、质量保证等等。在 IT 之源开设了个人专栏，愿与各位同仁切磋交流，共同进步！