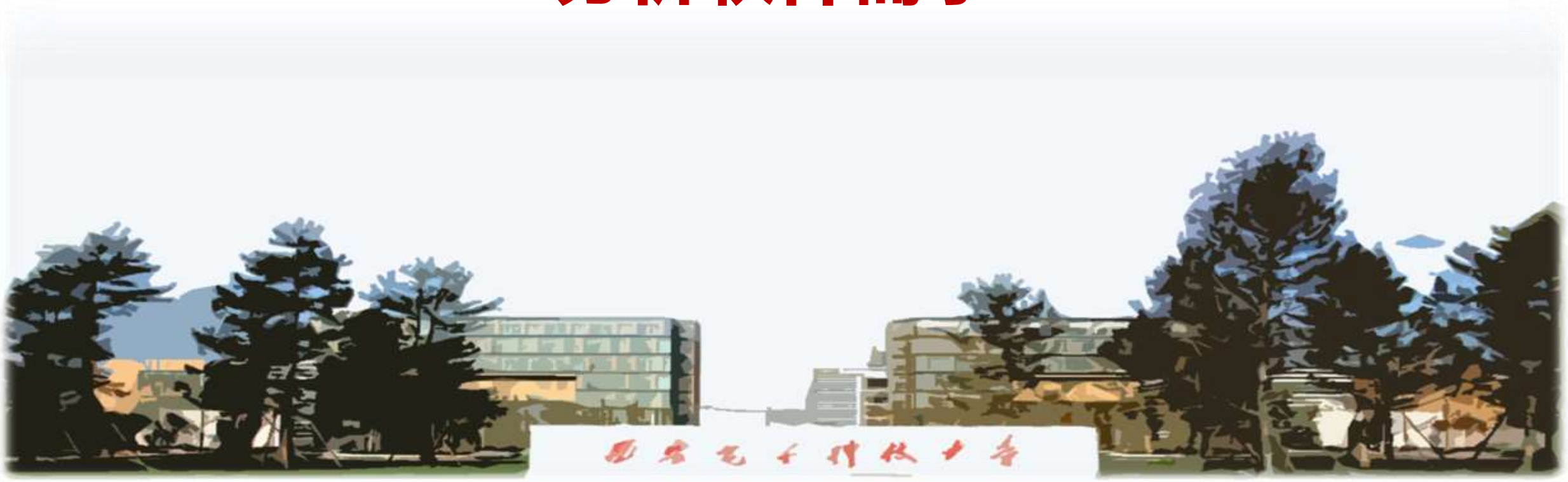
西安电子科技大学
XIDIAN UNIVERSITY

分析软件需求



内容

1. 分析软件需求概述

- ✓分析软件需求的任务
- ✓UML描述方法

2. 分析软件需求过程

- ✓分析和确立软件需求优先级
- ✓分析和建立软件需求模型

3. 软件需求文档化及评审

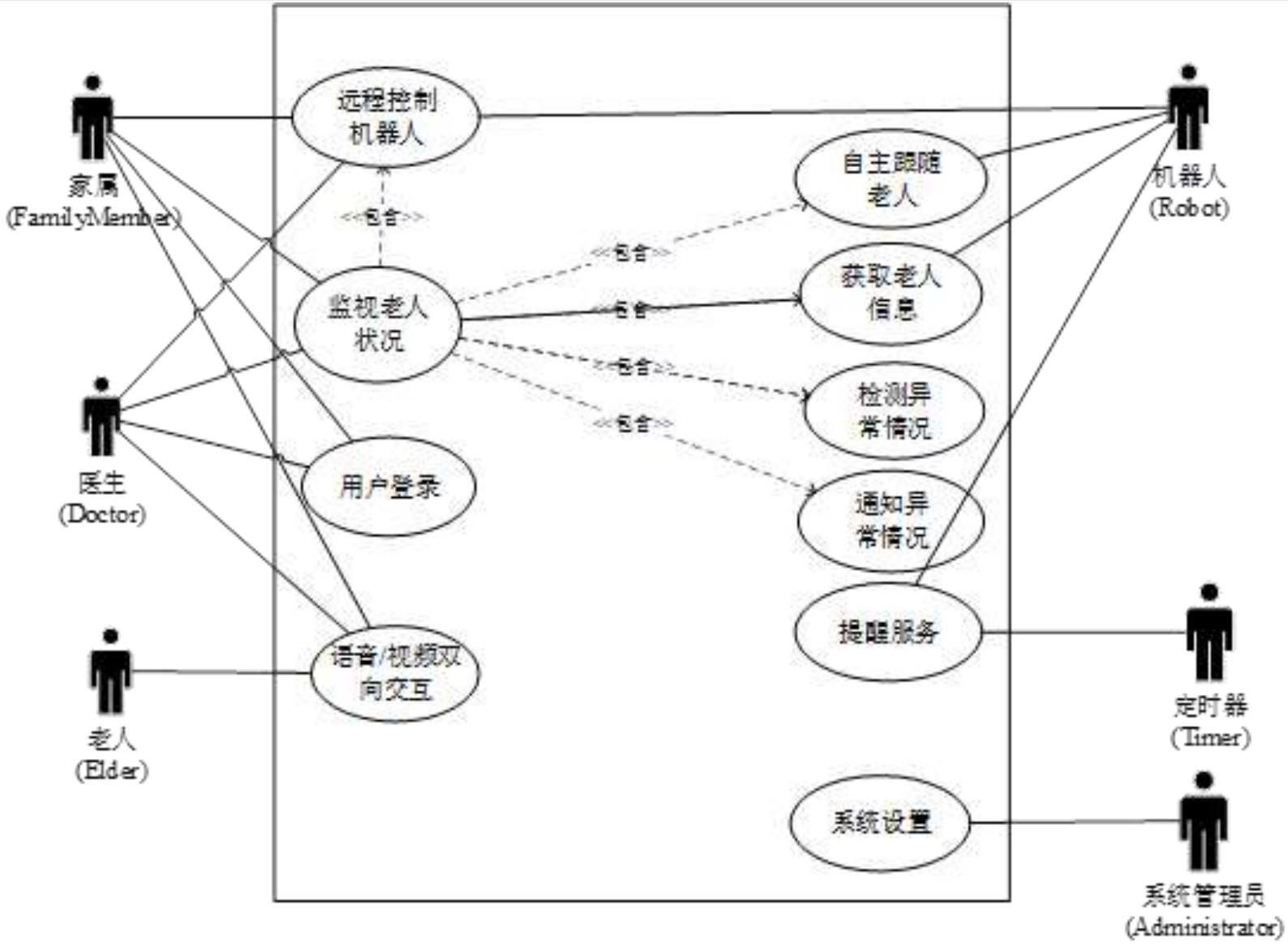
- ✓软件需求规格说明书
- ✓评审软件需求

1.1 为什么要分析软件需求

❑初步软件需求存在的问题

- ✓**不具体不详尽**，没有提供软件需求的细节性内容
- ✓**不清晰**，对软件需求的描述仍然不够清晰、直观和详实
- ✓**关系不明朗**，未深入分析不同软件需求项之间的关系
- ✓**存在潜在缺陷**，潜在的需求缺陷难以被发现
- ✓**没有区分不同软件需求**，有必要鉴别不同软件需求项的重要性差别，区分不同软件需求的开发优先级

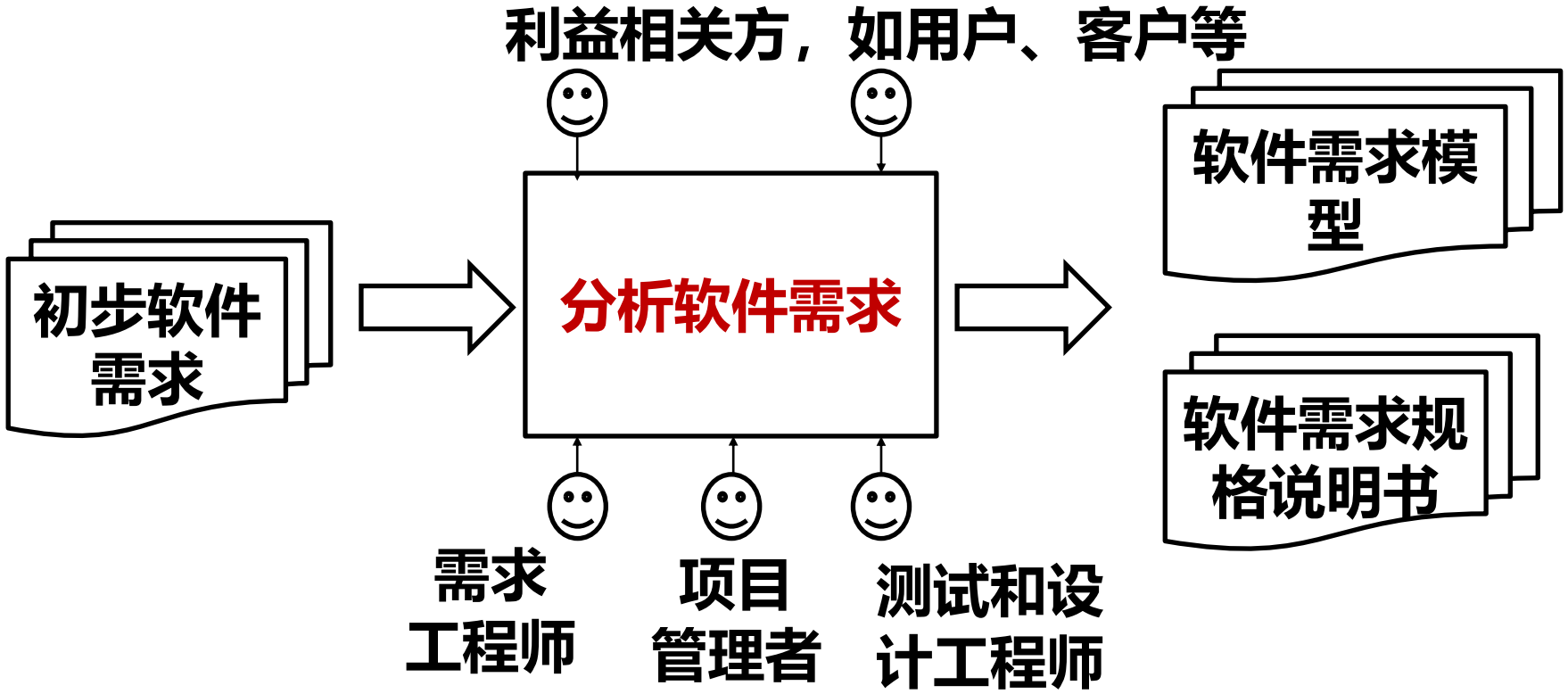
示例：初步软件需求



这张图说清楚软件需求了吗?

1.2 分析软件需求的任务

- 基于初步软件需求，进一步精化和分析软件需求，确定软件需求优先级，建立软件需求模型，发现和解决软件需求缺陷，形成高质量的软件需求模型和软件需求规格说明书



1.3 软件需求的不同视角表示

❑用例视角

- ✓具有哪些**功能**、功能间有何**关系**、功能与利益相关方有何**关系**
- ✓UML提供了**用例图**来分析和描述用例视角的软件需求模型

❑行为视角

- ✓用例是如何通过业务领域中一组对象以及它们间的**交互**来达成的
- ✓UML提供了**交互图、状态图**来描述行为视角的软件需求模型

❑结构视角

- ✓业务领域有哪些重要的领域**概念**以及它们之间具有什么样的**关系**
- ✓UML提供了**类图**来描述和分析业务领域的概念模型

支持需求建模和分析的UML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**

1.3.1 UML交互图的作用

- **刻画对象间的消息传递，分析如何通过交互协作完成功能**
  - ✓用例的功能实现方式
  - ✓软件系统在某种使用场景下对象间的交互协作流程
  - ✓软件系统的某个复杂操作的逻辑实现模型

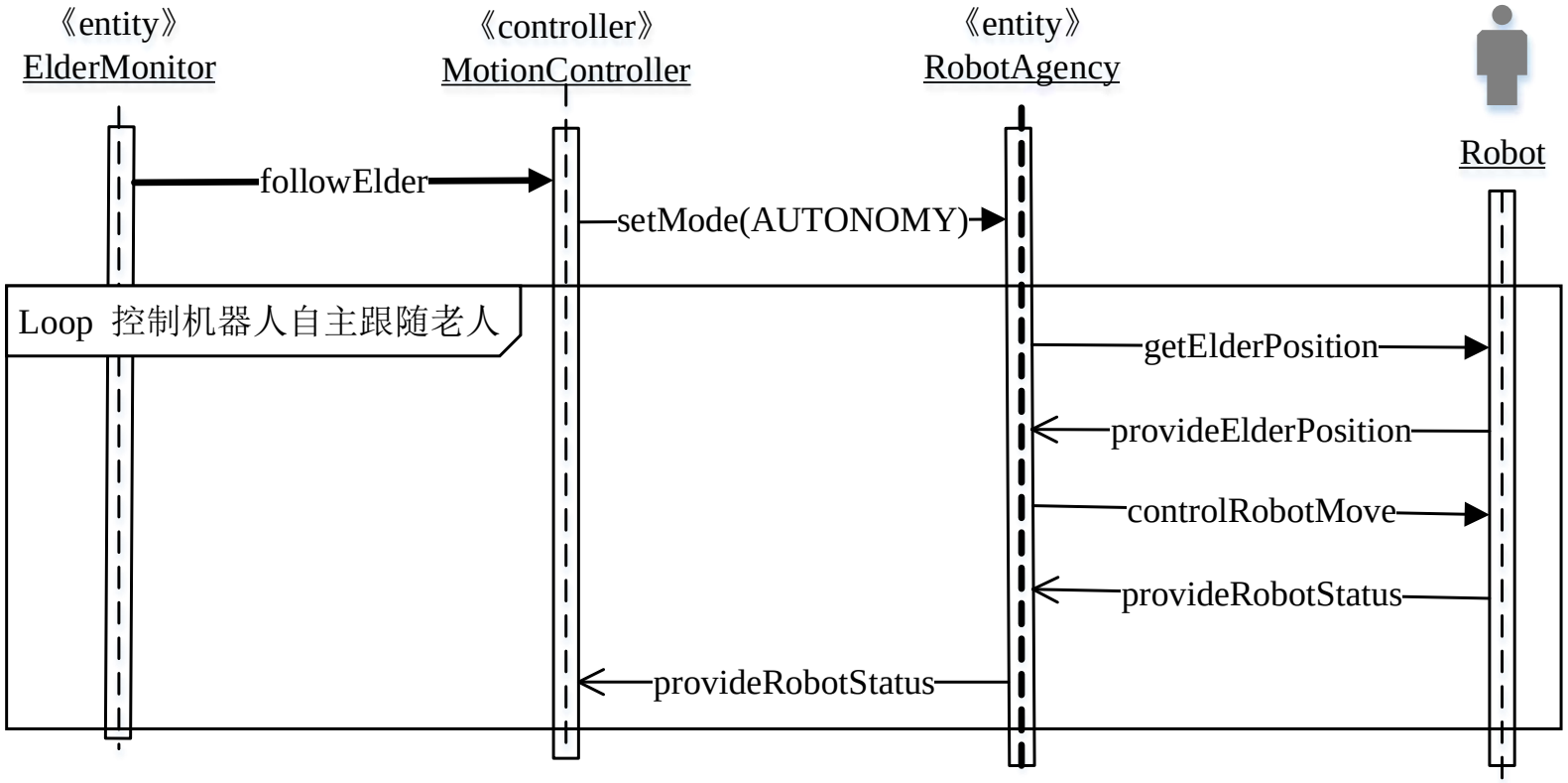
- **二类交互图**
  - ✓顺序图(Sequence Diagram)：强调消息传递的时间序
  - ✓通信图(Communication Diagram)：突出对象间的合作

