第 5 章　顺序图与协作图

本章学习目标

(1) 理解交互模型(顺序图、协作图)的基本概念。

(2) 掌握识别对象类、消息的方法。

(3) 掌握顺序图中各个消息发送的先后次序的分析。

(4) 掌握协作图中对象之间带消息标识链的连接的分析。

(5) 熟悉 UML 中交互模型建模的过程。

(6) 了解 UML 中交互建模的注意事项。

本章首先向读者介绍对象类、生命线、控制焦点、消息、链、多对象、主动对象的概念;然后重点介绍消息的语法格式,以及如何根据用例描述识别消息及消息之间的发送先后次序关系;再介绍 UML 的交互建模步骤;最后通过一个具体案例,重点介绍交互建模的过程。

5.1　交互模型概述

所有系统均可表示为两个方面:静态结构和动态行为。在 UML 中,使用类图描述系统的静态结构,说明系统包含哪些对象类以及它们之间的关系。使用顺序图、协作图、状态图、活动图描述系统的动态行为,说明系统中的各个对象是如何交互协作来实现系统的功能的。

软件系统中的任务是通过对象之间的合作来完成的。对象之间的合作是通过对象之间消息的传递实现的。对象之间的合作在 UML 中被称为交互,即交互是一组对象之间为完成某一任务(如完成一个操作)而进行的一系列信息交换的行为说明。交互可以对软件系统为实现某一任务而必须实施的动态行为进行建模。

交互模型(interaction modeling)是用来描述对象之间以及对象与参与者(actor)之间的动态协作关系以及协作过程中行为次序的图形文档。它通常用来描述一个用例的行为,显示该用例中所涉及的对象和这些对象之间的消息传递情况。因此,可以使用交互模型对用例图中控制流建模,用它们来描述用例图的行为。

交互模型包括顺序图(sequence diagram)和协作图(collaboration diagram)两种形式。在 UML 2.0 中又增加了交互概观图和定时图。顺序图着重描述对象按照时间顺序的消息交换,协作图着重描述系统成分如何协同工作(即侧重于空间的协作)。顺序图和协作图从不同的角度表达了系统中的交互及系统的行为,它们之间可以相互转换。一个用例需要多个顺序图或协作图,除非特别简单的用例。

交互模型可以帮助分析人员对照检查每个用例中所描述的用户需求,审查这些需求是否已经落实到能够完成这些功能的类中去实现,提醒分析人员去补充遗漏的类或方法。交互模型和类图可以相互补充,类图对系统中的所有类及对象的描述比较充分,但对对象之间的消息交互情况的表达不考虑;而交互模型不考虑系统中的所有类及对象,但可以表示系统

中某几个对象之间的交互。

需要说明的是，交互模型描述的是对象之间的消息发送关系，而不是类之间的关系。在交互模型中一般不会包括系统中所有类的对象，但同一个类可以有多个对象出现在交互模型中。交互模型适合于描述一组对象的整体行为，其本质是对象间协作关系的模型。

5.2 顺 序 图

顺序图也称为时序图。Rumbaugh 对顺序图的定义是：顺序图是显示对象之间交互的图，这些对象是按时间顺序排列的。顺序图中显示的是参与交互的对象及对象之间消息交互的顺序。

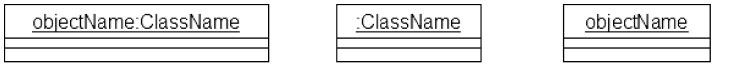
顺序图可以用来描述场景，也可以用来详细表示对象之间及对象与参与者之间的交互。在系统开发的早期阶段，顺序图应用在高层表达场景上；在系统开发的后续阶段，顺序图可以显示确切的对象之间的消息传递。顺序图是由一组协作的对象及它们之间可发送的消息组成的，强调消息之间的顺序。正是由于顺序图具备了时间顺序的概念，从而可以清晰地表示对象在其生命周期的某一时刻的动态行为。顺序图可以说明操作的执行、用例的执行或系统中的一次简单的交互情节。

