

# 一、UML概述

+ | ★ 收藏 | 1312 | 210

## 统一建模语言

语音

编辑

讨论

上传视频

**同义词** UML（统一建模语言）一般指统一建模语言

本词条由“科普中国”科学百科词条编写与应用工作项目 审核。

统一建模语言(Unified Modeling Language, UML)是一种为**面向对象**系统的产品进行说明、可视化和**编制**文档的一种标准语言，是非专利的第三代建模和规约语言。UML是面向对象设计的建模工具，独立于任何具体程序设计语言。 [1]

作品名称	统一建模语言	创作年代	1997年
外文名	UML	作 用	支持模型化和软件开发
作品别名	标准建模语言	产 源	OOA&D, OOAD

软件开发生命周期：做需求 -》 形成文档 -》 系统设计 -》 开发人员编写代码 -》 测试 -》 运维...

系统设计 -》 “画图纸” -》 UML -》 图形化语言（图标式语言） -》 不仅应用于Java -》 程序员根据设计开始开发/编码

## 二、UML建模工具

能够实现UML图的建模工具有：

IBM - Rational Rose

Sybase - Power Designer

韩国 - StarUML(简称：SU)

MS - Visio

枫叶云 (最新下载地址:www.fynote.com)

## 三、常见UML图\_类图

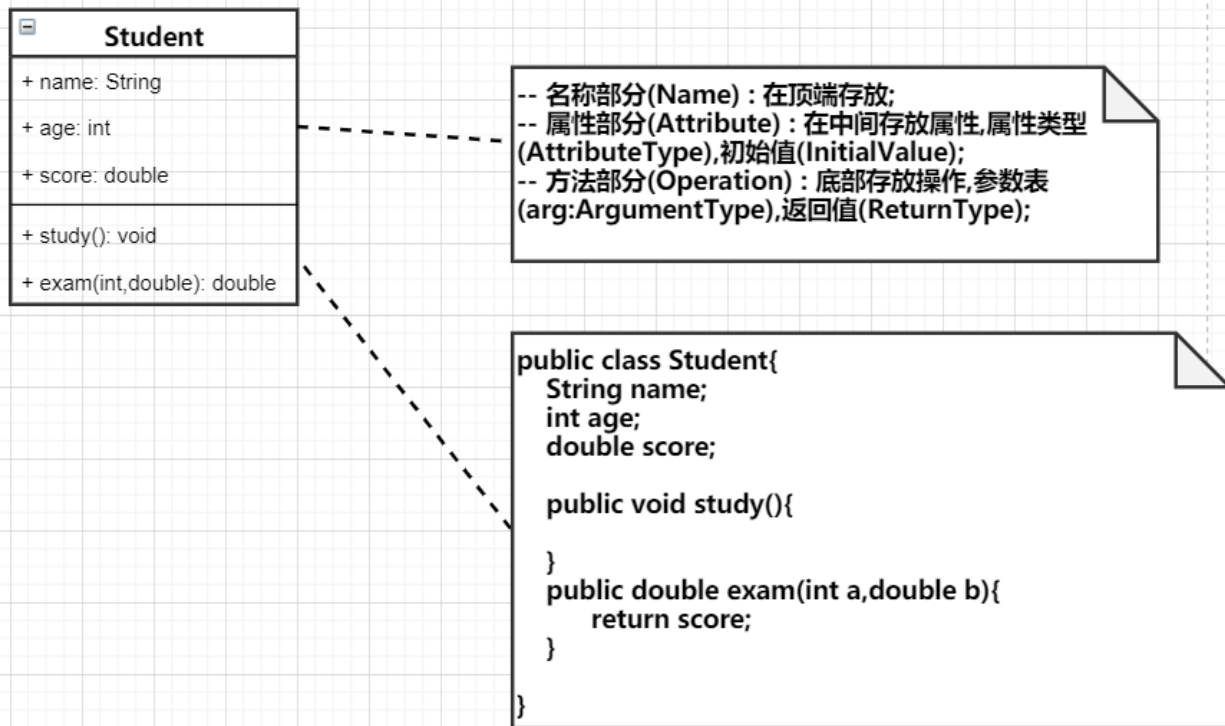
(1) 什么是类图？

定义系统中的类，描述类的内部结构（属性、方法等），表示类之间的关系（泛化、实现、依赖、关联、聚合、组合）；

(2) 类在UML中表示：

- 名称部分(Name) : 在顶端存放;
- 属性部分(Attribute) : 在中间存放属性,属性类型(AttributeType),初始值(InitialValue);
- 方法部分(Operation) : 底部存放操作,参数表(arg:ArgumentType),返回值(ReturnType);

\*\* ( 3 ) 注释部分 : \*\*解释说明



#### ( 4 ) 属性部分 :

( 4-1 ) 属性语法 : [可见性]属性名[ :类型][ =初始值][ {属性字符串} ] ;

-- 注意 : [] 中的内容可有可无 ;

-- 属性字符串用来指定关于属性的其它信息 , 不一定是属性值 , 如果希望添加一个属性定义规则 , 但是没地方添加 , 可以写在属性字符串中 ;

#### ( 4-2 ) 可见性 :

属性的可见性只有公有(public + ),私有(private - ),受保护(protected # ), UML中不存在默认 , 如果没有显示任何符号 , 就表示没有定义该属性;

- 公有 : 用 "+" 表示 , 可以在此类的外部使用查看该属性;
- 私有 : 用 "-" 表示 , 不可以从外部类中访问该属性;
- 保护 : 用 "#" 表示 , 常与 泛化一起使用;

#### ( 5 ) 方法部分 :

**( 5-1 ) 方法语法 :** [可见性]操作名[( 参数表 )][ : 返回类型][ { 属性字符串 }]

-- 注意 : [] 中的内容可有可无 ;

-- 注意 : 如果有多个参数列表的话 , 中间用逗号隔开

**( 5-2 ) 可见性 :**

可见性 : 主要包括 公有(public +), 私有(private -), 受保护(protected #), 包内公有(package ~);

-- 公有 : 用 "+" 表示, 只要调用对象能访问操作所在的包, 就能访问公有操作;

-- 私有 : 用 "-" 表示, 同一个类的对象才能调用私有的操作;

-- 保护 : 用 "#" 表示, 子类对象才可以调用受保护操作;

-- 包内 : 用 "~" 表示, 同一个包内的对象才可以调用包内公有的操作;

## 四、类图之类和类之间的关系

### ( 1 ) 继承关系 ( 泛化关系 Generalization )

a) 语义 :

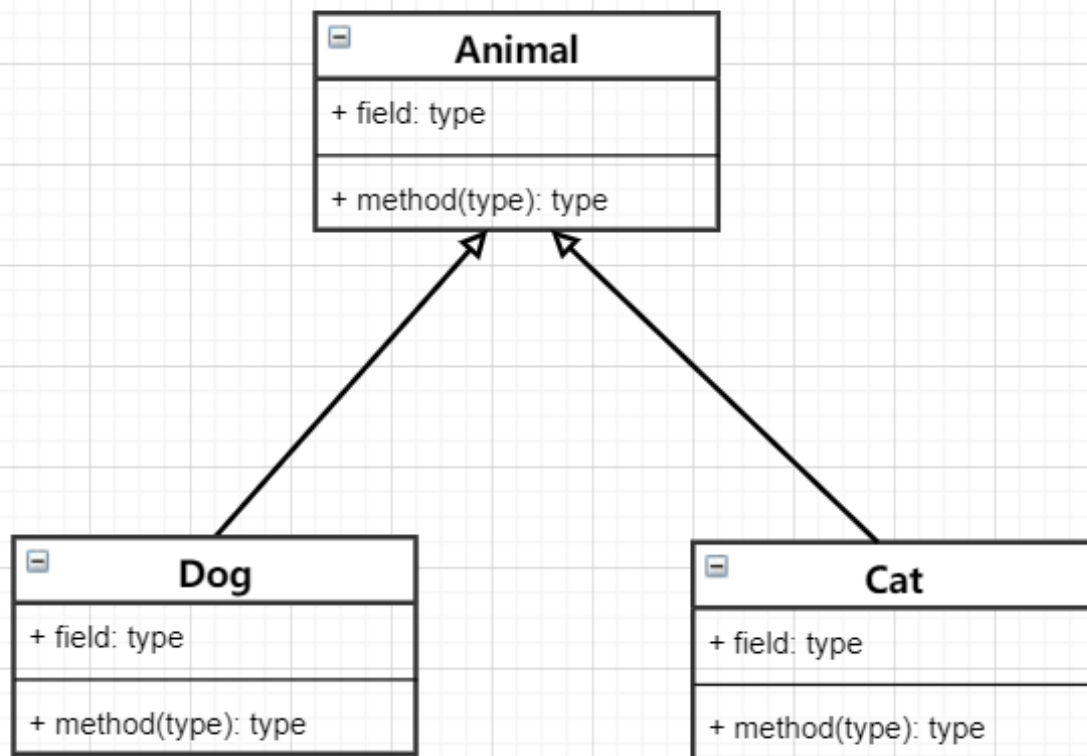
i. 类和子类的关系 , 接口和子接口的关系 ;

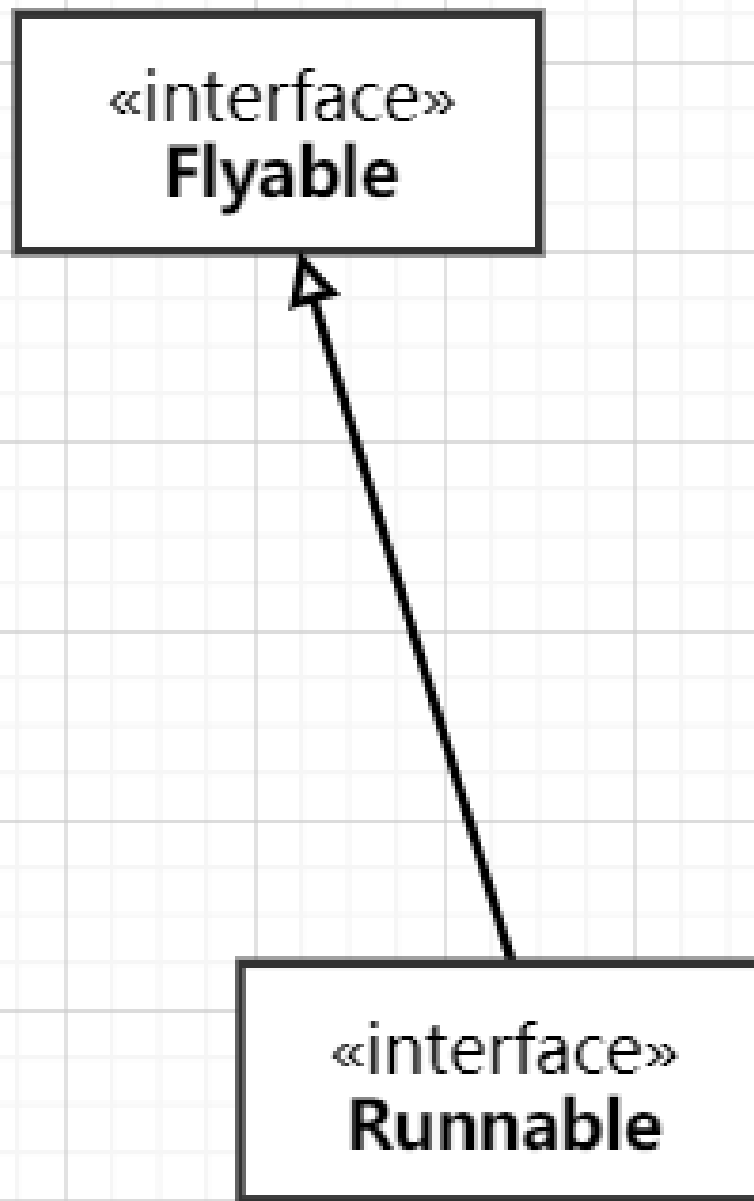
ii. 一个类 ( 称为子类、子接口 ) 继承另外的一个类 ( 称为父类、父接口 ) 的功能 , 并可以增加它自己的新功能

b) 语法 : extends

c) 符号 :