- **表达能力相同，二类图可相互转换**

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)，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”

❑消息传递

- ✓对象生命线间的**有向边**
- ✓“[*][监护条件] [返回值:=]消息名[(参数表)]”
- ✓“*”为迭代标记表示同一消息对同一类的多个对象发送

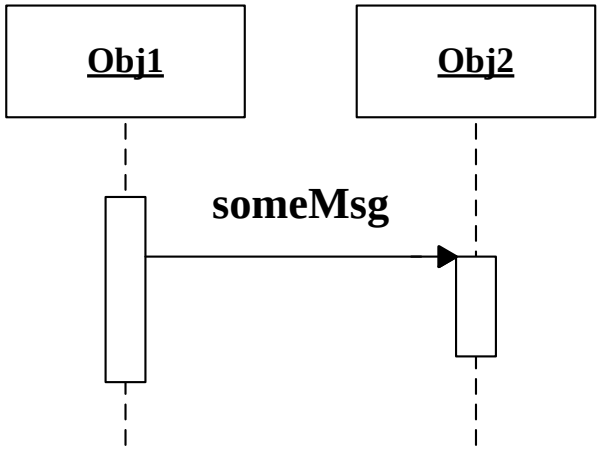
对象间的消息传递

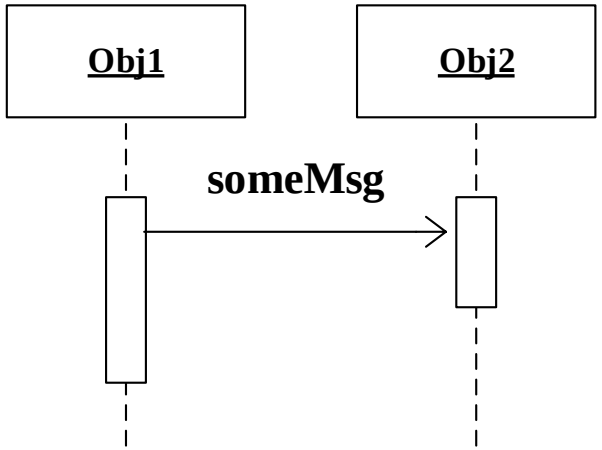
- **同步消息**
  - ✓发送者等待接收者将消息处理完后再继续
- **异步消息**
  - ✓发送者在发送完消息后不等待接收方即继续自己的处理
- **自消息**
  - ✓一个对象发送给自身的消息
- **返回消息**
  - ✓某条消息处理已经完成，处理结果沿返回消息传回
- **创建消息和销毁消息**
  - ✓消息传递目标对象的创建和删除

消息图元的表示

注意箭头的图形表示(线条和箭头形状)

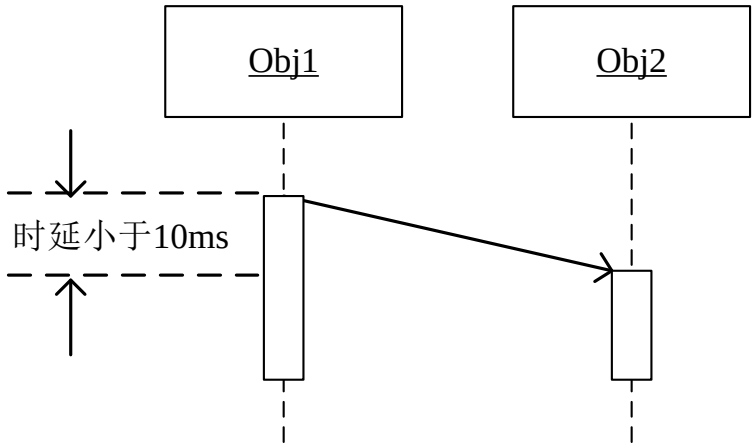
同步消息

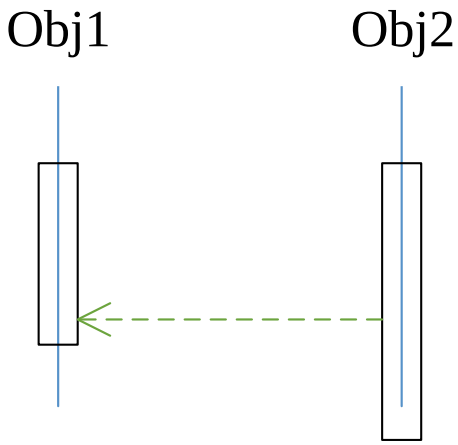




异步消息

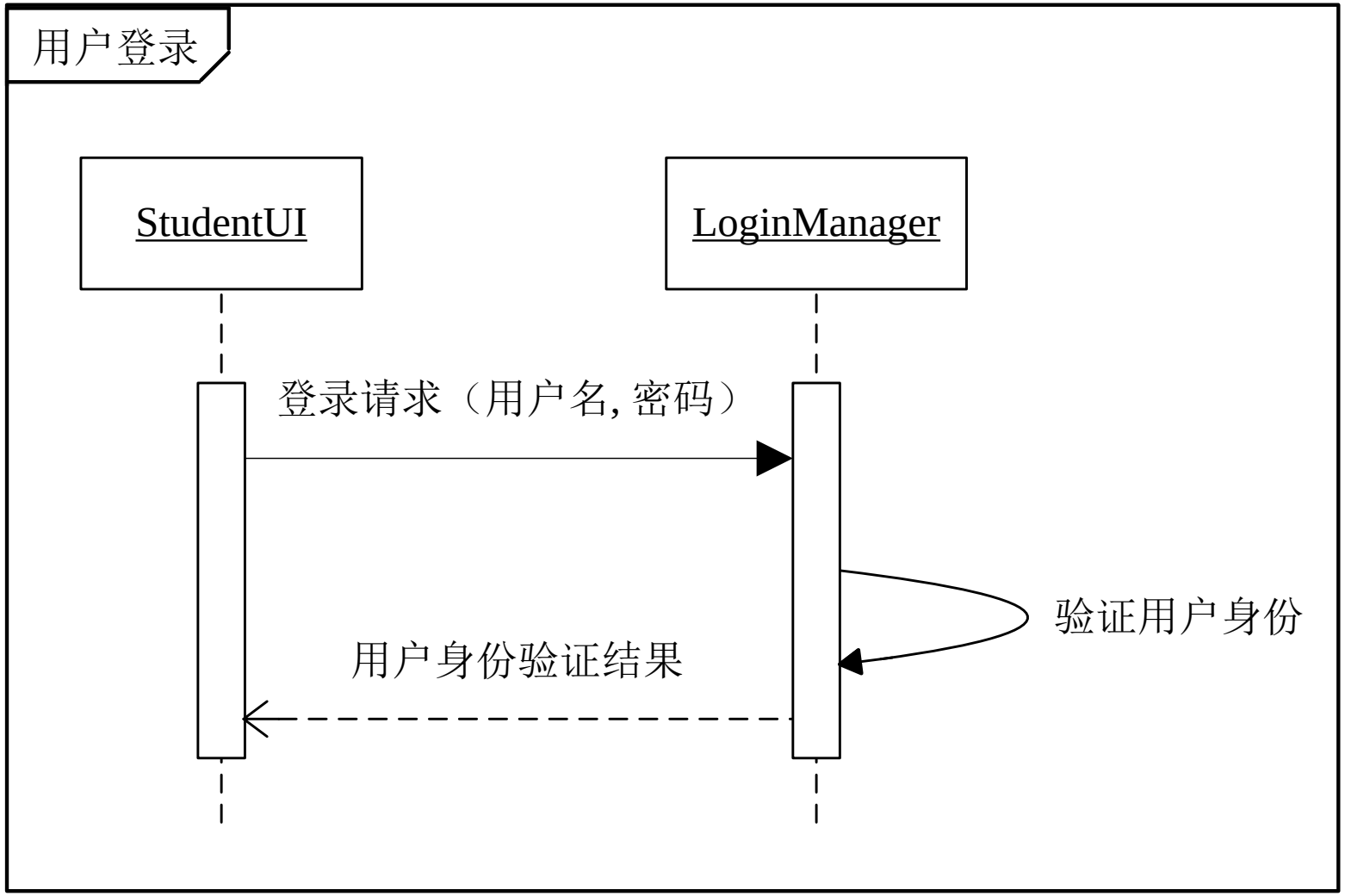
带时间延迟的消息





返回消息

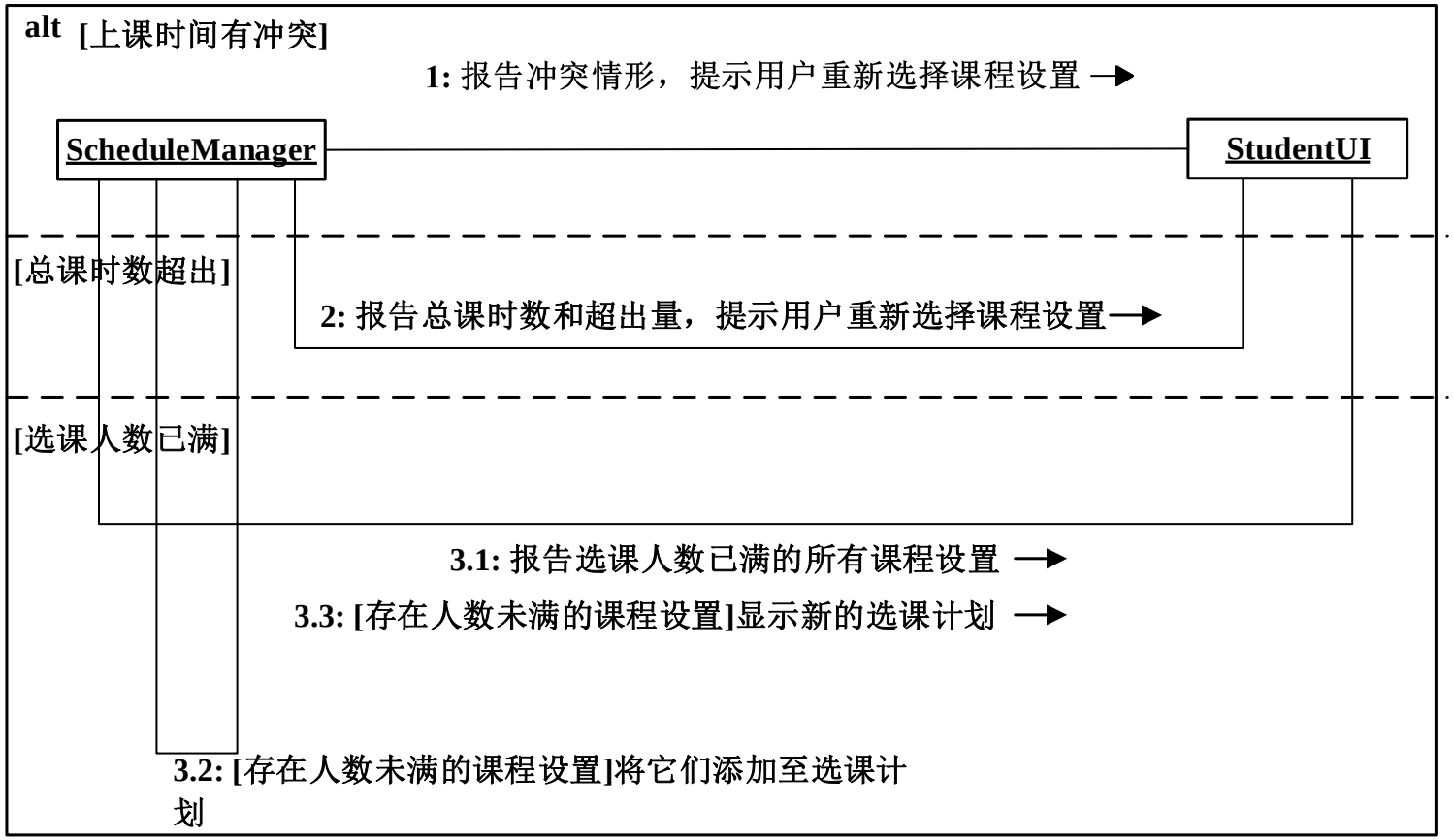
示例：顺序图



- ✓ 对象
- ✓ 生命线
- ✓ 活跃期
- ✓ 同步消息
- ✓ 返回消息
- ✓ 自消息
- ✓ 消息名称

通信图的表示

- 节点表示对象
- 对象间连接称为连接器
- 连接器上可标示一到多条消息
- 消息传递方向用靠近消息小箭头表示
- 消息序号采用多层标号

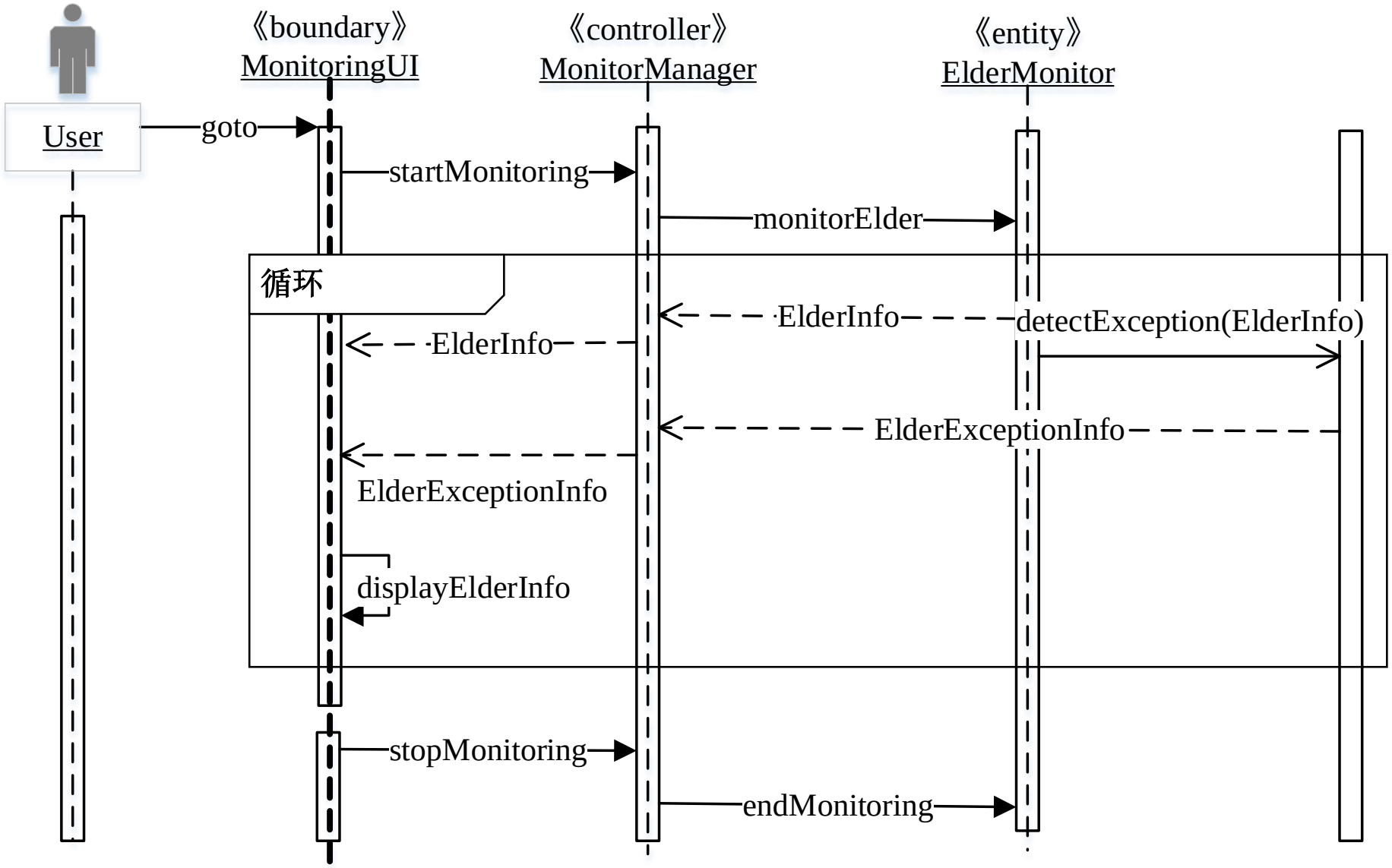


构建用例交互模型时遵循的策略

- 根据对象所处层次组织对象在顺序图中的位置，接近用户界面的对象靠左，接近后台处理的对象靠右；
- 尽量使消息边的方向从左至右来布局；
- 要根据构建交互模型的不同目的决定顺序图描述的详尽程度，既要防止陷入实现细节，又要防止在一张顺序图中表达各种可能的协作情况，导致顺序图过于复杂

