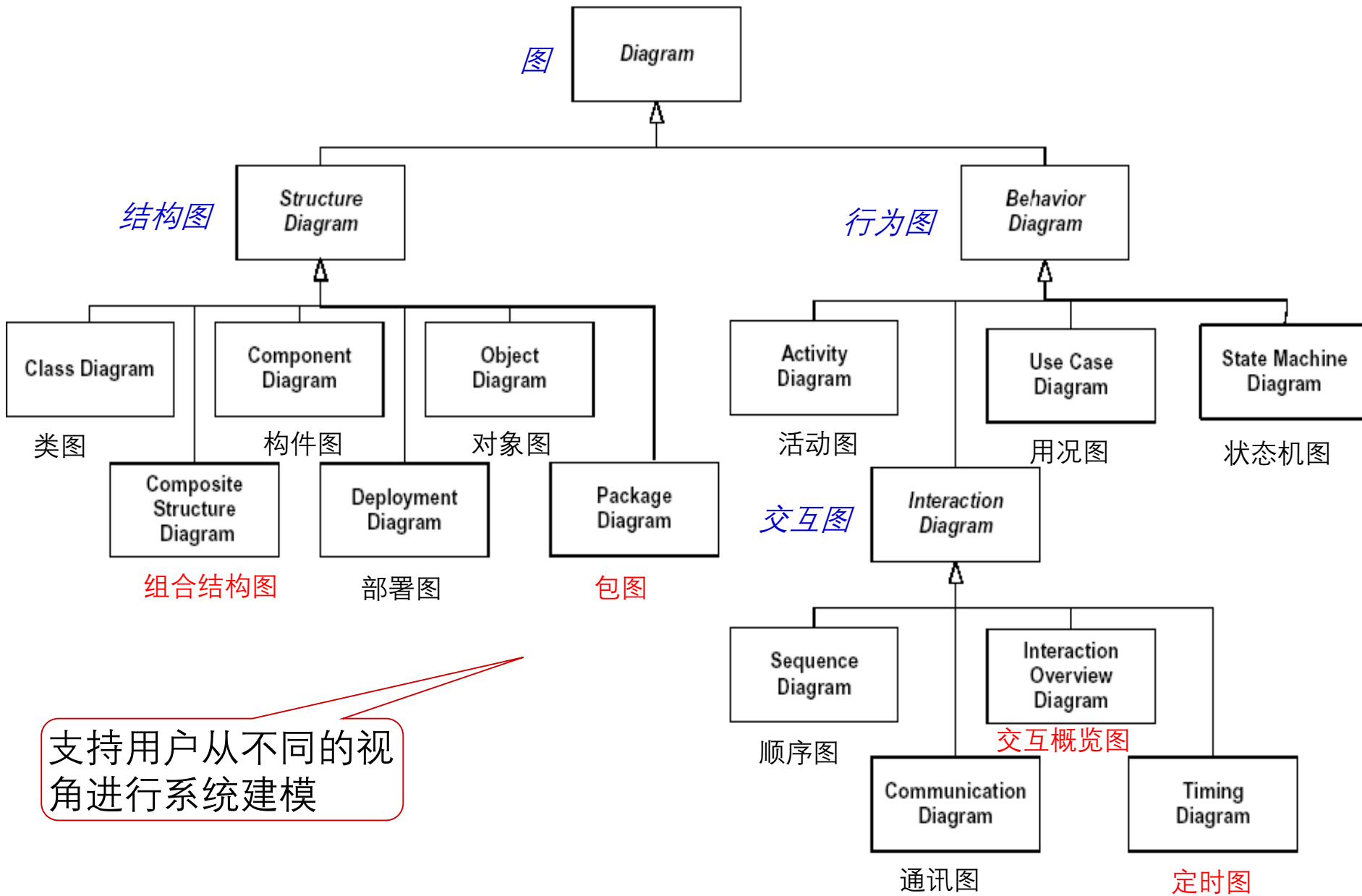


UML2中的各种图



支持用户从不同的视角进行系统建模

用况图：需求模型，是开展面向对象建模的基础，提倡尽可能使用。

类图：基本模型，是面向对象的建模最重要的模型，必不可少。

包图：辅助模型，各种模型图的组织机制，系统规模较大时使用。

顺序图：辅助模型，清晰地表示一组对象之间的交互，对类图起到补充作用。
在交互情况较复杂时使用。

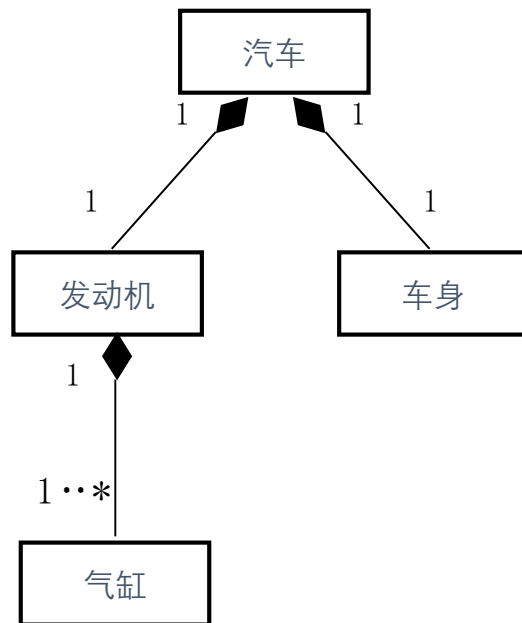
活动图：辅助模型，可描述对象的操作流程，也可描述高层的行为。

状态机图：辅助模型，对于状态与行为复杂的对象，可描述对象状态及转移，
以便更准确地定义对象的操作。

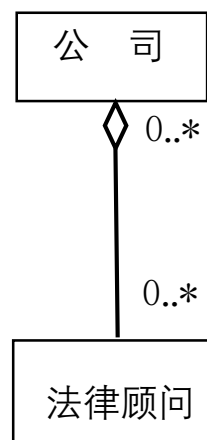
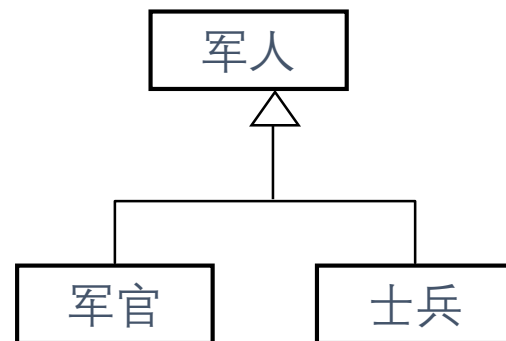
继承：一般-特殊结构。 is a kind of

聚合：has a”或“is a part of”

