



西安电子科技大学
XIDIAN UNIVERSITY

软件详细设计





内容

1. 软件详细设计概述

- ✓任务、过程和原则
- ✓详细设计的UML模型

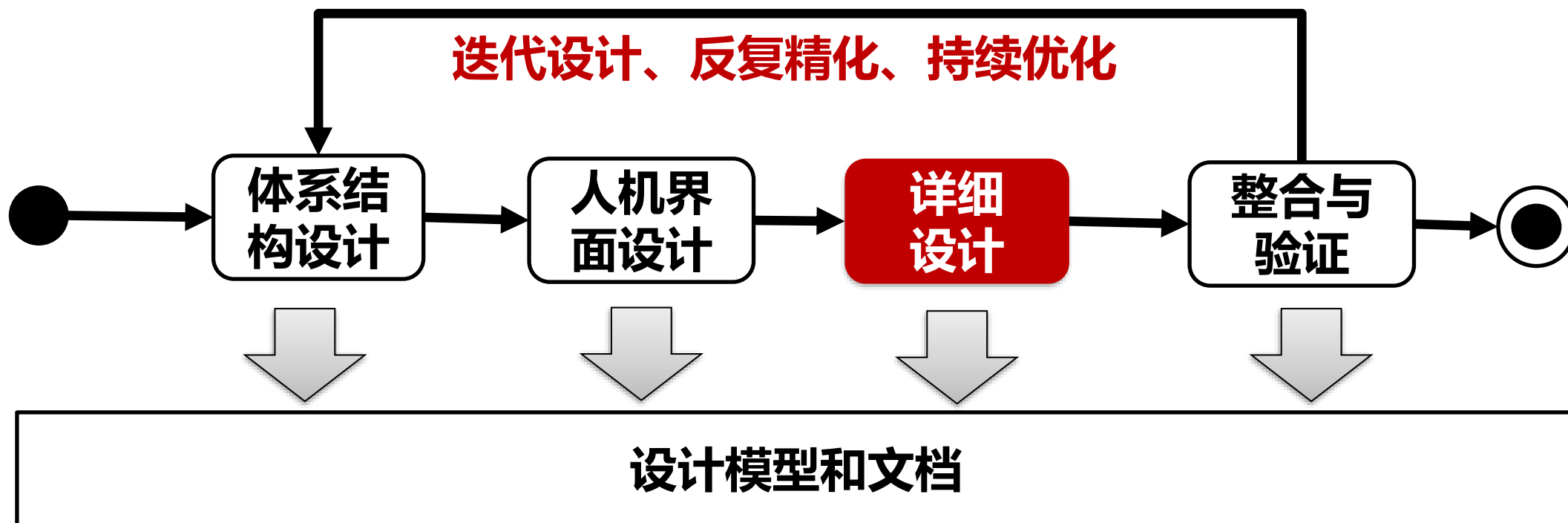
2. 软件详细设计活动

- ✓用例设计
- ✓类设计
- ✓数据设计
- ✓子系统和构件设计

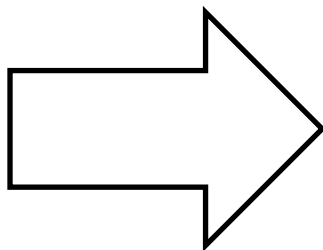
3. 详细设计文档化和评审



1.1 软件设计过程及关注点的变化



□ 结构性
□ 全局性
□ 关键性
□ 粗粒度



□ 过程性
□ 局部性
□ 细节性
□ 细粒度

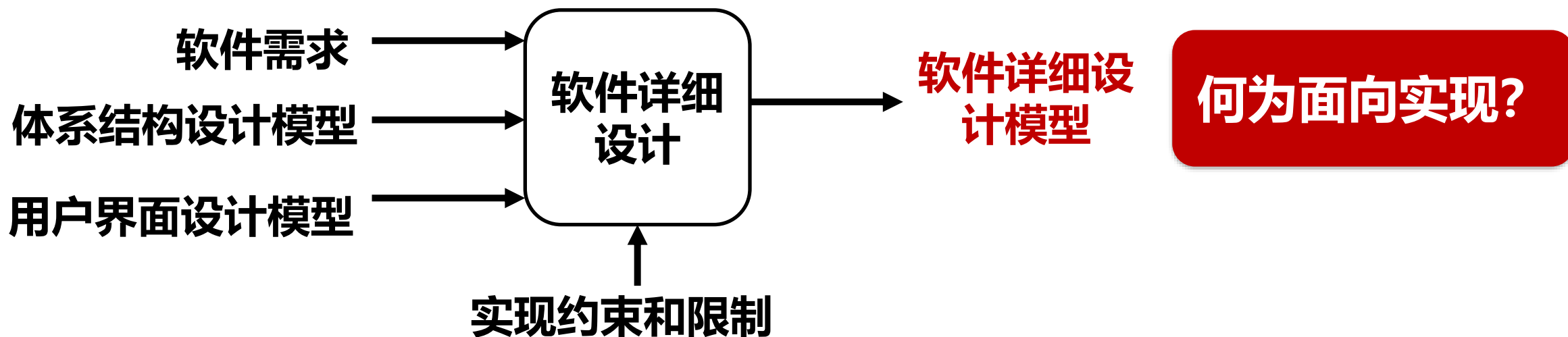
1.2 详细设计的任务

□**输入**：软件体系结构设计、用户界面设计、软件需求

□**任务**：对体系结构设计和用户界面设计成果进行**细化和精化**，获得高质量、面向实现的**设计模型**

✓精化和细化的具体对象：**子系统、构件、关键设计类和界面类**

✓不仅要提供设计元素的内部**结构性信息**，还要提供设计元素的**过程性信息**



详细设计是高层设计和底层实现间的桥梁

落实和细化

辅助和支持

高层软件体
系结构设计

软件详细
设计

软件
实现



新客网 xker.com



详细设计的定位

□ 体系结构与软件实现间的“桥梁”

- ✓ 基于体系结构设计和用户界面设计
- ✓ 后续程序设计的基础和依据

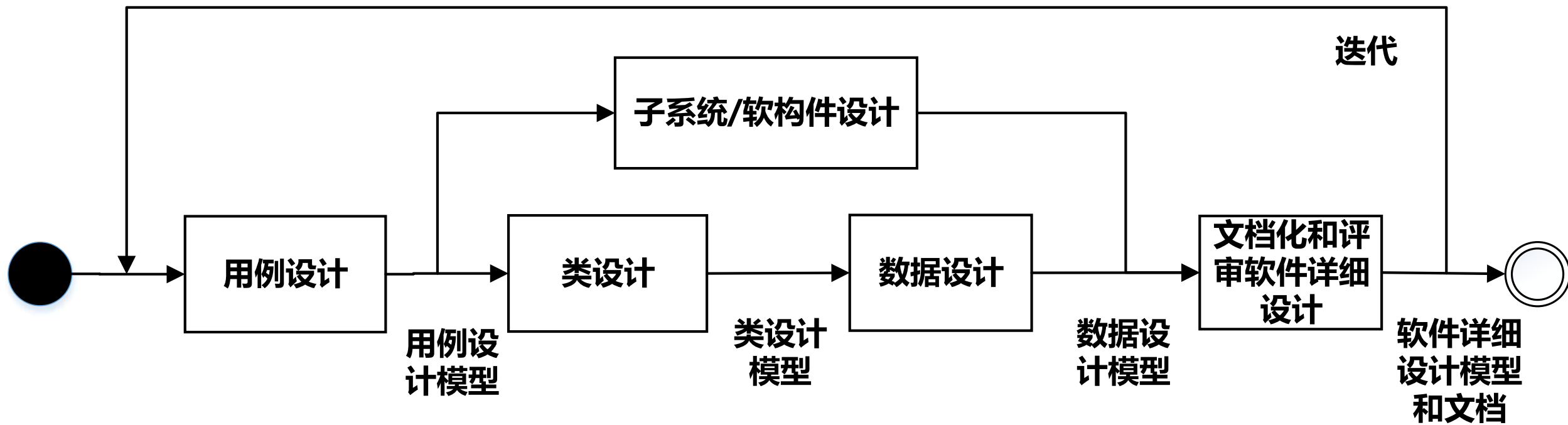
□ 确保体系结构设计得到落实的“关键”

- ✓ 基于体系结构设计和用户界面设计，关注于体系结构中设计元素的**内部细节设计**

□ 更加关注软件的“实现”

- ✓ 与体系结构设计相比较，详细设计抽象层次更**低**、粒度更**小**、更加关注实现**细节**

1.3 详细设计过程





软件详细设计的活动 (1/2)

□用例设计

- ✓ 给出**用例的具体实现解决方案**，描述用例是如何通过各个**设计元素**（包括子系统、软构件、设计类等）的交互和协作来完成的，既是对软件需求实现的刻画，也可用来评估软件体系结构设计和用户界面设计的合理性

□类设计

- ✓ 给出每个设计类的**具体实现细节**，包括类的属性定义、方法的实现算法等，使得程序员能够基于类设计给出这些类的实现代码



软件详细设计的活动 (2/2)

□数据设计

- ✓对软件所涉及的**持久数据及其操作**进行设计，明确持久数据的存储方式和格式，细化数据操作的实现细节（修删改查等）

□子系统/软构件设计

- ✓针对粗粒度的子系统和软构件，给出其**细粒度的设计元素**，如子系统、设计类等，明确这些设计元素之间的协作关系，使得它们能够实现子系统/软构件接口所规定的相关功能和服务



1.4 软件详细设计的要求和原则

□ 针对软件需求

- ✓ **从软件需求出发**，确保每一项软件需求都有相应的详细设计元素加以实现

□ 深入优化设计

- ✓ 精心设计，以充分优化软件系统的**性能、效能**等，提高软件**系统的可靠性、可重用性和可维护性**等

□ 设计足够详细

- ✓ 得到**详实程度**足以支持程序员编码的软件设计模型

□ 充分软件重用

- ✓ 从多个**不同的维度和层次**进行充分的软件重用，以提高软件开发的效率和**质量**，**减低开发成本**

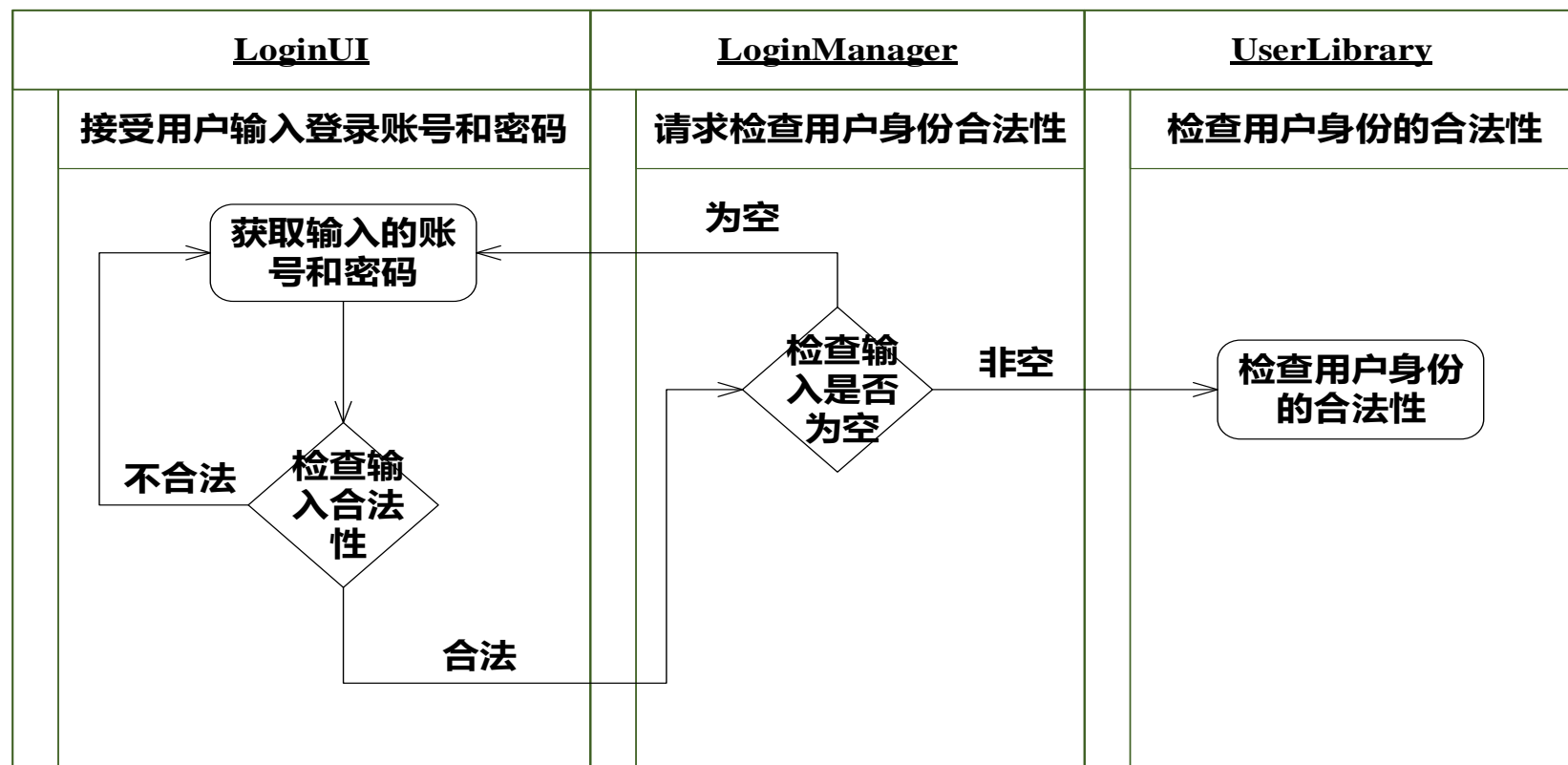
详细设计-活动图

| 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) | 描述各实体如何通过操作序列完成实例 |
| | 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 |