示例：顺序图

- 指出顺序图中各个图形要素及其含义



类和对象图

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

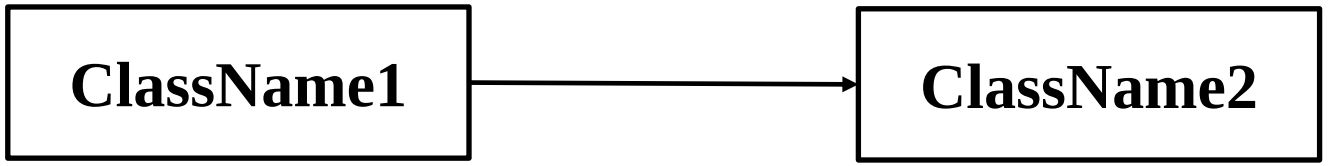
1.3.2 类图

❑功效

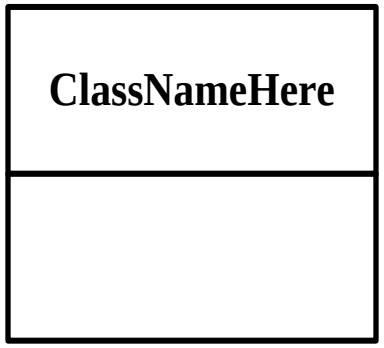
- ✓描述系统的**类构成**，刻画系统的**静态组成结构**

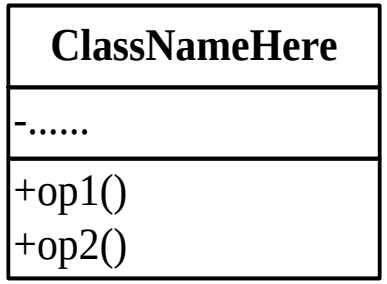
❑图的构成

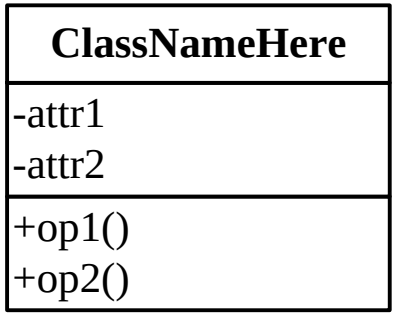
- ✓**结点**：表示系统中的类（或接口）及其属性和操作
- ✓**边**：类之间的关系

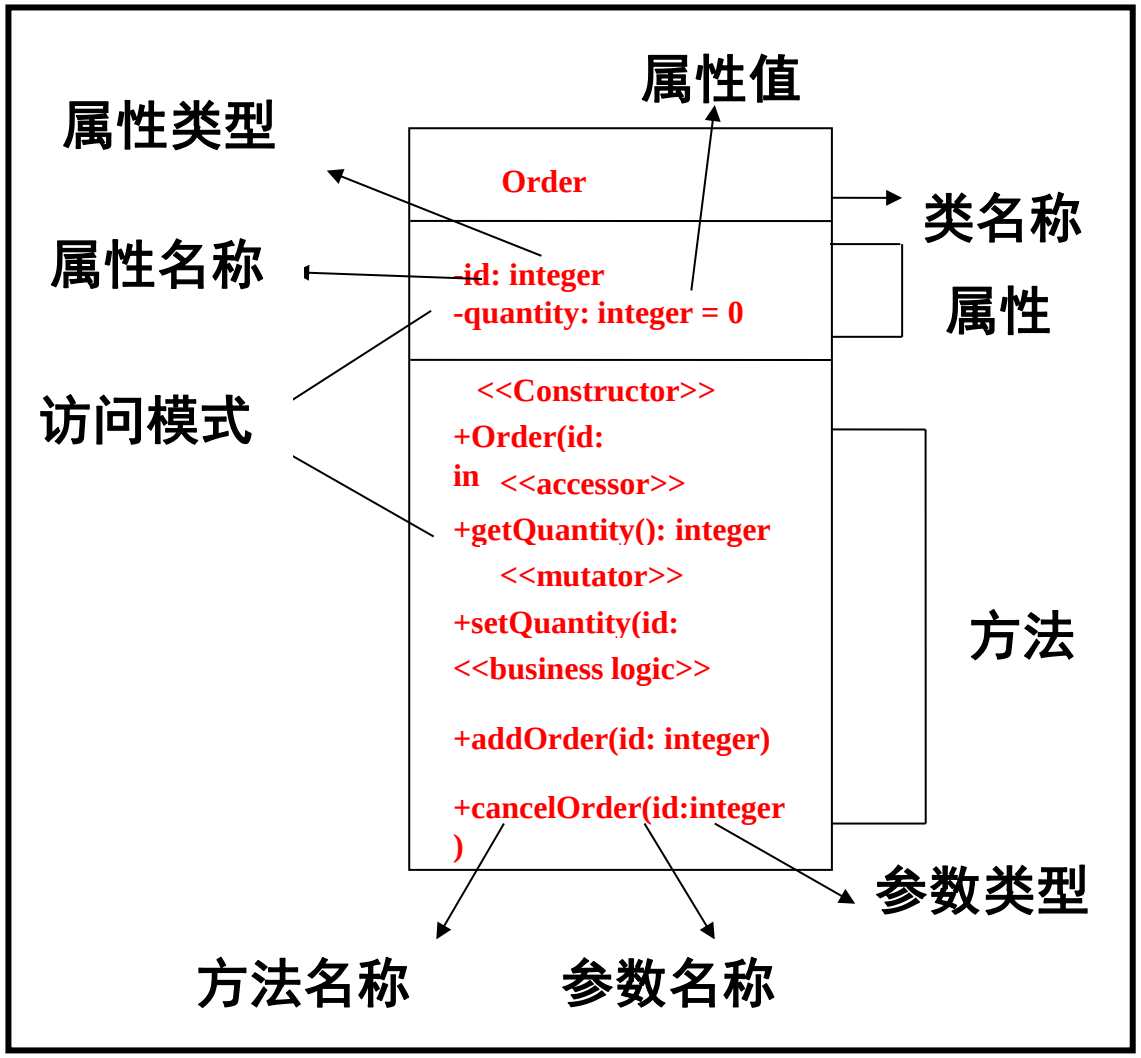


类的UML表示









- ✓ 类名
- ✓ 属性
- ✓ 操作

属性的表示

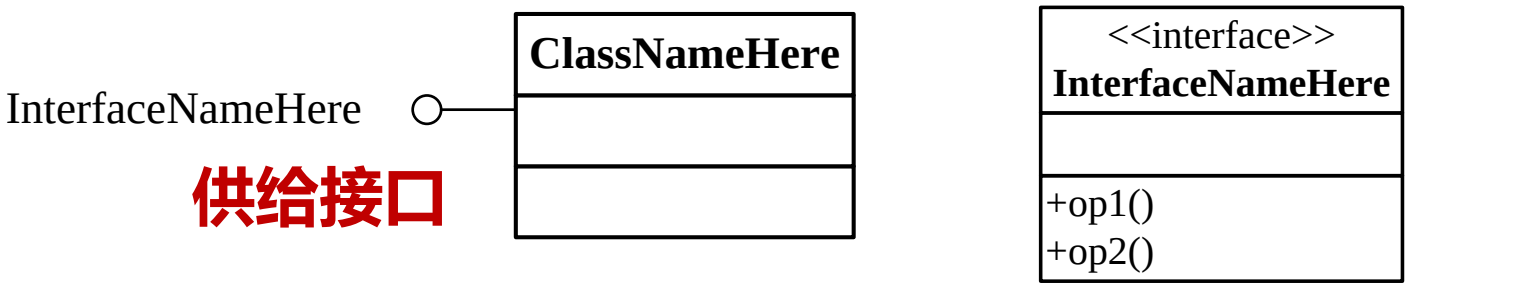
- **[可见性] 名称 [: 类型] [多重性] [= 初值] [{约束特性}]**
- **可见性**
  - ✓公开(+): **所有对象**均可访问
  - ✓保护(#): **所在类及子类对象**均可访问
  - ✓私有(-): **仅所在类的对象**才可访问
- **多重性：属性取值数量, 如1，0..1，0..* ，1..*，*
- **约束特性**
  - ✓可更改性：{readOnly}表示只读，缺省为{changeable}
  - ✓顺序性： {ordered}表示属性取值是有序的，缺省为{unordered}
  - ✓唯一性： {bag}表示属性取值元素允许出现重复元素（缺省）
  - ✓静态性：{static}表示静态属性，属性值由类所有实例对象共享

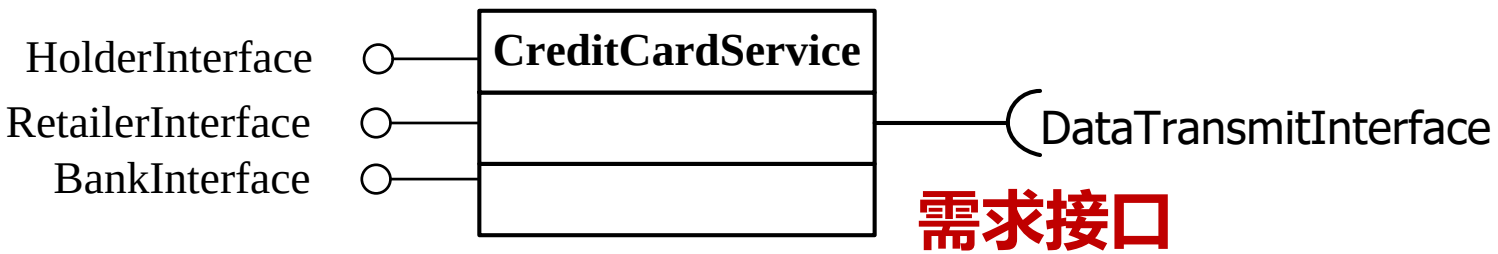
方法的表示

- [可见性] 名称[(参数表)] [: 返回类型] [{约束特性}]
- 操作的约束特性
  - ✓**查询操作**：{isQuery = true}表示查询操作，{ isQuery = false}表示修改操作，缺省为修改操作。
  - ✓**多态性**：{isPolymorphic = true}表示本操作允许多态，即可被子类中相同定义形式的操作所覆盖
  - ✓**并发性**：{concurrency = sequential} 任一时刻只有一个对象调用可执行。{concurrency = guarded} 并行线程可同时调用多个对象的本操作，但同一时刻只允许一个调用执行。{concurrency = concurrent} 并行线程可以同时调用多个对象的本操作且这些调用可并发执行
  - ✓**异常**：操作在执行过程中可能引发异常

接口(Interface)

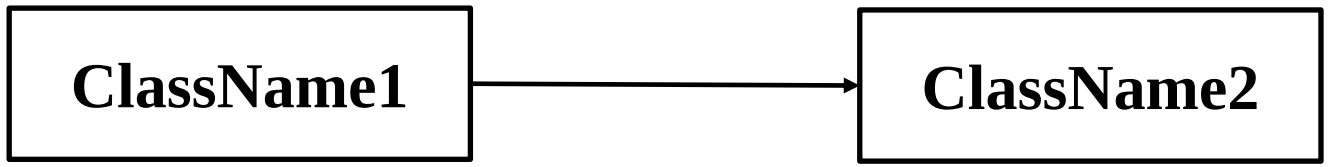
- 一种**不包含操作实现部分**的特殊类
- 接口的形式
  - ✓**供给接口**：对外提供的接口
  - ✓**需求接口**：需要使用的接口





类间的关系

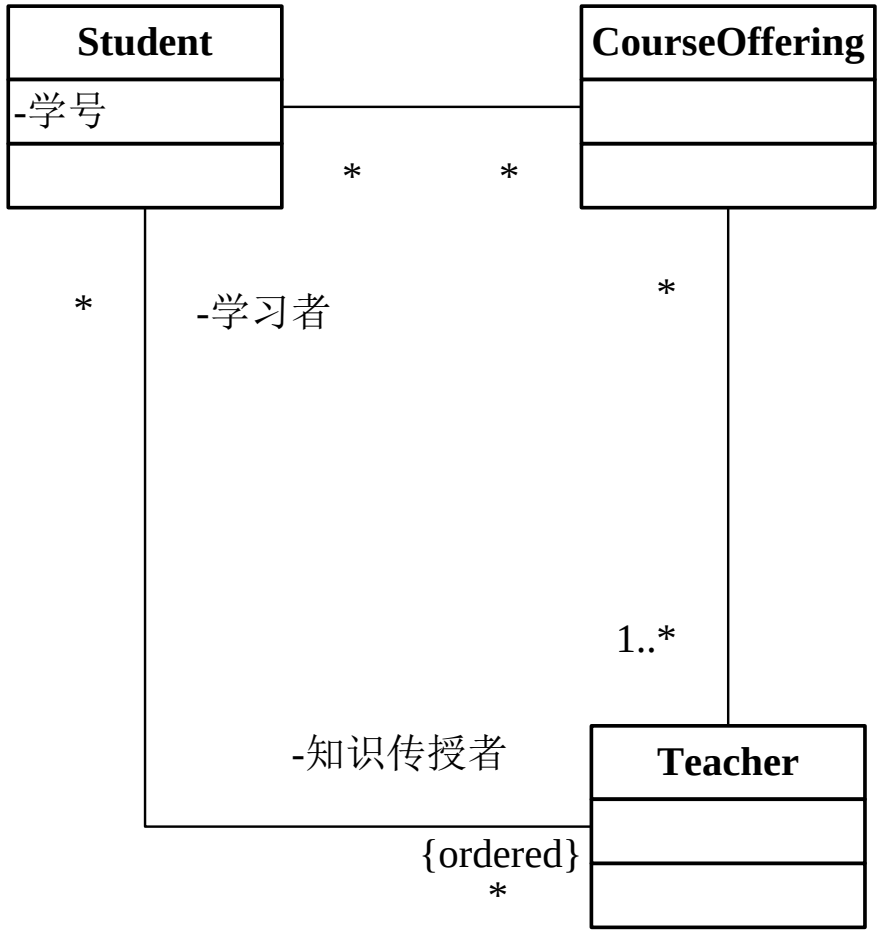
- 关联
- 依赖
- 继承
- 实现
- 聚合
- 组合



类间关系-关联

❑表示类间的逻辑联系

- ✓**多重性**：位于关联端的类可以有多少个实例对象与另一端的类的单个实例对象相联系
- ✓**角色名**：参与关联的类对象在关联关系中扮演的角色或发挥的作用
- ✓**约束特性**：针对参与关联的对象或对象集的逻辑约束



