

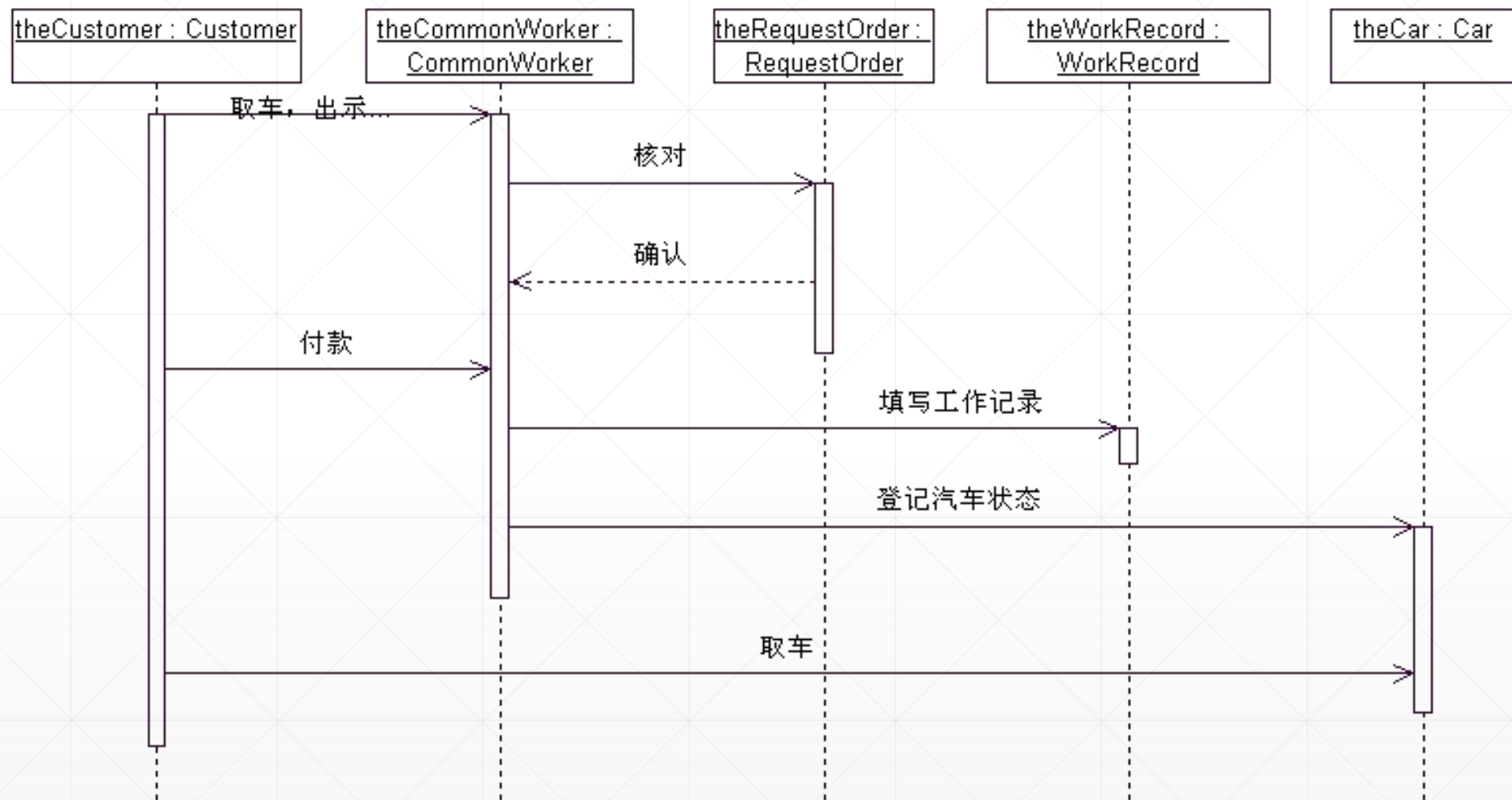
什么是顺序图？

顺序图是强调消息时间顺序的交互图。

顺序图描述了对对象之间传送消息的时间顺序，用来表示用例中的行为顺序。

顺序图将交互关系表示为一个二维图。即在图形上，顺序图是一张表，其中显示的对象沿横轴排列，从左到右分布在图的顶部；而消息则沿纵轴按时间顺序排序。创建顺序图时，以能够使图尽量简洁为依据布局。

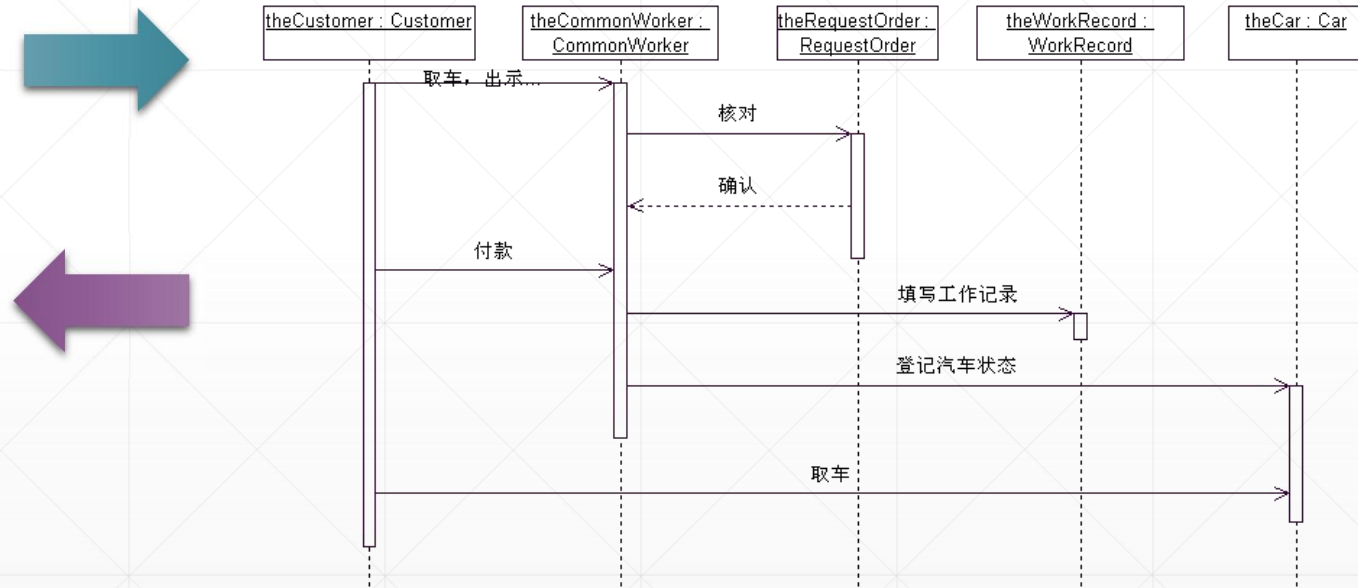
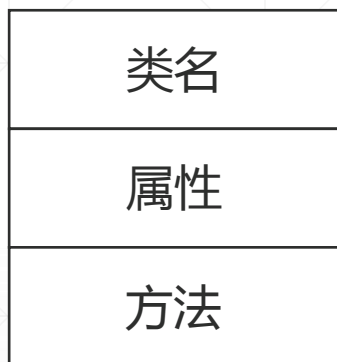
购买小车的顺序图示例