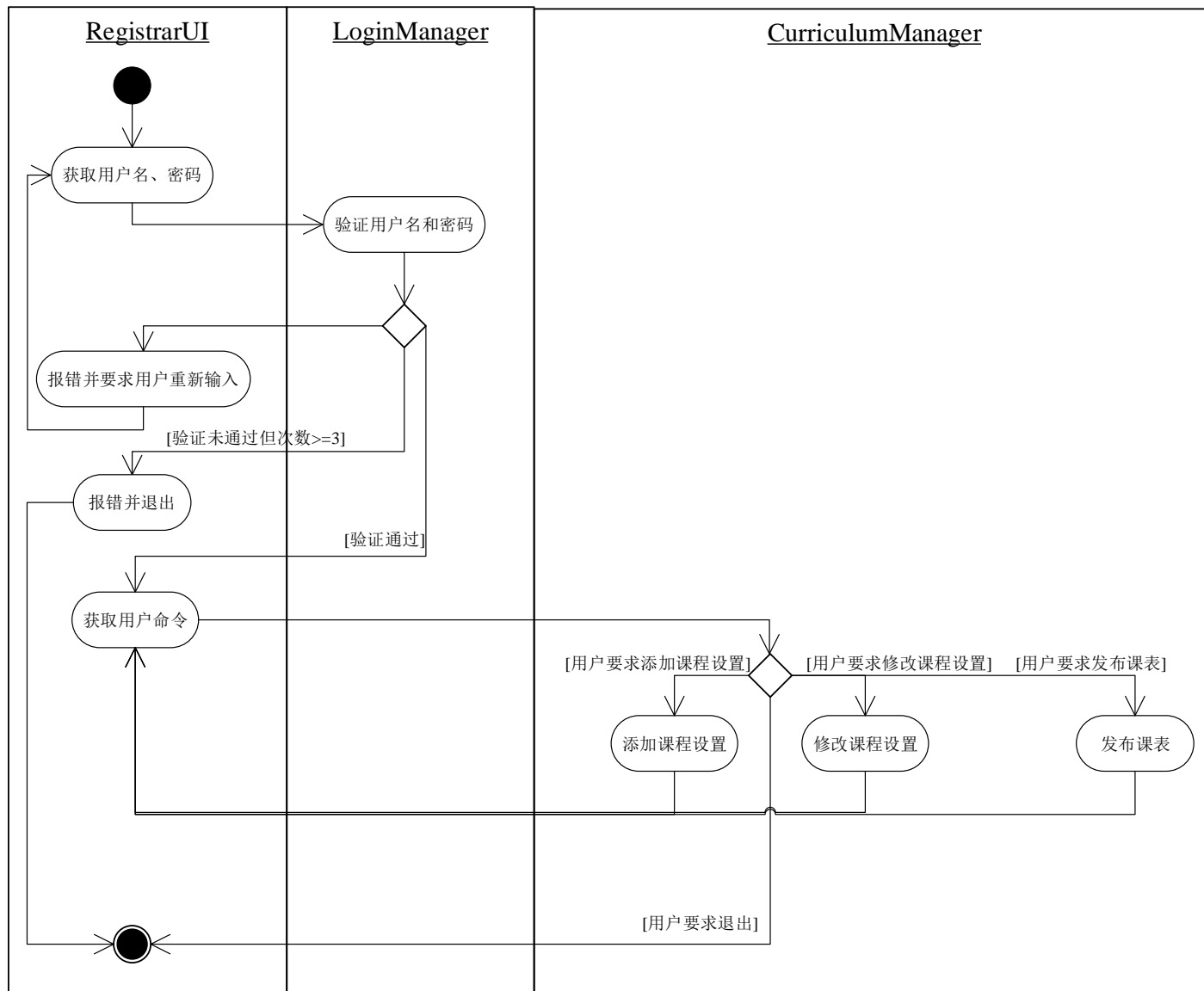
1.5 何为活动图

□活动图引入了泳道机制表示活动的并发执行，将活动图分割成各个活动分区，每个区域由一个对象或一个控制线程负责

✓这一机制可以清晰地表示对象或线程的职责、它们之间的并发、协同和同步



示例：活动图



活动图的构成

□ **活动点**：表示计算过程，用矩形框表示

□ **决策点**：根据条件进行活动决策，用菱形框表示

□ **边**：表示控制流或信息流

- ✓ 控制流：表示一个操作完成后对其后续操作的触发
- ✓ 信息流：刻画操作间的信息交换

□ **并发控制**

- ✓ 控制流经此节点后分叉（Fork）成多条可并行执行的控制流，或多条并行控制流经此节点后同步合并（Join）为单条控制流

□ **泳道**

- ✓ 将活动图用形如游泳池中的泳道分隔成数个活动分区，每个区域由一个对象或一个控制线程负责

泳道

- 将活动图用形如游泳池中的泳道分隔成数个活动分区，每个区域由一个**对象或一个控制线程**负责
- 每个活动节点应位于负责执行该活动的对象或线程所在的区域内
- 带泳道的活动图更清晰地表示了对象或线程的职责、它们之间的分工、协同和同步

活动图的绘制原则

- 从决策点出发的每条边上均应标注条件，且这些条件必须覆盖完整且互不重叠
- 必须确保分叉和汇合节点之间的匹配性
 - ✓ 对任一分叉节点，其导致的并发控制流必须最终经由一个汇合节点进行控制流的同步和合并
- 一张活动图可以浓缩成另一活动图中的单个活动节点，前者称为子活动图，后者称为父活动图



内容

1. 软件详细设计概述

- ✓任务、过程和原则
- ✓详细设计的UML模型

2. 软件详细设计活动

- ✓2.1 用例设计
- ✓2.2 类设计
- ✓2.3 数据设计
- ✓2.4 子系统和构件设计

3. 详细设计文档化和评审





需求用例及其实现

□用例反映了**软件需求**，基于需求来指导和细化设计

- ✓在需求分析阶段，为各用例建立了用例交互模型，分析了业务逻辑的实施流程

□用例设计要明确需求是如何实现的

- ✓基于用例交互模型开展用例设计，考虑用例实现的解决方案

2.1 用例设计的任务

□任务

- ✓针对需求分析模型中的每个用例，基于体系结构和用户界面设计模型给出的设计元素，**设计用例的软件实现方案**

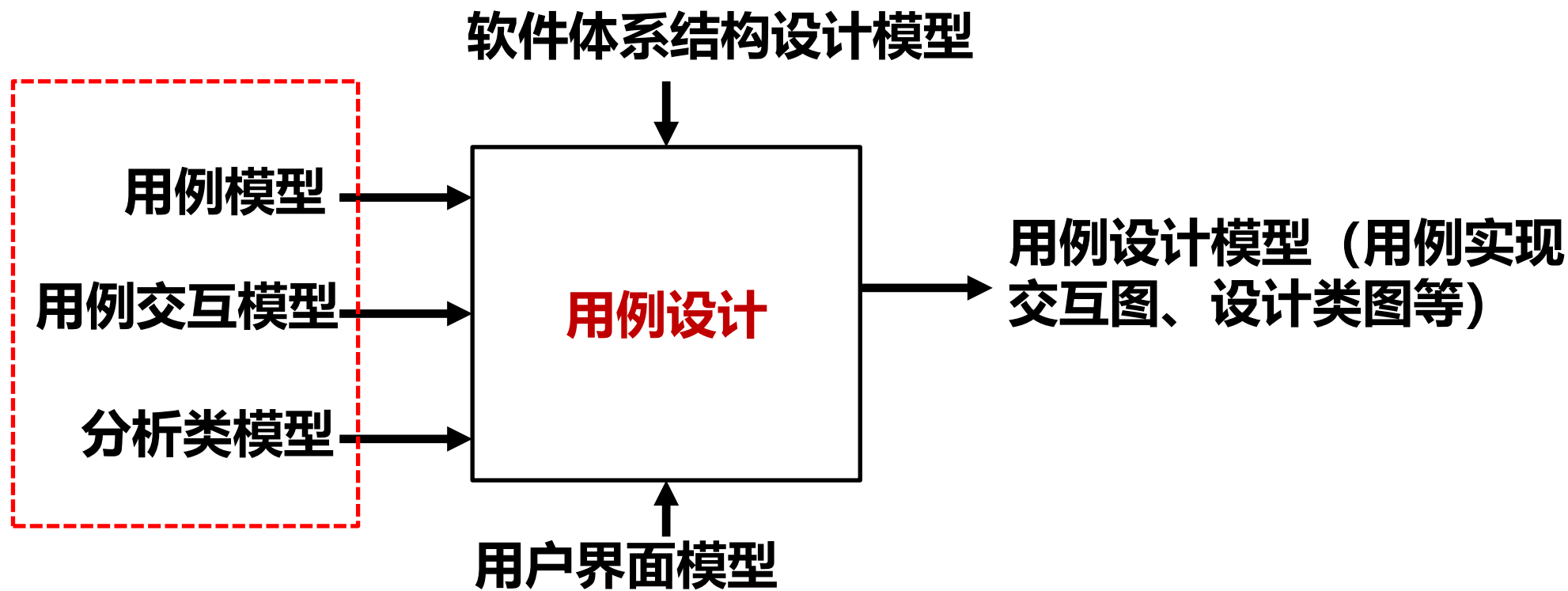
□设计和建模

- ✓各个设计元素如何通过协作完成用例
- ✓关联各个设计元素，包括**子系统、构件、对象类、界面**等
- ✓产生详细的设计信息，如**新的设计元素、细节、关联**

□结果

- ✓描述用例设计的顺序图、类图等设计模型

用例设计的任务





用例设计原则

□ 以软件需求为基础

- ✓ 以需求模型为前提，包括用例模型、交互模型、分析类模型等
- ✓ **不能抛开软件需求来给出用例的实现方案**

□ 通过整合设计元素来实现用例

- ✓ 整合软件体系结构设计、用户界面设计等产生了一系列设计元素，包括：子系统、构件、关键设计类等等
- ✓ **不能抛开前期设计工作成果**

□ 精化软件设计

- ✓ 用例设计不仅要给出用例实现的解决方案，也要依此为目的进一步精化软件设计，以获得更为详实的设计信息
- ✓ **为后续的详细设计奠定基础**

□ 多视角的设计

- ✓ 用例设计不仅要从小用例实现的视角给出软件需求的解决方案，还要从系统全局的视角给出目标软件系统的设计类图

用例的实现方案

□用例如何通过各个设计元素来实现的

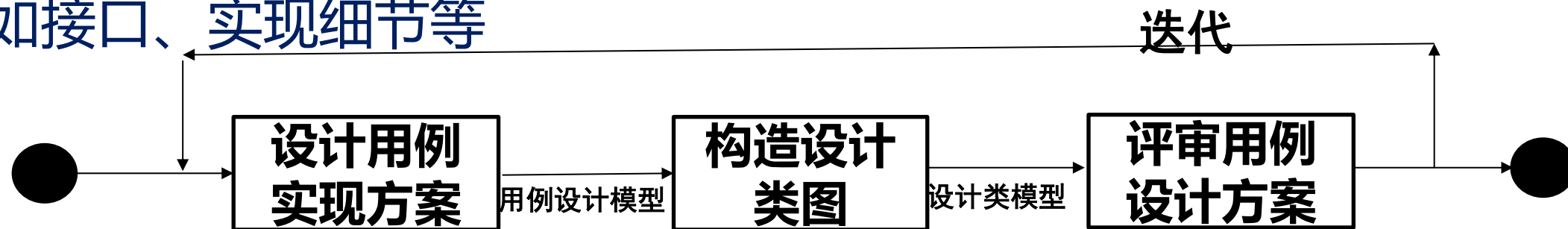
✓子系统、构件、设计类、界面元素类等

□这些设计元素之间在用例实现过程中的协同和交互

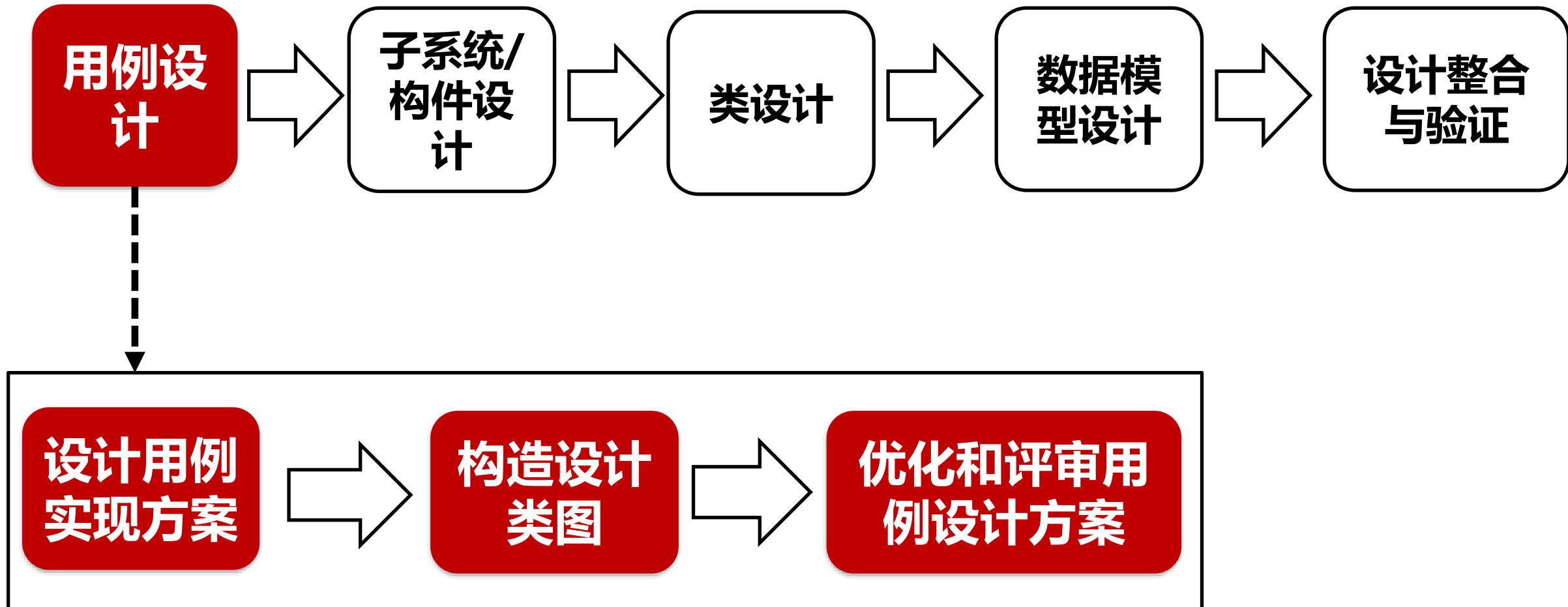
✓消息传递

□精化各个设计元素的设计

✓如接口、实现细节等



用例设计的过程



2.1.1 设计用例实现方案

□基础

- ✓ 分析每个用例的UML交互图，它是开展用例设计的依据
- ✓ 考虑体系结构和用户界面设计要素，它们是用例实现参与者
- ✓ 明确各个设计元素（如对象类、构件等）的职责

□方法

- ✓ 在分析阶段用例交互图的基础上，将交互图的分析类转化为用例实现的设计类，同时引入体系结构设计和用户界面设计所生成的设计元素，共同形成关于用例实现的交互模型

□结果

- ✓ 生成用例实现的顺序图
- ✓ 引入更多的设计元素，如构件、子系统、类等
- ✓ 精化更多的设计细节，如接口、方法、交互等

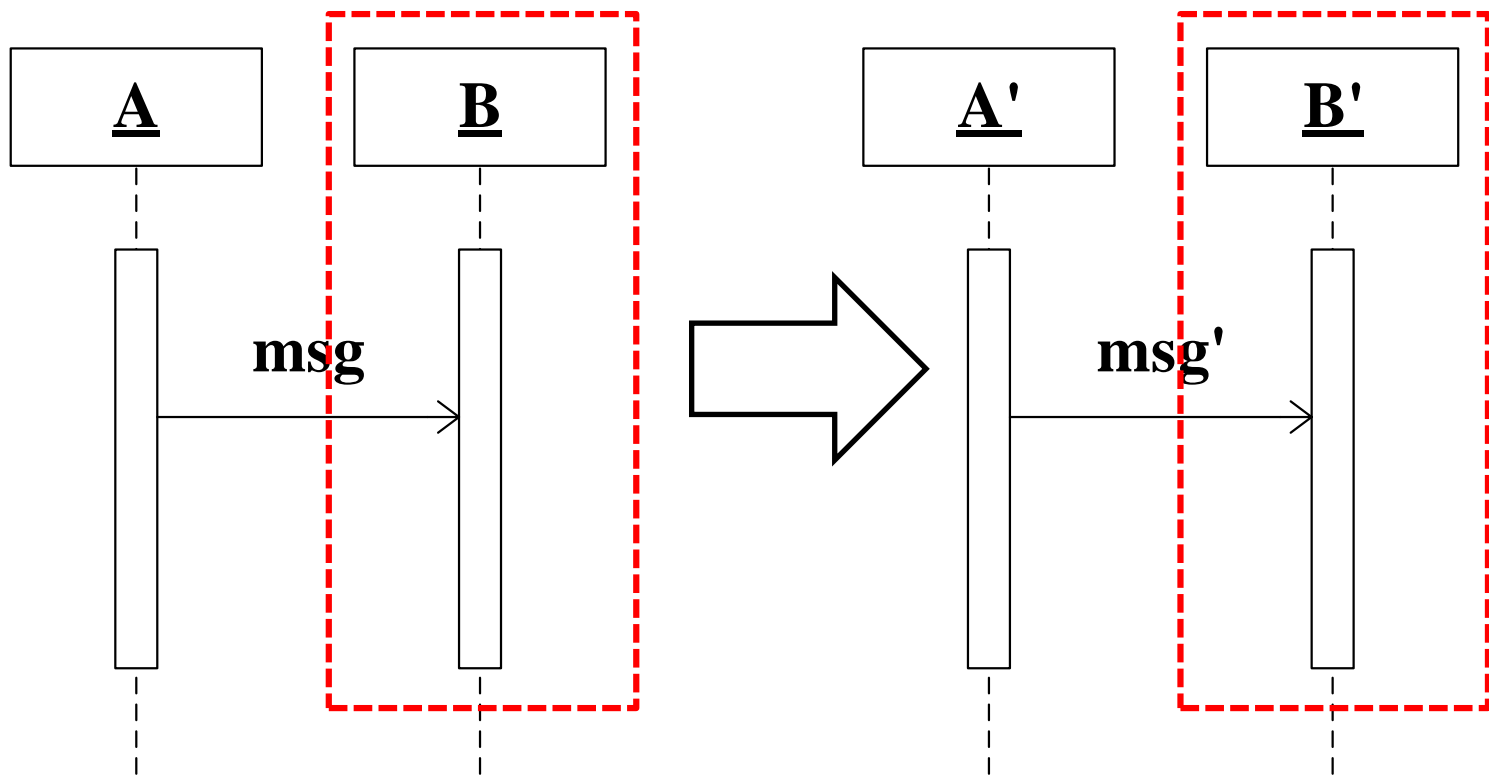
用例设计——分析类转换为设计类

□用例设计在分析阶段用例交互图的基础上，对交互图的分析类进行两方面的处理：

- ✓如果分析类对应于体系结构设计和用户界面设计的设计元素，用该设计元素替代分析类；
- ✓如果分析类在体系结构设计和用户界面设计的设计元素找不到对应物，将该分析类转化为设计类

