# 分析软件需求

# 内容

## 1. 需求分析概述

- ✓需求分析任务
- ✓UML分析模型

## 2. 需求分析过程

- ✓确定需求优先级
- ✓建立需求分析模型

## 3. 需求文档化及评审

- ✓软件需求规格说明书
- ✓评审软件需求

# 1.1 软件需求分析的任务

- 精化和分析软件需求
- 建立软件需求模型
- 确定软件需求优先级
- 发现需求缺陷
- 编写软件需求规格说明书

# 1.2 初步需求中的问题(1/3)

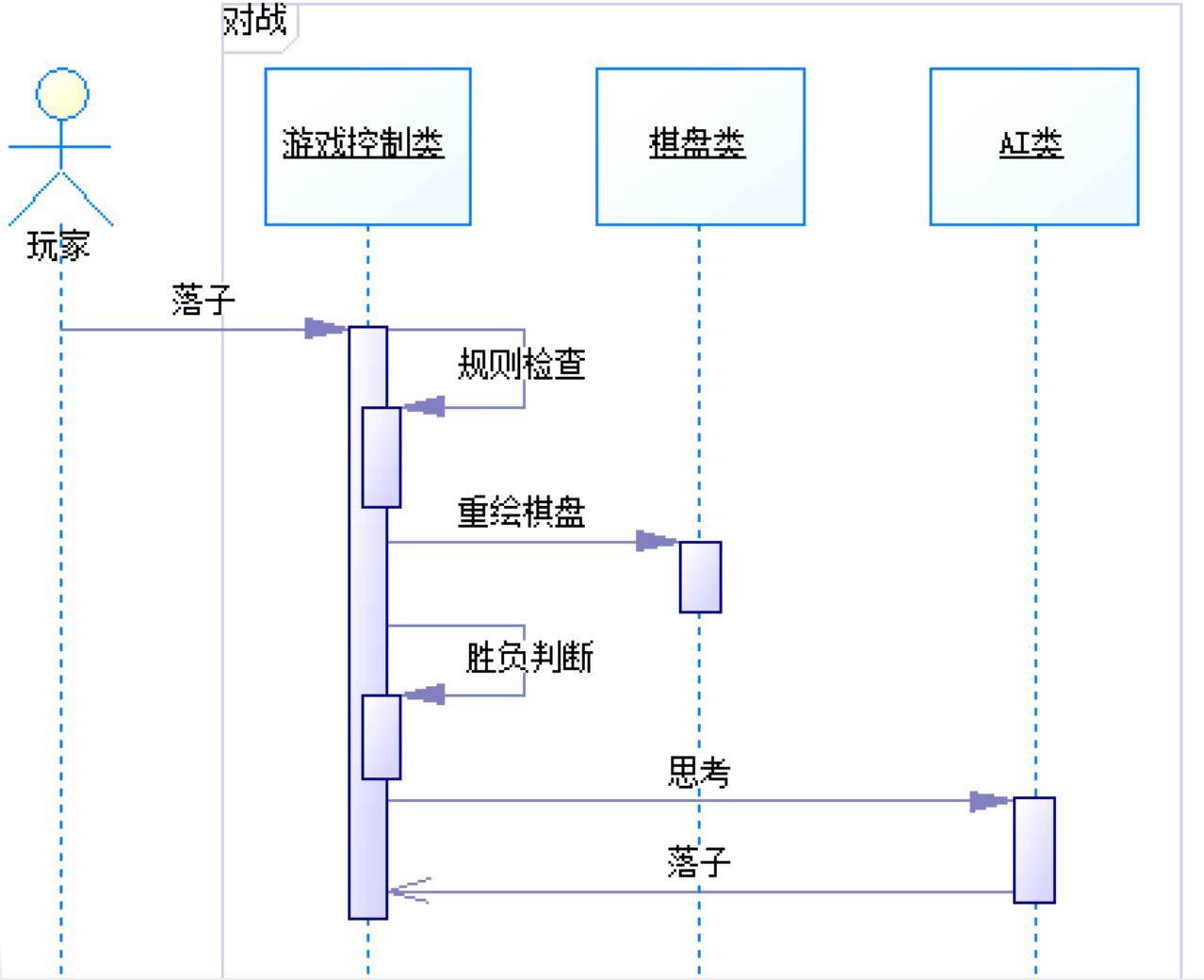
- **不具体**，没有提供软件需求的细节，无法支持后续的设计工作。
- 软件设计、实现的核心是**类**（包括行为），但目前需求仅明确了有哪些用例，不清楚用例实现涉及哪些类，类之间如何协作。

- 例如：“五子棋”游戏系统包括“下棋”、“悔棋”等用例，以与AI“下棋”为例，其事件流包括：
  - ✓ 玩家落子
  - ✓ 系统检查是否合规；如果合规，则更新棋盘；判断玩家是否已获胜，如果获胜，用例结束，否则AI思考并落子，更新棋盘显示。

- 对该用例的需求细化工作包括：

# 1.2 初步需求中的问题(2/3)

## （1）根据用例描述中的事件流，绘制交互模型



# 1.2 初步需求中的问题(2/3)

## (2) 从交互模型中识别用例的参与对象，进而绘制类图

用例“下棋”中识别出的类：棋盘类、玩家类、游戏控制类、Ai类。

进一步分析类之间的关系，即可得到UML分析类图。

## (3) 完善、分析类图中类的行为

游戏控制类方法

- ✓ 规则检查　　胜负判断

棋盘类方法　　AI类方法

- ✓ 重绘棋盘　　思考(生成策略)

# 1.3 软件需求模型及表示

- 由上例可知，细化软件需求，需要从多个不同视角进行分析

| 需求模型 | 视角 | 软件需求的内涵 | 需求阶段 |
| --- | --- | --- | --- |
| **用例图** | 用例 | 软件的功能及相互间的关系 | 获取软件需求 |
| **交互图** | 行为 | 功能完成所涉及的对象及相互间的交互和协作 | 分析软件需求 |
| **状态图** | 行为 | 对象的状态变化 | 分析软件需求 |
| **类图** | 结构 | 业务领域概念及相互关系 | 分析软件需求 |

# 1.3.1 交互图

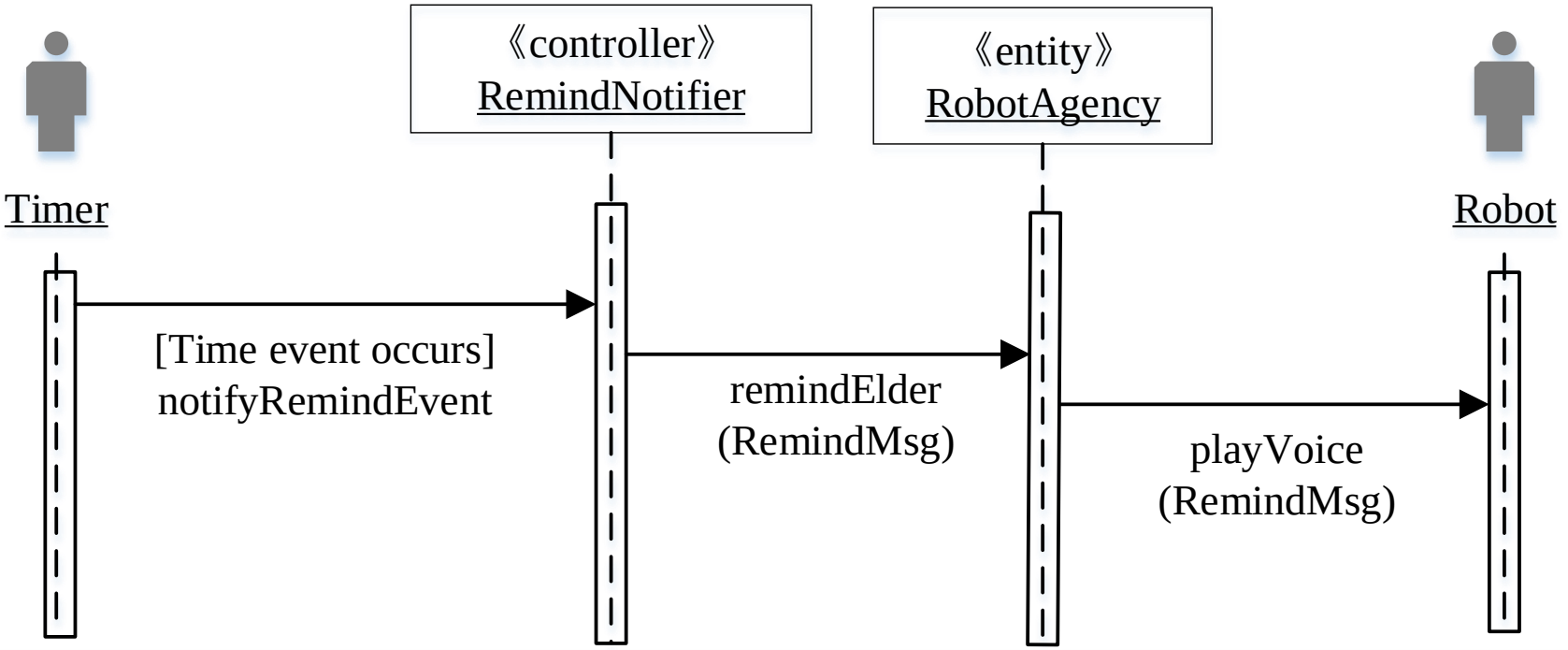
- **作用：刻画对象间的消息传递，分析某种用例场景下对象间的交互协作流程**

- **二类交互图**
  - ✓顺序图(Sequence Diagram)：强调消息传递的时间序
  - ✓通信图(Communication Diagram)：突出对象间的合作

- **表达能力相同，二类图可相互转换**

# （1）顺序图

- **描述对象间的消息交互序列**
  - ✓**纵向**：时间轴，**对象**及其**生命线**(虚线)和**活跃期**(长条矩形)
  - ✓**横向**：对象间的**消息传递**

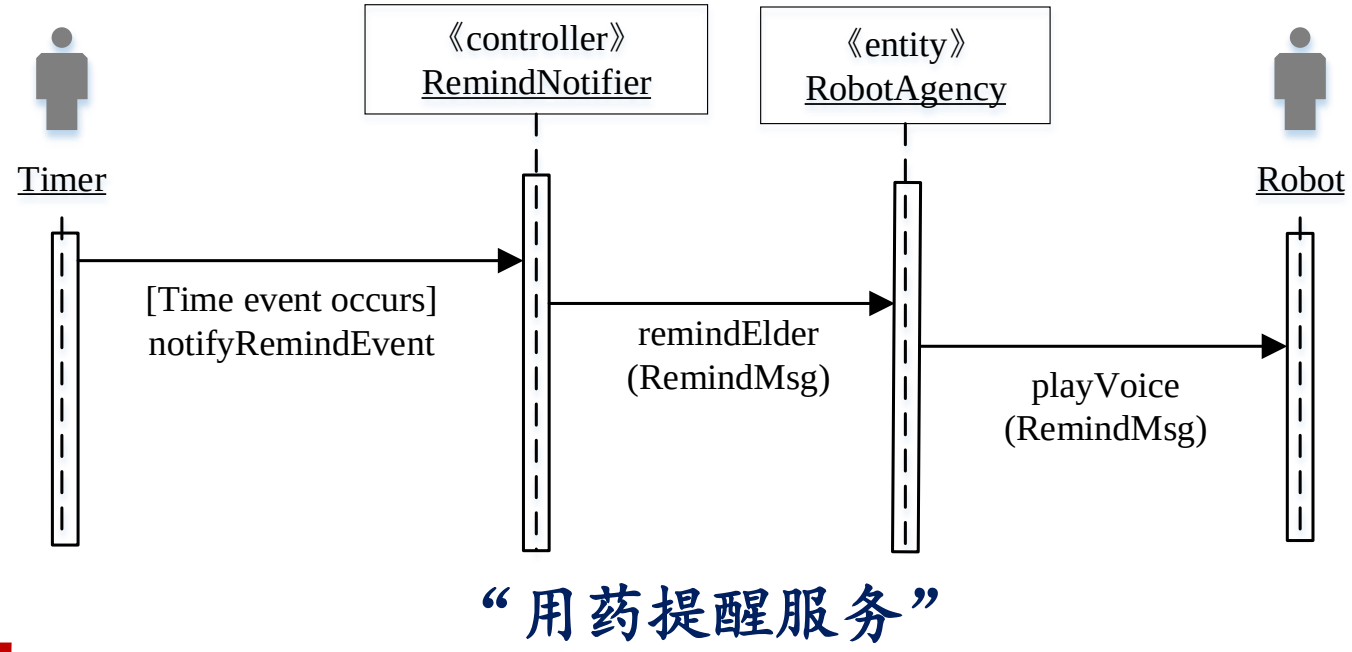


# 顺序图的表示方式

## ❑对象

- ✓ [对象名]：[类名]
- ✓ 示例：

Tom：Student 或 Student




“用药提醒服务”

## ❑消息传递

- ✓ 对象生命线之间的**有向边**
- ✓ [*][监护条件] [返回值:=]消息名[(参数表)]
- ✓ “*”表示迭代，表示将一个消息发送给同一类的多个对象

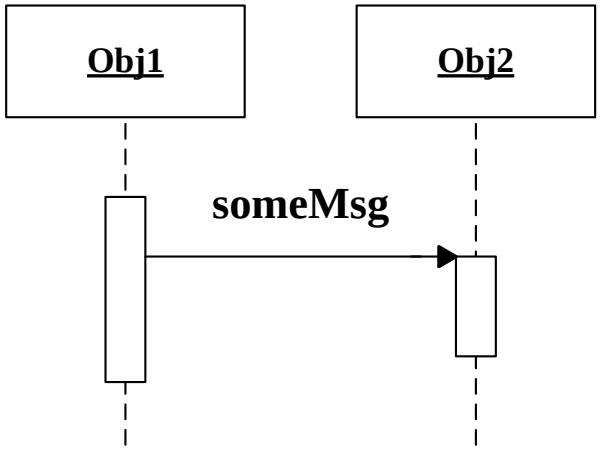
# 对象间的消息传递

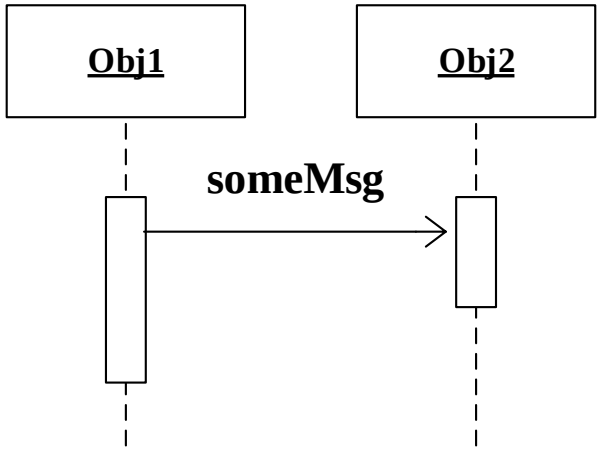
- **同步消息**
  - ✓发送者等待接收者将消息处理完后再继续
- **异步消息**
  - ✓发送者在发送完消息后不等待接收方即继续自己的处理
- **自消息**
  - ✓一个对象发送给自身的消息
- **返回消息**
  - ✓某条消息处理已经完成，处理结果沿返回消息传回
- **创建消息和销毁消息**
  - ✓消息传递目标对象的创建和删除