顺序图是一个二维图形。在顺序图中水平方向为对象维，沿水平方向排列的是参与交互的对象。其中对象间的排列顺序并不重要，但一般把表示参与者的对象放在图的两侧，主要参与者放在最左边，次要参与者放在最右边（或表示人的参与者放在最左边，表示系统的参与者放在最右边）。顺序图中的垂直方向为时间维，沿垂直向下方向按时间递增顺序列出各对象所发出和接收的消息。

顺序图中包括的建模元素有对象（参与者实例也是对象）、生命线（lifeline）、控制焦点（focus of control）、消息（message）等。

5.2.1 对象

顺序图中对象的命名方式主要有 3 种（协作图中的对象命名方式也一样），如图 5.1 所示。对象名都要带下画线。



(a) 第一种命名方式　(b) 第二种命名方式　(c) 第三种命名方式

图 5.1　顺序图中对象的命名方式

第一种命名方式包括对象名和类名。第二种命名方式只显示类名，不显示对象名，即表示这是一个匿名对象。第三种命名方式只显示对象名，不显示类名，即不关心这个对象属于什么类。

5.2.2 生命线

生命线在顺序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间。因而可以看成是时间轴。

生命线表示对象在一段时间内的存在。只要对象没有被撤销，这条生命线就可以从上到下延伸。在水平方向上的对象并不是都处于一排的，而是错落有致的。其规则是：在图的顶部放置在所有的通信开始前就存在的对象。如果一个对象在图中被创建，那么就把创建对象消息的箭线的头部画在对象图标符号上。当一个对象被删除或自我删除时，该对象用×标识，标记它的析构(销毁)。在所有的通信完成后仍然存在对象的生命线，要延伸超出图中最后一个消息箭线。图 5.2 给出了这些概念的图示说明。

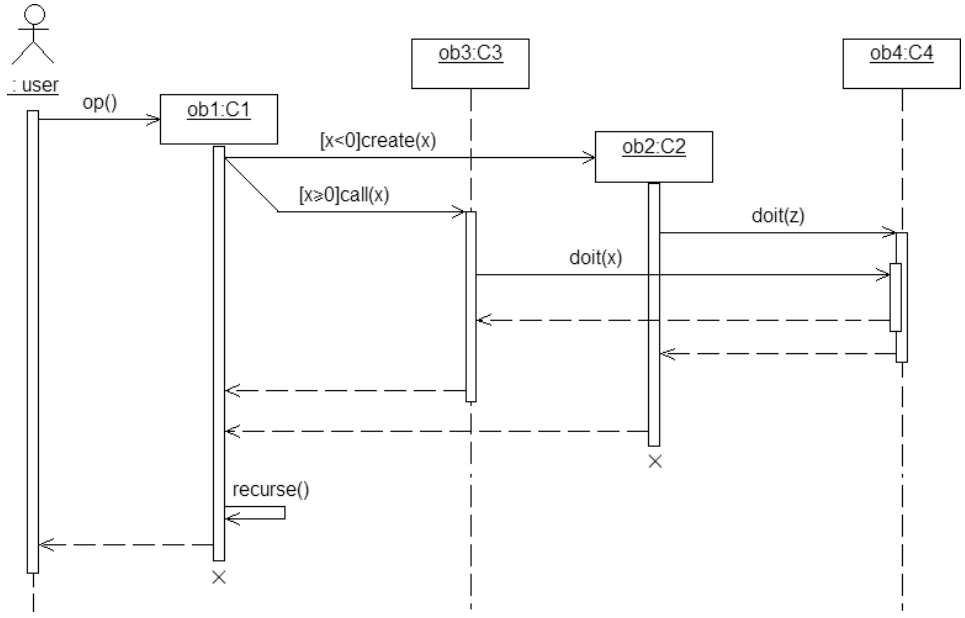


图 5.2　顺序图示例

【例 5.1】 在图 5.2 中，ob3:C3 和 ob4:C4 在所有的通信开始前就已经存在了，因此，它们位于图的顶端。ob3:C3 和 ob4:C4 在所有的通信完成后仍然存在，故生命线在图中超出了最后一个消息箭线。

ob1:C1 和 ob2:C2 是在交互过程中被创建和撤销的，故消息的箭线指向 ob1:C1 和 ob2:C2 的图标头部，并且在矩形的末端标记×。ob2:C2 是当 $x<0$ 条件满足时由 obl:Cl 创建的对象。obl:Cl 与 ob2:C2 在完成操作后结束生命周期。

