6.3 关于 UML 的图：表达格式 - 模型表达工具

- UML 为不同抽象层提供了 6 种可对系统静态部分建模的图形工具：

① 类图；② 构件图；③ 组合结构图；

④ 对象图；⑤ 部署图；⑥ 制品图。

可将系统的静态方面看作是系统相对稳定的骨架的表示，正如房屋的静态方面是由墙、门、窗、管子、电线等事物的布局组成一样

以上图形工具解释如下：

① 类图：类图显示了类（及其接口）、类的内部结构以及与其他类的联系。是面向对象分析与设计所得到的最重要的模型。

② 构件图：在转入实现阶段之前，可以用它表示如何组织构件。构件图描述了构件及构件之间的依赖关系。

③ 组合结构图展示了类或协作的内部结构。

④ 对象图：展示了一组对象以及它们之间的关系。用对象图说明在类图中所发现的事物的实例的数据结构和静态快照。

⑤ 部署图：部署图展示运行时进行处理的结点和在结点上生存的制品的配置。部署图用来对系统的静态部署视图建模 .

⑥ 制品图：展示了一组制品以及其间依赖关系。利用制品图可以对系统的静态实现视图建模。

♦ UML 为不同抽象层提供了 7 种可对系统动态部分建模的图形工具：

① 用况图：需求模型。

② 状态图：当对象的行为比较复杂时，可用状态图作为辅助模型描述对象的状态及其状态转移，从而更准确地定义对象的操作。

可将系统的动态方面看作是对系统变化部分的表示。正如房屋的动态方面包含了气流和人在房间中的走动一样

③ 活动图：注重从活动到活动的控制流，可用来描述对象的操作流程，也可以描述一组对象之间的协作行为或用户的业务流程。

④ 顺序图：注重于消息的时间次序。可用来表示一组对象之间的交互情况。

⑤ 通信图：注重于收发消息的对象的组织结构。可用来表示一组对象之间的交互情况。

⑥ 交互概观图：用于描述系统的宏观行为，是活动图和顺序图的混合物。

⑦ 定时图：用于表示交互，它展现了消息跨越不同对象或角色的实际时间，而不仅仅关心消息的相对顺序。

6.3.1 静态模型表达工具 - 类图

① 定义：类图显示了类（及其接口）、类的内部结构以及与其他类的联系，**是面向对象分析和设计所得到的最重要的模型 .**

作用：●可视化地表达系统的静态结构模型。

② 类图的内容：

- 通常包含：●类；●接口；●依赖、泛化和关联关系等
- 还可以包含注解和约束，以及包或子系统，甚至，可包含一个实例，以便使其可视化。

注：这些成分，确定了所表达系统的各种形态 .

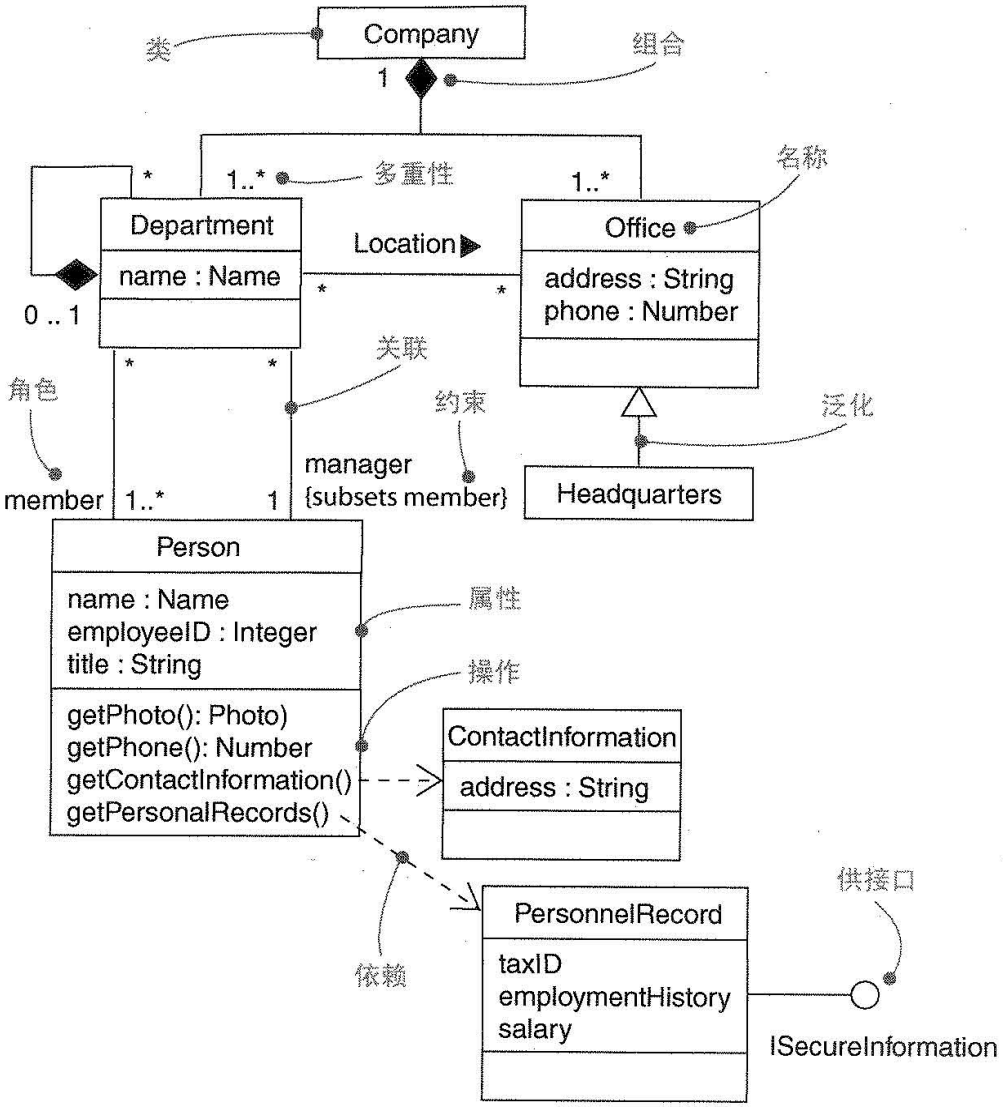


图 1 类图示例 1

例如：图 2 显示了某学校的信息系统的类图，该图表明，学校有多个系，每个系有多名教员，每位教员教授多门课程；该学校还有许多学生，每位学生要参加多门课。

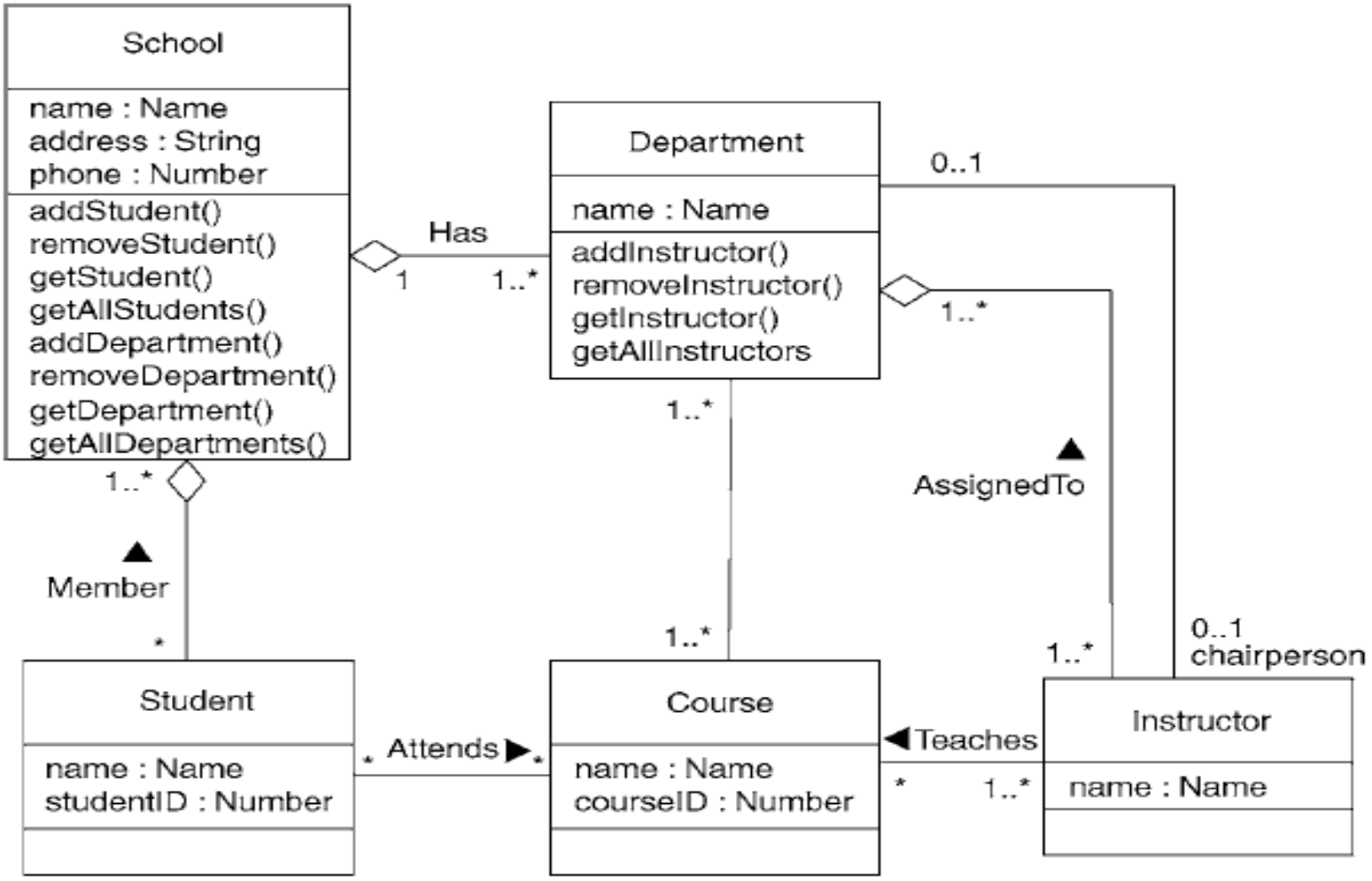


图 2 类图示例 2

③ 类图的一般用法

类图主要用于对系统的静态视图进行建模（投影），支持表达系统的功能需求，即系统提供给最终用户的服务。

创建类图包括以下四方面工作：

❶ 对系统中的概念（词汇）建模，形成类图中的基本元素

使用 UML 中的术语“类”，来抽象系统中各个组成部分，包括系统环境．然后，确定每一类的责任，最终形成类图中的模型元素．

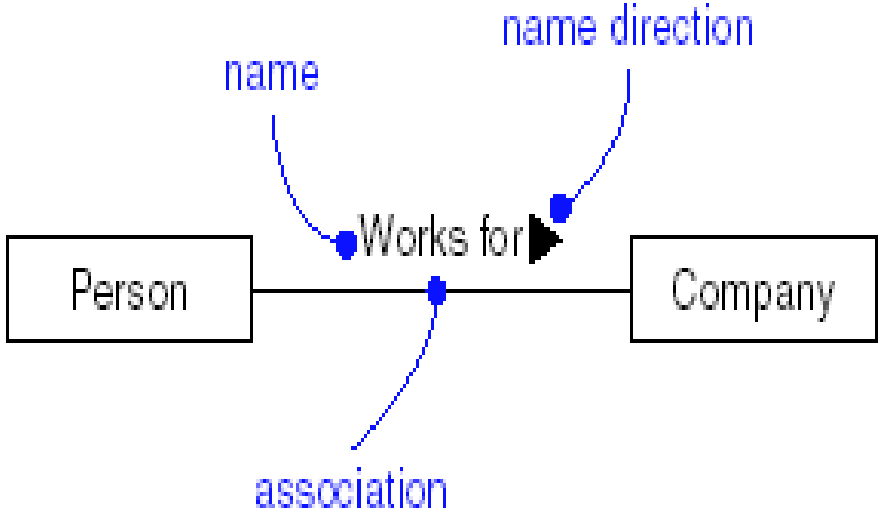
❷ 对待建系统中的各种关系建模，形成该系统的初始类图

.

使用 UML 中表达关系的术语，例如关联、泛化和依赖等来抽象系统中各成分之间的关系，形成该系统的初始类图 .

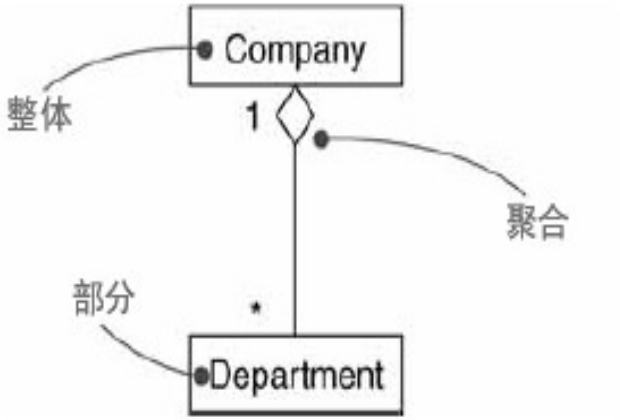
●当用关联关系建模时，是在对相互同等的两个类建模．给定两个类间的关联，则这两个类以某种方式相互依赖，并且常常从两边都可以导航．

A）对于每一对类，如果需要从一个类的对象到另一个类的对象导航，就要在这两个类之间建立一个关联；



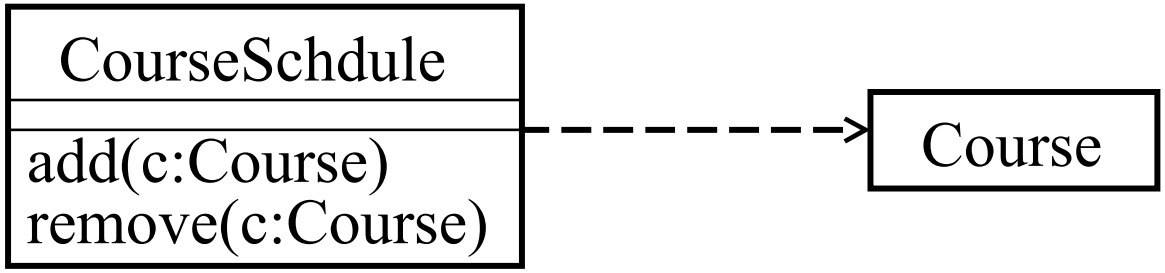
B）对于每一对类，如果一个类的对象要与另一个类的相互交互，而后者不作为前者的过程局部变量或操作参数，就要在这两个类之间建立一个关联；

C）如果关联中的一个类与另一端的类相比，前者在结构或者组织上是一个整体，后者看来像它的部分，则在靠近整体的一端用一个菱形对关联修饰，从而将其标记为聚合 .

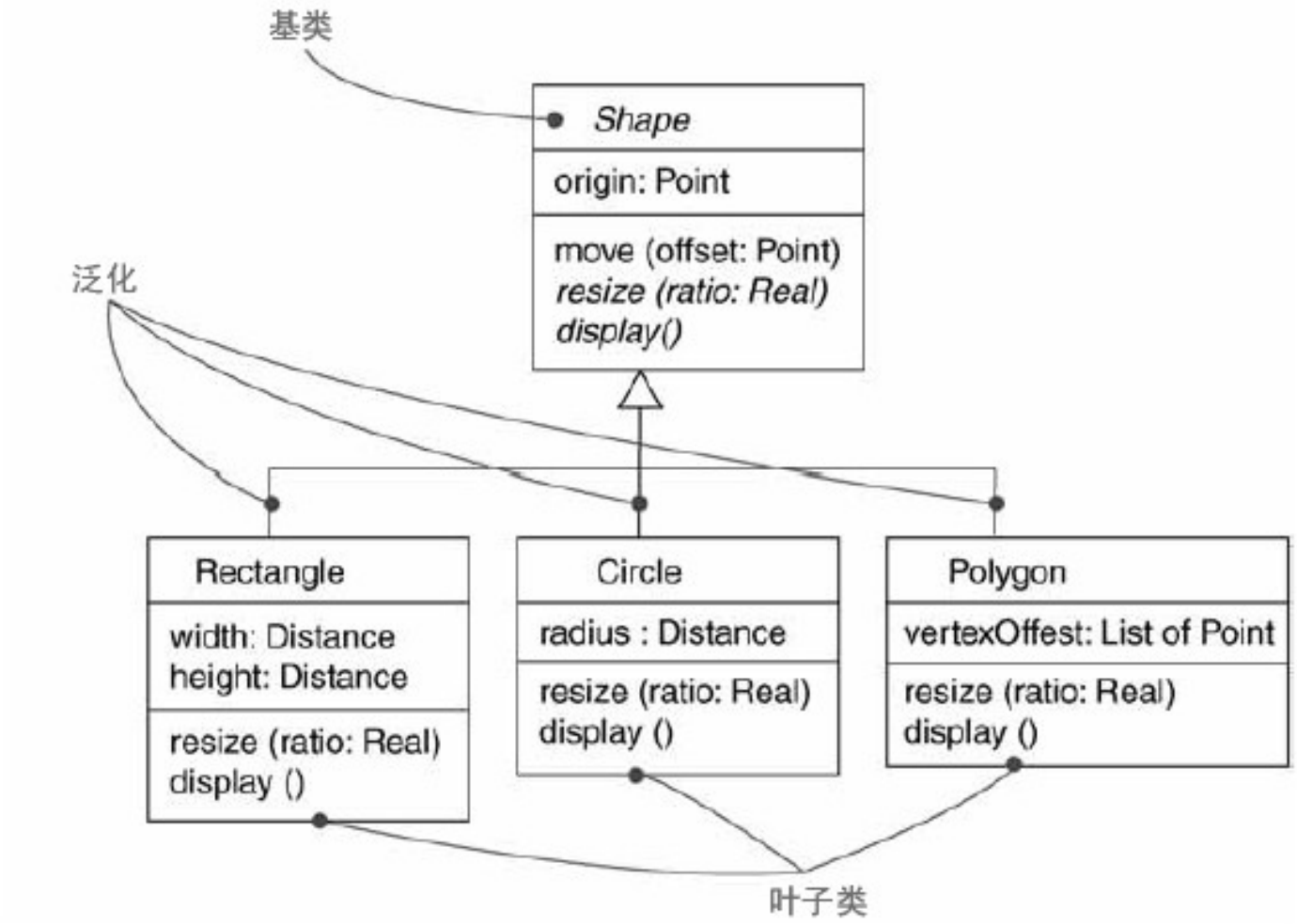


D）对于每一个关联，都要说明其多重性（特别当多重性不为 * 时，其中 * 是默认的多重性）

- 依赖关系是使用关系，常见的依赖关系是两个类之间的连接，其中一个类只是使用另一个类作为它的操作参数
  - 创建一个依赖，从含有操作的类指向被该操作用作参数的类.