# 消息图元的表示

## 注意箭头的图形表示(线条和箭头形状)

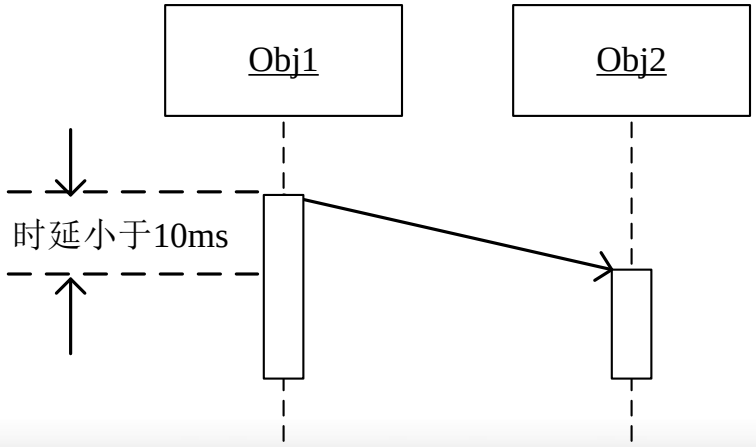
同步消息

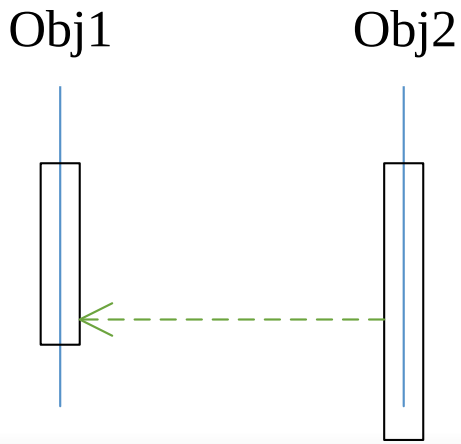



异步消息

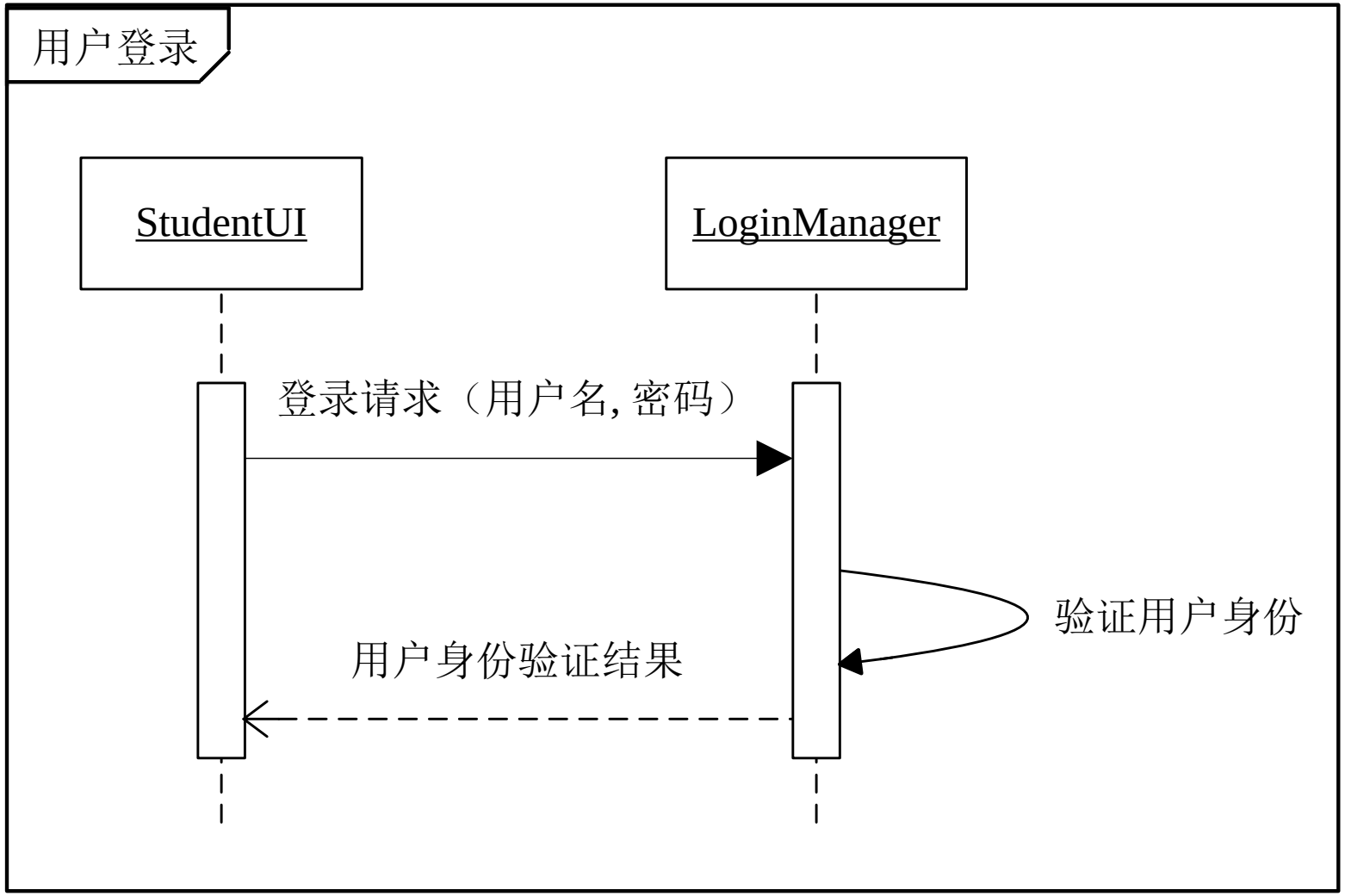
带时间延迟的消息




返回消息

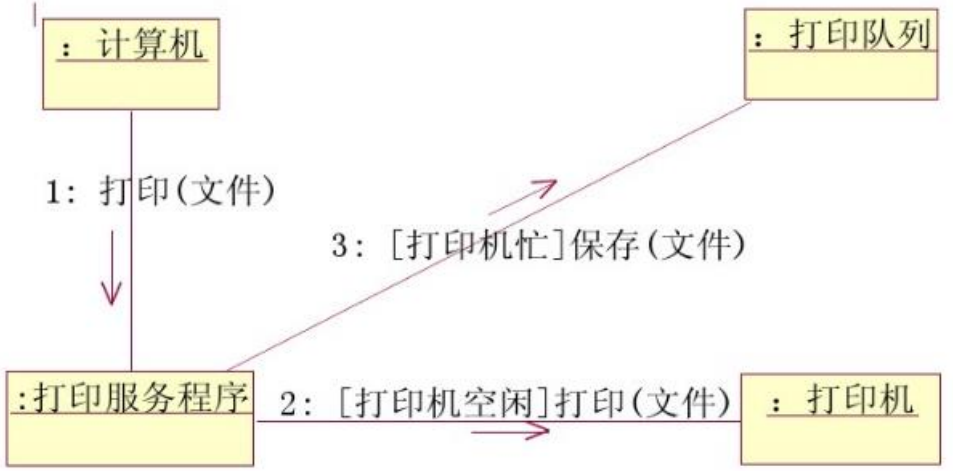
# 示例：顺序图

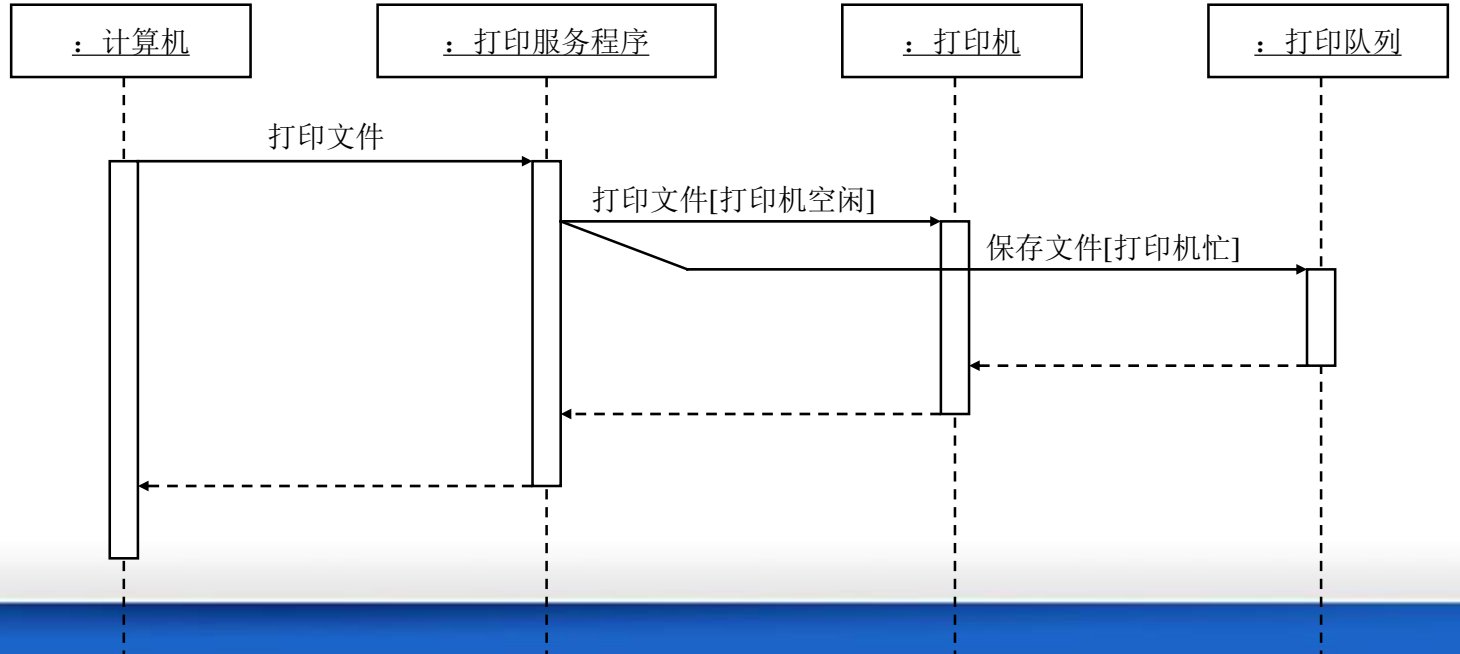



**特别注意图符的画法和要求**

# （2）通信图的表示

- 节点表示对象
- 对象间连接称为连接器
- 连接器上可标示发送的消息，小箭头表示消息方向



# 顺序图和通信图的选取原则

- **顺序图和通信图语义上等价**
  - ✓没必要针对同一建模目标同时创建这二个图

- **选择原则**
  - ✓当需要强调消息传递的**时间序列**时采用顺序图
  - ✓当需要强调对象间的**交互协作关系**时采用通信图

# 1.3.2 类图和对象图

| 视点 | 图（diagram） | 说明 |
| --- | --- | --- |
| 结构 | 包图（package diagram） | 从包层面描述系统的静态结构 |
| | **类图（class diagram）** | **从类层面描述系统的静态结构** |
| | **对象图（object diagram）** | **从对象层面描述系统的静态结构** |
| | 构件图(component diagram) | 描述系统中构件及其依赖关系 |
| 行为 | 状态图(statechart diagram ) | 描述状态的变迁 |
| | 活动图(activity diagram) | 描述系统活动的实施 |
| | 通信图(communication diagram) | 描述对象间的消息传递与协作 |
| | 顺序图(sequence diagram) | 描述对象间的消息传递与协作 |
| 部署 | 部署图（deployment diagram） | 描述系统中工件在物理运行环境中的部署情况 |
| 用例 | 用例图（use case diagram） | 从外部用户角度描述系统功能 |

# 类图

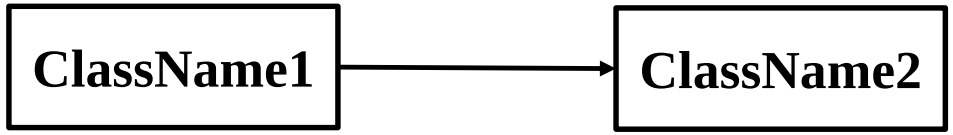
## ❑功效

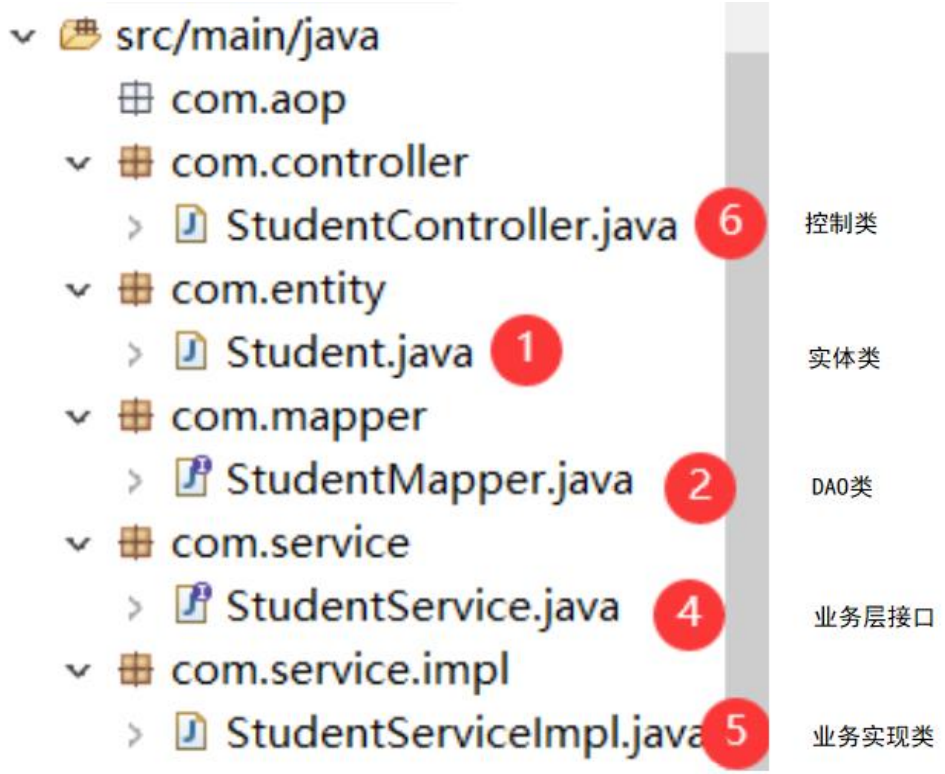
✓描述系统的**类构成**，刻画系统的**静态组成结构**

## ❑图的构成

✓**结点**：表示系统中的类（或接口）

✓**边**：　类之间的关系

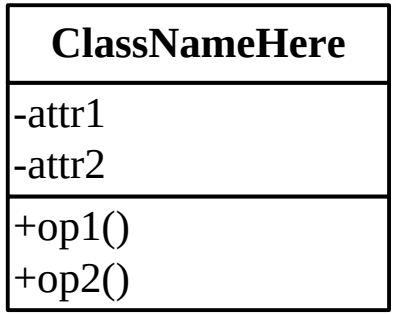


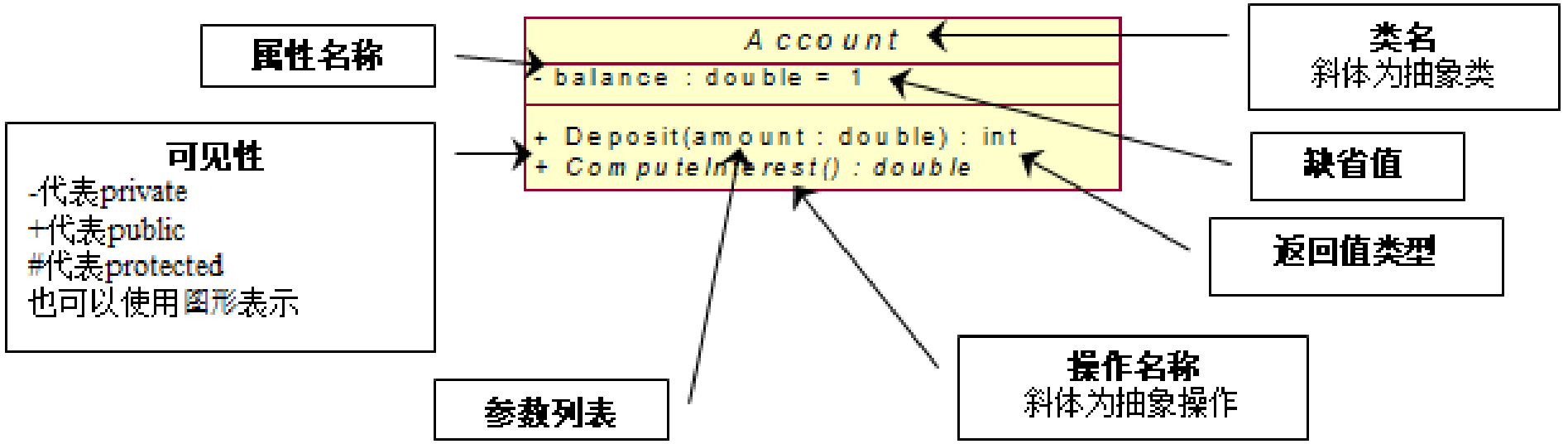

基于SSM框架的项目结构

**特别注意图符的画法和要求**

# 类的UML表示



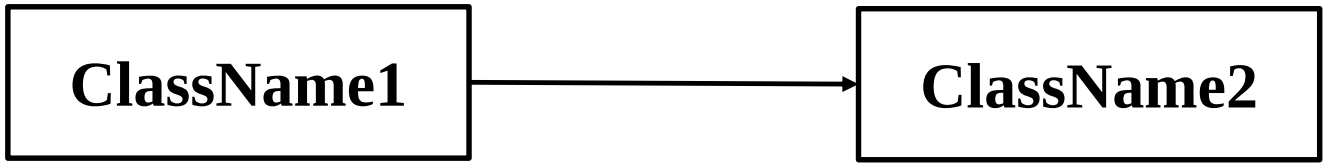

- ✓ 类名
- ✓ 属性
- ✓ 操作




**特别注意图符的画法和要求**

# 类间的关系

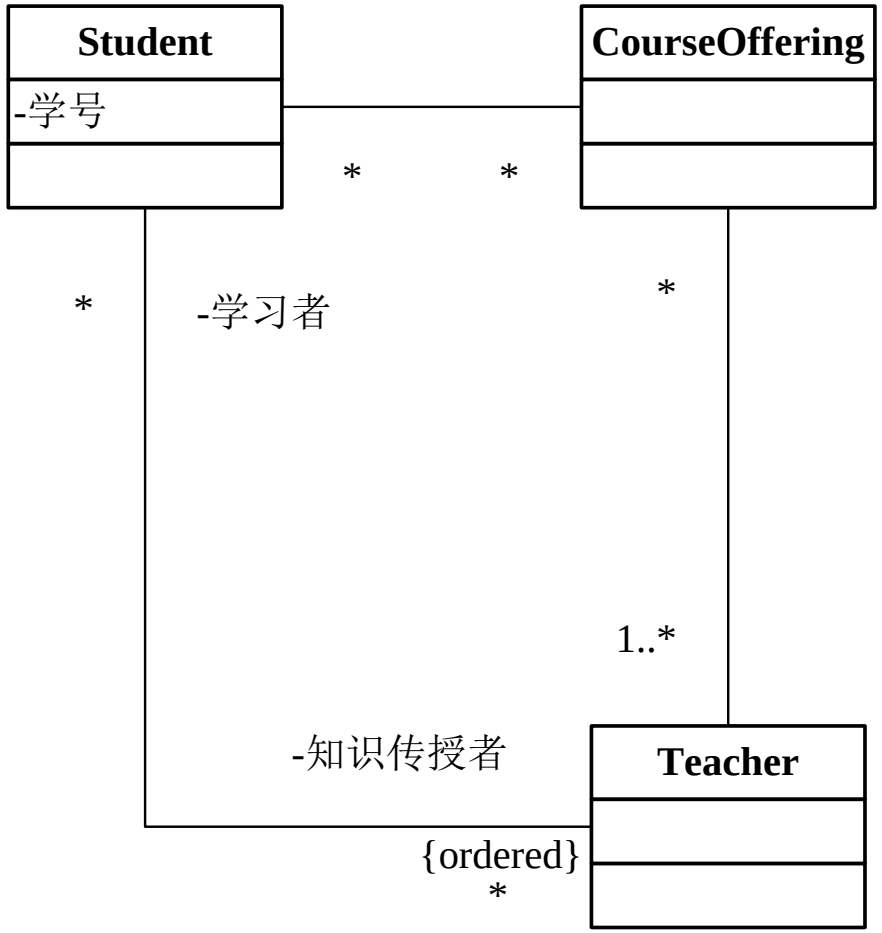
- 关联
- 依赖
- 继承
- 实现
- 聚合
- 组合




不同的关系采用不同的线条和箭头表示

# 类间关系-关联(1/5)

- 表示类间的逻辑联系
  - ✓**多重性**：一个类可以有多少个对象与另一个类的对象联系
  - ✓**角色名**：关联类对象在关系中扮演的角色
  - ✓**约束特性**：针对关联对象或对象集的逻辑约束



