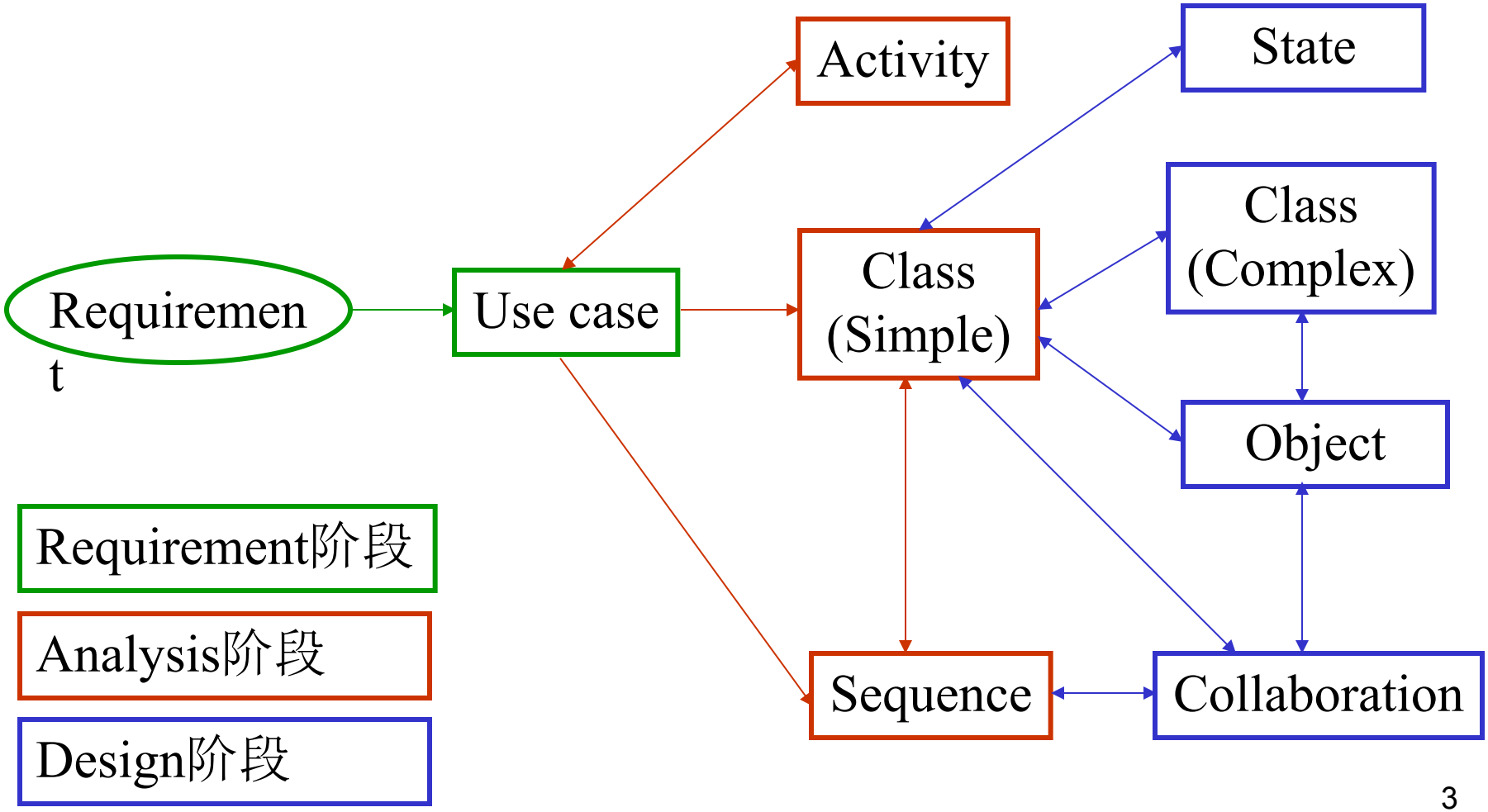
# 第10章 顺序图
# (sequence diagram)

# 学习目标

- 学习完本章节，要求达到以下状态：
  - 能够说明交互图的基本概念
  - 能够说明顺序图的表示方法和使用方法
  - 能够读懂顺序图并理解其中的含义

# UML 1.x中各种图的关系



# 交互图(interaction diagram)

- 在对软件密集型(software-intensive)系统建模时，如何对它的**动态方面**建模？怎样才能可视化一个运行的系统？
- **交互图(interaction diagram)**是一种详细表示对象之间以及对象与系统外部的参与者(actor)之间动态联系的图形文档。
- UML 1.x版本中，交互图有两种形式，即**顺序图(sequence diagram)** 和**协作图(collaboration diagram)** 。

交互图说明：

- **交互图**是用来描述对象之间的动态协作关系以及协作过程中的行为次序，它常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。
- 一般一个use case需要一个或多个顺序图或协作图，除非特别简单的use case.
- **顺序图**和**协作图**从不同的角度表达了系统中的交互和系统的行为，它们之间可以相互转化。
- **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。

# 交互图的用途

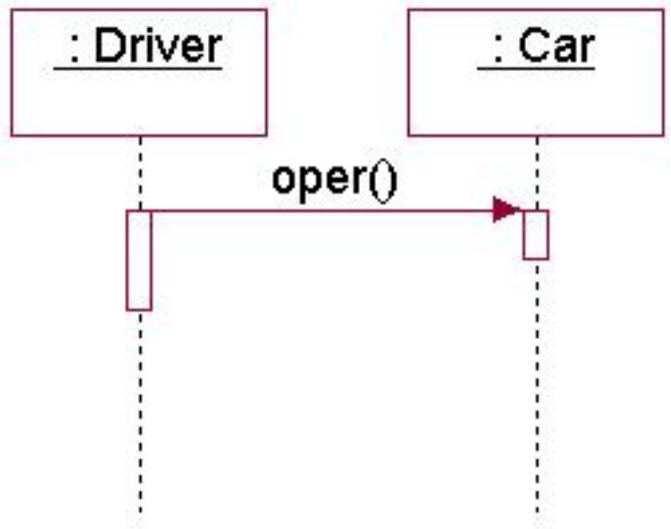
- 帮助分析员对照检查每个use case 中描述的用户需求，是否已经落实到一些能够完成这些功能的类中去实现。提醒分析员去补充遗漏的类或服务。
- 和**类图**相互补充
  - 类图对对象之间的消息（交互情况）表达不够详细；
  - 交互图表示少数几个对象之间的交互。

# 交互图

- 表示“人”或者“物”(生命线lifeline)之间的消息交互。。
- 根据用例图中的脚本来做成交互图。
- 顺序图和通信图之间可以互相转换，根据需要选择使用其中一种。

# 顺序(Sequence)图

- 定义：A **sequence diagram** is a diagram that shows object interactions arranged in time sequence. In particular, it shows the **objects** participating in an interaction and the sequence of messages exchanged.