什么时候会用到顺序图？

用例

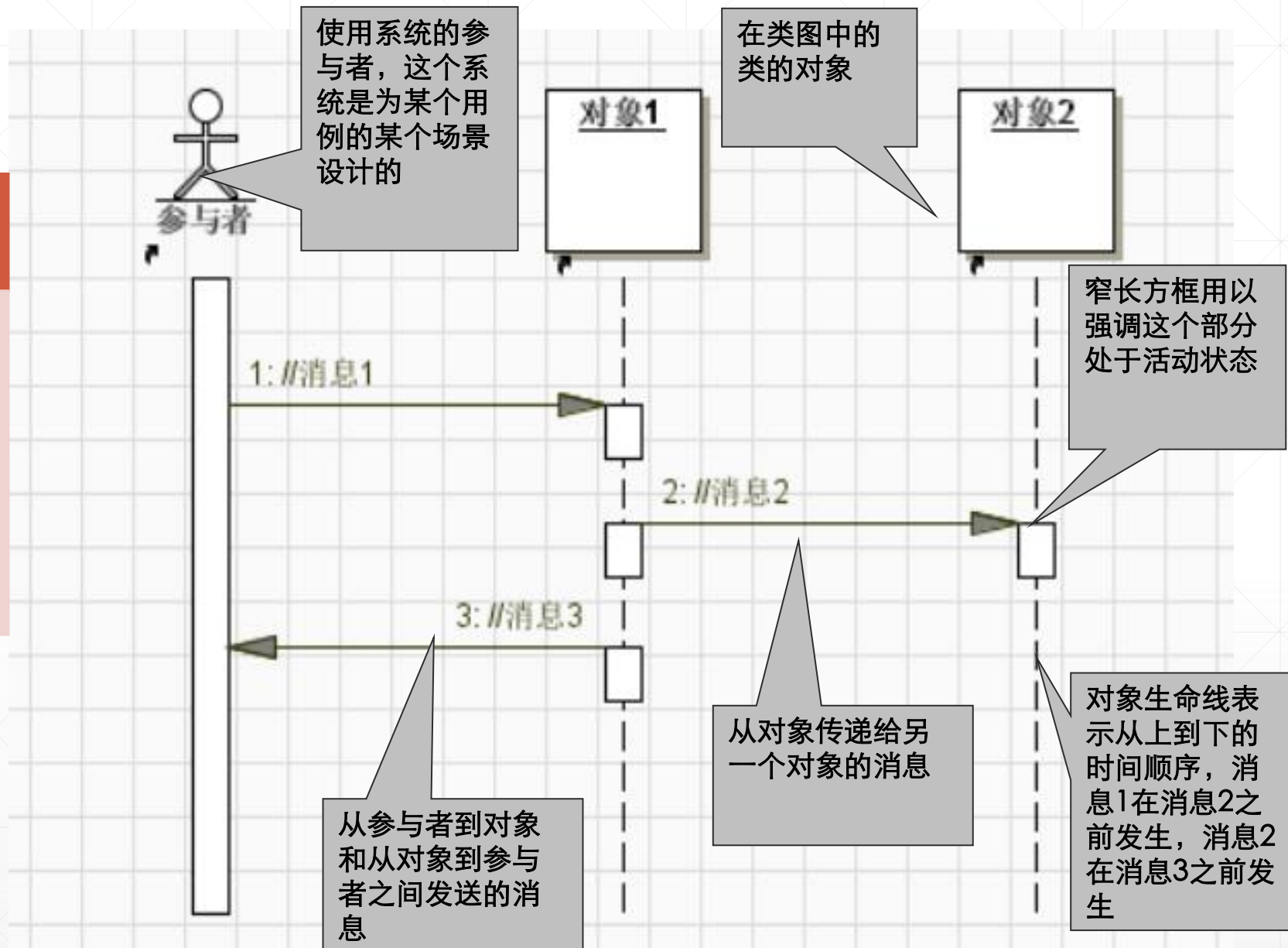
涉及多个类



顺序图的组成

顺序图包含4个元素：

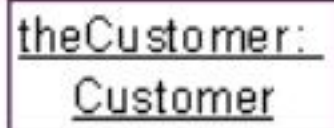
- 对象 (Object)
- 生命线 (Lifeline)
- 消息 (Message)
- 激活 (Activation)



对象

顺序图中对象的符号和对象图中对象所用的符号一样。

将对象置于顺序图的顶部意味着在交互开始的时候对象就已经存在了，如果对象的位置不在顶部，那么表示对象是在交互的过程中被创建的。



theCustomer:
Customer

The image shows a UML object notation symbol. It consists of a rectangular box with a double border. The top part of the box contains the text 'theCustomer:' and the bottom part contains the text 'Customer'. Both parts are underlined.