整体-部分结构



关联

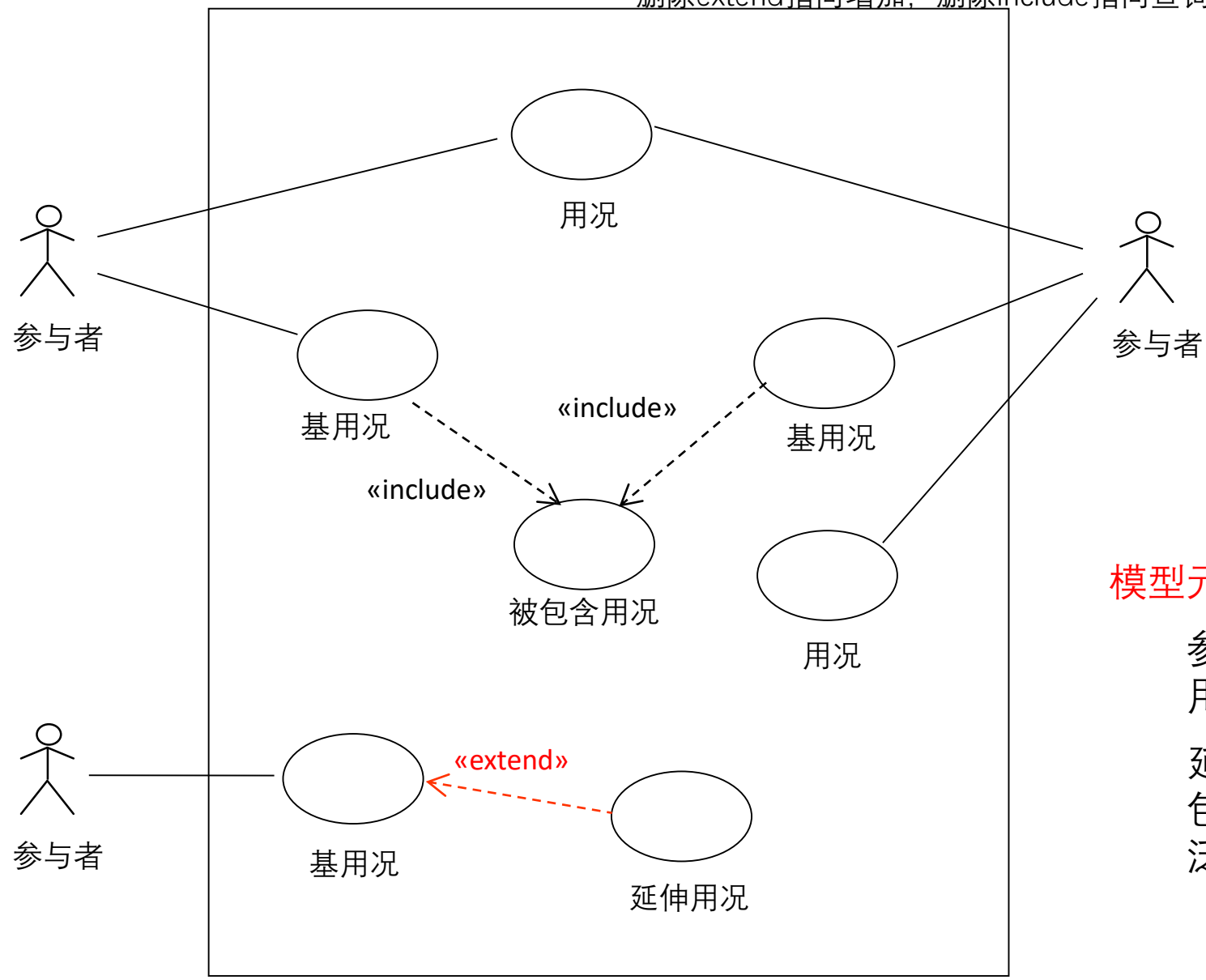


组合：拆开后没有 空心
聚合：拆开后还有

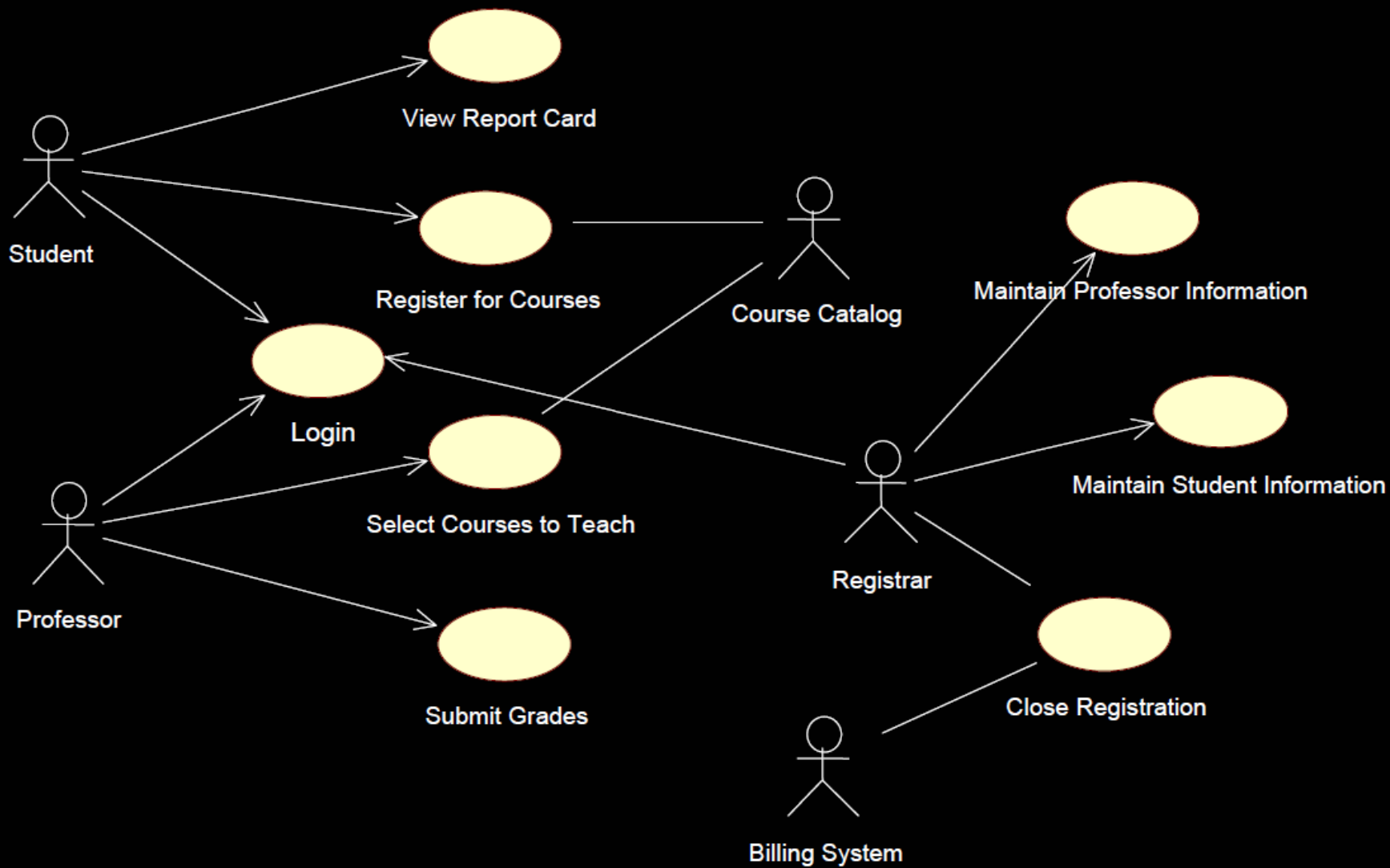
教师与被指导的学生

用况图

延伸，可能 extend指向被延伸；
包含，必须 include指向被包含，
泛化is a
删除extend指向增加，删除include指向查询



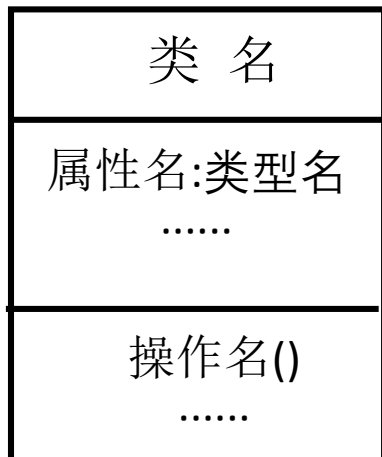
- 模型元素:
- 参与者
 - 用况
 - 延伸
 - 包含
 - 泛化



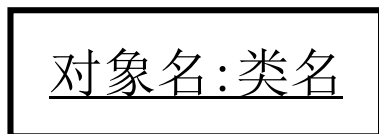
类图



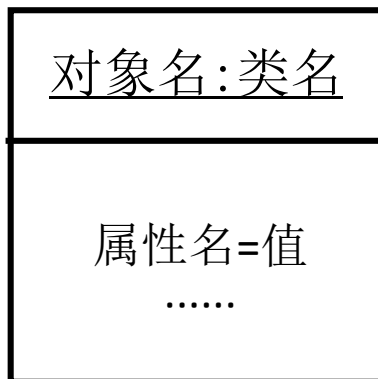
压缩方式



展开方式



压缩方式

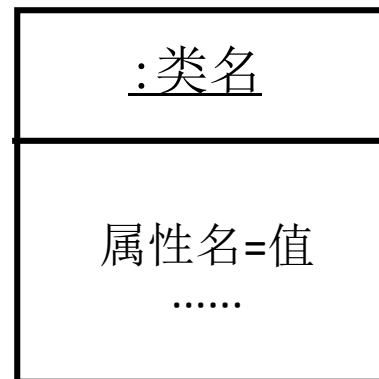


细节方式

匿名对象

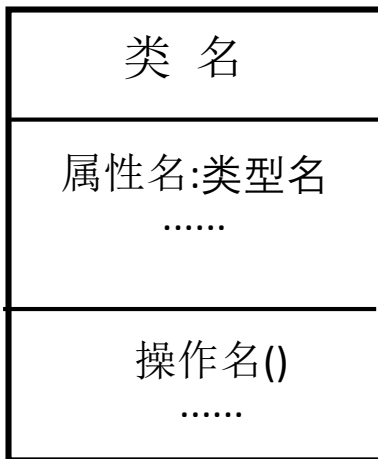


压缩方式

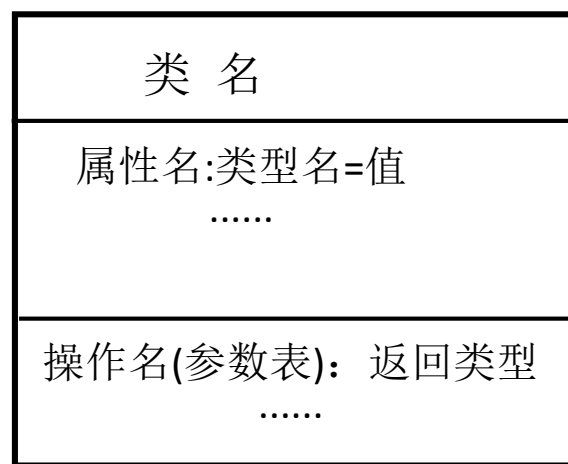


细节方式

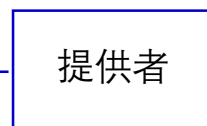
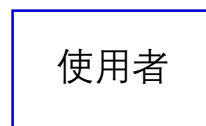
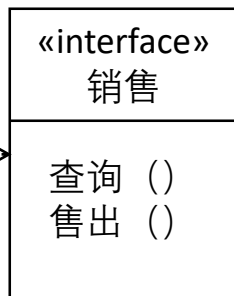
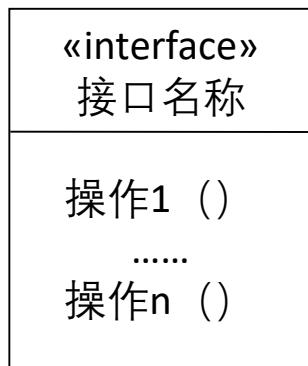
属性和操作



分析级细节方式



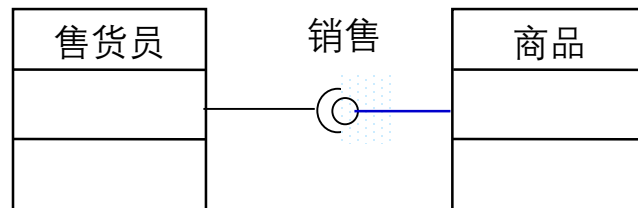
实现级细节方式



同一个接口

对实现者而言是**供接口** (provided interface)

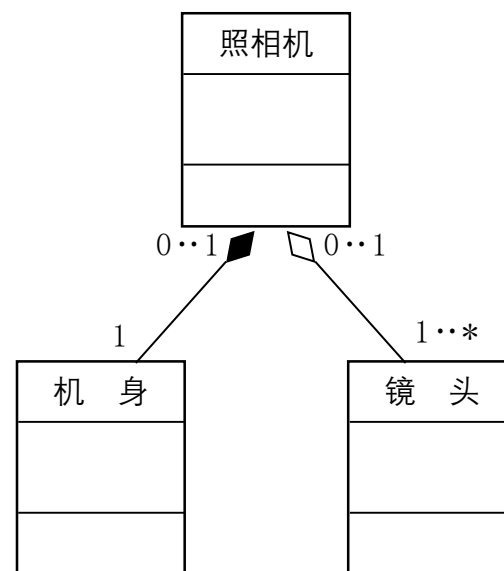
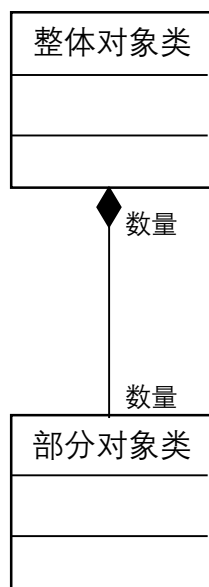
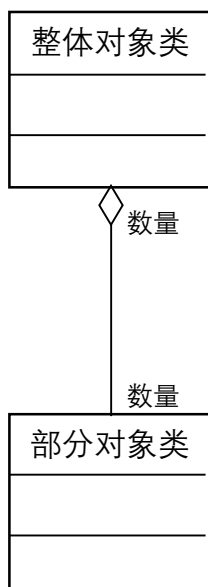
对使用者而言是**需接口** (required interface)



使用

实现

提供者的供接口 (托球)
使用者的需接口 (托座)



组合 (composition) 是聚合关系的一种特殊情况，它表明整体对于部分的强拥有关系，即整体与部分之间具有紧密、固定的组成关系。

UML把聚合定义为关联的一种特殊情况
而组合关系是聚合关系的特殊情况



数量约束

固定数值：例如 1

数值范围：例如 0..1

符号： * 表示多个

0..* = * 1..* 表示 1到多个

多重性的表示

一对一： $\frac{1}{1}$

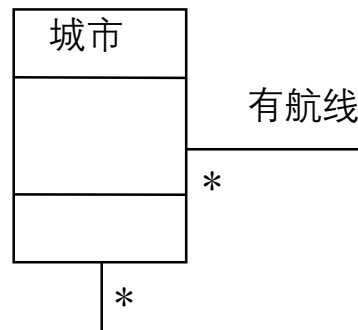
一对多： $\frac{1}{*}$

多对多： $\frac{*}{*}$

例子



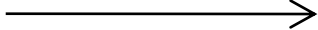
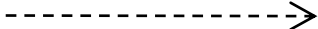
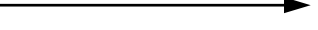
教师为学生指导论文



(d) 城市之间有航线

关联 (association) 是两个或者多个类上的一个关系（即这些类的对象实例集合的笛卡儿积的一个子集合），其中的元素提供了被开发系统的应用领域中一组有意义的信息。

UML对各种箭头的用法

箭头种类	图形符号	用 途
实线开放箭头		关联的导航性（类图） 异步消息（顺序图）
虚线开放箭头		依赖（类图、包图、用况图、构件图） 从消息接收者的操作返回（顺序图）
实线封闭箭头		同步消息（顺序图、协作图）

类非主动不用标、对象（匿名对象去掉对象名：）、属性和操作实现级，分析级不用写=值，参数表后



【泛化generalization】是一种继承关系，表示一般与特殊的关系，它指定了子类如何特化父类的所有特征和行为。例如老虎是动物的一种，即有老虎的特征也有动物的共性。箭头指向父类。



【组合composition】是整体与部分的关系，但部分不能离开整体而单独存在。如公司和部门。组合关系是关联关系的一种，是比聚合关系还要强的关系，他要求普通的聚合关系中代表整体的对象负责代表部分的对象的生命周期。代码体现：成员变量。菱形指向整体。



【关联association】是一种拥有的关系，它使一个类知道另一个类的属性和方法，如老师和学生，丈夫与妻子关联可以是双向的，也可以是单向的。双向的关联可以有两个箭头或者没有箭头，单向的关联有一个箭头。代码体现：成员变量。指向被拥有者。