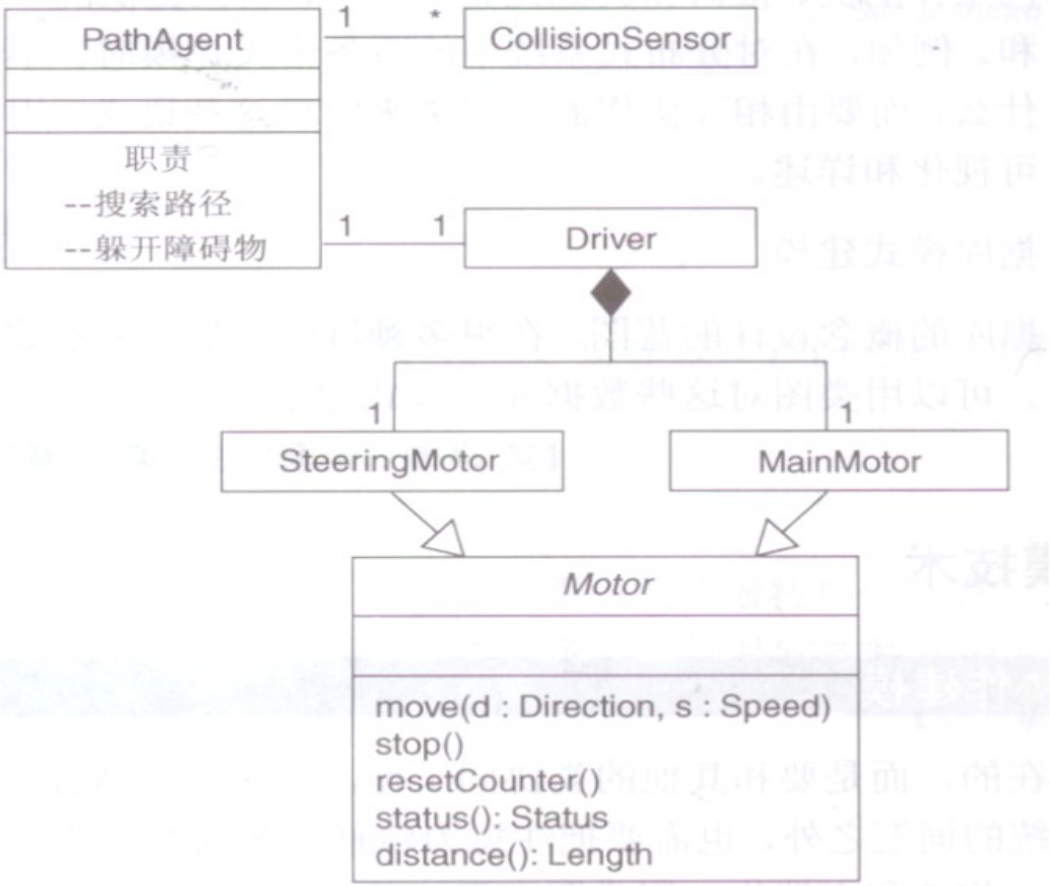
- 泛化关系是“is-a-kind-of” 关系，在对系统的词汇建模中，经常遇到结构或行为上与其他类相似的类，可以提取所有共同的结构特征和行为特征，并把它们提升到较一般的类中，特殊类继承这些特征
  - 给定一组类，寻找两个或以上的类的共同责任、属性和操作
  - 把这些共同的责任、属性和操作提升为较一般的类
  - 画出从每个特殊类到它的较一般的父类的泛化关系，用以表示较特殊的类继承较一般的类


基类
Shape
origin: Point
move (offset: Point)
resize (ratio: Real)
display()
泛化
Rectangle
width: Distance
height: Distance
resize (ratio: Real)
display ()
Circle
radius : Distance
resize (ratio: Real)
display ()
Polygon
vertexOffest: List of Point
resize (ratio: Real)
display ()
叶子类

❸ 模型化系统中的协作，给出该系统的最终类图.

使用类和 UML 中表达关系的术语，模型化一些类之间的协作，用类图对这组类以及它们之间的关系建模 .

例如下图显示了使机器人沿着一条路径移动所涉及的类

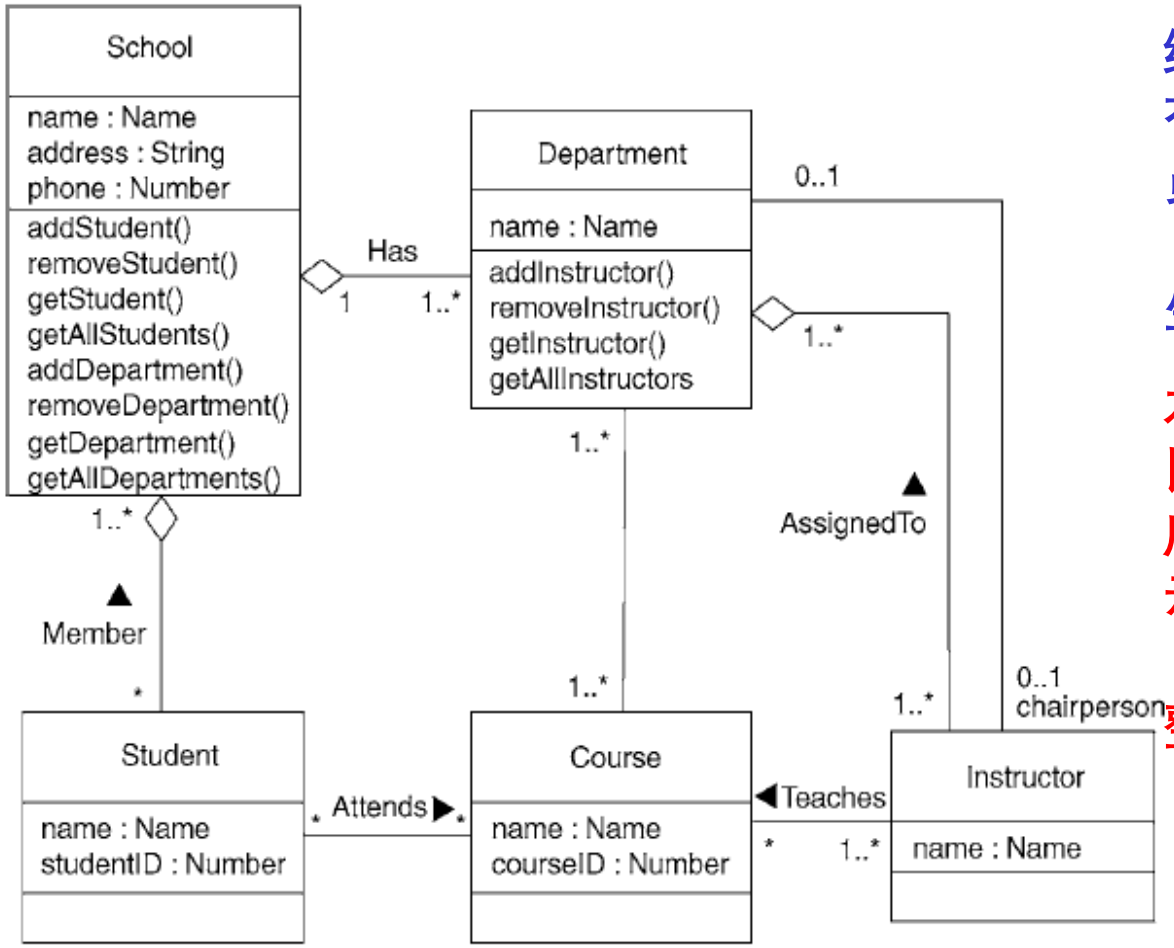


这就是一个协作

附加内容，考试不要求

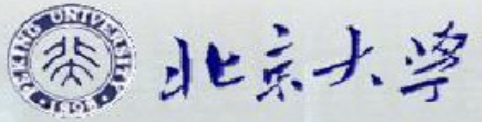
➍ 对逻辑数据库模式建模（附加）

当需要给出数据库概念设计的指导，可对要在数据库中存储的信息，采用类图对相应的数据库模式进行中建模。



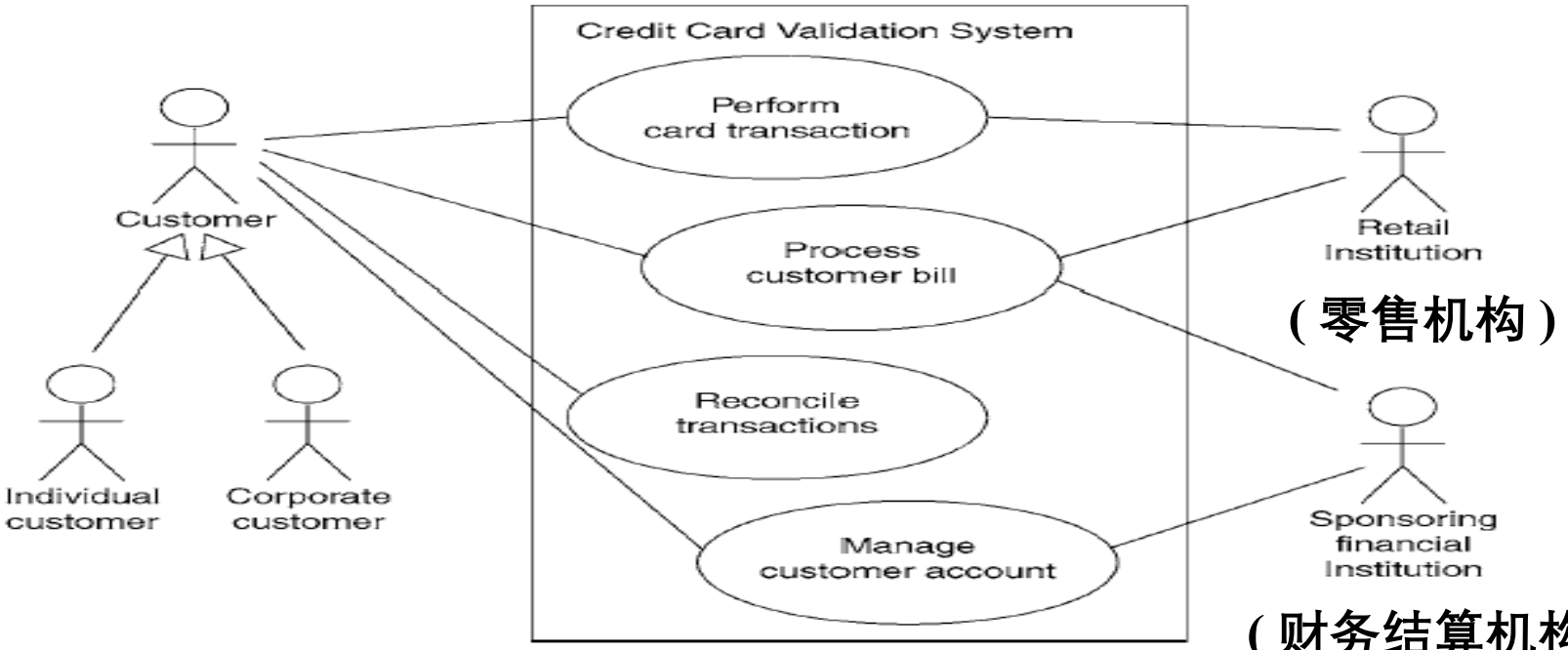
下图显示了某学校的信息系统的类图，该图表明，学校有多个系，每个系有多名教员，每位教员教授多门课程；该学校还有学生，每位学生要参加多门课。

本图显示的这些类的细节足以构造一个物理数据库 .School 和 Department 显示了几个操纵其部件的操作．这些操作对维护数据的完整性很重要．



6.3.2 系统行为（功能）的建模工具 -USE CASE 图

注：对行为的抽象，一直是人们的一个研究课题。



用况图是表现一组 Use cases 、 Actors 以及它们之间关系的图。

6.3.2.1 USE CASE 图的内容

通常包含 6 个抽象:

- 主题（Subject）；
- 用况（Use cases）；
- 参与者（Actor）
- 依赖；
- 泛化；
- 关联。

USE CASE 图还可以包含包，形成一些更大的功能块。有时为了对一个特定的执行系统进行可视化，也把用况的实例放到 USE CASE 图中。

以上抽象确定了所表达的系统的各种形态 .

