UML 基础: 序列图

本文来自于 Rational Edge本文作为 UML 基础的、关于统一建模语言的基础图的一系列文章的一部分，提供对序列图（Sequence Diagram，又称为 顺序图）的详细介绍。它也介绍了最近的 UML 2.0 规范的几个新符号元件。

Donald Bell, IBM 全球服务, EMC

2005 年 2 月 15 日

内容

现在是二月，而且到如今你或许已经读到、或听到人们谈论UML 2.0 —— 包括若干进步的 UML 的新规范，所做的变化。考虑到新规范的重要性，我们也正在修改这个文章系列的基础，把我们的注意力从 OMG 的 UML 1.4 规范，转移到 OMG 的已采纳 UML 2.0草案规范（又名 UML 2）。我不喜欢在一系列文章的中间，把重点从 1.4 变为 2.0 ，但是 UML 2.0 草案规范是前进的重要一步，我感觉需要扩充文字。由于一些理由，OMG 改良了 UML 。主要的理由是，他们希望 UML 模型能够表达模型驱动架构（MDA），这意味着 UML 必须支持更多的模型驱动的符号。同时， UML 1.x 符号集合有时难以适用于较大的应用程序。此外,为了要使图变成更容易阅读，需要改良符号元件。（举例来说，UML 1.x 的模型逻辑流程太复杂，有时不可能完成。对UML 2 中的序列图的符号集合的改变，已经在序列化逻辑建模方面取得巨大的进步)。注意我上面所述的文字："已采纳UML2.0草案规范。"确实，规范仍然处于草案状态，但是关键是草案规范已经被 OMG 采用，OMG是一个直到新标准相当可靠，才会采用它们的组织。在 UML 2 完全地被采用之前，规范将会有一些修改，但是这些改变应该是极小的。主要的改变将会是在 UML 的内部 —— 包括通常被实施 UML 工具的软件公司使用的功能。本文的主要目的是继续把我们的重点放在基础UML图上；这个月，我们进一步了解序列图。再次请注意，下面提供的例子正是以新的 UML 2 规范为基础。

序列图主要用于按照交互发生的一系列顺序，显示对象之间的这些交互。很象类图，开发者一般认为序列图只对他们有意义。然而，一个组织的业务人员会发现，序列图显示不同的业务对象如何交互，对于交流当前业务如何进行很有用。除记录组织的当前事件外，一个业务级的序列图能被当作一个需求文件使用，为实现一个未来系统传递需求。在项目的需求阶段，分析师能通过提供一个更加正式层次的表达，把用例带入下一层次。那种情况下，用例常常被细化为一个或者更多的序列图。

组织的技术人员能发现，序列图在记录一个未来系统的行为应该如何表现中，非常有用。在设计阶段，架构师和开发者能使用图，挖掘出系统对象间的交互，这样充实整个系统设计。

序列图的主要用途之一，是把用例表达的需求，转化为进一步、更加正式层次的精细表达。用例常常被细化为一个或者更多的序列图。序列图除了在设计新系统方面的用途外，它们还能用来记录一个存在系统（称它为"遗产"）的对象现在如何交互。当把这个系统移交给另一个人或组织时，这个文档很有用。

符号

既然这是我基于 UML 2的 UML 图系列文章的第一篇，我们需要首先讨论对 UML 2 图符号的一个补充，即一个叫做框架的符号元件。在 UML 2中，框架元件用于作为许多其他的图元件的一个基础，但是大多数人第一次接触框架元件的情况，是作为图的图形化边界。当为图提供图形化边界时，一个框架元件为图的标签提供一致的位置。在 UML 图中框架元件是可选择的；就如你能在图 1 和 2 中见到的，图的标签被放在左上角，在我将调用框架的“namebox”中，一种卷角长方形，而且实际的 UML 图在较大的封闭长方形内部定义。