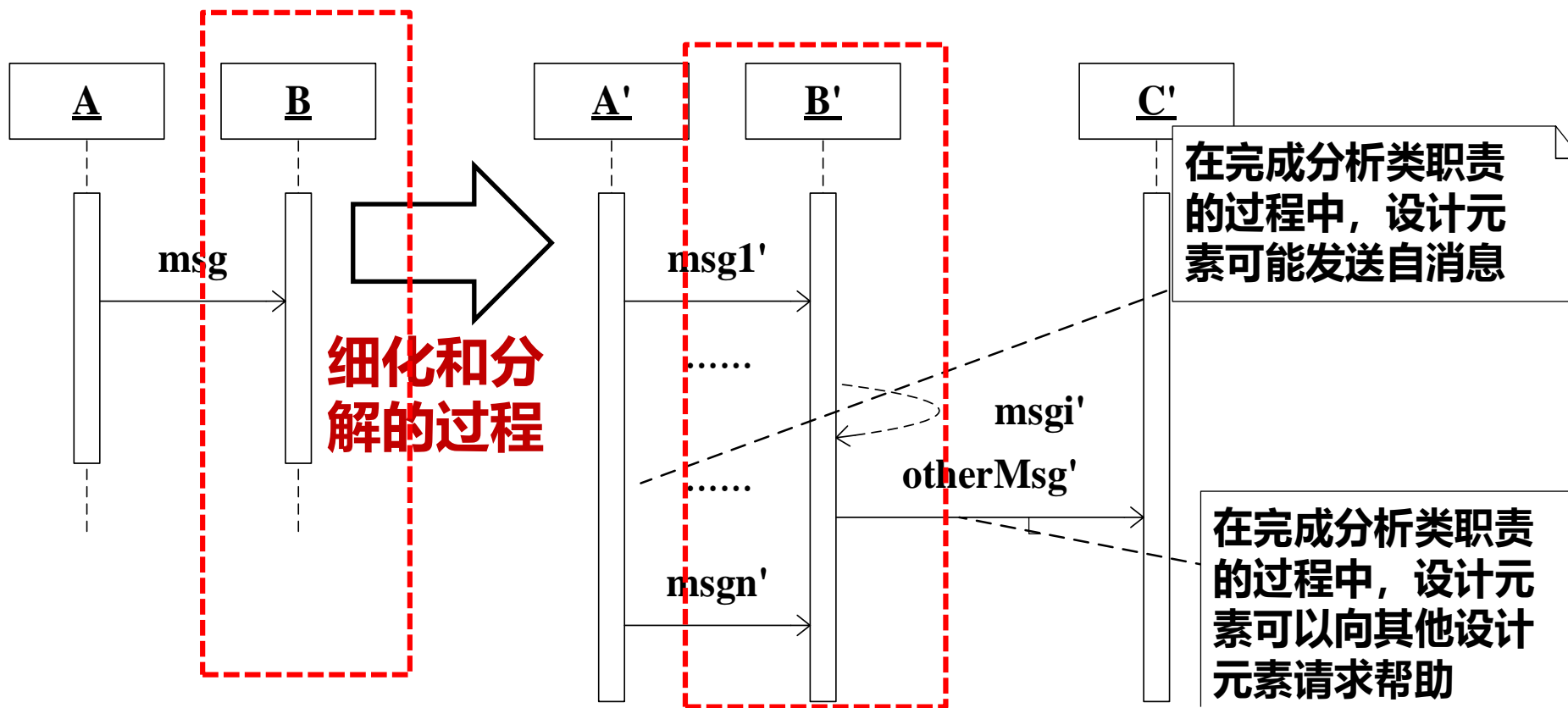
分析类与设计元素间对应关系(1/3)

□ 一个分析类的一项职责由一个设计元素的单项操作完整地实现



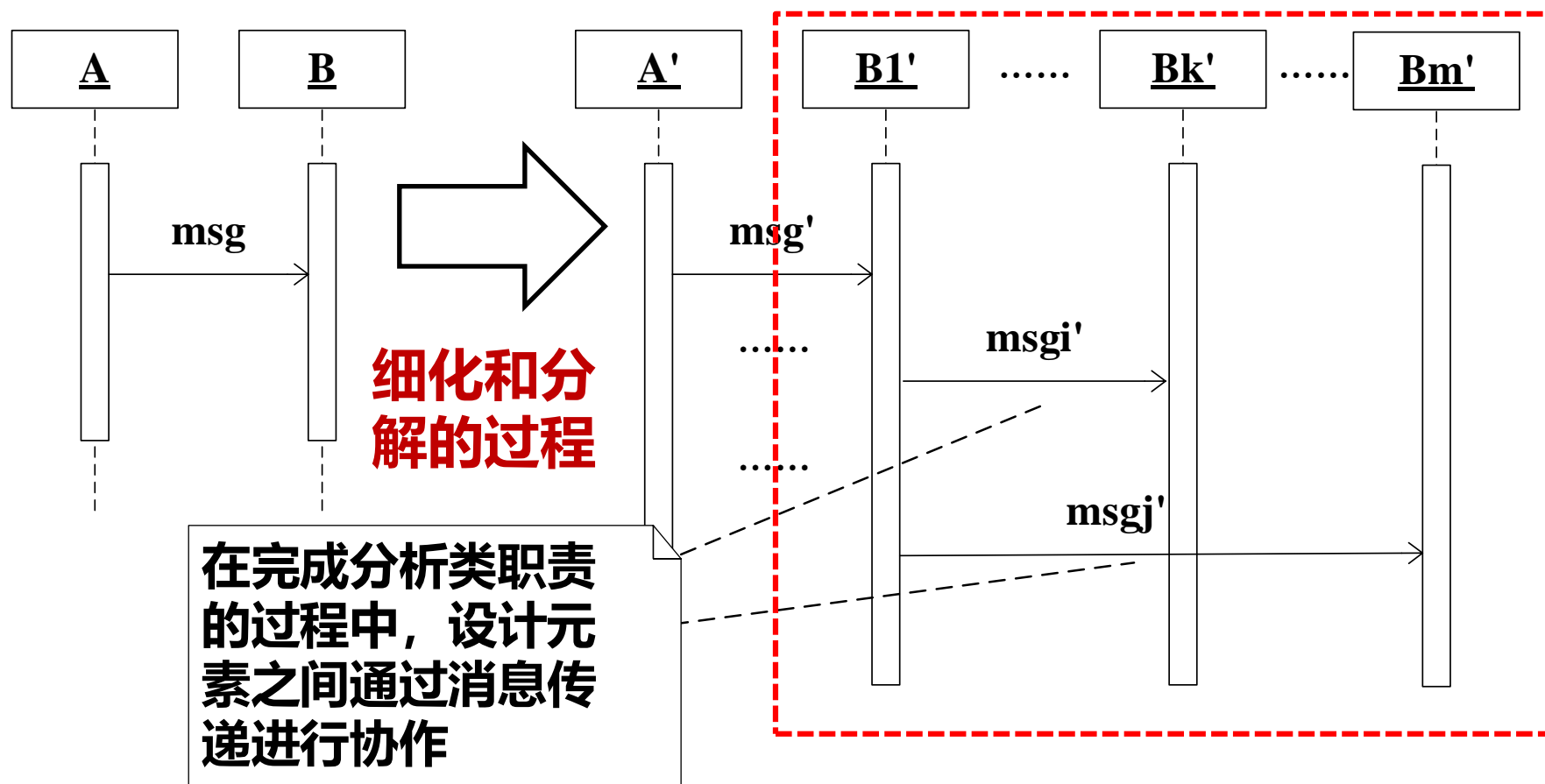
分析类与设计元素间对应关系(2/3)

□ 一个分析类的一项职责由一个设计元素的多项操作来实现

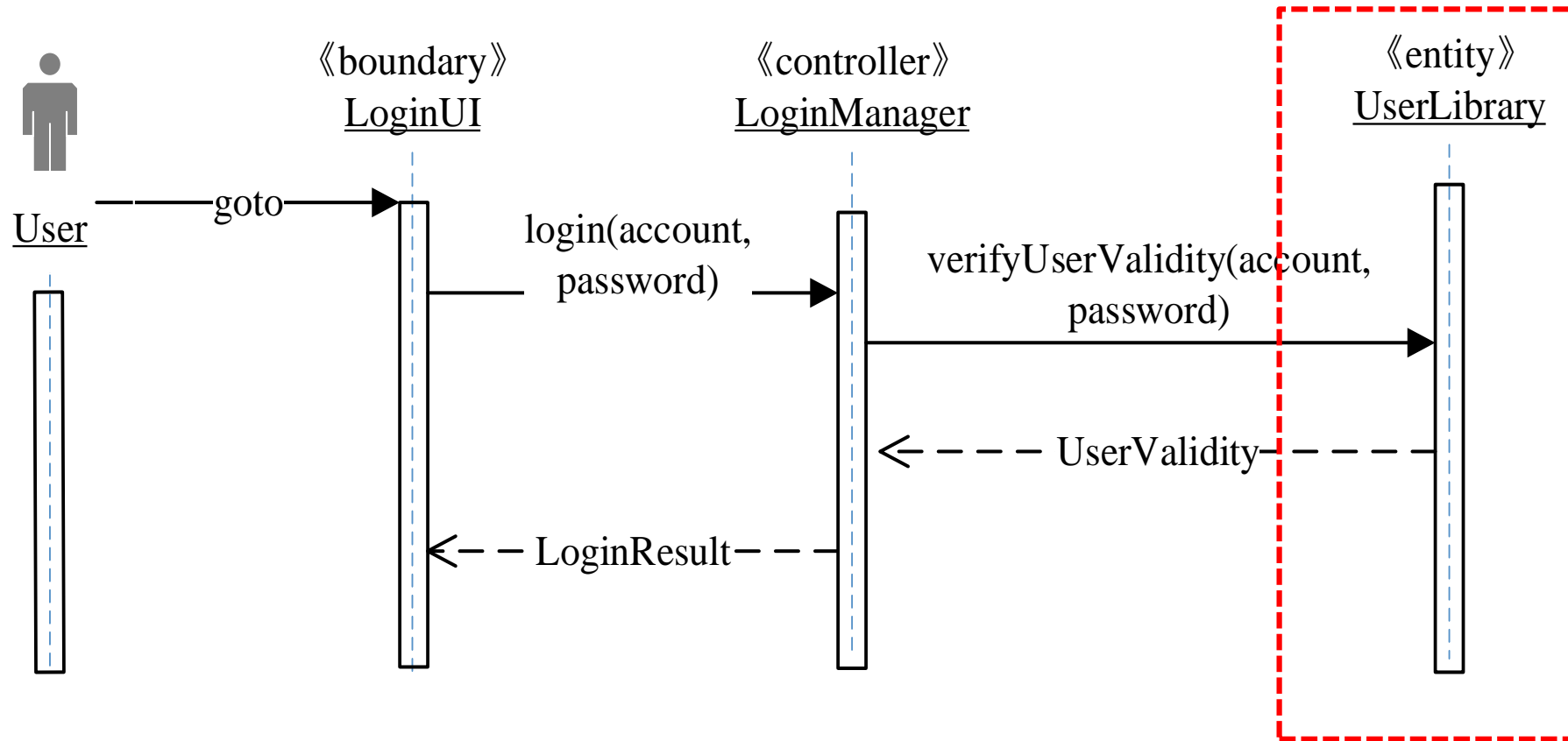


分析类与设计元素间对应关系(3/3)

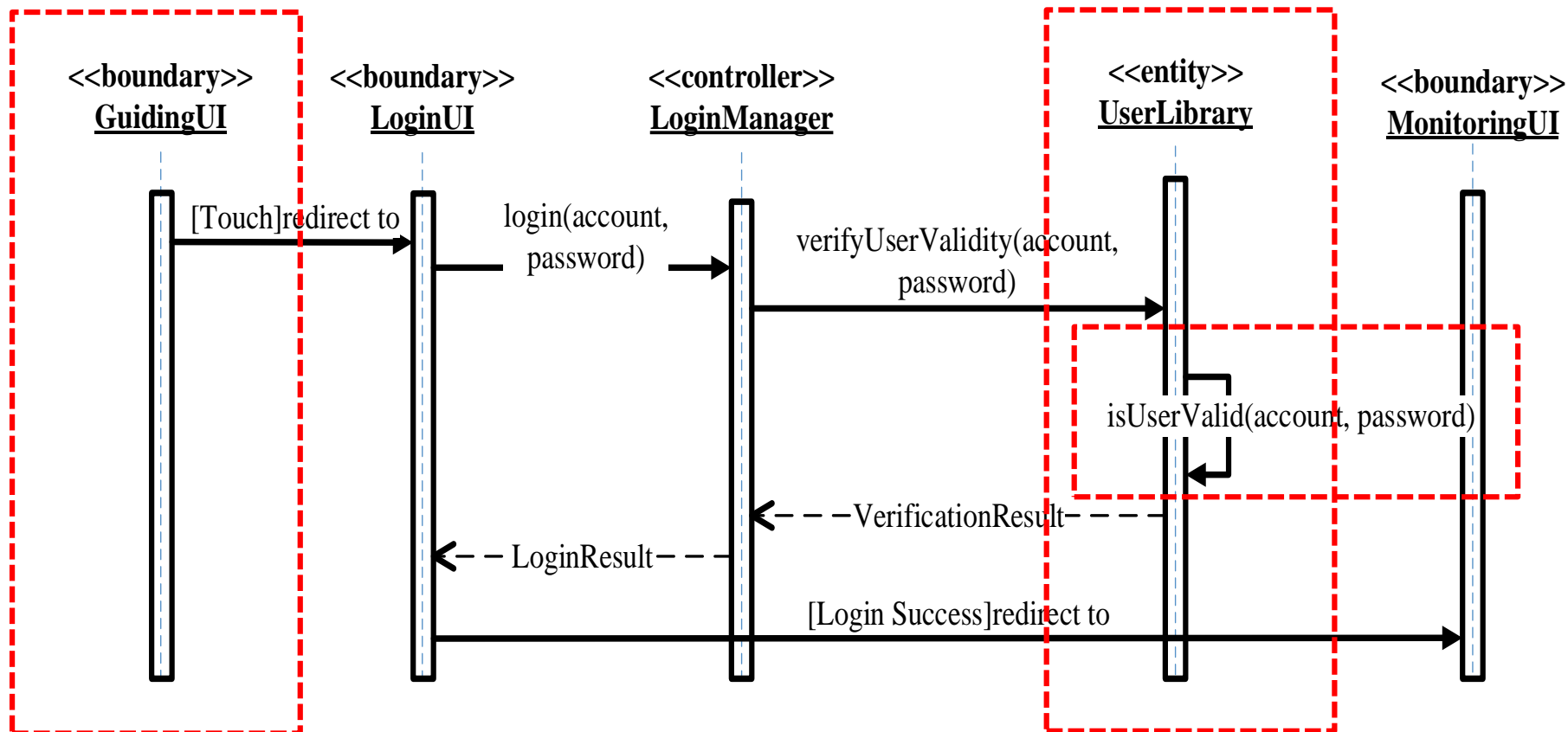
□ 一个分析类的一项职责由**多个设计元素**协同完成



示例1: “用户登录” 用例顺序图 (需求)