对象

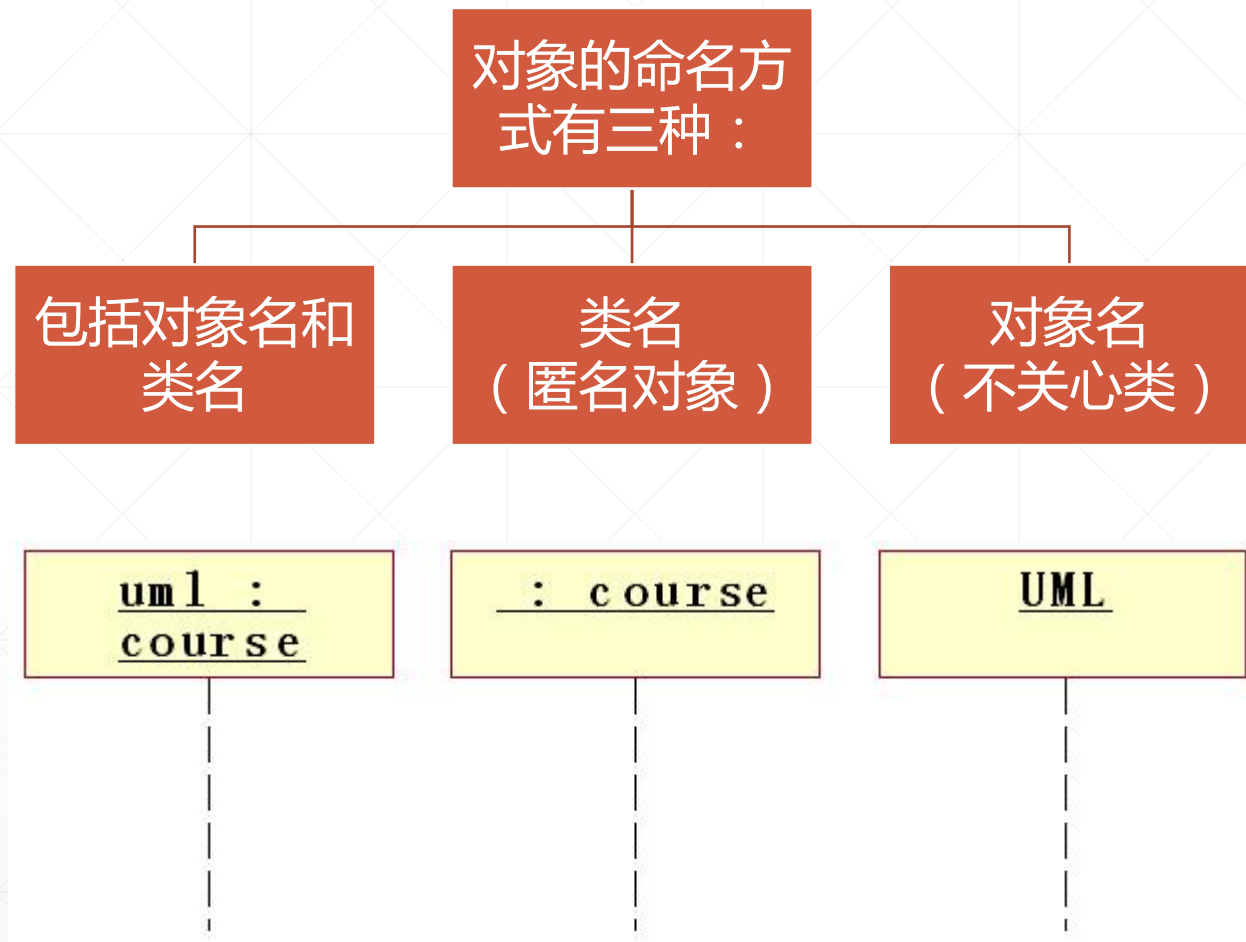
参与者和对象按照从左到右的顺序排列

一般最多两个参与者，他们分列两端。启动这个用例的参与者往往排在最左边；接收消息的参与者则排在最右端；

对象从左到右按照重要性排列或按照消息先后顺序排列。



对象



生命线

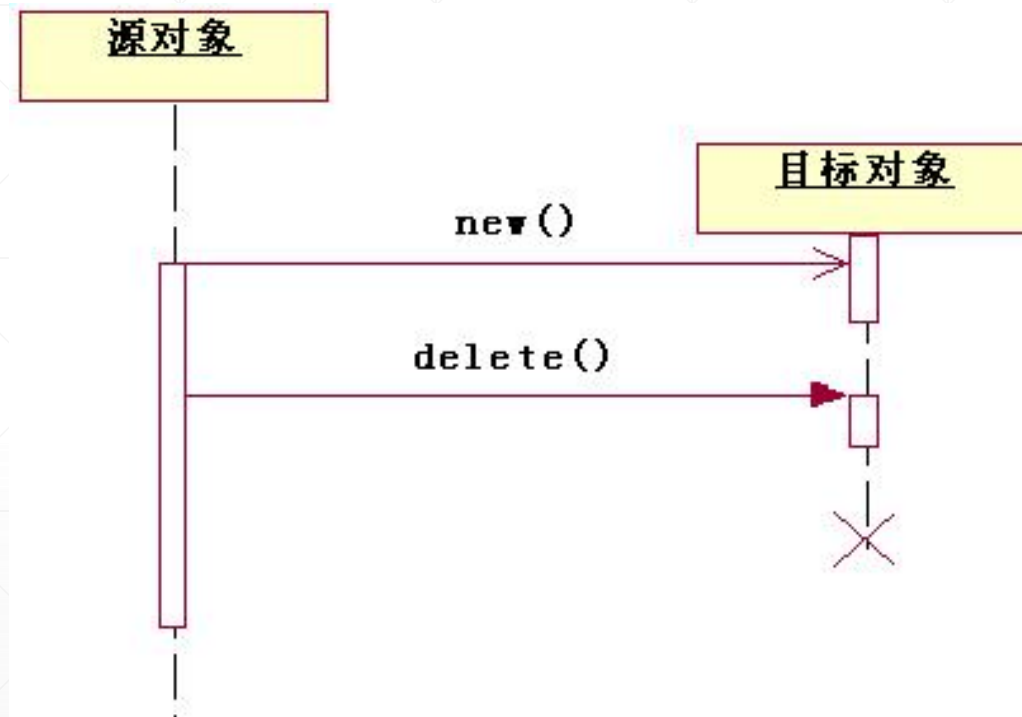
每个对象都有自己的生命线，用来表示在该用例中一个对象在一段时间内的存在

生命线使用垂直的虚线表示

如果对象生命期结束，则用注销符号表示

对象默认的位置在图顶部，表示对象在交互之前已经存在

如果是在交互过程中由另外的对象所创建，则位于图的中间某处。



激活

激活表示该对象被占用以完成某个任务，去激活指的则是对象处于空闲状态、在等待消息。

在UML中，为了表示对象是激活的，可以将该对象的生命线拓宽成为矩形。其中的矩形称为激活条(期)或控制期，对象就是在激活条的顶部被激活的，对象在完成自己的工作后被去激活。