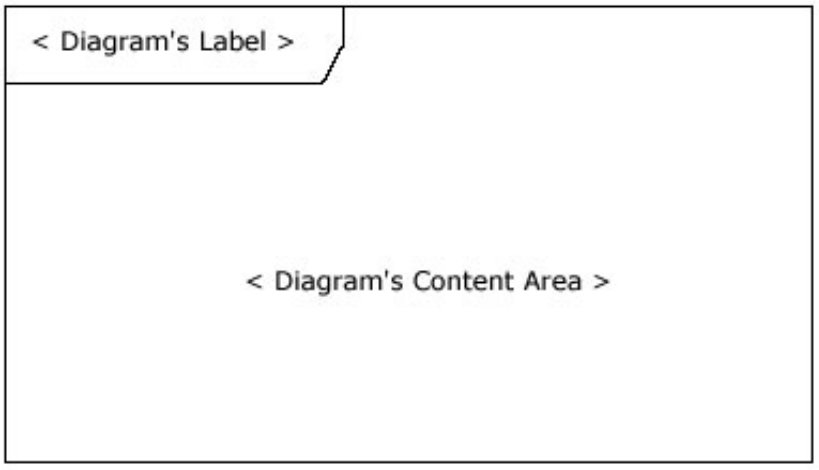

图 1: 空的 UML 2 框架元件

除了提供一个图形化边框之外，用于图中的框架元件也有描述交互的重要的功能, 例如序列图。在序列图上一个序列接收和发送消息（又称交互），能通过连接消息和框架元件边界，建立模型（如图 2 所见到）。这将会在后面“超越基础”的段落中被更详细地介绍。

图 2: 一个接收和发送消息的序列图

注意在图 2 中，对于序列图，图的标签由文字“sd”开始。当使用一个框架元件封闭一个图时，图的标签需要按照以下的格式:

> 图类型 图名称

UML 规范给图类型提供特定的文本值。（举例来说，sd代表序列图，activity代表活动图，use case代表用例图）。

基础

序列图的主要目的是定义事件序列，产生一些希望的输出。重点不是消息本身，而是消息产生的顺序；不过，大多数序列图会表示一个系统的对象之间传递的什么消息，以及它们发生的顺序。图按照水平和垂直的维度传递信息：垂直维度从上而下表示消息/调用发生的时间序列，而且水平维度从左到右表示消息发送到的对象实例。

生命线

当画一个序列图的时候，放置生命线符号元件，横跨图的顶部。生命线表示序列中，建模的角色或对象实例。[1] 生命线画作一个方格，一条虚线从上而下，通过底部边界的中心（图 3）。生命线名字放置在方

格里。

图 3: 用于一个实体名为freshman的生命线的Student类的一个例子

UML 的生命线命名标准按照如下格式:

> 实体名：类名

在如图3所示的例子中，生命线表示类Student的实体，它的实体名称是freshman。这里注意一点，生命线名称带下划线。当使用下划线时，意味着序列图中的生命线代表一个类的特定实体，不是特定种类的实体（例如，角色）。在将来的一篇文章中，我们将会了解结构化建模。现在，仅仅评述序列图，可能包含角色（例如*买方*和*卖方*），而不需要叙述谁扮演那些角色（例如**Bill**和**Fred**）。这准许不同语境的图重复使用。简单拖放，序列图的实例名称有下划线，而角色名称没有。

图 3 中我们生命线例子是一个命名的对象，但是不是所有的生命线都代表命名的对象。相反的，一个生命线能用来表现一个匿名的或未命名的实体。当在一个序列图上，为一个未命名的实例建模时，生命线的名字采用和一个命名实例相同的模式；但是生命线名字的位置留下空白，而不是提供一个例图名字。再次参考图 3，如果生命线正在表现Student类的一个匿名例图，生命线会是: “Student”。同时, 因为序列图在项目设计阶段中使用，有一个未指定的对象是完全合法: 举例来说，“freshman”。

消息

为了可读性，序列图的第一个消息总是从顶端开始，并且一般位于图的左边。然后继发的消息加入图中，稍微比前面的消息低些。

为了显示一个对象（例如，生命线）传递一个消息给另外一个对象，你画一条线指向接收对象，包括一个实心箭头（如果是一个同步调用操作）或一个棍形箭头（如果是一个异步讯号）。消息/方法名字放置在带箭头的线上面。正在被传递给接收对象的消息，表示接收对象的类实现的一个操作/方法。在图 4 的例子中，analyst对象调用ReportingSystem 类的一个实例的系统对象。analyst对象在调用系统对象的 getAvailableReports 方法。系统对象然后调用secSystem 对象上的、包括参数userId的 getSecurityClearance 方法，secSystem的类的类型是 SecuritySystem。[2]