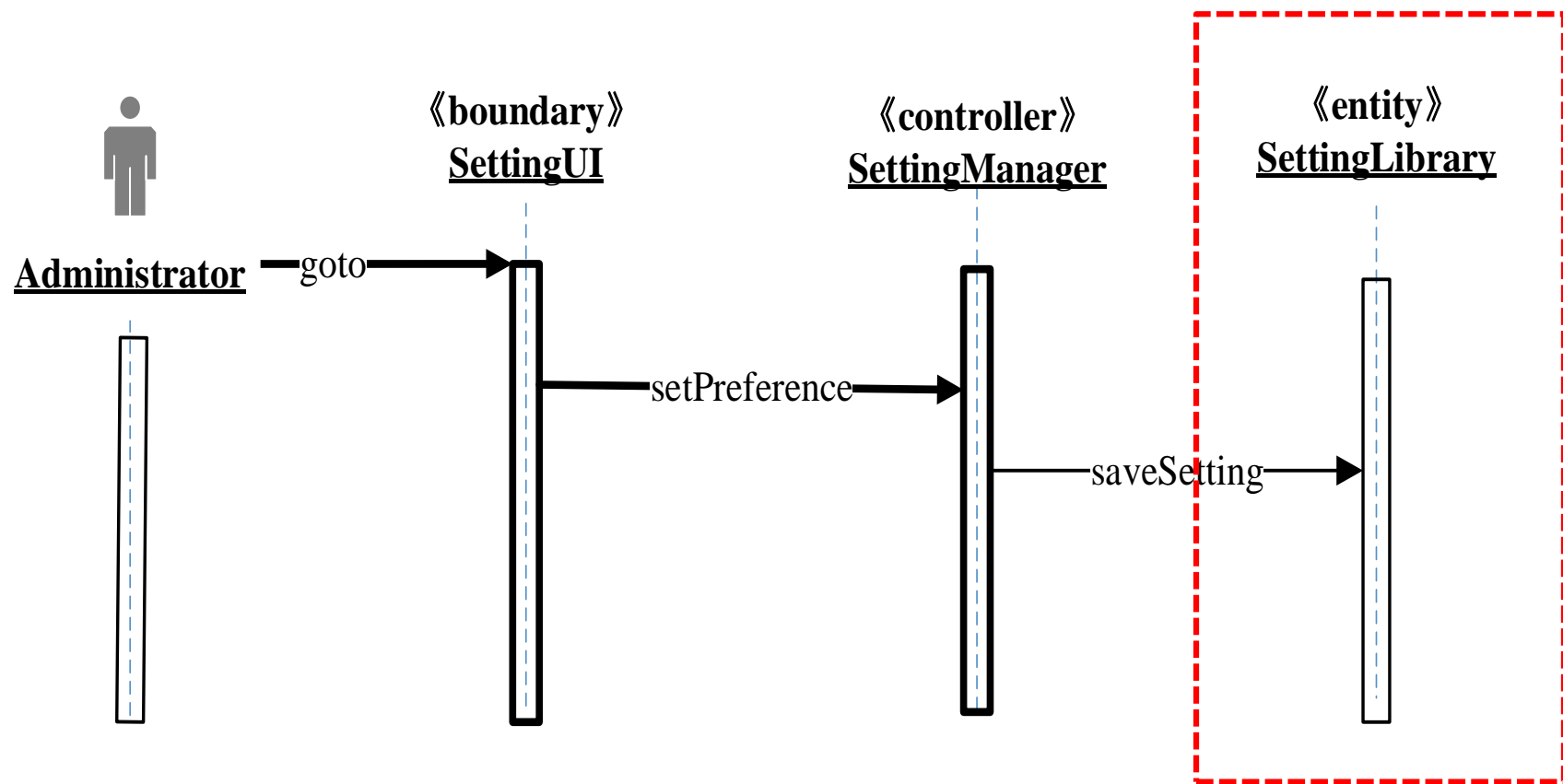
示例2: “用户登录” 用例设计方案 (设计)



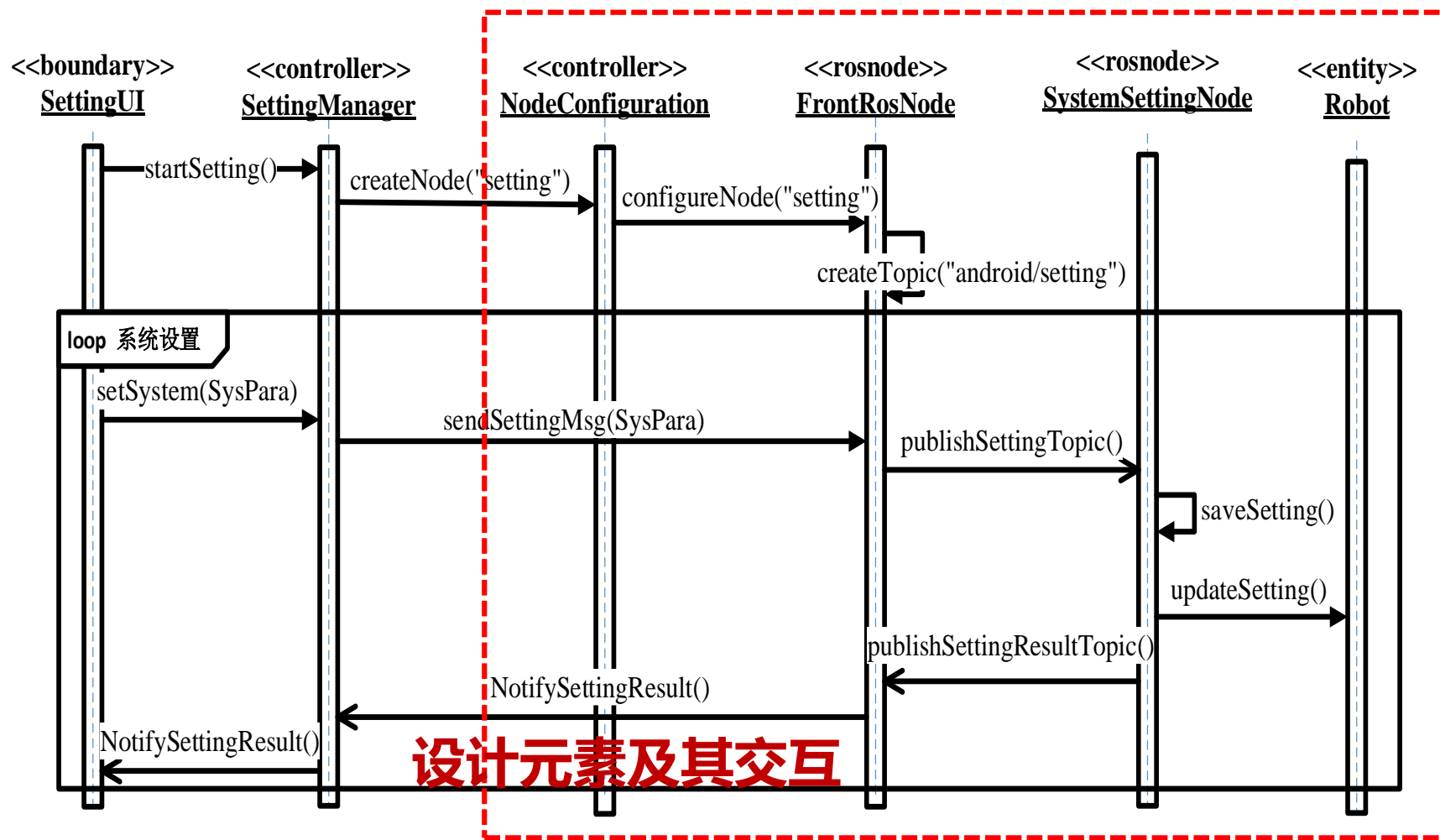
设计元素及其交互

所有设计元素最终通过代码加以实现

示例2: “系统设置” 用例的顺序图 (需求)



示例2: “系统设置” 用例设计方案 (设计)



所有设计元素最终通过代码加以实现

2.1.2 构造设计类图

□基于用例实现方案给出详细设计模型中的设计类图

- ✓类图中的设计元素可以为子系统、构件和UML类
- ✓既包括用例设计模型中相关对象所对应的类（边界类、控制类、实体类等），也包括构成软件系统的子系统、构件或新引进的设计类

□对UML类图稍作扩充以表示详细设计类图

- ✓允许在类图中出现子系统和构件，它们的类别可以采用不同的UML构造型或者不同的图元符号来表示



构造设计类图的方法

□创建初始的设计类

- ✓从分析类图、体系结构图、用例实现的顺序图寻找

□确定设计类的职责

- ✓每个设计类都有职责，取决于对交互图中消息的响应

□确定设计类间的关系

- ✓如果A与B有消息，那么它们间有关系：关联、聚合和组合

□确定设计类的属性

- ✓需要保存哪些数据项、其他对象类信息

□形成整体设计类图

- ✓模块化、职责和功能单一化原则，调整和优化



注意事项

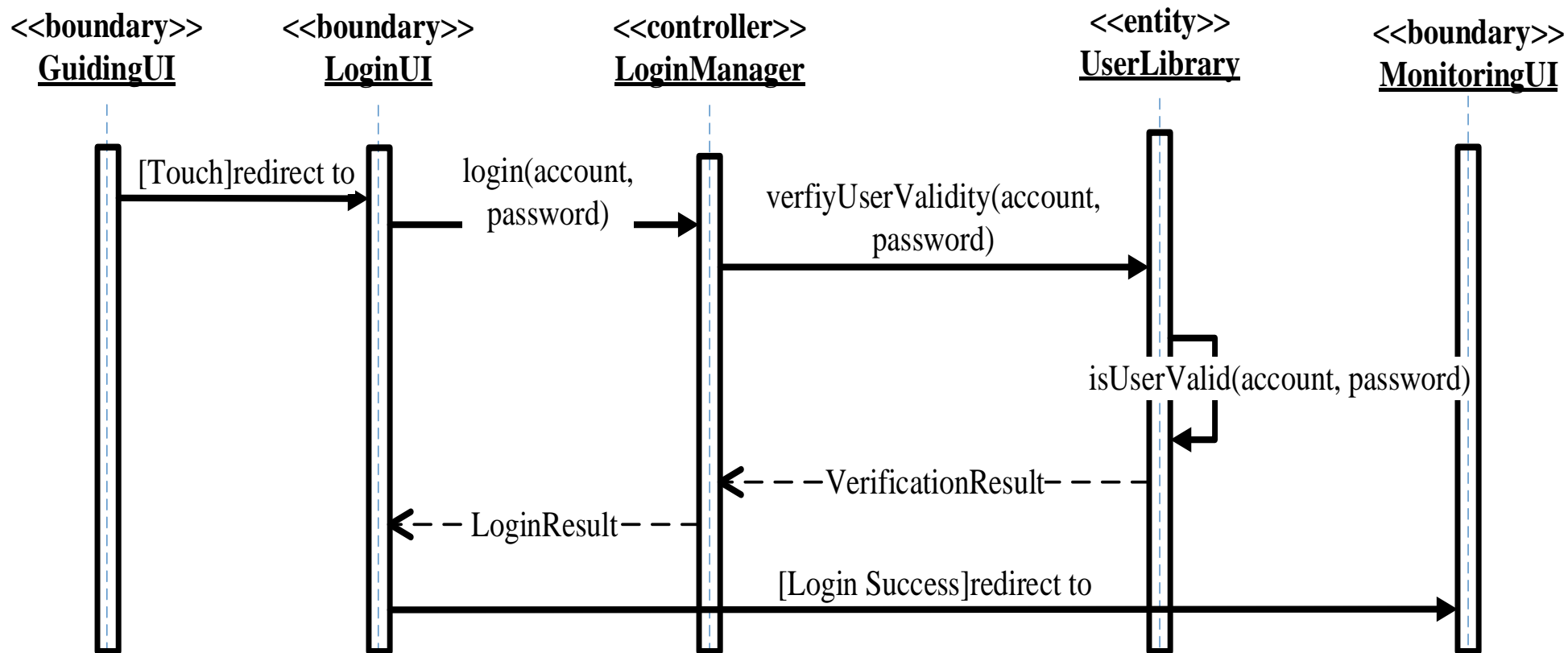
□确保设计类图与分析类图之间、设计类图与用例设计模型之间的一致性

- ✓分析类图中的类在设计类图中有相应的对应物
- ✓用例设计模型中的设计元素（主要是指参与用例实现的对象、对象间的消息传递）在设计类图中要有相应的对应物（主要是指设计类及其方法）



示例：“用户登录”用例实现的设计类图

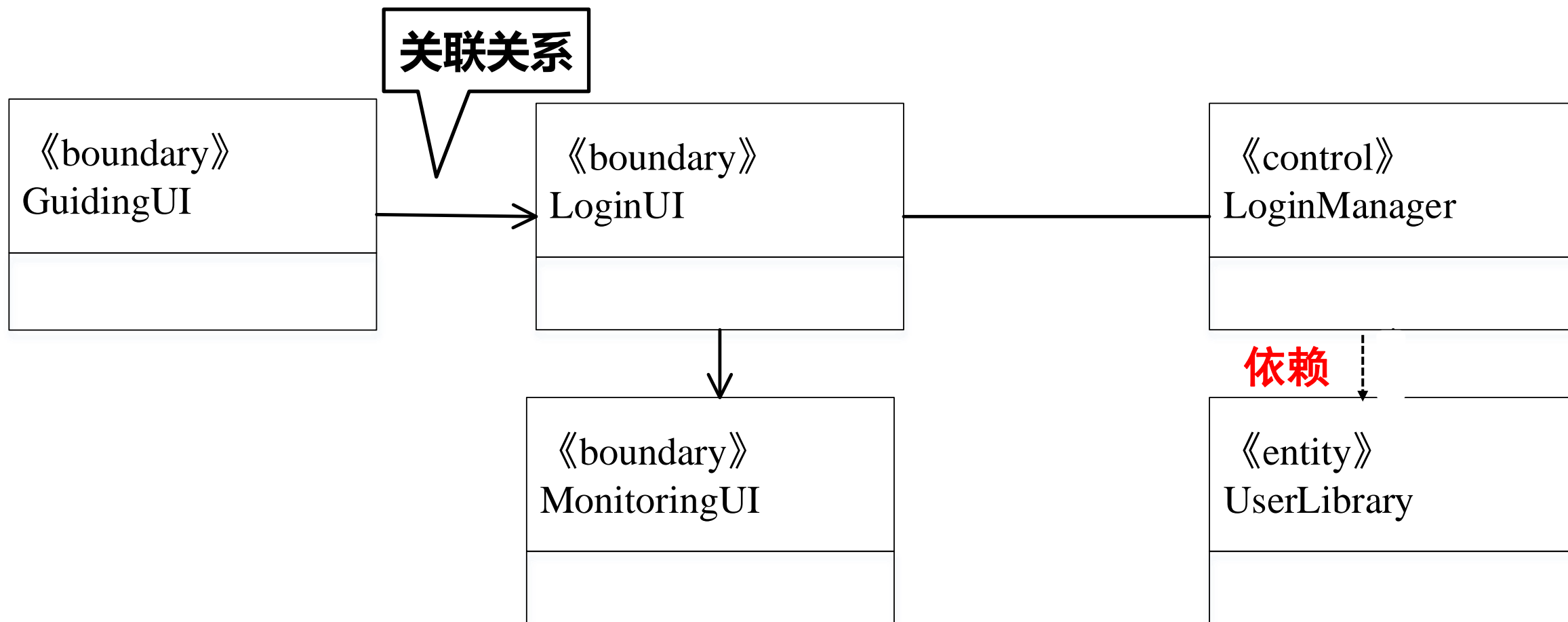
□用例设计的交互图





示例：“用户登录”用例实现的设计类图

□用例设计对应的设计类图



2.1.3 优化和评审用例设计方案

□ 尽可能重用已有软件资产来实现用例并依此构建用例实现方案

- ✓ 如开源软件、云服务、遗留系统、软件开发包等等

□ 合并设计元素

- ✓ 将具有相同或相似职责的多个设计元素（包括类、子系统和构件等）进行整合，归并为一个设计元素。
- ✓ 将设计元素（如类等）中具有相同或相似功能的多个方法整合为一个方法，以减少不必要的冗余设计。