类间关系-聚合与组合

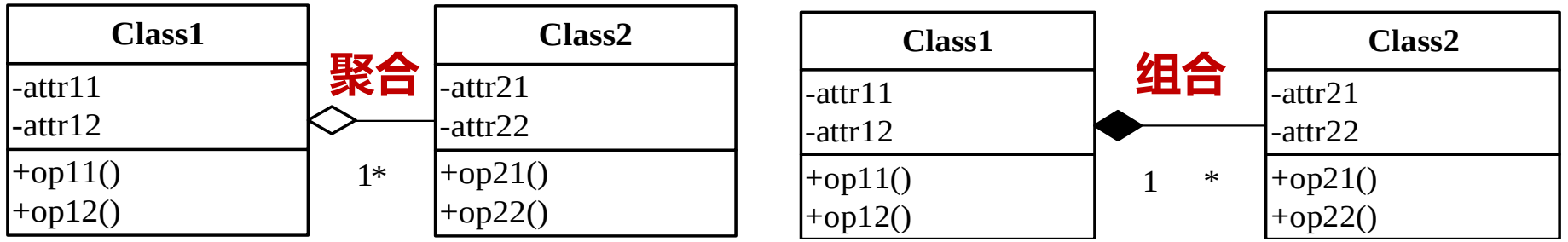
❑聚合关系(Aggregation)

- ✓**部分类**对象是多个**整体类**对象的组成
- ✓示例：一名学生可同时参与多个兴趣小组

举例说明组合和聚合的差异性

❑组合关系(Composition)

- ✓**部分类对象只能位于一个整体类对象中**
- ✓一旦整体类对象消亡，部分类对象也不能苟活



注意箭头图符

类间关系-依赖

❑依赖关系

✓有**语义上关系**且一个类对象变化会导致另一类对象作相应修改

❑依赖形式 A依赖于B

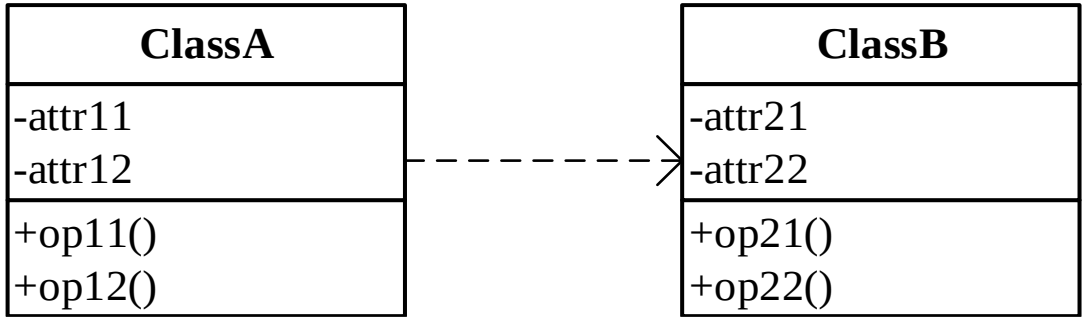
✓**使用**：A类使用B类的某项服务

✓**追踪**：B类的变化导致A类的变化

✓**精化**：A类是在B类基础上引进设计、决策等形成

✓**实现**：A类给出了B类元素的实现

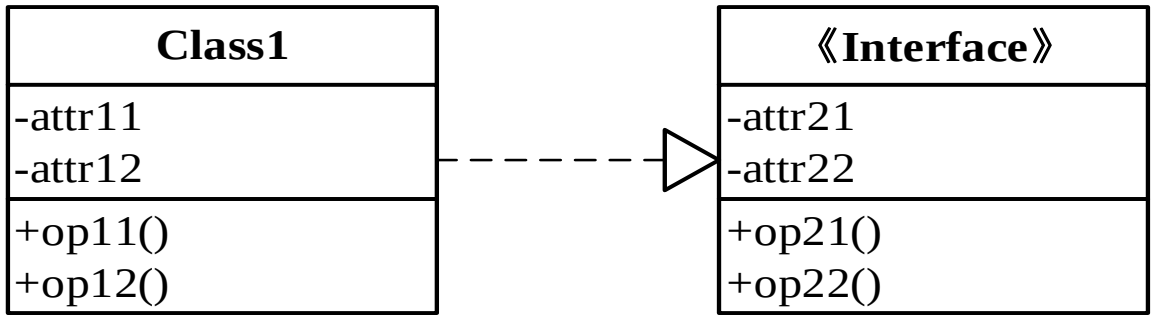
✓**派生**：A类可通过B类推导或计算出



箭头指向被依赖方

类间关系-实现

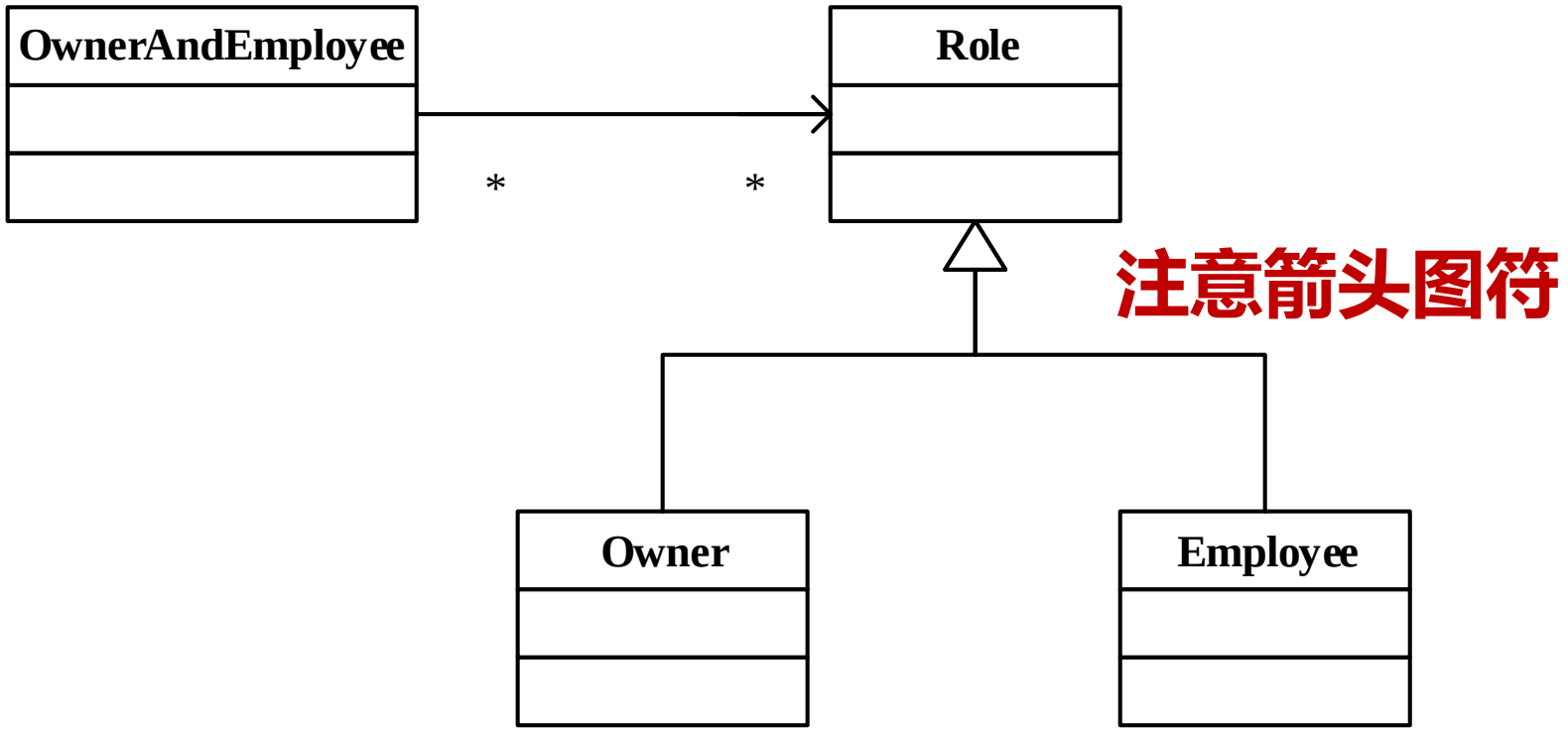
- 表示一个类实现了另一个类中定义的对外接口
  - ✓特殊依赖关系
- 典型应用
  - ✓一个具体类实现一个接口



注意箭头图符

类间关系-继承

- 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（override）父类中的原有特性
- 父类抽象多个子类中公共特性，从而避免冗余、简化操作



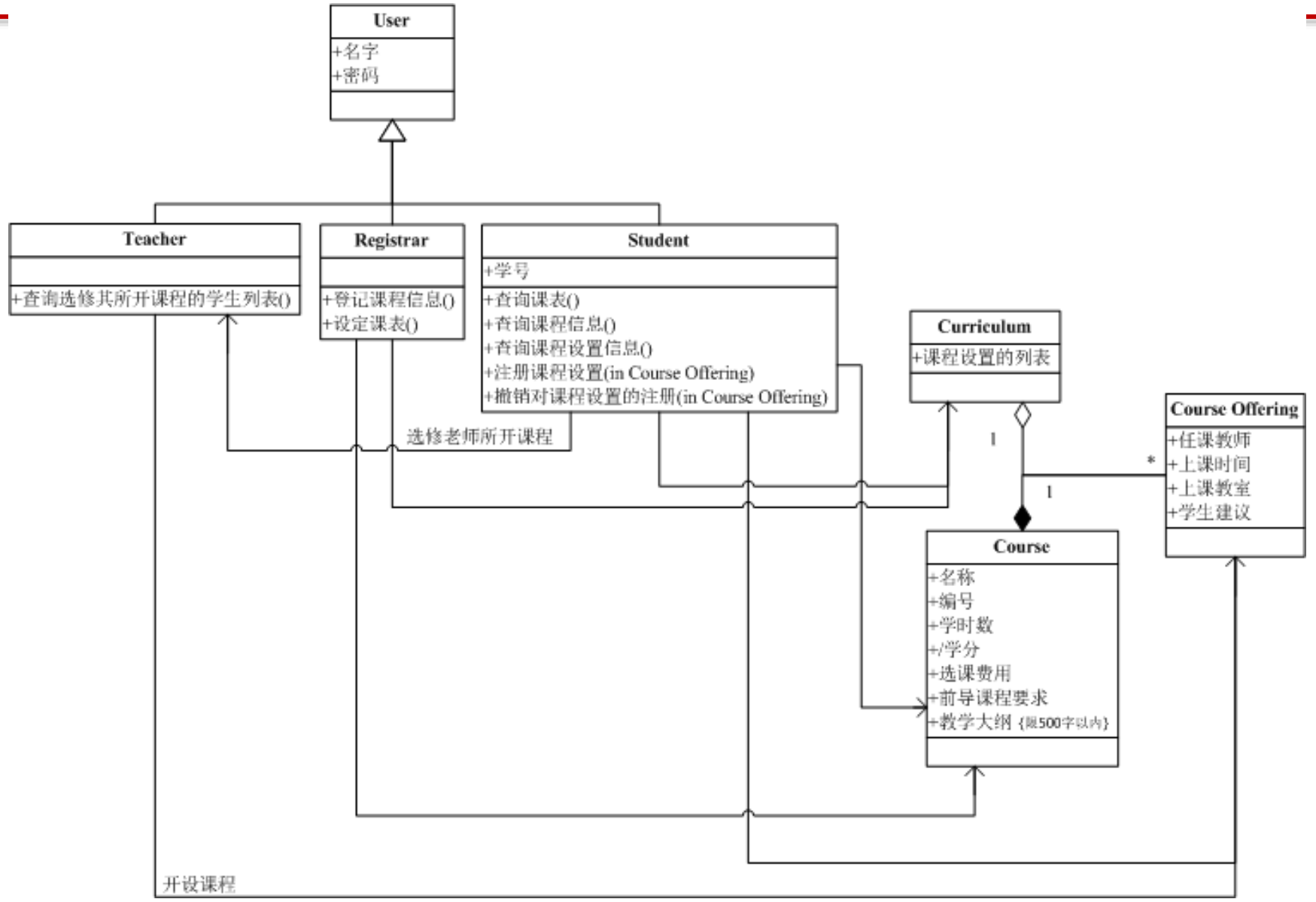
注意事项

- 用**单数名词**来描述类名，少用缩写

- 按照方向表示类间关系
  - ✓垂直方向表示继承关系
  - ✓水平方向表示关联、聚合、组合、依赖、实现关系

- 注意画图位置
  - ✓关联名应位于关联边的居中位置
  - ✓多重性、角色名、约束特性等应靠近关联端

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

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

1.3.4 状态图

❑功效

- ✓描述实体（对象、系统）在**事件**刺激下的**反应式动态行为**及其导致的**状态变化**
- ✓刻画了实体的动态行为特征、每个状态下可响应事件、响应动作、状态迁移

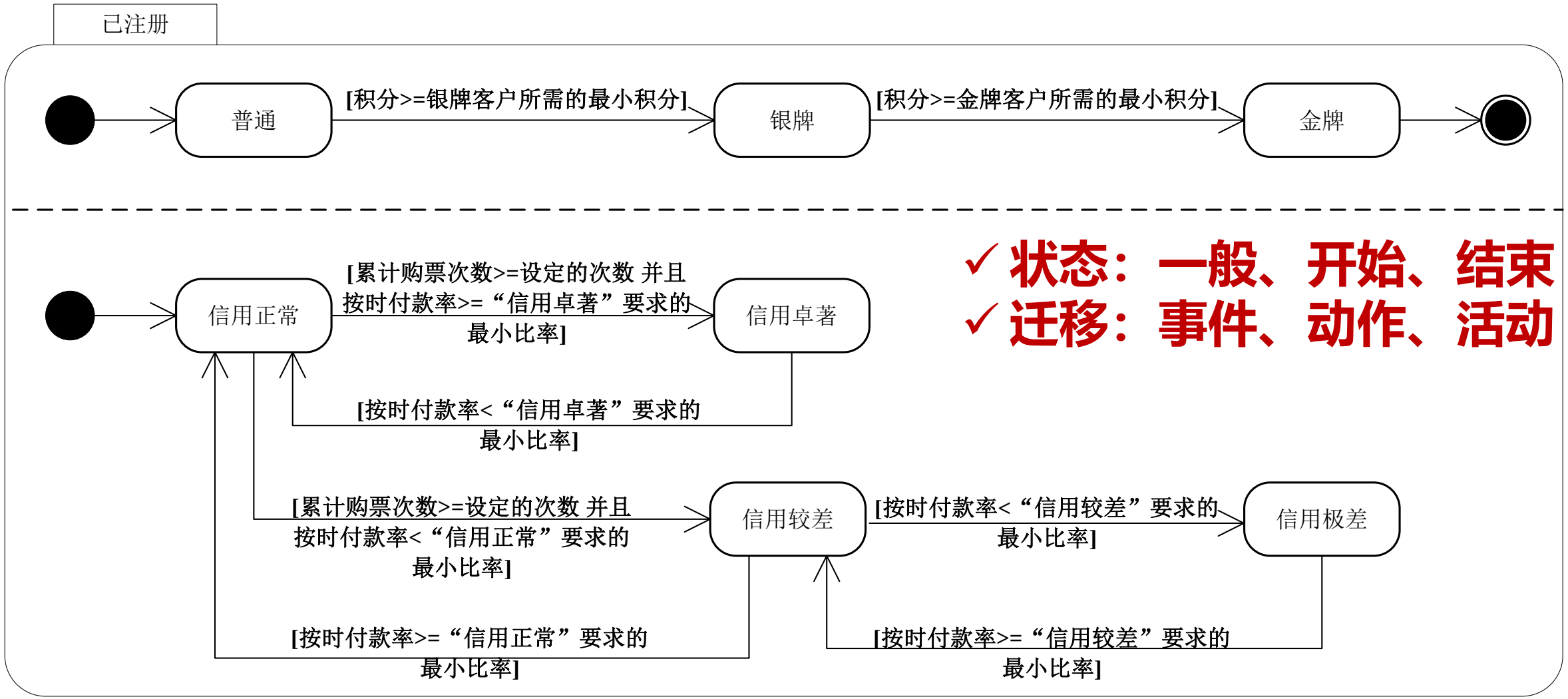
❑图的构成

- ✓**节点**：状态
- ✓**边**：迁移，即状态间因事件刺激而触发的状态变化

状态图的基本概念

- **状态：对象属性取值构成的一个约束条件**
  - ✓用于表征对象处于某种特定状况，该状态下对象对事件的响应行为完全一样
- **事件：某时刻点发生、需要关注的瞬时刺激或触动，将引发对象实施某些行为，导致对象状态的变化。**
  - ✓**消息型事件(同步)**：其他对象发来消息，比如“支付成功”消息
  - ✓**信号型事件(异步)**：其他对象传来异步信号
  - ✓**时间型事件**：到达特定时间点，“开车时间”
  - ✓**条件型事件**：对象属性取值满足特定条件，“购买车票却没有完成支付”
- **动作(action)**
  - ✓计算过程，**位于迁移边上**，简单，执行时间短
- **活动(activity)**
  - ✓计算过程，**位于状态中**，复杂、执行时间长