i. 一条带空心三角箭头的实线 , 从子类指向父类 , 或者子接口指向父接口。





## (2) 实现关系

a) 语义：

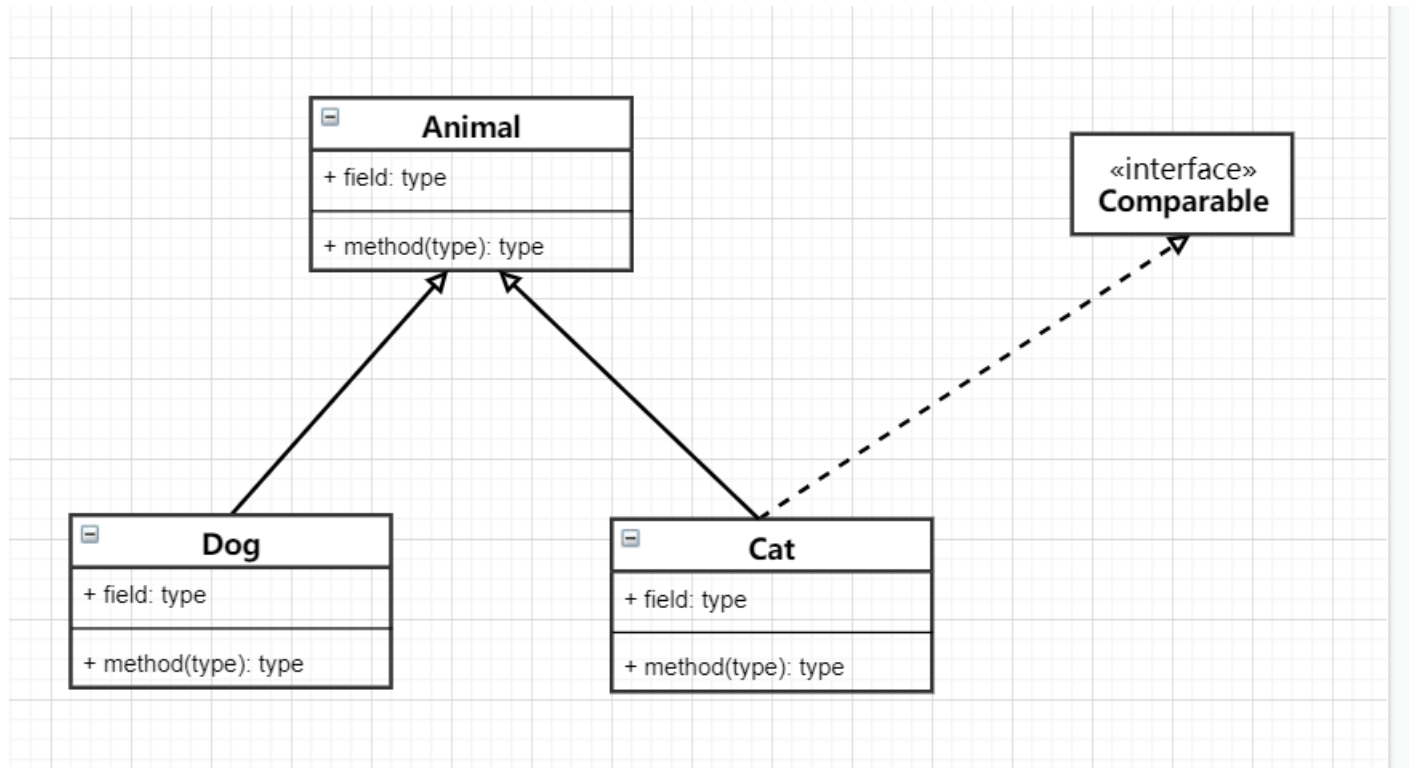
- i. 类和接口之间的关系；
- ii. 一个类可以实现多个接口，实现所有接口的功能；体现了规范和实现分离的原则

b) 语法：implements

c) 符号

i.

实现用一条带空心三角箭头的虚线表示，从类指向实现的接口



### (3) 依赖关系

a) 语义：一个类A使用到了另一个类B，但是这种使用关系是具有偶然性的、临时性的、非常弱的，但是类B的变化会影响到类A

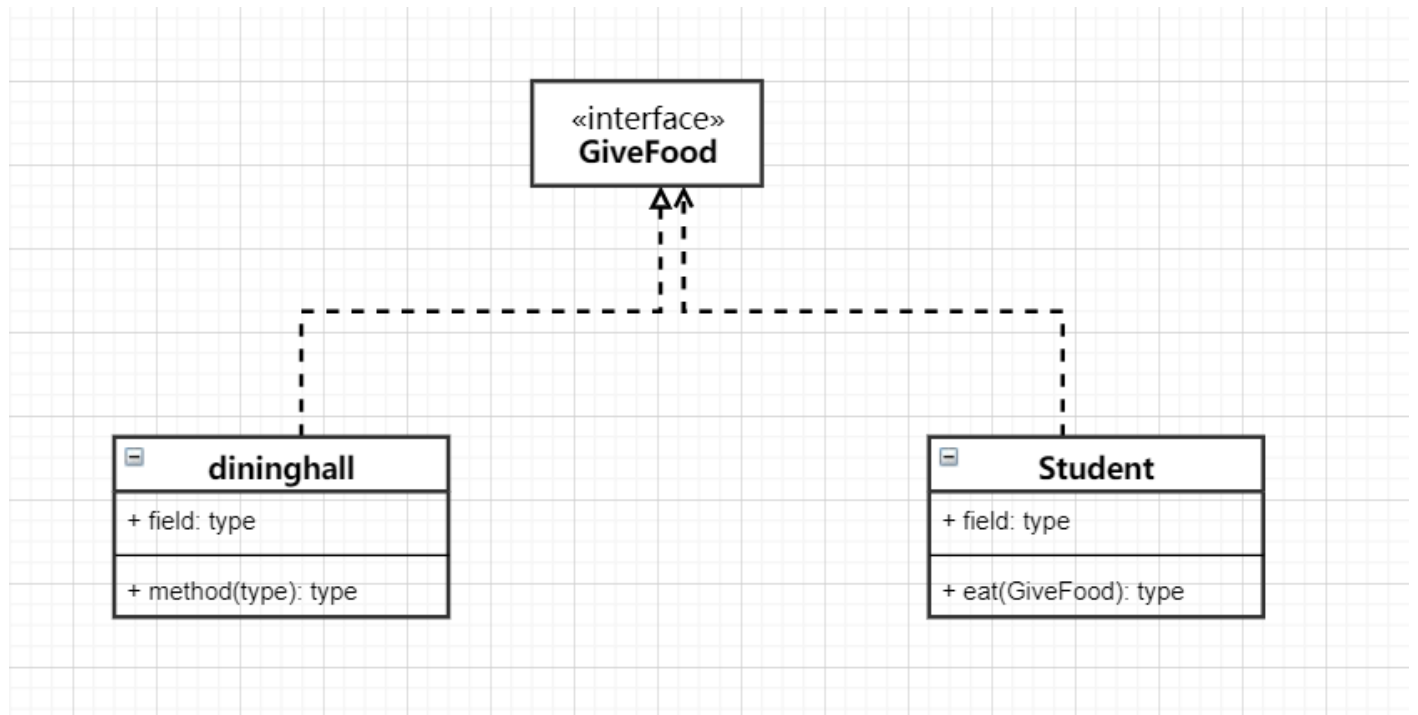
举例：学生 -> 方法：吃饭（食堂）

b) 语法：类B作为类A的方法的参数（或者局部变量）存在

c) 符号：

i.

由类A指向类B的带箭头虚线表示



## (4) 关联关系

a) 语义：

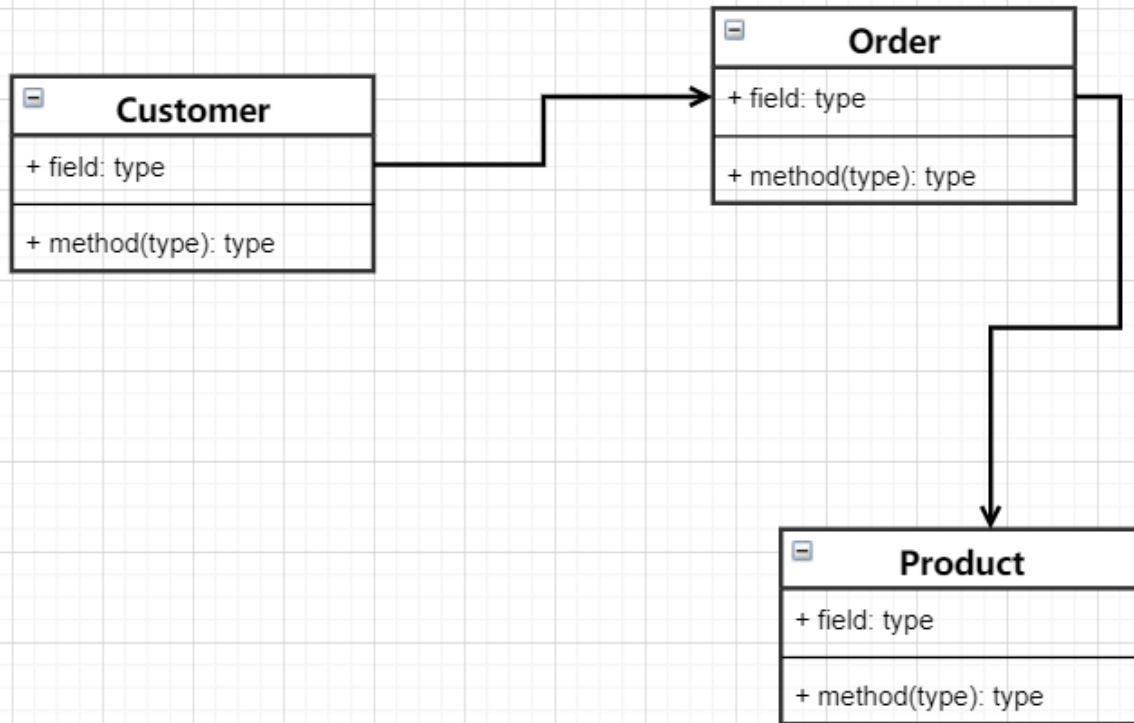
比依赖关系强，必然的，长期的，强烈的；

举例：顾客 -》 订单-》 商品

b) 语法：类B作为成员变量形成存在于类A中

c) 符号：

由类A指向类B的带箭头实线表示；



## ( 5 ) 聚合关系

a) 语义：

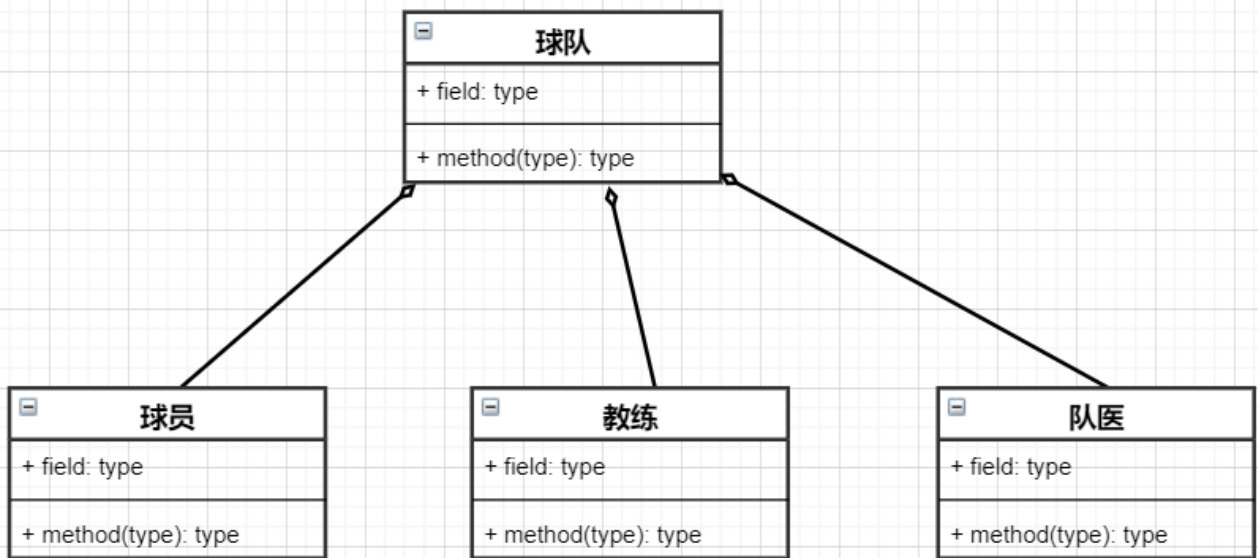
- i. 关联关系的一种特例（一个类作为另一个类的成员变量）
- ii. 整体和部分的关系
- iii. 整体部分可分离，整体的生命周期和部分的生命周期不同，has-a的关系
- iv. 计算机与CPU、公司与员工的关系、班级和学生的关系、球队和球员

b)

语法：同关联关系

符号：空心菱形加实线





## (6) 组合关系

a) 语义：

i. 关联关系的一种特例

ii. 整体和部分关系、整体部分不可分离、比聚合更强，contains-a的关系

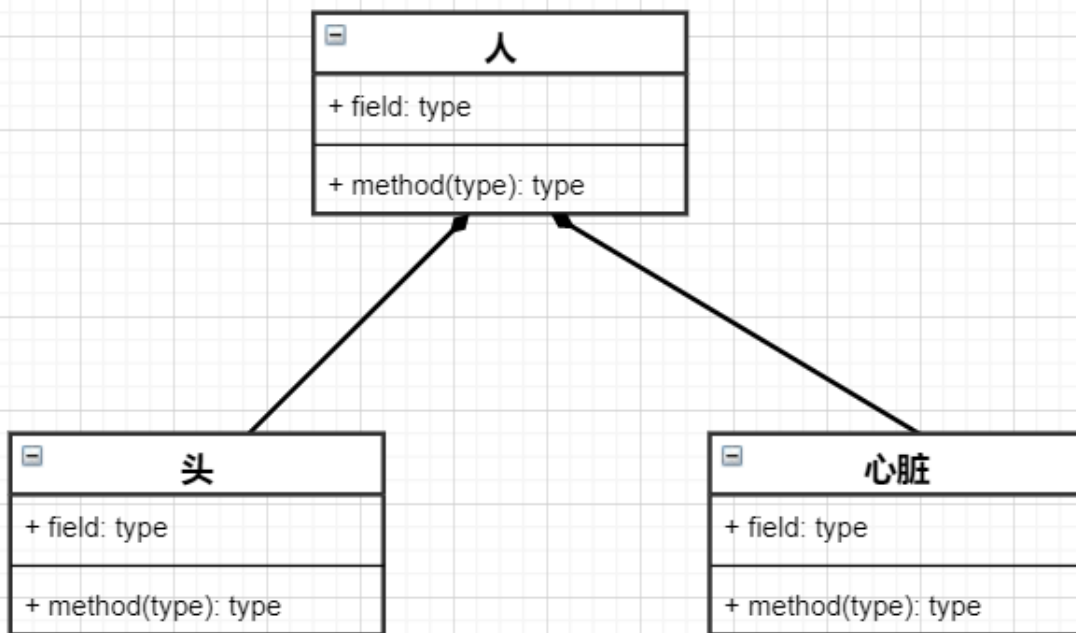
iii. 整体的生命周期和部分的生命周期相同

iv. 人和四肢的关系

b)