【实现realization】是一种类与接口的关系，表示类是接口所有特征和行为的实现。箭头指向接口。

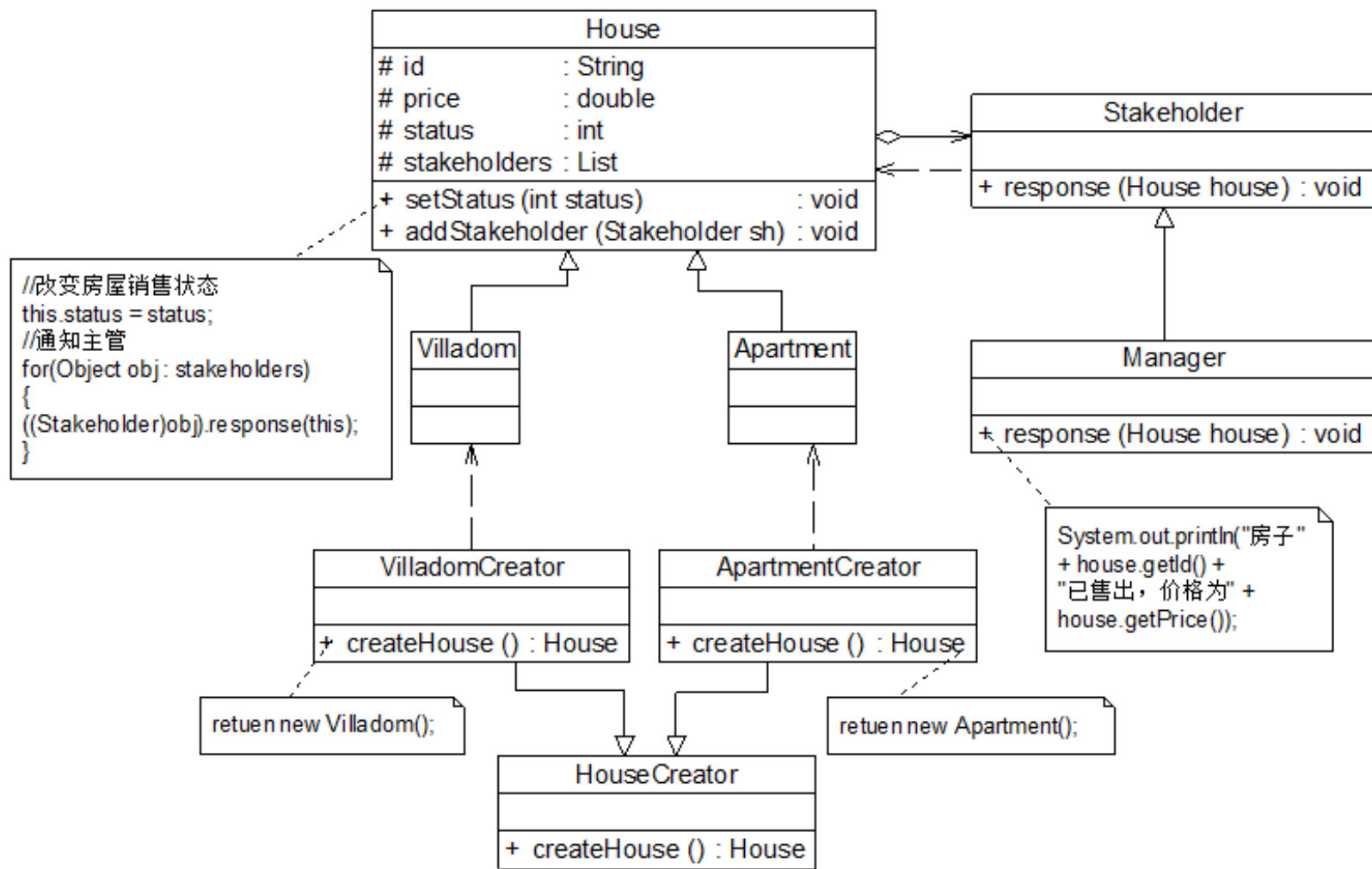


【聚合aggregation】是整体与部分的关系，且部分可以离开整体而单独存在。如车和车轮。聚合关系是关联关系的一种，是强的关联关系，关联和聚合在语法上无法区分，必须考察具体的逻辑关系。代码体现：成员变量。菱形指向整体。



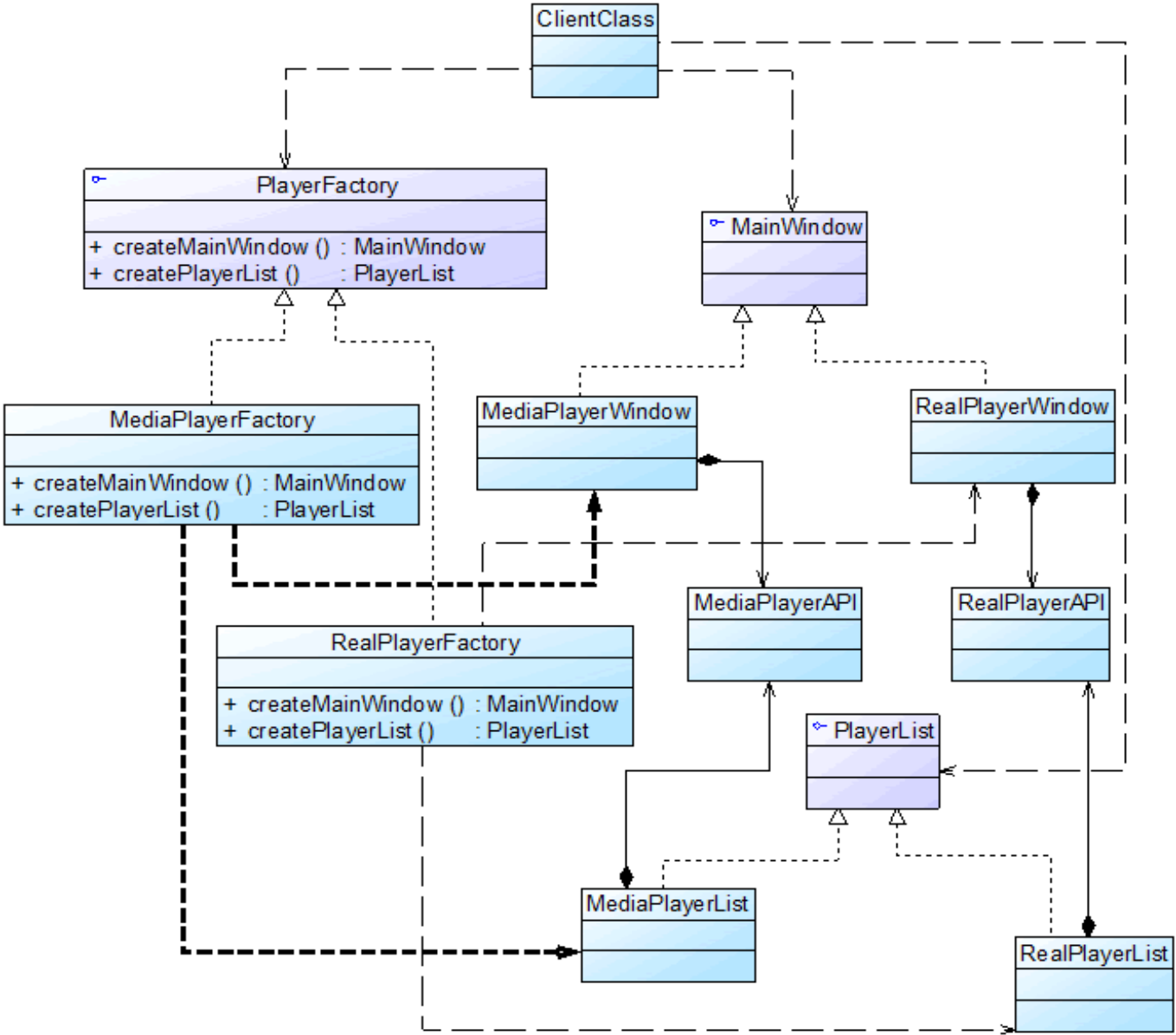
【依赖dependency】是一种使用的关系，即一个类的实现需要另一个类的协助，所以要尽量不适用双向的互相依赖。代码体现：局部变量、方法和参数或者对静态方法的调动。指向被使用者。

某房地产公司欲开发一套房产信息管理系统，根据如下描述选择合适的设计模式进行设计：(1) 该公司有多种房型，如公寓、别墅等，在将来可能会增加新的房型；(2) 销售人员每售出一套房子，主管将收到相应的销售消息。对于描述(1)可以选择使用工厂方法模式，对于描述(2)可以选择使用观察者模式，本题参考类图如下所示：



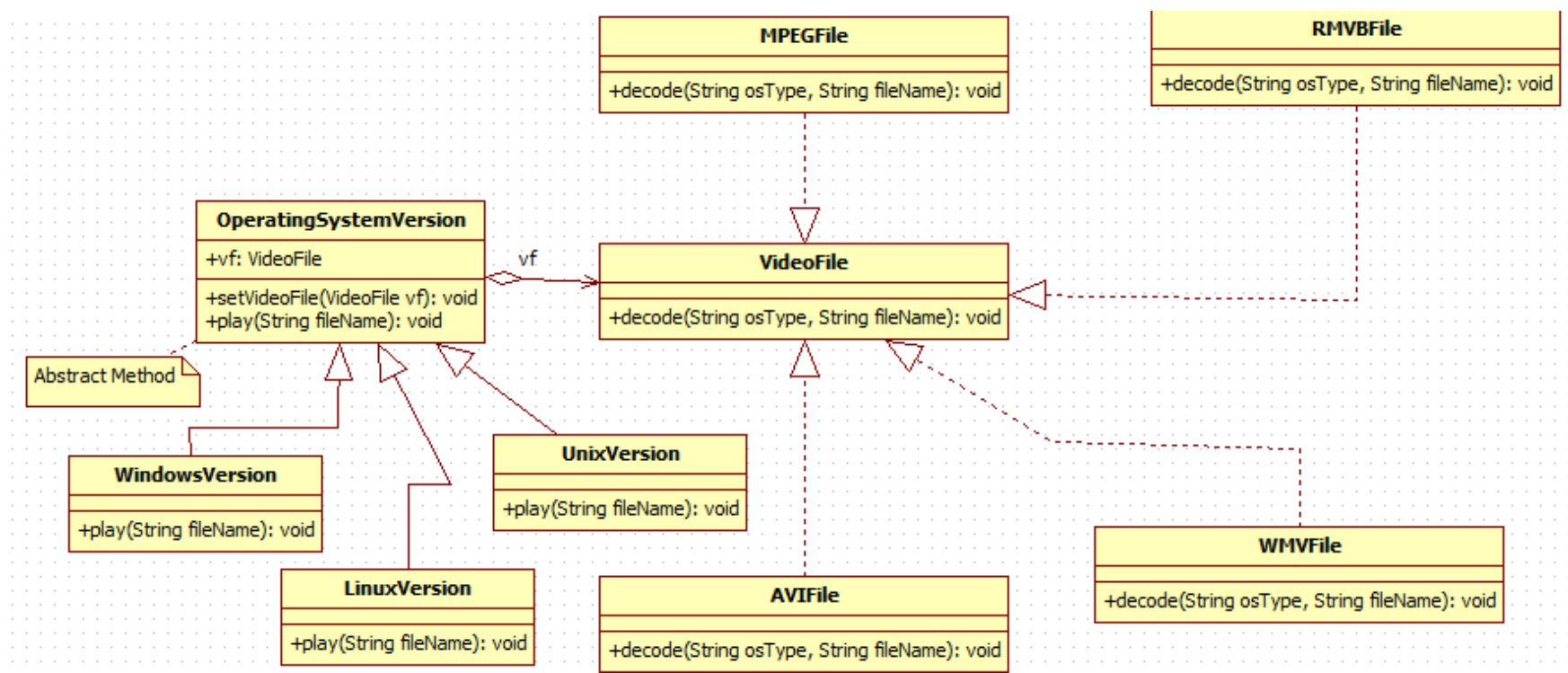
在类图中，HouseCreator是抽象房屋工厂类，其子类VilladomCreator用于创建别墅Villadom，子类ApartmentCreator用于创建公寓Apartment，Villadom和Apartment都是抽象房屋类House的子类，此时应用了工厂方法模式，如果增加新类型的房屋，只需对应增加新的房屋工厂类即可，原有代码无须做任何修改；House类同时作为抽象观察目标，子类Villadom和Apartment作为具体观察目标，相关人员类Stakeholder作为抽象观察者，其子类Manager（主管）作为具体观察者，实现了在Stakeholder中声明的response()方法，当房屋售出时，房屋的状态status将发生变化，在setStatus()方法中调用观察者的response()方法，即主管将收到相应消息，此时应用了观察者模式。

Windows Media Player和RealPlayer是两种常用的媒体播放器，它们的API结构和调用方法存在区别。现在你的应用程序需要支持这两种播放器API，而且在将来可能还需要支持新的媒体播放器，请问如何设计该应用程序？ 本题可使用适配器模式和抽象工厂模式，参考类图如下所示：

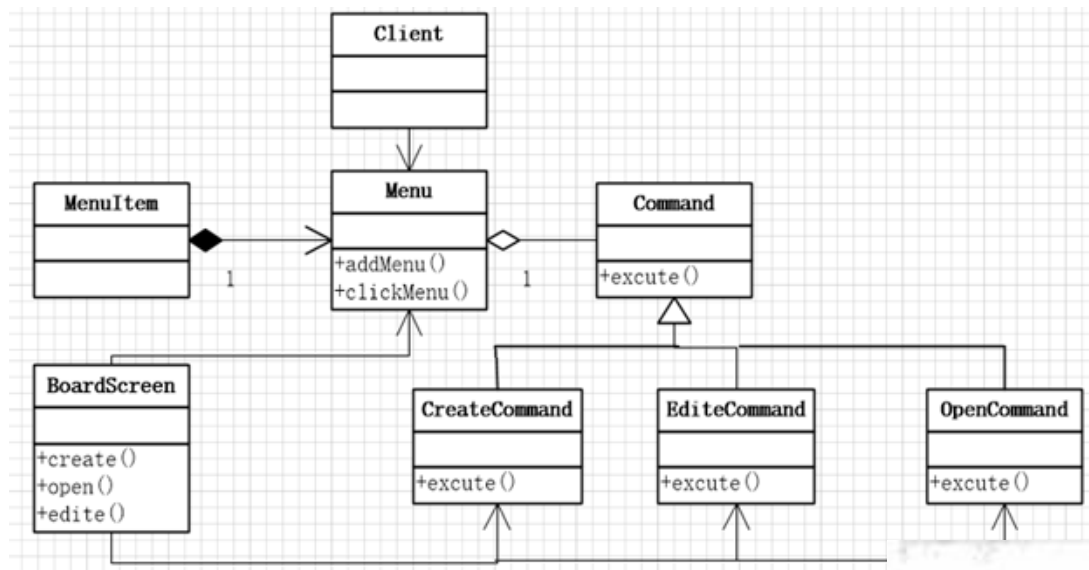


在该类图中，我们为两种不同的播放器提供了两个具体工厂类 **MediaPlayerFactory** 和 **RealPlayerFactory**，其中 **MediaPlayerFactory** 作为 Windows Media Player 播放器工厂，可以创建 Windows Media Player 的主窗口 (**MediaPlayerWindow**) 和播放列表 (**MediaPlayerList**) (为了简化类图，只列出主窗口和播放列表这两个播放器组成元素，实际情况下包含更多组成元素)；**RealPlayerFactory** 作为 RealPlayer 播放器工厂，创建 RealPlayer 的主窗口 (**RealPlayerWindow**) 和播放列表 (**RealPlayerList**)，此时可以使用抽象工厂模式，客户端针对抽象工厂 **PlayerFactory** 编程，如果增加新的播放器，只需增加一个新的具体工厂来生产新产品族中的产品即可。由于需要调用现有 API 中的方法，因此还需要使用适配器模式，在具体产品类如 **MediaPlayerWindow** 和 **MediaPlayerList** 调用 Windows Media Player API 中的方法，在 **RealPlayerWindow** 和 **RealPlayerList** 中调用 RealPlayer API 中的方法，实现对 API 中方法的适配，此时具体产品如 **MediaPlayerWindow**、**RealPlayerWindow** 等充当适配器，而已有的 API 如 **MediaPlayerAPI** 和 **RealPlayerAPI** 是需要适配的适配者。