激活期

当一条消息被传递给对象的时候，它会触发该对象的某个行为，这时就说该对象被激活了。

在UML中，激活用一个在生命线上的细长矩形框表示。

矩形本身被称为对象的激活期或控制期，对象就是在激活期顶端被激活的。

激活期说明对象正在执行某个动作。当动作完成后，伴随着一个消息箭头离开对象的生命线，此时对象的一个激活期也宣告结束。



消息

面向对象方法中，消息是对象间交互信息的主要方式。

- 在任何一個软件系统中，对象都不是孤立存在的，它们之间通过消息进行通信。
- 消息是用来说明顺序图中不同活动对象之间的通信。因此，消息可以激发某个操作、创建或撤销某个对象。

结构化程序设计中，模块间传递信息的方式主要是过程（或函数）调用。

- 对象A向对象B发送消息，可以简单地理解为对象A调用对象B的一个操作（operation）。

在顺序图中，消息是由从一个对象的生命线指向另一个对象的生命线的直线箭头来表示的，箭头上还可以表明要发送的消息名及序号。

- 顺序图中消息编号可显示，也可不显示。协作图中必须显示。

顺序图中，尽力保持消息的顺序是从左到右排列的。在各对象之间，消息的次序由它们在垂直轴上的相对位置决定。

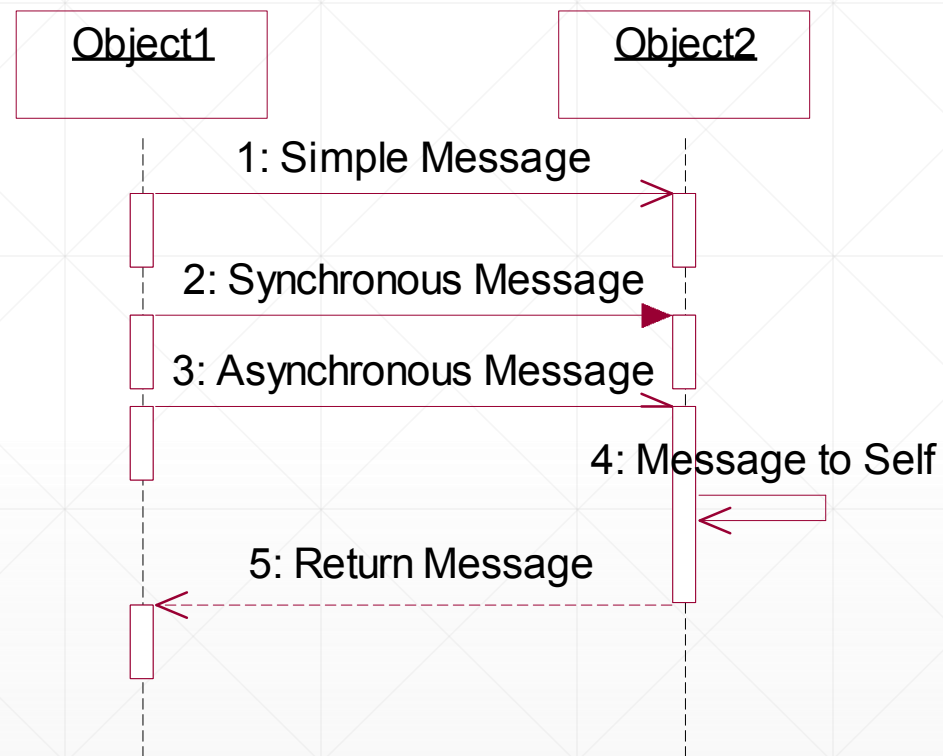
- 一个顺序图的消息流开始于左上方，消息2的位置比消息1低，这意味着消息2的顺序比消息1要迟。因为西方的阅读习惯是从左到右。

消息

在UML中，消息使用箭头来表示，箭头的类型表示了消息的类型。

进行顺序图建模时，所用到的消息主要包括以下几种类型：

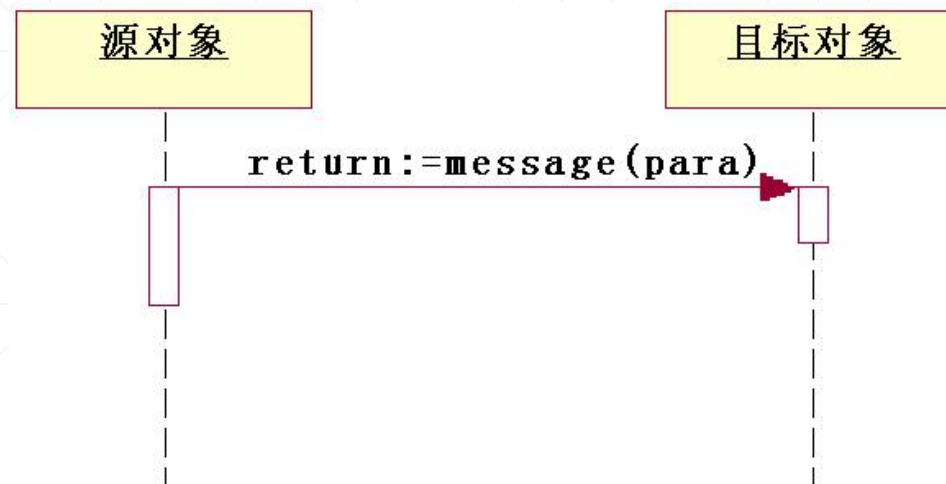
- 简单消息（Simple Message）
- 同步消息（Synchronous Message）
- 异步消息（Asynchronous Message）
- 反身消息（Message to Self）
- 返回消息（Return Message）