□ 重组设计元素

- ✓ 借助继承或代理机制，对设计元素进行必要的组织和重组，以发现不同类之间的一般和特殊关系，抽象出公共的方法。
- ✓ 确保所有用例实现方案在软件系统中和谐共存，无逻辑冲突

□用例设计的顺序图

□用例设计的类图

用例设计小结

- 任务：为需求模型中每个用例设计软件实现方案
- 原则：整体性、正确性、优化性等
- 输出：用例实现方案的交互图、设计类图





内容

1. 软件详细设计概述

- ✓ 任务、过程和原则
- ✓ 详细设计的UML模型

2. 软件详细设计活动

- ✓ 2.1 用例设计
- ✓ **2.2 类设计**
- ✓ 2.3 数据设计
- ✓ 2.4 子系统和构件设计

3. 详细设计文档化和评审



类设计的任务

□任务

- ✓对**界面类、关键设计类、设计类**等进行设计优化和精化,
- ✓明确设计类的内部实现细节, 包括类的属性、方法、接口以及类对象的状态变迁
- ✓精化到可以提交软件实现的程度

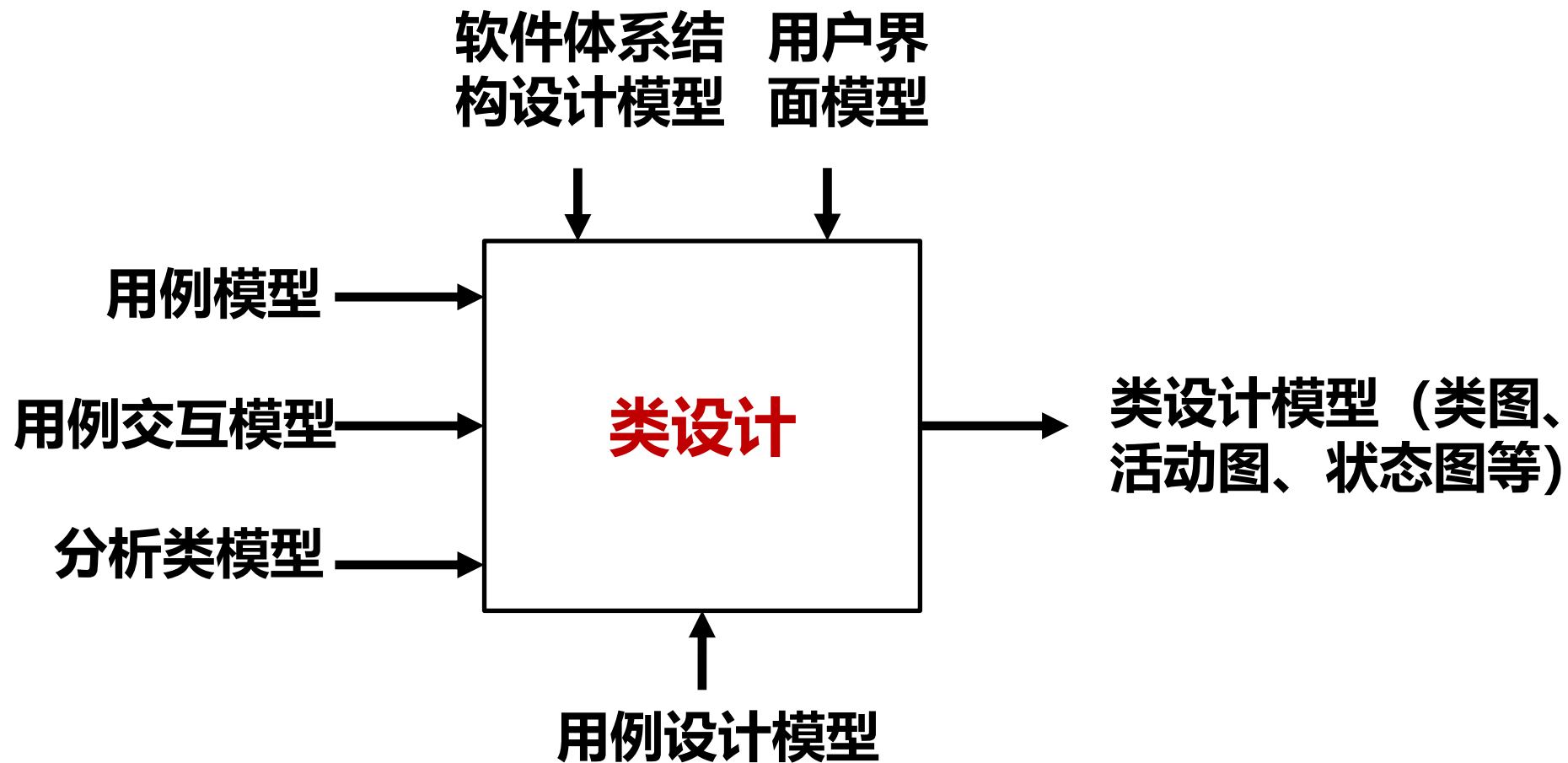
□设计与建模

- ✓明确类的**可见范围, 类的操作和属性, 类之间的关系**等
- ✓对类设计进行建模

□结果

- ✓类图、状态图、活动图等

类设计的任务



类设计的地位和作用

□ “承上”

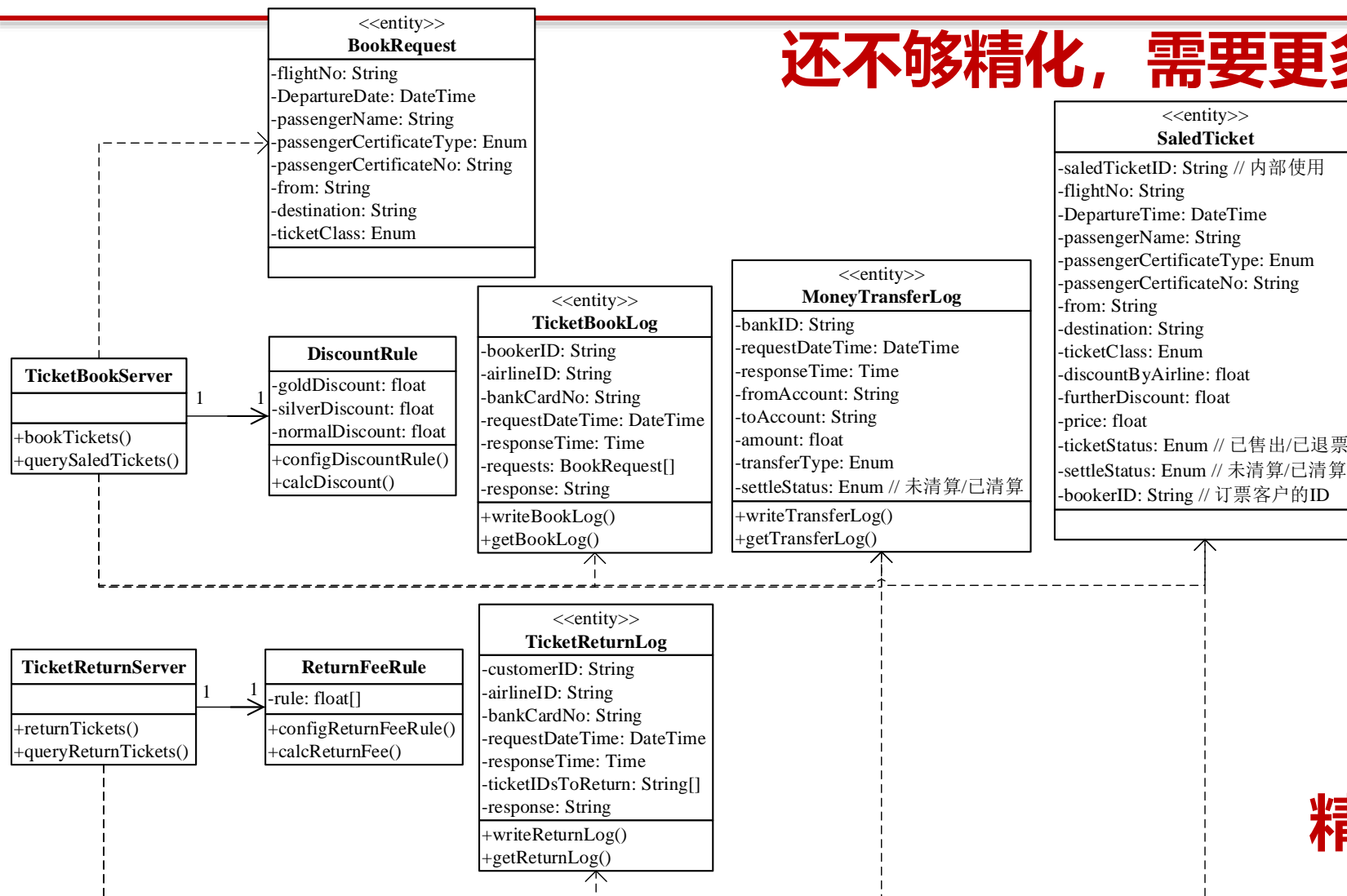
- ✓ 类设计要充分考虑软件需求，基于体系结构设计、用户界面设计、用例设计、子系统/构件设计的成果

□ “启下”

- ✓ 类设计要为后续阶段的编码和实现奠定基础，需要产生足够详细的设计结果

类设计示例

还不够精化，需要更多的详细设计模型



精化类图

基于该类图，程序员指导如何进行编码吗？



- 类的可见范围
- 属性的数据类型
- 方法的实现算法
- 类间关系
- 状态变化或者活动情况

类设计原则

□ 准确化

- ✓ 对类的内部结构、行为等给予准确的表达，以支持程序员精准地理解类设计，进而编写出类的程序代码

□ 细节化

- ✓ 对类的接口、属性、方法等方面给予足够详细的设计，以便程序员能够对类进行编程

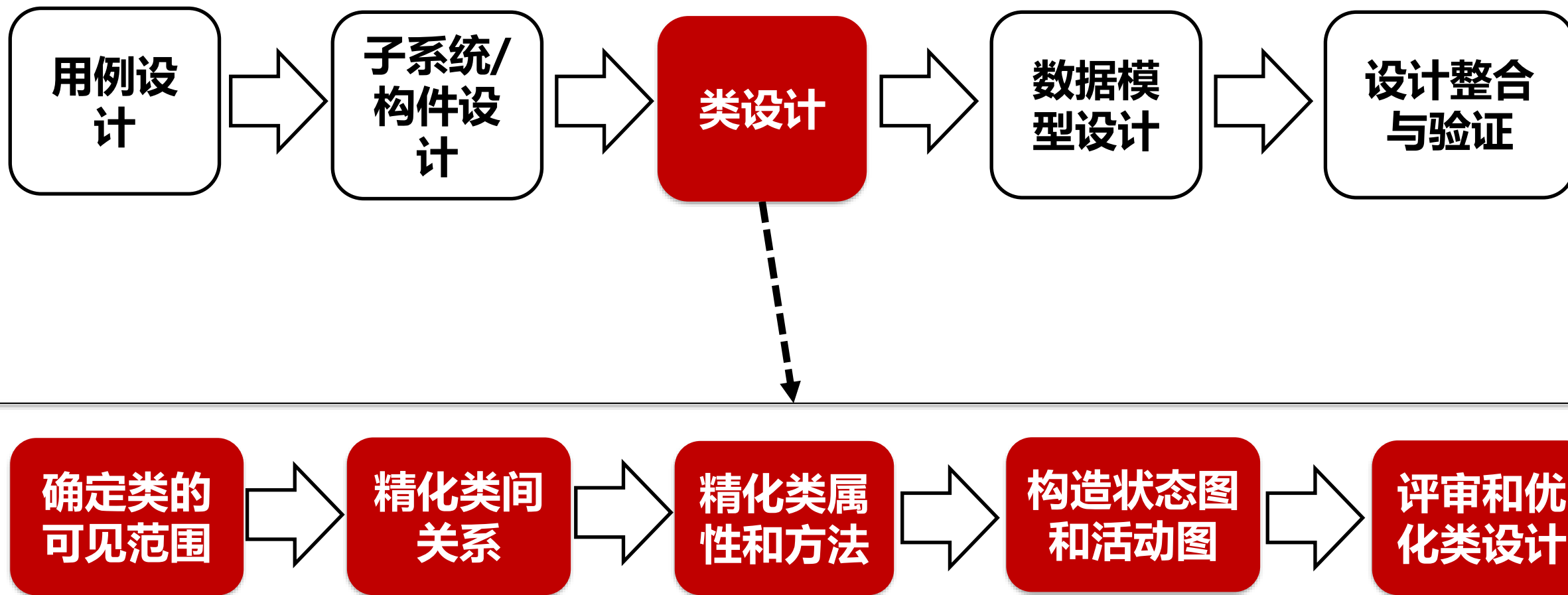
□ 一致性

- ✓ 要确保类的关系、属性、方法等的设计是相互一致的，类的内部属性、方法等设计与类的职责、关系等是相互一致的

□ 遵循软件设计的基本原则

- ✓ 按照模块化、高内聚度、低耦合度、信息隐藏等基本原则来进行类设计，必要时需要基于这些原则对所设计的类进行必要的拆分和合并，以提高类设计的质量

类设计过程



2.2.1 确定类的可见范围

□可见范围：确定类在什么范围内可为其他类访问

- ✓如果类仅仅被其所在的包所使用，那么该类是“私有的”，否则是“公开的”

□原则

- ✓尽量缩小类的可见范围，除非确有必要，否则应将类“隐藏”于包的内部

- **public**: 公开级范围，软件系统中所有包中的类均可见和可访问该类
- **protected**: 保护级范围，只对其所在包中的类以及该类的子类可见和访问
- **private**: 私有级范围，只对其所在包中的类可见和访问