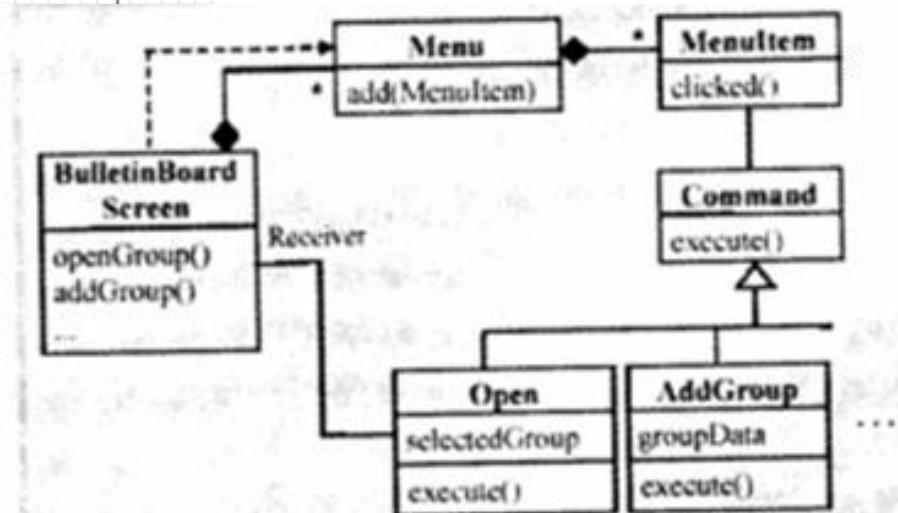
如果需要开发一个跨平台视频播放器，可以在不同操作系统平台（如 Windows、Linux、Unix等）上播放多种格式的视频文件，如MPEG、RMVB、AVI、WMV等常见视频格式。现使用桥接模式设计该播放器。类图



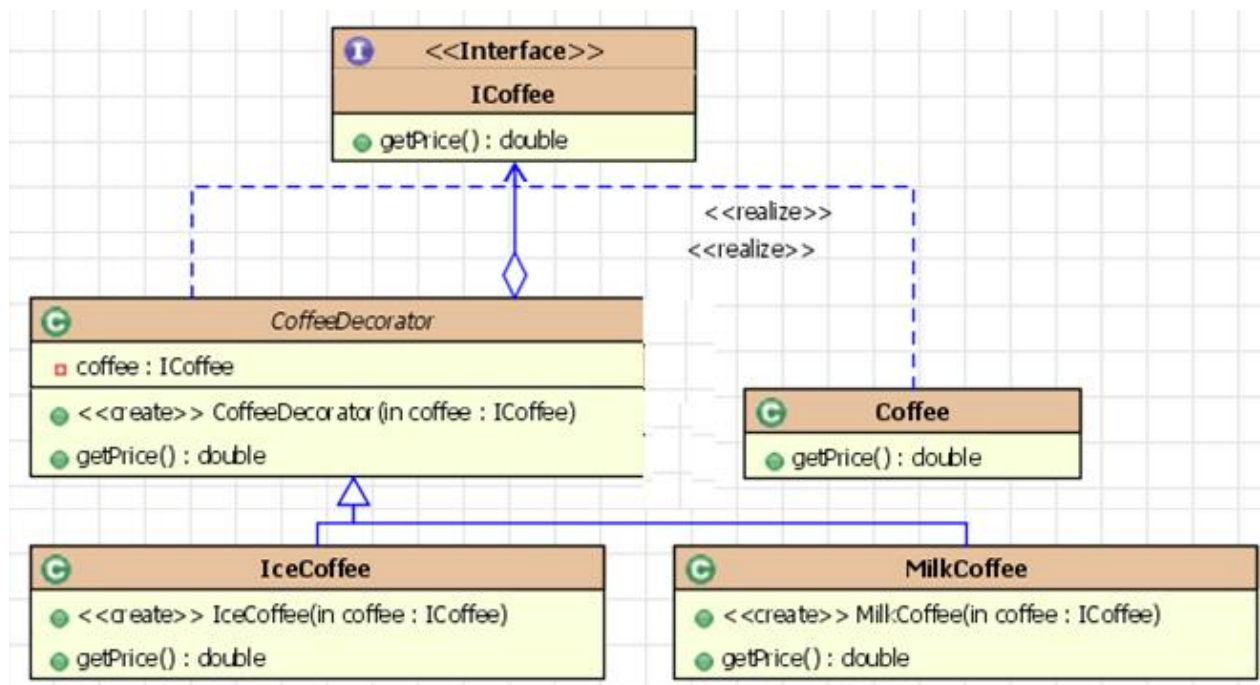
某公司欲开发一个基于Windows平台的公告板系统，系统提供主菜单（Menu）其中有菜单项（MenuItem）。通过Menu类的addMenuItem()方法增加菜单项。菜单项的打开方法是click()，每个菜单项包含一个抽象命令类，具体命令类包括OpenCommand()、CreateCommand()、EditCommand()等，命令类具有一个execute()方法，用于调用公告板系统界面类（BoardScreen()）的open()、create()、edit()等方法。使用Command模式来设计。类图：

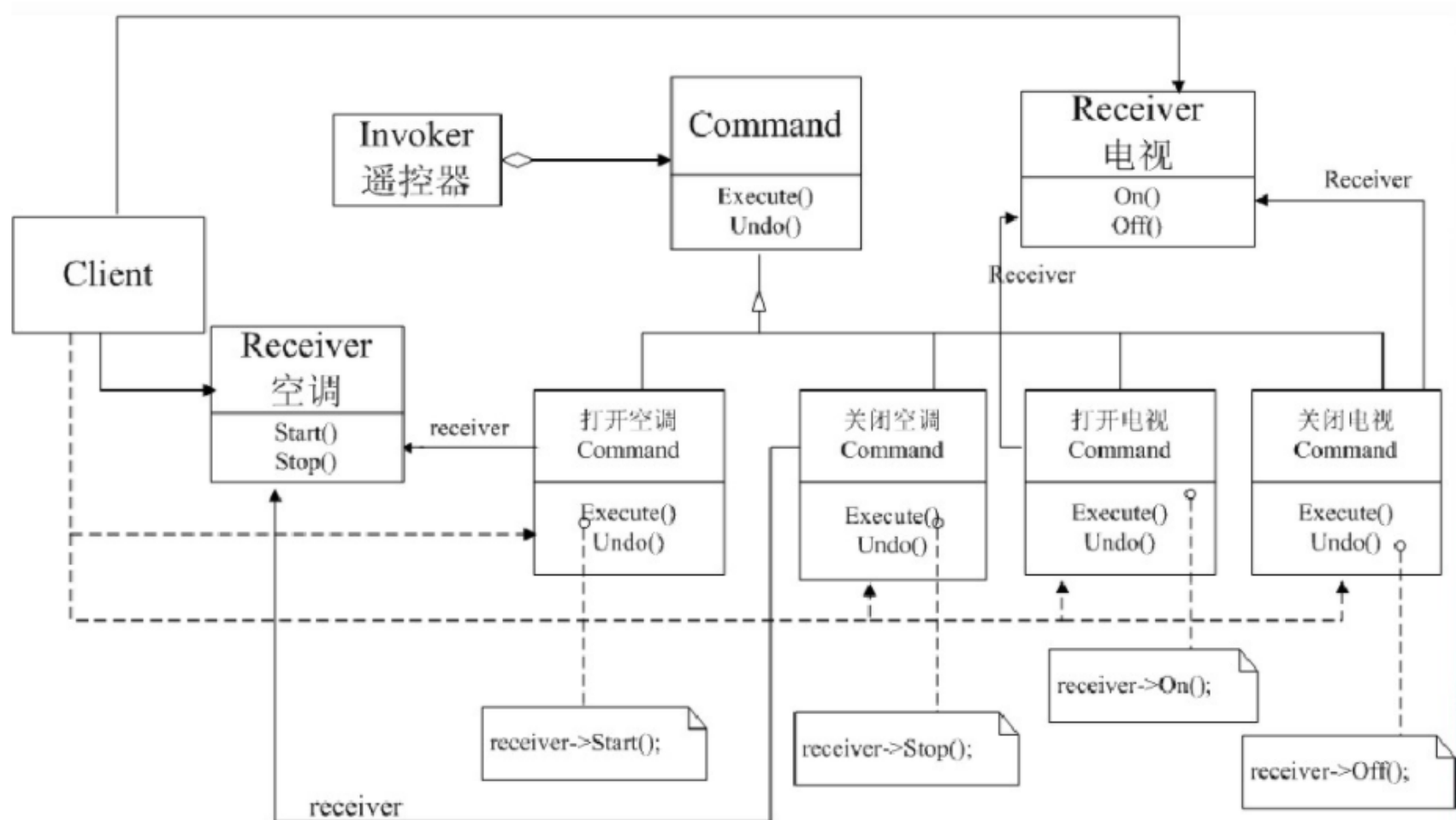


图中与Command模式中的“Invoker”角色相对应的类是（MenuItem），与“ConcreteCommand”角色相对应的类是（Open）。

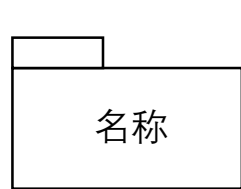


咖啡店为了进一步满足顾客的口味，允许顾客可以在普通咖啡中自由地加入多种配料（包括加糖、加奶、加冰等）。咖啡店规定：普通咖啡5元一杯，加糖多收1元，加奶多收2元，加冰多收1元。使用装饰模式设计的类图如下所示：





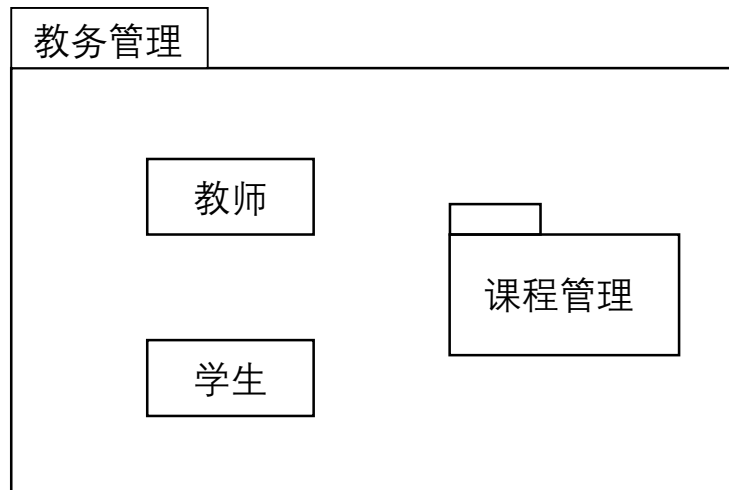
包图 (package diagram)