语法：同关联关系

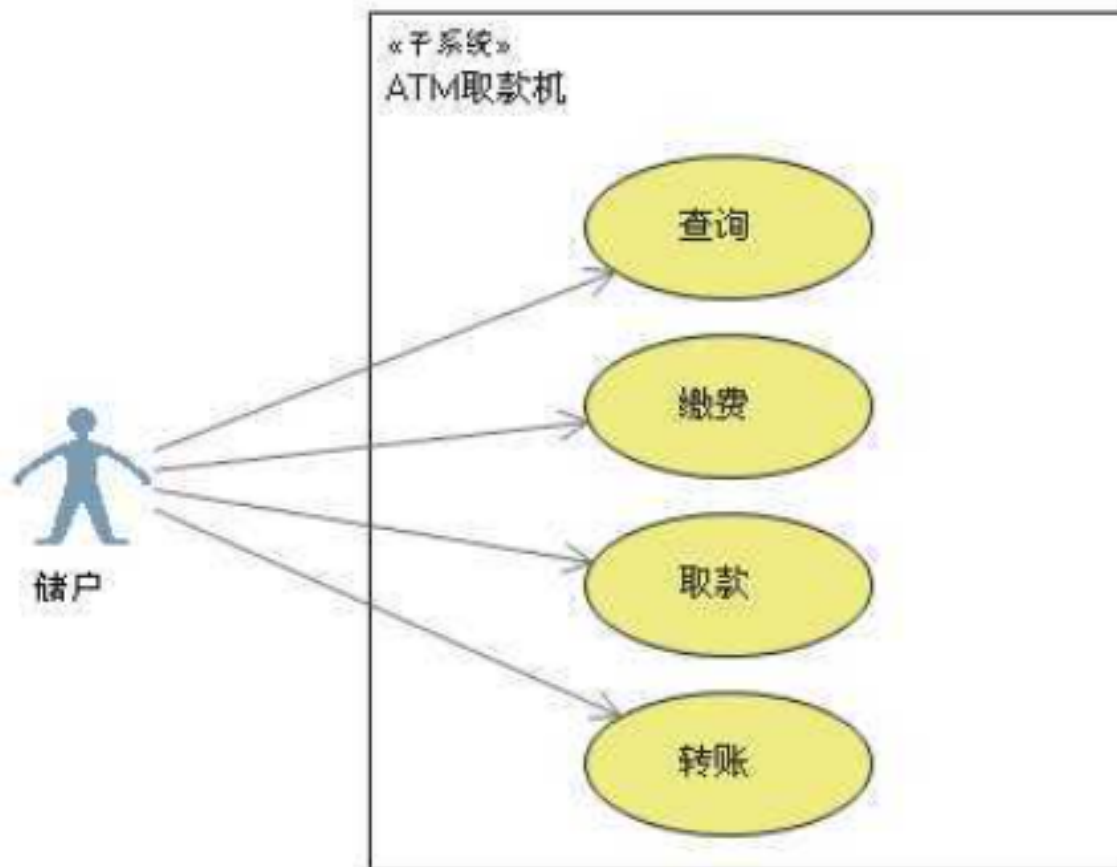
符号：实心菱形加实线



## 五、常见UML图\_用例图

### 1、用例图是什么？

用例图是指由参与者、用例，边界以及它们之间的关系构成的用于描述系统功能的视图。



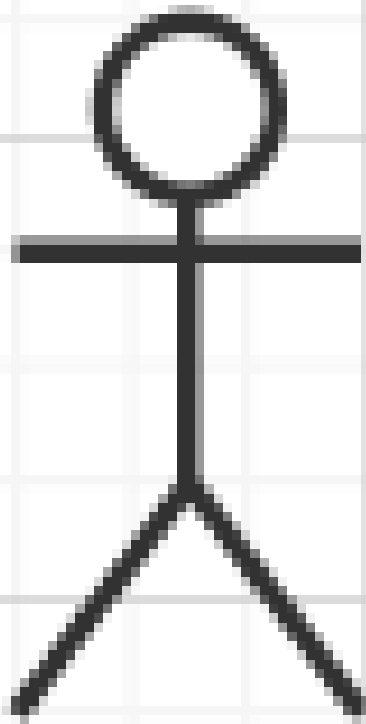


用例图主要用来描述角色以及角色与用例之间的连接关系。说明的是谁要使用系统，以及他们使用该系统可以做些什么。一个用例图包含了多个模型元素，如系统、参与者和用例，并且显示这些元素之间的各种关系，如泛化、关联和依赖。它展示了一个外部用户能够观察到的系统功能模型图。

【用途】：帮助开发团队以一种可视化的方式理解系统的功能需求。


## 2、用例图所包含的元素

(1) 参与者(Actor)——与应用程序或系统进行交互的用户、组织或外部系统。用一个小人表示。




Actor

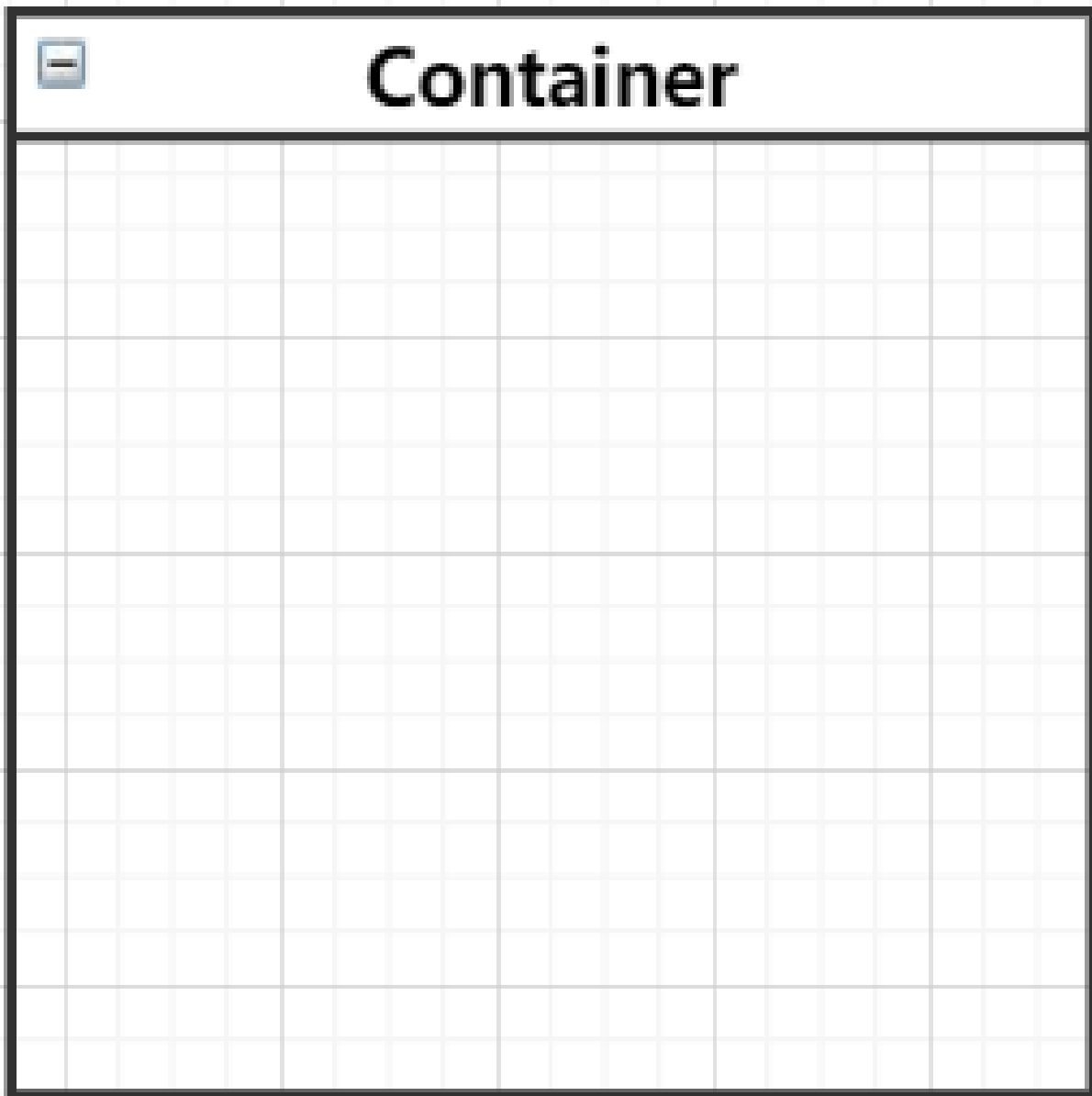
(2) 用例(Use Case)——用例就是外部可见的系统功能，对系统提供的服务进行描述。用椭圆表示。



# Use Case

( 3 ) 子系统(Subsystem)—— 用来展示系统的一部分功能，这部分功能联系紧密。(就是具体的系统功能)





### 3、用例图所包含的的关系

用例图中涉及的关系有：关联、泛化、包含、扩展。

如下表所示：

--

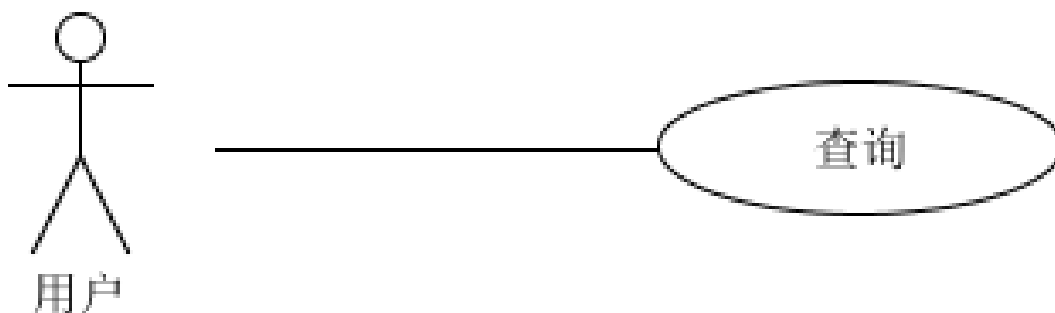


关系类型↴	说明↴	表示符号↴
关联↴	参与者与用例之间的关系↴	—————
泛化↴	参与者之间或用例之间的关系↴	—————▷
包含↴	用例之间的关系↴	---«includes»—>
扩展↴	用例之间的关系↴	---«extends»—>

### a. 关联(Association)

表示参与者与用例之间的通信，任何一方都可发送或接受消息。

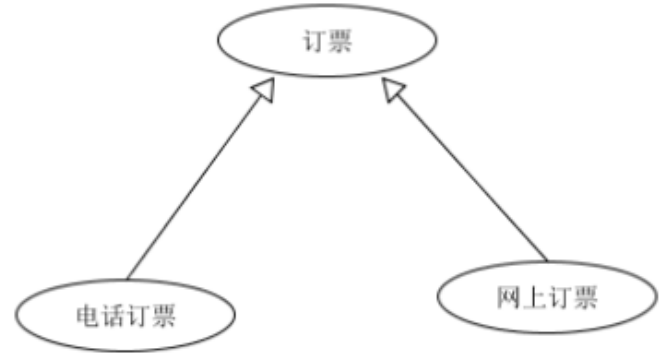
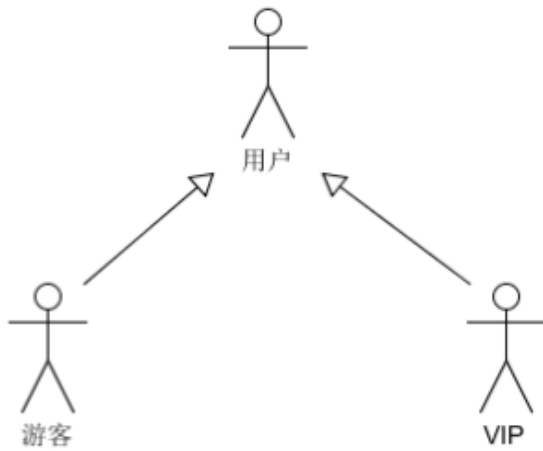
【箭头指向】：无箭头，将参与者与用例相连接，指向消息接收方



### b. 泛化(Inheritance)

就是通常理解的继承关系，子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。在实际应用中很少使用泛化关系，子用例中的特殊行为都可以作为父用例中的备选流存在。