# 关联关系代码实现

## ❑Java实现


```
class  Department{
   private Employee e;
   private Employee[] es;
   public getCost(){
      return e.getcost();
  }
}
```

```
class Employee{
   private name;
   private cost;
   public getCost(){
      return cost;
  }
}
```

# 类间关系-聚合与组合(2/5)

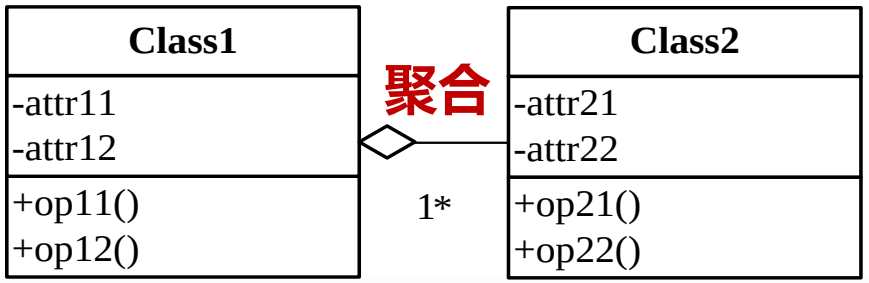
## ❑聚合关系(Aggregation)

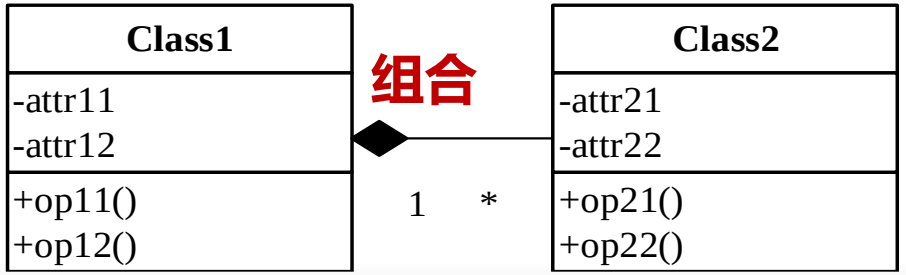
- ✓聚合表示一种逻辑上的**整体-部分关系**
- ✓关系连接的两个类对象，其生命周期是独立的
- ✓示例：汽车与轮胎

**注意组合和聚合的差异性**

## ❑组合关系(Composition)

- ✓组合是关联更强的聚合。**部分与整体共存亡**，是物理包容。
- ✓两者不可分割，其生命周期也是一致的

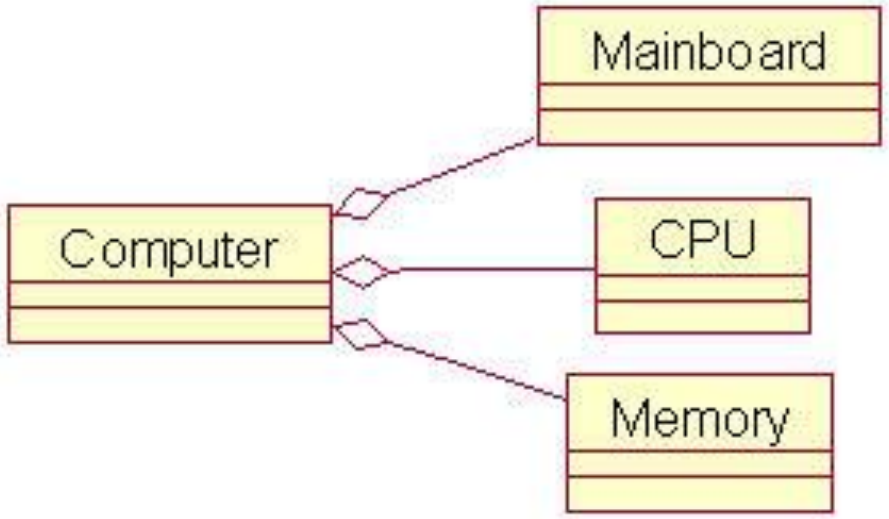



菱形所在的一端是整体

# 聚合关系实例

- 电脑由主板、CPU等聚合而成




```
public class Computer{
    private CPU cpu;
    public CPU getCPU(){
        return cpu;
    }
    public void setCPU(CPU cpu){
        this.cpu=cpu;
    }
    //开启电脑
    public void start(){
        //cpu运作
        cpu.run();
    }
```

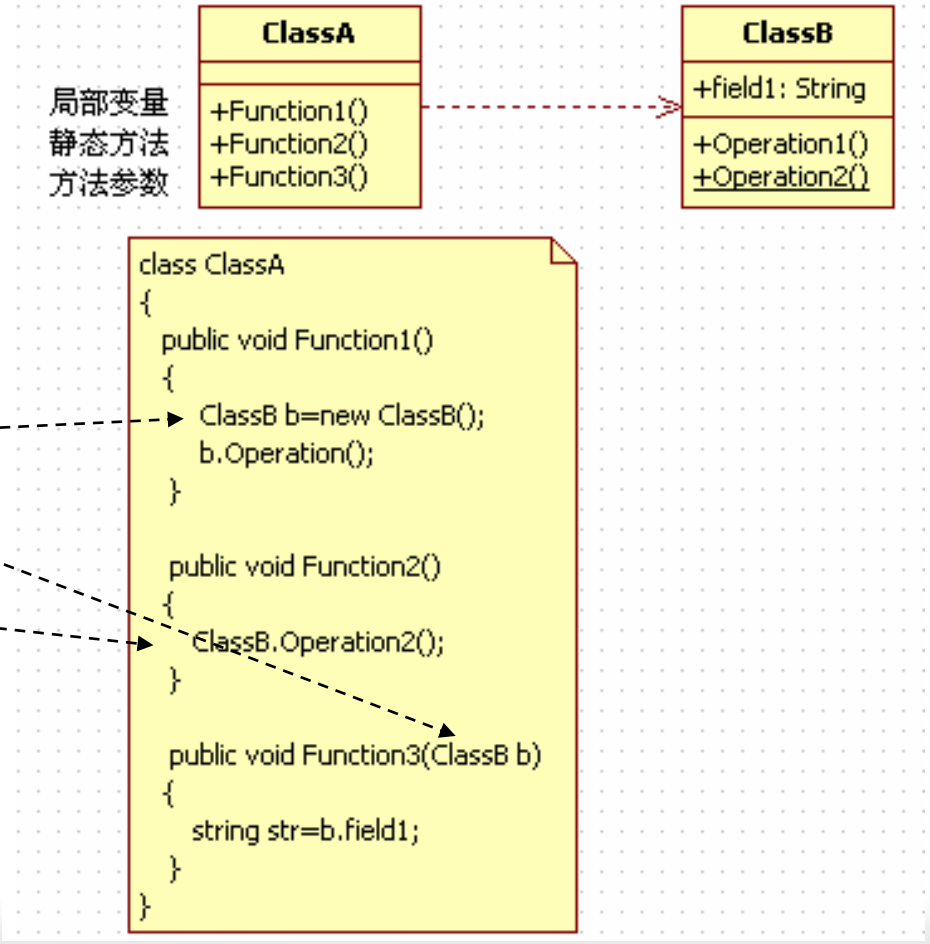
# 类间关系-依赖(3/5)

## ❑依赖关系

- ✓一个类B的变化会导致另一类A作相应修改，则称A依赖于B
- ✓依赖关系是一种临时性的、非常弱的关系

## ❑类A依赖于B，表现形式有三种

- ✓B是A中成员方法的参数
- ✓或A的局部变量
- ✓或静态方法调用



# 依赖与关联关系的区别

## ❑区别依赖与关联

✓当某个类以成员变量形式出现在另一类中，两者是关联关系；

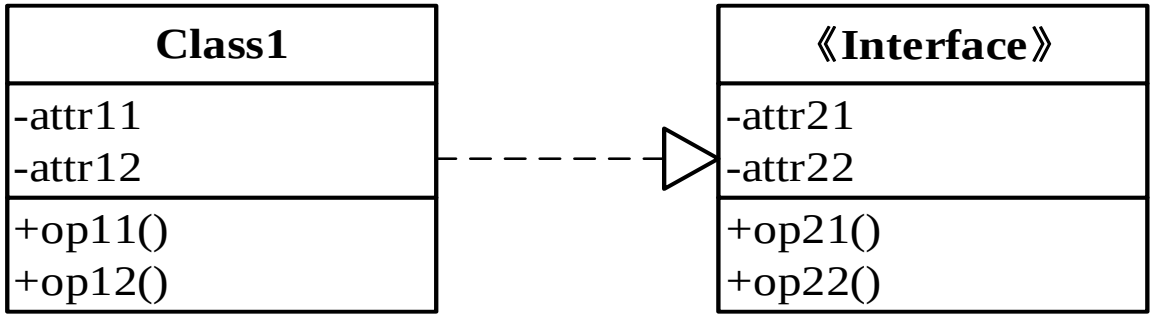
✓某个类以局部变量的形式出现在另一类中，二者是依赖关系。

✓关联关系是一种静态存在，不管你用不用，关系它都是存在的；

✓依赖关系是一种临时存在，调用时才有关系，不用就没关系。

# 类间关系-实现(4/5)

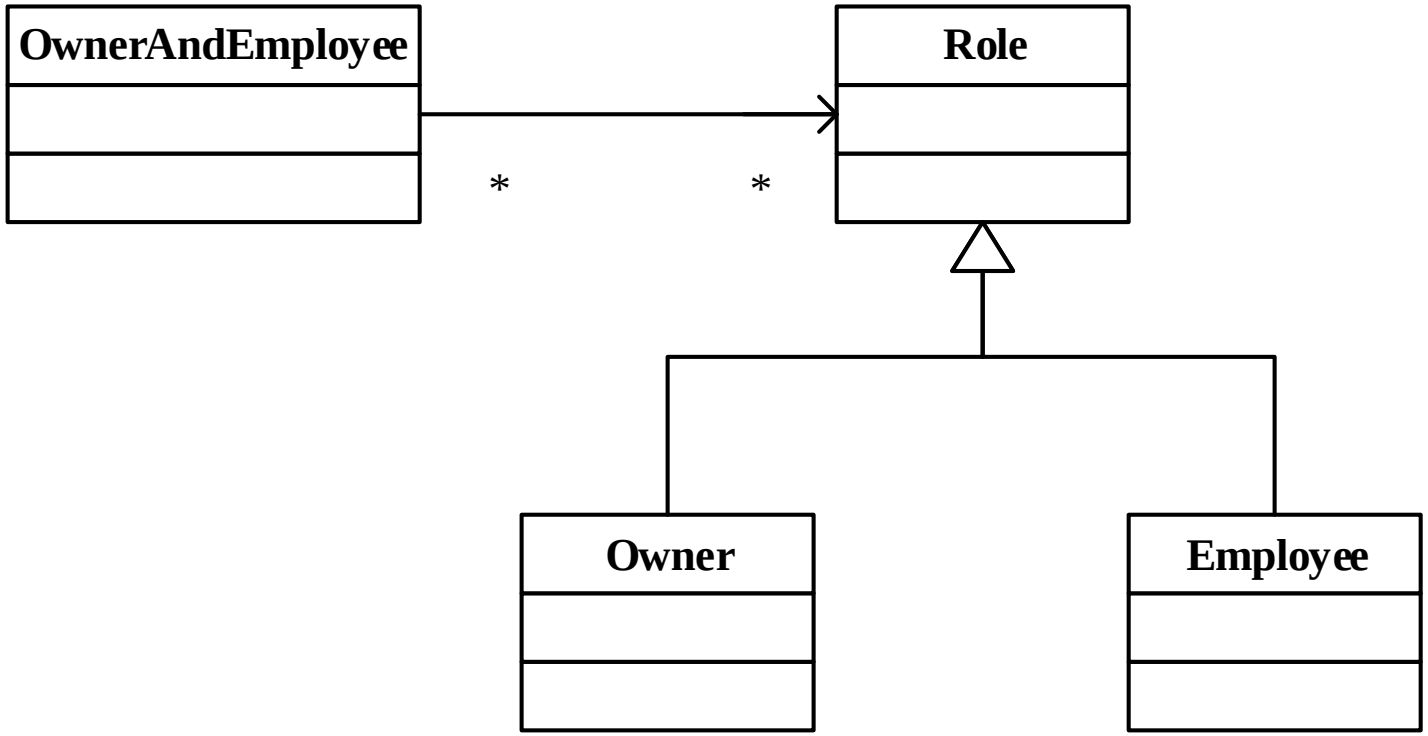
- 表示一个类实现了一个接口




特别注意图符的画法和要求

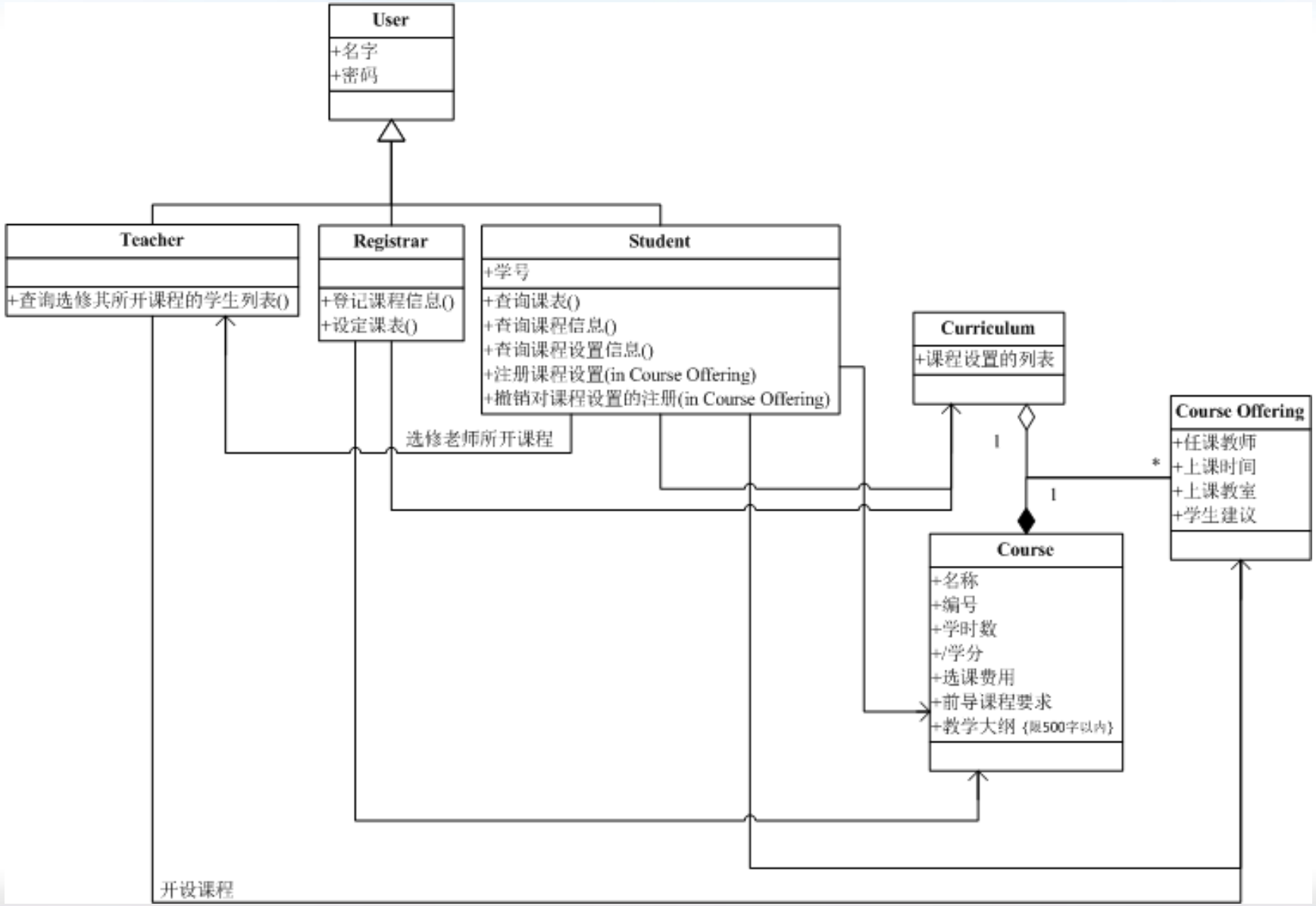
# 类间关系-继承(5/5)