因为对象 user 激活对象 obl:Cl，所以 obl:Cl 的图标头部要低于图的顶端一截。因为对象 obl:Cl 创建对象 ob2:C2，所以 ob2:C2 的图标头部要低于 ob1:C1 的图标头部一截。

5.2.3　控制焦点

控制焦点是顺序图中表示时间段的符号，在这个时间段内，对象将执行相应的操作。控制焦点表示为在生命线上的小矩形。

矩形的顶端和它的开始时刻对齐，即控制焦点符号的顶端画在进入的消息箭线所指向之处。矩形的顶端表示活动(即对象将执行相应的操作)的开始时刻。矩形的末端和它的结束时刻对齐，即控制焦点符号的底端画在返回的消息箭线的尾部。矩形的末端表示活动的结束时刻。

控制焦点可以嵌套，即当对象调用它自己的方法或接收另一个对象的回调时，在现有的控制焦点上要表示出一个新的控制焦点。嵌套的控制焦点可以更精确地说明消息的开始和结束位置。如图5.2所示，对象 ob4:C4 上的控制焦点出现了嵌套的现象。表明对象ob4:C4中，消息 doit(z)激活的活动还未结束，消息 doit(x)又激活了另一个活动。

与生命线上的小矩形相关的另外一个概念是激活期(activation)。激活期表示对象执行一个动作的期间，即对象激活的时间段。根据定义可以知道，控制焦点和激活期事实上表示的是同一个意思。

5.2.4 消息

一条消息是一次对象间的通信。顺序图中的消息可以是信号、操作调用或类似于C++中的RPC(Remote Procedure Calls)和Java中的RMI(Remote Method Invocation)的事件。当收到消息时，接收对象立即开始执行活动，即对象被激活了。

消息在顺序图中表示为从一个对象(发送者)的生命线指向另一个对象(目标)的生命线的带箭头的实线。每一条消息必须有一个说明，内容包括名称和参数。

在图5.2中，op()和[$x<0$]create()等都是消息。消息名字应以小写字母开头。

消息按时间顺序从顶到底垂直排列。如果多条消息并行，它们之间的顺序不重要。消息可以有序号，但因为顺序是用相对关系表示的，通常省略序号。带箭头的虚线用来表示从过程调用的返回。在控制的过程流中，可以省略返回箭线，这种用法假设在每个调用后都有一个配对的返回。对于非过程控制流，如果需要，应该显式地标出返回消息。

5.2.5 分支

分支是指从同一点发出多条消息并指向不同的对象。有两种类型的分支：条件分支和并行分支。

引起一个对象的消息产生分支有两种情况。

情况一，在复杂的业务处理过程中，要根据不同的条件进入不同的处理流程，通常称为条件分支。

情况二，当执行到某一点时，需要向两个或两个以上的对象发送消息，通常称为并行分支。

在Rational Rose 2003版本中，不支持分支的画法。如果有必要，可以通过对分支中的每一条消息画出一个顺序图的方式来实现。

5.2.6 从属流

从属流是指从同一点发出多条消息并指向同一个对象的不同生命线，即由于不同的条件而执行了不同的生命线分支。

在Rational Rose 2003版本中，不支持从属流的画法。如果有必要，可以通过对从属流中的每一条消息画出一个顺序图的方式来实现。

5.3 顺序图中的消息

顺序图中的一个重要概念是消息。消息也是 UML 规范说明中变化较大的一个内容。UML 在 1.4 及以后版本的规范说明中对顺序图中的消息做了简化，只规定了调用消息、异步消息和返回消息这 3 种消息。而在 UML 1.3 及以前版本的规范说明中还有简单消息这种类型。此外 Rose 对消息又做了扩充，增加了阻止(balking)消息、超时(time-out)消息等。