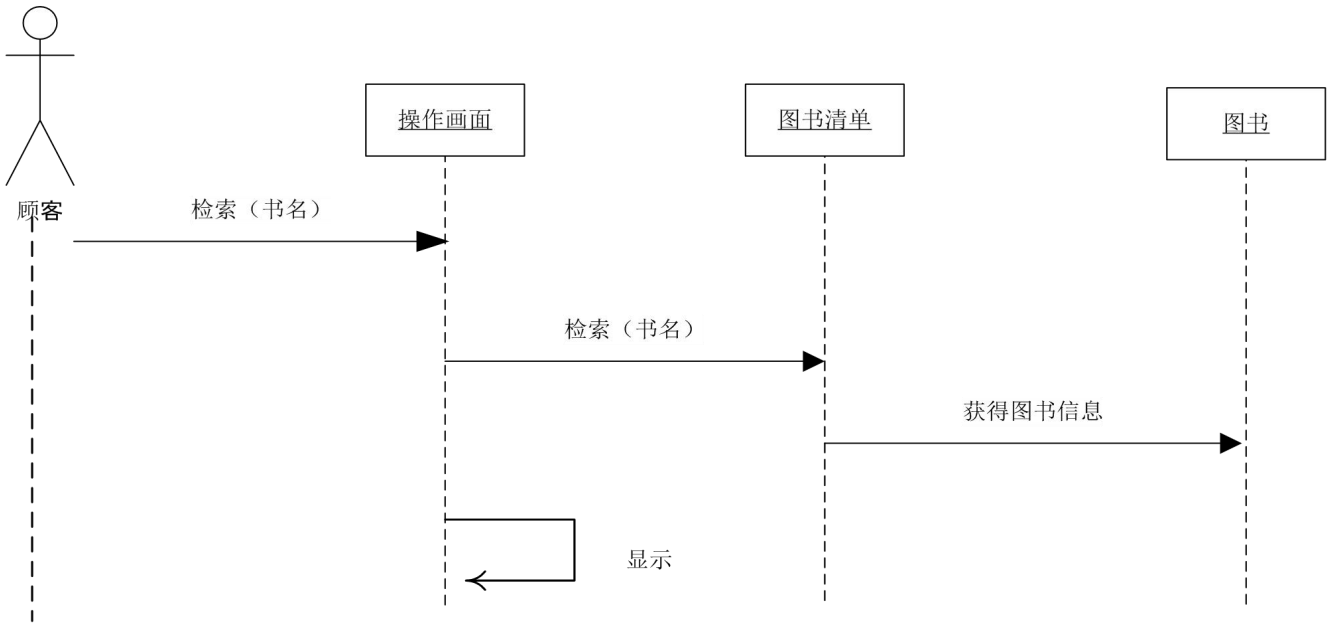


说明：

- **顺序图**是一个二维图形。在顺序图中水平方向为对象维，沿水平方向排列参与交互的对象；竖向方向为时间维，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。
- 水平轴上的对象间的相互顺序并不重要。
- **顺序图**不表示对象间的关联(**associations**)关系。

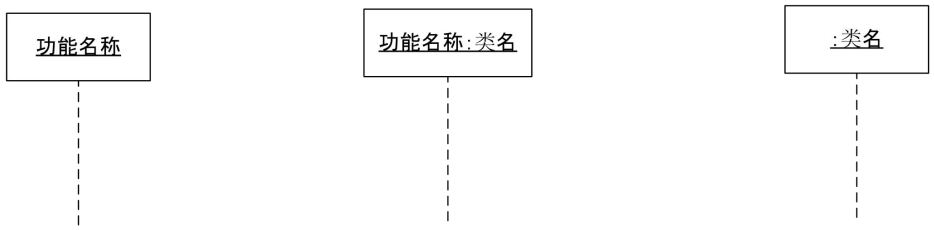
# 顺序图(sequence diagram)的概要

- 按时间顺序来表示生命线之间的消息交互。
  - 可用来表示用例实现的时候类或者组件之间的交互情况。
- 顺序图的特征：
  - 消息从上到下，从左到右进行排列，处理顺序一目了然。
- 顺序图的组成元素：
  - 生命线
  - 消息

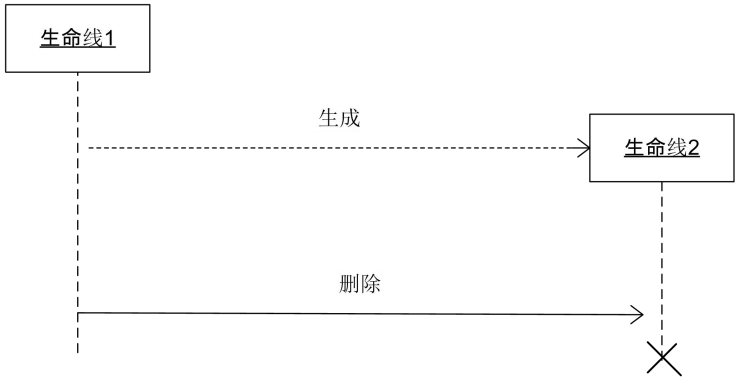


# 生命线(lifeline)

- 表示参与交互活动的元素(对象，组件等实体)
- 生命线的表示方法
  - 矩形框下加一条虚线来表示，矩形框中记述生命线的名称。虚线表示生命线的存在。
  - 生命线的名称有三种表示方法：

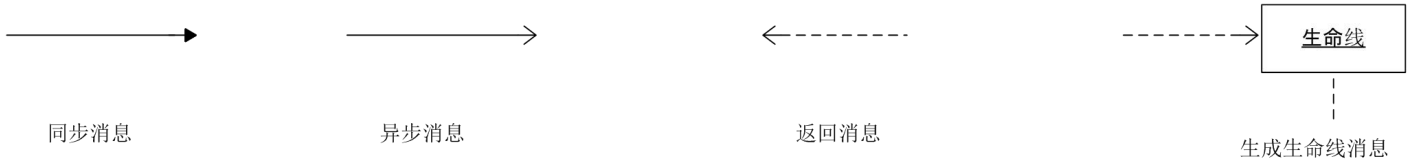



- 生命线的生成和消失
  - 生命线名称矩形框下面的虚线表示生命线的生存期间。
  - 生成一个生命线的消息要用虚线箭头表示，新生成的生命线连接在箭头方。
  - 在生命线的末端用“X”符号表示生命线在此刻消失。



# 消息

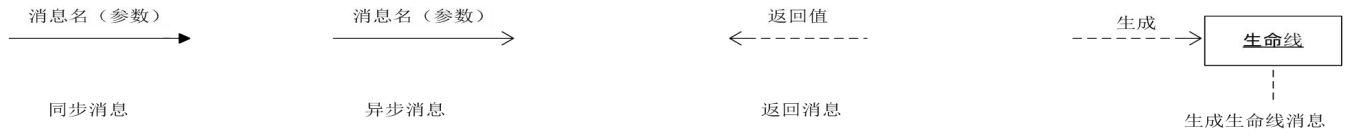
- 表示生命线之间的通信。
- 在生命线之间通过带箭头的线来连接，箭头所指的生命线为接收消息方，接受消息方的生命线接受到该消息后激活相应的动进行响应。
- 消息有四种：




- 消息上方可以加上消息名称，参数或者返回值，格式为：

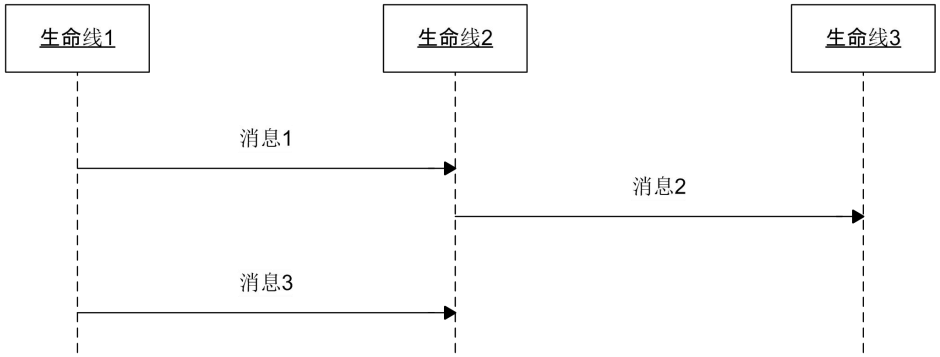
  发送消息： 消息名(参数1，参数2.。。)

  返回消息： 变量=消息名(参数1，参数2.。。):返回值



# 同步消息

- 表示消息执行的同步顺序(层次化关系)。
  - 某个生命线发出的上一条同步消息处理完全结束之后，才能发送和处理下一条消息。
  - 用实线实心箭头表示。
  - 可以明确控制消息发送和处理的先后顺序。