- 子类**继承**父类所有可继承的特性，且可通过**添加**新特性，或覆盖父类中的原有特性



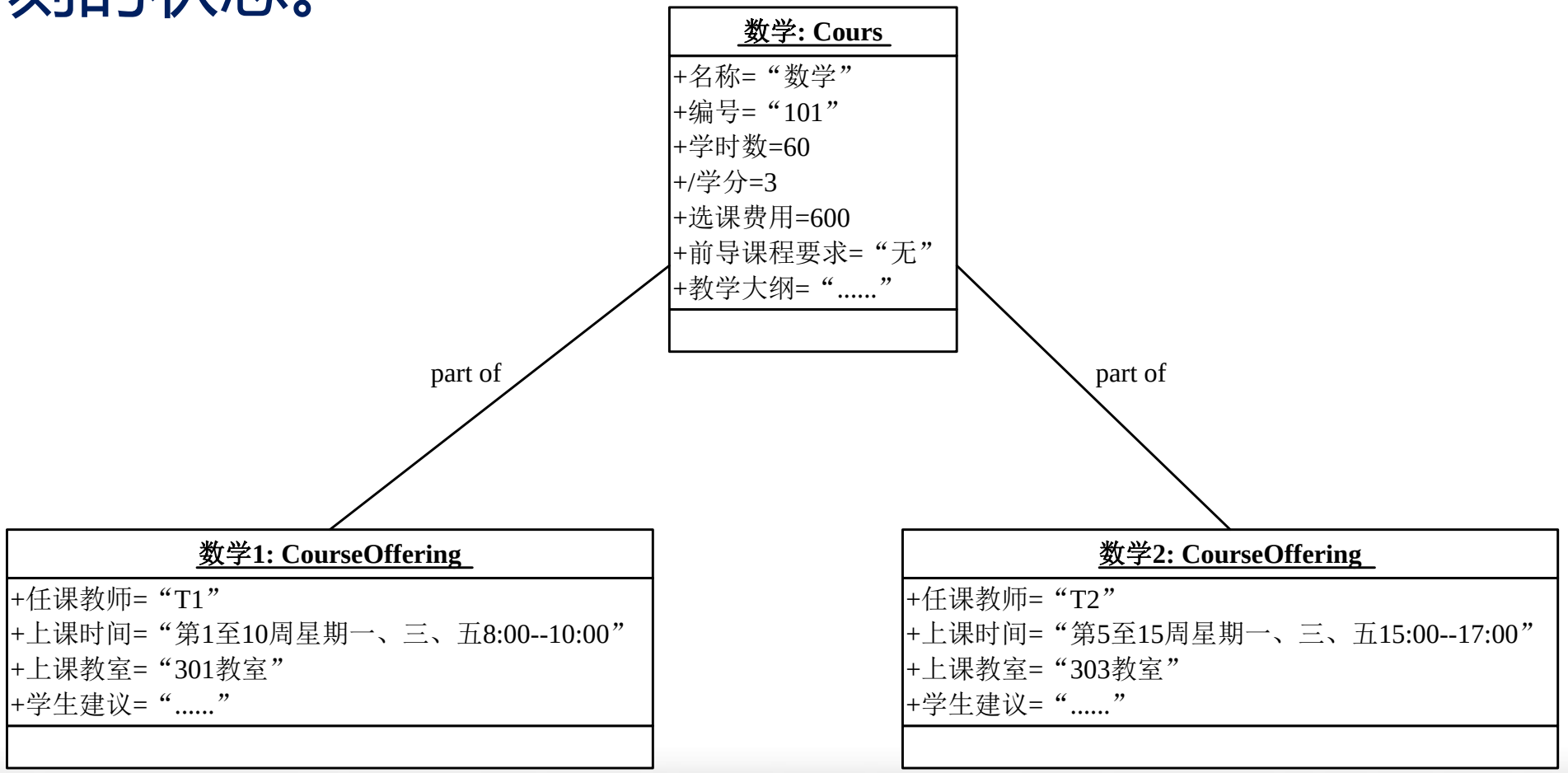

**特别注意图符的画法和要求**

# 示例：类图


User
+名字
+密码
Teacher
+查询选修其所开课程的学生列表()
Registrar
+登记课程信息()
+设定课表()
Student
+学号
+查询课表()
+查询课程信息()
+查询课程设置信息()
+注册课程设置(in Course Offering)
+撤销对课程设置的注册(in Course Offering)
Curriculum
+课程设置的列表
Course Offering
+任课教师
+上课时间
+上课教室
+学生建议
Course
+名称
+编号
+学时数
+/学分
+选课费用
+前导课程要求
+教学大纲 {限500字以内}
选修老师所开课程
开设课程
1
*
1

# 对象图

- 对象图可被看作是类图的实例，用来表达各个对象在某一时刻的状态。



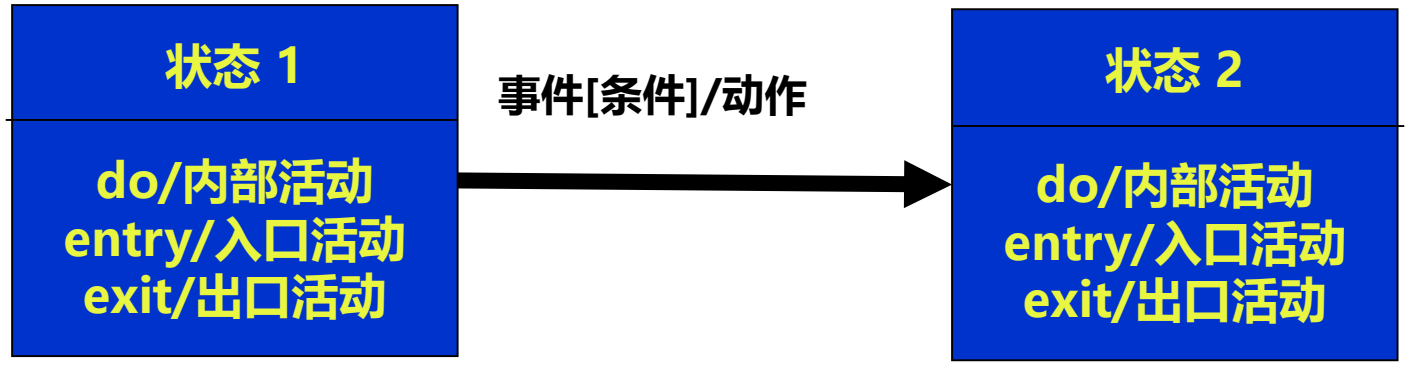
# 1.3.3 状态图

| 视点 | 图（diagram） | 说明 |
| --- | --- | --- |
| 结构 | 包图（package diagram） | 从包层面描述系统的静态结构 |
| | 类图（class diagram） | 从类层面描述系统的静态结构 |
| | 对象图（object diagram） | 从对象层面描述系统的静态结构 |
| | 构件图(component diagram) | 描述系统中构件及其依赖关系 |
| **行为** | **状态图(statechart diagram )** | **描述状态的变迁** |
| | 活动图(activity diagram) | 描述系统活动的实施 |
| | 通信图(communication diagram) | 描述对象间的消息传递与协作 |
| | 顺序图(sequence diagram) | 描述对象间的消息传递与协作 |
| 部署 | 部署图（deployment diagram） | 描述系统中工件在物理运行环境中的部署情况 |
| 用例 | 用例图（use case diagram） | 从外部用户角度描述系统功能 |

# 状态图

## ❑功效

- ✓描述实体（对象、系统）在**事件**刺激下的**行为**及其导致的**状态变化**



# 状态图的基本元素

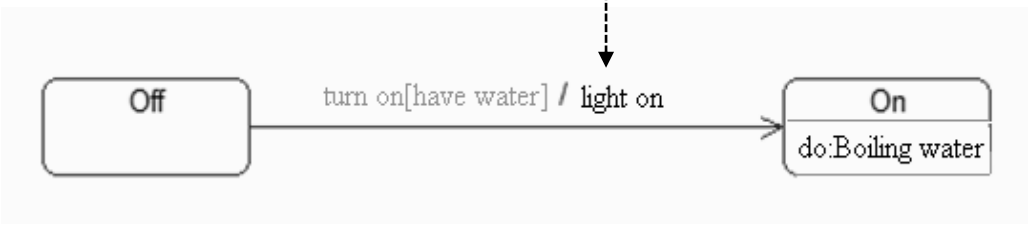
## ❑状态

- ✓一旦对象进入本状态，那么本状态**入口活动**将被执行
- ✓一旦对象退出本状态，那么本状态的**出口活动**将被执行
- ✓**do活动**是当对象进入本状态，执行完入口活动后执行的活动


## ❑事件

## ❑动作：状态迁移时应执行的行为

- ✓计算过程，**位于迁移边上**，执行时间短

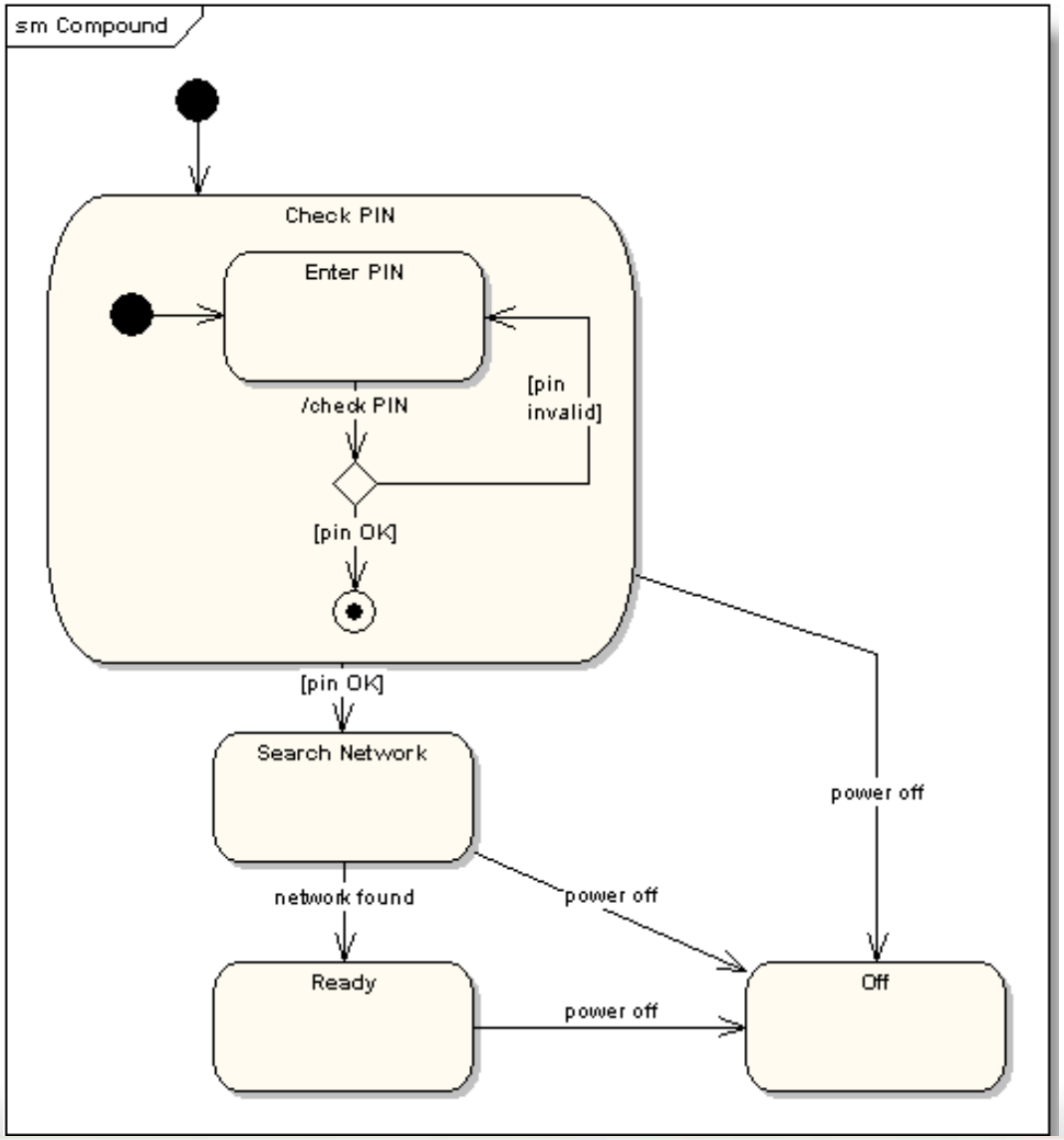



## ❑活动

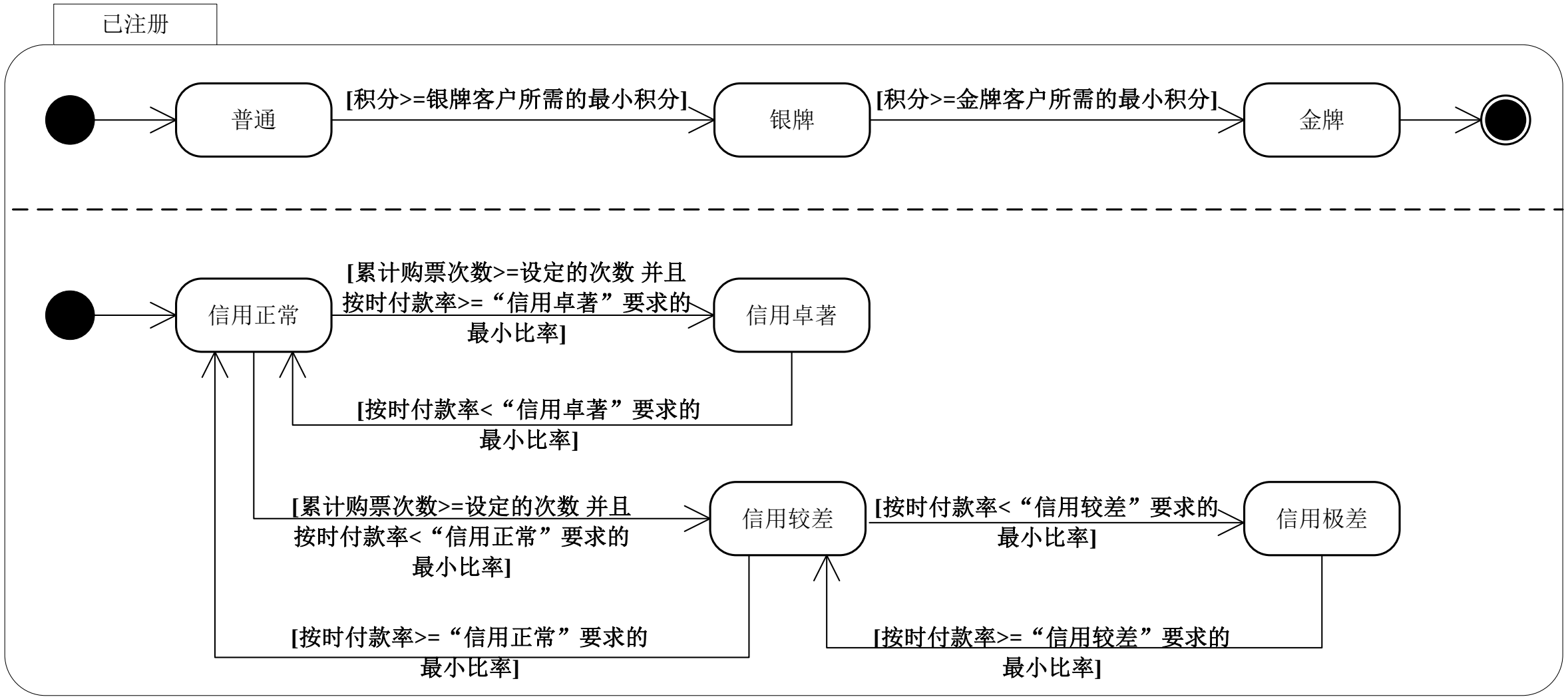
- ✓计算过程，**位于状态中**，执行时间长

# 状态图

- 几种特殊状态
  - ✓一个起始状态
  - ✓ 多个终止状态
  - ✓ 复合状态



# 示例：状态图


已注册
普通
[积分>=银牌客户所需的最小积分]
银牌
[积分>=金牌客户所需的最小积分]
金牌
信用正常
[累计购票次数>=设定的次数 并且
按时付款率>=“信用卓著”要求的
最小比率]
信用卓著
[按时付款率<“信用卓著”要求的
最小比率]
[累计购票次数>=设定的次数 并且
按时付款率<“信用正常”要求的
最小比率]
信用较差
[按时付款率<“信用较差”要求的
最小比率]
信用极差
[按时付款率>=“信用正常”要求的
最小比率]
[按时付款率>=“信用较差”要求的
最小比率]