【箭头指向】：指向父用例



PS：父类的功能少，子类的功能多

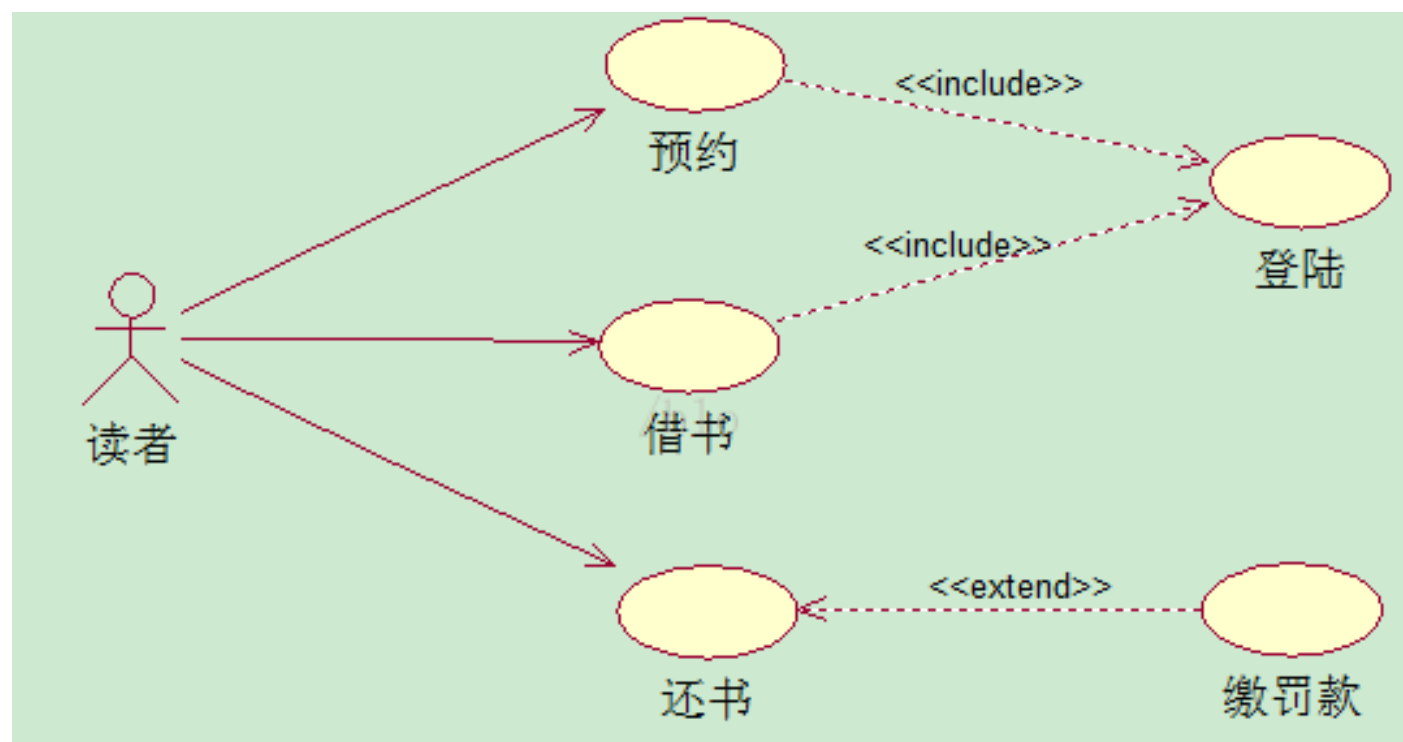
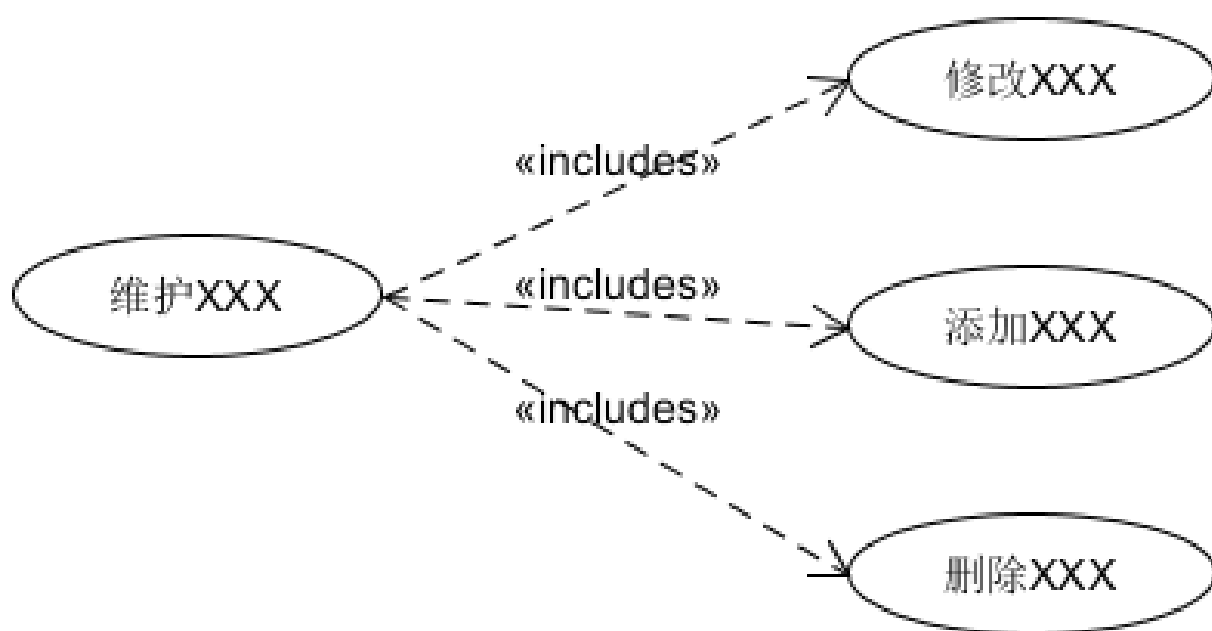
### c. 包含(Include)

包含关系用来把一个较复杂用例所表示的功能分解成较小的步骤。包含关系对典型的应用就是复用，也就是定义中说的情景。但是有时当某用例的事件流过于复杂时，为了简化用例的描述，我们也可以把某一段事件流抽象成为一个被包含的用例；相反，用例划分太细时，也可以抽象出一个基用例，来包含这些细颗粒的用例。这种情况类似于在过程设计语言中，将程序的某一段算法封装成一个子过程，然后再从主程序中调用这一子过程。

例如：业务中，总是存在着维护某某信息的功能，如果将它作为一个用例，那添加、修改以及删除都要在用例详述中描述，过于复杂；如果分成添加用例、修改用例和删除用例，则划分太细。这时包含关系可以用来理清关系。

【箭头指向】：指向分解出来的功能用例





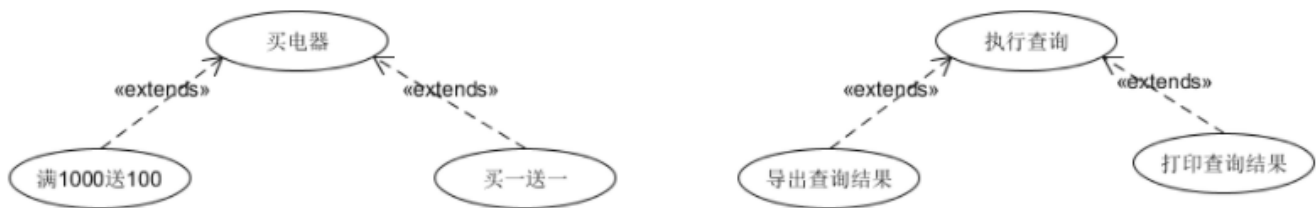
PS：预约功能、结束功能都包含 登陆功能，将通用功能（复用功能）提取

#### d. 扩展(Extend)

扩展关系是指用例功能的延伸，相当于为基础用例提供一个附加功能。将基用例中一段相对独立并且可选的动作，用扩展（Extension）用例加以封装，再让它从基用例中声明的扩展点（Extension Point）上进行扩展，从而使基用例行为更简练和目标更集中。扩展用例为基用例添加新的行为。扩展用例可以访问基用例的属性，因此它能根据基用例中扩展点的当前状态来判断是否执行自己。但是扩展用例对基用例不可见。