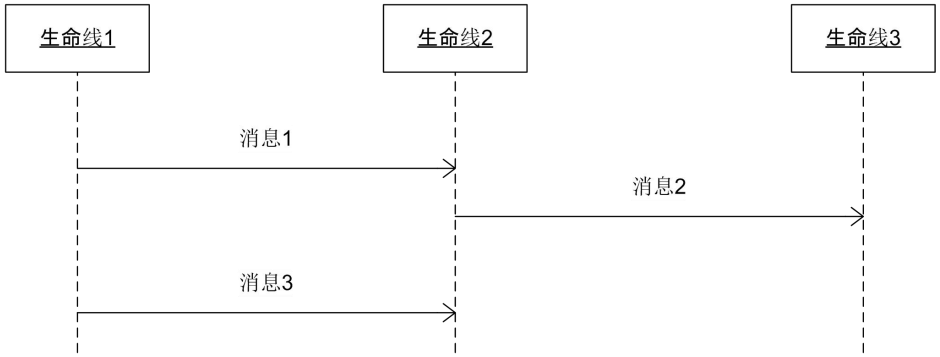



**消息2的处理结束之后，消息1的处理才完全结束。**

**等消息1的处理完全结束后，生命线1才能发送消息3.**

# 异步消息

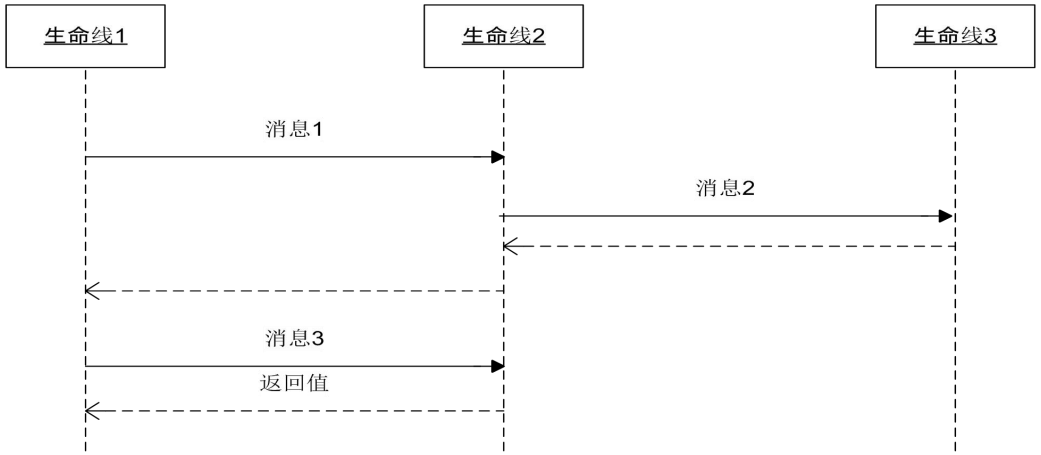
- 表示没有同步关系的消息。
  - 某个生命线不用等待发出的上一条异步消息处理结束，即可发送和处理下一条消息。
  - 用实现空心箭头表示。
  - 可以用来实现多条消息的并行发送。




只要消息1成功发出后，不用等待消息1的处理完全结束，生命线1就能发送消息3.

# 返回消息

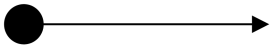
- 表示对于同步消息的应答。
  - 不强调返回关系和返回值的情况下，可省略返回消息的表示。。



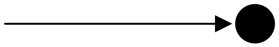

**返回消息可以携带返回值，也可省略返回值。**

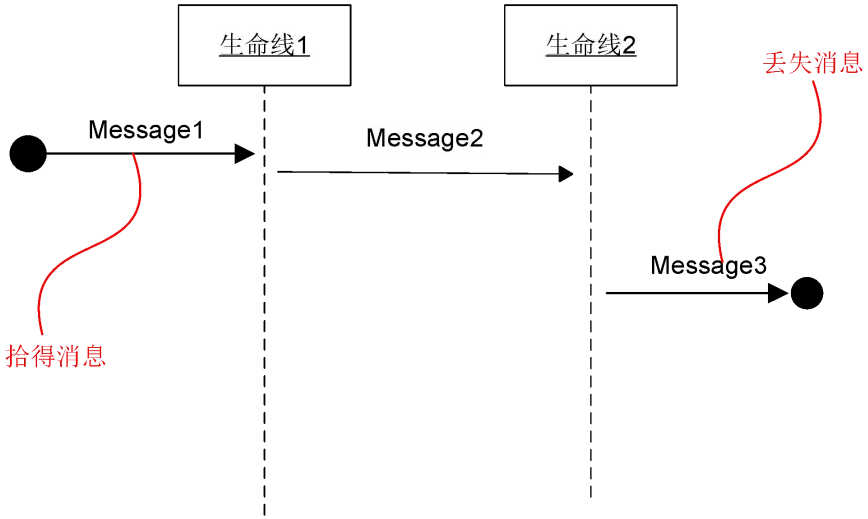
# 参考:拾得消息(found message)和丢失消息( lost message )

- 顺序图中的消息不一定都有明确的发送源和接收方。
  - 对于发送方不明确的消息成为拾得消息(found message)。如下表示:



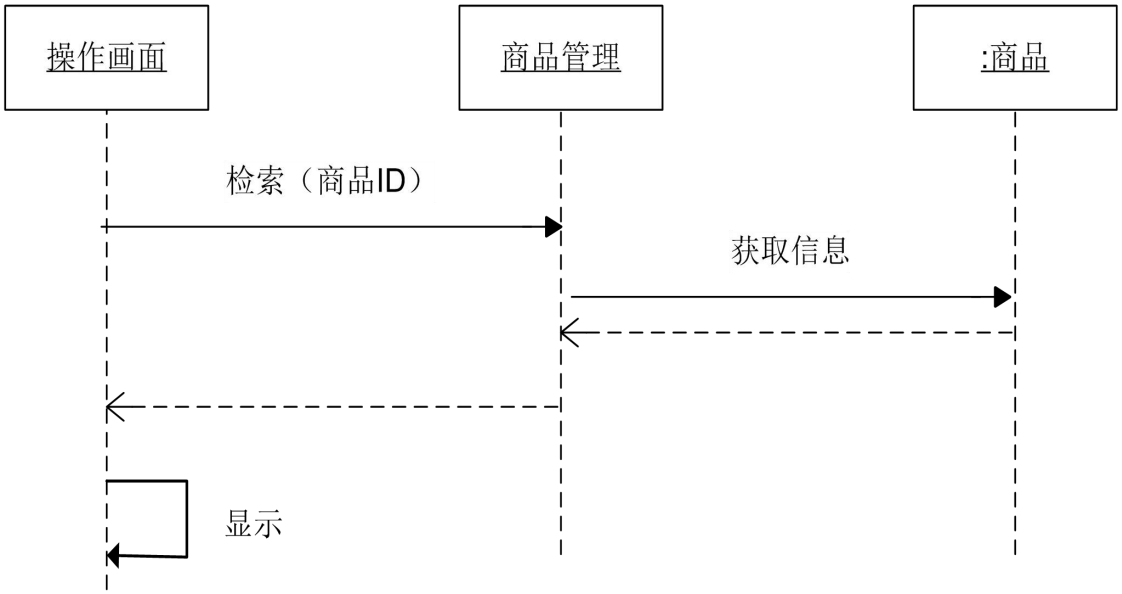
  - 对于接收方不明确的消息成为丢失消息(lost message)。如下表示:





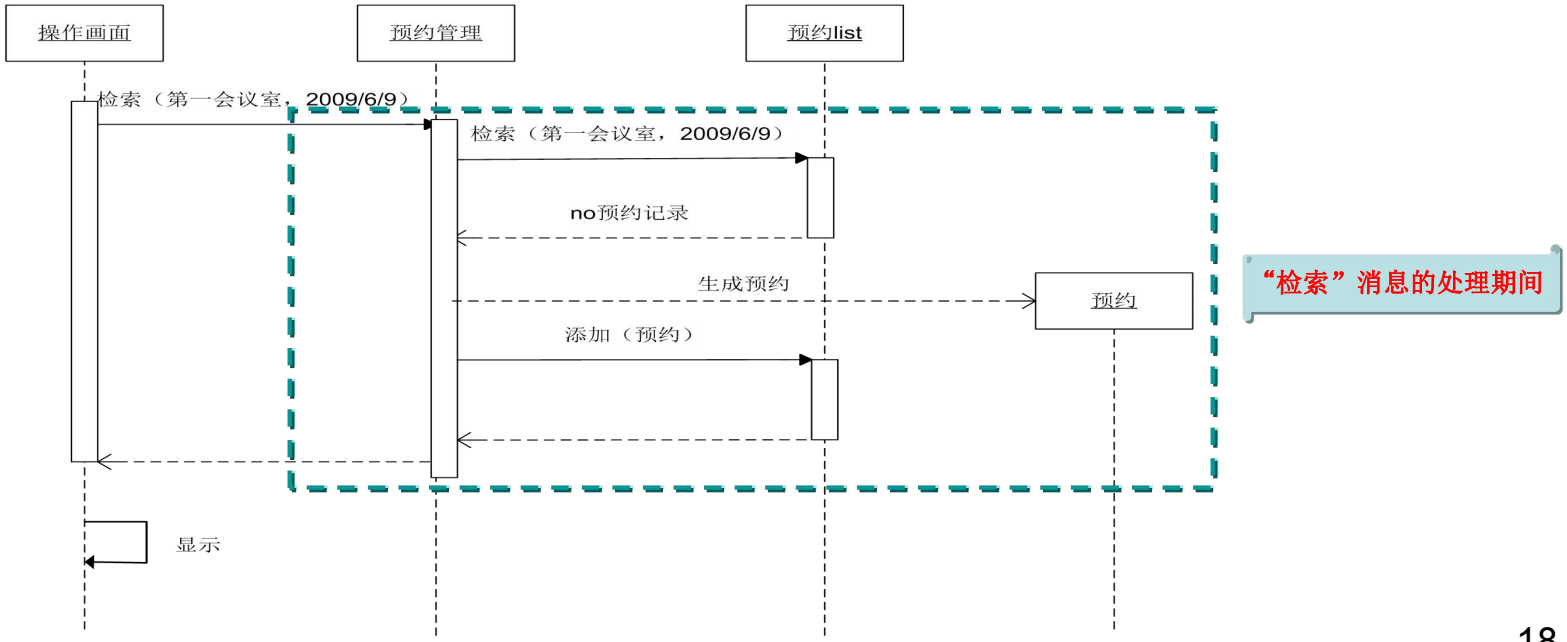
# 递归调用

- 表示生命线对自身发送消息调用自身操作的情况。
  - 从生命线向自身通过折线进行消息发送的表示。



# 激活框

- 激活框包含以下两方面的信息：
  - 生命线的某个处理被执行的期间。
  - 生命线的某个处理的执行和调用方之间的控制关系。
- 激活框的表示方法：
  - 在生命线上用细长的矩形框来表示。
  - 矩形框最上端表示处理的开始时点，矩形框的最下端表示处理的结束时点。
- 在顺序图中，激活框为可选项，可省略。
  - 需要明确表示消息和被调用的处理之间的控制关系，以及消息处理的先后顺序时，可使用激活框。



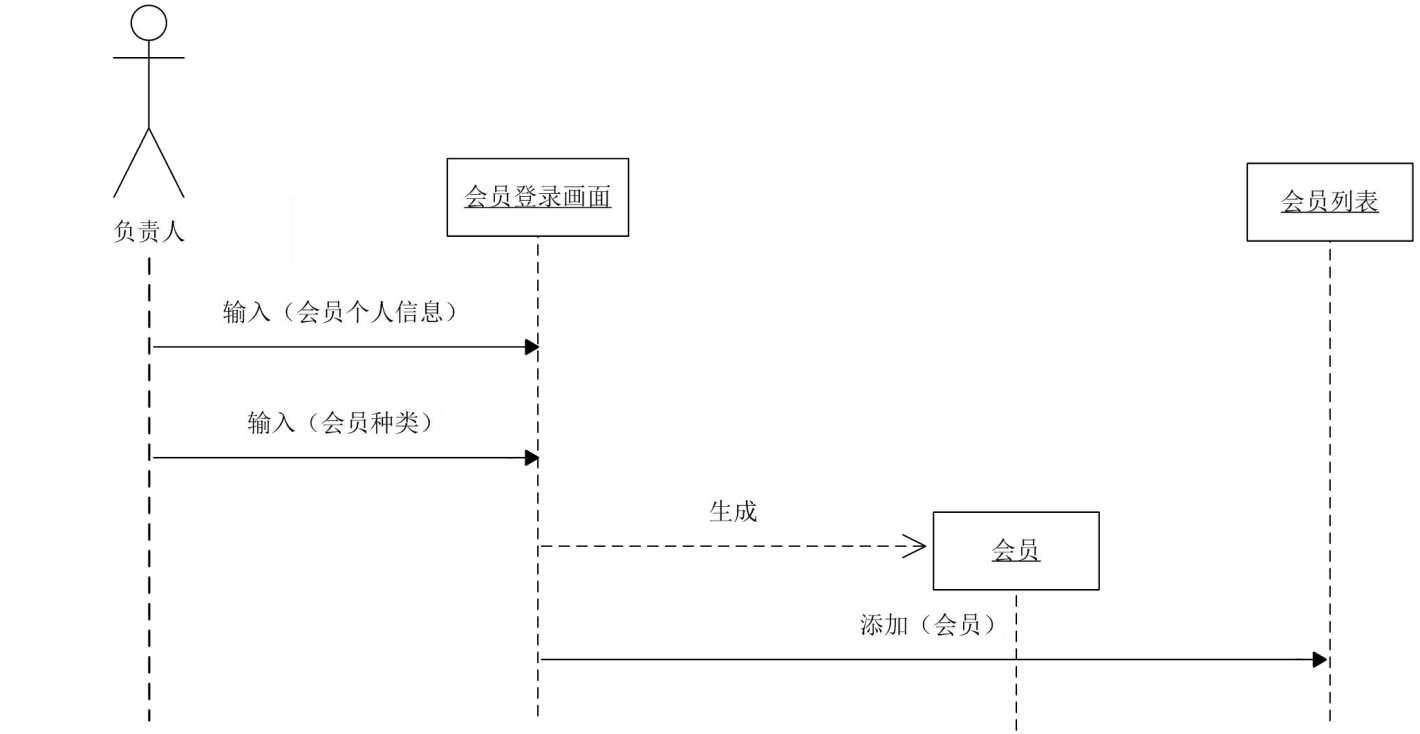
# 例题:画顺序图

◆ 想要描述的脚本如下:

某电子商务系统,负责人在有人申请入会时,首先在会员登录画面上输入入会申请人的个人信息,然后再输入会员申请的会员种类。

系统根据输入的信息新生成一会员,添加到会员列表中。

# 题解：

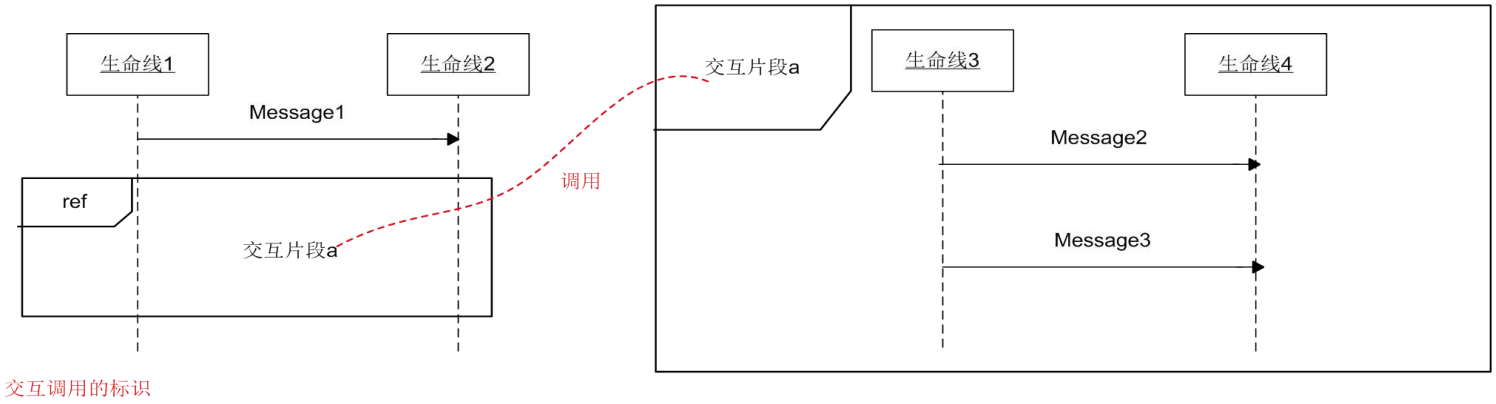


# 交互调用

◆ 在一个交互的执行过程中去参照调用另外一个在它处定义好了的交互序列称为交互调用。

■ 在框架的head部分用关键字“ref”进行标识，框架中央指明要调用的交互序列名称。

■ 方便合理组织顺序图中的元素，简化图中的交互调用关系，增强复杂交互关系的可读性和可维护性。

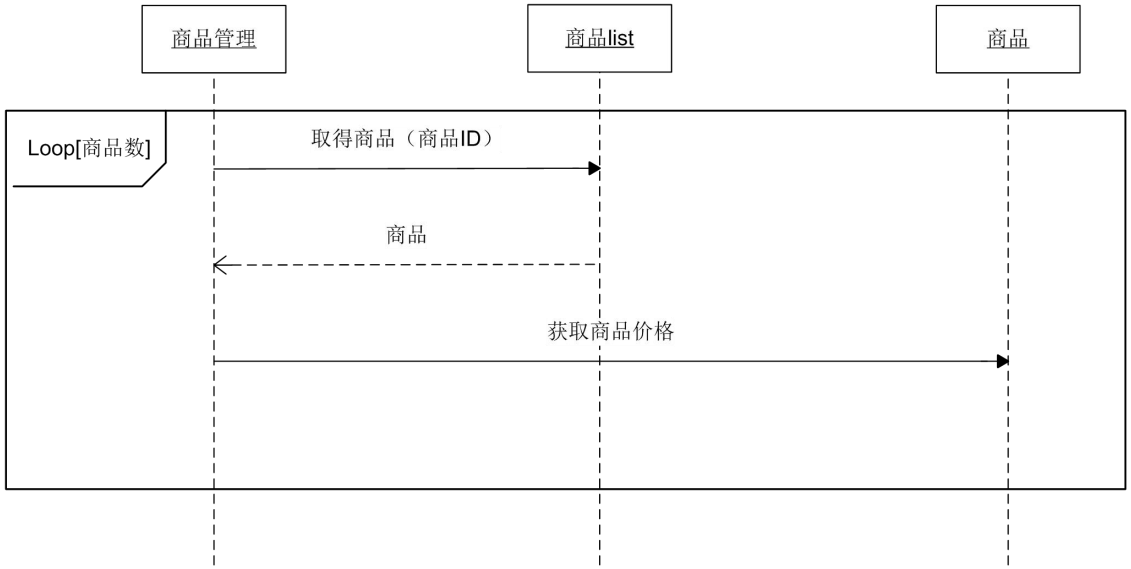


# 组合片段(combine fragment)

- 对复杂的交互图进行有意义的片段划分。
  - 方便表现具有循环，或者条件分支等交互序列。
- 常用的组合片段有：
  - loop
    - 表示循环执行某个交互序列。
  - alt (alternatives)
    - 根据条件选择不同交互处理序列，各个分支的条件必须具有互斥关系。
  - opt (option)
    - 表示只有在给定的条件满足的时候才能执行的交互序列，不能带有条件分支。
  - ctitical (critical region)
    - 表示不能接受从其它处理来的中断的交互序列。
  - break
    - 表示只有在给定条件满足的时候才能中断的交互序列。
  - par(parallel)
    - 表示有并行处理的交互序列。

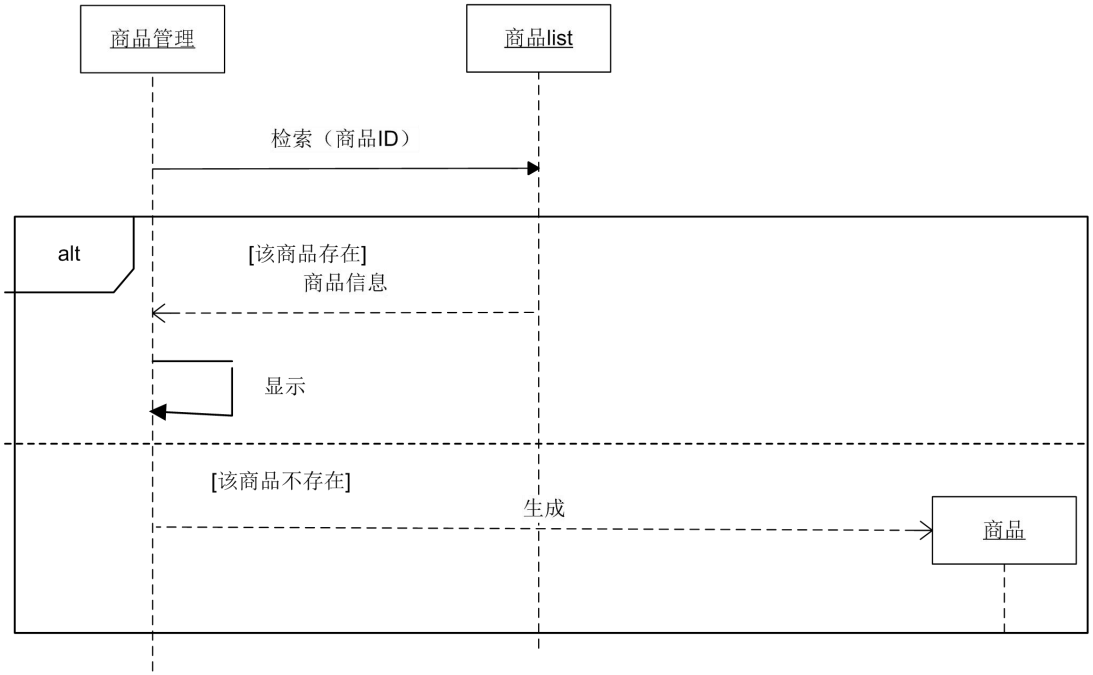
# loop组合片段的例子

- 在组合片段框的head部用如下格式标识：
  loop【循环次数】
  - 循环次数表达式分为两类
    - 循环次数表达式　LOOP【m】, LOOP【m,n】　m,n均为常数
    - 循环条件表达式　LOOP【条件表达式】



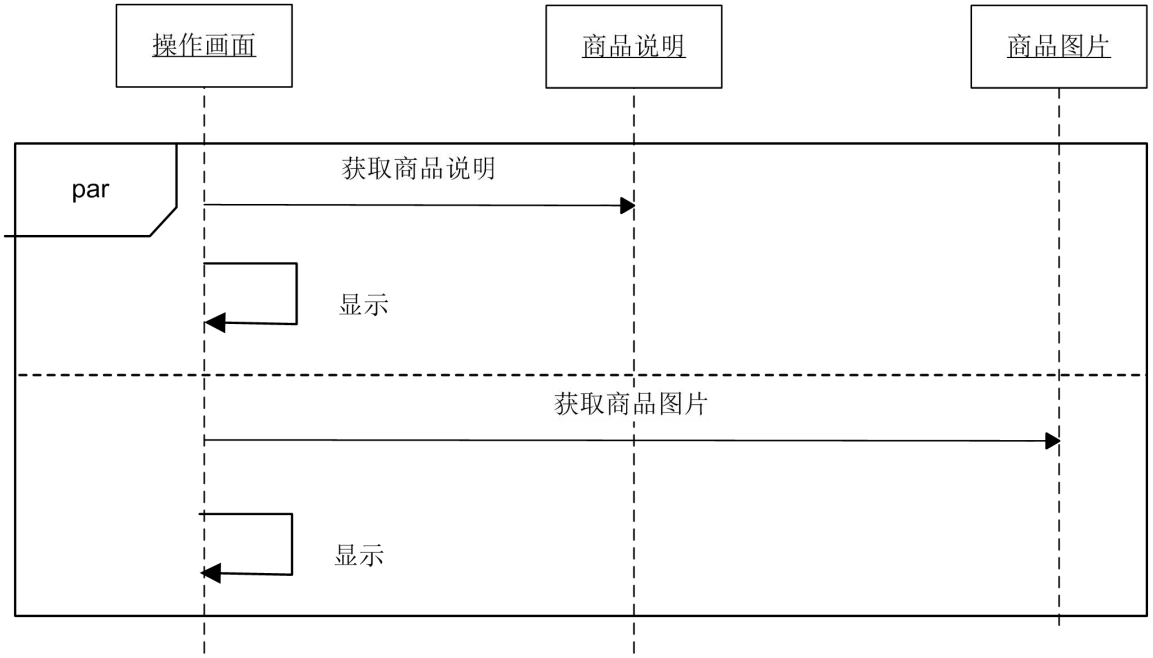
# alt组合片段的例子

- 在组合片段框的head部用关键字“alt”来进行标识：
  - 组合片段框中央用虚线进行划分，【】中表明处理分支条件。



# par组合片段的例子

- 在组合片段框的head部用关键字“par”来进行标识：



# 总结

- 在面向对象系统中，程序是随着系统中的各个实体(对象，组件等)之间的消息交互，不断往前执行着。
- UML中的交互图有
  - 顺序图:按照时间顺序描述生命线之间的消息交互。
  - 通信图:着眼于生命线之间的链接来描述生命线之间的消息交互。
- 顺序图由生命线和消息组成。
- 消息有以下几种:
  - 同步消息
  - 异步消息
  - 返回消息
  - 生成消息
- 在顺序图中可选用激活框来明确表示消息和处理之间的控制关系。
- 在顺序图中可使用组合片段和交互调用方法来增强顺序图的可读性。