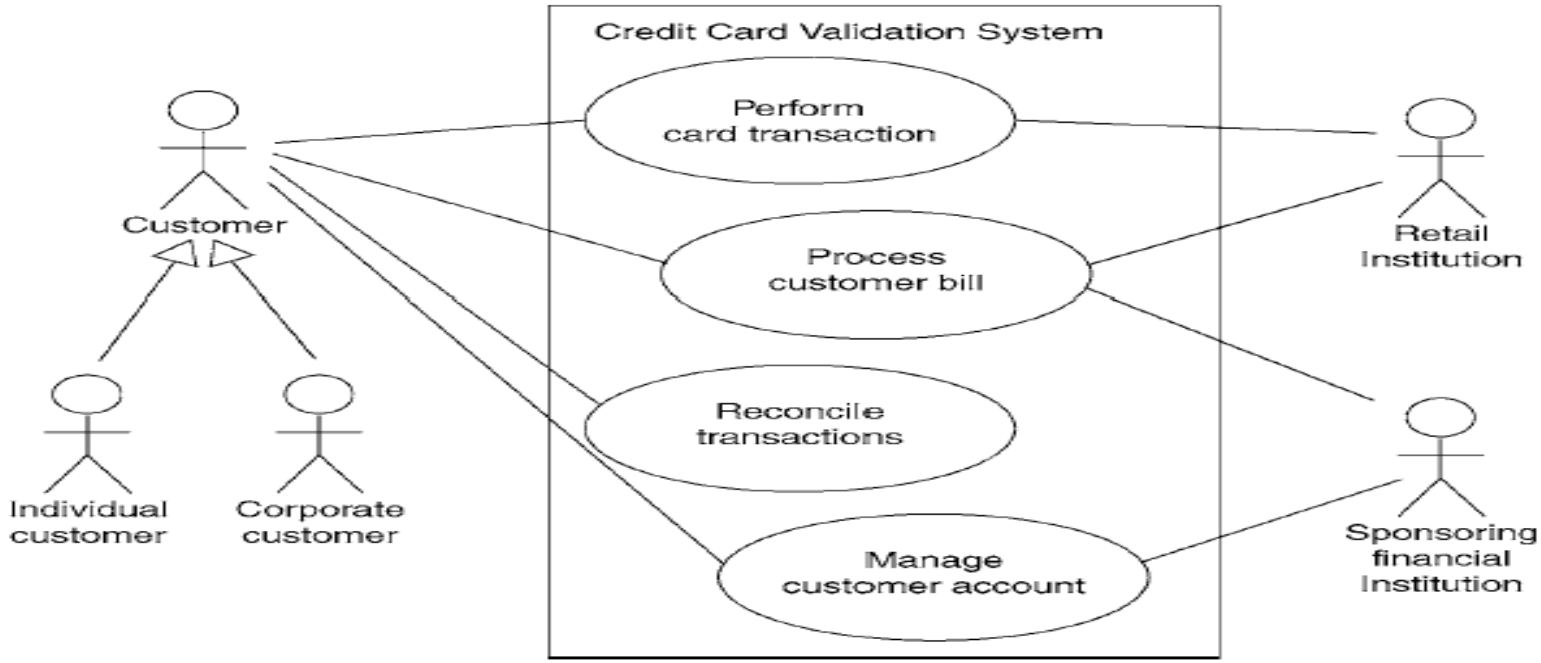
注 : 为使以 USE CASE 图表达的系统更易理解，包含注解和约束。

6.3.2.2 USE CASE 图中的术语

（1）主题

是由一组用况所描述的一个系统或子系统。其中的这些用况描述了该主题的完整行为，而参与者则表示与该主题进行交互的另一种类。

例如：以 Credit Card Validation System 所标识的矩形就是一个主题

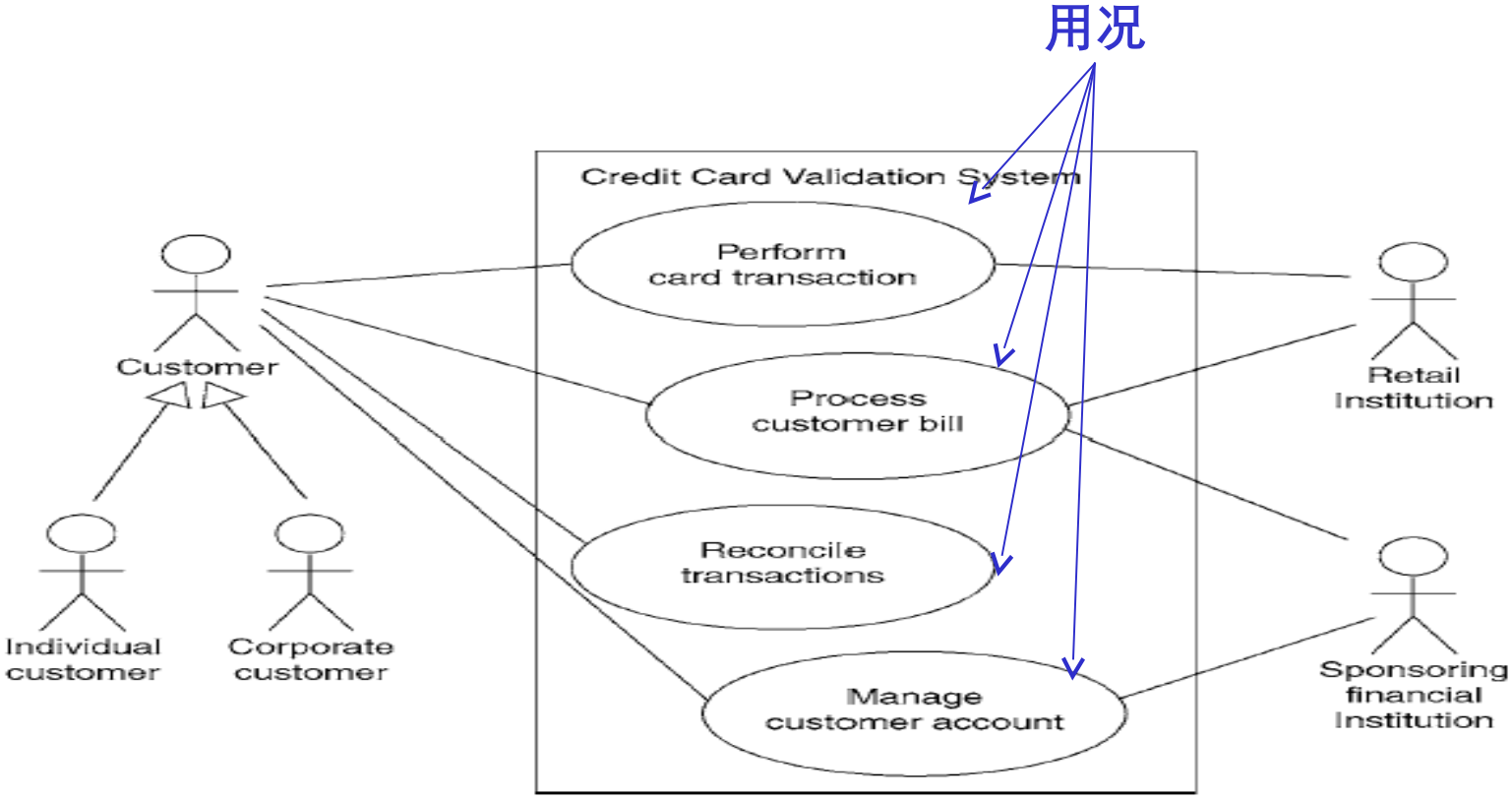


（2） USE CASE （用况）

定义：（从 2 个视角）

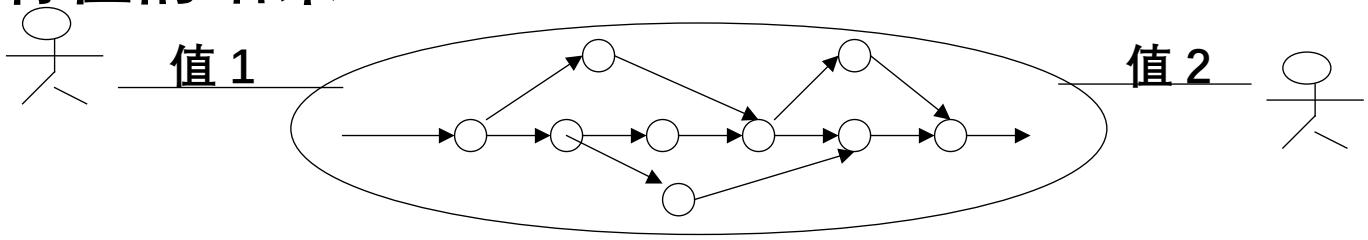
- *使用视角*：用况表达了参与者使用系统的一种方式。

例如：“做一次拼写检查”、“对一个文档建立索引”。



- *系统设计视角*：

一个 use case 规约了系统可以执行的一个动作序列，包括一些可能的变体，并对特定的操作者（ actor ）产生可见的、有值的结果。



注 :❶ 功能的体现：以 USE CASE 规约的系统功能是通过与操作者可见结果的“交互”予以体现的。即一个 USE CASE 捕获了参与交互的各方关于其行为的一个约定。

❷ 描述：对于一个 use case 的行为，可以根据具体情况，通过交互（图）、活动（图）和状态机予以描述，或通过前置条件和后置条件予以描述，或通过自然语言予以描述。

对以后开发活动的影响：

❶ Use Case 是系统分析和设计阶的输入之一，是类、对象、操作的源；并作为分析和设计一个依据；

❷ Use Case 是制定开发计划，测试计划，设计测试用例的依据之一。

❸ Use Case 可以划分系统与外部实体的界限，是系统开发的起点。

（3） actor(参与者)

定义：参与者是一组高内聚的角色，当用户与 USE CASE 交互时，该用户扮演了这一角色。

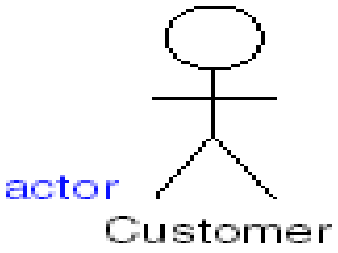
3 点说明：

❶ 一个参与者一般可以表达与系统交互的那些人(的角色)、硬件(的角色)或其它系统(的角色)。

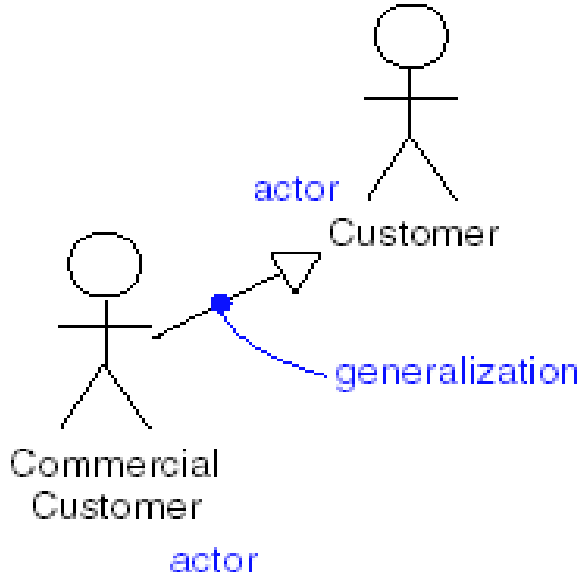
❷ 参与者实际上不是软件应用的一部分，而是在应用的环境之中，其实例代表以某种特定方式与系统进行交互。

❸ 一个客体对象可以扮演多个参与者，例如一个人既可以是参与者 LoanOffier，又是参与者 Costomer。一个参与者代表

表示：



关系：可以定义参与者之间的泛化关系，例如：



（4）关系

•关联：参与关系，即操作者参与一个 USE CASE 。例如，操作者的实例与 USE CASE 实例相互通讯。

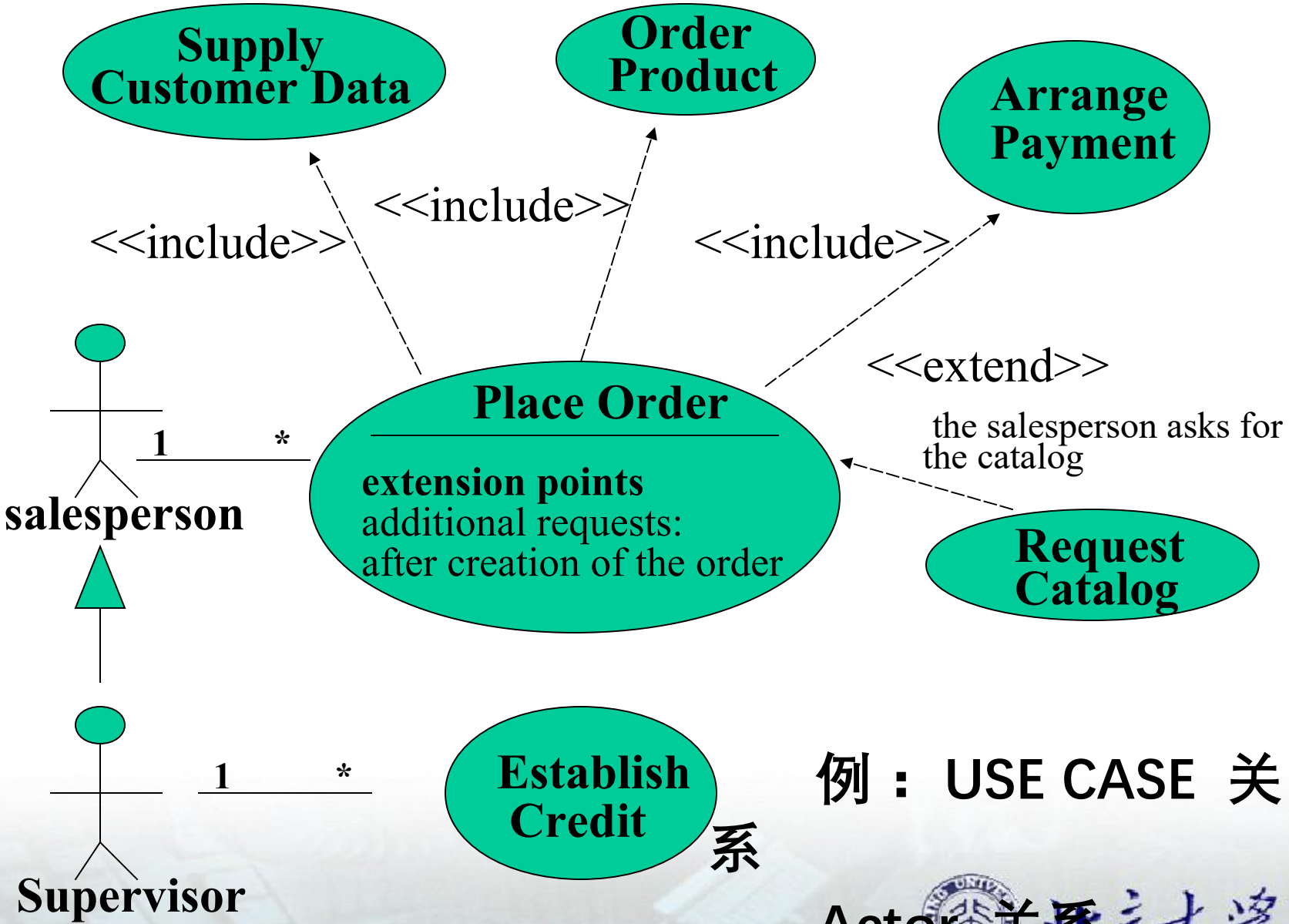
关联是操作者和 USE CASE 之间的唯一关系。

•扩展：USE CASE A 到 USE CASE B 的一个扩展关系，指出了 USE CASE B 的一个实例可以由 A 说明的行为予以扩展（根据该扩展所说明的特定条件），并依据该扩展点定义的位置， A 说明的行为被插入到 B 中。

•包含：USE CASE A 到 USE CASE B 的一个包含，指出 A 的一个实例将包含 B 说明的行为，即这一行为将包含在 A 定义的那部分中。

•泛化：USE CASE A 到 USE CASE B 的泛化，指出 A 是 B 的特殊情况。

注：扩展和包含是依赖的变体。



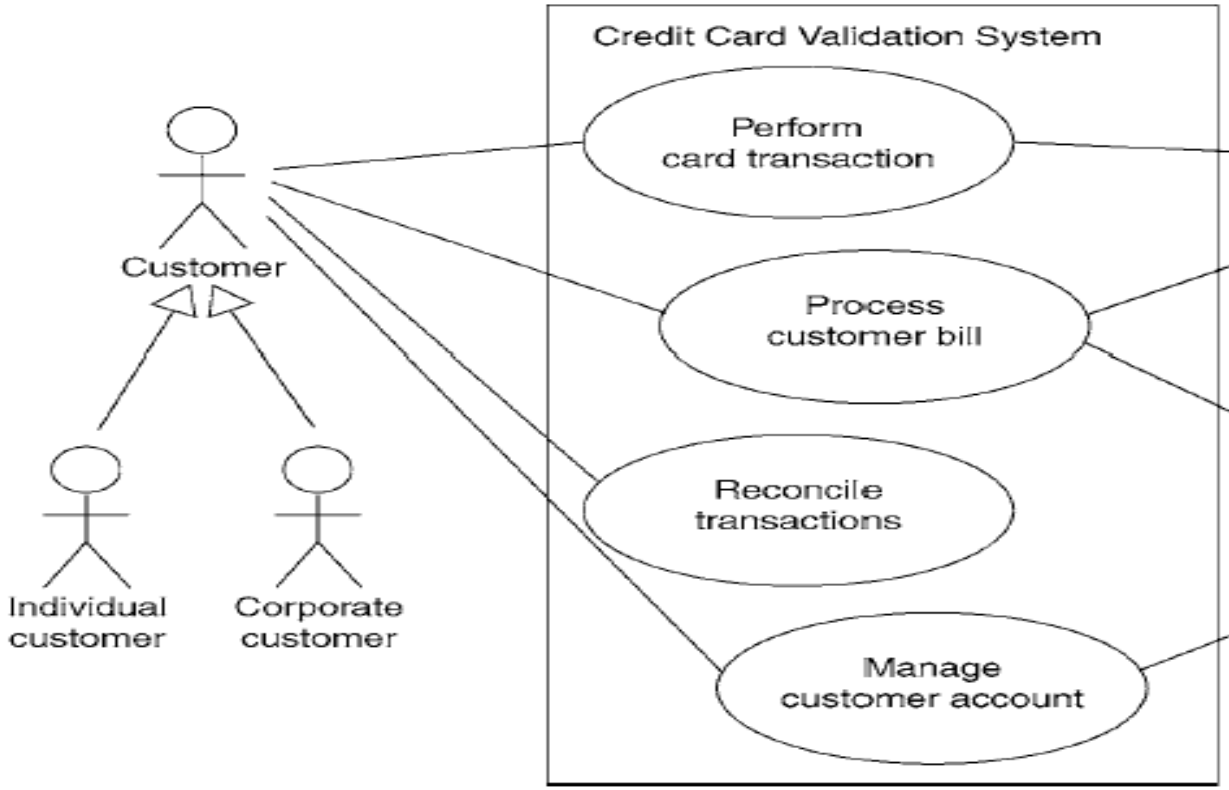
例：USE CASE 关系

Actor关系

6.3.2.3 用况图的使用

（1）对系统语境建模

任意一个系统，均有其内部的事物和外部的事物。例



零售机构（顾客通过该机构刷卡，购买商品或服务。

财务结算机构（负责信用卡帐户的结算服务）

在如上的信用卡确认系统中，

- 内部事物有：帐户、事务处理和欺诈行为检测代理．

  其责任是完成外部事物所期望由系统提供的行为。

- 外部事务有：信用卡顾客和零售机构，财务结算机构。

  它们与内部事物进行交互，构成了该系统的语境．

  该语境定义了系统存在的环境。

由此可见，若采用 UML 用况图对系统语境进行建模，就应关注存在于系统周围的参与者，确定什么作为系统的参与者，什么不作为系统的参与者，使该图只包含那些在其生命周期内所必须的参与者。

对系统语境建模应遵循的基本策略

- 决定哪些行为是系统的一部分，哪些行为是由外部实体执行的，以此标识系统边界，同时定义主题。
- 在标识系统的参与者时，应考虑以下问题：

  谁需要得到系统的帮助，以完成其任务；

  谁执行系统的功能；

  系统与哪些硬件设备或其他系统交互；

  谁执行一些辅助功能进行系统的管理和维护。
- 将一些相似的参与者组织为一般 / 特殊结构。
- 在需要加深理解的地方，为每个参与者提供一个衍型。

最后，将这些参与者放入用况图中，并建立它们与系统用况之间的关联 - 通信路径 .