包的压缩方式

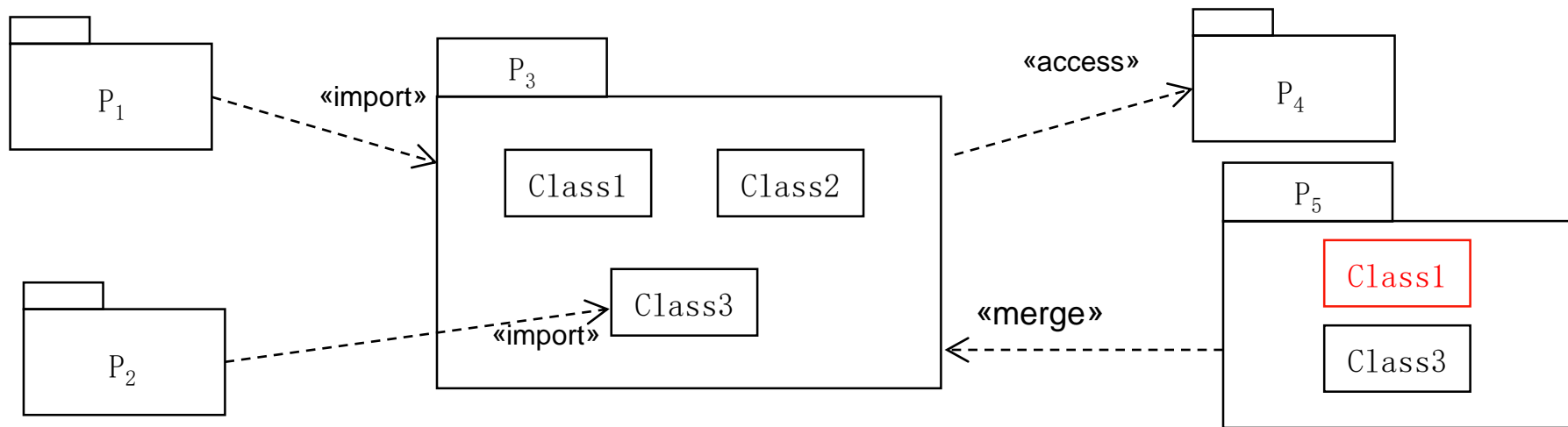


包的展开方式



一个包的例子

引入 (import) 是包之间的一种依赖关系，表明源包中的模型元素能够**直接**引用目标包中的模型元素。



访问 (access) 关系与引入关系类似

合并 (merge) 是包之间的一个有向关系，表明目标包的概念定义被合并到源包中。

顺序图 (sequence diagram)

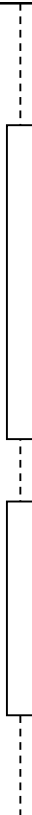
对象
生命线
实体

:类名

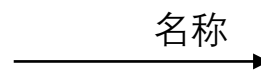


操作
(控制焦点)
执行规约

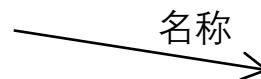
:类名



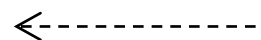
消息
(激发)



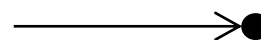
同步消息



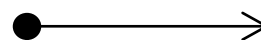
异步消息



返回消息



丢失消息



发现消息

参与者

对象 (生命线)

对象创建

用况

条件

同步消息

消息返回

操作

自调用

操作重叠

异步消息

t=now

{t..t+2sec.}

时间约束

状态不变式

生命线终止



:class A

:class B

:class C

:class D

:class E

[x>0]

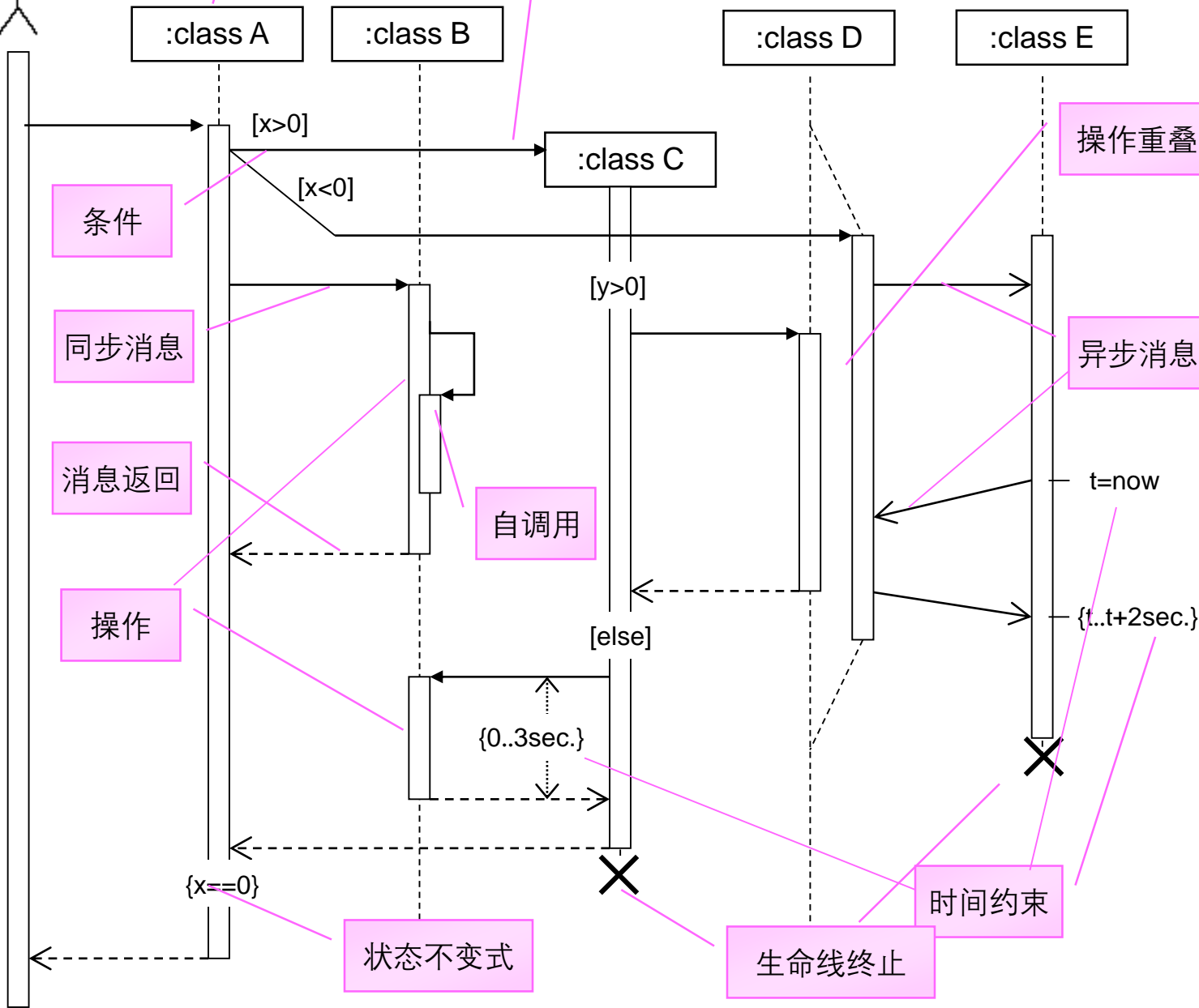
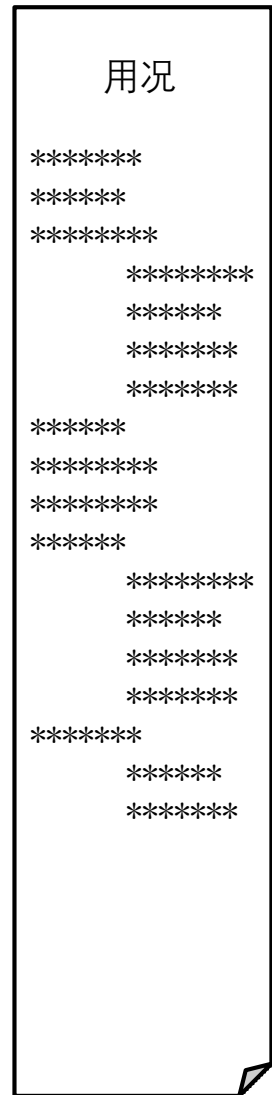
[x<0]

[y>0]

[else]

{x==0}

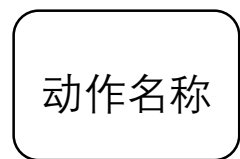
{0..3sec.}



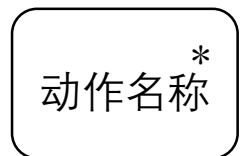
活动图 (activity diagram)

活动图由结点 (node) 和边 (edge) 两种基本元素构成

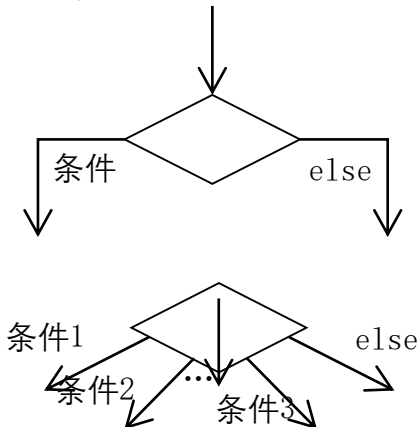
活动结点——动作、判断、合并、分岔、汇合、起点、结束
活动边——控制流和对象流



一般动作



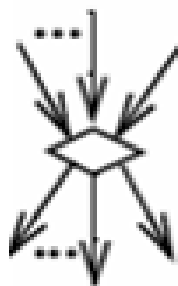
重复动作



判断



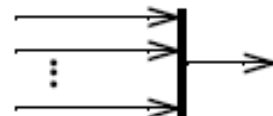
合并



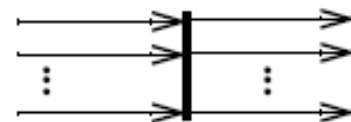
合并与判断结合



分岔



汇合



汇合与分岔结合使用



起点



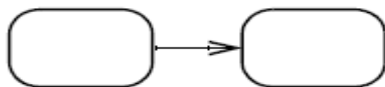
活动结束



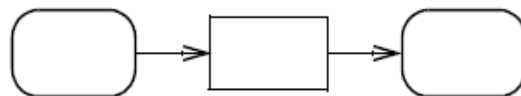
流结束



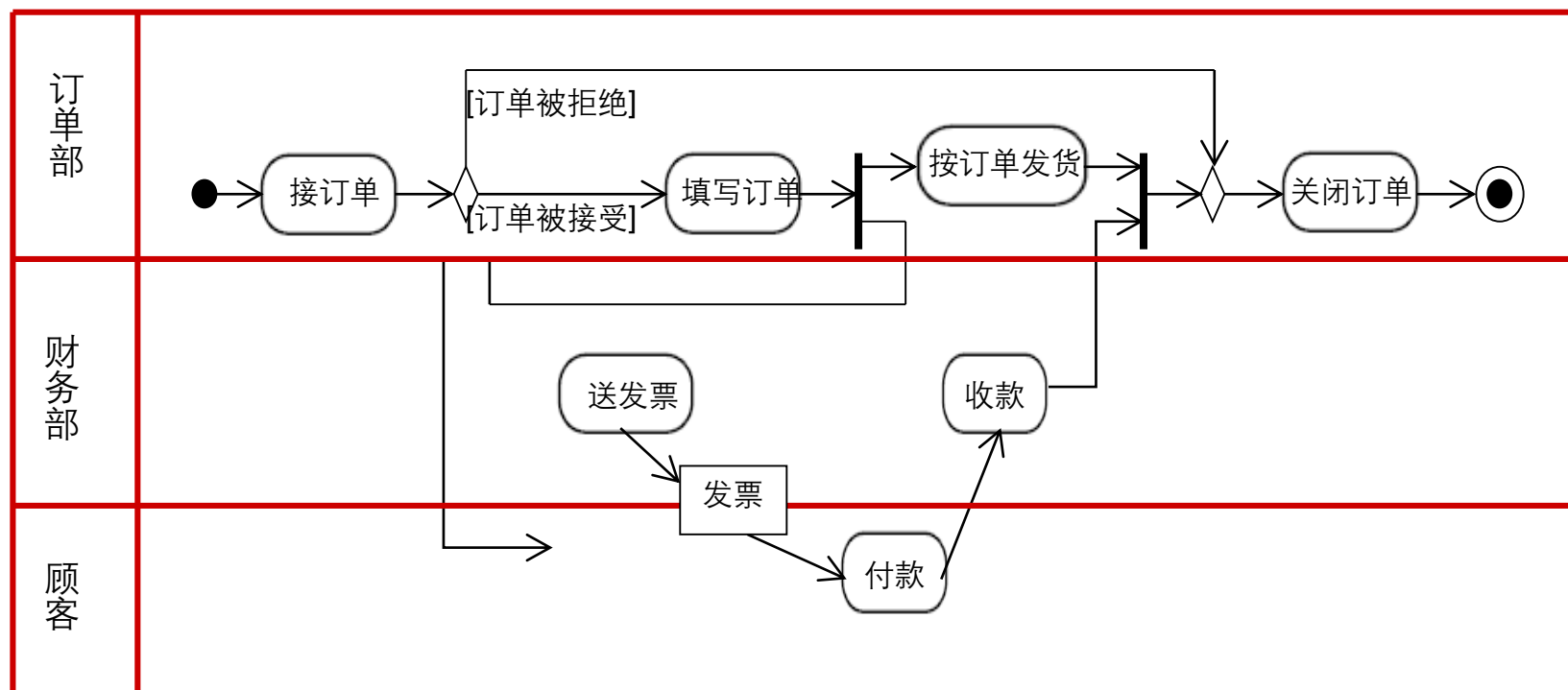
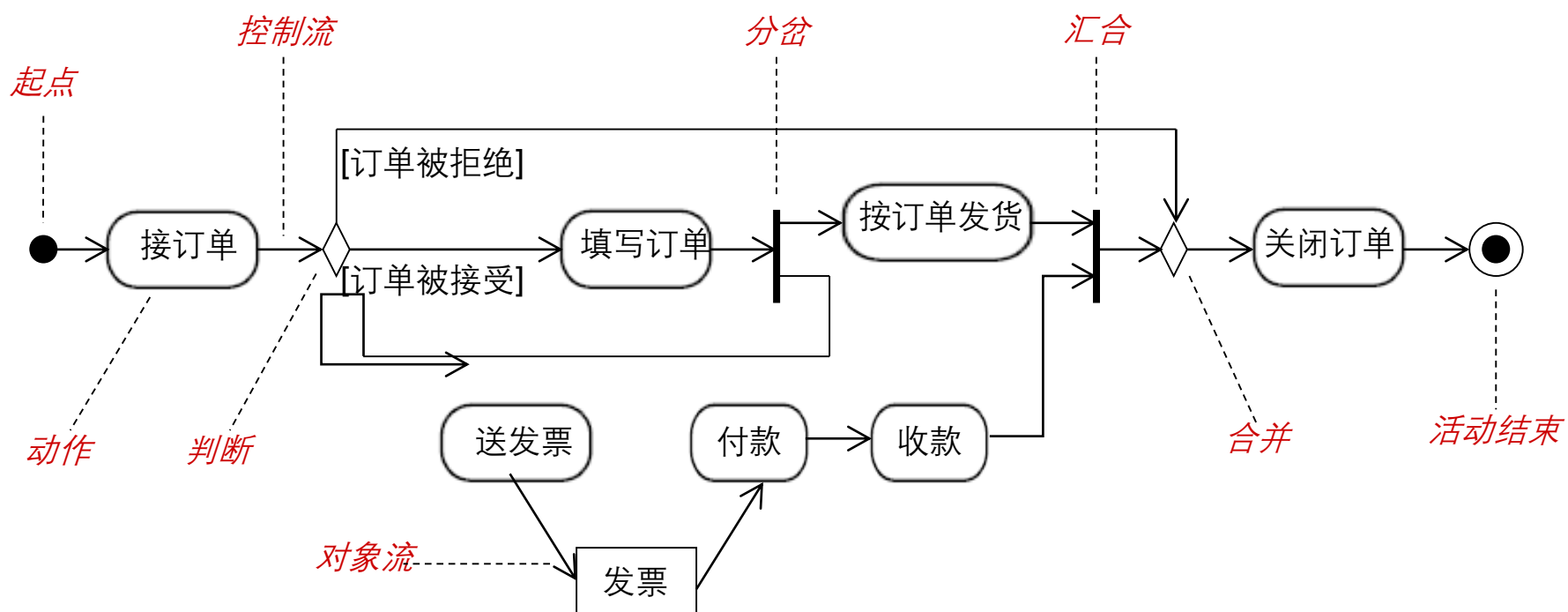
控制流