示例：状态图



- ✓ 状态：一般、开始、结束
- ✓ 迁移：事件、动作、活动

状态的表示

- **由状态名及可选的入口活动、出口活动、do活动、内部迁移构成**
  - ✓一旦对象经迁移边从其他状态进入本状态，那么本状态**入口活动**将被执行
  - ✓一旦对象经迁移边从本状态进入其他状态，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并执行完入口活动（如果有的话）后应该执行的活动。
  - ✓**内部迁移**不会引起对象状态的变化，所以它虽有源状态（包含此迁移的状态），但没有目标状态

迁移的表示

- 迁移标注为“[事件] [监护条件] [/ 动作]”
  - ✓“**事件**”表示触发此次状态变迁的事件
  - ✓“**监护条件**”表示约束状态迁移真正发生的条件表达式
  - ✓“**动作**”表示状态迁移期间应当执行的动作（action）
  - ✓在监护条件和动作中均可引用触发事件参数和对象属性

- 迁移表示为状态节点间的有向边
  - ✓自迁移是指源状态与目标状态相同的特殊外部迁移

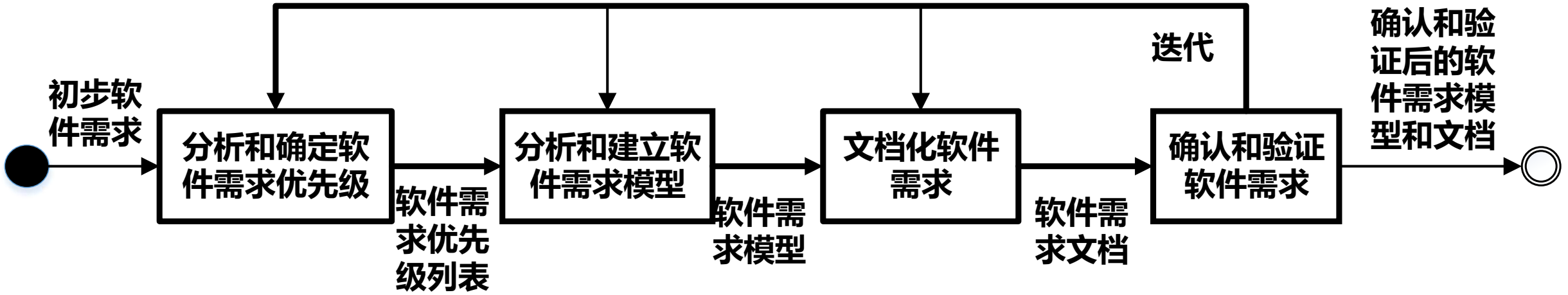
状态图绘制原则

- **绘图**
  - ✓状态节点按行上下对齐，按列左右对齐
  - ✓迁移边绘制成水平线段、垂直线段或者由水平、垂直两种线段组合而成的折线
  - ✓避免斜线
- **初态应位于左上角，终态尽量靠近右下角**
- **迁移边上的文本标注应靠近源状态**

内容

1. 分析软件需求概述
   - ✓分析软件需求的任务
   - ✓UML描述方法
2. **分析软件需求过程**
   - ✓分析和确立软件需求优先级
   - ✓分析和建立软件需求模型
3. **软件需求文档化及评审**
   - ✓软件需求规格说明书
   - ✓评审软件需求

需求分析过程


初步软件需求
分析和确定软件需求优先级
软件需求优先级列表
分析和建立软件需求模型
软件需求模型
文档化软件需求
软件需求文档
确认和验证软件需求
迭代
确认和验证后的软件需求模型和文档

2.1 分析和确定软件需求优先级

- 分析软件需求**重要性**
- 分析软件需求**优先级**
- 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
  - ✓在解决问题方面起到举足轻重的作用，提供了软件系统所特有的功能和服务，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

- **用户对实现这些软件需求的优先级要求也不一样**
  - ✓有些软件需求需要优先实现，尽早交付给用户使用，以发挥其价值；有些软件需求可以滞后实现，晚点交付使用

- **优先级的考虑因素**
  - ✓结合软件项目开发的具体约束，考虑不同软件需求的重要性，确定软件需求的实现优先级，确保在整个迭代开发中有计划、有重点地实现软件需求
  - ✓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**

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

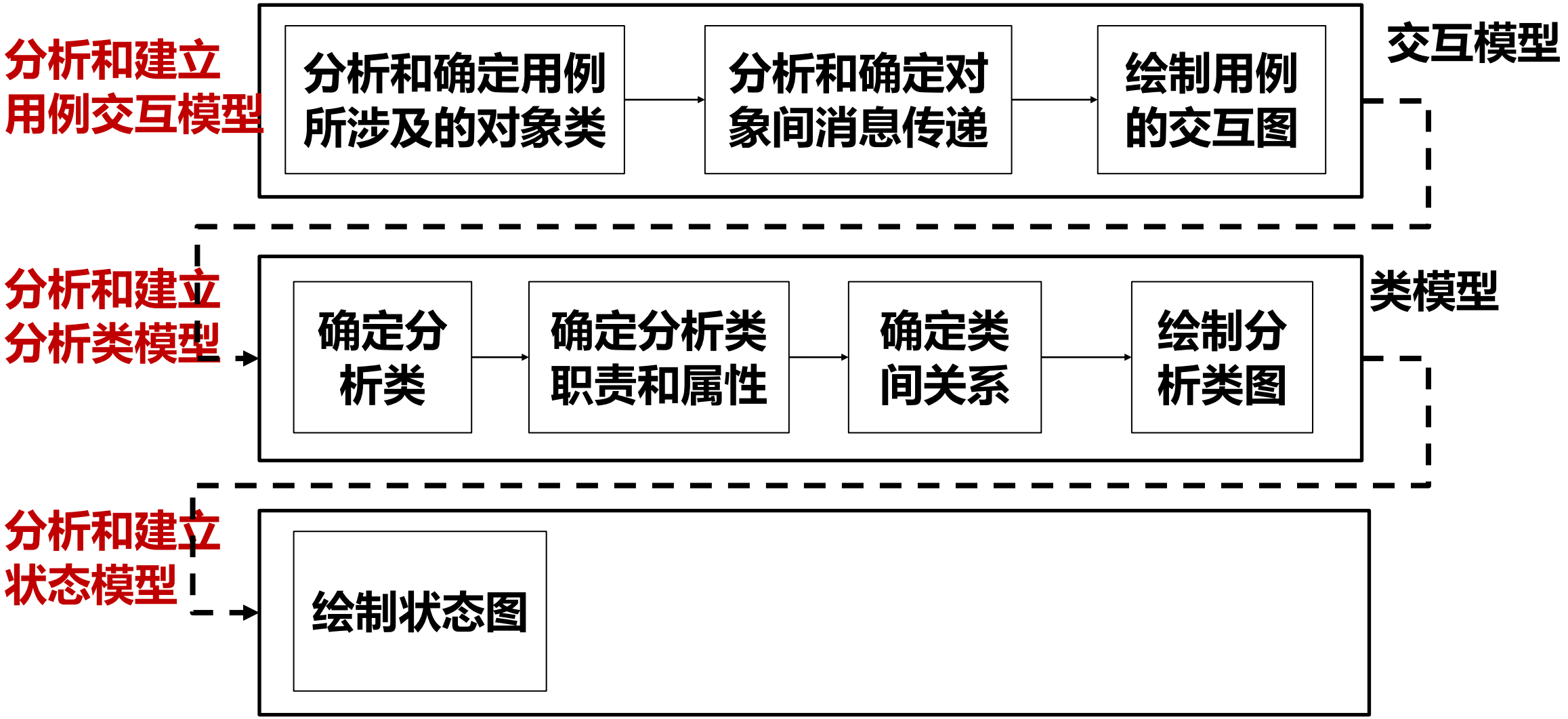
2.1.3 确定用例分析和实现的次序

- **确保有序地开展需求分析、软件设计和实现工作，使得每次迭代开发有其明确的软件需求集，每次迭代开发结束之后可向用户交付他们所急需的软件功能和服务**
  - ✓结合软件开发的迭代次数、每次迭代的持续时间、可以投入的人力资源等具体情况
  - ✓充分考虑相关软件需求项的开发工作量和技术难度等因素，确定需求用例分析和实现的先后次序

示例：“空巢老人看护软件”需求开发安排

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 分析和建立软件需求模型的具体步骤



2.2.1 分析和建立用例的交互模型

❑任务

✓分析和描述用例是如何通过一组对象之间的交互来完成的

❑步骤

① 分析和确定用例所涉及的对象及其类

② 分析和确定对象之间的消息传递

③ 绘制用例的交互图

分析和确定用例所涉及的对象及其类

- 软件需求用例的处理通常涉及三种不同类对象
  - ✓边界类
  - ✓控制类
  - ✓实体类

- 这些类是在用例分析阶段所识别并产生的，通常将它们称为分析类

边界类

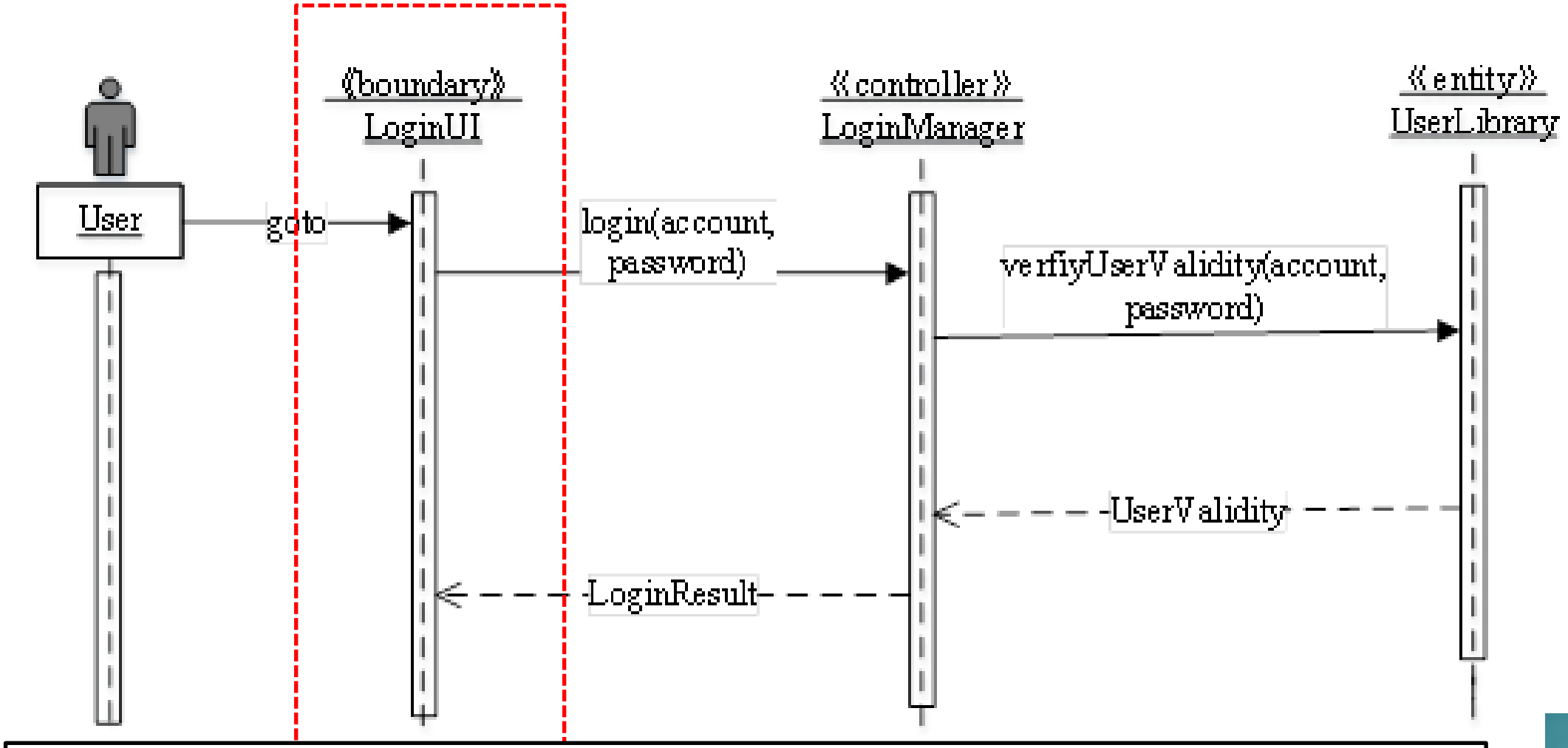
❑何为边界类

- ✓每个用例或者由外部执行者触发，或者需要与外部执行者进行某种信息交互，因而用例的业务逻辑处理需要有一个类对象来负责目标软件系统与外部执行者之间的交互
- ✓由于这些类对象**处于系统的边界**，需与系统外的执行者进行交互，因而将这些对象所对应的类称之为边界类

❑边界类对象主要起到以下作用

- ✓交互控制，处理外部执行者的输入数据，或者向外部执行者输出数据
- ✓外部接口，如果外部执行者表现为其他的系统或者设备，那么边界类对象需要与系统之外的其他系统或设备进行信息交互