为什么要缩小类的可见范围？



2.2.2 精化类间的关系

□明确类间的语义关系

- ✓ 类间关系的语义强度从高到低依次是：**继承，组合，聚合，（普通）关联，依赖**

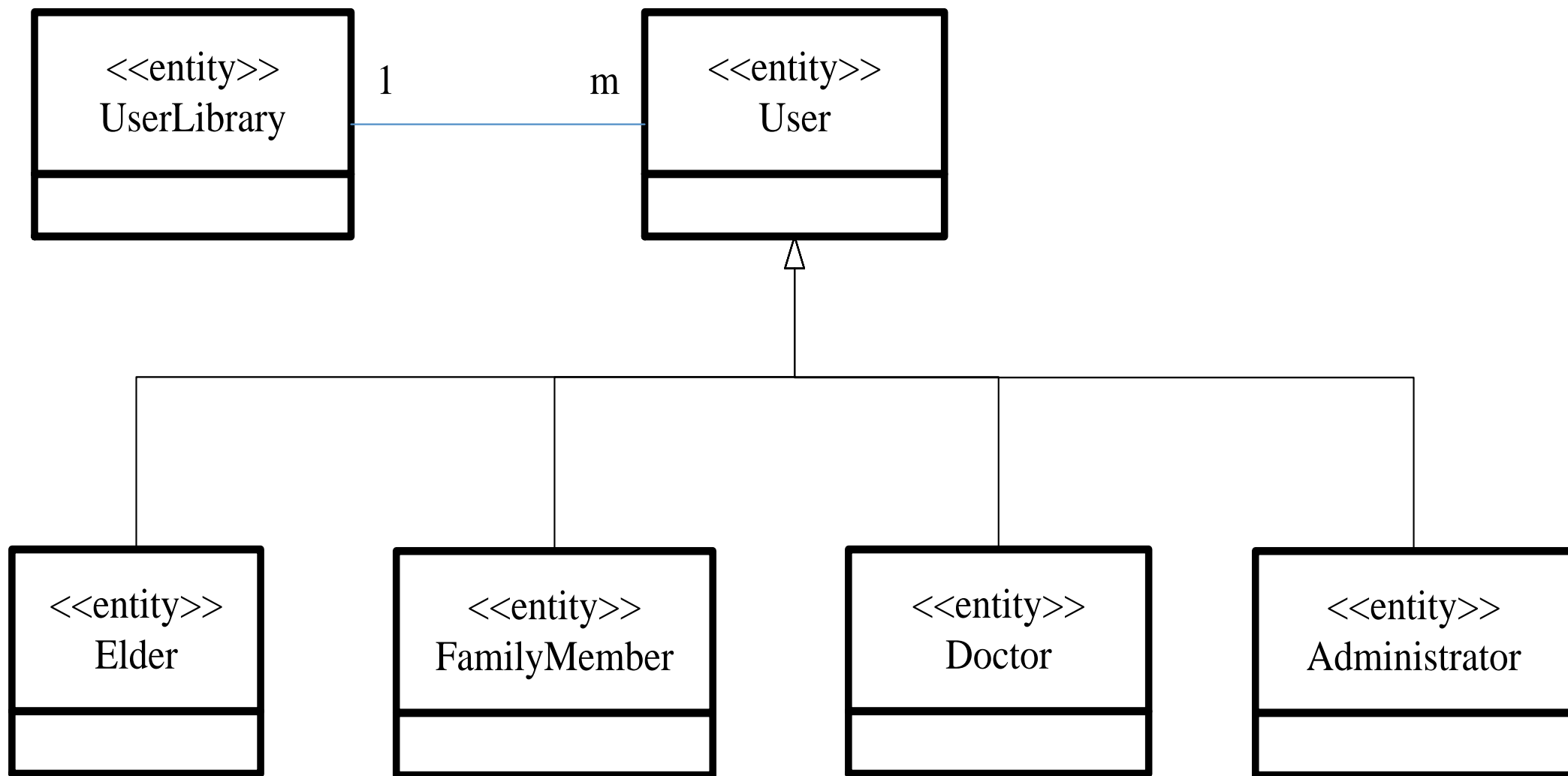
□类间关系定义的原则

- ✓ “**自然抽象**”原则，类间关系应该自然、直观地反映软件需求及其实现模型
- ✓ “**强内聚、松耦合**”的原则，即、尽量采用语义连接强度较小的关系

□确定类间的关系的数量对应

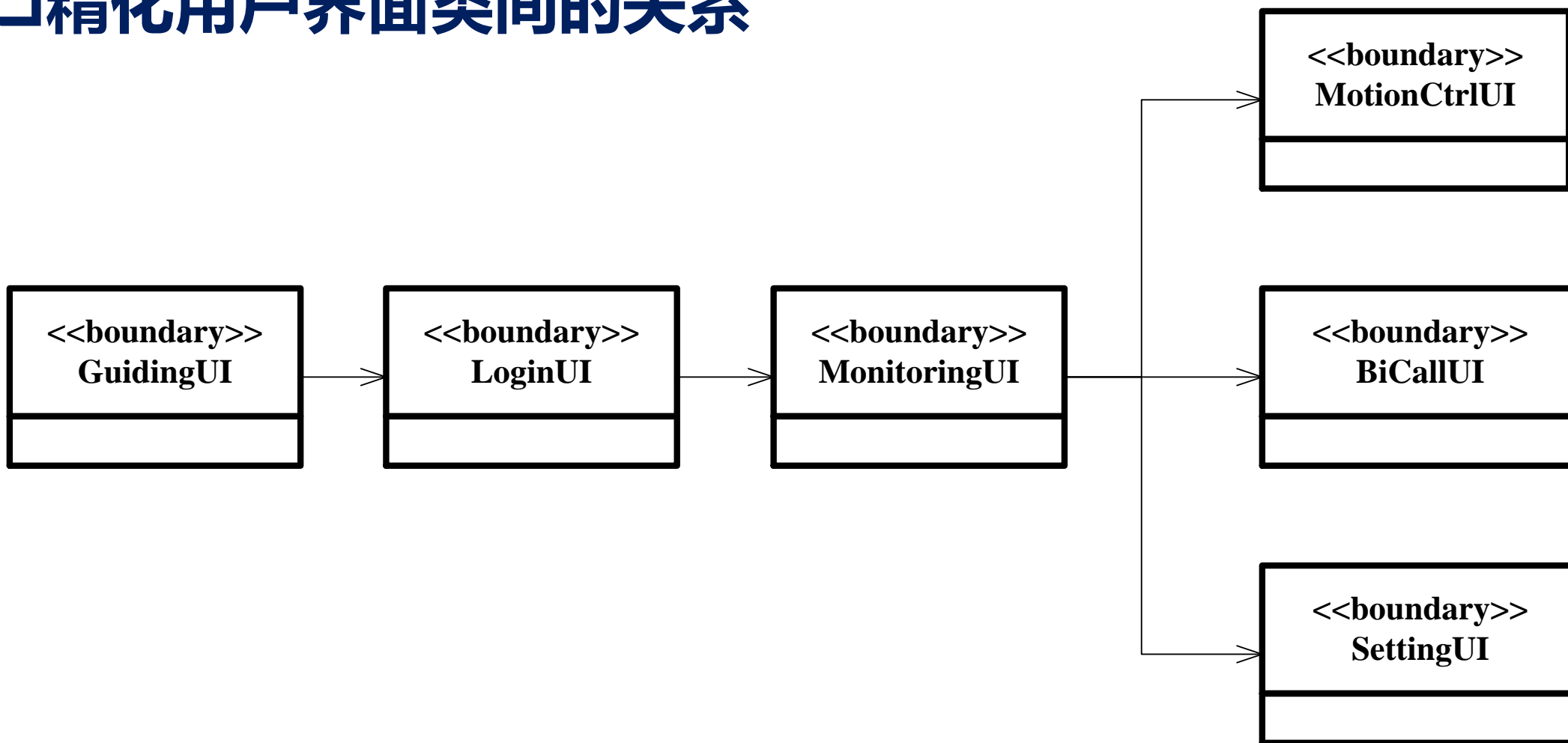
- ✓ $1:1$, 即一对一
- ✓ $1:n$, 即一对多
- ✓ $0:n$, 即0对多
- ✓ $n:m$, 即多对多等等

示例1：精化类间的关系



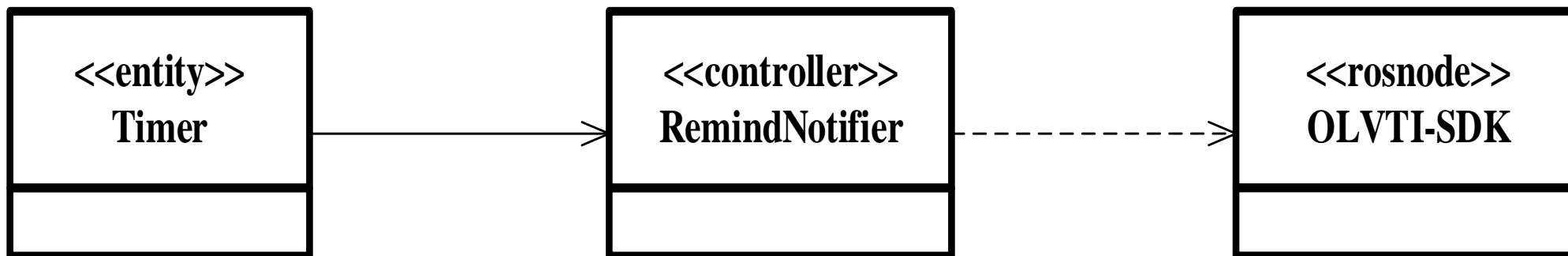
示例2：精化类间的关系

□精化用户界面类间的关系



示例3：精化类间的关系

□精化关键设计类间的关系



2.2.3 精化类的属性和方法

□精化类属性的设计

□精化类方法的设计

(1) 精化类属性的设计

□属性是类的基本组成部分，属性的取值定义了类对象的状态，细化和明确以下几个方面的设计信息：

✓类属性的命名

□用业务领域的名词或者名词短语来命名

✓类属性的可见范围

□public，对软件系统中的所有类均可见

□protected：仅对本类及其子类可见

□private：仅对本类可见

□遵循“信息隐藏”原则，尽可能地缩小作用范围

示例：精化User类属性的设计

□二项属性：用户名 “name” 和用户密码 “password”

- ✓ 类型均为String
- ✓ 可见范围均为 “private”
- ✓ 属性的初始值均为空串

示例：精化LoginUI类属性的设计

□静态元素、用户输入元素和命令界面元素

- ✓ “loginPicture” 类型为静态元素
- ✓ “account” 类型为用户输入元素，如文本框
- ✓ “password” 类型为用户输入元素（如文本框）

□可见范围

- ✓均为 “private”

□初始值

- ✓ “loginPicture” 属性的初始值不为空，要有一个预加载的图标；
“account” 和 “password” 的初始值为空串

示例：精化Robot类属性的设计

□ **private int velocity**

✓表示机器人的速度

□ **private int angle**

✓表示运动角度

□ **private int distance**

✓表示与老人的距离

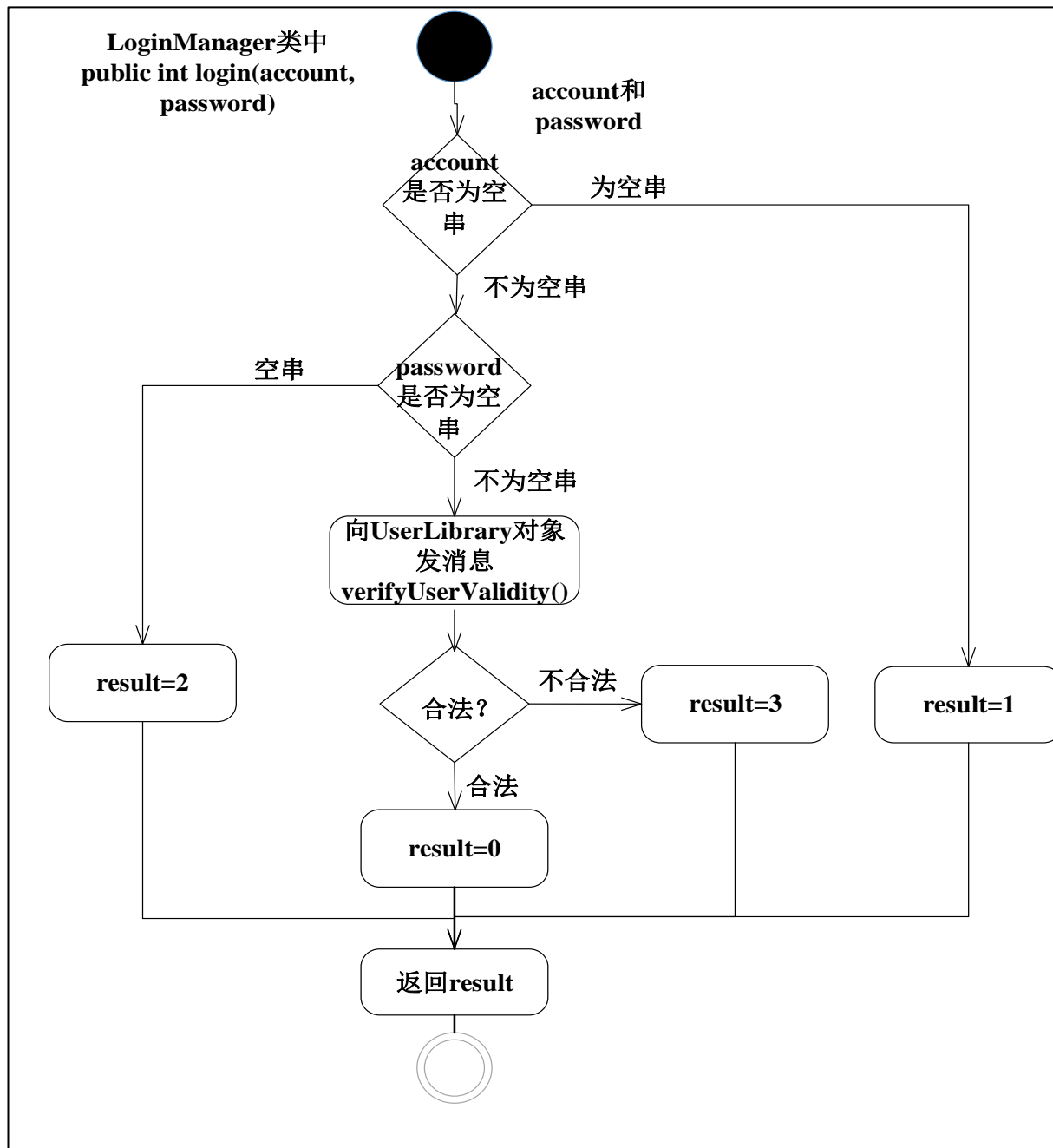
□ **private int state**

✓表示运动状态，包括“IDLE”空闲状态、“AUTO”自主跟随状态、“MANNUAL”手工控制状态

(2) 精化类方法的设计

□ 细化和明确类中各个方法的以下设计信息

- ✓ 方法名称
- ✓ 参数表（含参数的名称和类型）
- ✓ 返回类型
- ✓ 作用范围
- ✓ 功能描述
- ✓ 实现算法（用流程图、UML活动图表示）
- ✓ **前提条件（pre-condition）、出口断言（post-condition）等**