# 内容

1. 需求分析概述
   - ✓需求分析任务
   - ✓UML分析模型
2. **需求分析过程**
   - ✓确定需求优先级
   - ✓建立需求分析模型
3. **需求文档化及评审**
   - ✓软件需求规格说明书
   - ✓评审软件需求

# 2.1 确定软件需求优先级

(1)分析软件需求**重要性**

(2)分析软件需求**优先级**

(3)确定用例分析和实现的**次序**

# 2.1.1 分析软件需求重要性

- **从用户和客户的视角，软件需求的重要性是不一样的**
- **核心软件需求**
  - ✓对解决问题起到举足轻重的作用，提供了软件特有的功能和服务，体现了软件系统的特色和优势
- **外围软件需求**
  - ✓提供了次要、辅助性的功能和服务

# 示例：“空巢老人看护软件”的需求重要性

| 序号 | 用例名称 | 用例标识 | 重要性 |
| --- | --- | --- | --- |
| 1 | 监视老人 | UC-MonitorElder | 核心 |
| 2 | 自主跟随老人 | UC-FollowElder | 核心 |
| 3 | 获取老人信息 | UC-GetElderInfo | 核心 |
| 4 | 检测异常状况 | UC-CheckEmergency | 核心 |
| 5 | 通知异常状况 | UC-NotifyEmergency | 核心 |
| 6 | 控制机器人 | UC-ControlRobot | 外围 |
| 5 | 视频/语音交互 | UC- A&VInteraction | 外围 |
| 6 | 提醒服务 | UC-AlertService | 外围 |
| 9 | 用户登录 | UC-UserLogin | 外围 |
| 10 | 系统设置 | UC-SetSystem | 外围 |

# 2.1.2 分析软件需求优先级

- **需求优先级的考虑因素**
  - ✓按照软件需求的**重要性**来确定其优先级
  - ✓按照用户的**实际需要**来确定软件需求的优先级
    - -用户急需的，需要尽早交付使用的需求，具有高优先级

# 示例：确定“空巢老人看护软件”的需求优先级

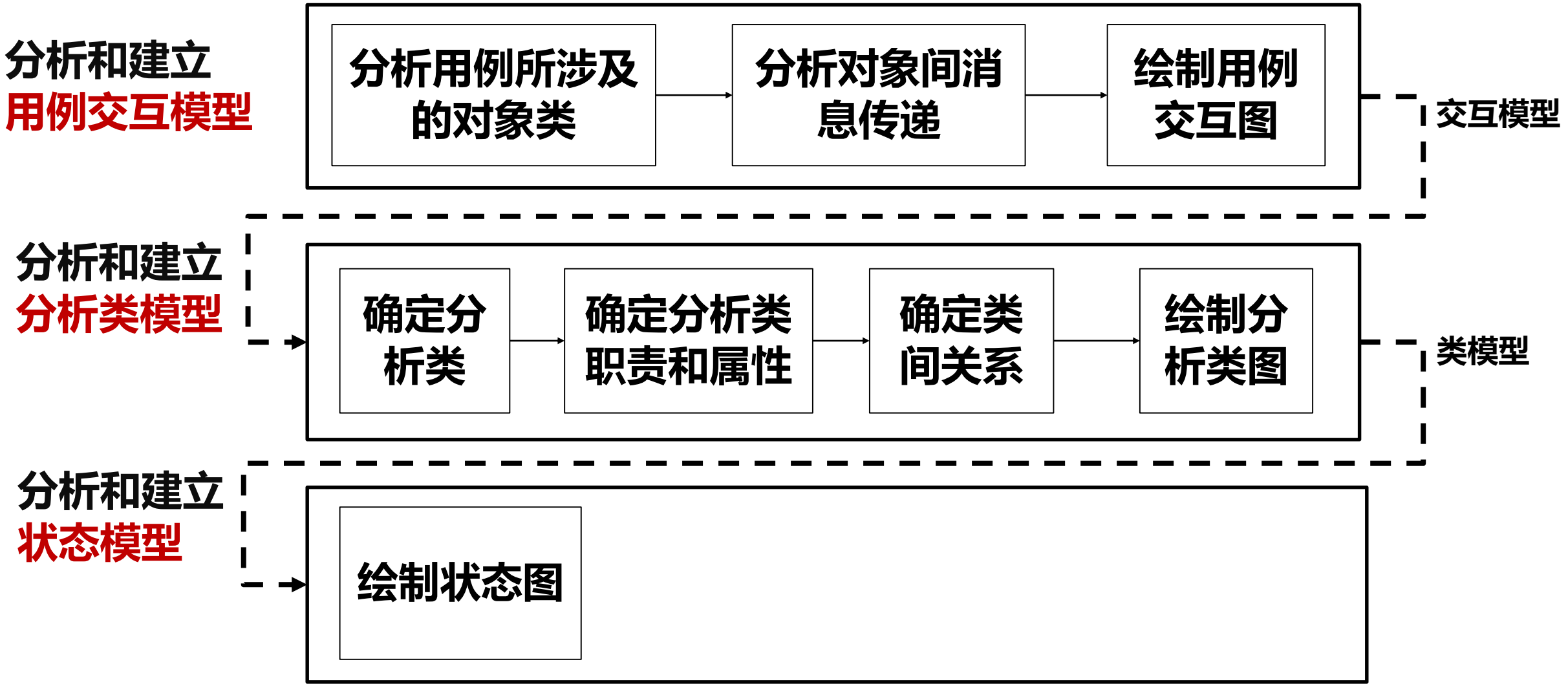
| 用例名称 | 用例标识 | 重要性 | 优先级 |
| --- | --- | --- | --- |
| 监视老人 | UC-MonitorElder | 核心 | 高 |
| 获取老人信息 | UC-GetElderInfo | 核心 | 高 |
| 检测异常状况 | UC-CheckEmergency | 核心 | 高 |
| 通知异常状况 | UC-NotifyEmergency | 核心 | 高 |
| 自主跟随老人 | UC-FollowElder | 核心 | 高 |
| 视频/语音交互 | UC- A&VInteraction | 外围 | 中 |
| 控制机器人 | UC-ControlRobot | 外围 | 低 |
| 提醒服务 | UC-AlertService | 外围 | 低 |
| 用户登录 | UC-UserLogin | 外围 | 低 |
| 系统设置 | UC-SetSystem | 外围 | 低 |

# 2.1.3 确定用例分析和实现的次序

- 根据**迭代计划**中迭代的持续时间、人力资源等情况，结合软件需求的**优先级**、工作量等因素，确定用例分析和实现的先后次序

| 用例名称 | 用例标识 | 优先级 | 迭代次序 |
|---|---|---|---|
| 监视老人 | UC-MonitorElder | 高 | 第一次迭代 |
| 获取老人信息 | UC-GetElderInfo | 高 | |
| 检测异常状况 | UC-CheckEmergency | 高 | |
| 通知异常状况 | UC-NotifyEmergency | 高 | 第二次迭代 |
| 自主跟随老人 | UC-FollowElder | 高 | |
| 视频/语音交互 | UC- A&VInteraction | 中 | 第三次迭代 |
| 控制机器人 | UC-ControlRobot | 低 | |
| 提醒服务 | UC-AlertService | 低 | 第四次迭代 |
| 用户登录 | UC-UserLogin | 低 | |
| 系统设置 | UC-SetSystem | 低 | |

# 2.2 建立软件需求模型的步骤



# 2.2.1 建立用例交互模型

## ❑任务

✓分析用例是如何通过一组对象之间的**交互**来完成功能目标的

## ❑步骤

① 分析用例所涉及的对象及类

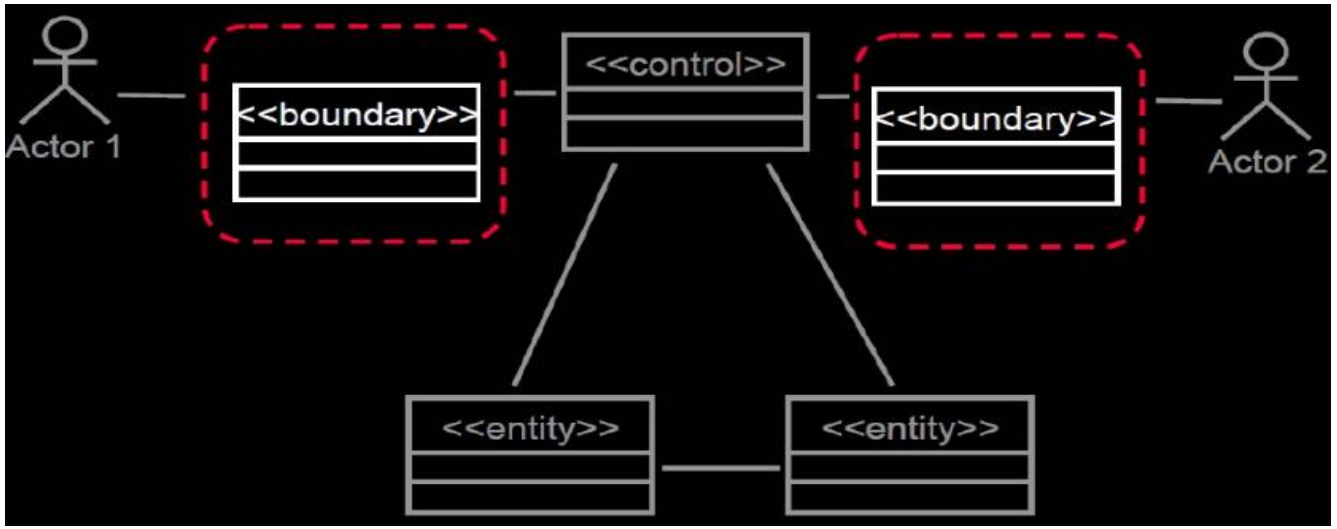
② 分析对象之间的消息传递

③ 绘制用例的交互图

# ① 分析用例所涉及的对象及类

- **用例描述中主要涉及三种类对象**

  - ✓边界类
  - ✓控制类
  - ✓实体类




- 这些类是在用例分析阶段产生的，通常称为分析类

# 边界类

## ❑何为边界类

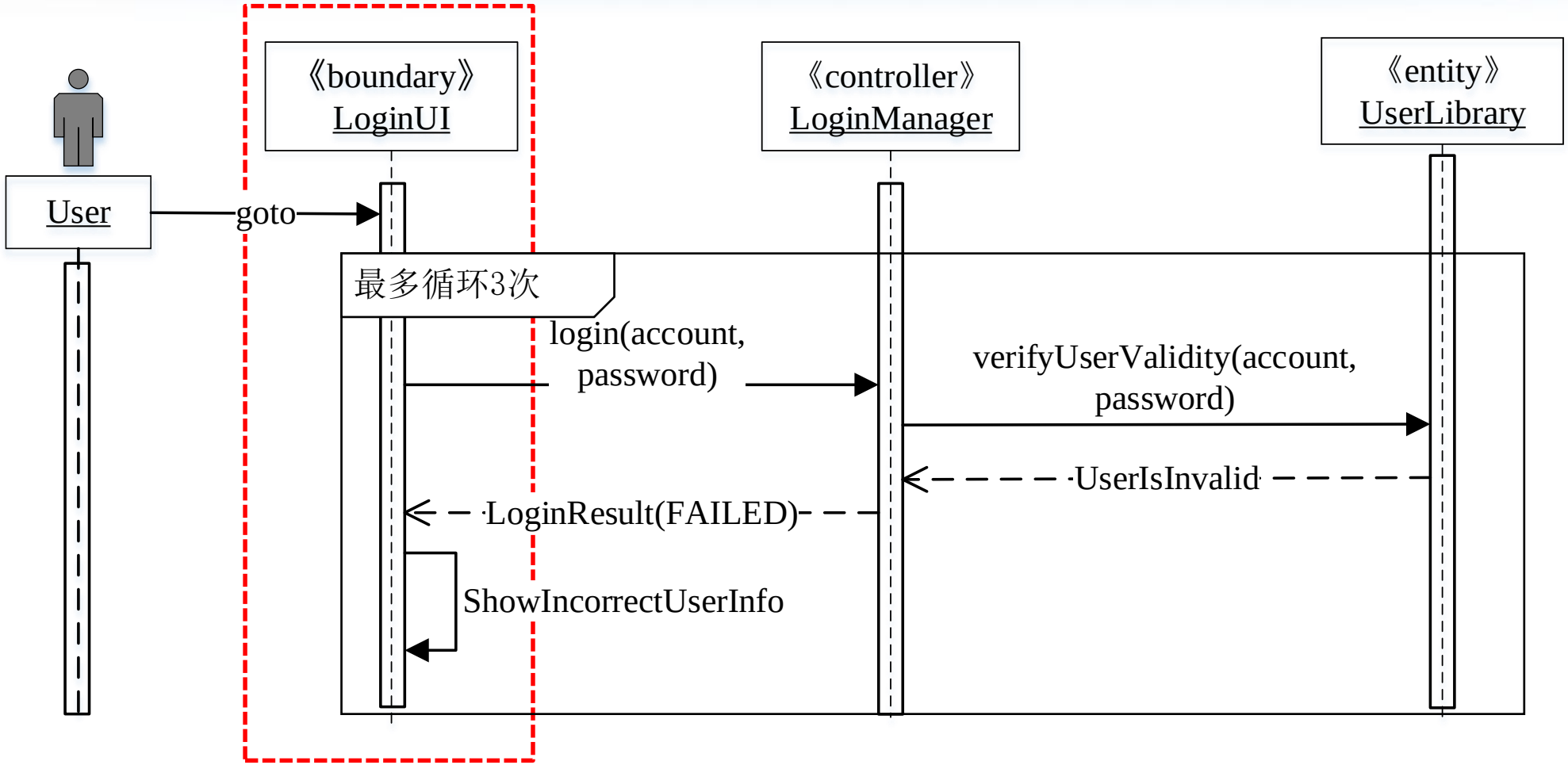
✓处于系统的边界，与系统外的执行者进行交互的类，称为边界类

## ❑边界类对象作用

✓人机交互，接收用户输入，或者向用户输出，如UI界面类。

✓外部接口，如果执行者为外部系统或设备，那么边界类对象需要与它们进行信息交互， 如银行支付网关。

# 示例：边界类



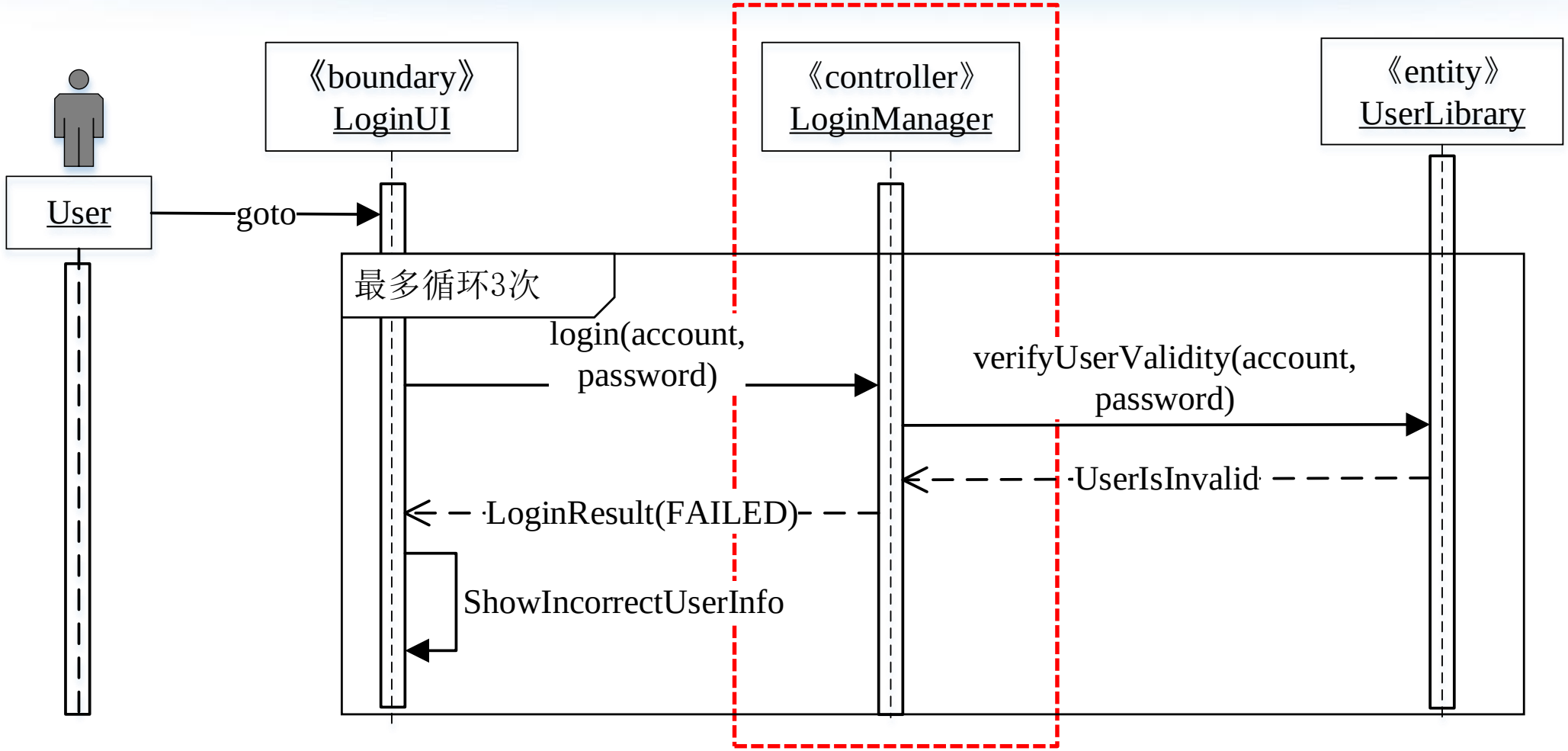

**边界类在顺序图中有何特点?**

# 控制类

- **控制类对象作为用例任务完成的协调者**
  - ✓接收、校验、解析边界类对象发来的请求
  - ✓对任务进行适当的分解
  - ✓调度系统中的其他类对象，以协同完成规定任务