同步消息

同步消息最常见的情况是调用，即消息发送者对象在它的一个操作执行时调用接收者对象的一个操作，此时消息名称通常就是被调用的操作名称。

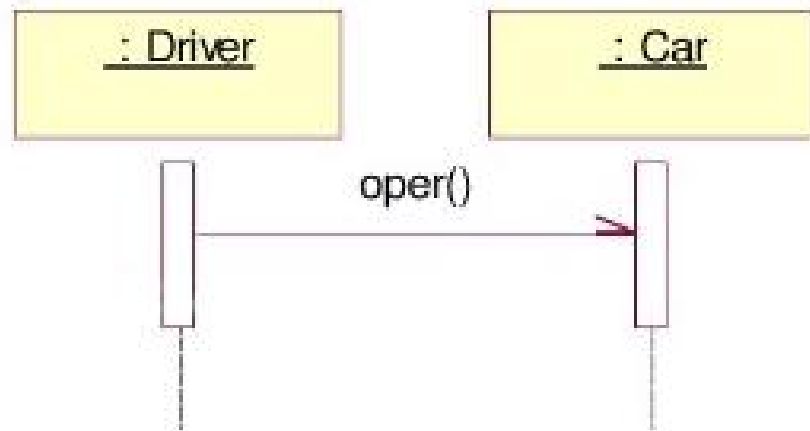
当消息被处理完后，可以回送一个简单消息，或者是隐含的返回。



异步消息

异步消息表示发送消息的对象不用等待回应的返回消息，即可开始另一个活动。

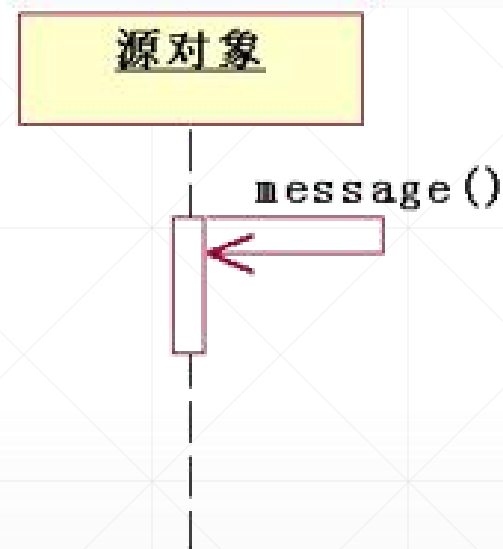
异步消息在某种程度上规定了发送方和接收方的责任，即发送方只负责将消息发送到接收方，至于接收方如何响应，发送方则不需要知道。对接收方来说，在接收到消息后它既可以对消息进行处理，也可以什么都不做。



反身消息

顺序图建模过程中，一个对象也可以将一个消息发送给它自己，这就是反身消息。

- 如果一条消息只能作为反身消息，那么说明该操作只能由对象自身的行为触发。
- 这表明该操作可以被设置为private属性，只有属于同一个类的对象才能够调用它。
- 在这种情况下，应该对顺序图进行彻底的检查，以确定该操作不需要被其他对象直接调用。

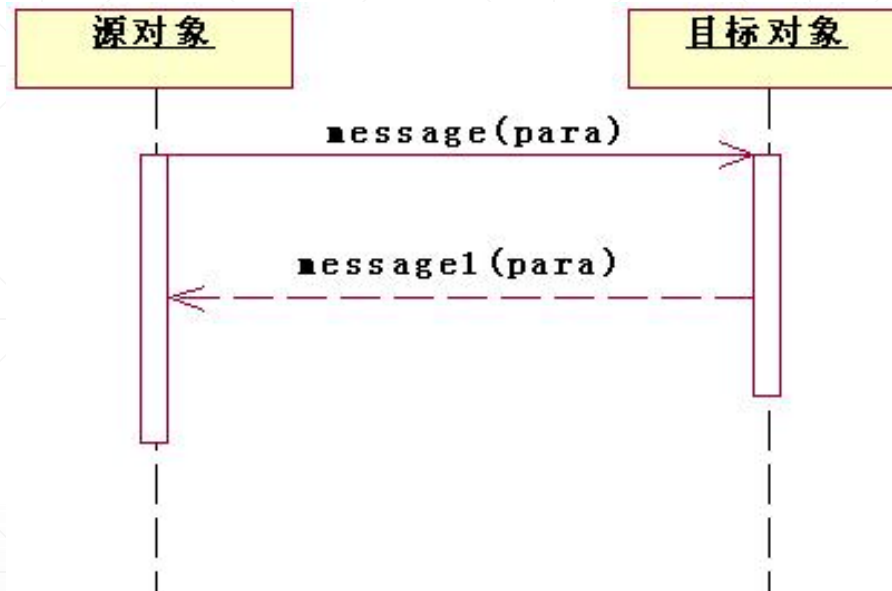


返回消息

返回消息是顺序图的一个**可选择部分**，它表示控制流从过程调用的返回。

返回消息**一般可以缺省**，隐含表示每一个调用都有一个配对的调用返回。

是否使用返回消息依赖于建模的具体/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。

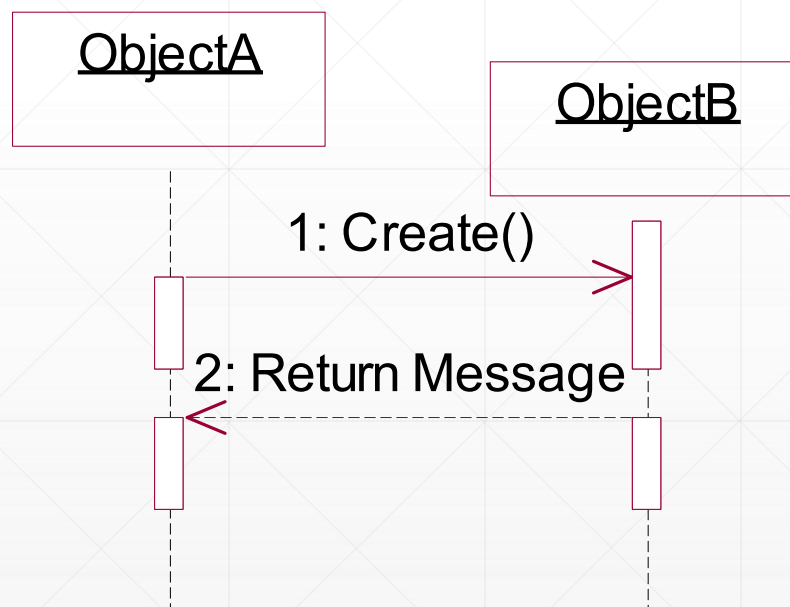
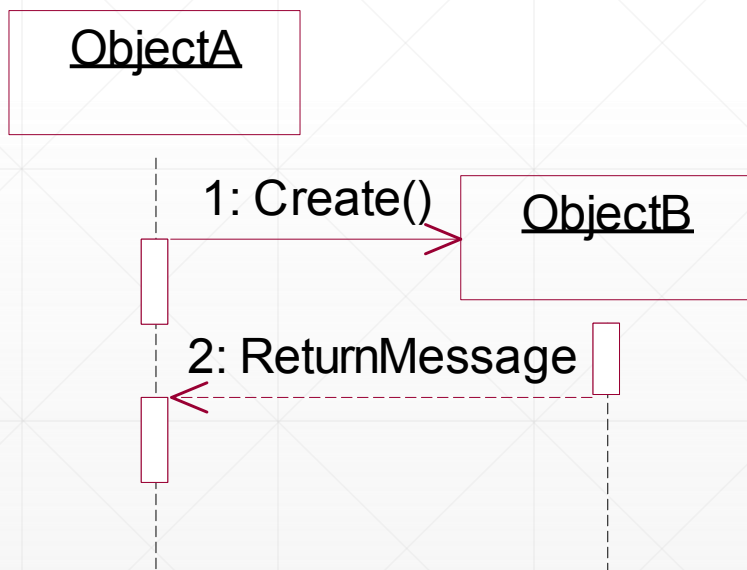