# 练习题

- 请参考《UMTP-L1培训》练习题。

# 总结

- 顺序图和通信图之间的比较：
  - 顺序图是按照时间顺序从上到下排列消息来描述系统中生命线之间的消息交互情况。
    通信图是着眼于生命线之间的链接来描述生命线之间的消息交互情况。
  - 顺序图中的消息可以省略消息番号。
    通信图中的消息必须明确消息番号来表示消息发送的先后顺序。
  - 顺序图中，用组合片段“par”来描述消息的并行处理。
    通信图中，用“数字+字母”的消息番号定义方式来描述消息的并行处理。
- 描述系统静态结构的类图和描述系统动态行为的交互图之间要保持一致性：
  - 交互图中的生命线对应于类图中的类。
  - 交互图中有消息交互的生命线，对应在类图中的两个类之间应该有一定的关联。
  - 交互图中的消息对应于类图中接受该消息生命线所对应的类的某个操作。

# 建立sequence图的步骤

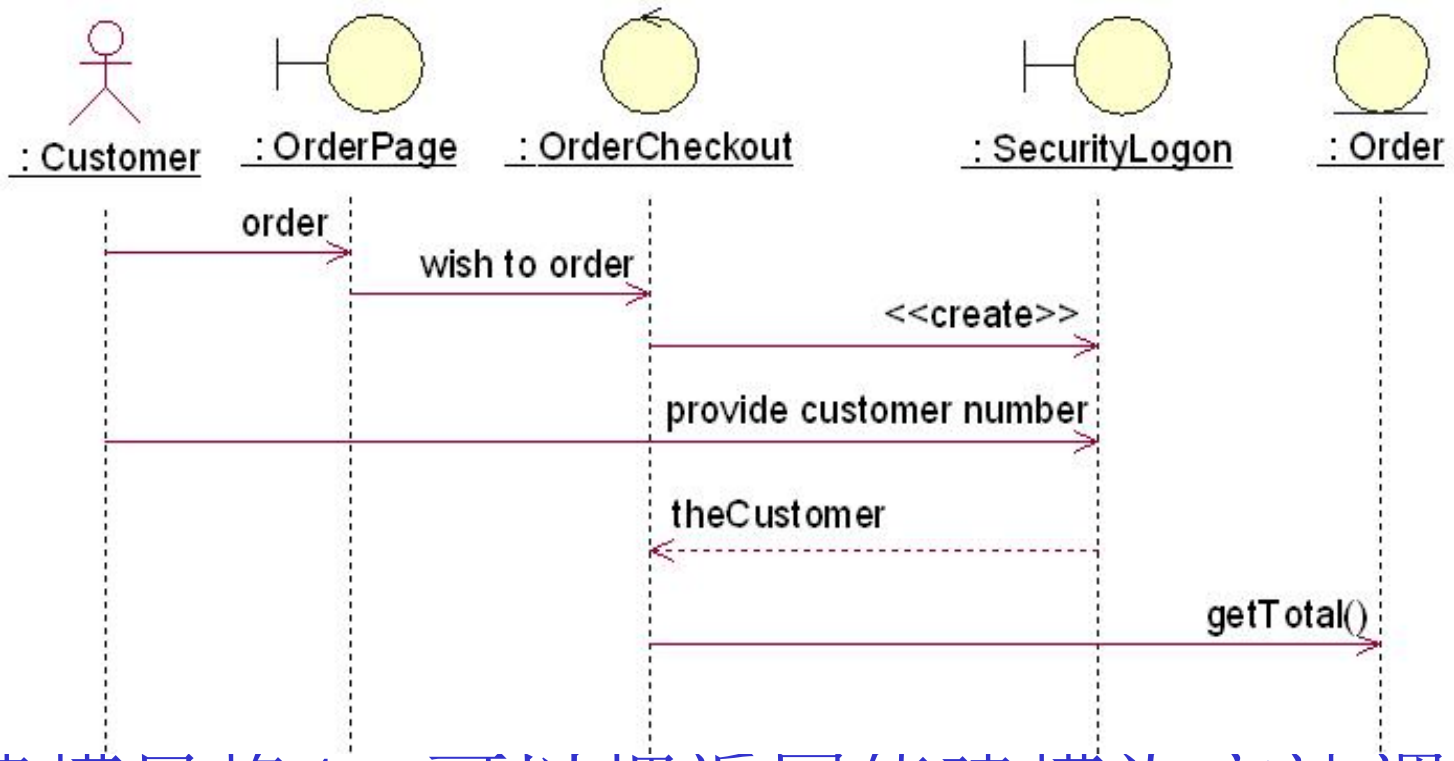
1. 确定交互过程的上下文(context)；
2. 识别参与交互过程的对象；
3. 为每个对象设置生命线，即确定哪些对象存在于整个交互过程中，哪些对象在交互过程中被创建和撤销；
4. 从引发这个交互过程的初始消息开始，在生命线之间从顶到下依次画出随后的各个消息；
5. 如果需要表示消息的嵌套，或/和表示消息发生时的时间点，则采用FOC；
6. 如果需要说明时间约束，则在消息旁边加上约束说明；
7. 如果需要，可以为每个消息附上前置条件和后置条件。

# 顺序图建模风格

- 建模风格1：把注意力集中于关键的交互。
  - 创建模型时要把注意力集中于系统的关键方面，而不要包括无关的细节。
  - 例如：如果顺序图是用于描述业务逻辑的，就没必要包括对象和数据库之间的详细交互，像save()和delete()这样的消息可能就足够了。或者简单地假定持久性(persistence)会适当地被处理，而不用考虑持久性方面的细节问题。

- 建模风格2：对于参数，优先考虑使用参数名而不是参数类型。
  - 例如，消息addDeposit(amount, target)比addDeposit(Currency, Account)传递了更多的信息。
  - 在消息中只使用类型信息不能传递足够的信息。
  - 参数的类型信息用UML类图捕获更好。
  - 建模风格只是建议，不是规定。如果只是需要表示“有某些东西要传递”这个事实，而不需要提供更多的细节，则可以指明参数的类型作为占位符。

- 建模风格3：不要对明显的返回值建模。
  - 例：创建安全登录对象的行为会导致生成一个顾客对象，这个是不明显的；而向订单对象发送请求其总数的消息，其返回值是显然的。




- 建模风格4：可以把返回值建模为方法调用的一部分。

# 顺序图常见问题分析

- **顺序图中消息的循环发送**
  - 在消息名字前加循环条件

例：

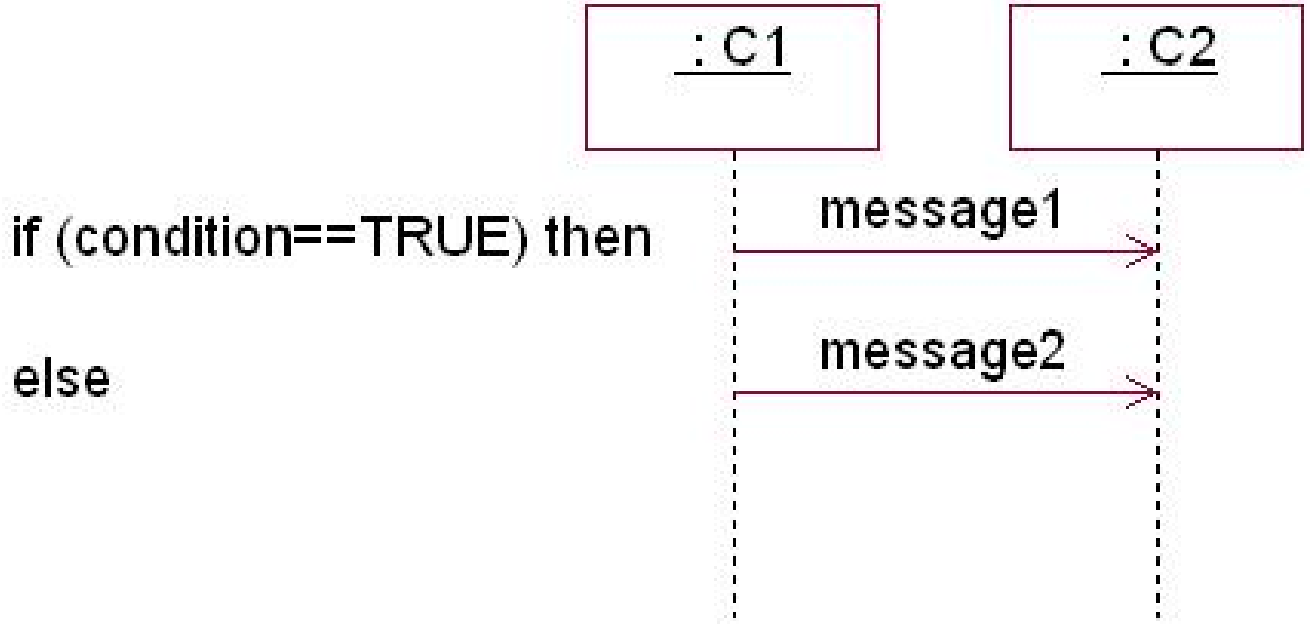
1.1 *[ for all order lines]: message1()