示例：用活动图描述的 Login() 方法的详细设计

□对象创建方法

- ✓在实例化类对象时会被执行，其职责能通常是完成类对象的初始化工作，包括初始化属性值等等

□对象删除方法

- ✓在类对象生命周期结束前被执行，其职责通常是完成对象生命周期结束前的一些事务性工作，如释放对象所占用的资源等等

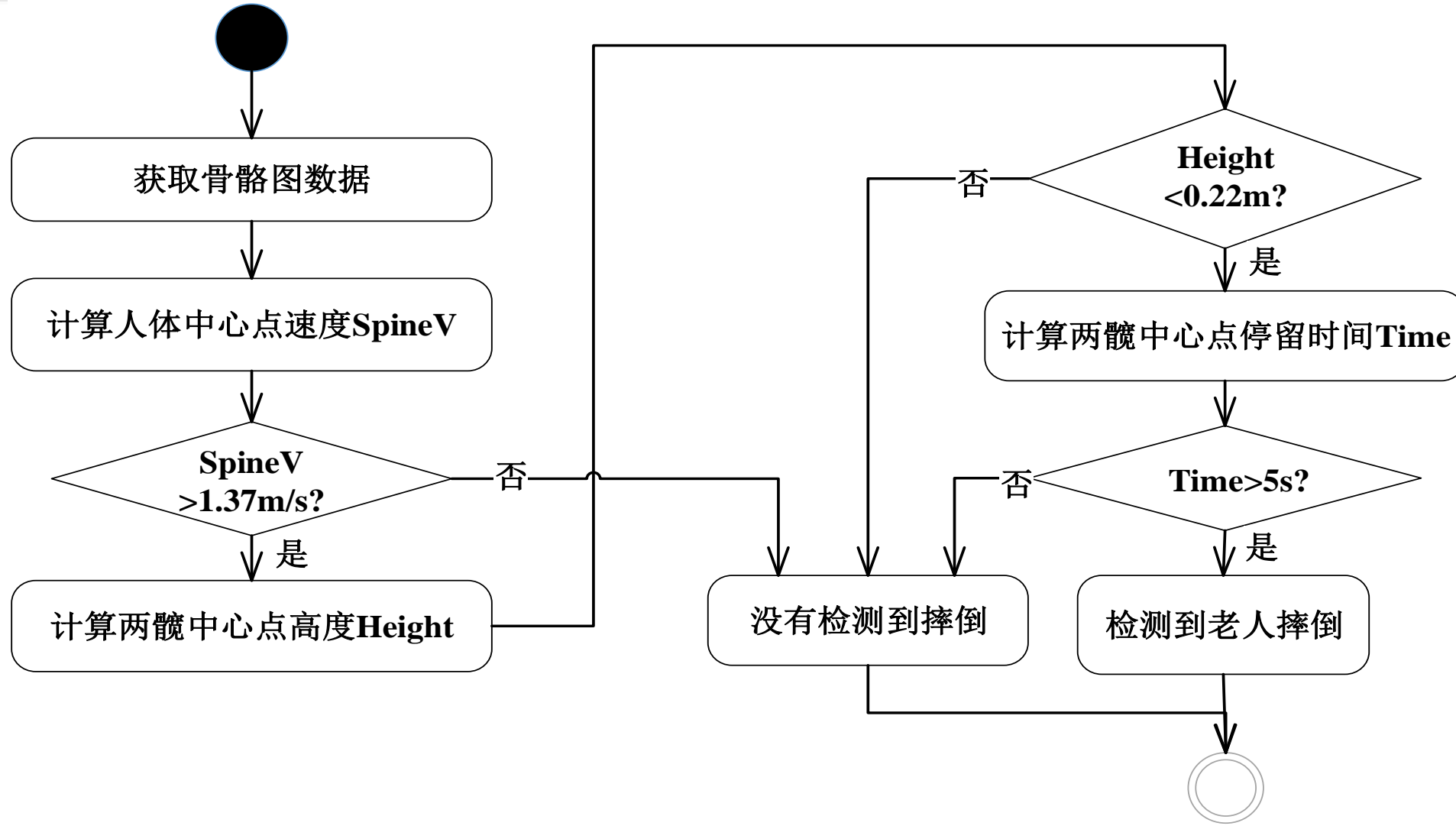
□对象比较方法

- ✓比较类的两个实例对象，判断它们是否相同

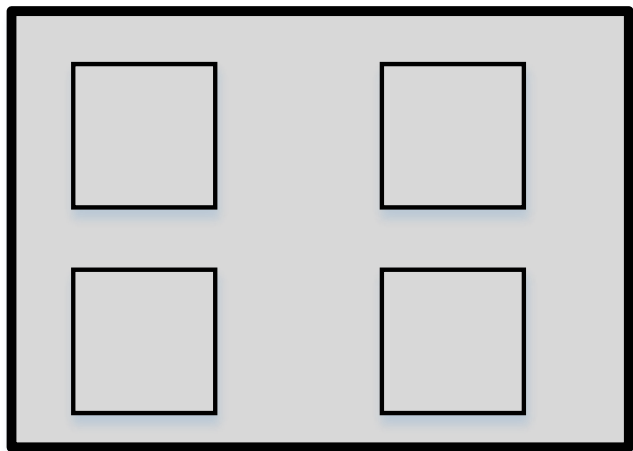
□对象复制方法

- ✓将类的一个实例对象的属性值复制到另一对象

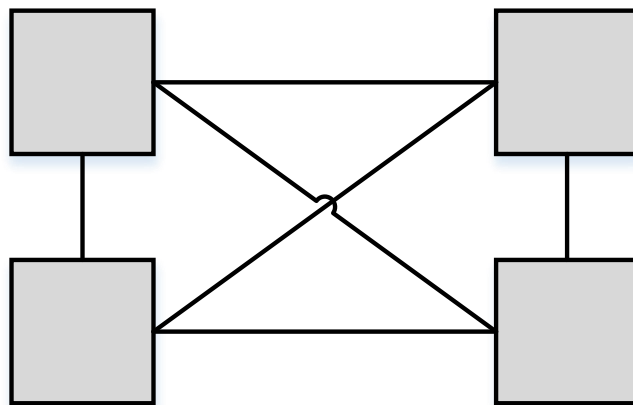
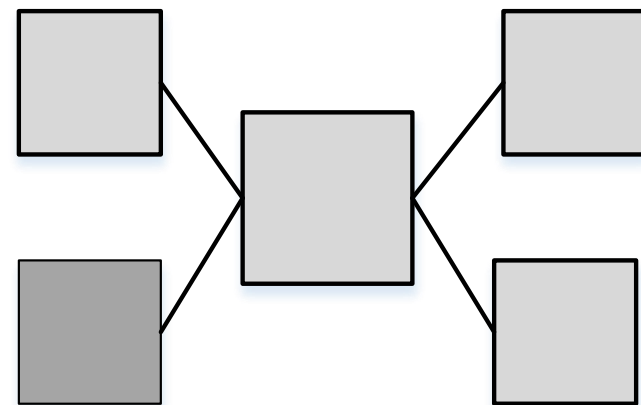
示例：精化detectFallDown()方法的详细设计



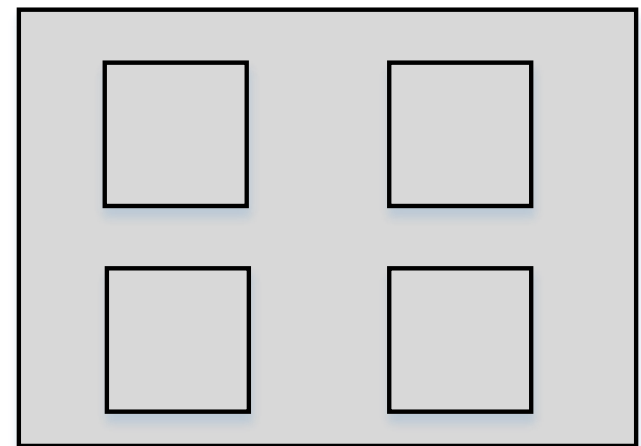
分解和合并类方法



将一个方法中多个松耦合的要素分解为多个方法



将多个紧耦合的方法合并为一个方法



优化类方法的设计

2.2.4 构造类对象的状态图

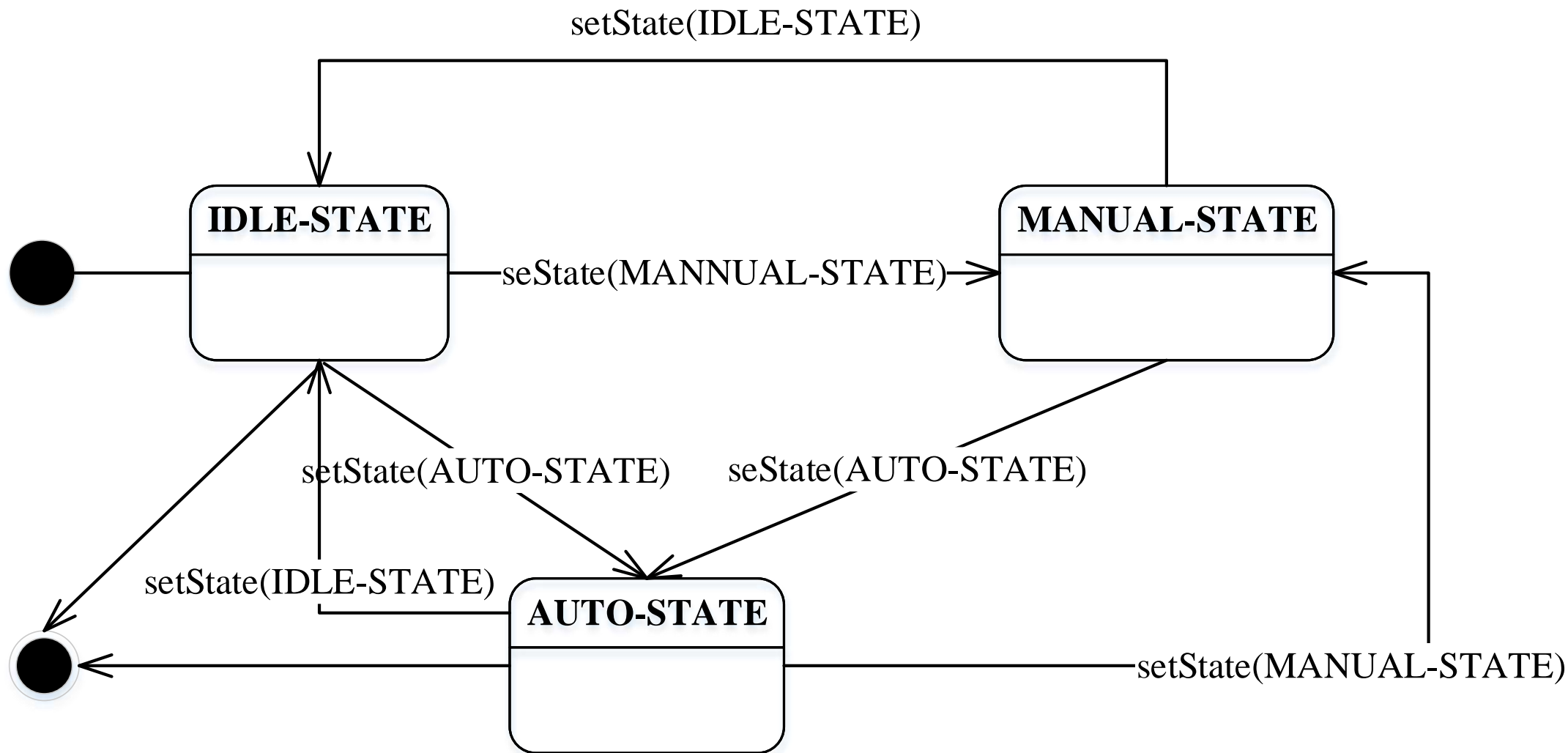
□状态图

- ✓如果一个类的对象具有较为复杂的状态，在其生命周期中需要针对外部和内部事件实施一系列的活动以变迁其状态，那么可以考虑构造和绘制类的状态图

□活动图

- ✓如果某个类在实现其职责过程中需要执行一系列的方法、与其他的对象进行诸多的交互，那么可以考虑构造和绘制针对该类某些职责的活动图

示例：Robot类对象的状态图



2.2.5 评审和优化类设计

- 根据“**强内聚、松耦合**”的原则，判断设计的模块化程度，必要时可以对类及其方法进行拆分和组合
- 评判类设计的详细程度，是否足以支持后续的软件编码和实现，依此为依据对类设计进行细化和精化
- 按照**简单性、自然性**等原则，评判类间的关系是否恰如其分地反映类与类之间的逻辑关系，是否有助于促进软件系统的自然抽象和重用
- 按照**信息隐藏的原则**，评判类的可见范围、类属性和方法的作用范围等是否合适，以尽可能地缩小类的可见范围，缩小操作的作用范围，不对外公开类的属性

类设计输出的软件制品

- 详细的类属性、方法和类间关系设计的**类图**
- 描述类方法实现算法细节的**活动图**
- 必要的**状态图**（可选）



内容

1. 软件详细设计概述

- ✓ 任务、过程和原则
- ✓ 详细设计的UML模型

2. 软件详细设计活动

- ✓ 2.1 用例设计
- ✓ 2.2 类设计
- ✓ **2.3 数据设计**
- ✓ 2.4 子系统和构件设计

3. 详细设计文档化和评审



为什么要进行数据设计

- 软件系统涉及各种信息，需要将其抽象为计算机可以理解和处理的数据
- 有些数据需要持久保存的，存放在永久存储介质中
 - ✓ 开展数据设计，以支持信息的抽象、组织、存储和读取
- 有些数据则需要存放在内存空间中，由运行的进程对其进行处理
 - ✓ 在类设计中抽象和封装为类属性及其数据类型

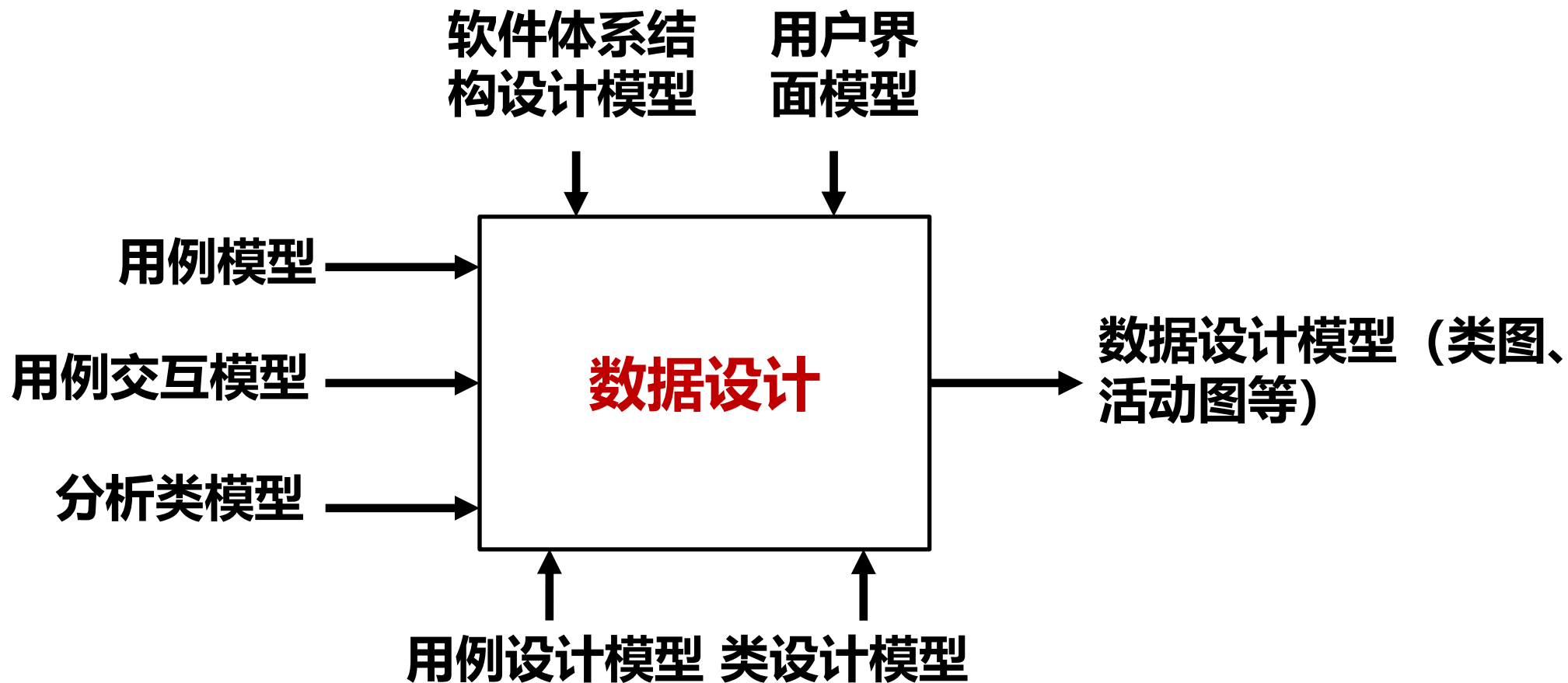
□任务

- ✓确定目标软件系统中需要持久保存的数据条目;
- ✓明确数据存储的组织方式（例如关系数据库中的表、关键字、外键等）
- ✓设计支持数据存储和读取的操作
- ✓为提高数据存储、操作性能而设计持久存储机制优化设施