图 4: 一个在对象之间传递消息的实例

除了仅仅显示序列图上的消息调用外，图 4 中的图还包括返回消息。这些返回消息是可选择的；一个返回消息画作一个带开放箭头的虚线，向后指向来源的生命线，在这条虚线上面，你放置操作的返回值。在图 4 中，当 getSecurityClearance 方法被调用时，secSystem 对象返回 userClearance 给系统对象。当 getAvailableReports 方法被调用时，系统对象返回 availableReports。

此外，返回消息是序列图的一个可选择部分。返回消息的使用依赖建模的具体/抽象程度。如果需要较好的具体化，返回消息是有用的；否则，主动消息就足够了。我个人喜欢，无论什么时候返回一个值，都包括一个返回消息，因为我发现额外的细节使一个序列图变得更容易阅读。

当序列图建模时，有时候，一个对象将会需要传递一个消息给它本身。一个对象何时称它本身？一个纯化论者会争辩一个对象应该永不传递一个消息给它本身。然而，为传递一个消息给它本身的对象建模，在一些情境中可能是有用的。举例来说，图 5 是图 4 的一个改良版本。 图 5 版本显示调用它的

determineAvailableReports 方法的系统对象。通过表示系统传递消息“determineAvailableReports”给它本身，模型把注意力集中到过程的事实上，而不是系统对象。

为了要画一个调用本身的对象，如你平时所作的，画一条消息，但是不是连接它到另外的一个对象，而是你把消息连接回对象本身。

图 5: 系统对象调用它的 determineAvailableReports 方法

图 5 中的消息实例显示同步消息；然而，在序列图中，你也能为异步消息建模。一个异步消息和一个同步的画法类似，但是消息画的线带一个棍形矛头，如图 6 所示。

图 6: 表示传递到实体2的异步消息的序列图片段

约束

当为对象的交互建模时，有时候，必须满足一个条件，消息才会传递给对象。约束在 UML 图各处中，用于控制流。在这里，我将会讨论UML 1.x 及UML 2.0两者的约束。在 UML 1.x 中，一个约束只可能被分配到一个单一消息。UML 1.x中，为了在一个序列图上画一个约束，你把约束元件放在约束的消息线上，消息名字之前。图 7 显示序列图的一个片段，消息addStudent 方法上有一个约束。

图 7:UML 1.x 序列图的一个片段，其中addStudent 消息有一个约束

在图 7 中，约束是文本“[pastDueBalance=0]”。通过这个消息上的约束，如果应收帐系统返回一个零点的逾期平衡，addStudent 消息才将会被传递。约束的符号很简单；格式是:

> [Boolean Test]

举例来说，

> [pastDueBalance = 0]

组合碎片(变体方案，选择项，和循环)

然而，在大多数的序列图中，UML 1.x“in-line”约束不足以处理一个建模序列的必需逻辑。这个功能缺失是 UML 1.x 的一个问题。UML 2 已经通过去掉“in-line”约束，增加一个叫做组合碎片的符号元件，解决了这一个问题。一个组合碎片用来把一套消息组合在一起，在一个序列图中显示条件分支。UML 2 规范指明了组合碎片的 11 种交互类型。十一种中的三种将会在“基础”段落中介绍，另外两种类型将会在“超越基础”中介绍，而那剩余的六种我将会留在另一篇文章中介绍。（嗨，这是一篇文章而不是一本书。我希望你在一天中看完这部分！）

变体

变体用来指明在两个或更多的消息序列之间的、互斥的选择。[3] 变体支持经典的“if then else”逻辑的建模（举例来说，**如果** 我买三个，**然后** 我得到 我购买的20% 折扣；**否则** 我得到我购买的 10% 折扣）。

就如你将会在图 8 中注意到的，一个变体的组合碎片元件使用框架来画。单词“alt”放置在框架的namebox里。然后较大的长方形分为 UML 2 所称的操作元。[4] 操作元被虚线分开。每个操作元有一个约