5.3.1 调用消息

调用(procedure call)消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息接收者放弃或返回控制。调用消息可以用来表示同步的意义，事实上，在 UML 规范说明的早期版本中，就是采用同步消息这个术语的。

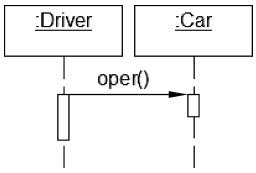


图 5.3 调用消息

调用消息的表示符号如图 5.3 所示，其中 oper()是一个调用消息。

调用消息的接收者必须是一个被动对象(passive object)，即它是一个需要通过消息驱动才能执行动作的对象。另外调用消息必有一个配对的返回消息，为了图的简洁和清晰，与调用消息配对的返回消息可以不用画出。

5.3.2 异步消息

异步(asynchronous)消息的发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接收者返回消息或控制。异步消息的接收者和发送者是并发工作的。

图 5.4 所示是 UML 规范说明 1.4 及以后版本中表示异步消息的符号。与调用消息相比，异步消息在箭头符号上不同。

需要说明的是，这是 UML 规范说明 1.4 及以后版本中表示异步消息的符号。同样的符号在 UML 规范说明 1.3 及以前版本中表示的是简单消息，而在 UML 规范说明 1.3 及以前版本中表示异步消息是采用半箭头的符号，如图 5.5 所示。

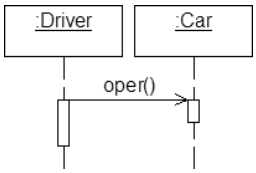


图 5.4 UML 规范说明 1.4 及以后版本中的异步消息

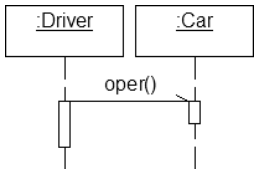


图 5.5 UML 规范说明 1.3 及以前版本中的异步消息

5.3.3 返回消息

返回(return)消息表示从过程调用返回。如果是从过程调用返回，则返回消息是隐含的，所以返回消息可以不用画出来。对于非过程调用，如果有返回消息，必须明确表示出来。

过程调用是指消息名和接收消息的接收对象的方法名相同，即消息直接调用了接收对象的某个方法。非过程调用是指消息是事件发生（即信号），该事件的出现修改了变量（全局变量或局部变量）的值，从而导致了接收对象的某个方法的执行。

上述的调用消息一定是过程调用，因此一定存在返回消息，只不过可以不用画出来。上述异步消息有的是过程调用，有的是非过程调用。无论是哪种情况，如果异步消息有返回消息，则一定要画出来。

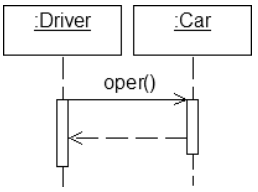


图 5.6　返回消息

返回消息都是异步消息，可以并发运行。图 5.6 所示是返回消息的表示符号，其中的虚线箭头表示对应于 oper()这个消息的返回消息。

5.3.4　阻止消息

除了调用消息、异步消息和返回消息这 3 种消息外，Rose 还对消息类型做了扩充，增加了阻止消息和超时消息。

阻止消息是指消息发送者发出消息给接收者，如果接收者无法立即接收消息，则发送者放弃这个消息。Rose 中用折回的箭头表示阻止消息，如图 5.7 所示。

5.3.5　超时消息

超时消息是指消息发送者发出消息给接收者并按指定时间等待。如果接收者无法在指定时间内接收消息，则发送者放弃这个消息。图 5.8 所示是超时消息的例子。

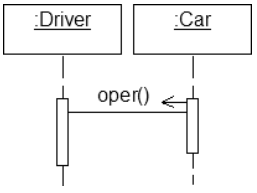


图 5.7　阻止消息

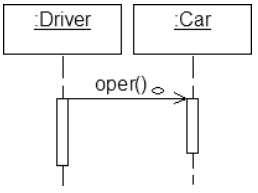


图 5.8　超时消息

5.3.6　消息的语法格式

UML 中规定的消息语法格式如下：

```
[predecessor] [guard-condition] [sequence-expression] [return-value :=]
message-name ([argument-list])
```