对于一个扩展用例，可以在基用例上有几个扩展点。

【箭头指向】：指向基础用例



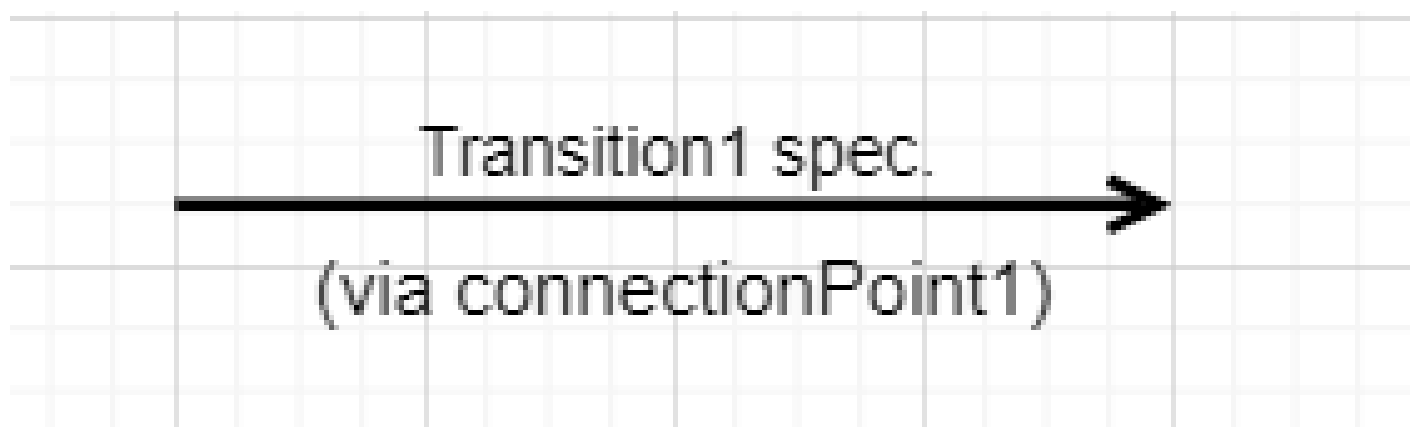
## 六、常见UML图\_状态图

### 1、状态图是什么？

状态图(statechart diagram)：用来描述一个特定的对象所有可能的状态，以及哪些事件将导致状态改变。

### 2、状态图的元素

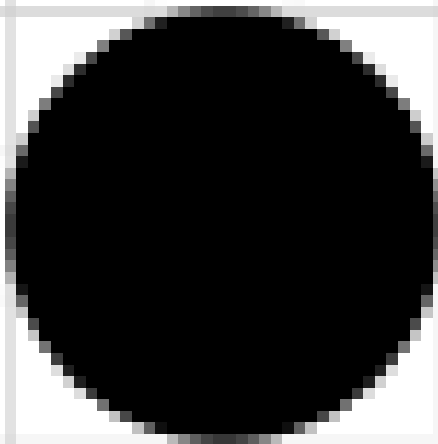
- (1) 箭头表示一个转换/一个动作
- (2) 箭头上的文字：表示一个事件



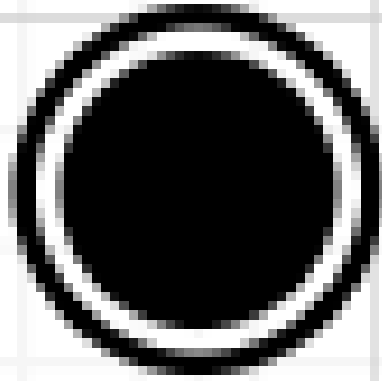
- (3) 长方形表示某种状态

# State

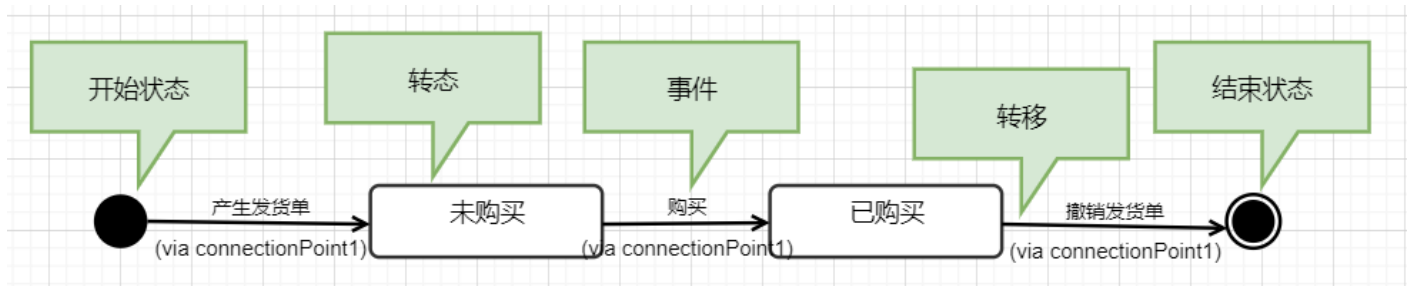
(4) 起始状态：是一种伪状态，只是表示从这里要开始（可选）



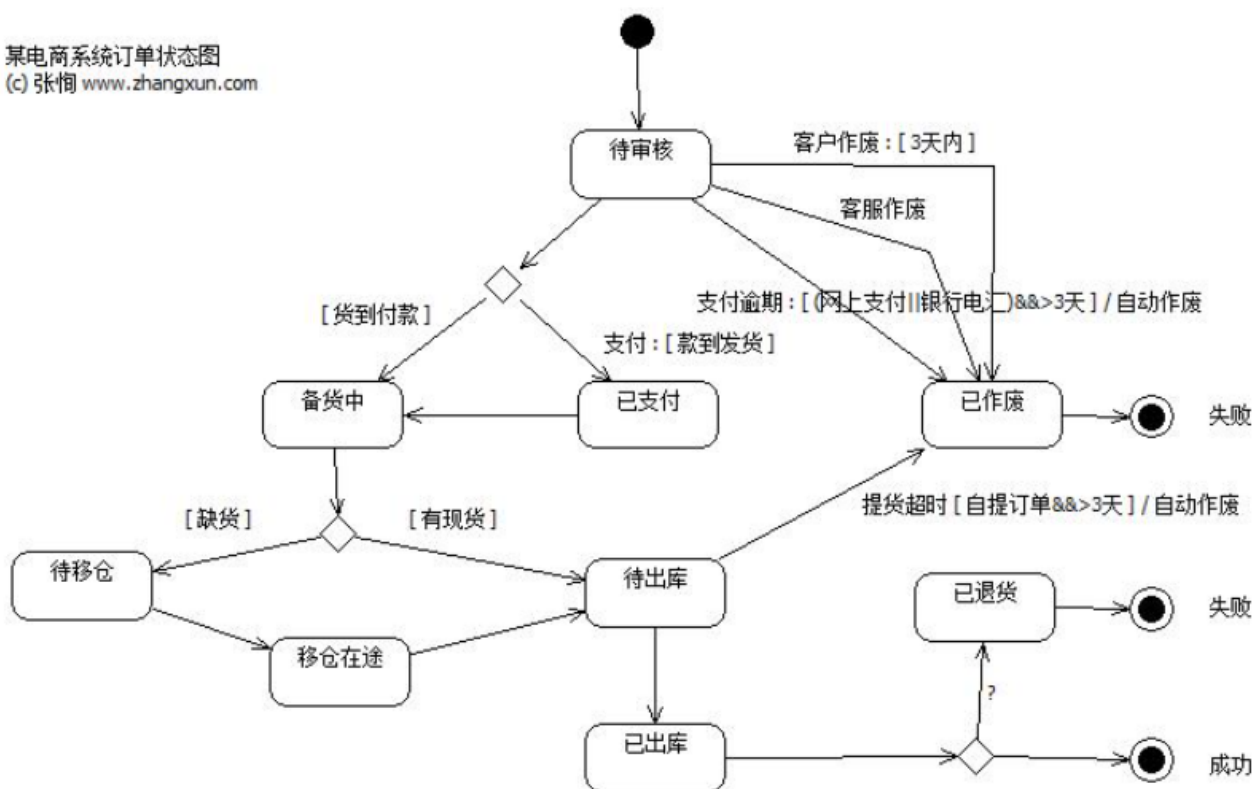
(5) 结束状态：是一种伪状态，只是表示从这里要结束（可选）



示例：



(以下图片截取自百度图库)



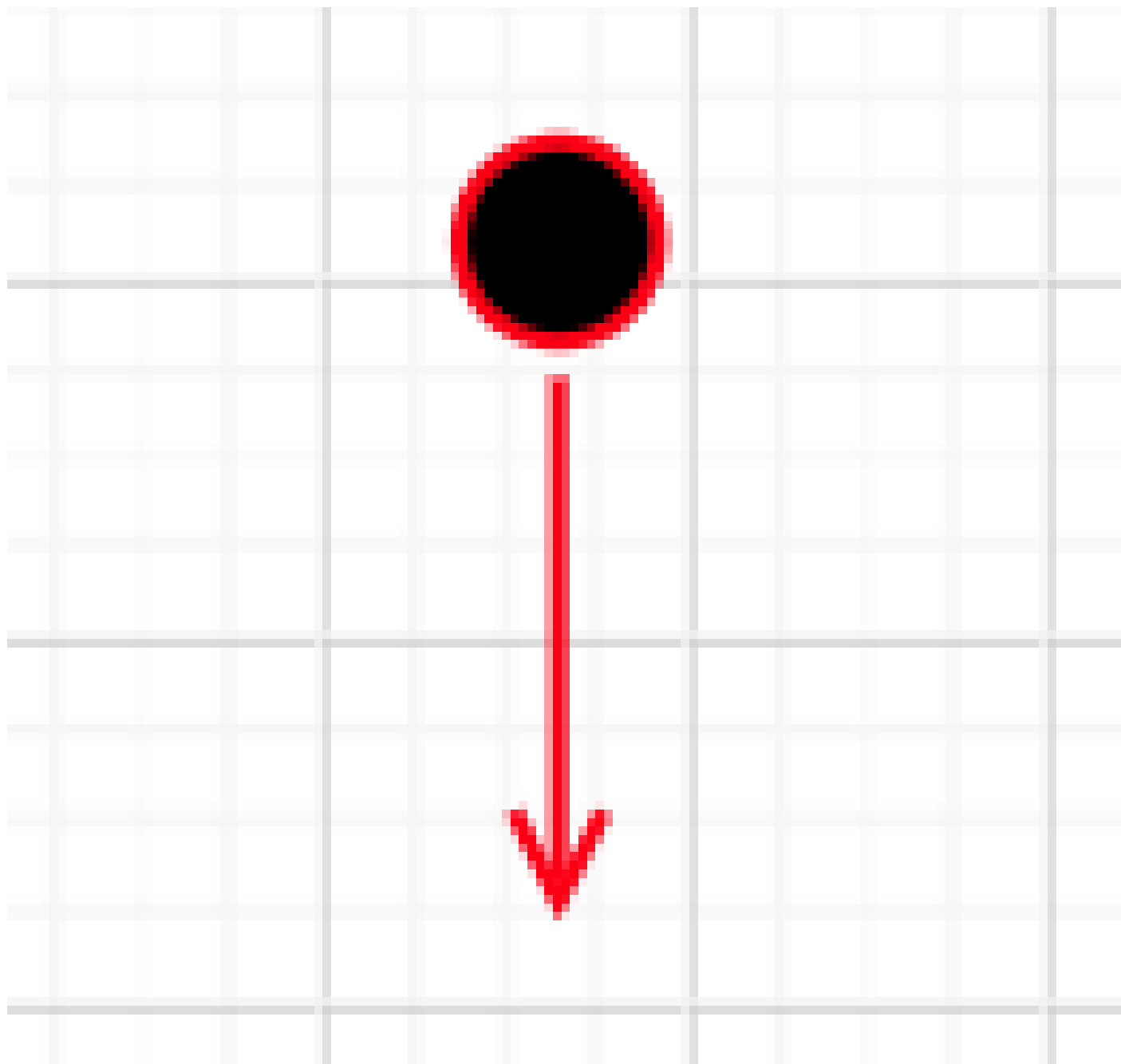
## 七、常见UML图\_活动图

### 1、活动图是什么？

活动图(activity diagram)是UML的动态规图之一，用来描述事物或对象的活动变化流程。类似流程图，描述从一个动作转移到另外一个动作，阐明了业务用例实现的工作流程。