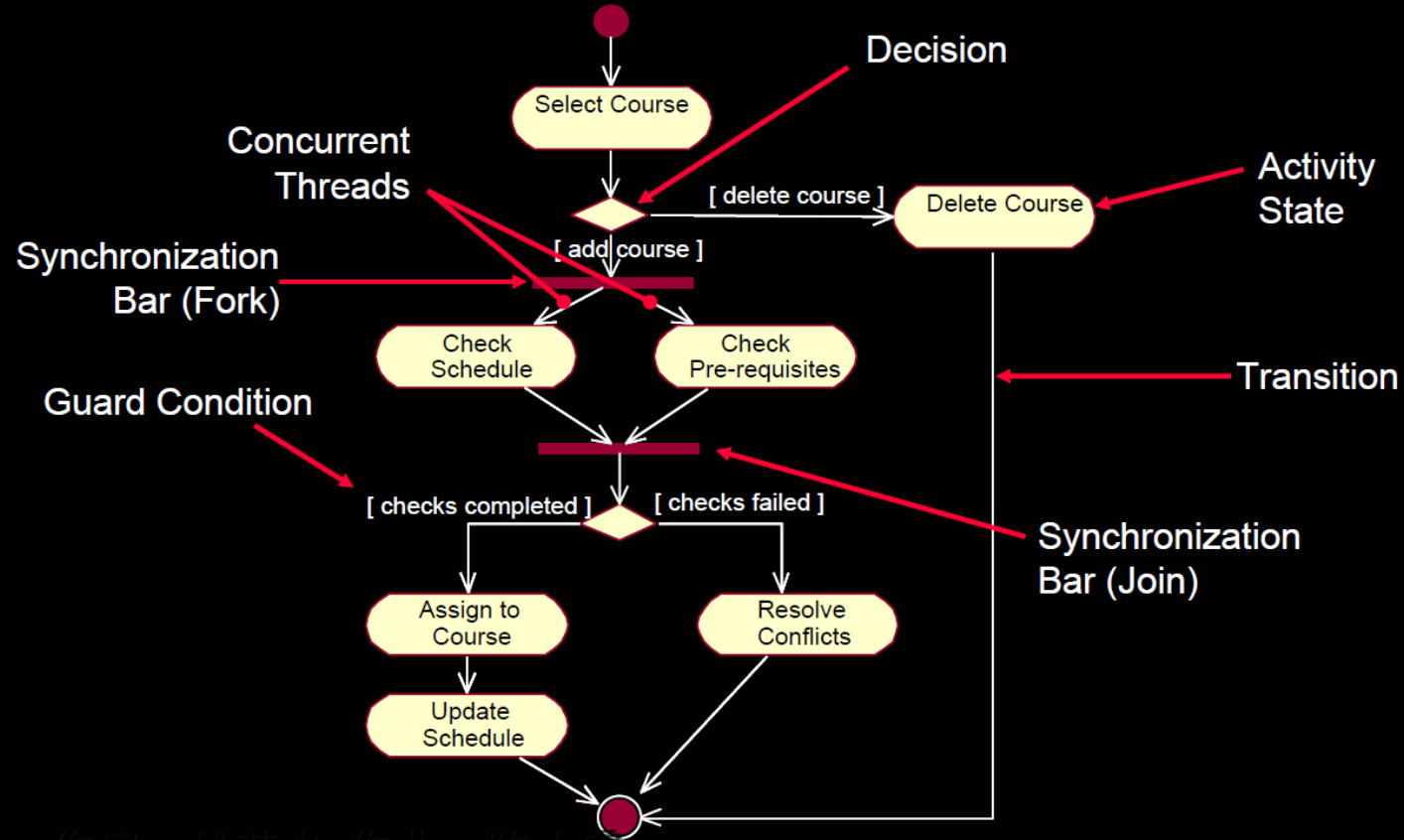
控制流及其连接的结点



对象流

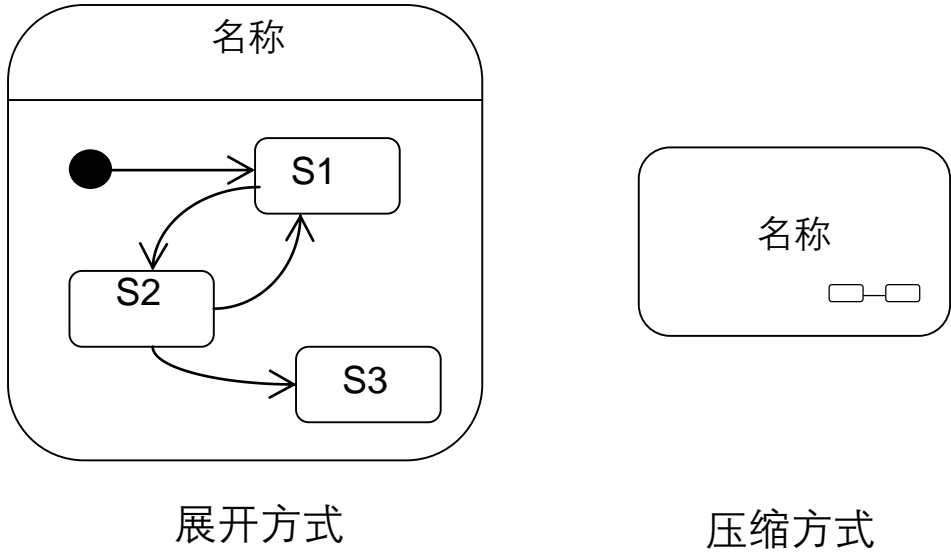
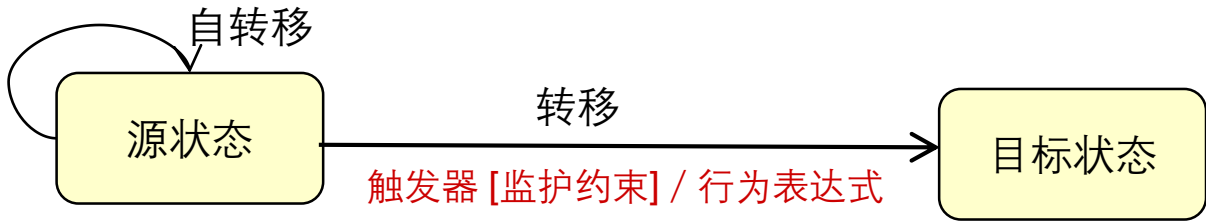
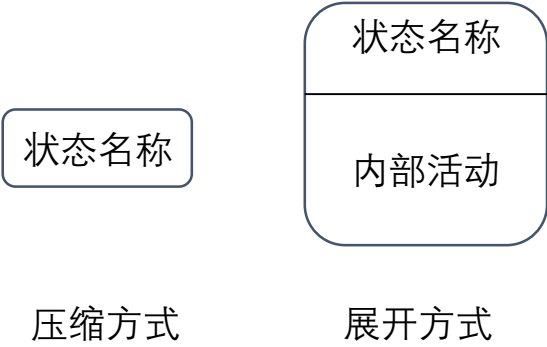


Example: Activity Diagram



作词：姚若龙 作曲：陈小霞

状态机图 (state machine diagram)



●
初始状态



进入点



浅历史

◎
最终状态

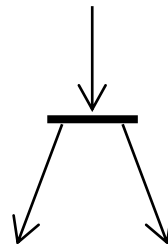


×
终止结点

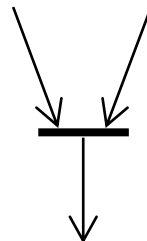
退出点



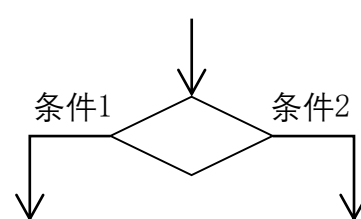
深历史



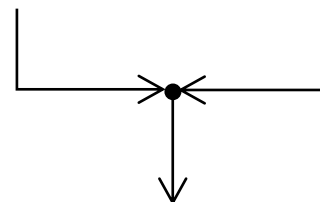
分岔



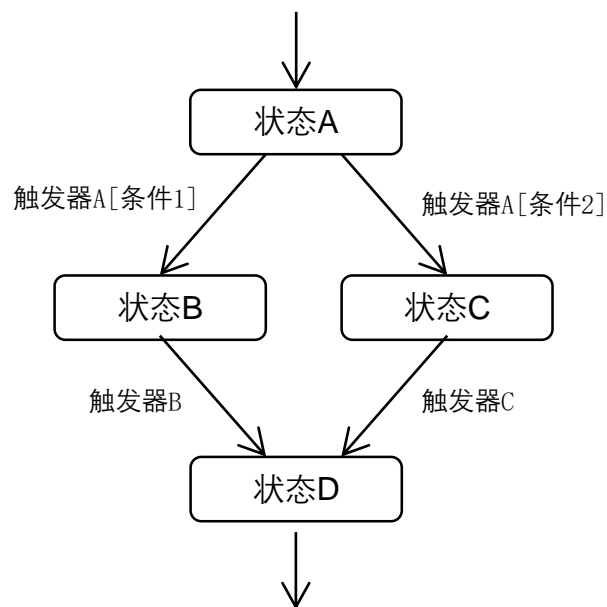
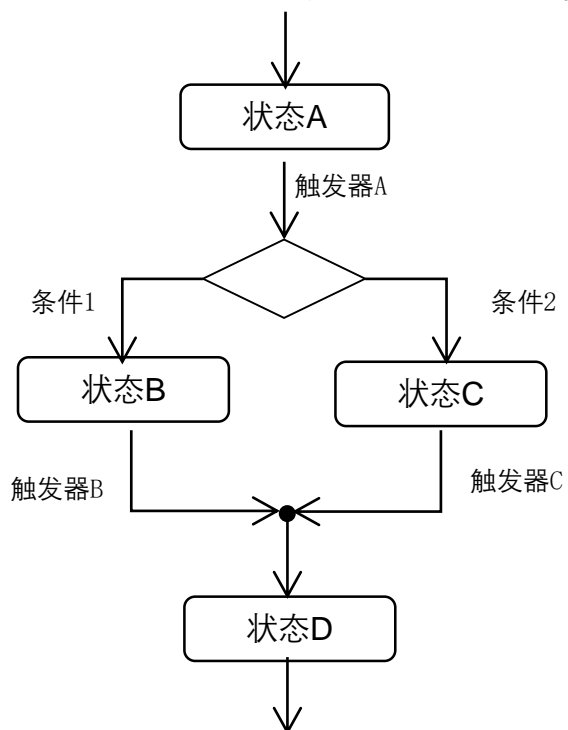
汇合