束进行测试，而这个约束被放置在生命线顶端的操作元的左上部。[5]如果操作元的约束等于“true”，然后那个操作元是要执行的操作元。

图 8：包含变体组合碎片的一个序列图片段

图 8作为一个变体的组合碎片如何阅读的例子，显示序列从顶部开始，即bank对象获取支票金额和帐户结余。此时，序列图中的变体组合碎片接管。因为约束“[balance >= amount]”，如果余额超过或等于金额，然后顺序进行bank对象传递 addDebitTransaction 和 storePhotoOfCheck 消息给account对象。然而，如果余额不是超过或等于金额，然后顺序的过程就是bank传递addInsuffientFundFee 和 noteReturnedCheck 消息给account对象，returnCheck 消息给它自身。因为“else”约束，当余额不大于或者等于金额时，第二个序列被调用。在变体的组合碎片中，不需要“else”约束；而如果一个操作元，在它上面没有一个明确的约束，那么将假定“else”约束。

变体的组合碎片没被限制在简单的“if then else”验证。可能需要大量的变体路径。 如果需要较多的变体方案，你一定要做的全部工作就是把一个操作元加入有序列约束和消息的长方形中。

选择项

选择项组合碎片用来为序列建模，这些序列给予一个特定条件，将会发生的；或者，序列不发生。一个选择项用来为简单的“if then”表达式建模。（例如，如果架上的圈饼少于五个，那么另外做两打圈饼）。

选择项组合碎片符号与变体组合碎片类似，除了它只有一个操作元并且永不能有“else”约束以外（它就是如此，没有理由）。要画选择项组合，你画一个框架。文字“opt”是被放置在框架的 namebox 里的文本，在框架的内容区，选择项的约束被放置在生命线顶端上的左上角。 然后选择项的消息序列被放在框架的内容区的其余位置内。这些元件如图 9 所示。

图 9：包括选择项组合碎片的一个序列图片段

阅读选择项组合碎片很容易。图 9 是图 7 的序列图片段的再加工，但是这次它使用一个选择项组合碎片，因为如果Student的逾期平衡等于0，需要传递更多的消息。按照图 9 的序列图，如果Student的逾期平衡等于零，然后传递addStudent，getCostOfClass和chargeForClass消息。如果Student的逾期平衡不等于零，那么在选择项组合碎片中，序列不传递任何一个消息。

例子图 9的序列图片段包括一个选择项约束；然而，约束不是一个必需的元件。在高层次、抽象的序列图中，你可能不想叙述选择项的条件。你可能只是想要指出片段是可选择的。

循环

有时候你将会需要为一个重复的序列建模。在 UML 2 中，为一个重复的序列建模已经改良，附加了循环组合碎片。

循环组合碎片表面非常类似选择项组合碎片。你画一个框架，在框架的 namebox 中放置文本“loop”。在框架的内容区中，一个生命线的顶部，循环约束[6]被放置在左上角。然后循环的消息序列被放在框架内容区的其余部分中。在一个循环中，除了标准的布尔测试外，一个约束能测试二个特定的条件式。特定的约束条件式是写作“minint = [the number]”（例如，“minint = 1”）的最小循环次数，和写作“maxint = [the number]”（例如，“maxint = 5”）的最大循环次数。通过最小循环检验，循环必须运行至少指定次

数，而循环执行次数不能达到约束指定的最大循环次数。

在图 10 中显示的循环运行，直到 reportsEnu 对象的 hasAnotherReport 消息返回false。如果循环序列应该运行，这个序列图的循环使用一个布尔测试确认。为了阅读这个图，你和平常一样，从顶部开始。当你到达循环组合碎片，做一个测试，看看值 hasAnotherReport 是否等于true。如果 hasAnotherReport 值等于true，于是序列进入循环片断。然后你能和正常情况一样，在序列图中跟踪循环的消息。