对象的创建和撤销

对象的创建有两种情况：

- 顺序图中的对象的默认位置是在图的顶部，如果对象在这个位置上，那么说明在发送消息时，该对象就已经存在了；
- 如果对象是在执行的过程中创建的，那么它的位置应该处在图的中间部分。

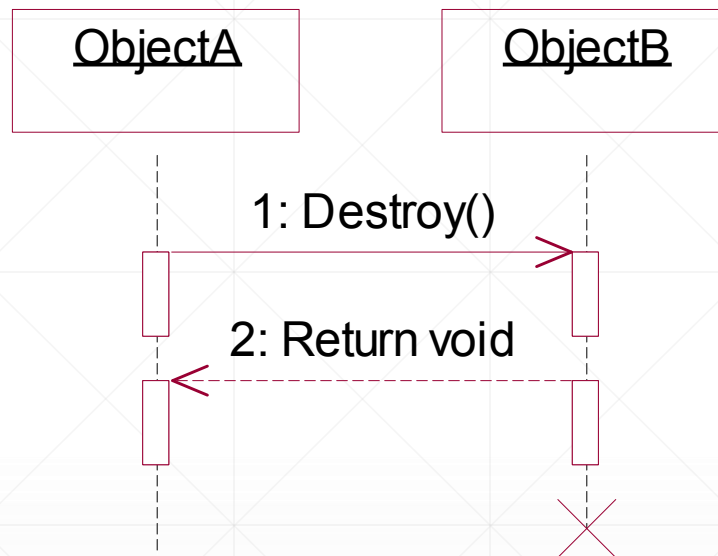
对象的创建有两种方法：



对象的创建和撤销

对象的撤销有两种情况：

- 在处理新创建的对象，或顺序图中的其他对象时，都可以发送“destroy”消息来撤销对象。
- 要想说明某个对象被撤销，需要在被撤销对象的生命线末端放一个“×”符号进行标识。