示例：边界类



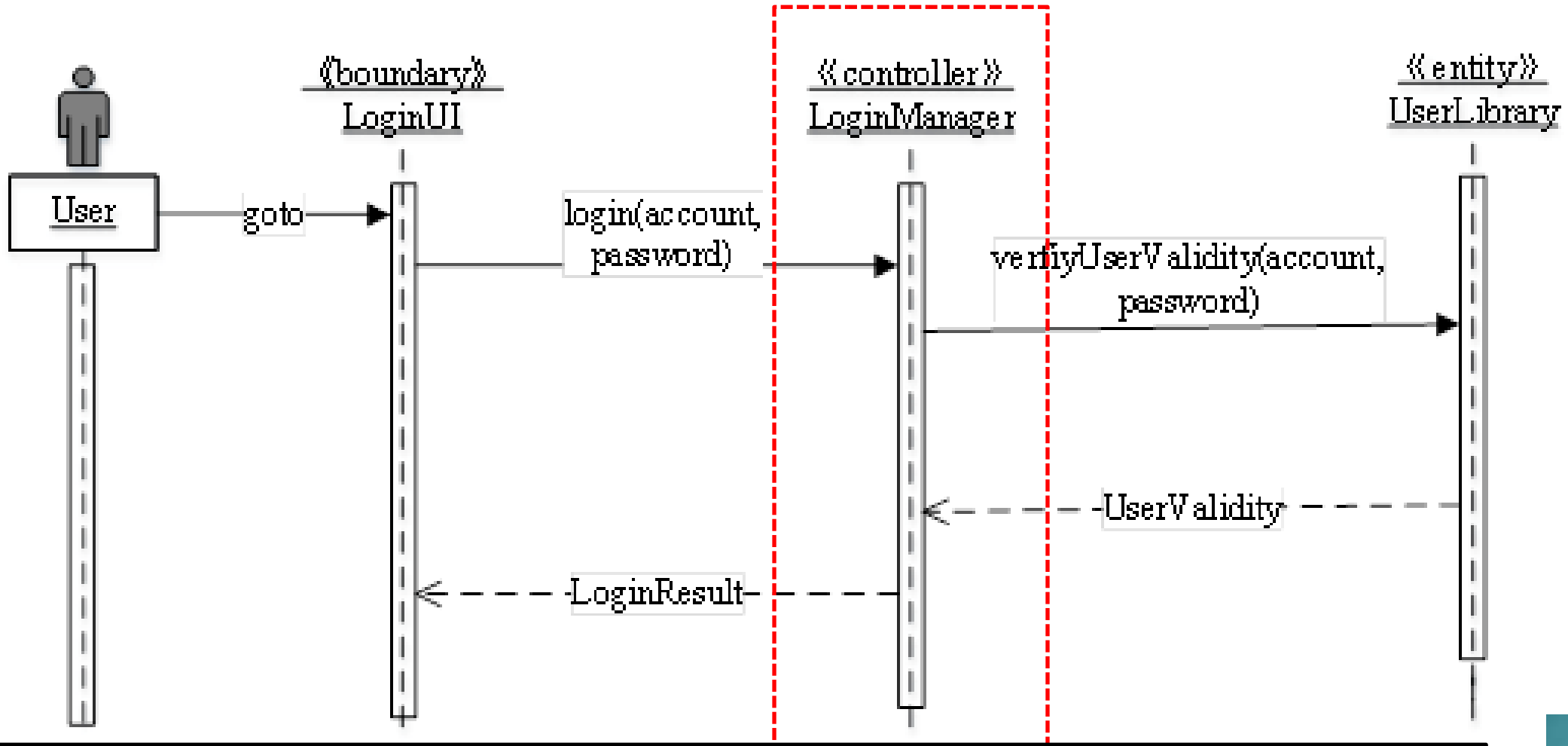
边界类在顺序图中有何特点?

控制类

- **控制类对象作为完成用例任务的主要协调者**
  - ✓负责处理边界类对象发来的任务请求，对任务进行适当的分解，并与系统中的其他对象进行协同，以控制他们共同完成用例规定的任务或行为

- **一般而言，控制类并不负责处理具体的任务细节，而是负责分解任务，并通过消息传递将任务分派给其他对象类来完成，协调这些对象之间的信息交互**

示例：控制类

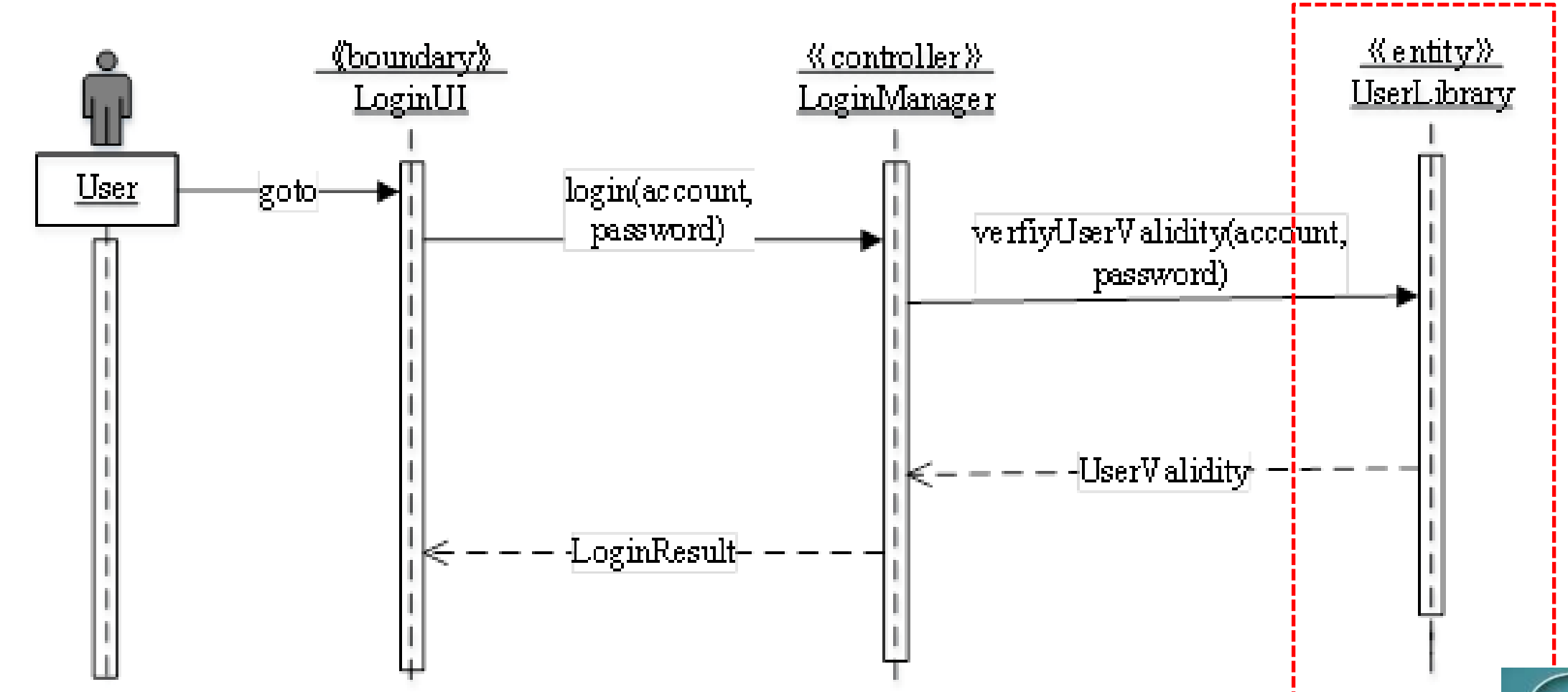


控制类在顺序图中有何特点?

实体类

- **用例所对应业务流程中的所有具体功能最终要交由具体的类对象来完成，这些类称之为实体类**
  - ✓一般地，实体类对象负责保存目标软件系统中具有持久意义的信息项，对这些信息进行相关的处理（如查询、修改、保存等），并向其他类提供信息访问的服务
  - ✓实体类的职责是要落实目标软件系统中的用例功能，提供相应的业务服务

示例：实体类



实体类在顺序图中有何特点?

2.2.2 分析和确定对象之间的消息传递

- 确定消息的名称
- 确定消息传递的信息

确定消息的名称

❑消息的名称

- ✓直接反映了对象间**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**，也即发送方对象希望接收方对象提供什么样的功能和服务
- ✓**意图**：请求、通知

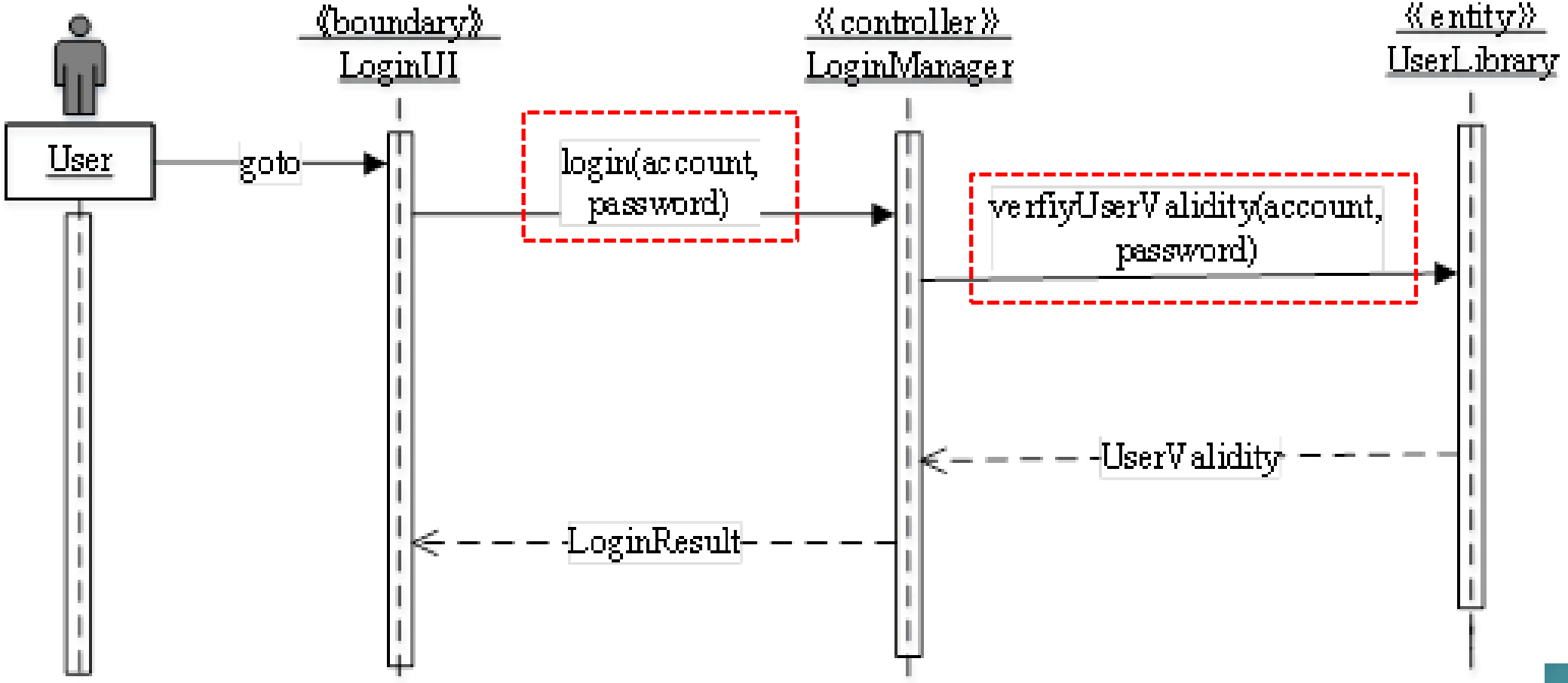
❑消息名称的表示

- ✓一般地，消息名称用**动名词**来表示
- ✓需求工程师应尽可能用**应用领域中通俗易懂的术语**来表达消息的名称和参数，以便于用户和需求工程师等能直观地理解对象间的交互语义

确定消息传递的信息

- **对象间的交互除了要表达消息名称和交互意图之外，在许多场合还需要提供必要的交互信息**
  - ✓这些信息通常以消息参数的形式出现，也即一个对象在向另一个对象发送消息的过程中，需要提供必要的参数，以向目标对象提供相应的信息
  - ✓**信息**：通知和请求的内容
- **在构建用例的交互图过程中，如果用例的业务流程能够明确相应的交互信息，那么就需要确定消息需附带的信息**
- **通常，消息参数用名词或名词短语来表示**

示例：消息的名称和信息



消息的意图和信息是什么?

2.2.3 绘制用例的交互图

❑绘制顺序图的策略

- ✓用例的外部执行者应位于图的最左侧，紧邻其右的是用户界面或外部接口的边界类对象，再往右是控制类对象，控制类的右侧应放置实体类对象，它们的右侧是作为外部接口的边界类对象
- ✓对象间的消息传递采用**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，为其构造一个交互图就足够了
- ✓对于较复杂的用例而言，需要为此用例绘制多张交互图，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

用例交互图的工作流程

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，将信息从外部形式转换为内部形式，并通过消息传递将相关信息发送给控制类对象

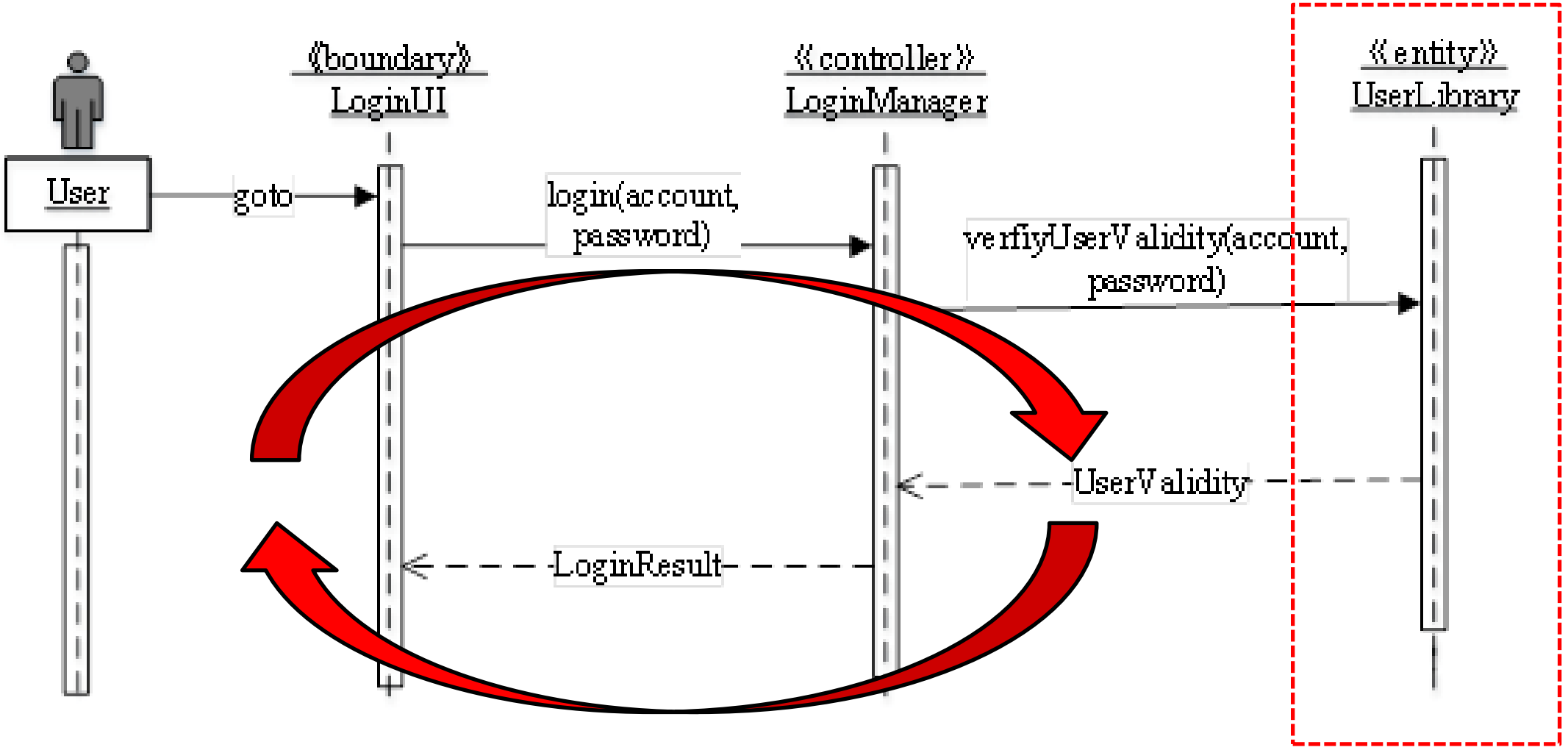
③ 控制类对象根据业务逻辑处理流程，产生和分解任务，与相关的实体类对象进行交互以请求完成相关的任务，或者向实体类对象提供业务信息，或者请求实体类对象持久保存业务逻辑信息，或者请求获得相关的业务信息