超越基础
我已经介绍了序列图的基础，应该使你可以为将会在系统中通常发生的大部份交互建模。下面段落将会介绍用于序列图的比较高阶的符号元件。

引用另外一个序列图

当做序列图的时候，开发者爱在他们的序列图中，重用存在的序列图。[7]在 UML 2 中开始，引进“交互进行”元件。追加交互进行的可以说是 UML 2 交互建模中的最重要的创新。交互进行增加了功能，把原始的序列图组织成为复杂的序列图。由于这些，你能组合（重用）较简单的序列，生成比较复杂的序列。这意味你能把完整的、可能比较复杂的序列，抽象为一个单一的概念单位。

一个交互进行元件使用一个框架绘制。文字“ref”放置在框架的 namebox 中，引用的序列图名字放置在框架的内容区里，连同序列图的任何参数一起。引用序列图的名字符号如下模式:

> `序列图名[(参数)][:返回值]`

两个例子:

> 1. `Retrieve Borrower Credit Report(ssn) : borrowerCreditReport`

或者

> 2. `Process Credit Card(name, number, expirationDate, amount : 100)`

在例子 1 中，语法调用叫做Retrieve Borrower Credit Report的序列图，传递给它参数 ssn。序列 Retreive Borrower Credit Report返回变量 borrowerCreditReport 。

在实例 2 中，语法调用叫做Process Credit Card的序列图，传递给它参数name，number，expiration date，和 amount。然而，在例子 2 中，amount参数将会是值100。因为例子2没有返回值标签，序列不返回值（假设，建模的序列不需要返回值）。

图 11: 一个引用两个不同序列图的序列图

图 11 显示一个序列图，它引用了序列图“Balance Lookup”和“Debit Account”。序列从左上角开始，客户传递一个消息给teller对象。teller对象传递一个消息给 theirBank 对象。那时，调用Balance Lookup序列图，而 accountNumber作为一个参数传递。Balance Lookup序列图返回balance变量。然后检验选择项组合碎片的约束条件，确认余额大于金额变量。在余额比金额更大的情况下，调用Debit Account序列图，给它传递参数accountNumber 和amount。在那个序列完成后，withdrawCash 消息为客户返回

cash。

重要的是，注意在图 11 中，theirBank 的生命线被交互进行Balance Lookup隐藏了。因为交互进行隐藏生命线，意味着theirBank 生命线在“Balance Lookup”序列图中被引用。除了隐藏交互进行的生命线之外，UML 2 也指明，生命线在它自己的“Balance Lookup”序列中，一定有相同的 theirBank 。

有时候，你为一个序列图建模，其中交互进行会重叠*没有* 在交互进行中引用的生命线。在那种情况下，生命线和正常的生命线一样显示，不会被重叠的交互进行隐藏。

在图 11 中，序列引用“Balance Lookup”序列图。“Balance Lookup”序列图在图 12 中显示。因为例子序列有参数和一个返回值，它的标签 —— 位于图的 namebox 中 —— 按照一个特定模式:

> `图类型 图名 [ 参数类型:参数名]`

> `[ : 返回值类型]`

两个例子:

> 1. `SD Balance Lookup( Integer : accountNumber) : Real`

或

> 2. `SD Available Reports( Financial Analyst : analyst) : Reports`

图 12 举例说明例子 1，在里面，Balance Lookup序列把参数 accountNumber 作为序列中的变量使用，序列图显示返回的Real对象。在类似这种情况下，返回的对象采用序列图实体名。

图 12: 一个使用 accountNumber 参数并返回一个Real对象的序列图

图 13 举例说明例子 2，在里面，一个序列图获取一个参数，返回一个对象。然而，在图 13 中参数在序列的交互中使用。

门

前面的段落展示如何通过参数和返回值传递信息，引用另一个序列图。然而，有另一个方法在序列图之间传递消息。门可能是一个容易的方法，为在序列图和它的上下文之间的传递消息建模。一个门只是一个消息，图形表示为一端连接序列图的框架边缘，另一端连接到生命线。使用门的图 11 和 12 ，在图 14 和 15 中可以被看到重构。图 15 的例图有一个叫做getBalance的入口门，获取参数 accountNumber。因为是箭头的线连接到图的框架，而箭头连接到生命线，所以 getBalance 消息是一个入口门。序列图也有一个出口门，返回balance变量。出口门同理可知，因为它是一个返回消息，连接从一个生命线到图的框架，箭头连接框架。