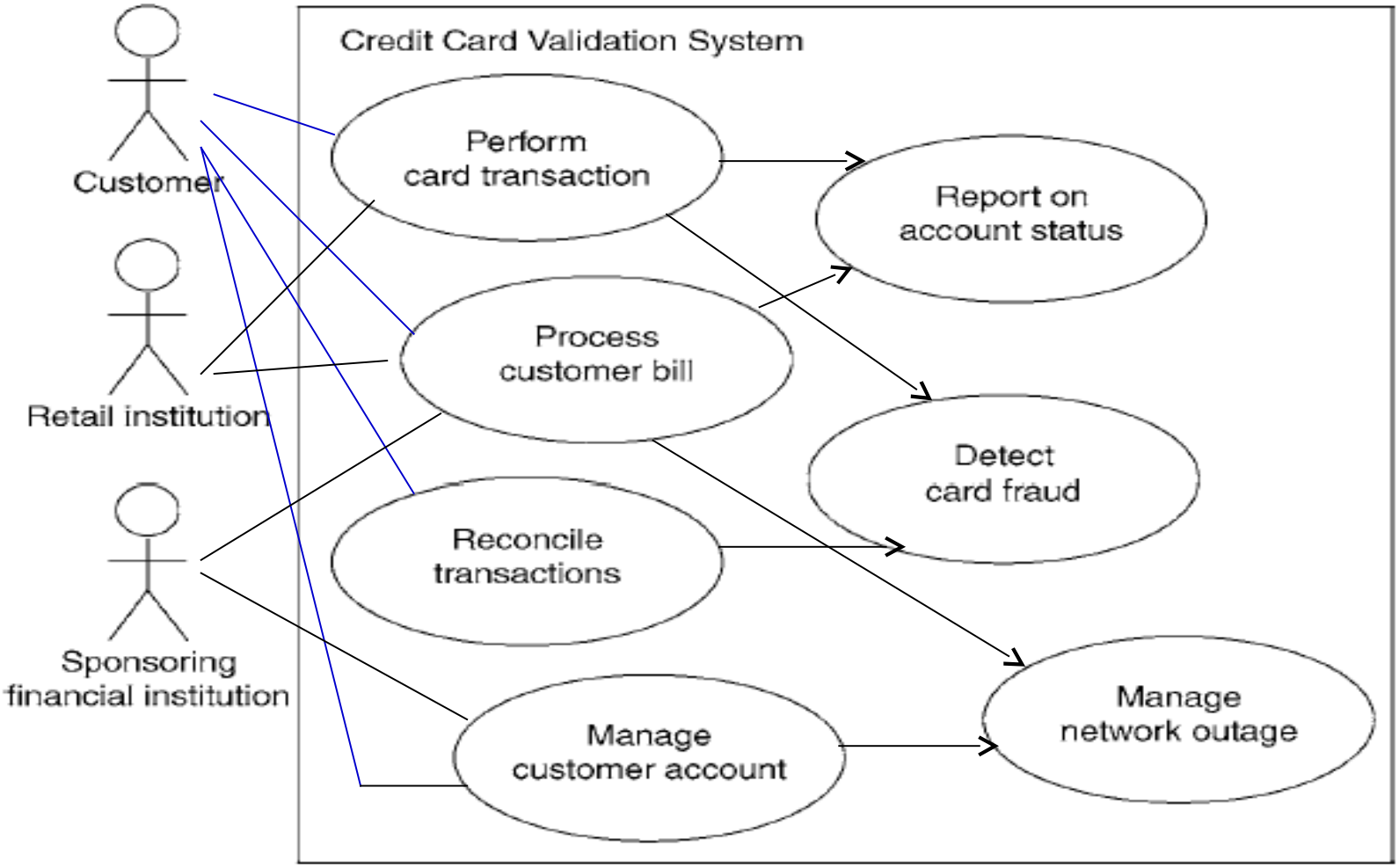
（2）对需求建模

对系统的需求建模应遵循的策略：

- 首先，通过标识参与者来建立系统的语境；
- 其次，对于每个参与者考虑他所期望或需要系统提供的行为。并把它们作为用况；
- 第三，通过分解用况所表达的公共行为，形成必要的泛化结构；分解异常行为，放入新的用况中以延伸较为主要的用况；
- 第四，模型化用况图中各种关系；
- 最后，通过注解和约束给出这些用况的非功能需求。

例如：

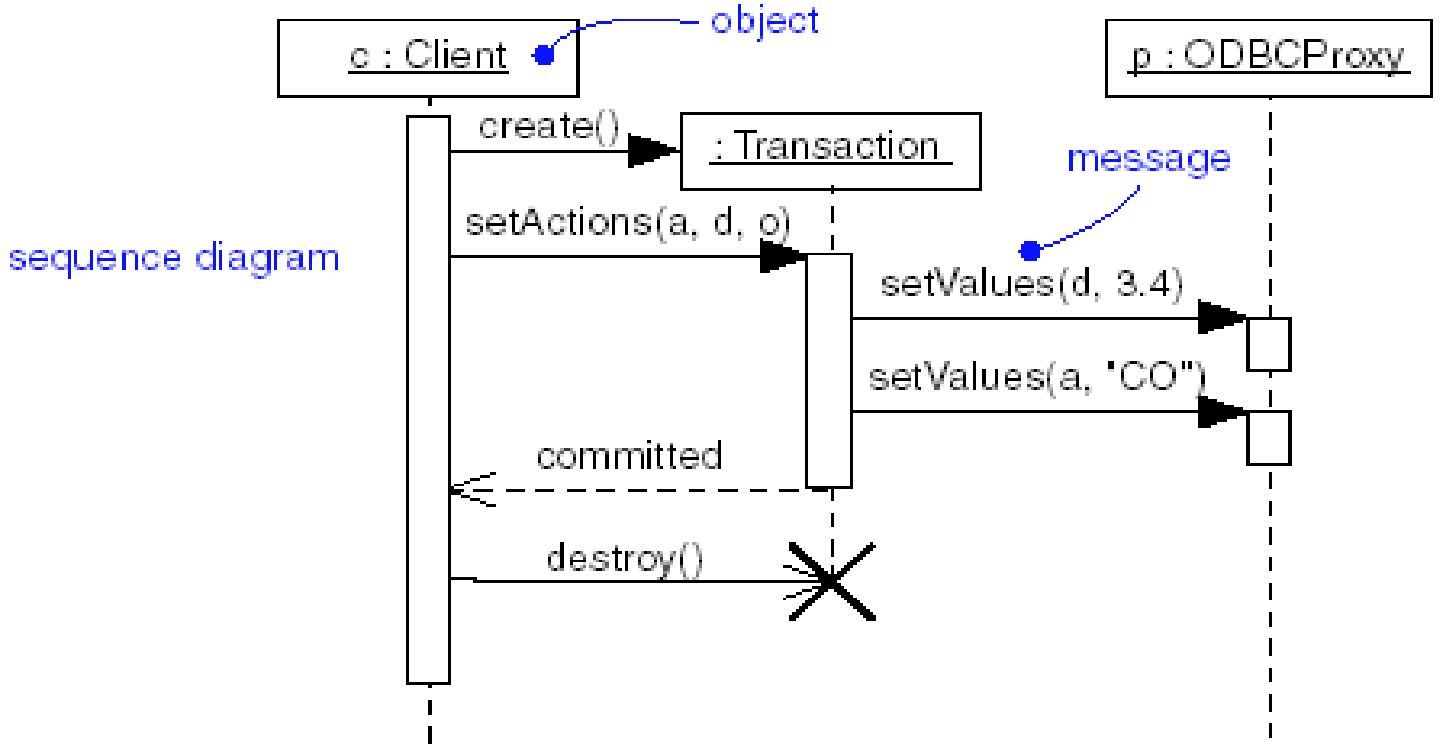
下图显示了对系统的需求建模



6.3.3 系统行为（交互）的建模工具 - 顺序图

定义：顺序图是一种交互图，即由一组对象以及这些对象之

间的关系（通信）组成，其中还包含这些对象之间



顺序图所包含的内容：

❶ 交互各方：角色或对象

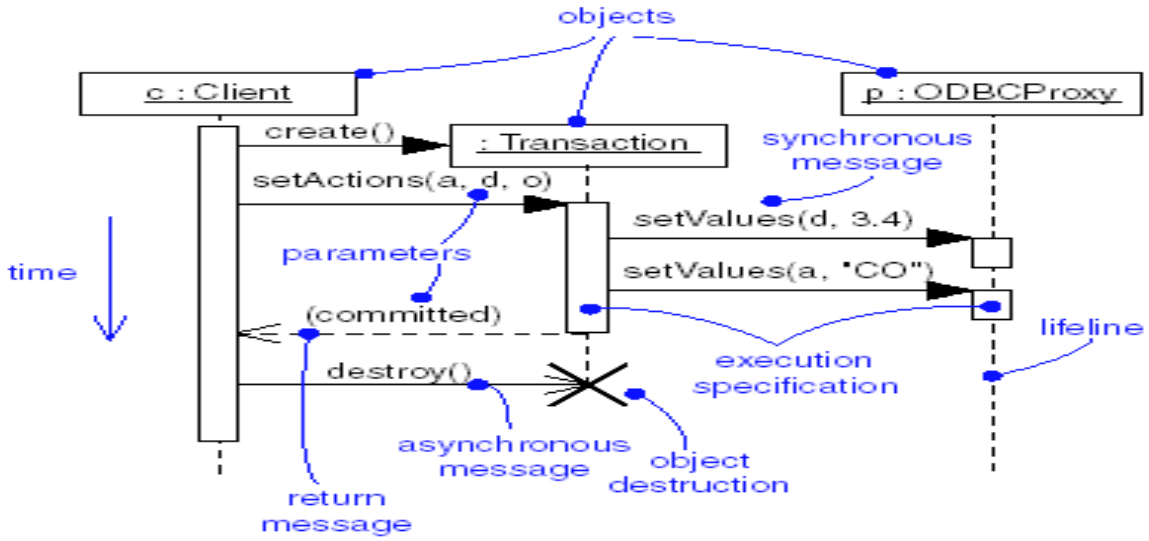
❷ 交互方式：同步或异步

❸ 交互内容：消息

像其它图形一样，可以包含注解和约束。

这些成分确定了交互的各种形态 .

从应用的角度来看，交互图是一个交互中各元素（各方、
方式和内容）的投影。其中把这些元素的语义应用到交互图中。



5 点说明

❶ 对象生命线

用于表示一个对象在一个特定的时间段中的存在。

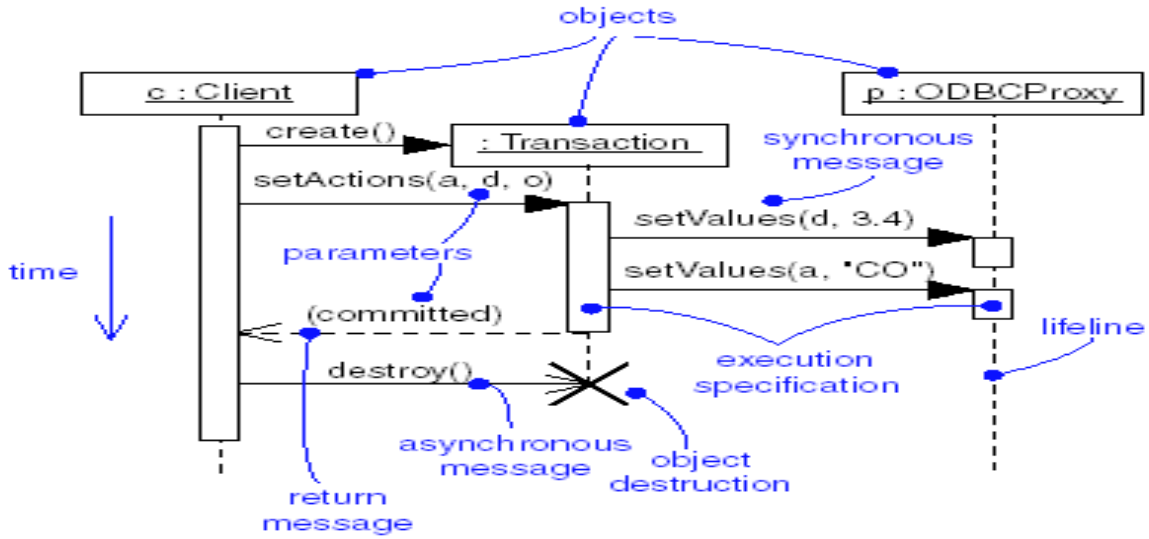
对象生命线被表示为垂直的虚线。

❷ 消息

顺序图包含了一些由时间定序的消息。消息被表示为一条箭头线，从一条生命线到另一条生命线。其中：

-- 如果消息是异步的，则用枝形箭头线表示；

-- 如果消息是同步的（调用），则用实心三角箭头线表示；同步消息的回复用枝形箭头虚线表示。



❸ 聚焦控制（the focus of control）

表达一个对象执行一个动作的时间段。可以表达嵌套的控制关注。

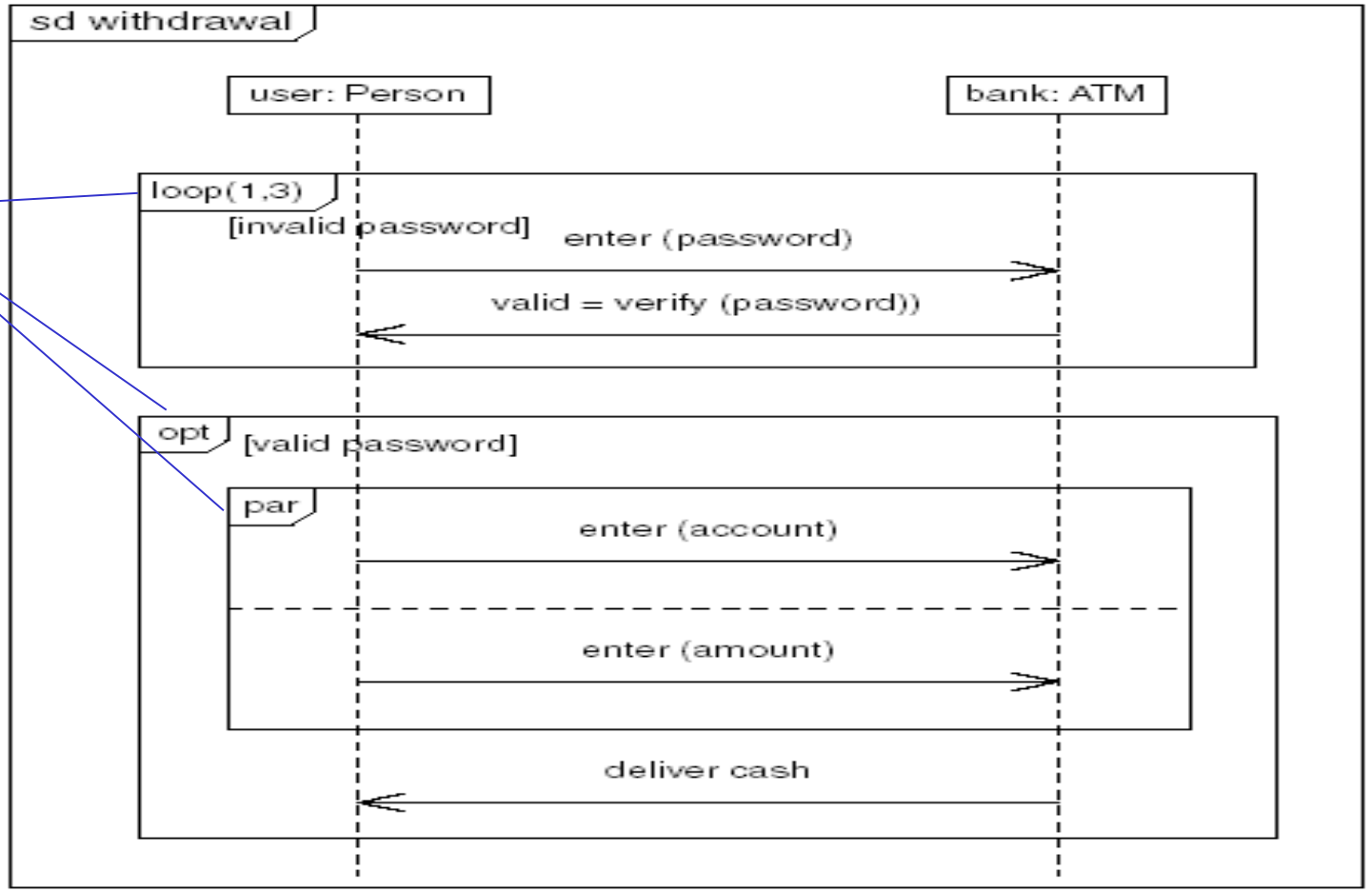
聚焦控制被表示为细高的矩形。

❹ 时序

一条生命线上的时序是非常重要的，使消息集合形成了一个偏序关系，建立了一个因果链。

❺ 顺序图中的控制结构：为了控制交互行为描述的复杂性，更清晰地表达顺序图中的复杂控制

结构化控制操作符



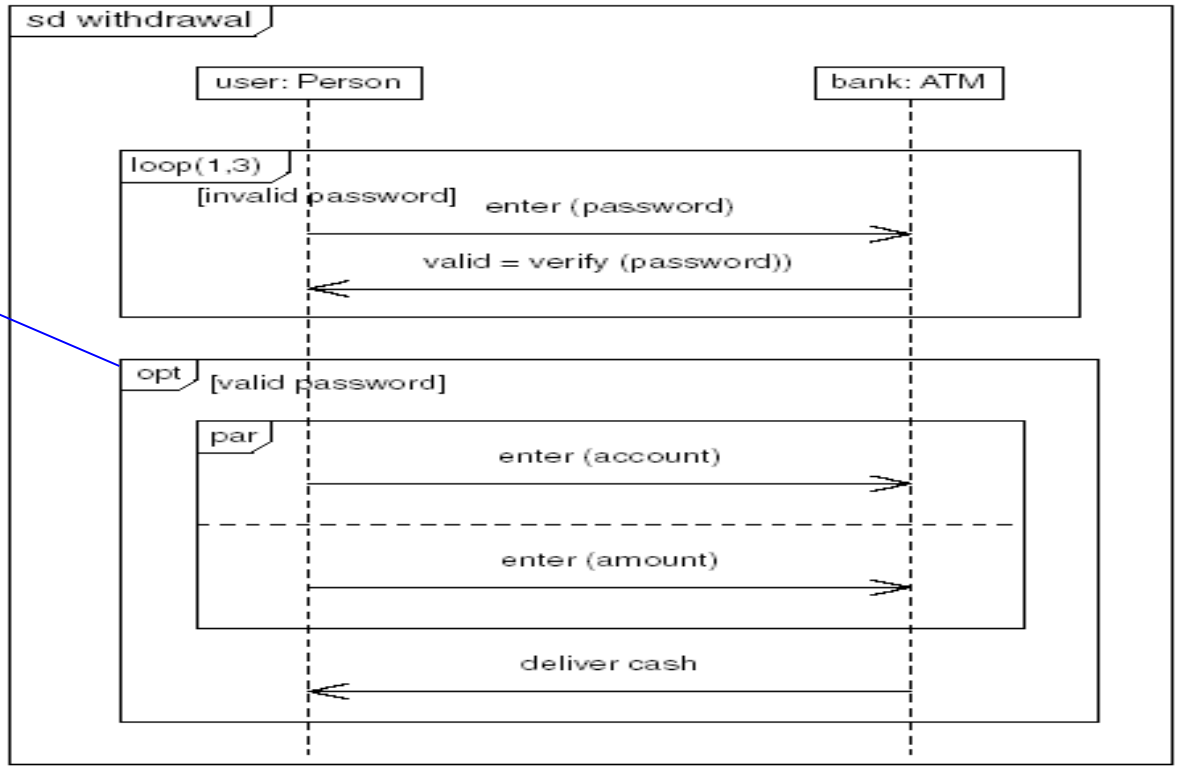
常见的控制类型有：

❶ 选择执行 (Optional execution)