- **控制类可能在某些业务中是客观存在的，也可能是工程师人为引入的**

# 示例：控制类



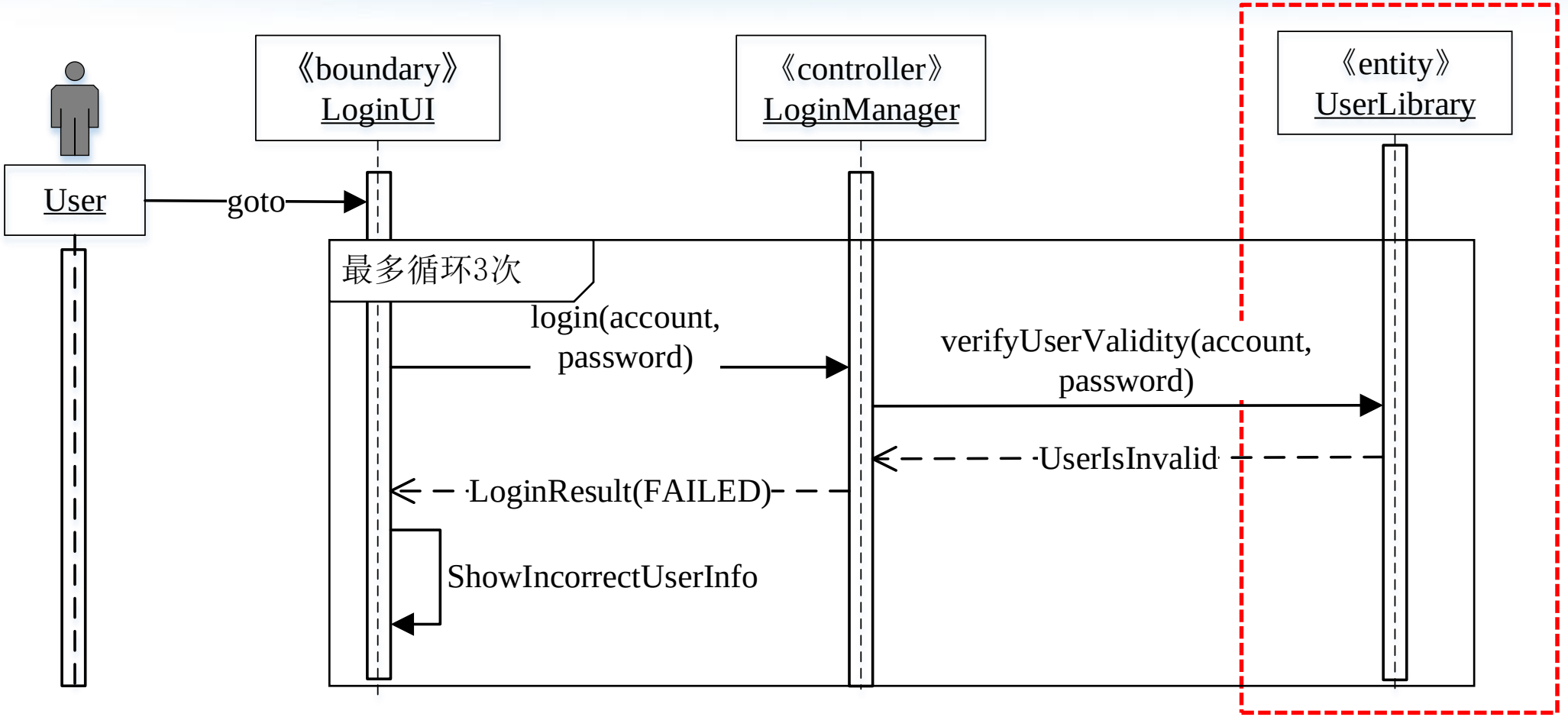

**控制类在顺序图中有何特点?**

# 实体类

- 实体类源于需求中的每个实体映射，其主要职责就是要实现用例中的具体功能，提供相应的业务服务。
  - ✓实体类中既包含实体的持久信息（属性），也包括对这些属性进行操作的方法（如查询、修改、保存等）。

- 发现实体类的两种视角
  - ✓对现实世界中对象的抽象
  - ✓基于消息中参数所归属的对象：谁拥有数据，谁就拥有对数据的处理“职责”

# 示例：实体类




**实体类在顺序图中有何特点?**

# ② 分析和确定对象之间的消息传递

- 确定消息的名称
- 确定消息传递的参数

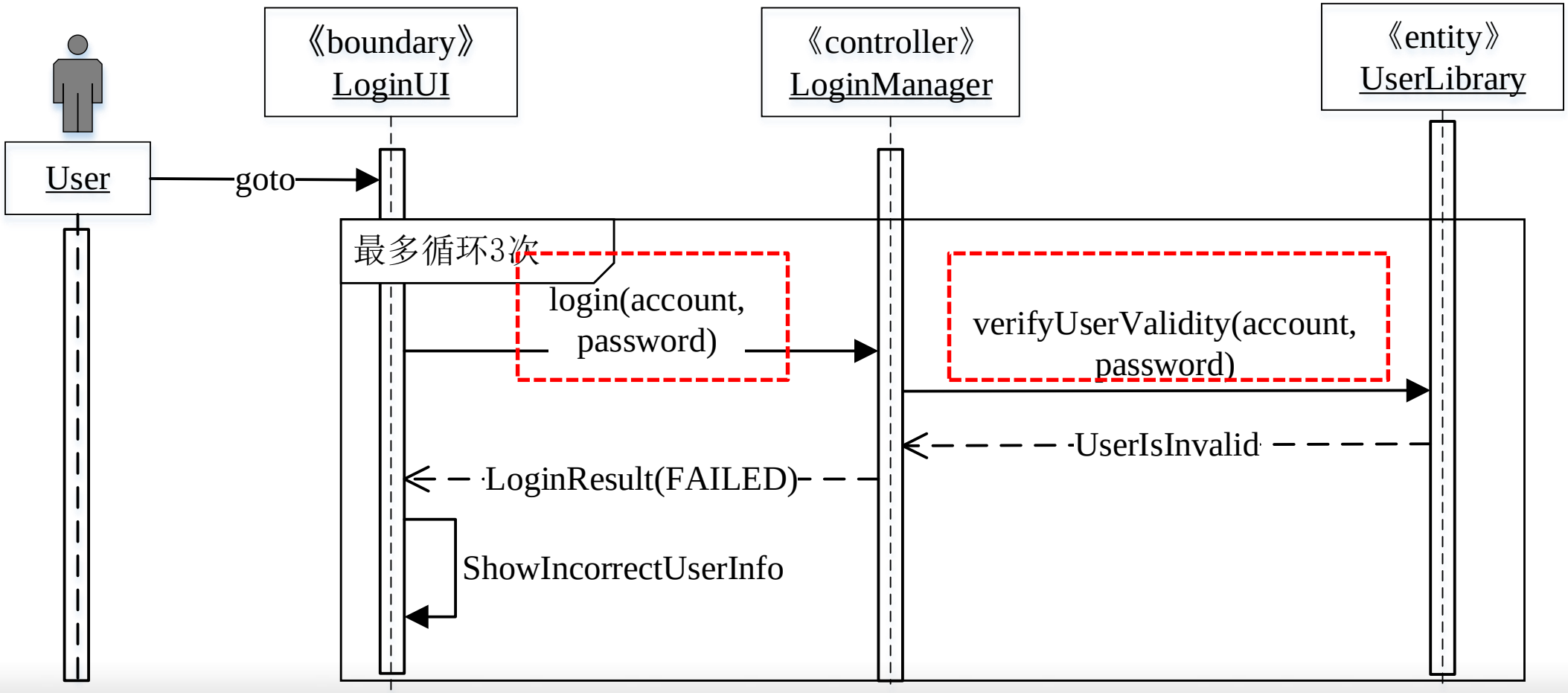
# 确定消息的名称

## ❑消息的名称

- ✓反映了对象间交互的意图，也体现了**接收方**对象所对应的类需**承担的职责和任务**
- ✓消息名称多用动名词表示

# 确定消息传递的信息

- **即确定消息传递的参数**
  - ✓通常，消息参数用名词或名词短语来表示



# ③ 绘制用例的交互图

## ❑绘制顺序图的策略

- ✓**从左到右**依次布局：外部执行者、UI边界类、控制类、实体类对象、作为外部系统的边界类对象（如有）
- ✓消息传递采用**自上而下的布局方式**，以反映消息交互的时序先后

## ❑一个用例至少构造一个交互图，以刻画用例的行为模型

- ✓对于较为简单的用例，构造一个交互图就足够了
- ✓对于较复杂的用例而言，需要绘制多张交互图，每张交互图刻画了用例在某种特定场景下的交互动作序列

# 顺序图的执行流程

① 外部执行者与边界类对象进行交互以启动用例的执行

② 边界类对象接收外部执行者提供的信息，将信息转换为内部形式，并将相关信息发送给控制类对象

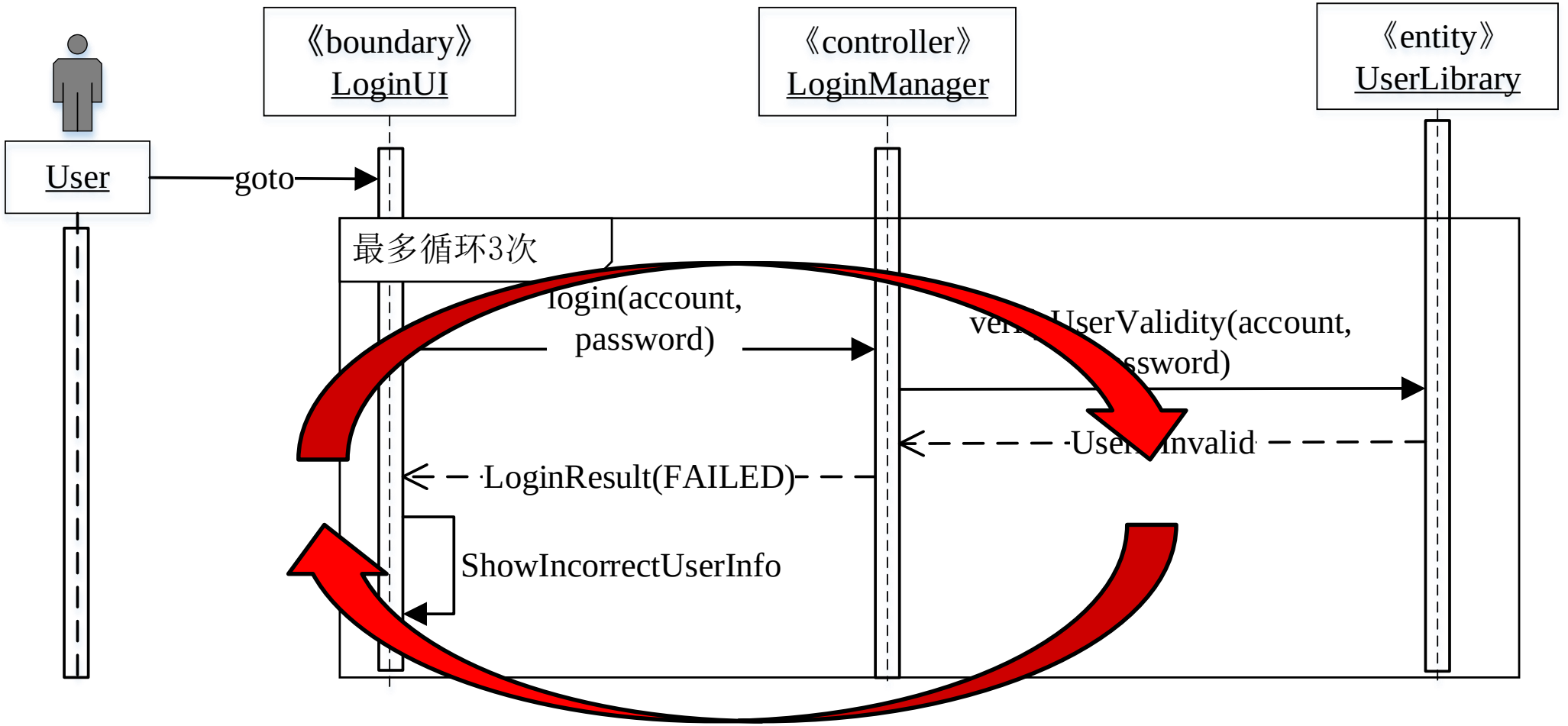
③ 控制类对象根据业务逻辑处理流程，产生和分解任务，与相关的实体类对象进行交互以请求完成相关的任务