2.1 *[i:=1..n]: message2()

- **顺序图中消息的条件发送**

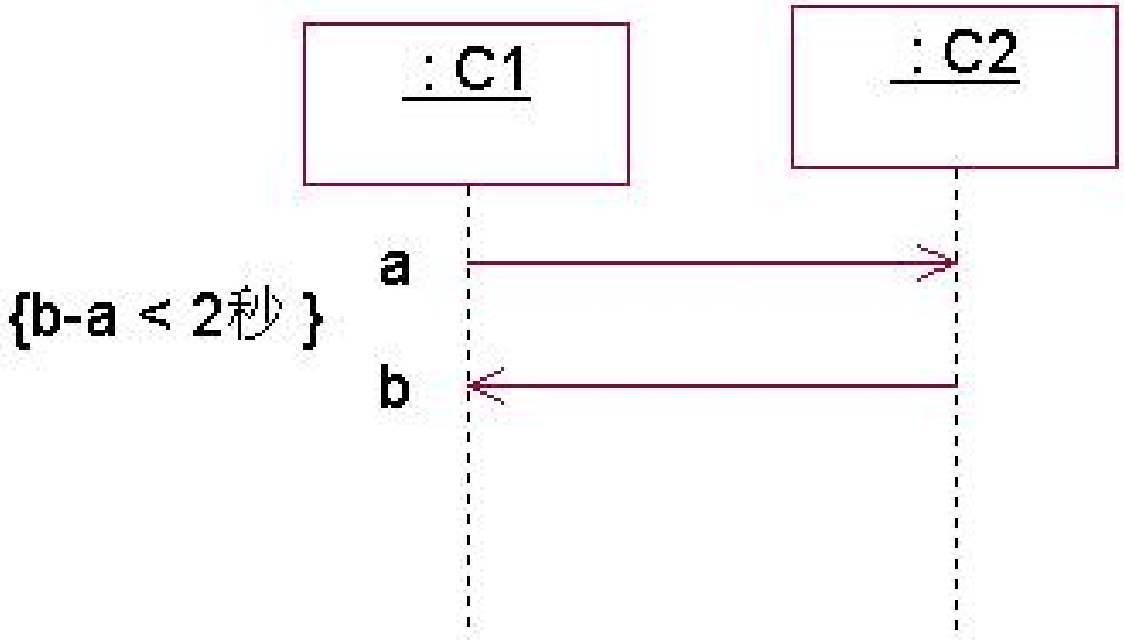
1. 在消息名字前加条件子句；
2. 使用文字说明；
3. 分成多个Sequence Diagram。

例：



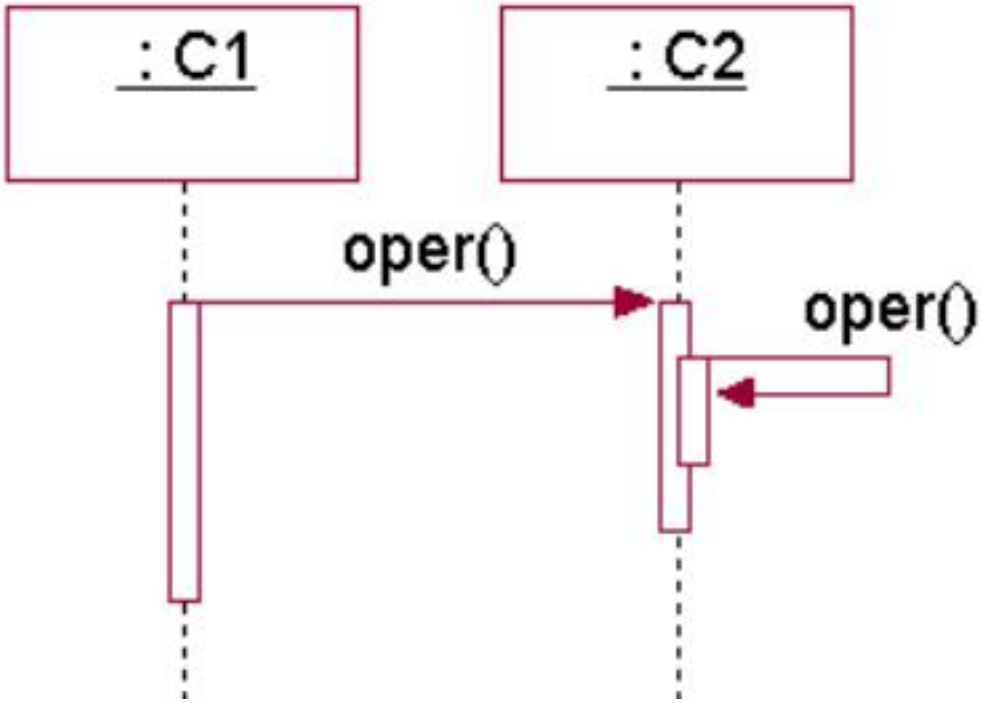
- **顺序图中时间约束的表示**
  - 用constraint(约束)来表示。