上述定义中用方括号括起的是可选部分，各语法成分的含义如下。

predecessor：必须先发生的消息的列表，是一个用来同步线程或路径的表达式。其中消息列表中的各消息号用逗号分隔，格式如下：

```
sequence-number ','…'/'
```

guard-condition：警戒条件，是一个在方括号中的布尔表达式，表示只有在 guard-condition 满足时才能发送该消息。格式如下：

```
'['boolean-expression']'
```

这里的方括号放在单引号中，表示这个方括号是一个字符，是消息的组成部分。

sequence-expression：消息顺序表达式。消息顺序表达式是用句点“.”分隔，以冒号“：”结束的消息顺序项(sequence-term)列表。格式如下：

```
sequence-term '.'...':'
```

其中，可能有多个消息顺序项，各消息顺序项之间用句点“.”分隔，每个消息顺序项的语法格式如下：

```
[integer|name] [recurrence]
```

其中，integer 表示消息序号，name 表示并发的控制线程。

例如，如果两个消息为 3.1a、3.1b，则表示这两个消息在激活期 3.1 内是并发的。

recurrence 表示消息是条件执行或循环执行，有以下几种格式：

```
'*'['['iteration-clause']']
```

表示消息要循环发送。

```
'['condition-clause']'
```

表示消息是根据条件发送的。

需要说明的是，UML 中并没有规定循环子句和条件子句的格式，分析人员可以根据具体情况选用合适的子句表示格式。另外如果循环发送的消息是并发的，可用符号 *||表示。

return-value：将赋值作为消息返回值的名字列表。返回值表示一个操作调用(消息)的结果。如果消息没有返回值，则 return-value 部分被省略。

message-name：消息名。

argument-list：消息的参数列表。

一些消息的例子如表 5.1 所示。

表 5.1 消息的例子

消 息 名	解 释
2:display(x,y)	简单消息
1.3.1:p:=find(specs)	嵌套消息，消息带返回值
[x<0]4:invert(x,color)	警戒条件消息或条件发送消息
4.2[x>y]:invert(x,color)	条件发送消息
3.1*:update()	循环发送消息
A3,B4/C2:copy(a,b)	线程间同步
1.1a,1.1b/1.2:continue()	同时发送的并发消息作为先发消息序列

【例 5.2】 表 5.1 给出的 7 个消息，其解释如下。

2:display(x,y)：表示序号为 2 的消息(即第 2 个发出的消息)，消息名为 display，消息参数为 x 和 y。这是一个最简单格式的消息。

1.3.1:p:=find(specs)：表示序号为1.3.1的消息(1.3.1为嵌套的消息序号,表示消息1的处理过程中的第3条嵌套消息的处理过程中的第1条嵌套的消息),返回值名为p,消息名为find,消息参数为specs。需要注意的是,嵌套的消息序号1.3.1暗指序号为1.2以及后续的1.2.x的消息已经处理完毕。

[x<0]4:invert(x,color)：表示序号为4的消息,消息名为invert,消息参数为x和color。[x<0]可以认为是警戒条件,也可以认为是消息顺序表达式中的条件发送格式。其含义是当条件x<0满足时,发送第4个消息invert。

4.2[x>y]:invert(x,color)：表示序号为4.2的消息(4.2为嵌套的消息序号,表示消息4的处理过程中的第2条嵌套的消息),消息名为invert,消息参数为x和color。[x>y]只能是消息顺序表达式中的条件发送格式。其含义是当条件x>y满足时,发送第4.2个消息invert。需要注意的是,条件[x>y]出现在消息序号的后面。

3.1 * :update()：表示序号为3.1的消息(3.1为嵌套的消息序号,表示消息3的处理过程中的第1条嵌套的消息),消息名为update,无消息参数。 * 是消息顺序表达式中的循环发送格式。其含义是循环发送第3.1个消息update多次。需要注意的是,循环 * 无论出现在消息序号的前面还是后面,都表示循环发送格式。