④ 实体类对象实施相关的行为后，向控制类对象反馈信息处理结果

⑤ 控制类对象处理接收到的信息，将处理结果通知边界类对象

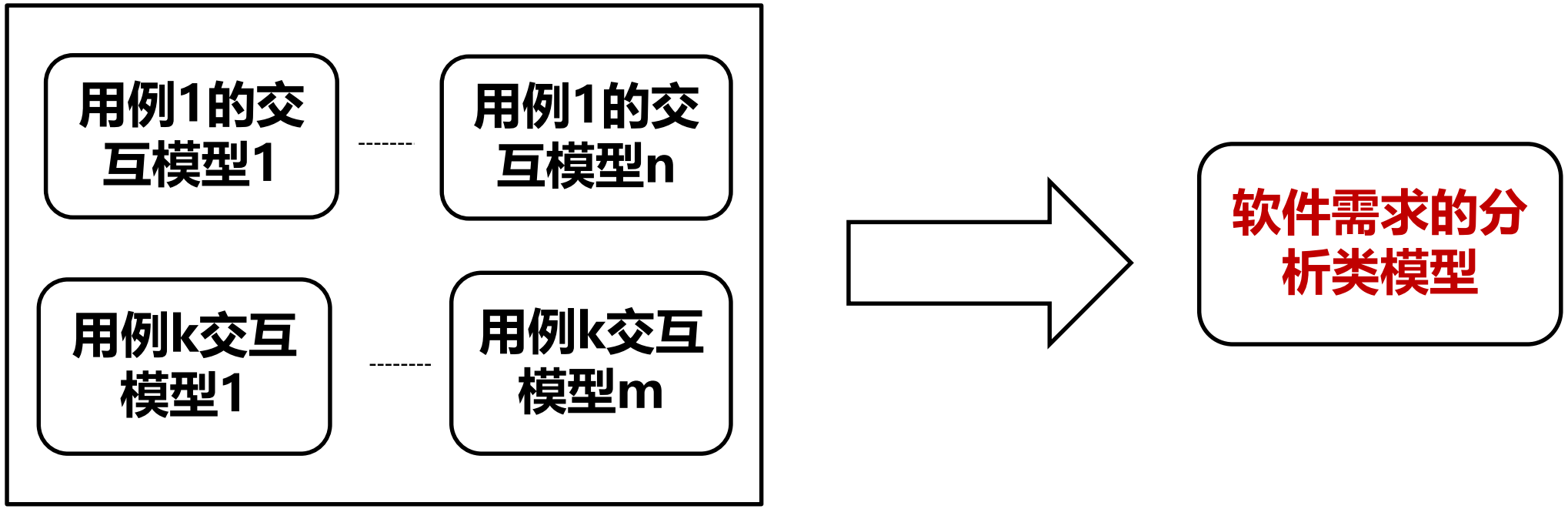
⑥ 边界类对象通过界面将处理结果展示给外部执行者

# 示例：顺序图的工作流程




**顺序图的工作流程有何特点?**

# 2.2.2 建立分析类模型



# 建立分析类模型的步骤

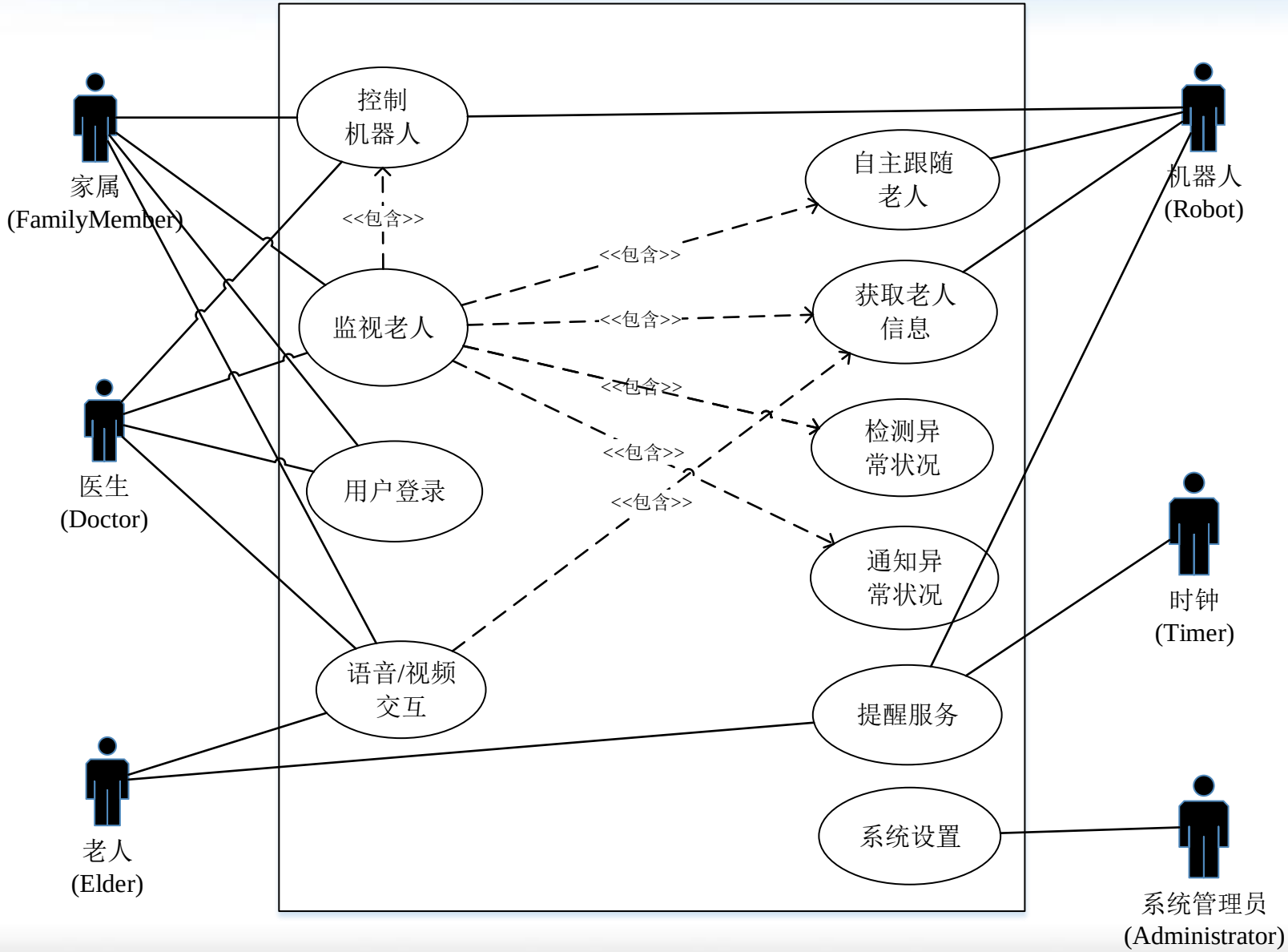
① 确定分析类
② 确定分析类的职责
③ 确定分析类的属性
④ 确定分析类之间的关系
⑤ 绘制分析类图

# ① 确定分析类

- **如何确定分析类?**
  - ✓用例模型中的**外部执行者**应该是分析类图中的类
  - ✓各个用例的顺序图中出现了某个**对象**，那么该**对象所对应的类**也属于分析类

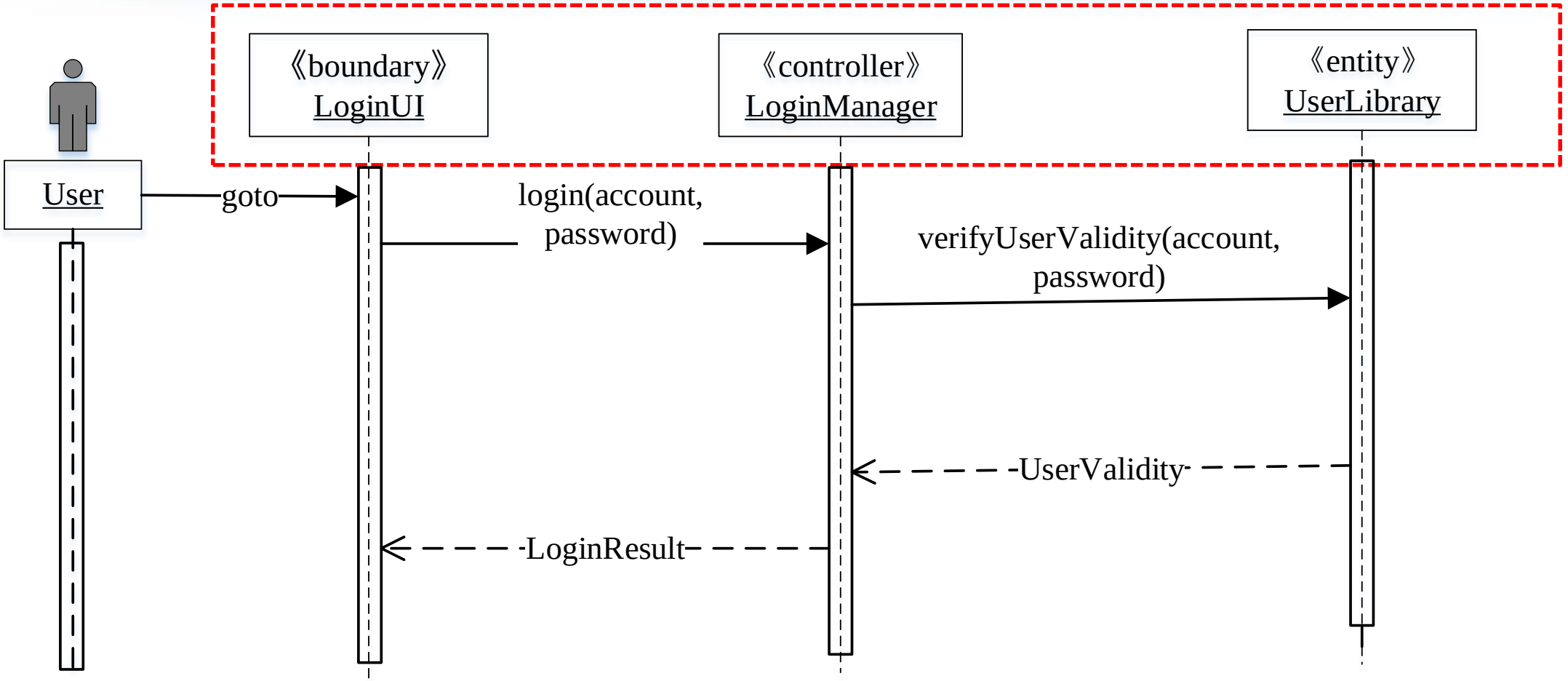
- 注意：“分析类”≠“代码实现中的类”，而是对应业务领域中的概念，或者说是实现类的“原型”

# 示例：根据用例图来确定分析类




**有哪些分析类?**

# 示例：根据交互图来确定分析类

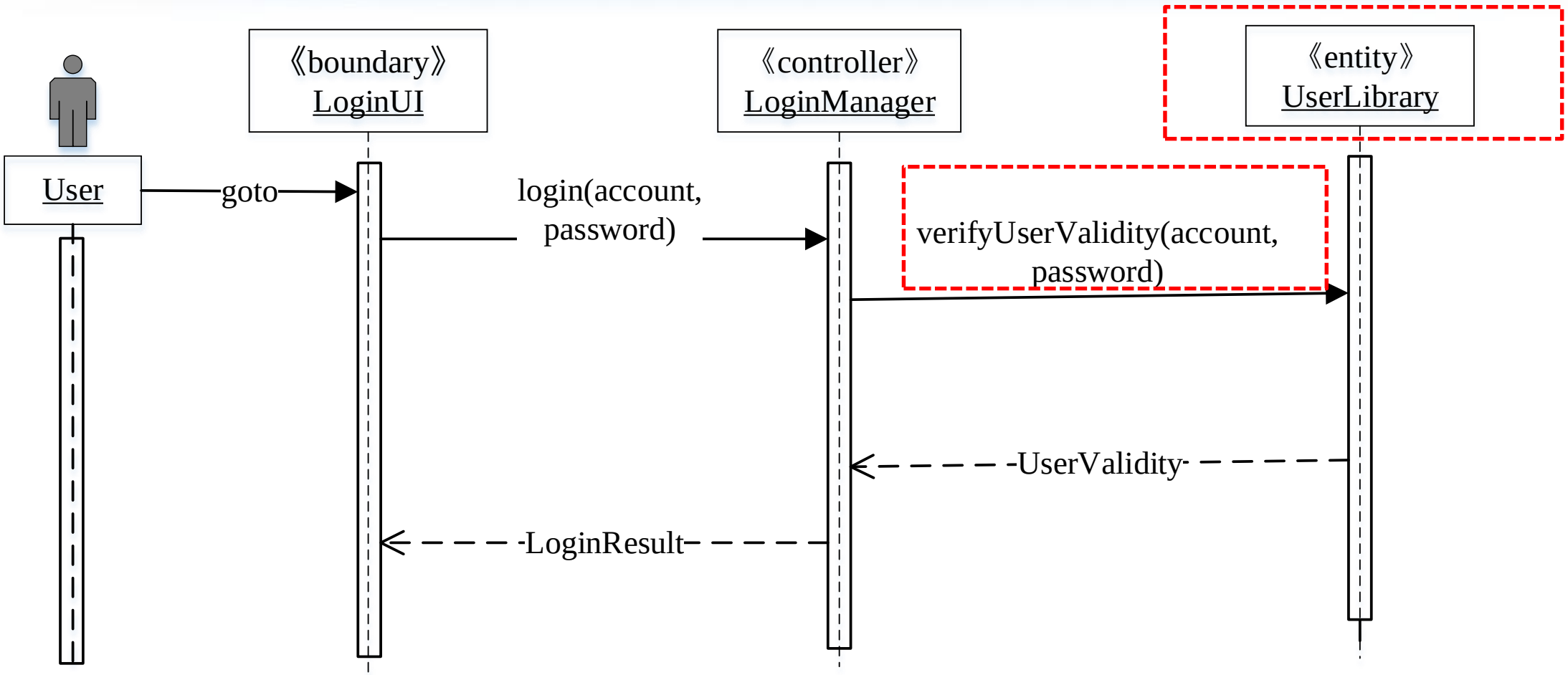



**有哪些分析类?**

# ② 确定分析类的职责

- 每一个分析类都有其职责，需提供相关的服务
- 对象接收的消息与其承担的职责之间存在一一对应关系，即:如果一个对象能够接收某项消息，它就应当承担与该消息相对应的职责
- 可用类的**方法名**来表示分析类的职责