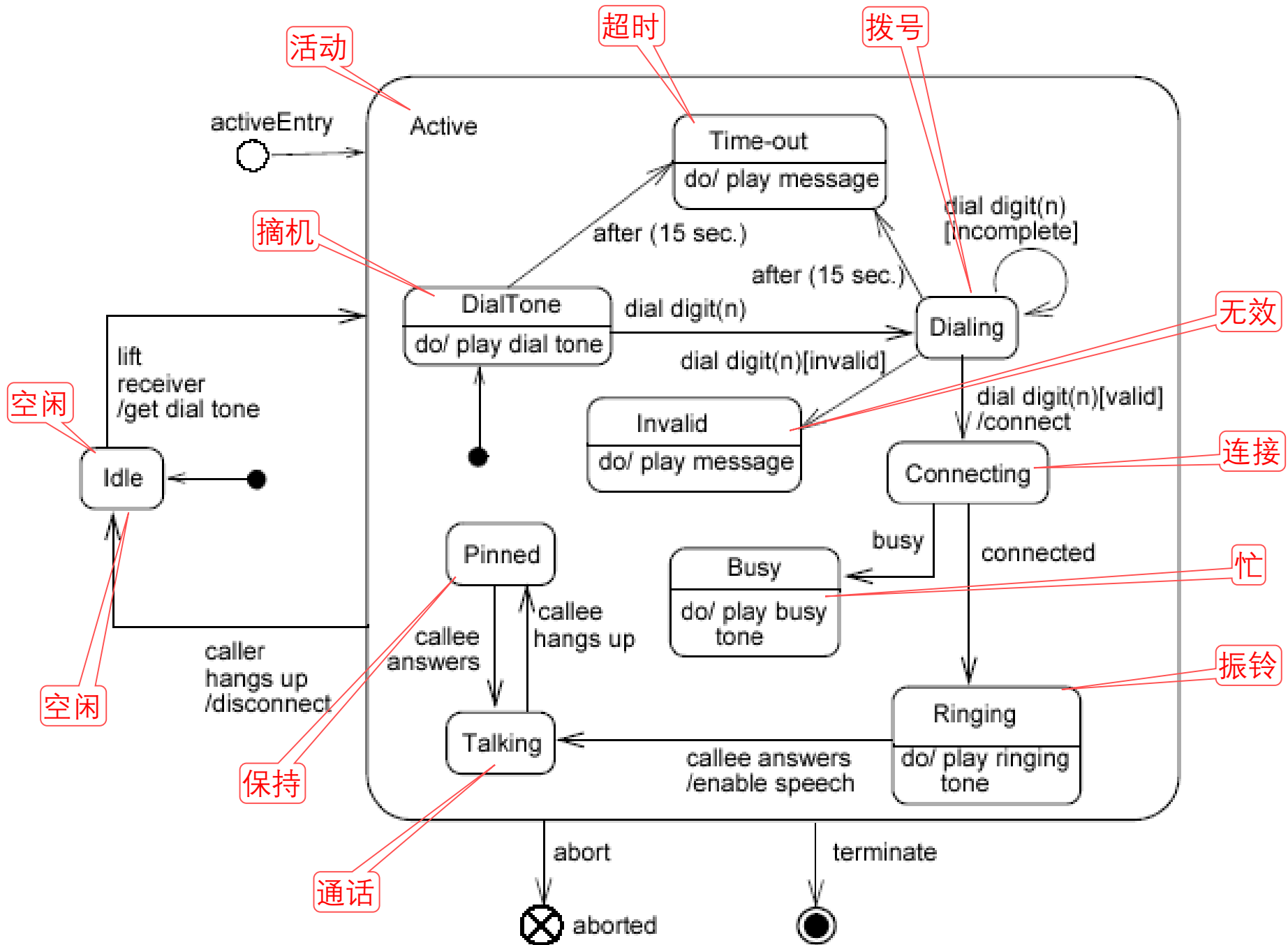


选择



接合





UML的其他几种图

对象图 (object diagram)

组合结构图 (composite structure diagram)

通信图 (communication diagram)

交互概览图 (interaction overview diagram)

定时图 (timing diagram)

部署图 (deployment diagram)

构件图，部署图：辅助模型，在转入实现阶段之前，可以用它们表示如何组织构件以及如何把软件制品部署的各个结点（计算机）上。

组合结构图、交互概览图、定时图：都可以作为辅助模型，无强烈建议。

对象图、通信图：建议不使用。

其他图 数据流图

某学校欲开发图书管理系统，以记录图书馆所藏图书及其借出和归还情况，提供给借阅者借阅图书功能，提供给图书馆管理员管理和定期更新图书表功能。主要功能的具体描述如下：

(1)处理借阅。借阅者要借阅图书时，系统必须对其身份(借阅者ID)进行检查。通过与教务处维护的学生数据库、人事处维护的职工数据库中的数据进行比对，以验证借阅者ID是否合法。若合法，则检查借阅者在逾期未还图书表中是否有逾期未还图书，以及罚金表中的罚金是否超过限额。如果没有逾期未还图书并且罚金未超过限额，则允许借阅图书，更新图书表，并将借阅的图书存入借出图书表。借阅者归还所借图书时，先由图书馆管理员检查图书是否缺失或损坏，若是，则对借阅者处以相应罚金并存入罚金表；然后，检查所还图书是否逾期，若是，执行“处理逾期”操作；最后，更新图书表，删除借出图书表中的相应记录。

(2)维护图书。图书馆管理员查询图书信息；在新进图书时录入图书信息，存入图书表；在图书丢失或损坏严重时，从图书表中删除该图书记录。

(3)处理逾期。系统在每周一统计逾期未还图书，逾期未还的图书按规则计算罚金，并记入罚金表，并给有逾期未还图书的借阅者发送提醒消息。借阅者在借阅和归还图书时，若罚金超过限额，管理员收取罚金，并更新罚金表中的罚金额度。