A3,B4/C2:copy(a,b)：表示先发送线程A的第3个消息和线程B的第4个消息后,再发送线程C的第2个消息,消息名为copy,消息参数为a和b。A3,B4为消息的前序,用来描述同步线程。C2为消息顺序表达式中的消息顺序项,C表示并发的控制线程,2表示消息序号。

1.1a,1.1b/1.2:continue()：表示同时发送的并发消息1.1a和1.1b之后,再发送序号为1.2的消息,消息名为continue。1.1a,1.1b为消息的前序,用来描述同步消息。1.2为消息顺序表达式中的消息顺序项,表示嵌套的消息序号。

5.3.7 调用消息和异步消息的比较

调用消息主要用于控制流在完成之前需要中断的情况,异步消息主要用于控制流在完成之前不需要中断的情况。

【例5.3】 在一个学生成绩管理系统中,需要实现教师登记学生分数的功能,如图5.9所示。

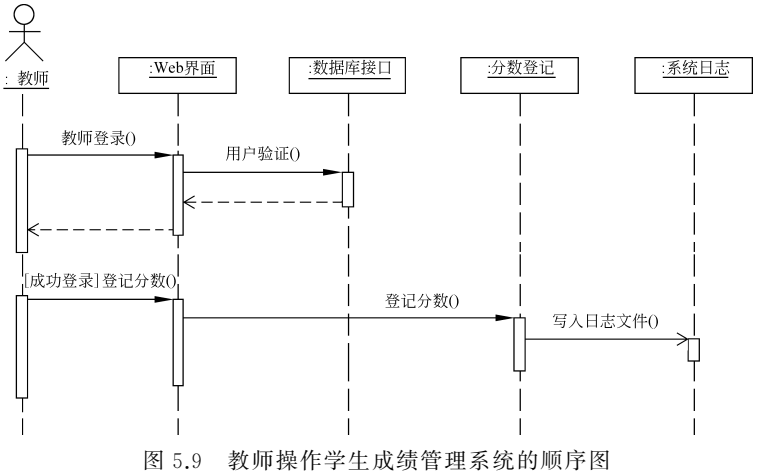


图5.9 教师操作学生成绩管理系统的顺序图

教师试图登录 Web 界面，Web 界面需要发送消息到数据库接口，由它完成教师输入的账号和密码的对错判断。在账号和密码的对错判断完成之前，教师登录必须被中断，即对象“:Web 界面”的第一个控制焦点表达的活动必须被中断。因此，“用户验证()”消息是调用消息。

教师成功登录后，对学生的考试分数进行登记。此时，出于安全考虑，系统要进行日志文件的写入，以便记录教师的整个操作过程。在进行分数登记时，同时会写入日志文件，即分数登记操作不需要中断。也就是说，对象“:分数登记”的控制焦点表达的活动不会被中断。因此，“写入日志文件()”消息是异步消息。

因为调用消息要求发送消息的对象在发出调用消息后，停止自己的活动，将控制权移交给接收消息的对象。所以，调用消息可以表现嵌套控制流，而异步消息表现的是非嵌套的控制流。

5.4 建立顺序图概述

5.4.1 建立顺序图

一般在一个单独的顺序图中只描述一个控制流，若需要，也可以使用分支的表示法。因为一个完整的控制流通常是复杂的，所以合理的方法是将一个大的控制流分为几个部分放在不同的顺序图中。

建立顺序图的策略如下。

(1) 按照当前交互的意图，详细地审阅相关资料(例如用例描述)，设置交互的语境，确定将要建模的工作流。

(2) 通过识别对象在交互中扮演的角色，在顺序图的上部列出所选定的一组对象，并为每个对象设置生命线。一般把发起交互的对象放在左边，这些对象是在结构建模中建立类图时分析得到的结果，可以是一般类、边界类、控制类、实体类。这里，一般类就是完成一个具体功能的类。