一种控制结构类型，其标签为 opt 。仅当进入该控制操作子（control operator），监护条件为真时，该控制操作子的体才予执行。

注：监护条件是一个布尔表达式，可以出现在该体中任意一个生命线顶端的方括号内，并且可以引用那个对象的属性。

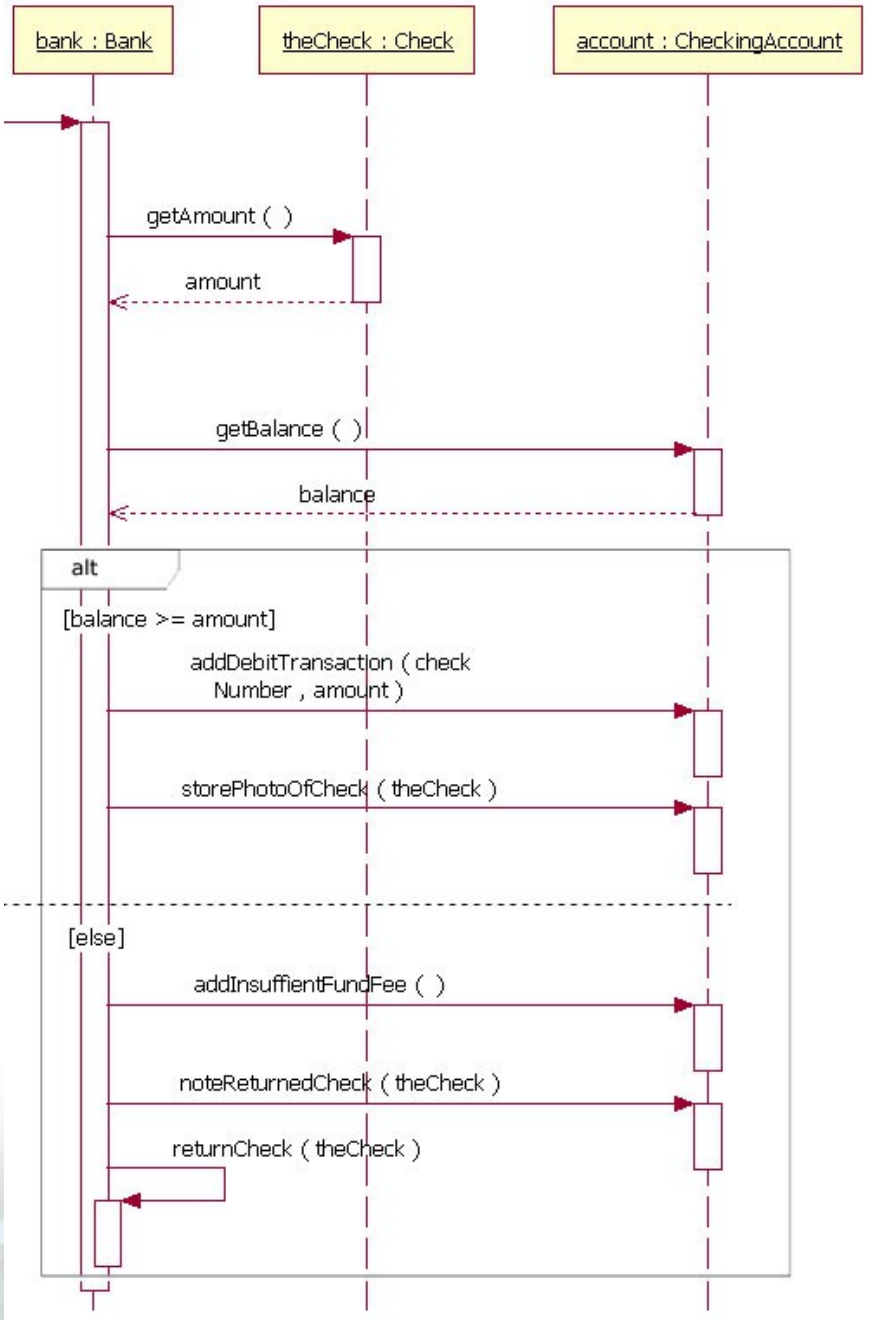
选择控制操作符



❷ 条件执行（Conditional execution）

一种控制结构类型，其标签为 alt 。该控制操作子的体通过水平线将其分为一些部分。每一部分表示一个条件分支，并有一个监护条件。其中：

- 若一个部分的监护条件为真，那么该部分就被执行。但是，最多一个部分可以被执行；如果多个监护条件为真时，选择哪一部分执行，这是一个非确定性的问题，其执行可以不同。
- 如果没有一个监护条件为真，那么控制将绕过该控制操作子而继续。
- 一个部分可以有一个特定的监护条件 [else] ；对于这一部分而言，如果没有其它监护条件为真，那么该部分才被执行。

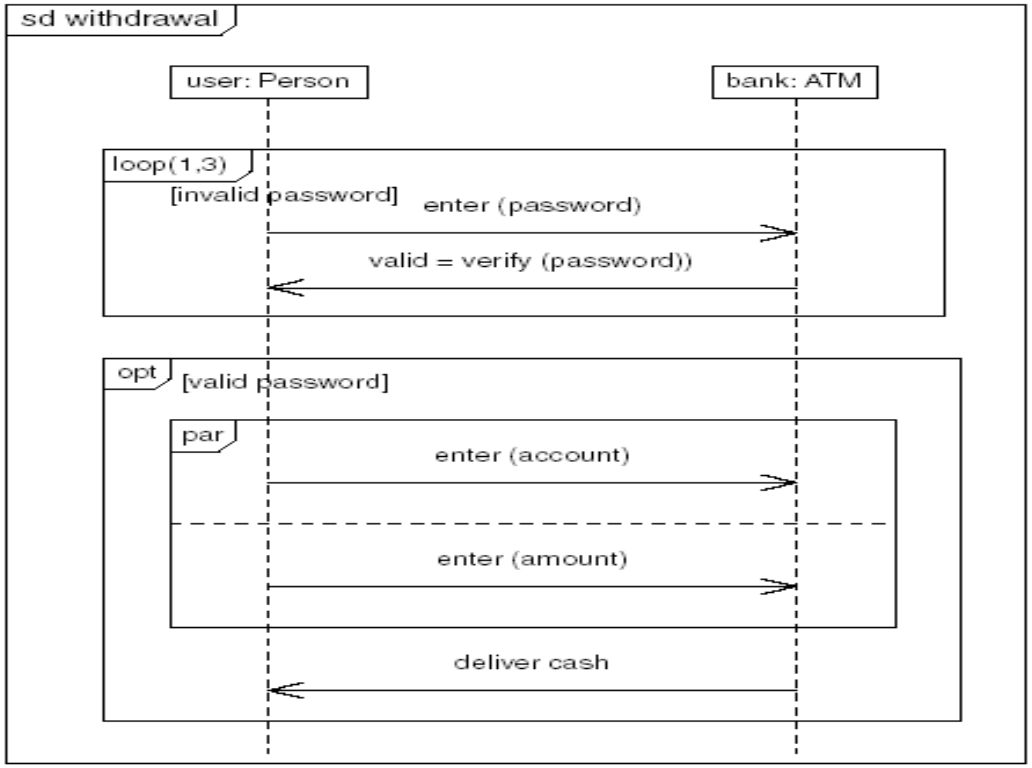


❸ 并发执行（Parallel execution）

一种控制结构类型，其标签为 par 。该控制操作子的体通过水平线将其分为多个部分。每一部分表示一个并行计算。在大多数情况下，每一部分涉及不同的生命线。

♦ 当进入该控制操作子时，所有部分并发执行。

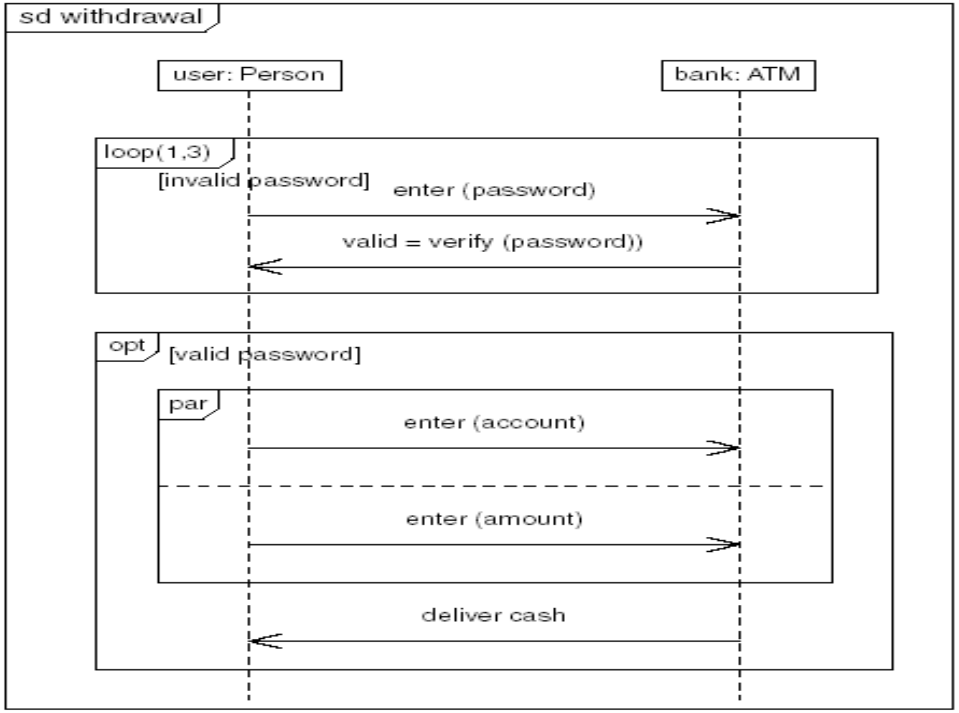
♦ 在每一部分中的消息的发送 / 接受是有次序的，但在整个并发部分中的消息次序则完全是任意的。



注：实际上存在很多情况，可分解为一些独立的、并发的活动，因此，这是一个非常有用的操作子。

❹ 迭代执行（ iterative execution ）

一种控制结构类型，其标签为 loop 。监护条件出现在该体中一个生命线的顶端，只要在每一次迭代之前该监护条件为真，该循环体就反复执行。当该体上面的监护条件为假时，控制绕过该控制操作子。

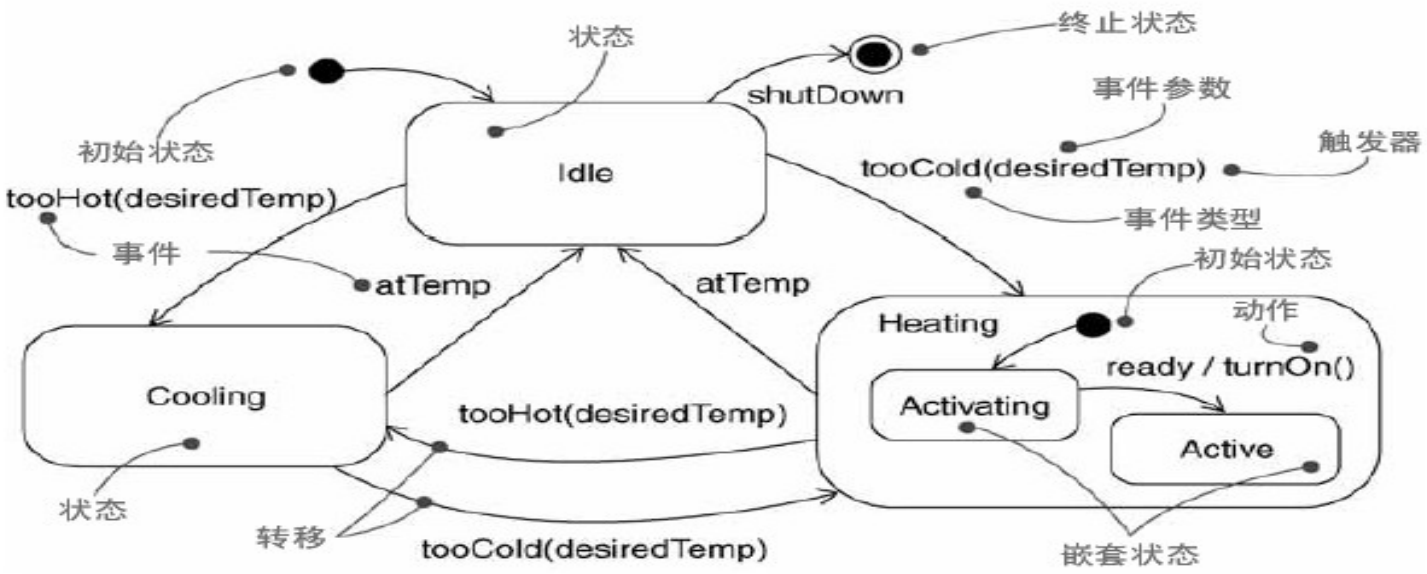


注：还存在其它控制操作子，但以上 4 种是最常使用的。

6.3.4 系统行为（生存周期）的建模工具 - 状态图

6.3.4.1 定义

状态图是显示一个状态机的图，其中强调了从一个状态到另一状态的控制流。



一个状态机是一种行为，规约了一个对象在其生存期间因响应事件并作出响应而经历的状态。

6.3.4.2 状态图的内容

通常包含： ❶ 简单状态和组合状态；

❷ 事件

❸ 转换

像其它图形一样，可以包含注解和约束。

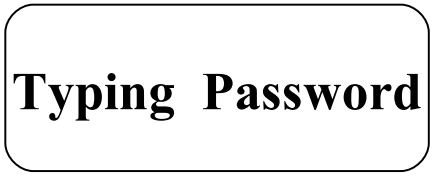
从使用的角度来看，一个状态图可以包含一个状态机中任意的、所有的特征（features），即状态图基本上是一个状态机中那些元素(分支、结合、动作状态、活动状态、对象、初始状态、最终状态、历史状态等)的一个投影。

状态图中所包含的内容，确定了一个特定的抽象层，该抽象层决定了以状态图所表达的模型之形态。

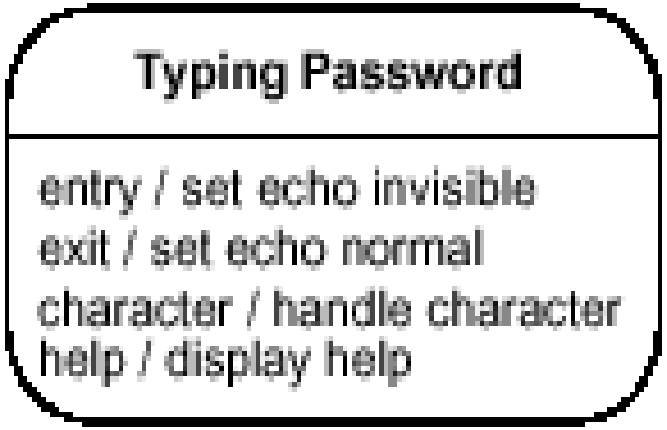
（1）状态

① 定义：一个状态是类目的一个实例（以后简称对象）在其生存期间的一种条件 (condition) 或情况（situation），该期间该对象满足这一条件，执行某一活动或等待某一消息。

② 表示：



(A)



(B)

一个状态表达了一个对象所处的特定阶段，所具有的对外呈现（外征）以及所能提供的服务。

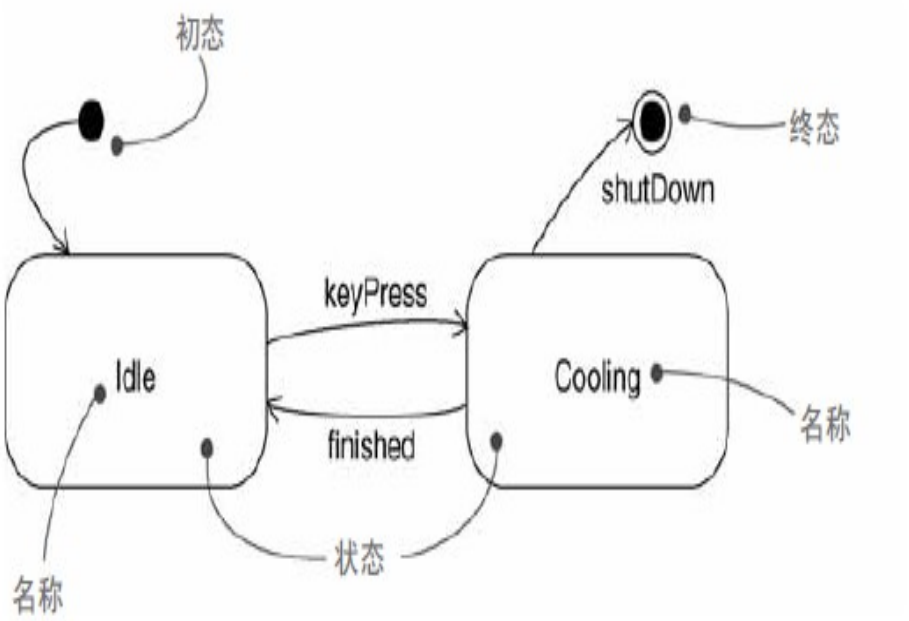
③ 状态分类：

UML 把状态分为初态、终态和正常状态：

初态：表达状态机默认的开始位置，用实心圆来表示；

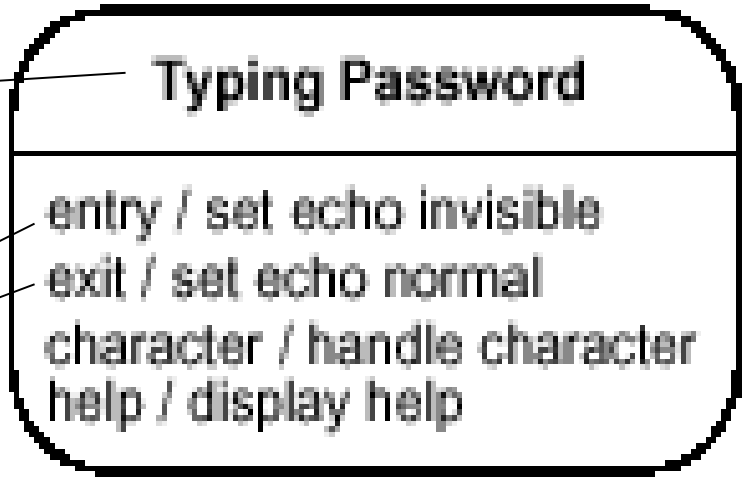
终态：表达状态机的执行已经完成，用内含一个实心圆的圆来表示；

正常状态：既不是初态又不是终态的状态，称为正常状态。



初态和终态都是伪状态，即只有名字。从初态转移到正常状态可以给出一些特征，例如监护条件和动作。

④ 状态的规约：



❶ 名字

是一个标识状态的文本串，作为状态名。

也可以有匿名状态－没有给出状态名。

❷ 进入 / 退出之效应 (effect)