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

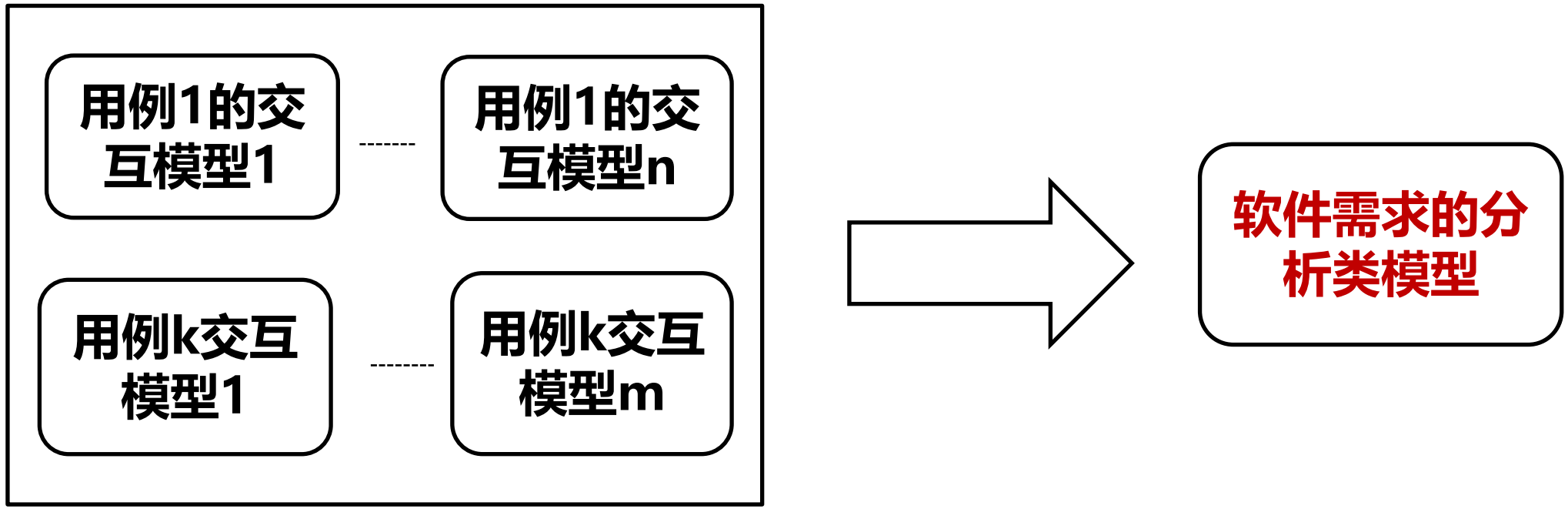
⑤ 控制类对象处理接收到的信息，将处理结果通知边界类对象

⑥ 边界类对象对接收到的处理结果信息进行必要的分析，将其从内部形式转换为外部形式，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


User
《boundary》
LoginUI
《controller》
LoginManager
《entity》
UserLibrary
goto
login(account,
password)
verfiyUserValidity(account,
password)
UserValidity
LoginResult

2.3 分析和建立软件需求的分析类模型



分析和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

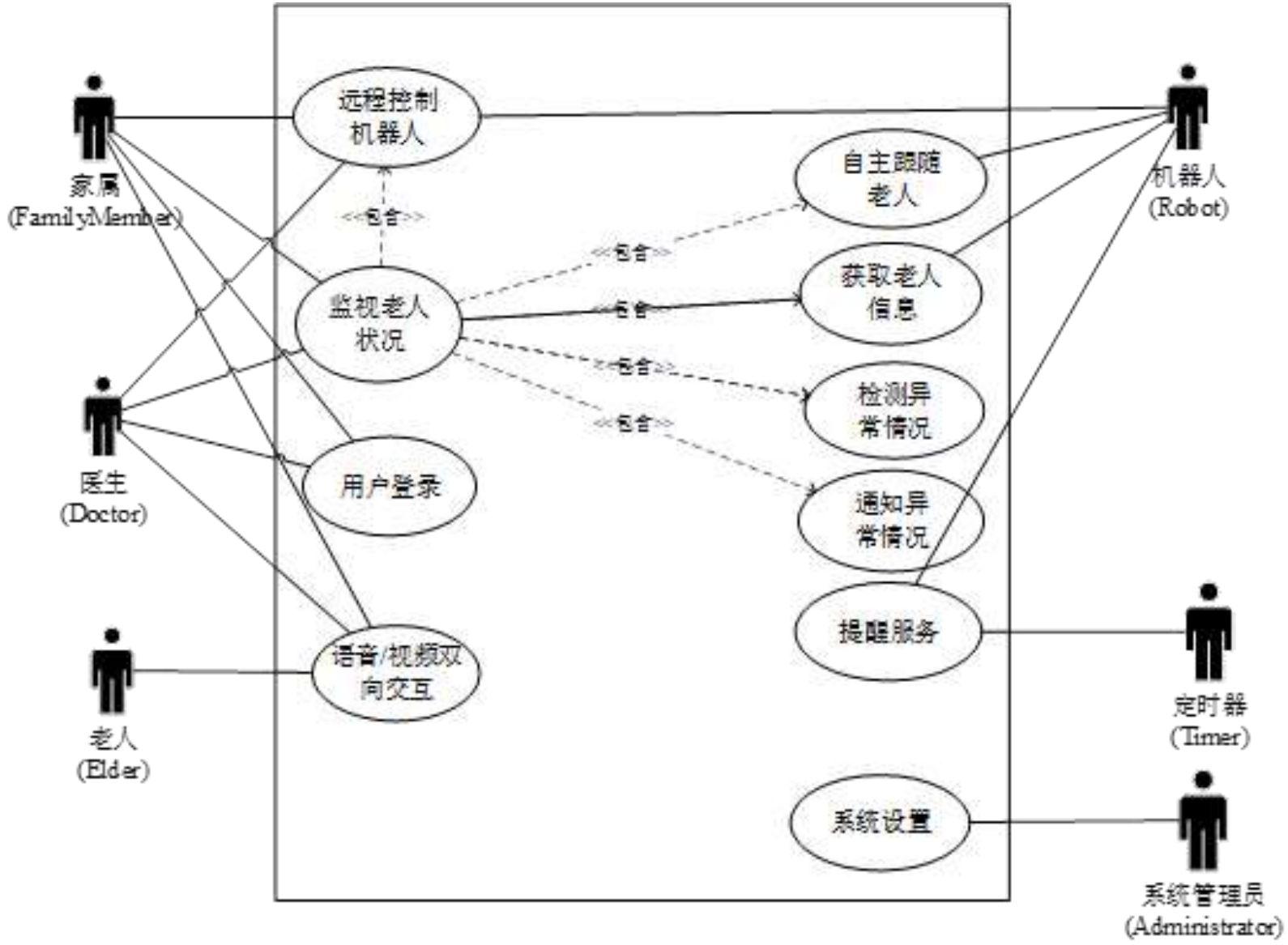
2.3.1 确定分析类

❑有哪几种分析类?

❑如何确定分析类?

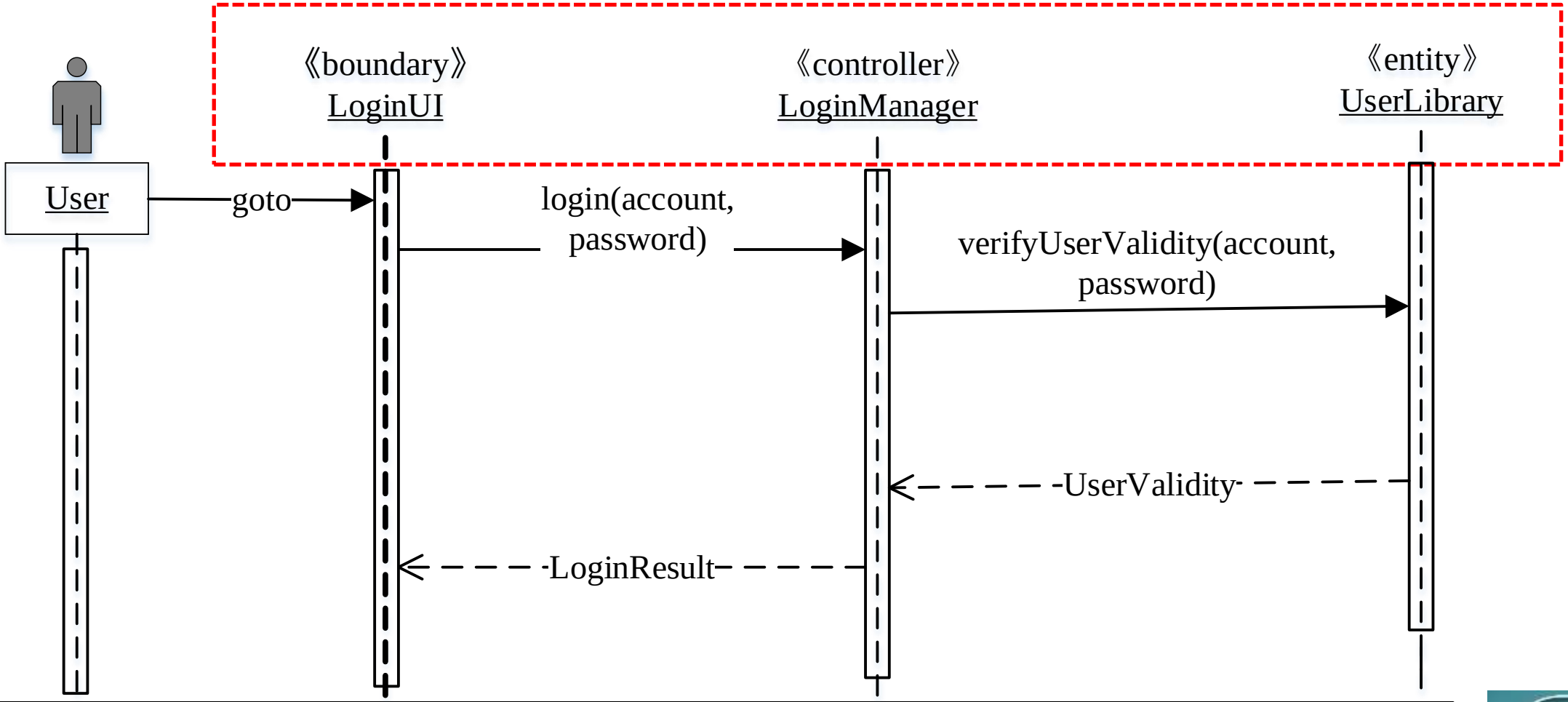
- ✓用例模型中的**外部执行者**应该是分析类图中的类
- ✓在各个用例的顺序图，如果该图中出现了某个**对象**，那么该**对象所对应的类**属于分析类，应出现在分析类图中

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

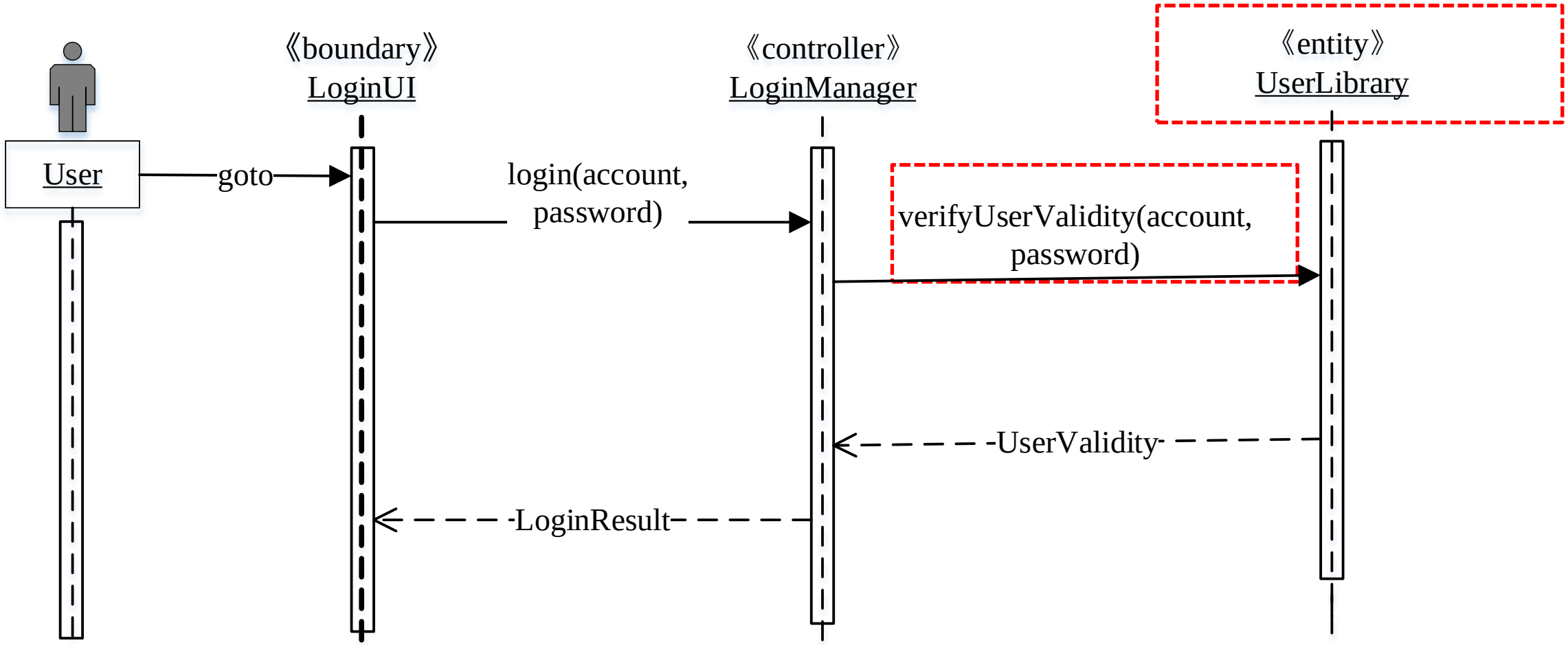


有哪些分析类?