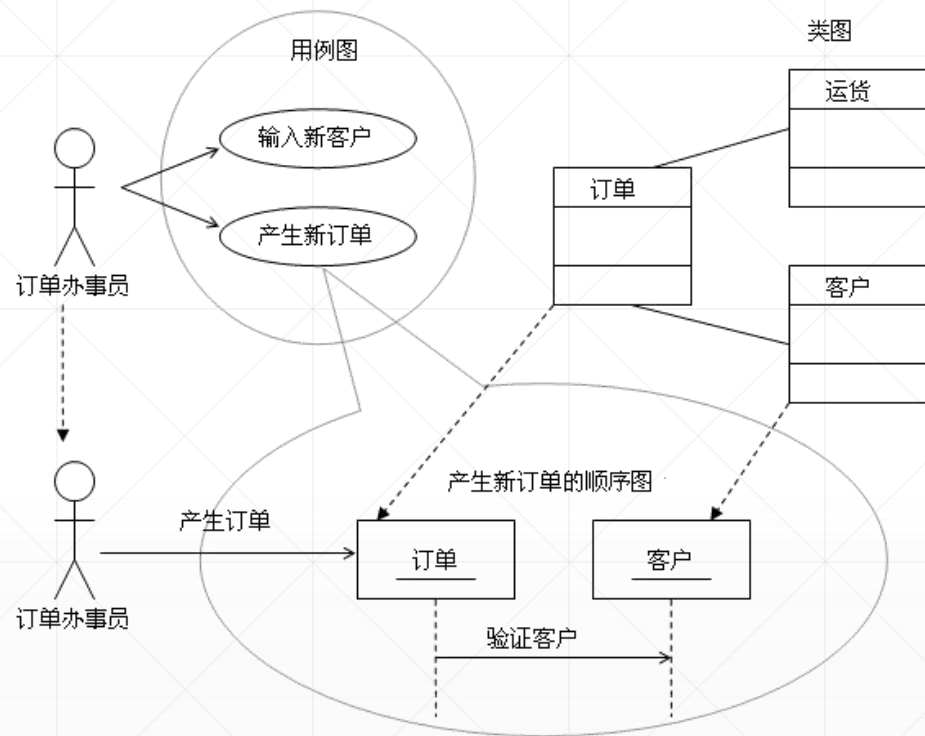


顺序图和用例

顺序图的主要用途之一是用来为某个用例的泛化功能提供其所缺乏的解释，即把用例表达的要求转化为更进一步的精细表达。

用例常常被细化为一个或多个顺序图。

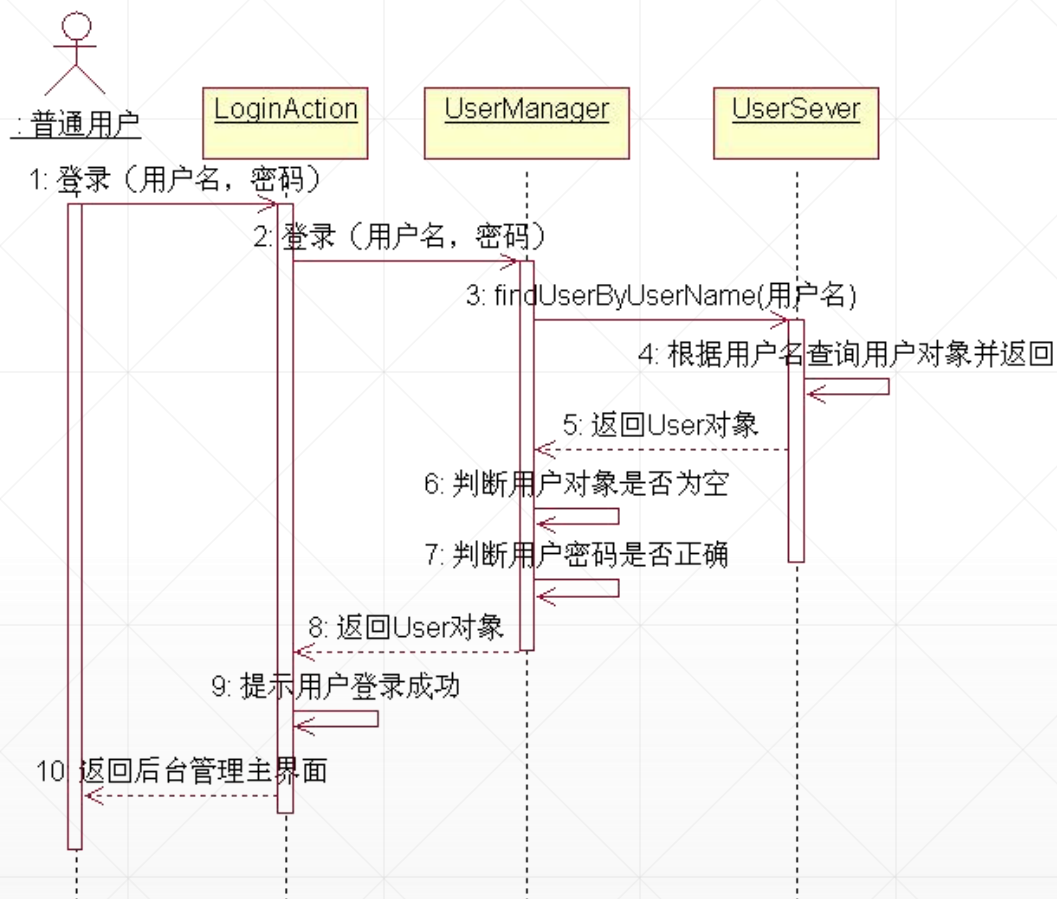
顺序图除了在设计新系统方面的用途之外，它还能用来记录一个存在于系统的对象现在如何交互。



顺序图和用例

登录用例：

- 用户将用户名和密码提交给LoginAction
- 由LoginAction调用UserManager
- UserManager到用户数据库User Server中查找用户对象并返回
- 由UserManager判断用户名是否为空、密码是否正确
- 然后将User对象返回
- 返回后台登录主界面。



顺序图建模

对系统动态行为建模的过程中，当强调按时间展开信息的传送时，一般使用顺序图建模技术。

一个单独的顺序图只能显示一个控制流。

- 一般情况下，一个完整的控制流是非常复杂的，要描述它需要创建很多交互图（包括顺序图和协作图），一些图是主要的，另一些图用来描述可选择的路径和一些例外，再用一个包对它们进行统一的管理。

顺序图建模参考策略

设置交互的语境

这些语境可以是系统、子系统、类、用例和协作的一个脚本。

对象从左到右

识别对象在交互语境中所扮演的角色，根据对象的重要性及相互关系，将其从左至右放置在顺序图的顶部。

设置每个对象的生命线

通常情况下，对象存在于整个交互过程中，但它们也可以在交互过程中创建和撤销。对于这类对象，在适当的时刻设置它们的生命线，并用适当的构造型消息显示地说明它们的创建和撤销。