# 示例：确定分析类的职责



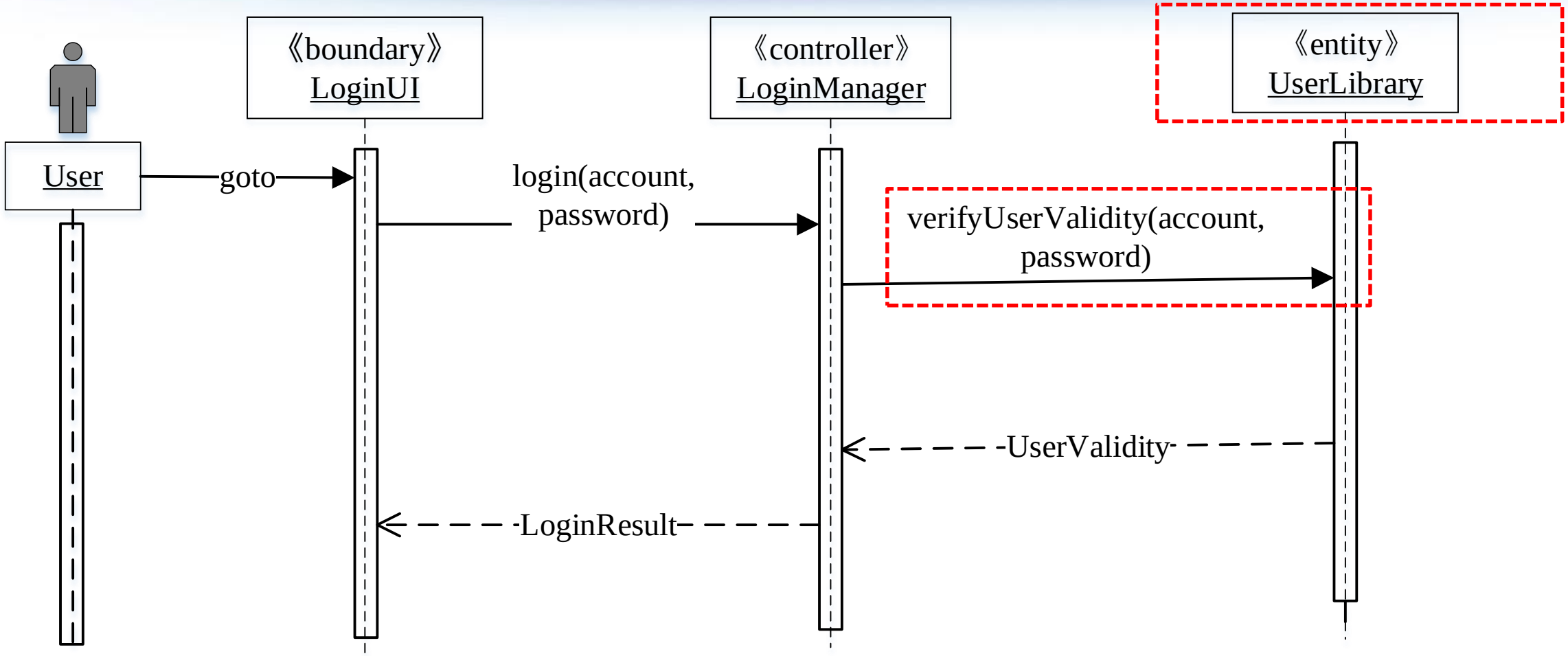

**UserLibrary类有何职责?**

# ③ 确定分析类的属性

- **分析类具有哪些属性取决于该类需要持久保存哪些信息**

- 如果顺序图中某类发送和接收的消息中附带参数，这意味着该类需要与此相对应的属性

# 示例：确定分析类的属性




**UserLibrary类有何属性?**

# ④ 确定分析类之间的关系

## ❑类之间的关系有多种形式

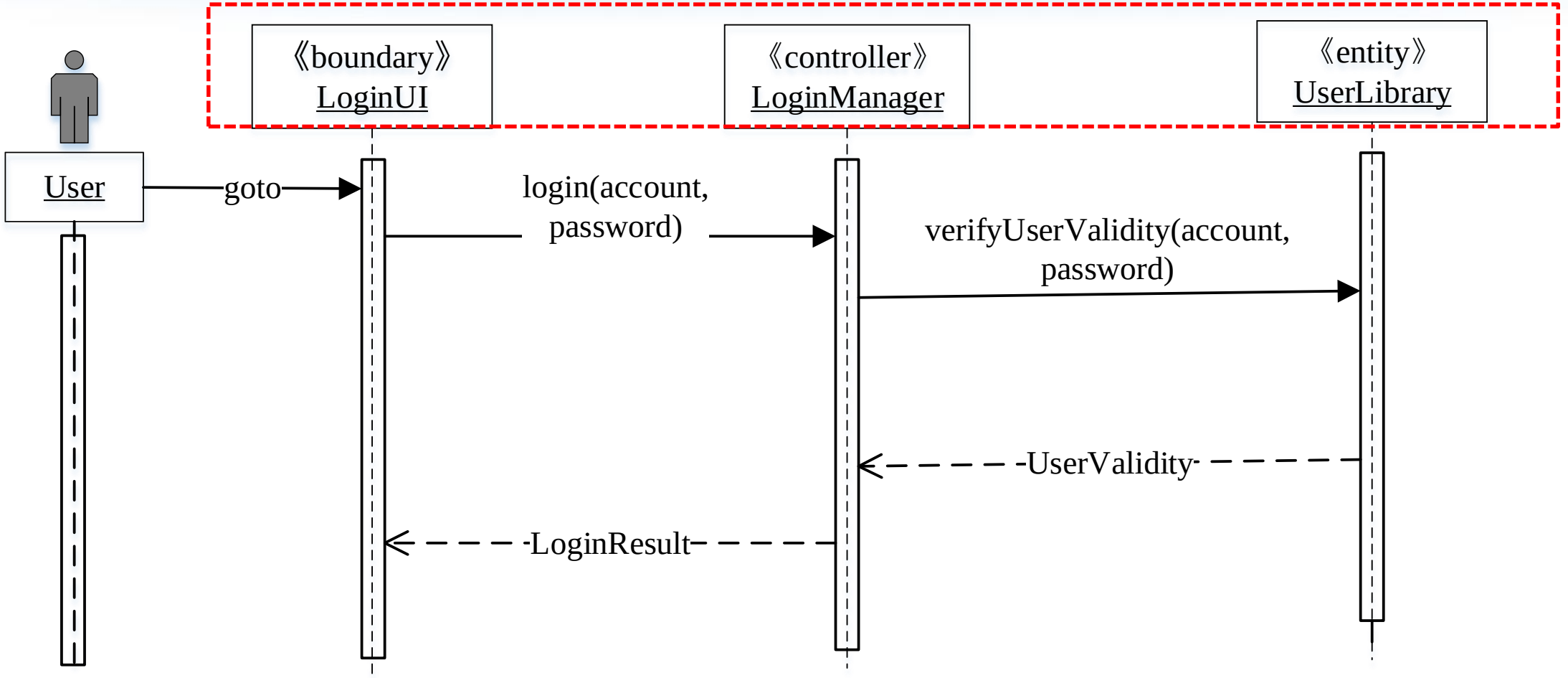
✓包括继承、关联、聚合和组合、依赖等

## ❑具体策略

✓顺序图中，如果存在从类A对象到类B对象的消息传递，那么意味着类A和B间存在**关联、依赖、聚合或组合**等关系

✓如果类之间存在一般和特殊的关系，那么可对这些分析类进行层次化组织，标识出它们间的**继承关系**

# 示例：确定分析类之间的关系

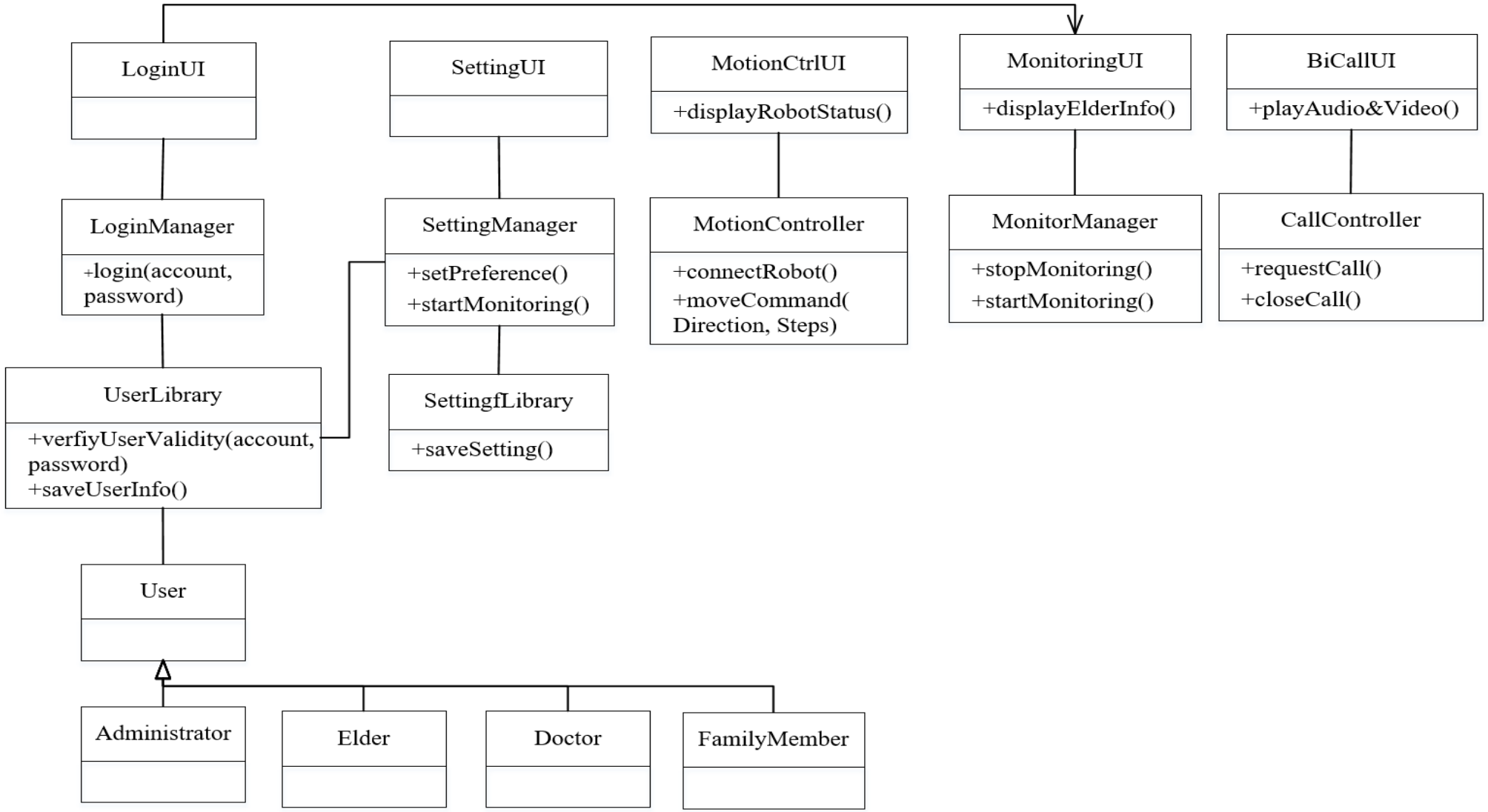



**哪些分析类之间存在怎样的关系?**

# ⑤ 绘制分析类图

- 建立分析类模型
- 如果系列规模较大，分析类的数量多，那么可以分多个子系统（包图）来绘制分析类图

# 示例：“空巢老人看护软件”分析类图



# 2.2.3 建立状态模型

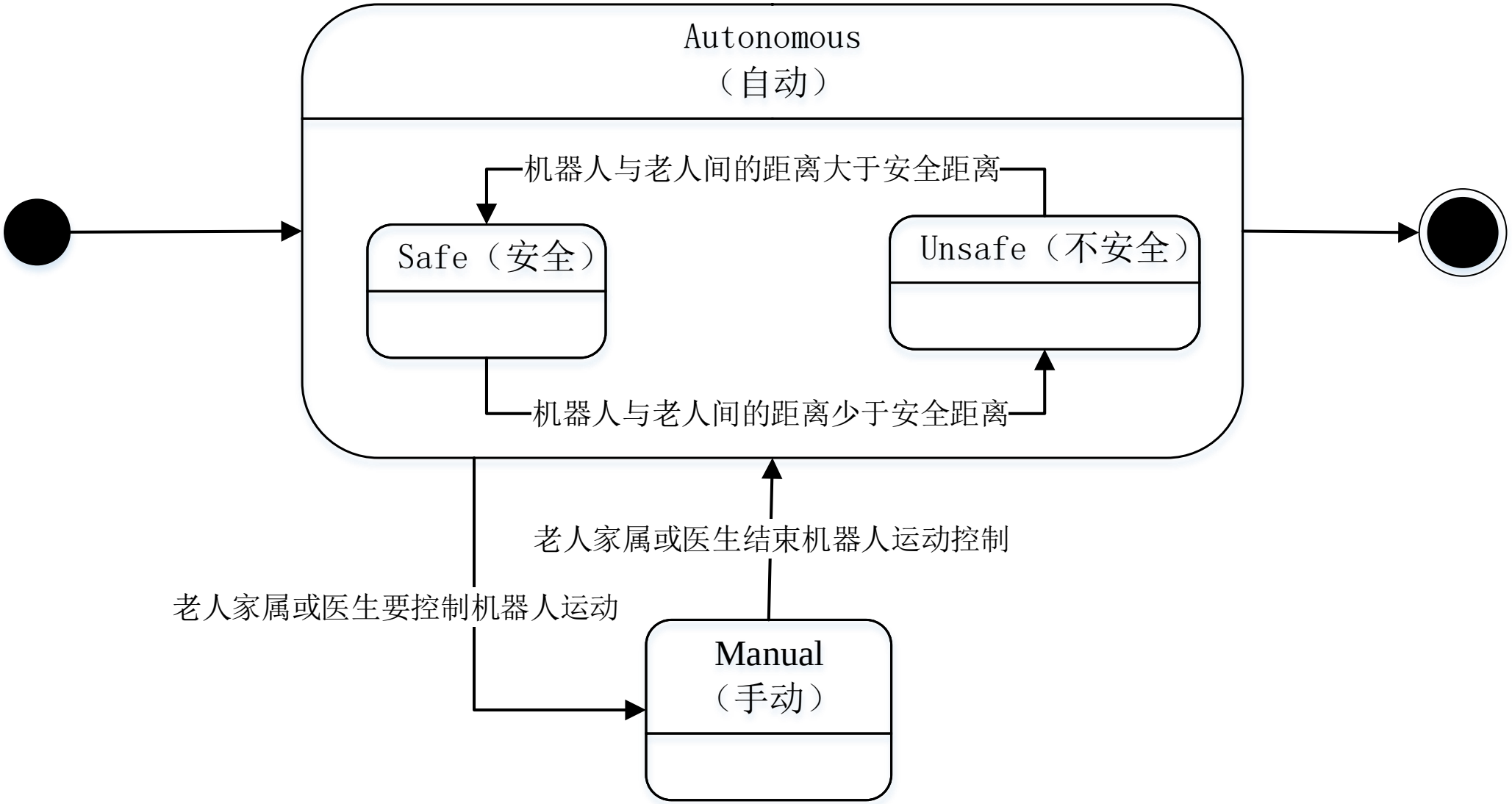
## ❑ 注意二点

- ✓状态模型是针对**对象**而言的，而非针对分析类
- ✓**无需为所有**的类对象建立状态模型，只需针对那些具有复杂状态的对象建立状态模型

## ❑ 状态图建模的两种场景

- ✓需要明晰事件顺序时：状态图能够描述状态的转移顺序，从而帮助工程师避免出现事件错序的情况，如订单必须先通过付款事件进入“已付款”状态，才能通过发货事件进入“已发货”状态。
- ✓定义依赖于状态的行为时：有些类对象其所处的状态不同，行为也会不同，这时需要通过状态图明确行为与状态的关系，如账户正常和冻结状态下，取款的行为是不同的。

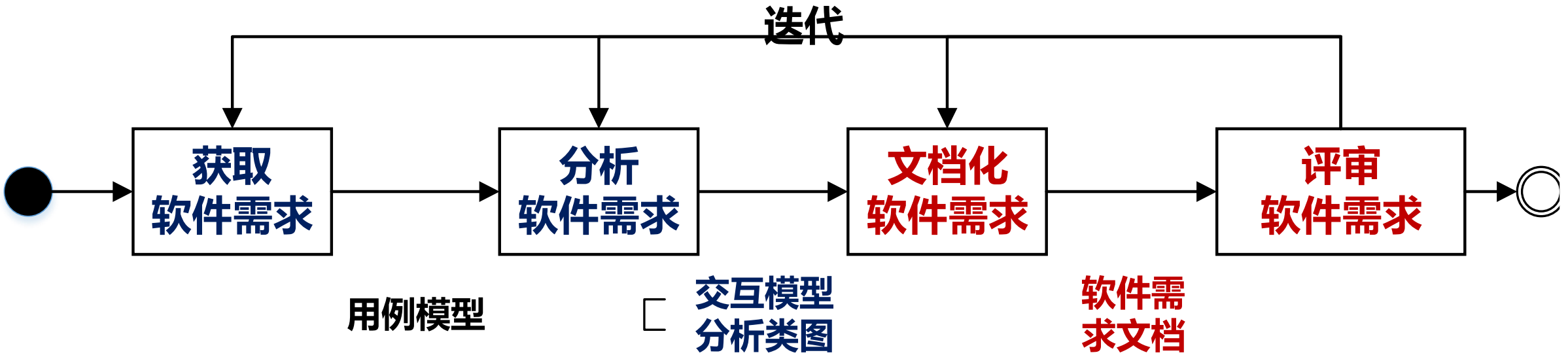
# 示例: 分析机器人类对象的状态图



# 内容

1. 需求分析概述
   - ✓需求分析任务
   - ✓UML需求分析模型
2. 需求分析过程
   - ✓确定需求优先级
   - ✓建立需求分析模型
3. **需求文档化及评审**
   - ✓软件需求规格说明书
   - ✓评审软件需求

# 3.1 编写软件需求文档



# 3.1 软件需求文档模板

1. 引言
    - 1.1 编写目标
    - 1.2 读者对象
    - 1.3 文档概述
    - 1.4 术语定义
    - 1.5 参考文献
2. 软件系统概述
    - 2.1 软件产品概述
    - 2.2 用户特征
    - 2.3 设计和实现约束
    - 2.4 假设与依赖

3. 功能性需求描述
    - 3.1 软件功能概述
    - 3.2 软件需求的用例模型
    - 3.3 软件需求的分析模型

4. 非功能性需求
5. 界面需求
6. 接口定义
7. 进度要求
8. 交付要求
9. 何种形式来交付
10. 验收要求

# 3.2 软件需求分析的输出

## ❑软件原型

- ✓以可运行软件的形式，直观地展示了软件的业务工作流程、操作界面、用户的输入和输出等方面的功能性需求信息

## ❑软件需求模型

- ✓以可视化的图形方式，从多个不同的视角描述软件的功能性需求，包括用例模型、用例的交互模型、分析类模型、状态模型等

## ❑软件需求文档

- ✓以图文并茂的方式，结合需求模型以及需求的自然语言描述，详尽刻画了软件需求，包括功能性和非功能性软件需求，软件需求的优先级列表等

# 3.3 软件需求评审步骤

- 阅读和汇报软件需求制品（评审会议）
- 收集和整理问题
- 讨论和达成一致
- 纳入配置

为什么要评审软件需求?

# 评审软件需求-内容（1/2）

- 内容完整性，是否包含了所有软件需求
- 内容正确性，是否客观、正确地反映了用户的实际要求
- 内容准确性，是否存在描述不清或存在二义性的表述
  - 系统能提供适当的文档浏览器供用户在线阅读各类文档
- 内容一致性，不同需求之间是否存在不一致情况
  - 需求A：系统应该实时显示最新的订单状态。
  - 需求B：系统在更新订单状态前必须进行人工验证。
- 内容多余性，是否存在不必要的软件需求
- 内容可追踪性，每一项软件需求是否可追踪的

# 评审软件需求-形式（2/2）

- 文档规范性，软件需求规格说明书书写是否遵循文档规范
- 图符规范性，软件需求模型是否正确地使用了UML的图符
- 表述可读性，软件需求文档文字表述是否简洁、可读性好
- 图表一致性，软件需求制品中的图表引用是否正确

# 3.4 如何解决软件需求问题（1/2）

- **遗漏的软件需求**
  - ✓再次征求用户、客户、领域专家等意见，以补充遗漏的软件需求
- **无源头的软件需求**
  - ✓剔除或暂时不用考虑该部分的软件需求，或者将这些软件需求置于低优先级
- **不一致、相冲突的软件需求**
  - ✓寻找到具有更高级别的用户或客户，由他们来最终确定软件需求
- **不正确和和不准确的软件需求**
  - ✓与用户、客户、领域专家等进行深入的沟通，以正确地理解软件需求的内涵

# 如何解决软件需求问题（2/2）

- **不规范的软件需求文档**
  - ✓对照软件需求规范标准和模板，按照其要求来撰写并产生软件需求规格说明书
- **不规范和不正确的软件需求模型**
  - ✓学习UML图符和模型的用法，在此基础上绘制出正确和规范的UML模型
- **费解的软件需求文档**
  - ✓消除冗余的文字表述，提高语言表达的简洁性，系统梳理和组织软件需求文档的格式，以提高软件需求文档的可读性

# 思考和讨论

- 如果软件需求模型和文档存在诸多的问题，没有得到有效的解决，会产生什么样的后果?

# 小结

- **分析软件是要精化和深化软件需求**
  - ✓基于初步软件需求，循序渐进
  - ✓确保软件需求的完整性、一致性和准确性
- **UML提供了一系列用于描述软件需求的图**
  - ✓用例图、交互图、分析类图、状态图
- **分析软件需求的步骤**
  - ✓分析和确立软件需求优先级、分析和建立软件需求模型、撰写软件需求规格说明书、评审软件需求模型和文档
- **分析软件需求的输出和评审**
  - ✓软件需求模型、软件需求原型、软件需求文档
  - ✓评审软件需求制品

# 综合实践一

- **任务：分析开源软件的新需求**
- **方法**
  - ✓借助UML进行需求建模，遵循软件需求规格说明书的标准或模板来撰写开源软件新增软件需求的文档
- **要求**
  - ✓建立新增软件需求的用例交互模型、分析类模型和必要的状态模型，按照软件需求规格说明书的规范标准，撰写相应的软件需求文档
- **结果：开源软件新增需求的用例交互图、分析类图、状态图以及软件需求规格说明书**

# 综合实践二

- **任务：精化和分析软件需求**
- **方法**
  - ✓精化和细化软件需求，采用UML的交互图、类图、状态图等，对精化的软件需求进行描述和建模，建立软件需求模型；在此基础上，遵循软件需求规格说明书的模板，撰写软件需求文档；要确保软件需求模型和文档的质量，对最终软件需求制品进行评审
- **要求**
  - ✓建立软件需求的用例交互模型、分析类模型和必要的状态模型，按照软件需求规格说明书的规范标准，撰写相应的软件需求文档
- **结果：软件需求的UML模型和软件需求规格说明书**

# 问题和讨论