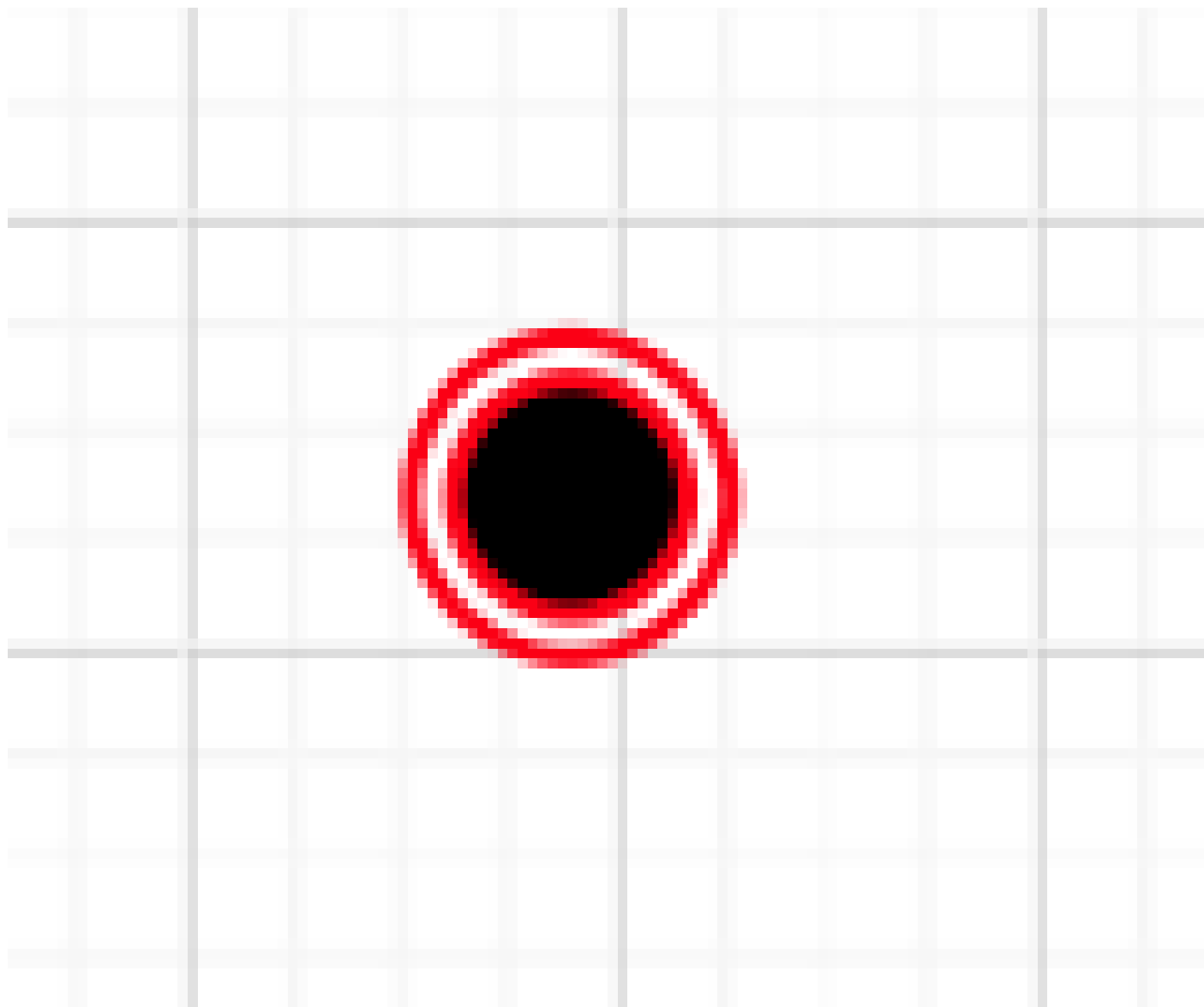
### 2、活动图的元素

(1) 开始：线条表示-活动流(ActionFlow):描述活动之间的有向关系，表示一个活动向另外一个活动之间的转移。用带箭头的实线表示。

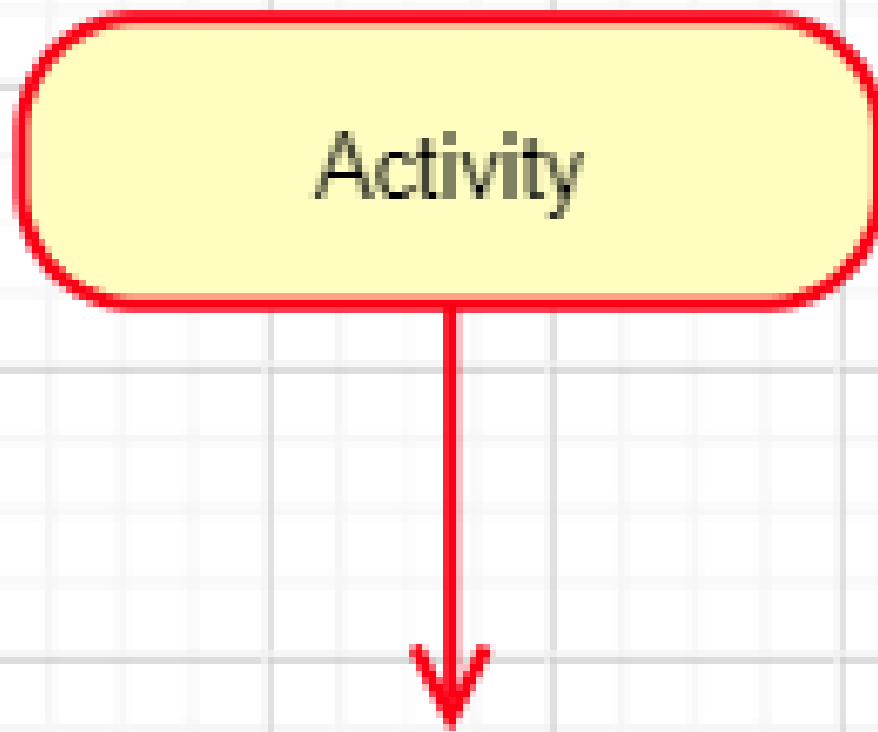


(2) 结束：

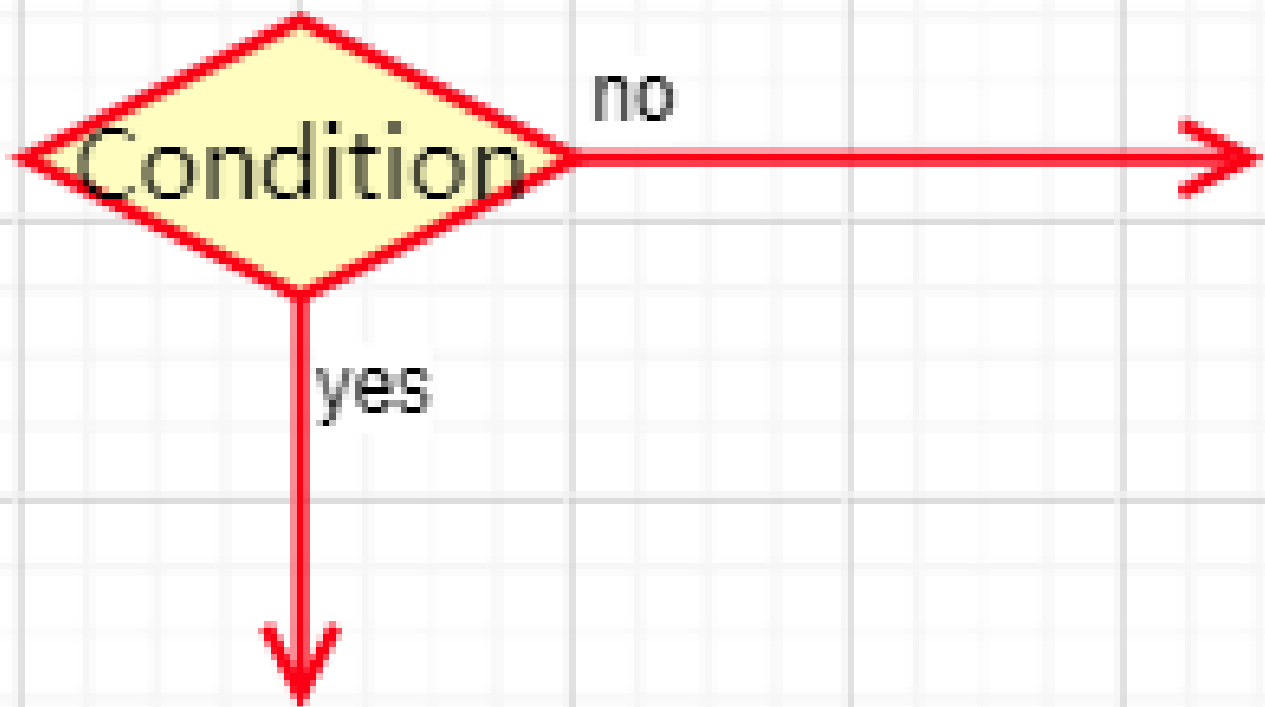




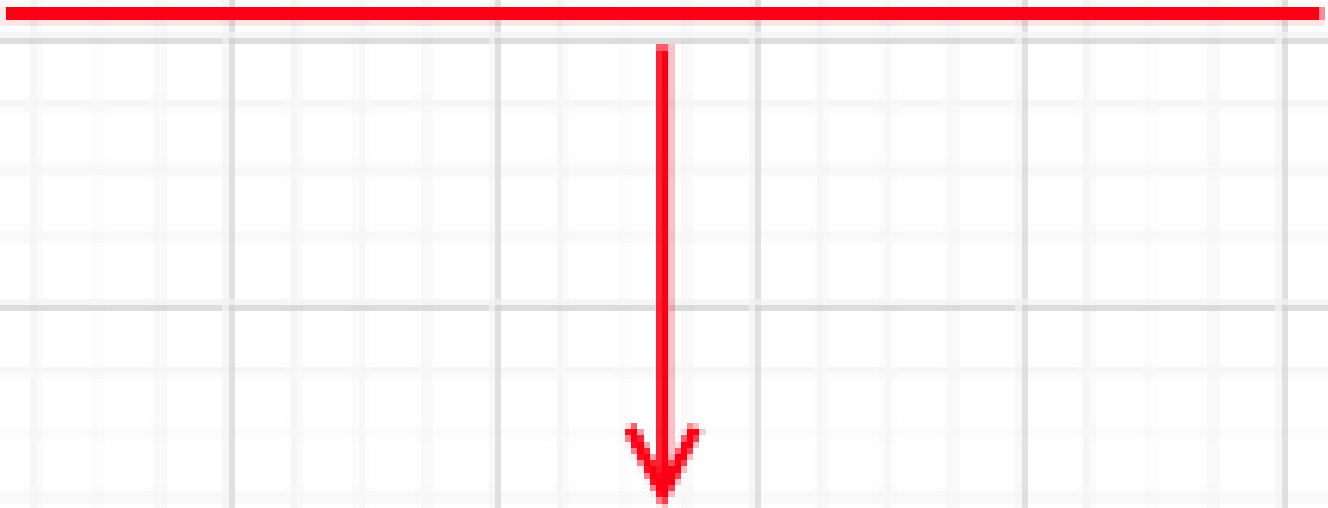
(3) 活动：



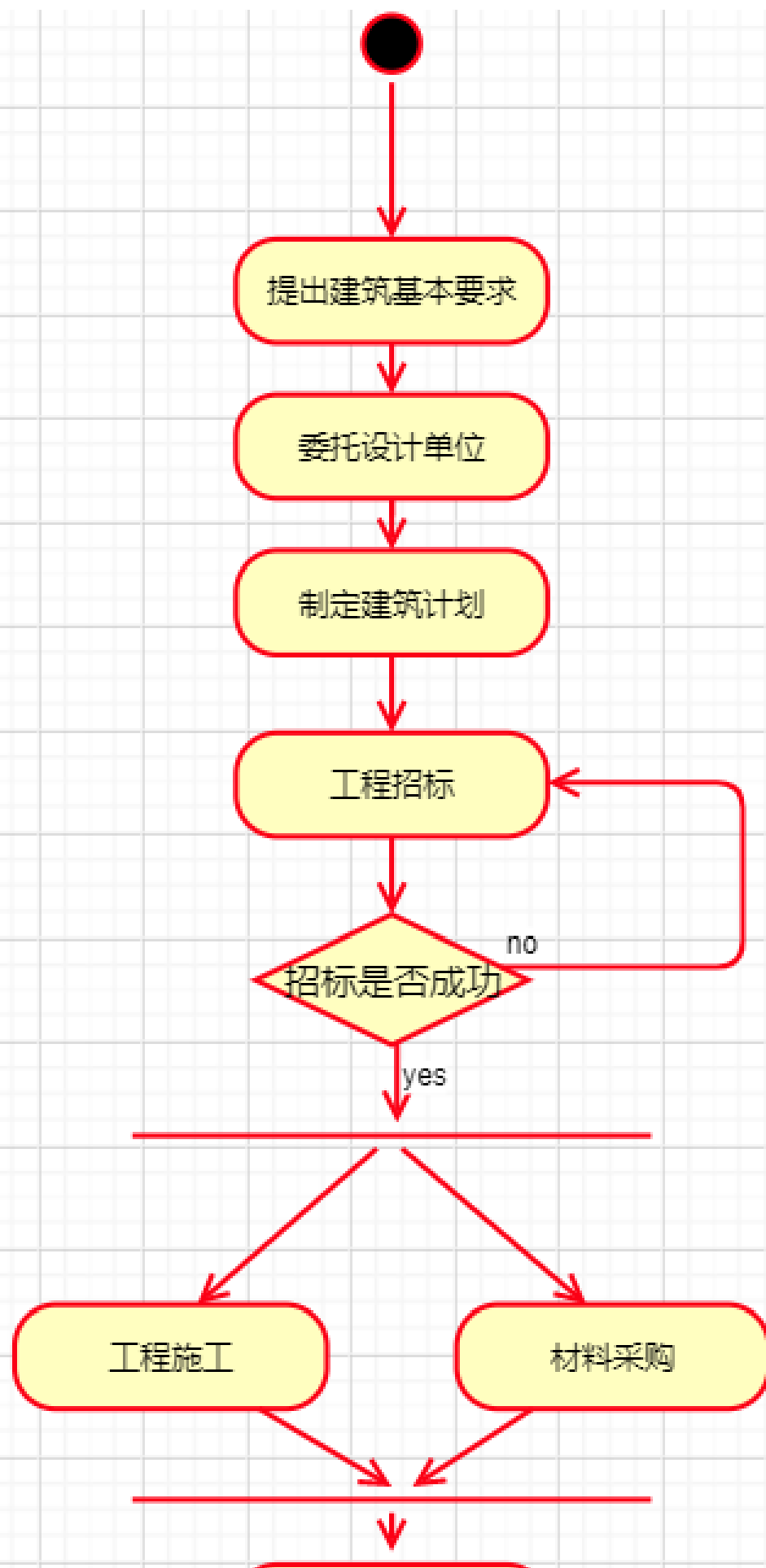
(4) 条件转移(分支)：表示从一个活动按照某种条件转移到几个不同的活动。

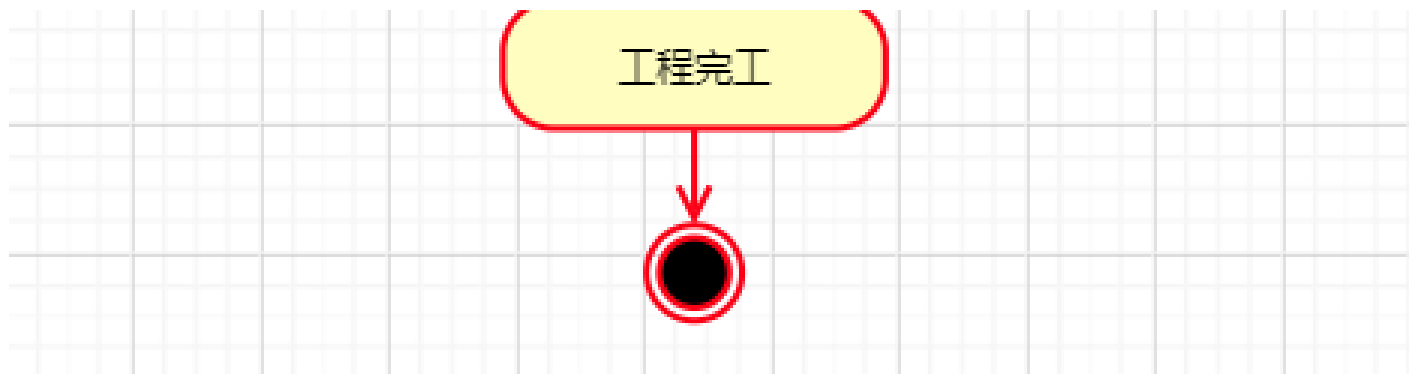


(5) 分劈和汇合：表示并发的同步行为，用同步杆表示。 - 》 有分劈、有汇总



举例：