图 14: 图 11 的重构，这次使用门

图 15: 图 12 的重构，这次使用门

组合碎片（跳转和并行）

在本文前面“基础”的段落中呈现的，我介绍了“变体”，“选择项”，和“循环”的组合碎片。这些三个组合碎片是大多数人将会使用最多的。然而，有二个其他的组合碎片，大量共享的人将会发现有用——跳转和并行。

跳转

跳转组合碎片几乎在每个方面都和选择项组合碎片一致，除了两个例外。首先，跳转的框架namebox的文本“break”代替了“option”。其次, 当一个跳转组合碎片的消息运行时，封闭的交互作用的其他消息将不会执行，因为序列打破了封闭的交互。这样，跳转组合碎片非常象 C++ 或 Java 的编程语言中的break关键字。

图 16: 来自图 8 的序列图片段的重构，片段使用跳转代替变体

跳转最常用来做模型异常处理。图 16 是图 8 的重构，但是这次图16使用跳转组合碎片，因为它把balance < amount的情况作为一个异常对待，而不是一个变体流。要阅读图 16，你从序列的左上角开始，向下读。当序列到达返回值“balance”的时候，它检查看看是否余额比金额更少。如果余额不少于金额，被传递的下一个消息是 addDebitTransaction 消息，而且序列正常继续。然而，在余额比金额更少的情况下，然后序列进入跳转组合碎片，它的消息被传递。一旦跳转组合的消息的已经被传递，序列不发送任何其它消息就退出（举例来说，addDebitTransaction）。

注意有关跳转的一件重要的事是，它们只引起一个封闭交互的序列退出，不必完成图中描述的序列。在这种情况下，跳转组合是变体或者循环的一部分，然后只是变体或循环被退出。

并行

今天的现代计算机系统在复杂性和有时执行并发任务方面不断进步。当完成一个复杂任务需要的处理时间比希望的长的时候，一些系统采用并行处理进程的各部分。当创造一个序列图，显示并行处理活动的时候，需要使用并行组合碎片元件。

并行组合碎片使用一个框架来画，你把文本“par”放在框架的 namebox 中。然后你把框架的内容段用虚线分为水平操作元。框架的每个操作元表示一个在并行运行的线程。

图 17: oven 是并行做两个任务的对象实例

图 17 可能没有举例说明做并行活动的对象的最好的计算机系统实例，不过提供了一个容易理解的并行活动序列的例子。序列如这样进行：hungryPerson 传递 cookFood 消息给oven 对象。当oven 对象接收那个消息时，它同时发送两个消息（nukeFood 和 rotateFood）给它本身。这些消息都处理后，hungryPerson 对象从oven 对象返回 yummyFood 。

总结

序列图是一个用来记录系统需求，和整理系统设计的好图。序列图是如此好用的理由是，因为它按照交互发生的时间顺序，显示了系统中对象间的交互逻辑。

参考

- UML 2.0 Superstructure Final Adopted Specification (第8章部分) http://www.omg.org/cgi-bin/doc?ptc/2003-08-02
- UML 2 Sequence Diagram Overview http://www.agilemodeling.com/artifacts/sequenceDiagram.htm
- UML 2 Tutorial http://www.omg.org/news/meetings/workshops/UML%202003%20Manual/Tutorial7-Hogg.pdf

脚注

1 在完全建模系统中，对象（类的实例）也将会在系统的类图中建模。

2 当阅读这个序列图时，假定分析师登录进入系统之内。

3 请注意，附着在不同的变体操作元上的、两个或更多的约束条件式的确可能同时是真，但是实际最多只有一个操作元将会在运行时发生（那种情况下变体的“wins”没有按照 UML 标准定义)。

4 虽然操作元看起来非常象公路上的小路，但是我特别不叫它们小路。泳道是在活动图上使用的 UML 符号。请参考*The Rational Edge* 早期关于 活动图 的文章。

5 通常，附上约束的生命线是拥有包含在约束表达式中的变量的生命线。

6 关于选择项组合碎片，循环组合碎片不需要在它上放置一个约束条件。

7 可能重用任何类型的序列图（举例来说，程序或业务）。我只是发现开发者更喜欢按功能分解他们的图。