是进入或退出该状态时所执行的动作。

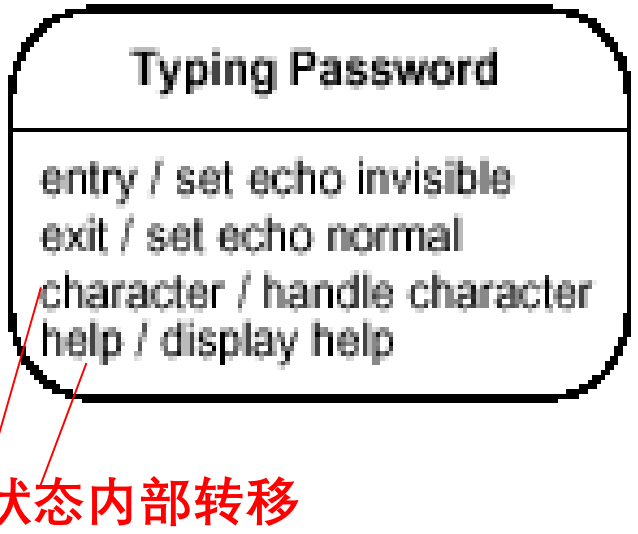
为了表达进入 / 退出之效应， UML 给出 2 个专用的动作标号：

- **entry** 该标号标识在进入该状态时所要执行的、由相应动作表达式所规定的动作，简称进入动作。
- **exit** 该标号标识在退出该状态时所要执行的、由相应动作表达式所规定的动作，简称退出动作。

一般情况下，进入／退出之效应不能有参数或监护条件，但位于类状态机顶层的进入效应可以具有参数，以表示在创建一个对象状态机时所要接受的参数。

❸ 状态内部转移

是指没有导致该状态改变的内部转移。一般情况下，在此给出对象在这个状态中所要执行的内部动作或活动列表。其中表达动作的一般格式为：动作标号’ /’ 动作表达式



动作标号标识了在该环境下所要调用的动作，而该动作是通过‘ /’ 之后的动作表达式所规约，其中可以使用对象范围内的任何属性和链。若该表达式为空，则可省略斜线分隔符。

为了表达状态内转换中的动作或活动， UML 给出了一个专用的动作标号： do ，该标号标识正在进行由其相应动作表达式所规定的活动，并且只要对象在一个状态中没有完成由该动作表达式所指定的活动，就一直执行之；当动作表达式指定的活动完成时，可能会产生一个完成事件。

注：动作标号“ entry” 、“ exit” 和“ do” 均不能作为事件名。

在以上的叙述中，使用了两个词：

动作（action）和活动 (activity)

一个活动是指状态机中一种可中断的计算，中断处理后仍可继续；

而一个动作是指不可中断的原子计算，它可导致状态的改变或导致一个值的返回。

可见，一个活动往往是有多个动作组成的。

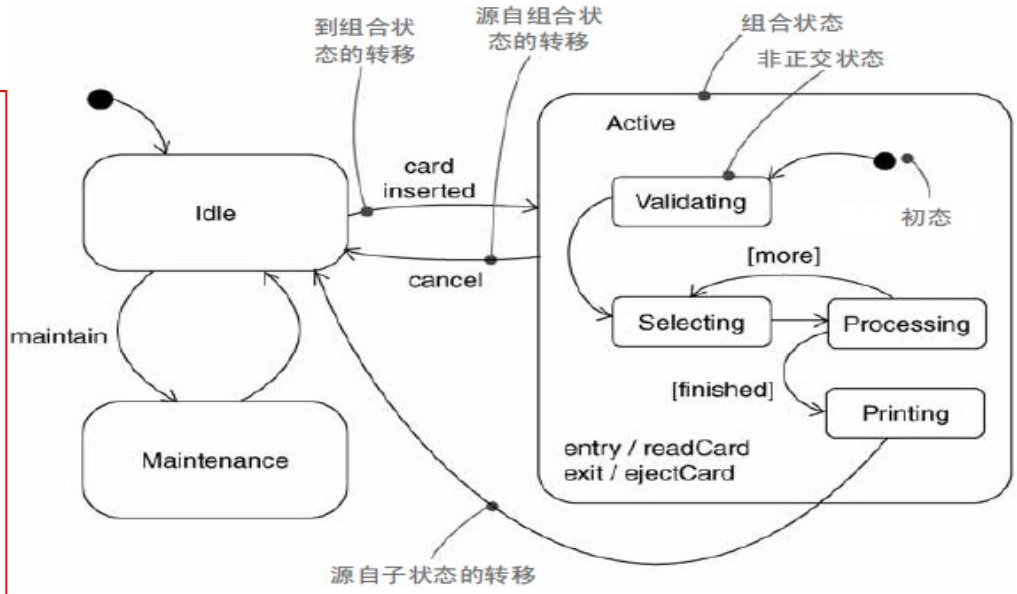
❹ 子状态

如果在一个状态机中引入另一个状态机，那么被引入的状态机称为子状态机。子状态是被嵌套在另一状态中的状态。相对地，把没有子状态的状态称为简单状态；而把含子状态的状态称为组合状态。

子状态机分类： 顺序子状态机（非正交）和并发子状态机（正交）

♦ 非正交子状态机

注意：右边是一个组合状态，包含一些子状态，表达了一个非正交状态机。一个非正交状态机最多有一个子初态和一个子终态。



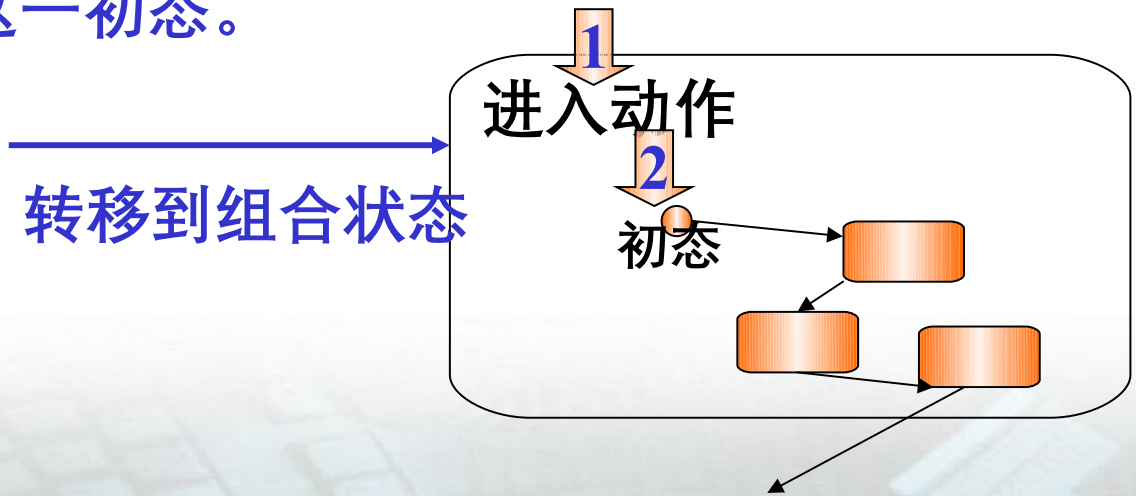
关于转入问题：

从组合状态之外的一个源状态（如上图中的“ Idle”），

- 可以转移到该组合状态，作为其目标状态；

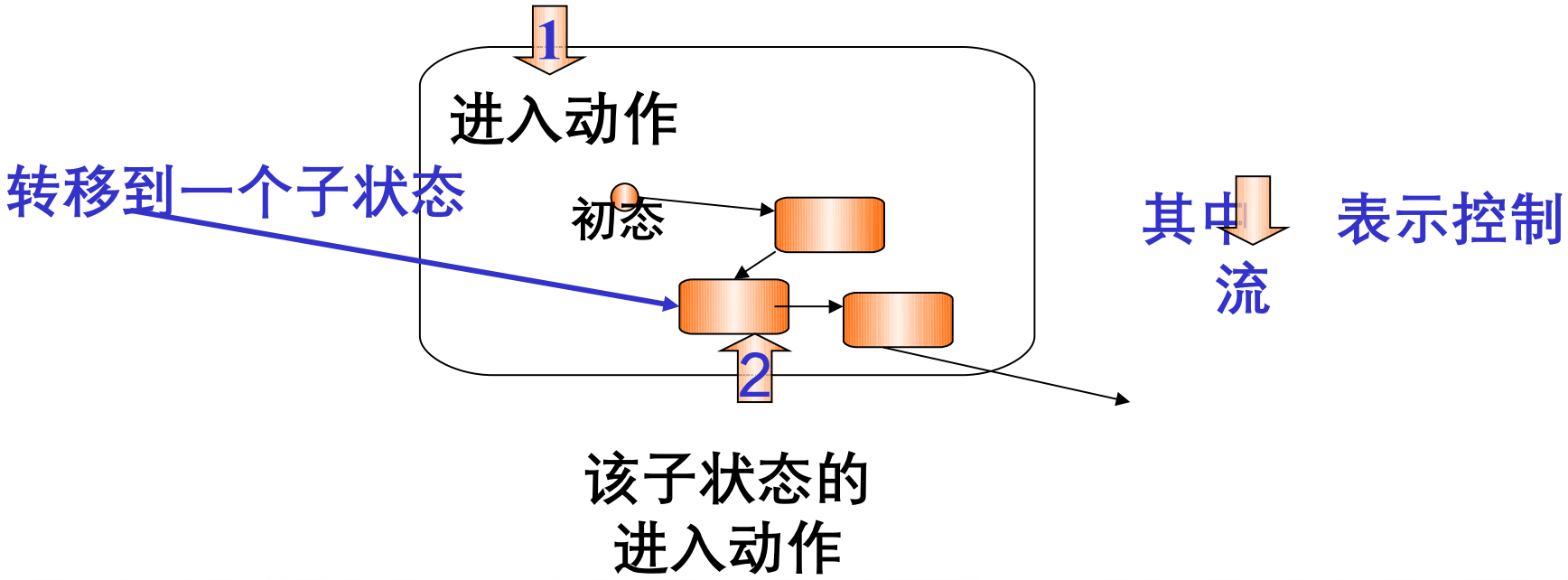
- 可以转移到组合状态中一个子状态，作为其目标状态。

在第一种情况里，这个被嵌套的子状态机一定有一个初态，以便在进入该组合状态并执行其进入动作后，将控制传送给这一初态。



其中 表示控制流

在第二种情况里，在执行完该组合状态的进入动作（如有的话）和该子状态的进入动作后，将控制传送给这一子状态。



关于离开问题：

❶ 离开一个组合状态的转移，其源

-- 可以是该组合状态，

-- 可以是该组合状态中的一个子状态。

无论哪种情况，控制都是：

* 首先离开被嵌套的状态，即执行被嵌套状态的退出动作（如有的话）

* 然后离开该组合状态，即执行该组合状态的退出动作（如有的话）。

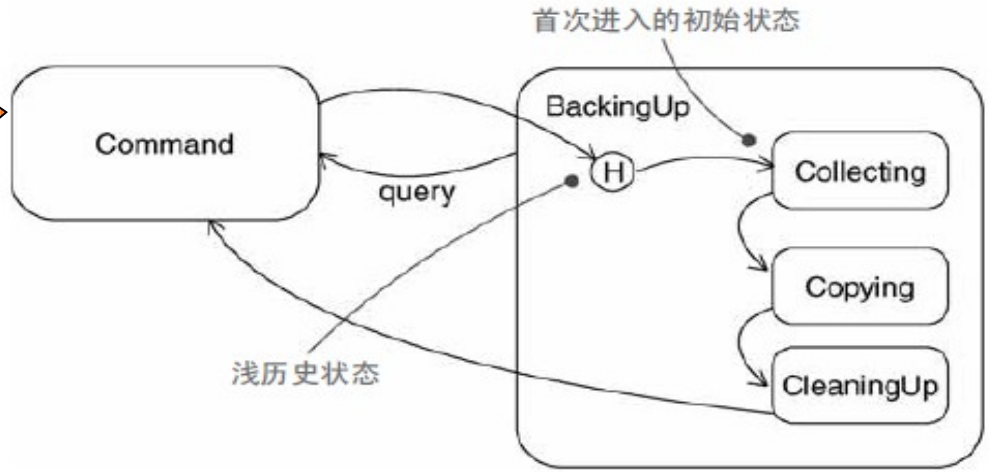
由此可见，如果一个转移，其源是一个组合状态，那么该转移的本质是终止被嵌套状态机的活动。

❷ 当控制到达该组合状态的子终态时，就触发一个活动完成的转移。

关于离开组合状态之前执行活动相关子状态标识问题

如果需要知道在转移出该组合状态之前所执行的那个活动所在的子状态， UML 引入了一个概念 - 浅历史状态，用于指明这样的子状态，并用H来表示之。如下图所示。

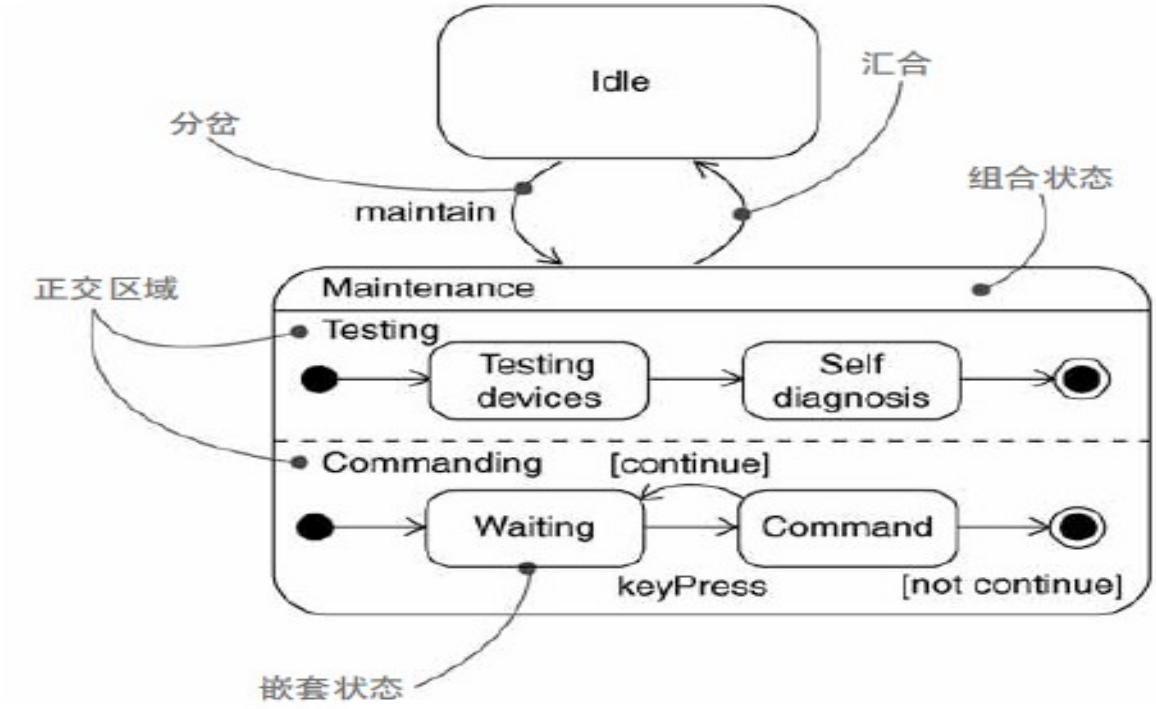
右边的图表示：在对一个通过网络进行无人值守的计算机备份的代理的行为建模时，如果它曾被中断 (如一个操作员的查询中断), 希望它用 H(浅历史状态) 记住是在该过程的什么地方被中断的 .