现采用结构化方法对该图书管理系统进行分析与设计，获得如图1-1所示的顶层数据流图和如图1-2所示的0层数据流图。

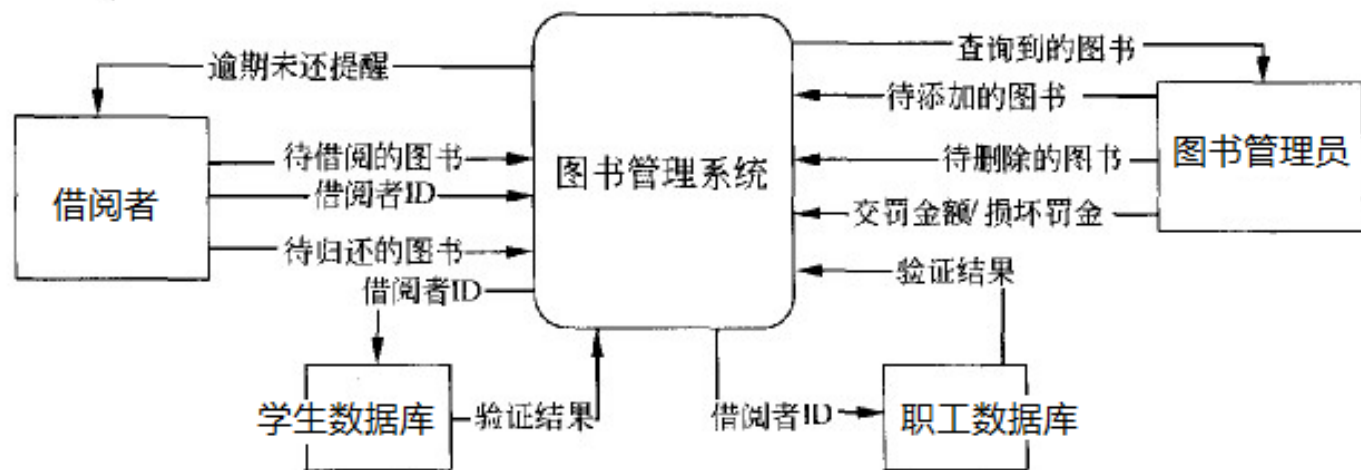


图 1-1 顶层数据流图

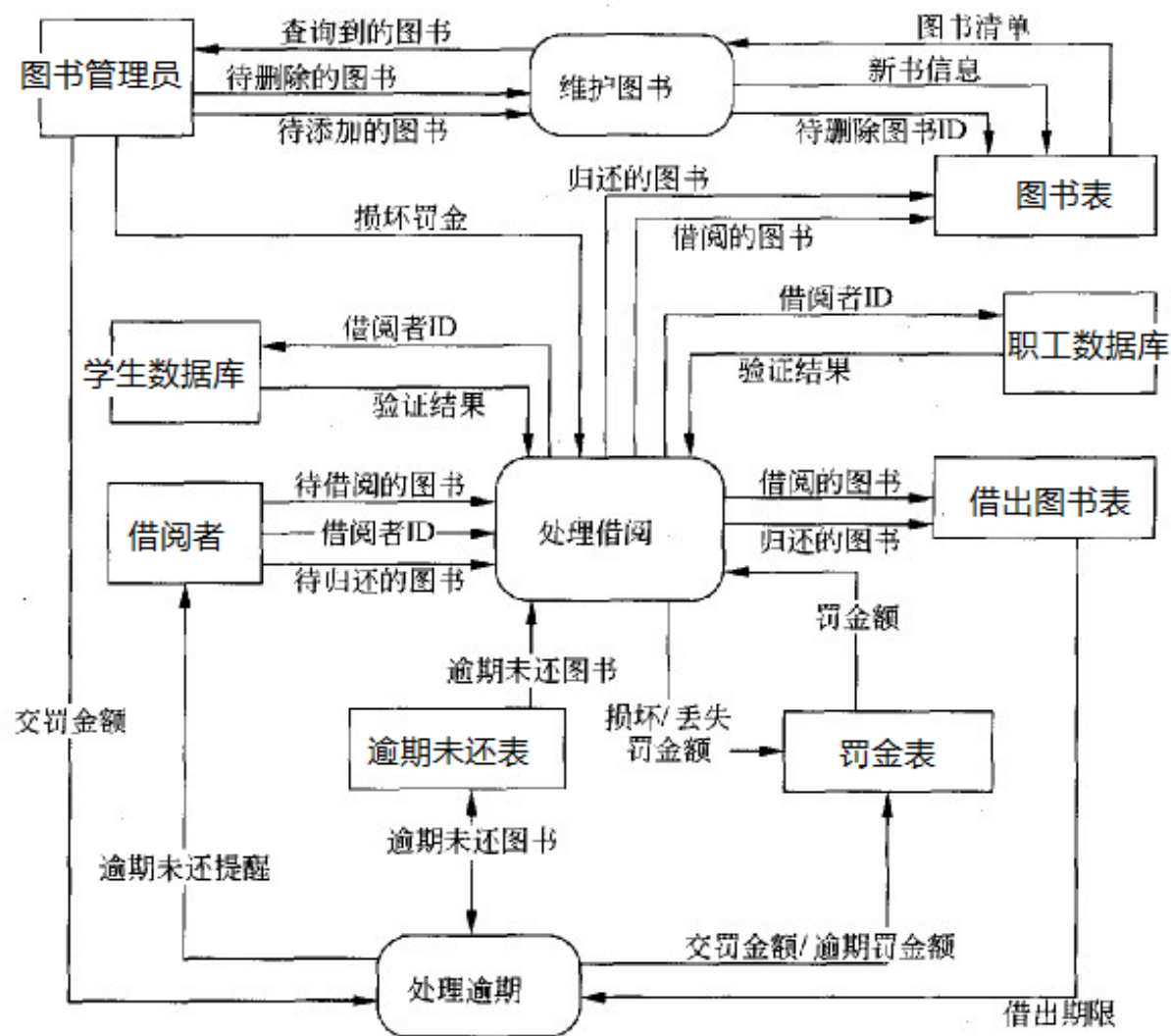
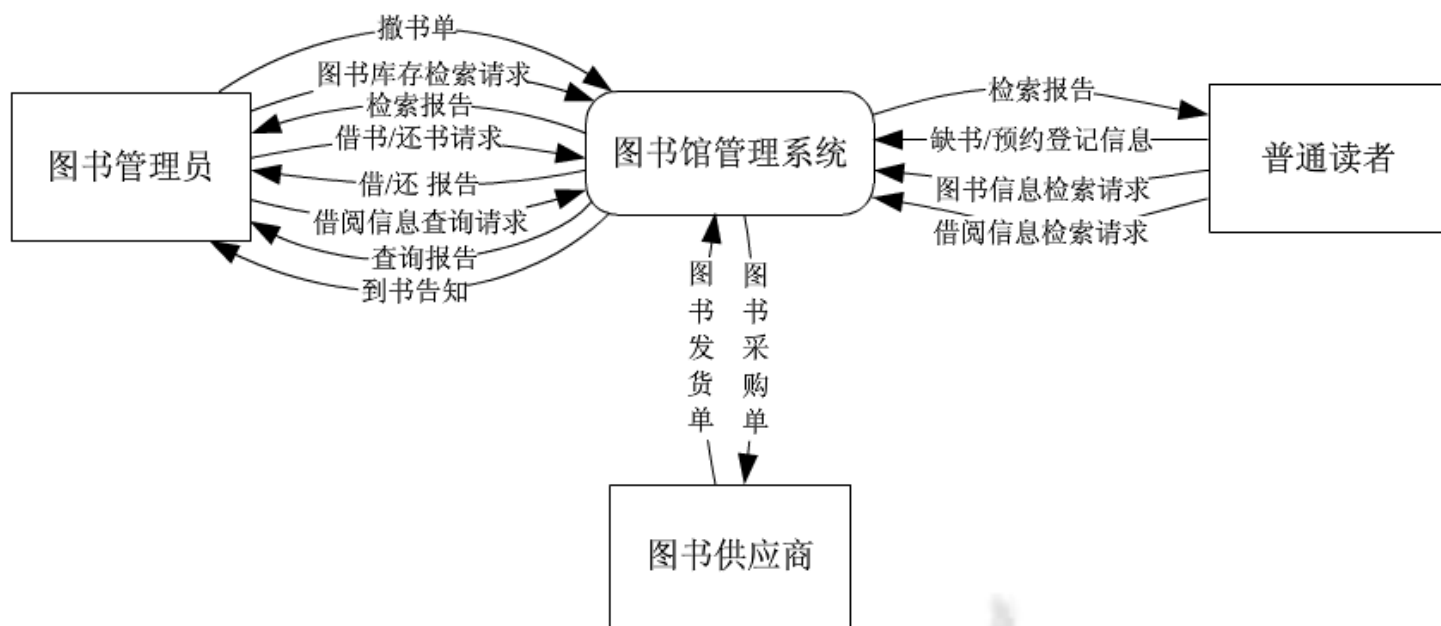
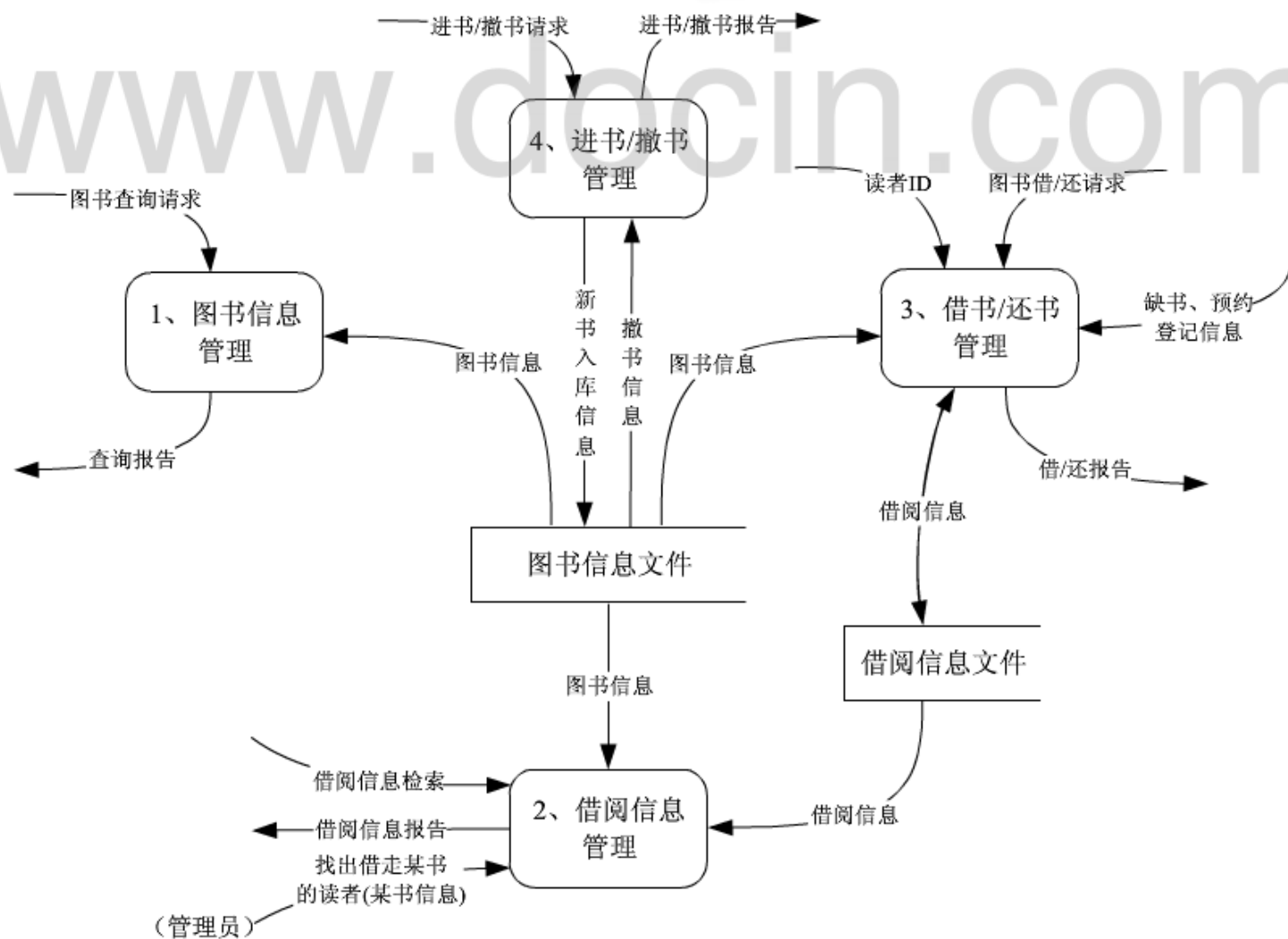


图 1-2 0层数据流图

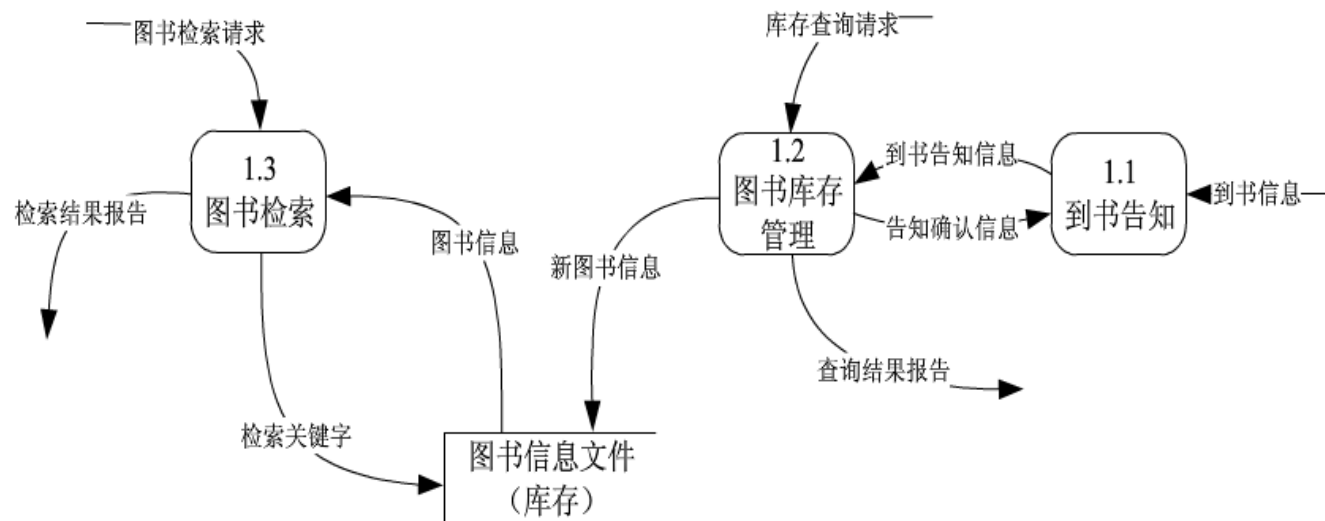
(顶层图)



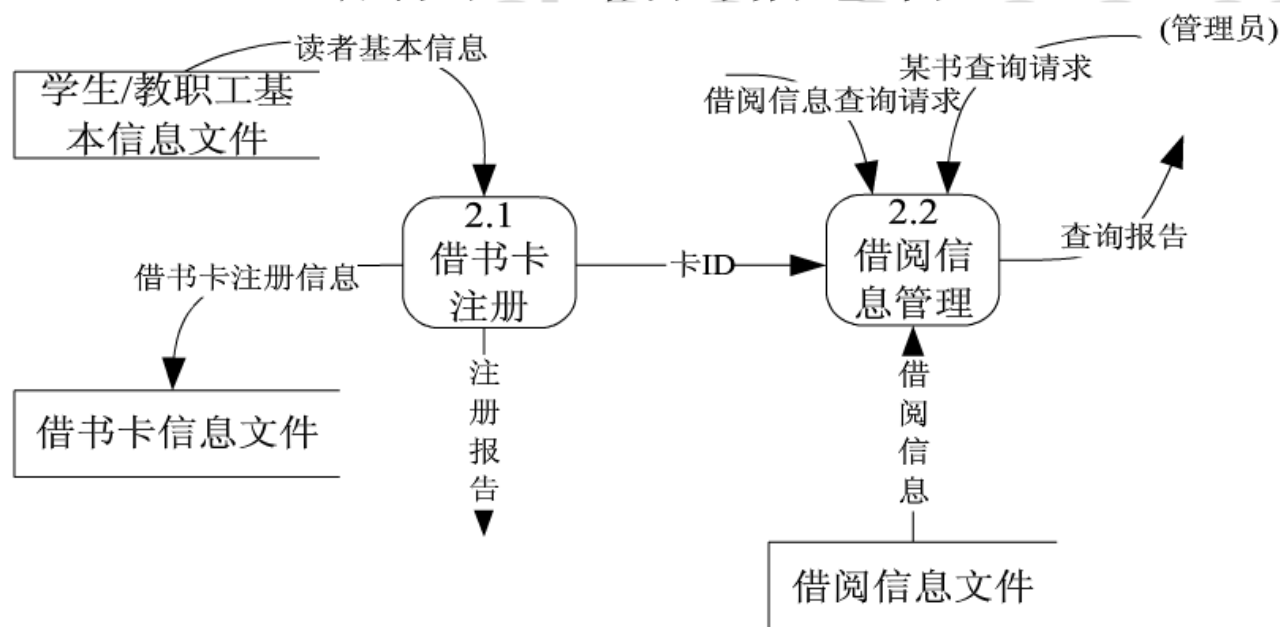
1层图



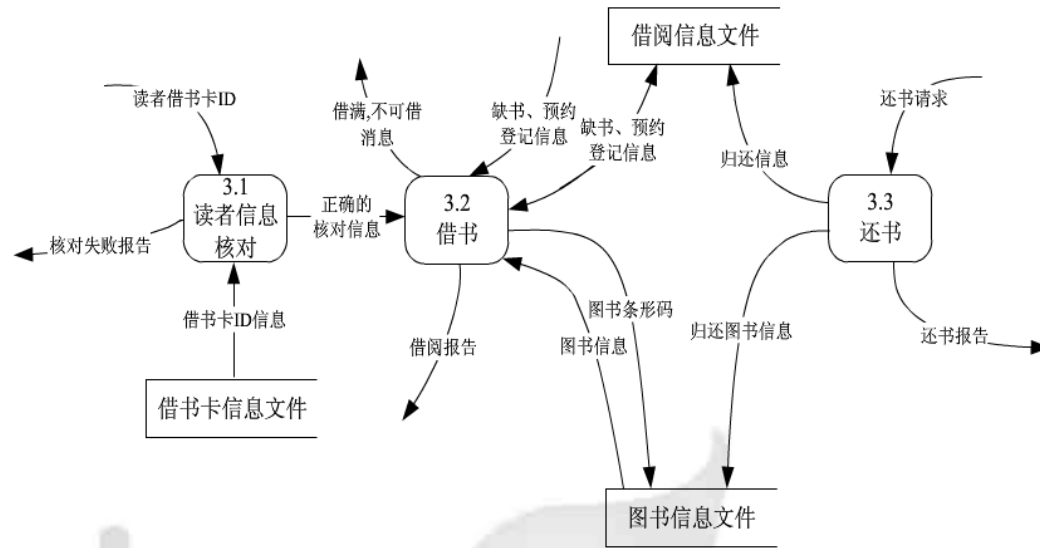
2层图1（图书信息管理）



2层图2（借阅信息管理）



2层图3（借书/还书管理）



2层图4（进书/撤书管理）