例：

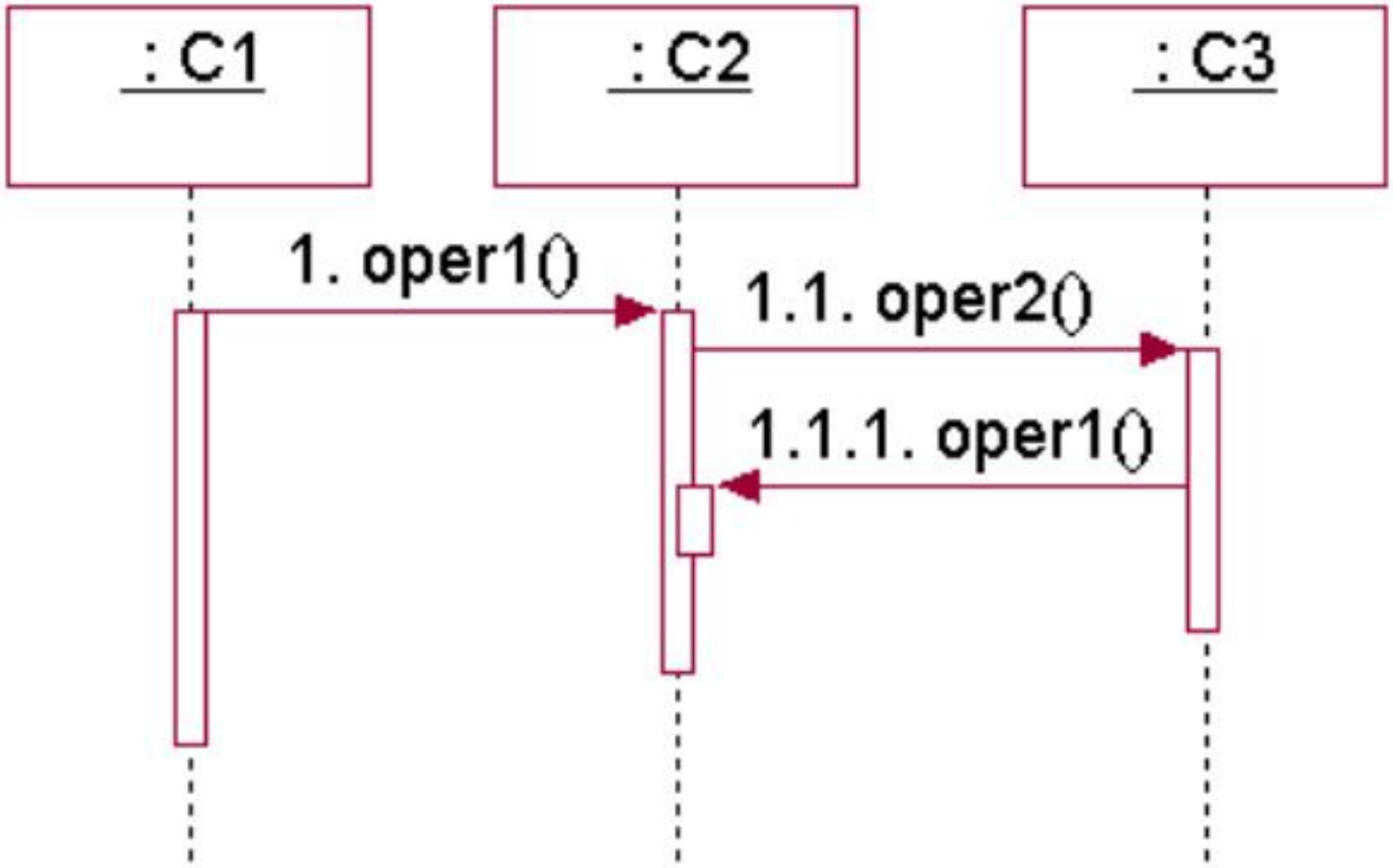


- **顺序图中递归的表示**
  - 利用嵌套的FOC表示

例1. 单个对象自身的递归。

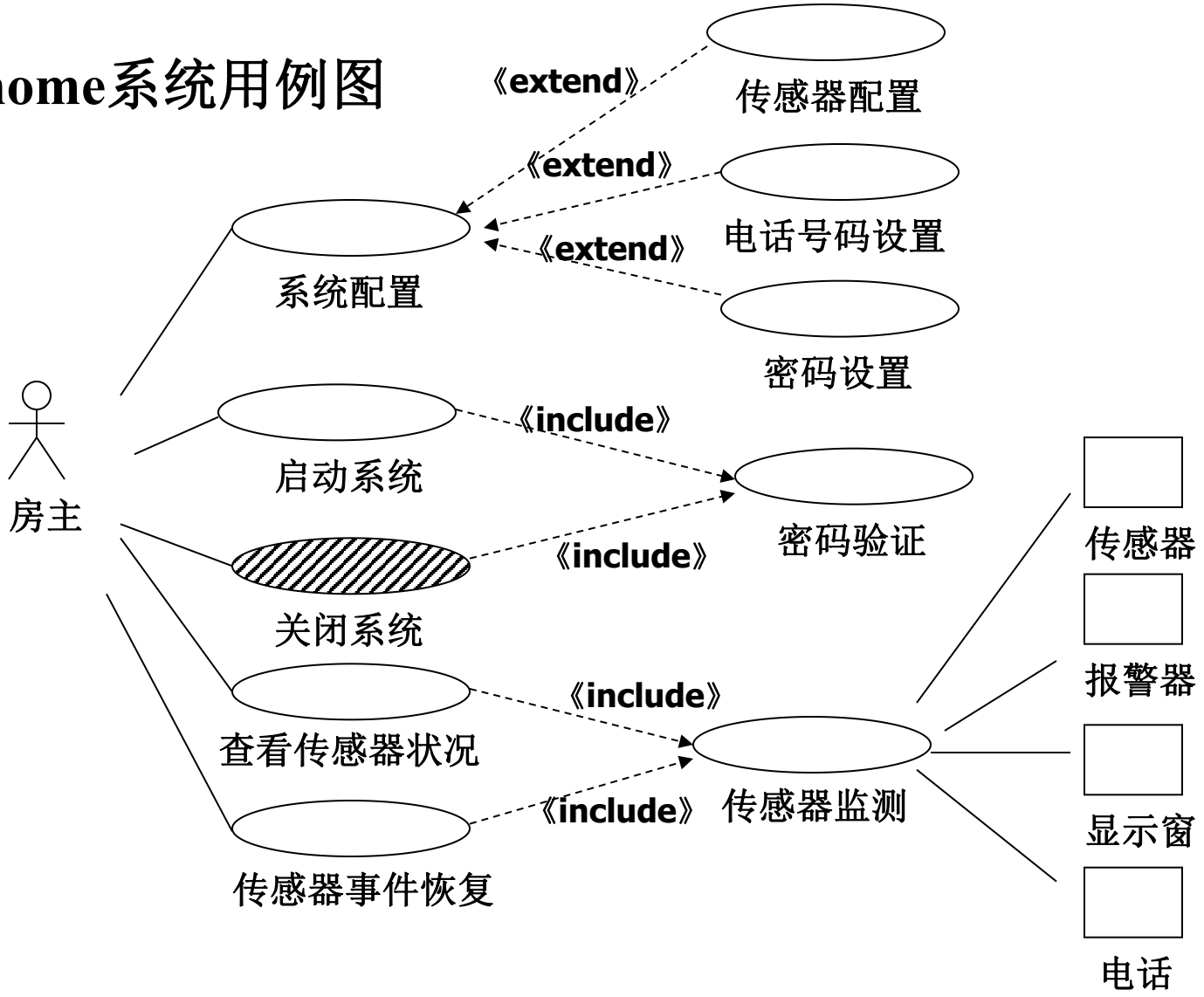


例2. 多个对象间相互递归调用的表示。

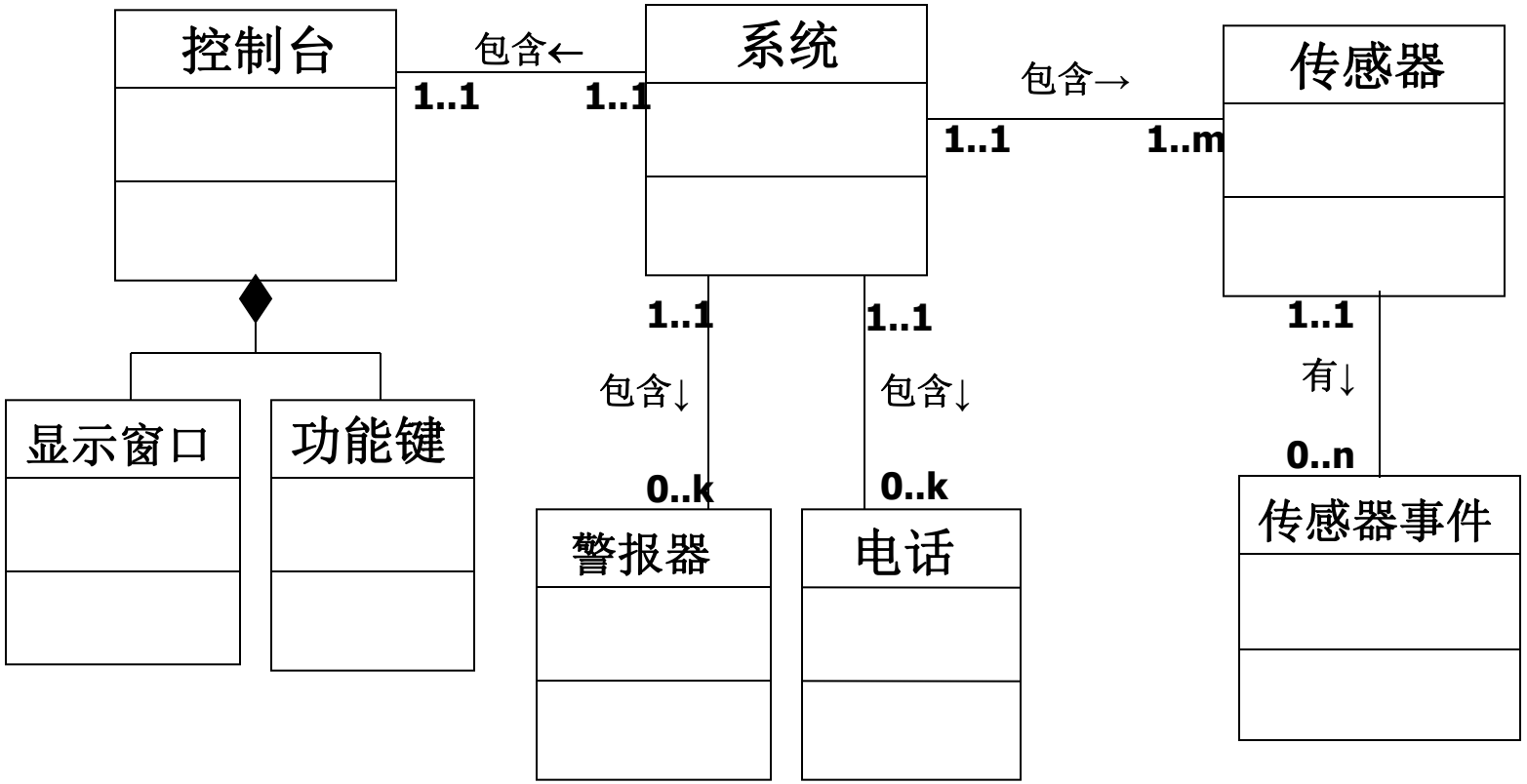


# 建立顺序图实例：Safehome

## Safehome系统用例图



# Safehome系统类图



# SafeHome关闭系统用例的顺序图