相对于 H 所表示的浅历史，可用 H* 来表示深历史，即在任何深度上表示最深的、被嵌套的状态。

- 正交子状态机

正交子状态机是组合状态中一些并发执行的子状态。例如：



其中，使用虚线段形成了两个正交区域（根据需要，可形成多个正交区域），分别以“ Testing” 和“ Commanding” 标记之，并且每个区域均有自己的初态和终态。

关于转入问题：分岔

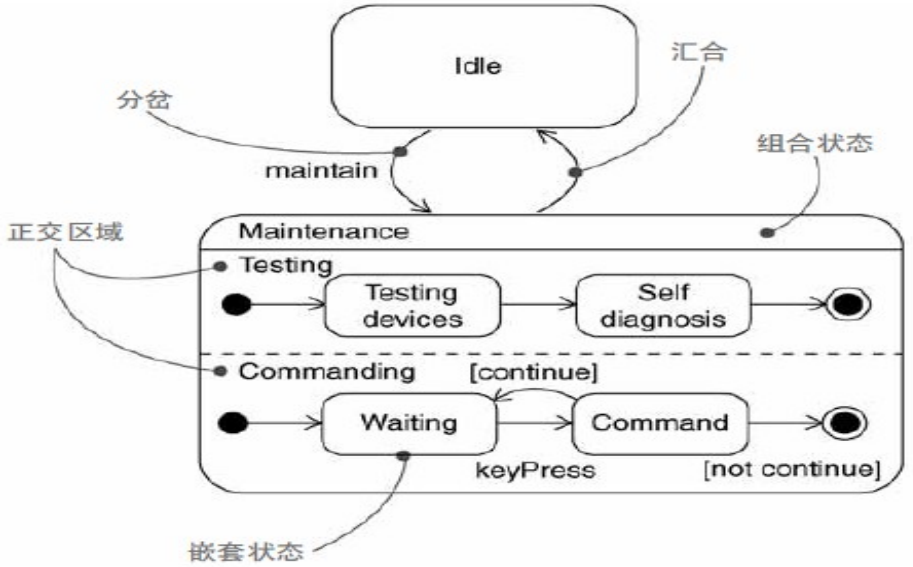
当一个转移到达具有多个正交区域的组合状态时，控制就被分成多个并发流 - 分岔。正交区域的执行是并行的，相当于存在两个被嵌套的状态机。

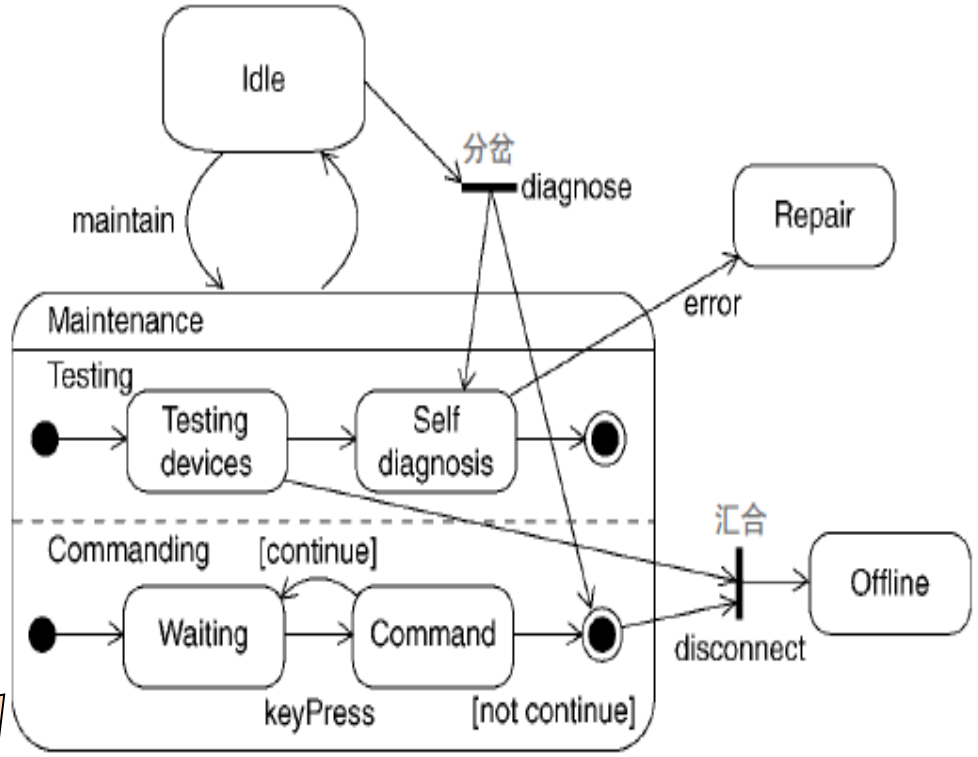
关于离开问题：汇合

-- 如果一个正交区域先于另一个到达它的终态时，那么该区域的控制将在该终态等待，直到另一个区域的控制达到自己的终态时，两个区域的控制才汇合成一个控制流 - 汇合。

-- 当一个转移离开这样的组合状态时，控制就汇成一个控制流。

-- 如果所有正交区域均达到它们的终态，或存在一个指示离开组合状态的转移，那么就汇成一个流。





上图显示了显式的分岔和汇合，以及隐式的分岔和汇合．
（1）进入组合状态 Maintenance 的转移 maintain 是一个到所有正交区域的默认初始状态的隐含分岔；（2）从 Idle 状态到两个嵌套状态，即 self diagnose 和区域 Commanding 的终止状态；（3）如果在 Self diagnose 状态活动时发生了错误，就会激活到 Repair 的隐式的汇合转移：无论是 Self diagnose 状态，还是 Commanding 区域内的任何活动状态都会退出；（4）一个到状态 Offline 的显式的汇合转移：只有当 Testing devices 状态和 Commanding 区域的终止状态是活动的，而且 disconnect 事件发生，才会激活这个转移；如果两个状态都不活动，则该事件无效．

❺ 被延迟事件

被延迟事件是那些在一个状态中不予处理的事件列表。往往需要一个队列机制，对这样的事件予以推迟并予排队，以便在该对象的另一状态中予以处理。

小结：状态的规约，包括：

- ♦ 命名
- ♦ 进入 / 退出之效应
- ♦ 状态内部转移
- ♦ 子状态与组合状态
- ♦ 被延迟事件

下面介绍状态图中第二个基本元素 - 事件。

（2）事件

一个事件是对一个有意义的发生的规约，该发生有其自己的时空。在状态机的语境下，一个事件是一个激励（stimulus），可引发状态的转换。

事件的种类:

内部事件：是在系统内对象之间传送的事件。例如，溢出

异常。

外部事件：是在系统和它的参与者之间传送的事件。例如

可以把衍型看成元类型 (一种定义其他类型的类型), 因为每一个衍型将创建一个相当于 UML 元模型中新类的等价物 .

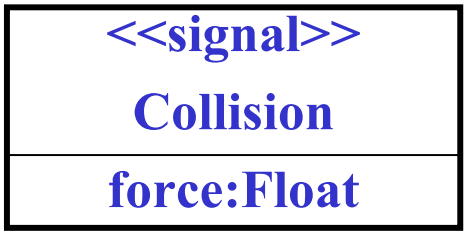
在 UML 中，可模型化 4 种事件：

❶ 信号（signal）

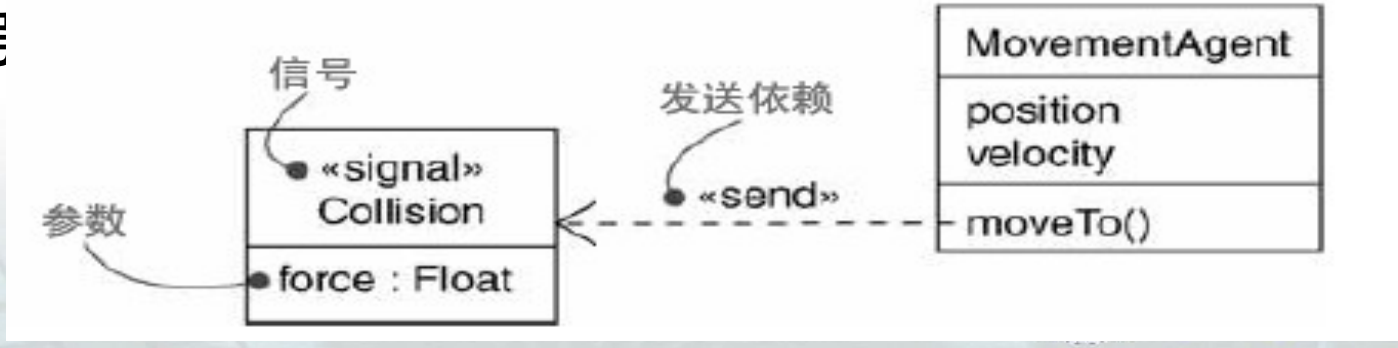
信号是消息的一个类目，是消息类型。

信号有属性和操作，信号之间可以有泛化。

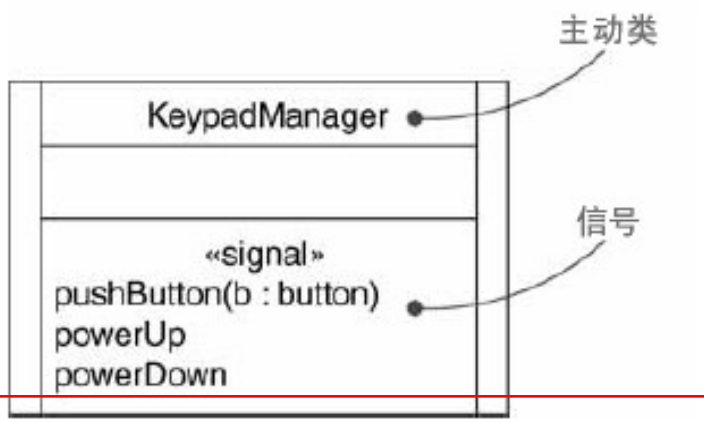
在 UML 中，可将信号模型化为具有名字 <<signal>> 衍型类 , 例如 :



并可用依赖衍型 <<send>> 来表示一个操作发送了一个特定的信号



在 UML 中，可以通过在类的附加栏中对信号命名来为对象可能接受的、有名的信号进行模型化。

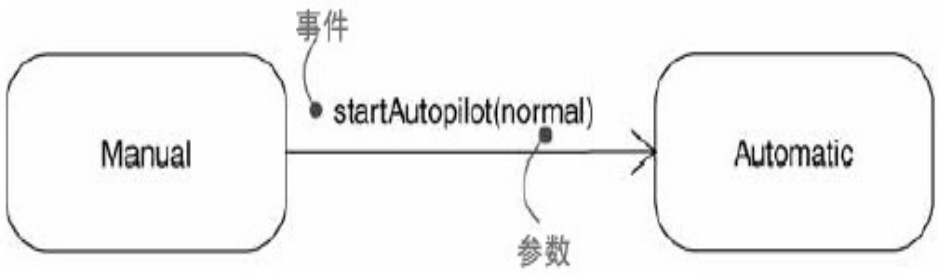


❷ 调用（call）

一个调用事件表示对象接受到一个操作调用的请求。

几点说明：

- 可以使用在类的定义中的操作定义来规约调用事件．
  - 该事件或触发状态机中的一个状态转换，或调用目标对象的一个方法。
- ――“信号”是一种异步事件，而“调用”一般是同步事件，但可以把

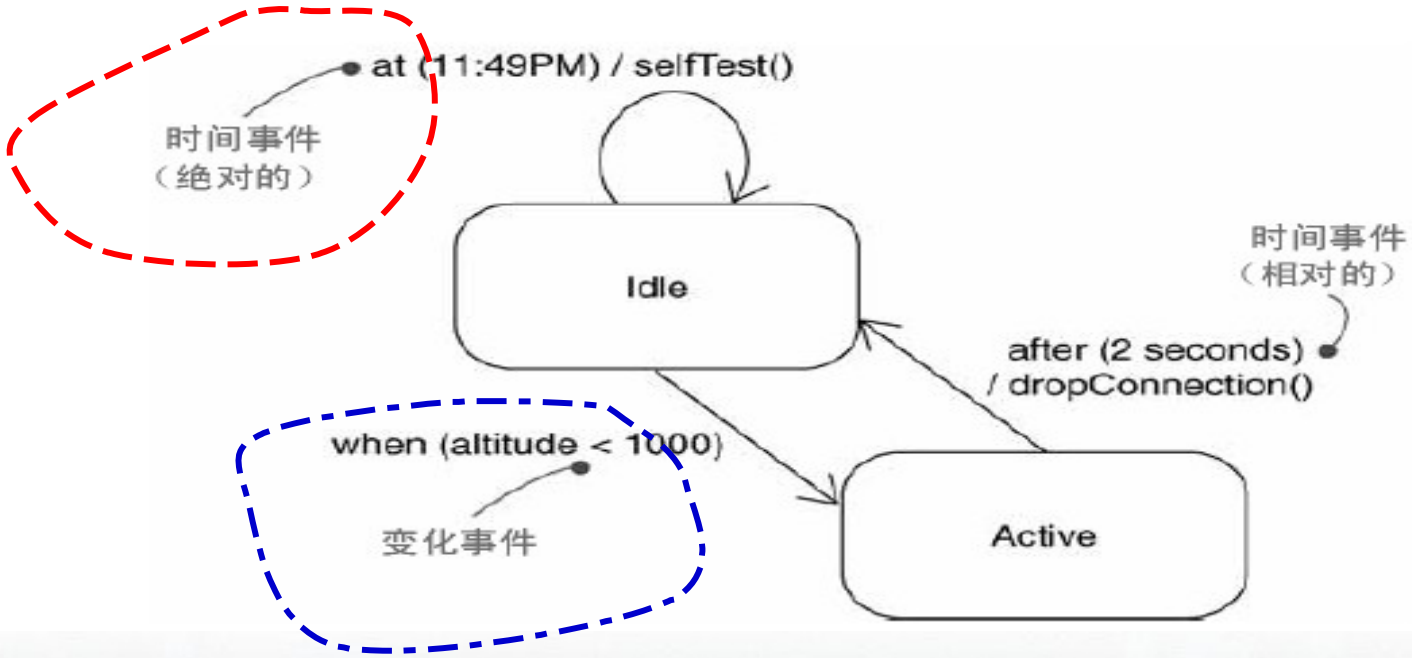


在 UML 中，将一个对象可能接受的调用事件模型化为该对象类的一个操作。

❸ 时间事件和变化事件

时间事件是表示推移一段时间的事件。

时间表达式可以是复杂的，也可以是简单的，例如：after 2 seconds,at(1 jan 2007,12.00)).



变化事件是表示一个条件得到满足或表示状态的一个变化。

❹ 发送事件和接受事件

发送事件是表示类的一个实例发送一个调用事件或信号事件。接受事件是表示类的一个实例接受一个调用事件或信号事件。

- 如果是一个同步调用事件，那么发送者和接受者都处在该操作执行期间的一个汇聚点上，即发送者的控制流一直被挂起，直到该操作执行完成；
- 如果是一个信号事件，那么发送者和接受者并不汇合，即发送者发送出信号后并不等待接受者的响应。
- 在以上两种情况下，事件可能被丢失（如果没有定义对该事件的响应的话），事件的丢失可能触发接受者的状态机（如果有的话）或引起一个常规的方法调用。

（3）状态转换

① 定义：一个状态转换是两个状态间的一种关系，指明：在第一个状态中的一个对象将执行一些确定的动作，当规约的事件发生并规约的条件满足时，进入第二个状态。

② 状态转换的规约：

一般涉及以下 5 个部分：

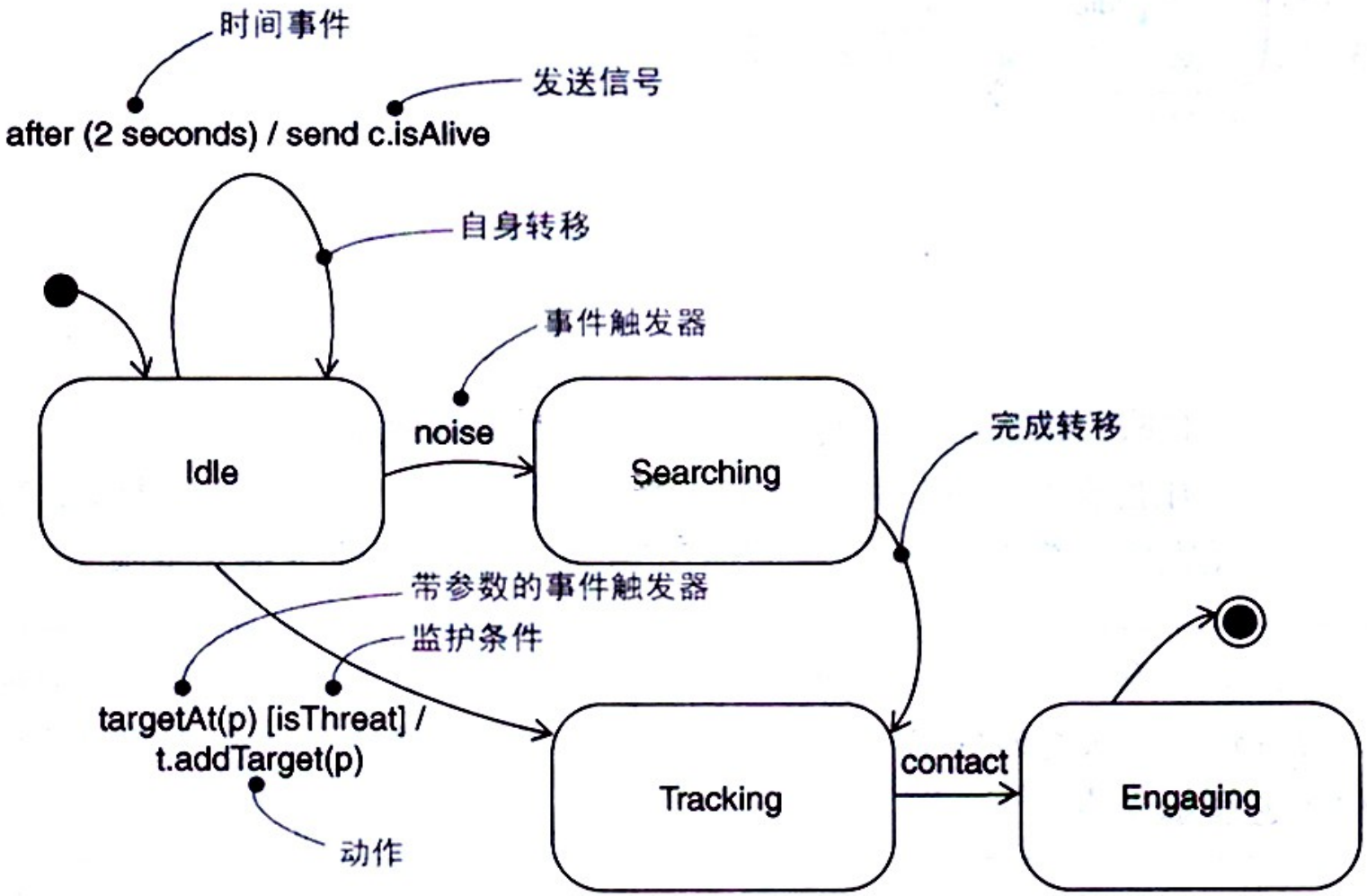
❶ 源状态：引发该状态转换的那个状态。

❷ 转换触发器：在源状态中由对象识别的事件，并且一旦满足其监护条件，则使状态发生转换。其中，在同一个简单状态图中，如果触发了多个转换，“点火”的是那个优先级最高的转换；如果这多个转换具有相同的优先级，那么就随机地选择并“点火”一个转换。