消息自上而下

从引发某个消息的信息开始，在生命线之间画出从顶到底依次展开的消息，显示每个消息的内容标识。

设置对象的激活期

可视化消息的嵌套或可视化实际计算发生时的时间点。

时间和空间约束

如果需要设置时间或空间的约束，可以为每个消息附上合适的时间和空间约束。

前置和后置条件

如果需要形式化的说明某控制流，可以为每个消息附上前置和后置条件。

建立顺序图的步骤

确定交互的范围；

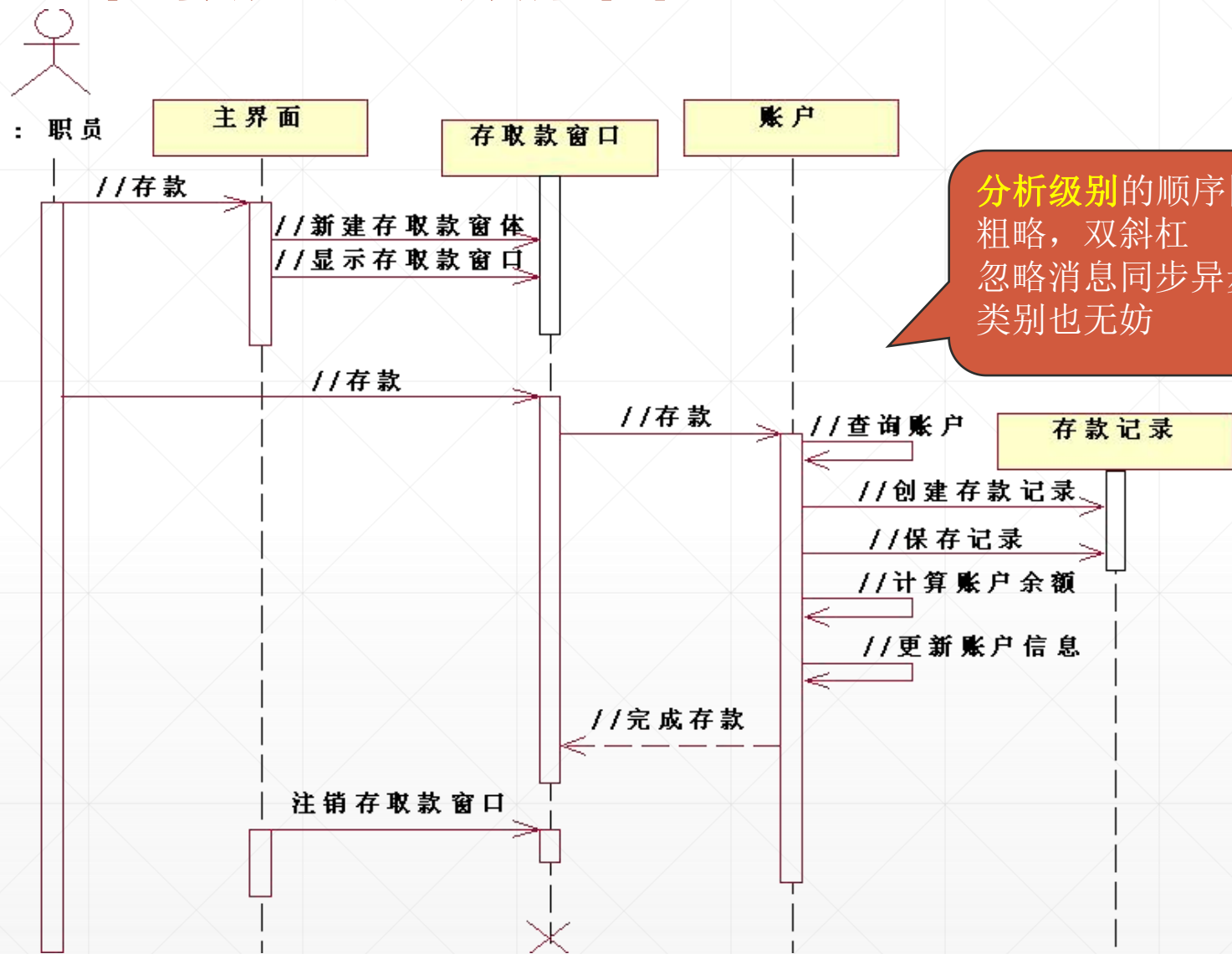
识别参与交互的对象和活动者；

设置对象生命线开始和结束；

设置消息；

细化消息。

顺序图示例：存款用例的顺序图

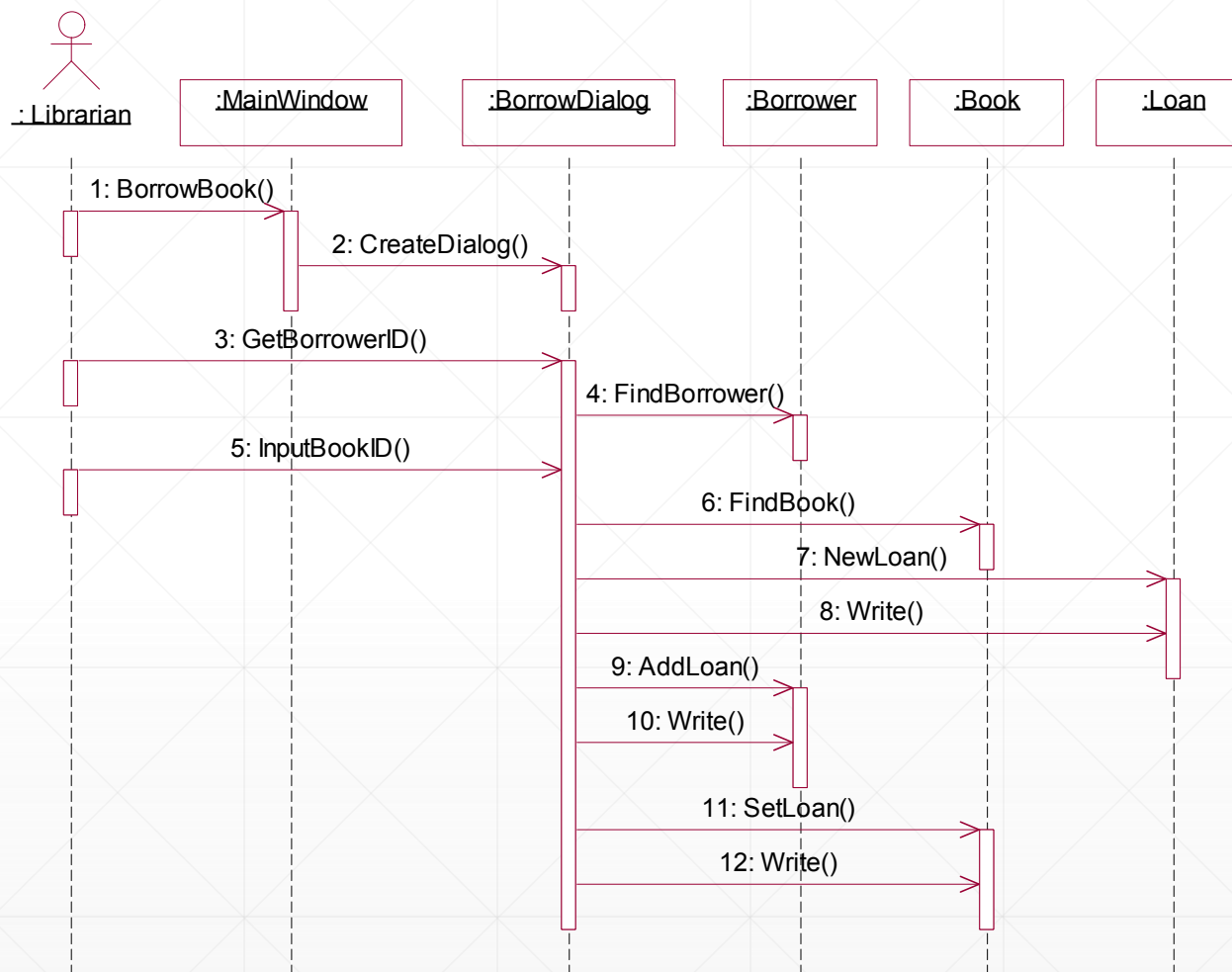


分析级别的顺序图，
粗略，双斜杠
忽略消息同步异步
类别也无妨

顺序图示例：借阅图书用例的顺序图

借阅图书的过程为：

- 图书管理员选择菜单项“借阅图书”，弹出BorrowDialog对话框；
- 图书管理员在该对话框中输入借阅者信息
- 然后由系统查询数据库，以验证该借阅者的合法性
- 若借阅者合法，则再由图书管理员输入所要借阅的图书信息，系统记录并保存该借阅信息。





授课教师：蓝天 电子邮箱：lantian1029@uestc.edu.cn