(3) 对于那些在交互期间要被创建和撤销的对象，在适当的时刻，用消息箭线在它们的生命线上显式地予以指明。发送消息的对象将创建消息箭线指向被创建对象(它是接收消息的对象)的图标符号上。发送消息的对象将撤销消息箭线指向被撤销对象(它是接收消息的对象)的生命线上，并标记×符号。

(4) 在各个对象下方的生命线上，按使用该对象操作的先后顺序排列各个代表操作的窄矩形条。从引发这个交互过程的初始消息开始，在生命线之间自顶向下依次画出随后的各个消息。

(5) 决定消息将怎样或以什么样的顺序在对象之间传递。通过发起对象发出的消息，分析它需要哪些对象为它提供操作，它向哪些对象提供操作。注意选择适当的消息类型(调用、异步、返回、阻止、超时)，画出消息箭线，并在其上标明消息名。追踪相关的对象，直到分析完与当前语境有关的全部对象。

(6) 两个对象的操作执行如果属于同一个控制线程，则接收者操作的执行应在发送者发出消息之后进行，并在发送者结束之前结束。不同控制线程之间的消息有可能在接收者

的某个操作的执行过程中到达。

（7）如果需要表示消息的嵌套，或/和表示消息发生时的时间点，则采用控制焦点。

（8）如果需要，则可以对对象所执行的操作的功能及时间或空间约束进行描述。

（9）如果需要，则可以为每个消息附上前置条件和后置条件。

（10）如果需要可视化消息的迭代或分支，则使用迭代或分支表示法。

5.4.2 顺序图与用例描述的关系

用例描述（Use Case Specification）是一个关于参与者与系统如何交互的规范说明。由于用例描述了参与者和软件系统进行交互时，系统所执行的一系列的动作序列。因此，这些动作序列不但应包含正常使用的各种动作序列（称为主事件流），而且还应包含对非正常使用时软件系统的动作序列（称为子事件流）。所以，主事件流描述和子事件流描述是用例描述的主要内容。

一个单独的顺序图中只描述一个控制流。如果控制流复杂，则可以将它分为几个部分放在不同的顺序图中。

一般情况下，用例描述使用自然语言描述参与者使用系统的一项功能时，系统所执行的一系列的动作序列。经过结构建模的分析，已经获得了完成该项功能所需要的所有类（包括一般类、边界类、控制类、实体类）。这些类所对应的对象通过彼此间发送消息，控制对方的动作执行。一个完整的消息发送序列（即在一个顺序图中从顶到底垂直排列的所有消息）表达了用例描述中一个事件流的具体的对象实现方案。因此，一个顺序图可以对应用例描述中的一个事件流。如果一个用例具有复杂的事件流，则需要多个顺序图进行描述。

所以说，通过顺序图，系统分析人员可以对照检查每个用例中所描述的用户需求，审查这些需求是否已经落实到能够完成这些功能的类中去实现，提醒分析人员去补充遗漏的类或方法，从而进一步完善类图。

另外，对于一个动作序列复杂的用例，其事件流可以分解为几部分。如果其中一部分只是包含了非参与者对象，那么对应的顺序图可以没有参与者对象的出现。

5.4.3 顺序图与类图的区别

类图描述的是类和类之间的静态关系。通过类之间的关联关系，类图显示了信息的结构及信息间的静态联系。通过类之间的依赖关系，类图显示了类之间属性上和（或）操作上的静态联系，但这种联系反映在程序代码上，不反映在类方法的调用上。

顺序图描述的是对象之间的动态关系。通过对象之间的消息传递，顺序图显示了对象方法上的调用关系和次序。

因此，在类图中会出现关联关系和依赖关系，但不会出现协作关系。在顺序图中会出现协作关系，但不会出现关联关系和依赖关系。

作为类图的一次快照，对象图描述的是对象之间的链的关系（链是关联关系的实例）。从本质上讲，对象图还是描述对象之间的关联关系，不会反映出对象之间消息传递的协作关系。

类之间的关联关系实质上对应着对象之间的链的连接关系。当软件系统开发完成并投入使用后，一旦永久存储了信息后，对象之间的链的连接就已经建立起来了。无论今后软件