❸ 监护（guard）条件：一个布尔表达式，当某个转换触发器接受一个事件时，如果该表达式有值为真，则触发一个转换；若有值为假，则不发生状态转换，并且此时如果没有其它可以被触发的转换，那么该事件就要丢失。

❹ 效应（effect）：一种可执行的行为。例如可作用于对象上的一个动作，或间接地作用于其他对象的动作，但这些对象对那个对象是可见的。

❺ 目标状态：转换完成后所处的那个状态。



③ 表达及其格式：

在 UML 中，把状态转换表示为从源状态出发、并在目标状态上终止的带箭头的实线。转换可以予以标记，其格式为：

转换触发器 ‘ [‘ 监护条件’]’ ‘/’ 动作表达式

其中：

- 转换触发器：描述带参数的事件。其格式为：

  事件名‘ (‘ 由逗号分隔的参数表‘)’

- 监护条件：通常是一个布尔表达式，其中可以使用事件参数，也可以使用具有这个状态机的对象之属性和链，甚至可在监护条件处直接指定对象可达的某个状态，例如：

  “in State1” 或“ not in State2” 。

- 动作表达式：给出触发转换时所执行的动作，其中可以使用对象属性、操作和链以及触发事件的参数，或在其范围内的其它特征。

6.3.4.3 状态图的一般用法

① 建立一个系统动态方面的模型，这些动态方面包括任意种类对象、任意系统结构（类、接口、构件和节点）视角下以事件定序的行为。

② 建立一个场景的模型，其主要途径是针对 use case 给出相应的状态图。

其中，不论是①还是②，通常都是对反应型对象（reactive object）的行为进行建模。

反应型对象，或称为事件驱动的对象，其行为特征是响应其外部语境中所出现的事件，并作出相应的反应。

为反应型对象建立一个状态机模型时，其步骤为：

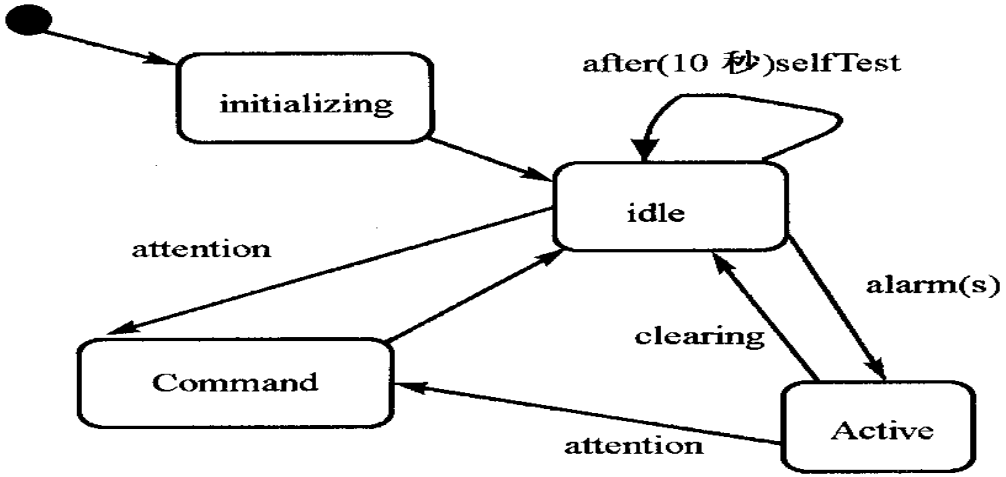
- 选择状态机的语境，即是类、是用况或是子系统。
- 选择其实例（例如类的对象）的初始状态和最终状态，并分别给出初始状态和最终状态的前置条件和后置条件，以便指导以后的建模工作。
- 标识某一可用的时间段，考虑该实例在此期间内存在的条件，以此判断该实例的其它状态。对此应以该实例的高层状态开始，继之再考虑这些状态的可能的子状态。
- 判断该实例在整个生存周期内所有状态的有意义的偏序。
- 判断可以触发状态转换的事件。可以逐一从一个合理定序的状态到另一个状态，来模型化其中的事件。

- 为这些状态转移填加动作，或为这些状态填加动作。
- 使用子状态、分支、合并和历史状态等，考虑简化状态机的方法。
- 检查在某一组合事件下所有状态的可达性。
- 检查是否存在死状态，即不存在任何组合事件可以使该实例从这一状态转换到任一其它状态。
- 跟踪整个状态机，检查是否符合所期望的事件次序和响应。

应用实例：创建一个控制器状态机的状态图，其中该控制器负责对一些传感器进行监视。

- 当创建这种控制器的一个对象时，由于要做一些初始化工作，因此该对象首先进入“初始化”（intializing）状态；
- 一旦完成初始化活动后，则自然进入“休眠”（Idle）状态；
- 在休眠状态中，按一定时间间隔，接受传感器的 alarm 事件 (具有参数 S，表示相关的传感器)，一旦当接收到一个 alarm 事件，或接受到用户发送的 attention 信号，控制就从休眠状态转移到“活化”（Active）状态，或转移到“命令”（Command）状态；
- 在活化状态中，仅当发生 clearing 事件或需要发送 attention 信号时，分别进入休眠状态或命令状态。

在嵌入式系统中，以上是一种常见的控制行为，其状态图如下所示：



其中：♦ 在该状态图的休眠状态上，有一个由时间事件触发的自转移，意指每隔 10 秒，接受传感器的 selfTest 事件。

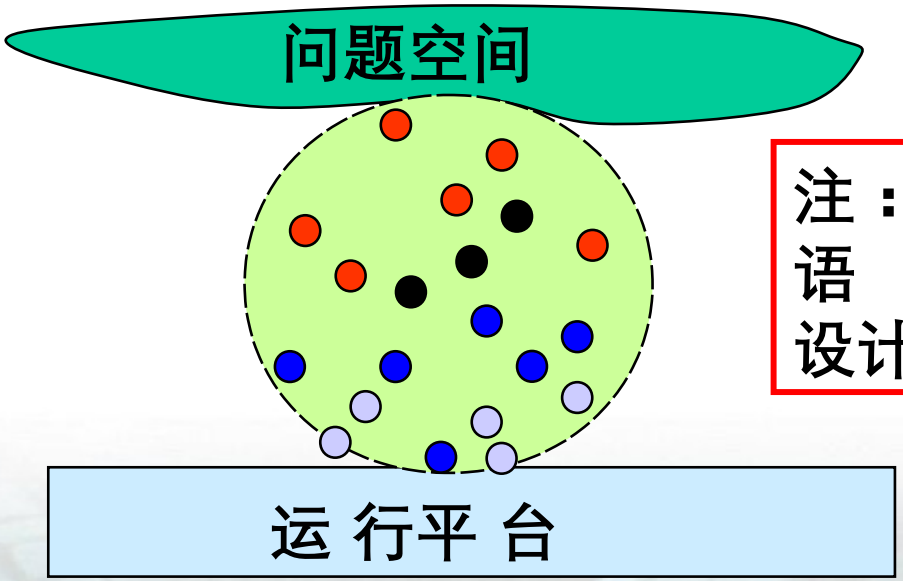
♦ 该状态图没有终止状态，意指该控制器不间断地运行。

6.4 UML 总结：

①UML 的作用

对“自顶向下”的建模人员来讲：

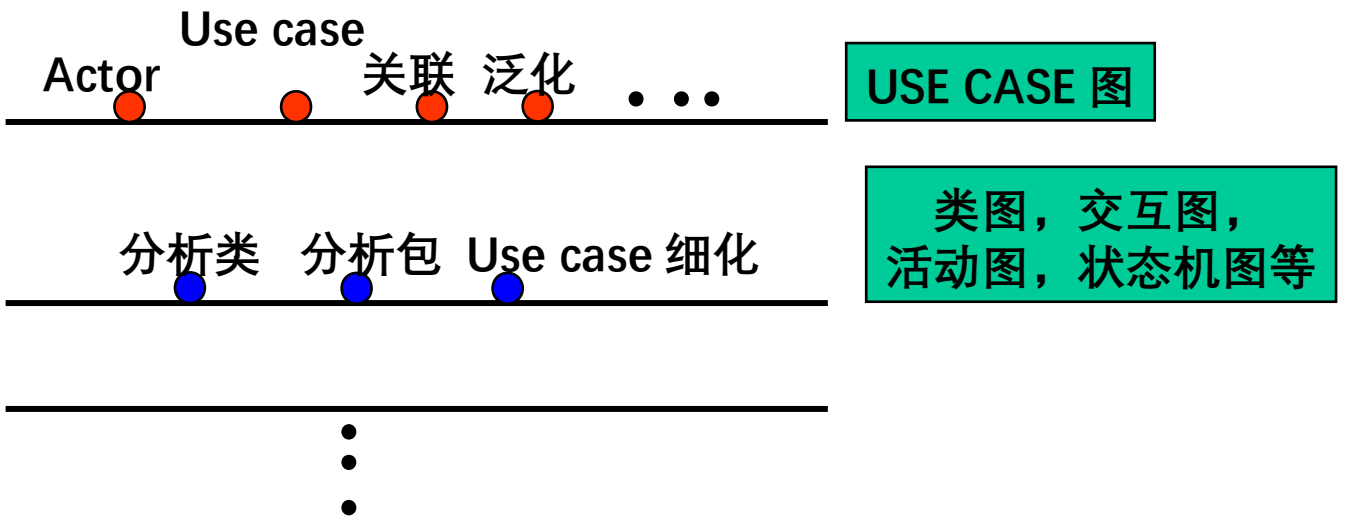
－－提供了跨越问题空间到目前“运行平台”之间丰富的建模元素．基于给定的术语，可确定不同的抽象层次，支持“概念建模”和“软件建模”。



注：UML 提供的术语 -- 体现了软件设计的不同原理

－－提供了相应的模型表示工具。

例如：



即紧紧围绕“面向对象方法是一种以对象和对象关系来创建系统模型的系统化软件开发方法学”，给出表达“对象”、“对象关系”的术语，并给出了表达模型的工具，其主要目的是：支持软件开发人员从不同目的（静态、动态）、针对不同粒度（系统、子系统、类目等），从不同抽象层和从不同视角来创建模型，并建立相应的文档。

从许多事物中舍弃个别的、非本质的特征，
抽取共同的、本质性的特征，就叫做抽象

具体地说：

❶ 为了支持抽象系统分析和设计中的事物， UML 给出了 8 个基本术语，即：类、接口、协作、用况、主动类、构件、制品、结点，并给出了这些基本术语的一些变体。

每个术语都体现着一定的软件设计原理，例如：

- 类体现了数据抽象、过程抽象、局部化以及信息隐蔽等原理

（1）过程抽象

任何一个完成确定功能的操作序列，其使用者都可把它看作一个单一的实体，尽管实际上它可能是由一系列更低级的操作完成的。

过程抽象不是 OOA 的主要抽象形式，因为 OO 方法不允许超出对象的界限在全系统范围内进行功能的描述。但过程抽象对于在对象范围内组织对象的操作是有用的。

（2）数据抽象

根据施加于数据之上的操作来定义数据类型，并限定数据的值只能由这些操作来修改和观察。

数据抽象是 OOA 的核心原则。它强调把数据（属性）和操作结合为一个不可分的系统单位（即对象），对象的外部只知道它做什么，而不知道它怎么做。

- 用况体现了问题分离、功能抽象等原理；
- 接口体现了功能抽象等。当使用这些术语创建系统模型时，其语义就映射到相应的模型元素。

❷ 为了表达模型元素之间的关系， UML 给出了 4 个术语，即：关联、泛化、细化和依赖，以及它们的一些变体。可以作为 UML 模型中的元素，用于表达各种事物之间的基本关系。

这些术语都体现了结构抽象原理，特别是泛化概念的使用，可以有效地进行“一般 / 特殊”结构的抽象，支持设计的复用。并且为了进一步描述这些模型元素的语义，还给出一些特定的概念和表示，例如给出限定符这一概念，是为了增强关联的语义。

❸ 为了组织以上两类模型元素， UML 给出了包这一术语，在实际应用中，可以把包作为控制信息复杂性的机制。

❹ 为了使创建的系统（或系统成分）模型清晰、易懂， UML 给出了注解这一术语。

❺ 为了表达概念模型和软件模型， UML 提供了 13 种图形化工具，它们是：类图、对象图、构件图、包图、部署图、组合结构图，以及 USE CASE 图、状态图、顺序图、通讯图、活动图、交互概观图，定序图。

前 6 个图可用于概念模型和软件模型的静态结构方面；而后 7 个模型可用于概念模型和软件模型的动态结构方面 . 其中：

♦ 类图可用于创建系统的结构模型，表达构成系统各成分之间的静态关系，给出有关系统（或系统成分）的一些说明性信息；

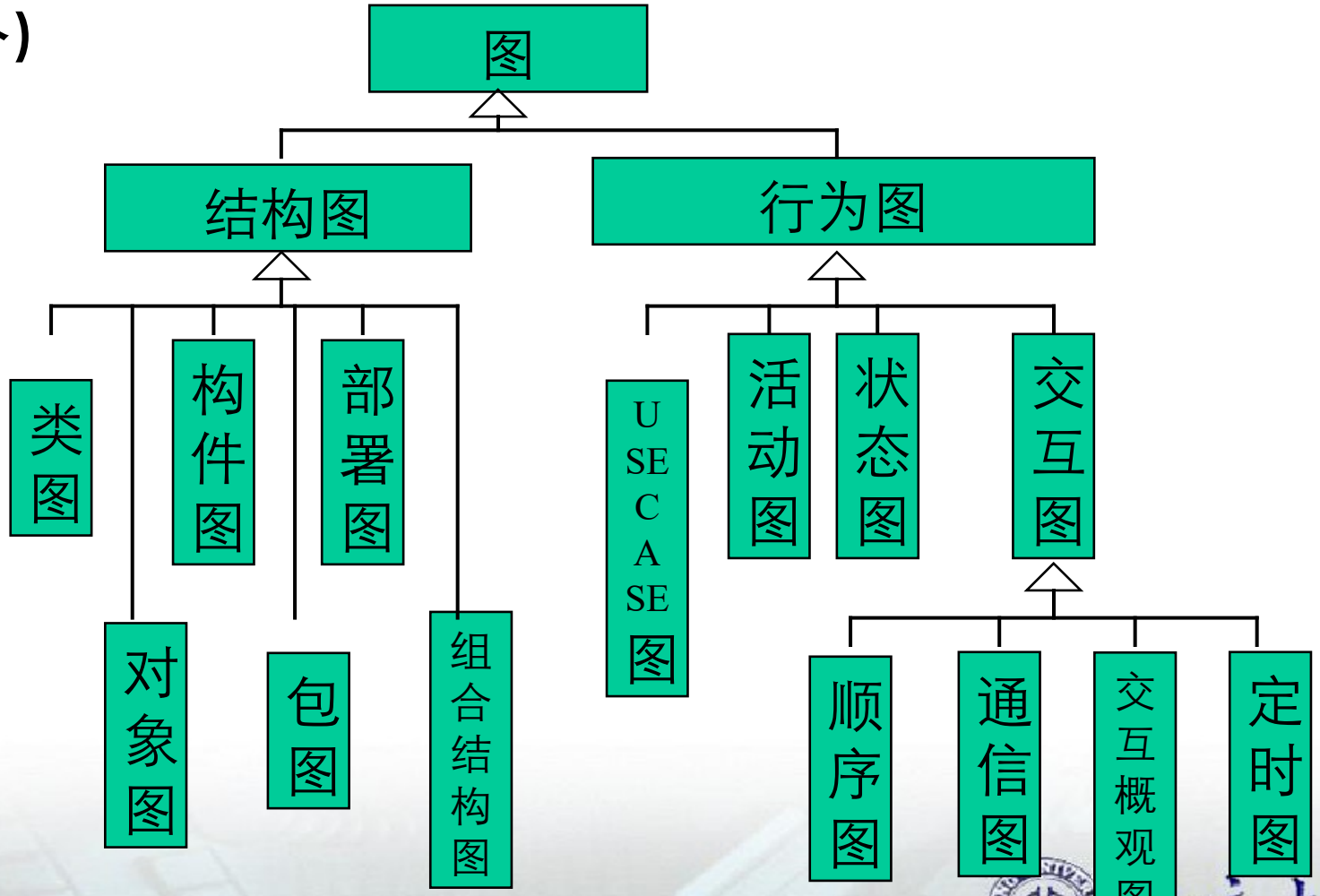
♦use case 图可用于创建有关系统（或系统成分）的功能模型，表达系统（或系统成分）的功能结构，给出有关系统（或系统成分）在功能需求方面的信息；

♦ 状态图可用于创建有关系统（或系统成分）的行为生存周期模型，表达有关系统（或系统成分）的一种动态结构，给出有关系统（或系统成分）在生存期间可有哪些阶段、每一阶段可从事的活动以及对外所呈现的特征等方面的信息；

♦ 顺序图可用于创建有关系统（或系统成分）的交互模型，表达有关系统（或系统成分）的一种交互结构，给出有关参与交互的各方、交互方式以及交互内容．

对“自底向上”的设计思想交流来讲：

－－提供了表达系统结构模型和行为模型的图形化工具（13 个）



②UML 与 RUP

RUP 是一种基于 UML 的一种过程框架，它比较完整地定义了将用户需求转换成产品所需要的活动集，并提供了活动指南以及对产生相关文档的要求．