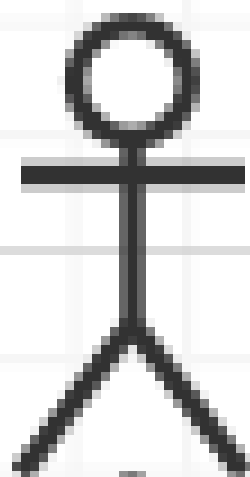
## 八、常见UML图\_时序图

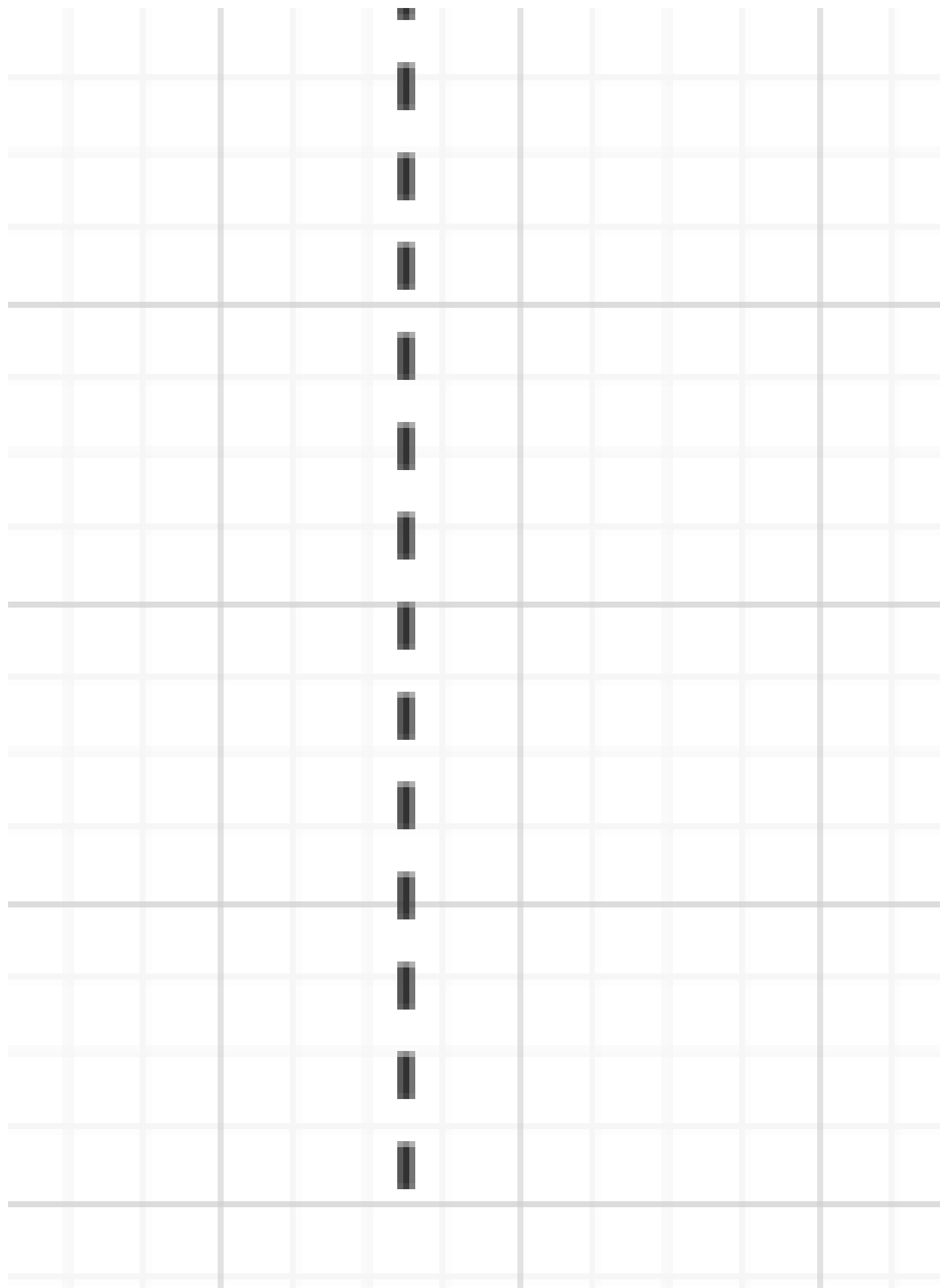
### 1、时序图是什么？

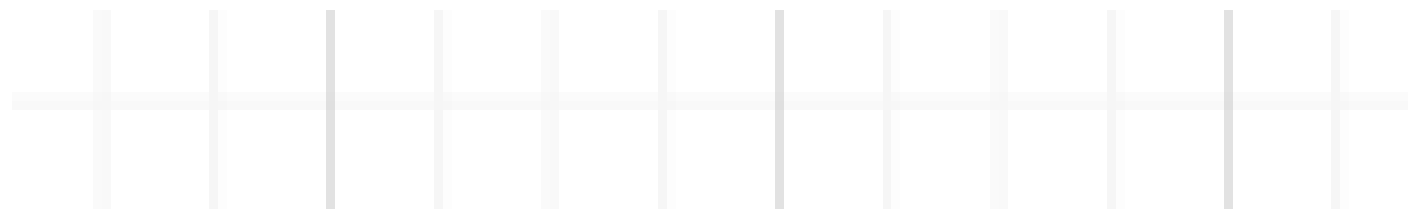
时序图（Sequence Diagram），又名序列图、循序图，是一种UML交互图，当用户进行某个操作的时候，按照时间的顺序看，各个模块之间如何调用的。描述了方法的调用过程，程序的执行流程，以及方法执行结束的返回值情况。所以用例图其中的一个用例会对应一个时序图，该时序图描述的是该功能/用例具体是怎么实现的，流程是什么。严格情况下，肯定是先设计再开发。

### 2、时序图的元素

（1）角色(Actor)



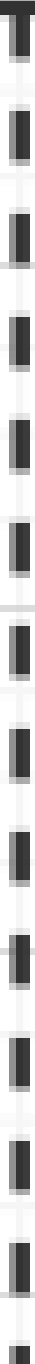


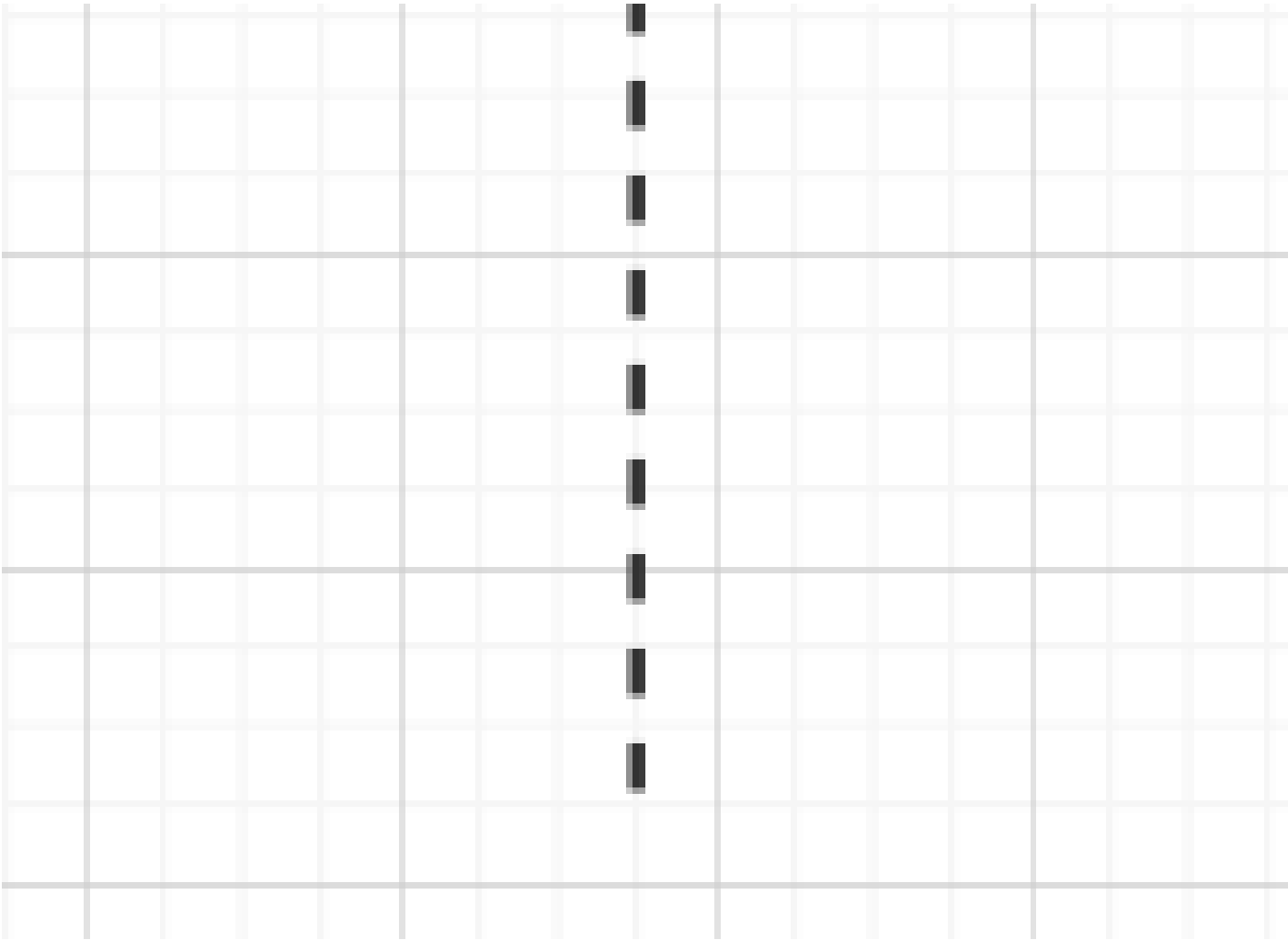


( 2 ) 对象(Object)