2.3.2 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，并采用简短的自然语言来详细刻画类的职责

示例：确定分析类的职责



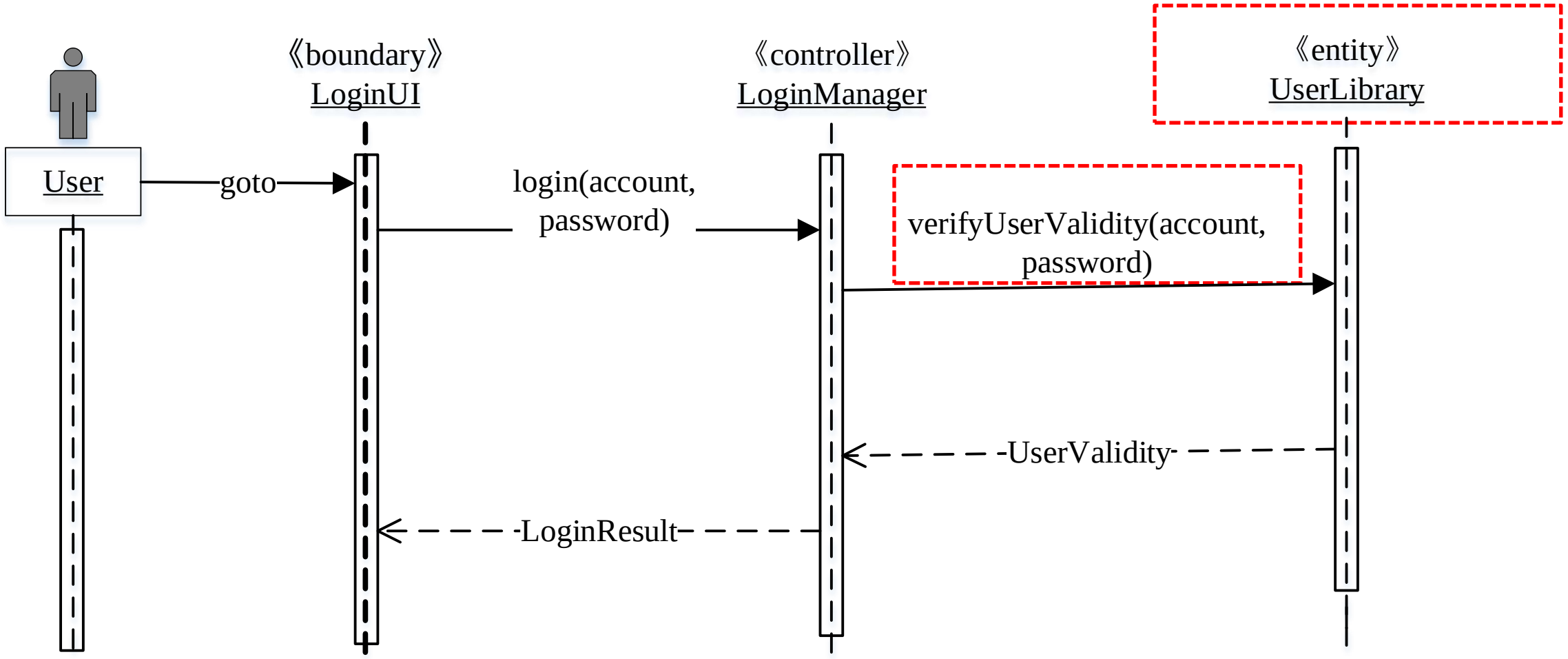
UserLibrary类有何职责?

2.3.3 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

- 用例顺序图中的每个对象所发送和接收的消息中往往附带有相关的参数，这意味着发送或接收对象所对应的类可能需要保存和处理与消息参数相对应的信息，因而可能需要与此相对应的属性

示例：确定分析类的属性

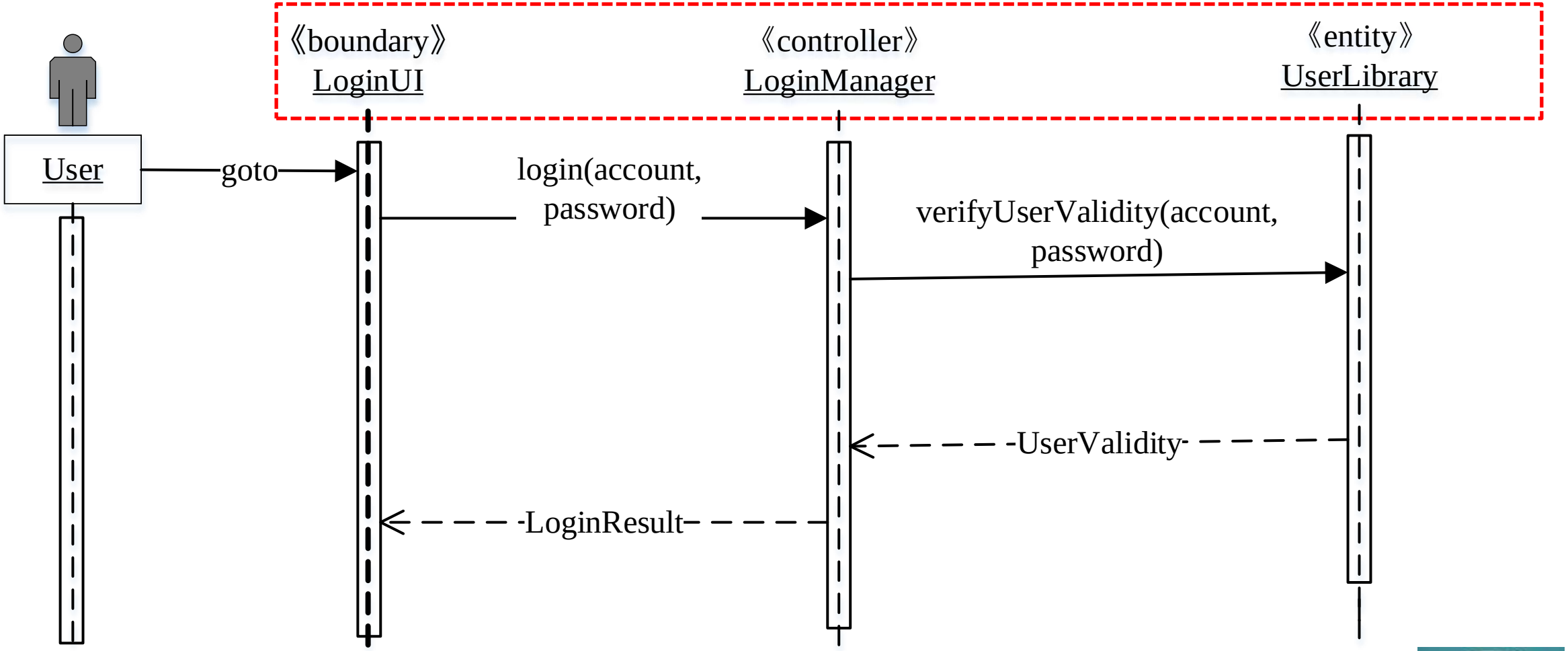


UserLibrary类有何属性?

2.3.4 确定分析类之间的关系

- **从全局视角理解和描述不同分析类间的关系**
- **类之间的关系有多种形式**
  - ✓包括继承、关联、聚合和组合、依赖等等
- **具体策略**
  - ✓在用例的顺序图中，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
  - ✓如果经过上述步骤所得到的若干个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



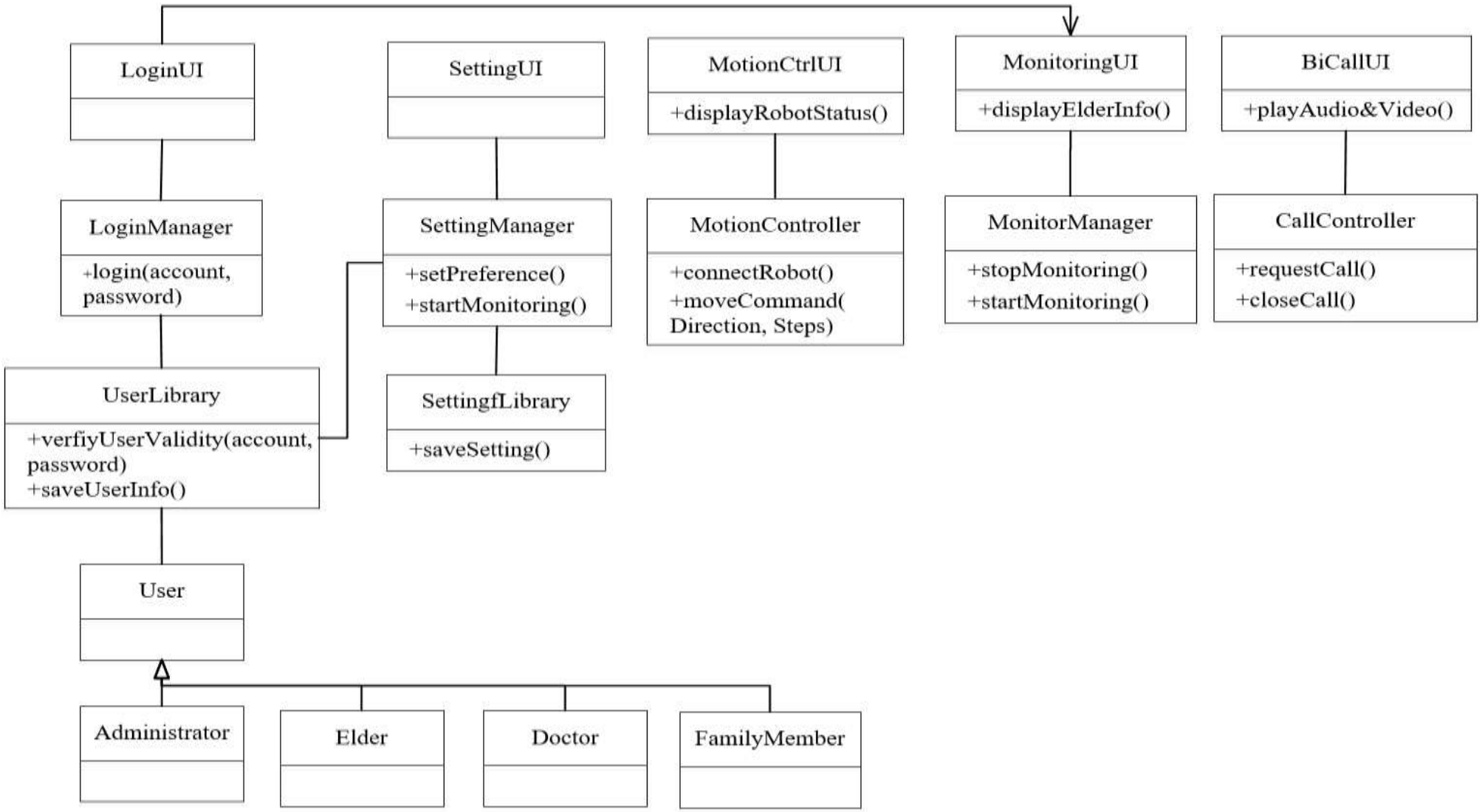
哪些分析类之间存在怎样的关系?

2.3.5 绘制分析类图

- **制出系统的分析类图，建立分析类模型**
  - ✓直观描述了系统中的分析类、每个分析类的属性和职责、不同分析类之间的关系

- **如果系列规模较大，分析类的数量多，关系复杂，难以用一张类图来完整和清晰地表示，那么可以分多个子系统来绘制分析类图**

示例：“空巢老人看护软件”分析类图

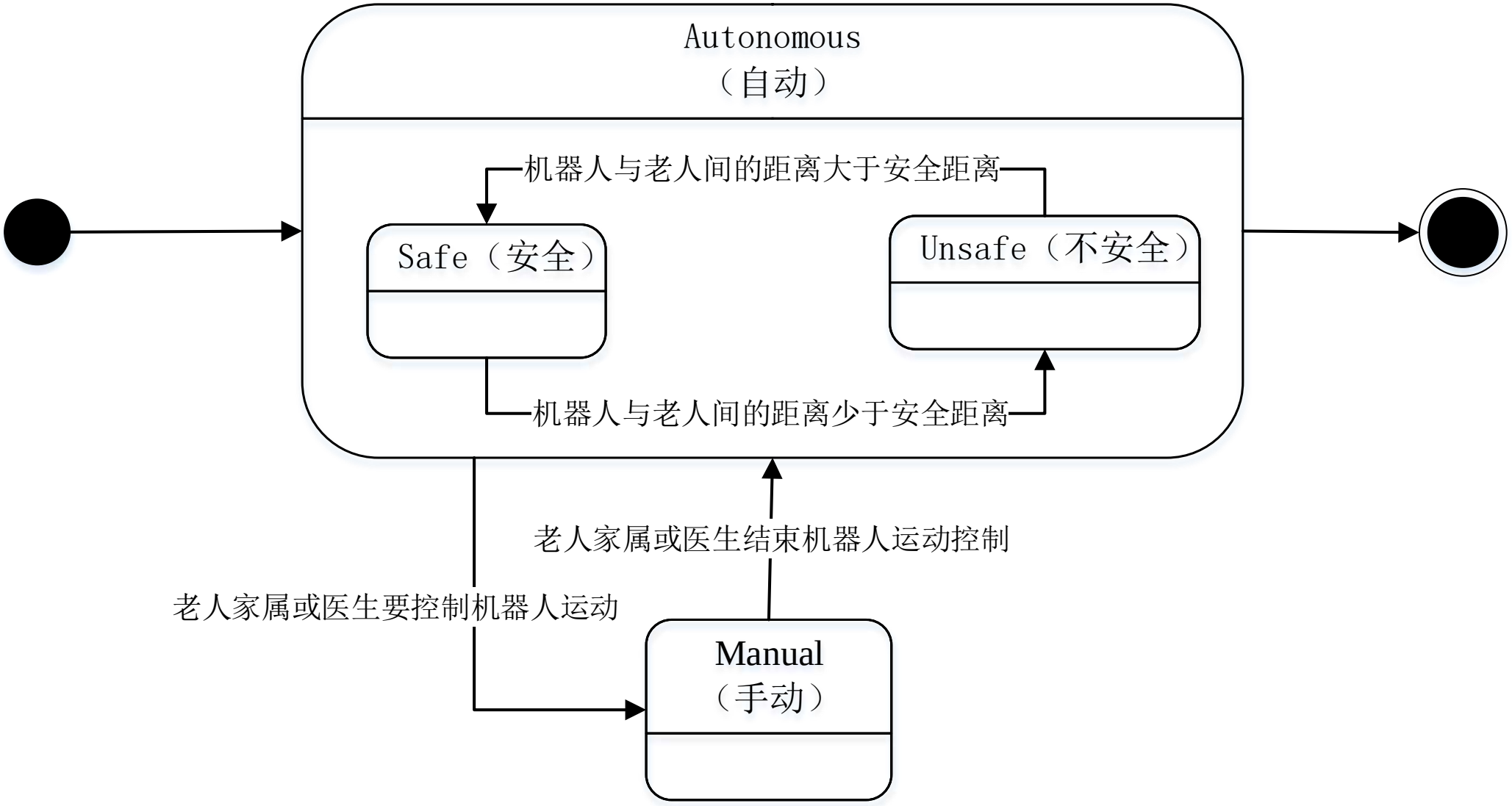


2.4 分析和建立软件需求的状态模型

- **用UML的状态图来描述这些对象的状态模型，以刻画对象拥有哪些状态、对象的状态如何受事件的影响而发生变化**

- **注意二点**
  - ✓状态模型是针对**对象**而言的，而非针对分析类
  - ✓需求工程师**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

示例: 分析机器人类对象的状态图



内容

1. 分析软件需求概述
   - ✓分析软件需求的任务
   - ✓UML描述方法
2. 分析软件需求过程
   - ✓分析和确立软件需求优先级
   - ✓分析和建立软件需求模型
3. **软件需求文档化及评审**
   - ✓软件需求规格说明书
   - ✓评审软件需求

3.2 撰写软件需求文档的注意事项

- **遵循规范**，按照规范来撰写软件需求文档
- **图文并茂**，将软件需求模型以及自然语言描述二者结合在一起，给出软件需求的清晰、准确和详实的表述
- **完整表述**，要给出软件功能性需求和非功能性需求的描述
- **共同参与**，需求工程师要与用户、客户等一起参与软件需求规格说明书的撰写
- **语言简练**，软件需求规格说明书的语言表述要简练，便于阅读和理解
- **前后一致**，在软件需求文档中，对同一个软件需求的表述前后要一致，不要产生相互矛盾或不一致的需求表达

3.3 分析软件需求的输出

❑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，直观地描述了软件的功能性需求，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

3.1 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

软件需求模型是文档中的重要组成成分

3.4 软件需求评审的步骤

- 阅读和汇报软件需求制品
- 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（1/2）

- **内容完整性**，是否包含了用户和客户的所有软件需求
- **内容正确性**，软件需求是否客观、正确地反映了用户和客户的实际期望和要求
- **内容准确性**，是否准确地反映了用户和客户的期望和要求
- **内容一致性**，所描述的软件需求是否存在不一致问题
- **内容多余性**，所描述的软件需求是否都是用户所期望的
- **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（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

3.5 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

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提供的、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
- **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