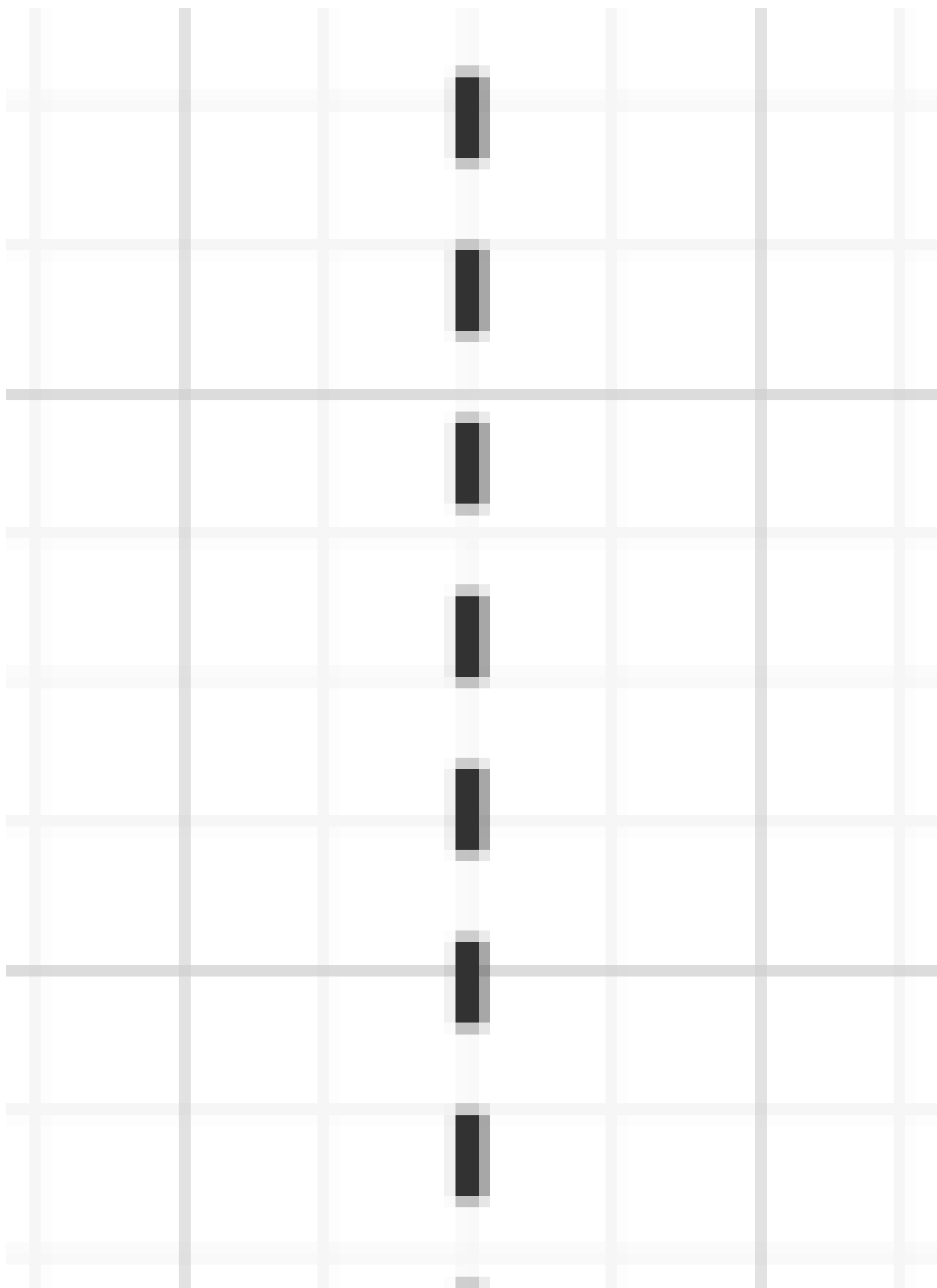


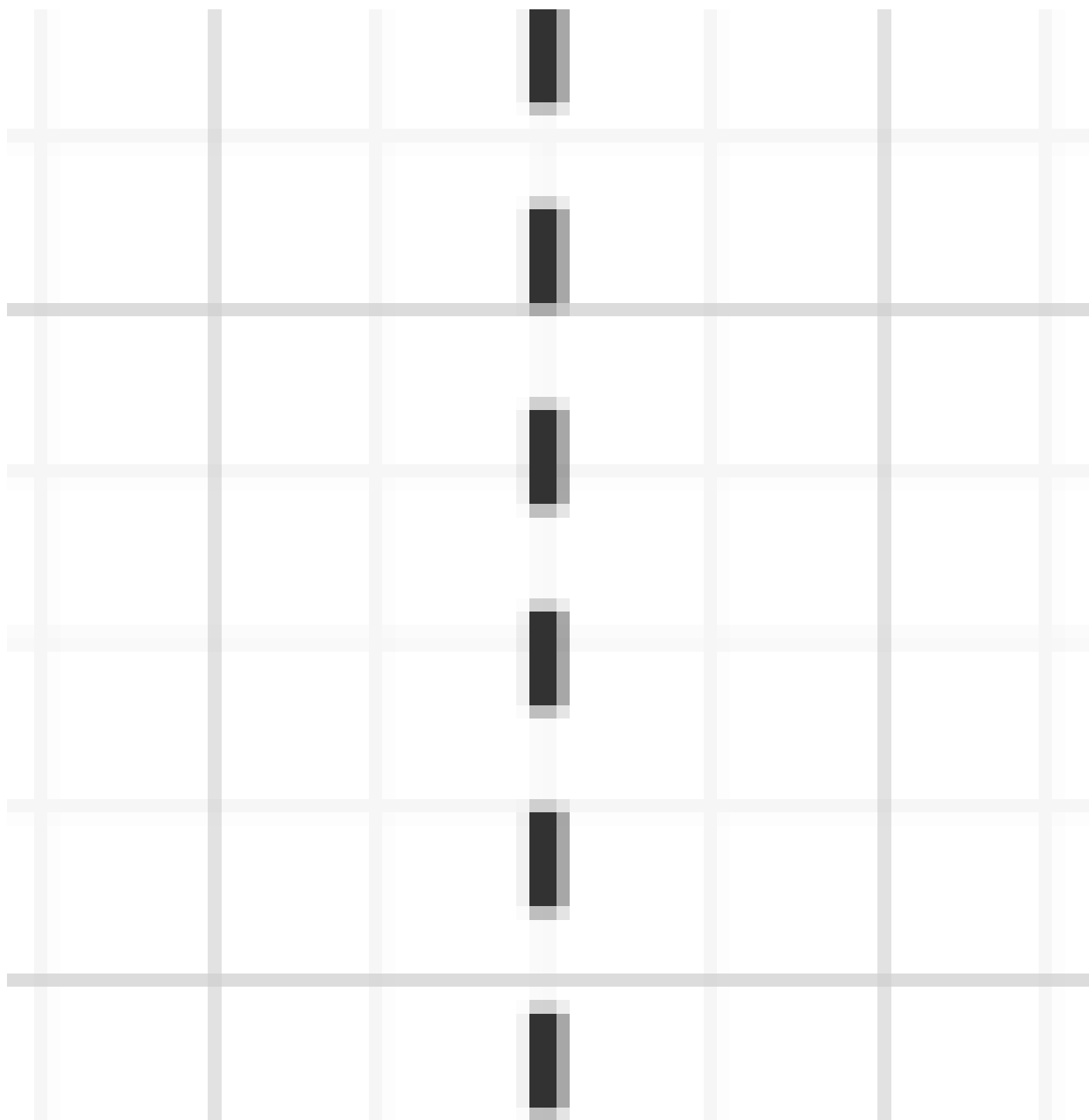
:Object



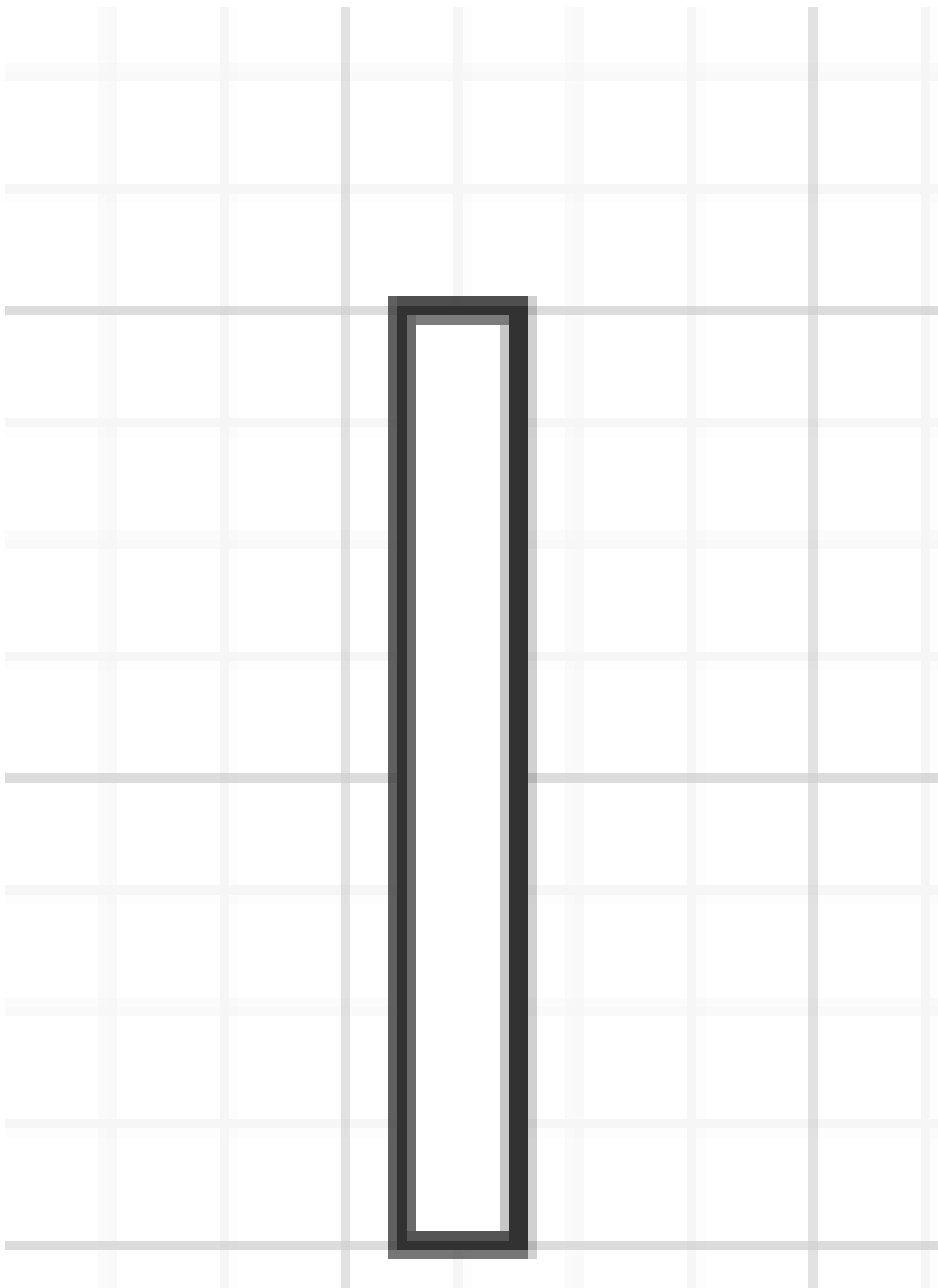


( 3 ) 生命线(LifeLine)





( 4 ) 控制焦点(Activation)

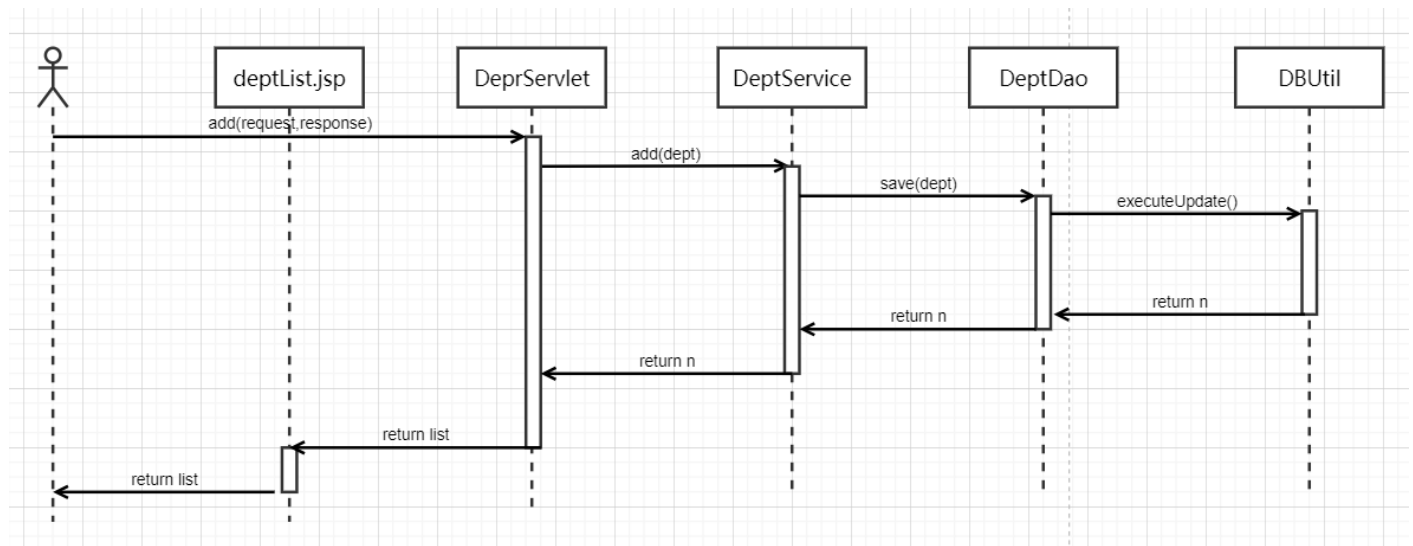


PS：控制焦点可以体现生命周期

( 5 ) 消息(Message)

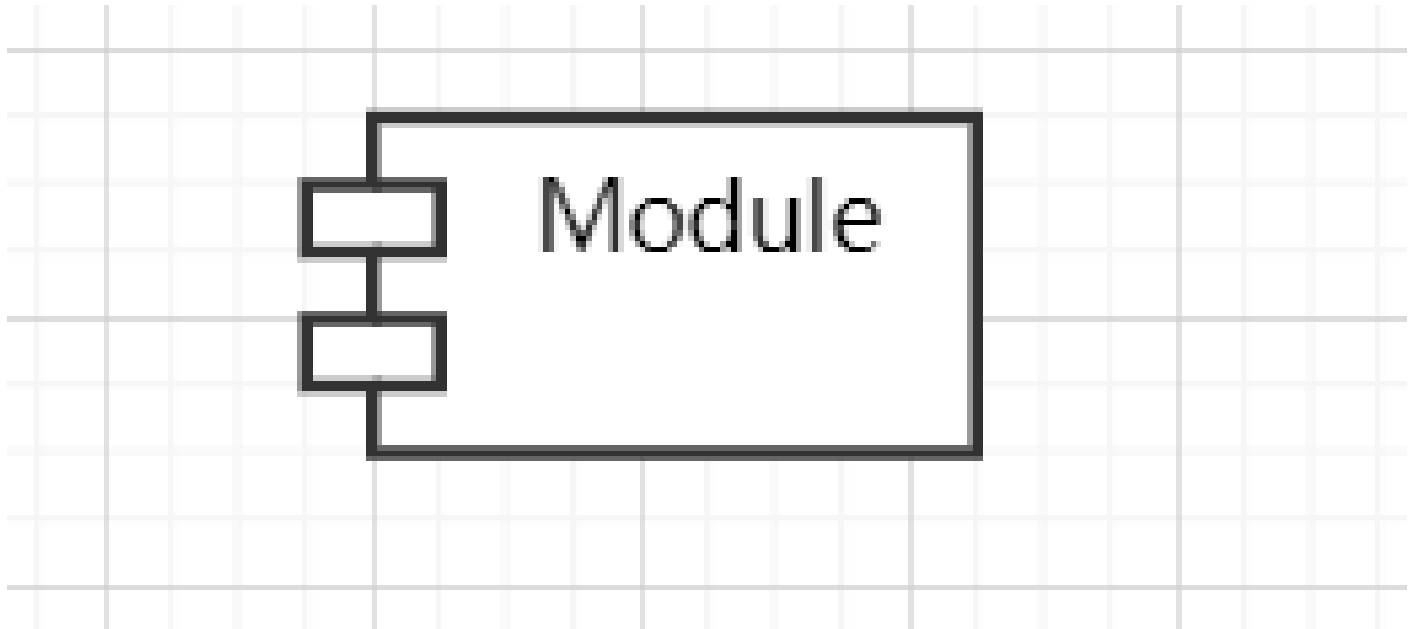
Message1 label

举例：



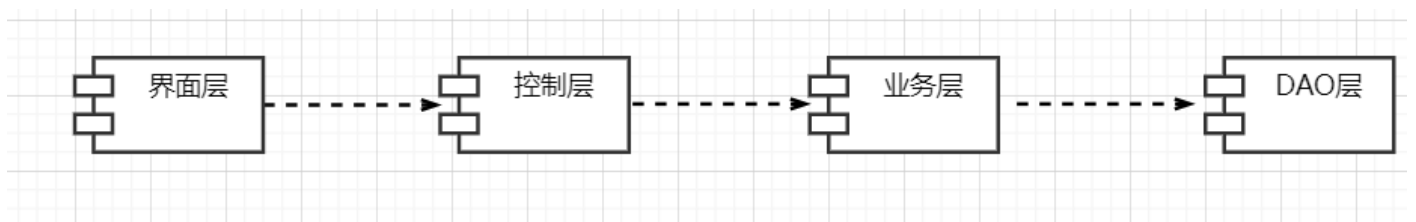
## 九、常见UML图\_组件图

组件图用来建立系统的各个组件之间的关系（网站分了多少层，每层有多少组件），它们是通过功能或者文件组织在一起，使用组件图可以帮助读者了解某个功能位于软件包的哪一位置，以及各个版本的软件包含那些功能。如javabeen、ejb、jsp都是组件。在UML中，组件元素为：



组件图可以用来帮助设计系统的整体构架。

举例：



## 十、UML图\_部署图

表现用于部署软件应用的物理设备信息