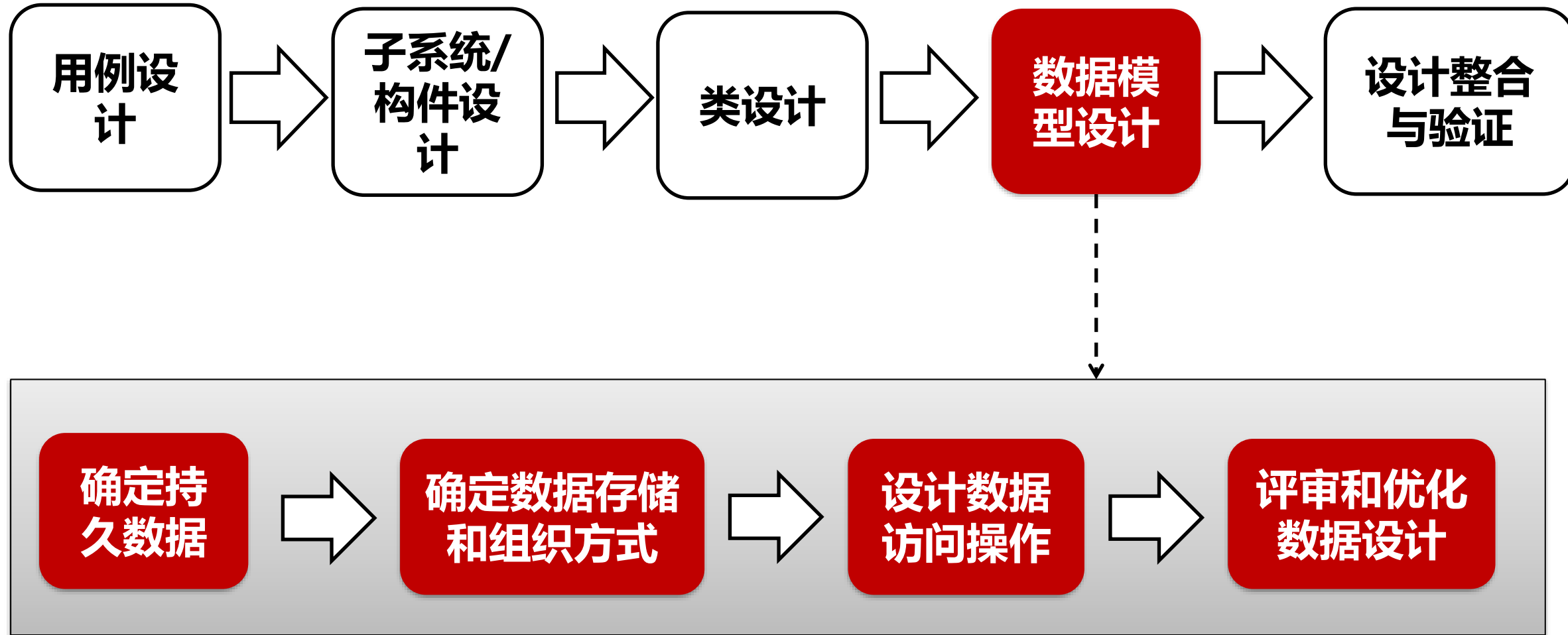
□设计与建模

- ✓设计数据的结构、存储、组织和访问
- ✓对数据设计的结果进行建模

数据设计的任务



数据设计过程



数据设计的原则

□可追踪

- ✓ 根据软件需求、体系结构设计、用例设计等模型开展数据设计

□无冗余

- ✓ 尽可能不要产生一些冗余、不必要的数据设计。

□考虑和权衡时空效率

- ✓ 反复折中数据的执行效率（如操作数据需要的时间）和存储效率（如存储数据所需的空间），以满足非功能性需求

□贯穿整个软件设计阶段

- ✓ 针对关键性、全局性的数据条目建立最初的数据模型
- ✓ 数据模型应该不断丰富、演进、完善，以满足用例、子系统、构件、类等设计元素对持久数据存储的需求

□验证数据的完整性

2.3.1 确定永久数据

□从需求模型和设计模型（尤其是类设计模型）明确需要处理哪些数据，哪些信息和数据需要持久保存

- ✓如用户的账号和密码
- ✓系统设置信息

2.3.2 确定持久数据的存储和组织方式

□将数据存储和数据文件中

- ✓确定数据存储的组织格式，以便将格式化和结构化的数据存放在数据文件之中

□将数据存储数据库中

- ✓设计支持数据存储的数据库表

确定持久数据条目

□确定设计模型中需要持久保存的类的对象及其属性

□面向对象设计模型与关系数据库模型的对应关系

- ✓类对应于“**表格**” (table)
- ✓对象对应于“**记录**” (record)
- ✓属性对应于表格中的“**字段**” (field)

可以用带构造型的UML类来表示关系数据模型中的表，**《table》**表示表格，**《key》**表示关键字字段

| <<table>>T_Log |
|--------------------|
| -<<key>>time |
| -actor |
| -actionDescription |
| |

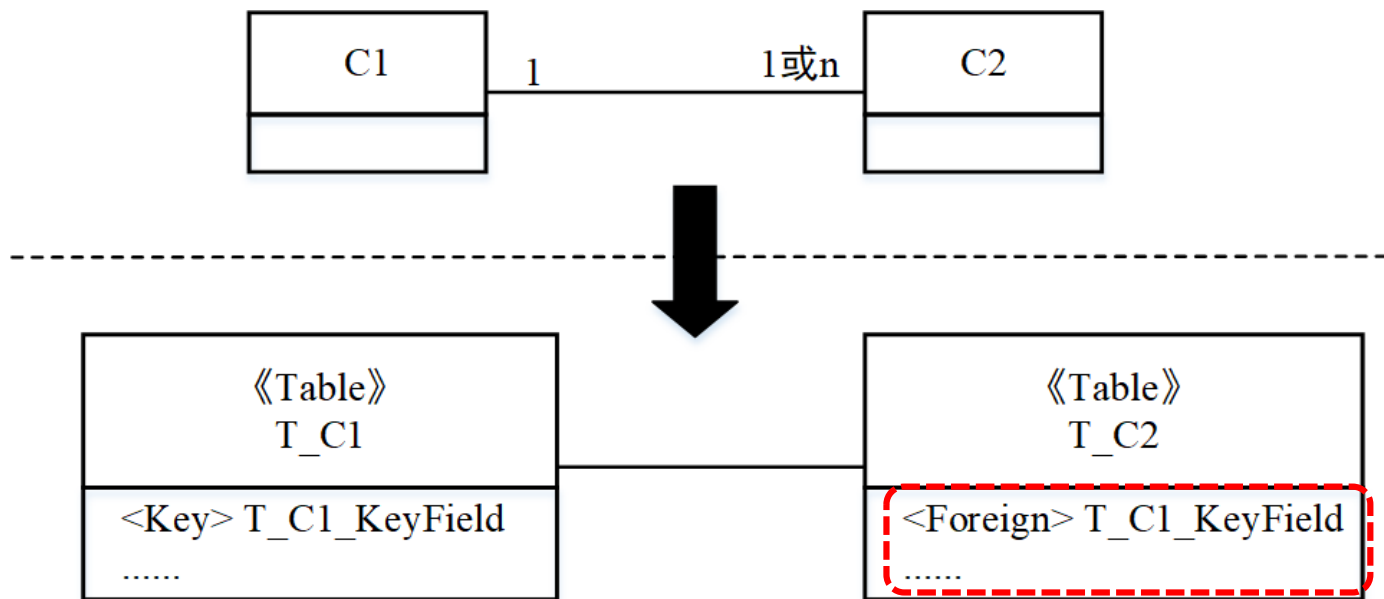
| <<table>>T_ExEvent |
|--------------------|
| -<<key>>time |
| -location |
| -type |
| -eventDescription |
| |

| <<table>>T_Sensor |
|-------------------|
| -<<key>>ID |
| -type |
| -location |
| -sensitivity |
| -status |
| |

1:1、1:n关联关系的映射

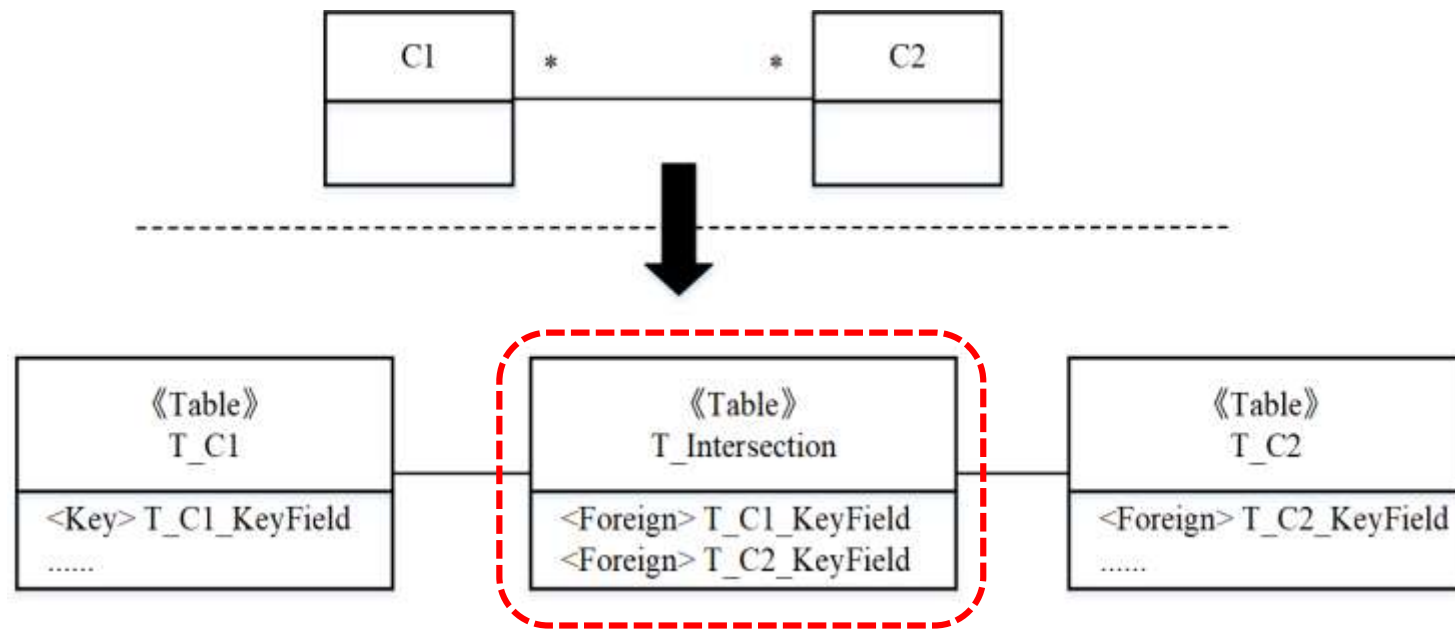
□根据类之间的关联关系及数量对应关系来确定相关数据在数据库表中的组织方式

- ✓假设类C1、C2对应的表格分别为T_C1、T_C2，只要将T_C1中关键字段纳入T_C2中作为**外键**，就可表示从T_C1到T_C2间的1:1、1:n 关联关系



n:m关联关系的映射

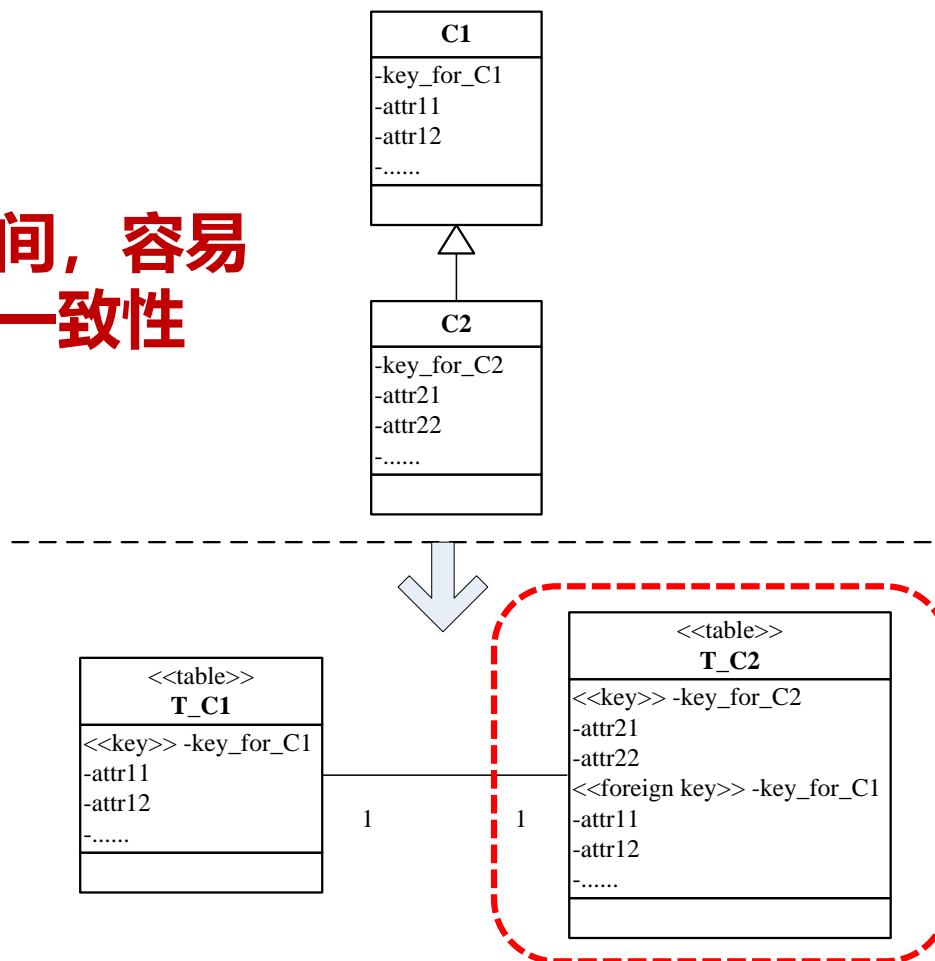
- 在T_C1、T_C2间引进新交叉表格**T_Intersection**，将T_C1、T_C2关键字段纳入T_Intersection中作为外键，在T_C1与T_Intersection之间、T_C2与T_Intersection之间建立一对多关系



继承关系的数据库表设计(1/2)

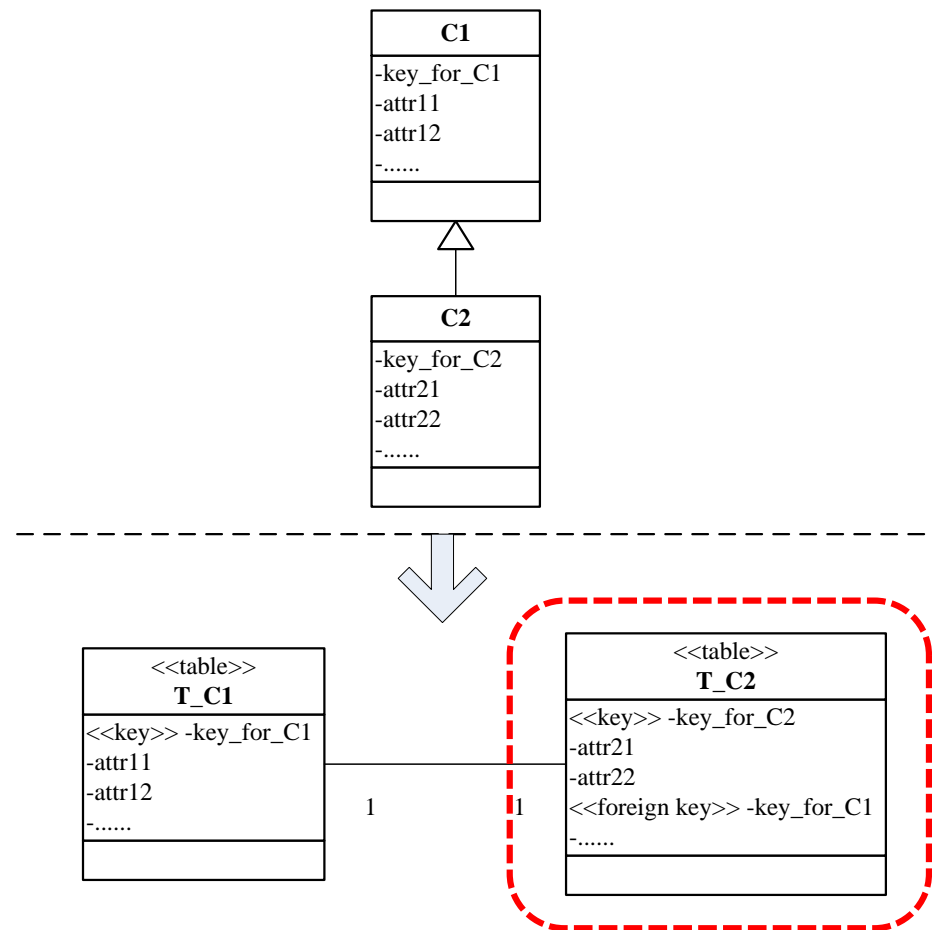
□ 假设C1是C2的父类，将T_C1中的所有字段全部引入至T_C2

弊端：浪费了持久存储空间，容易因数据冗余而导致数据不一致性



继承关系的数据库表设计(2/2)

- 假设C1是C2的父类，仅将
T_C1中关键字字段纳入
T_C2中作为外键
- 获取C2对象的全部属性，需
要联合T_C2中的记录和对应
于外键值的T_C1中的某条记
录
- 避免数据冗余，但在读取C2
对象时性能不如前种方法



示例：设计持久数据

<<table>>

T_User

<<key>>account string[50]

password string[6]

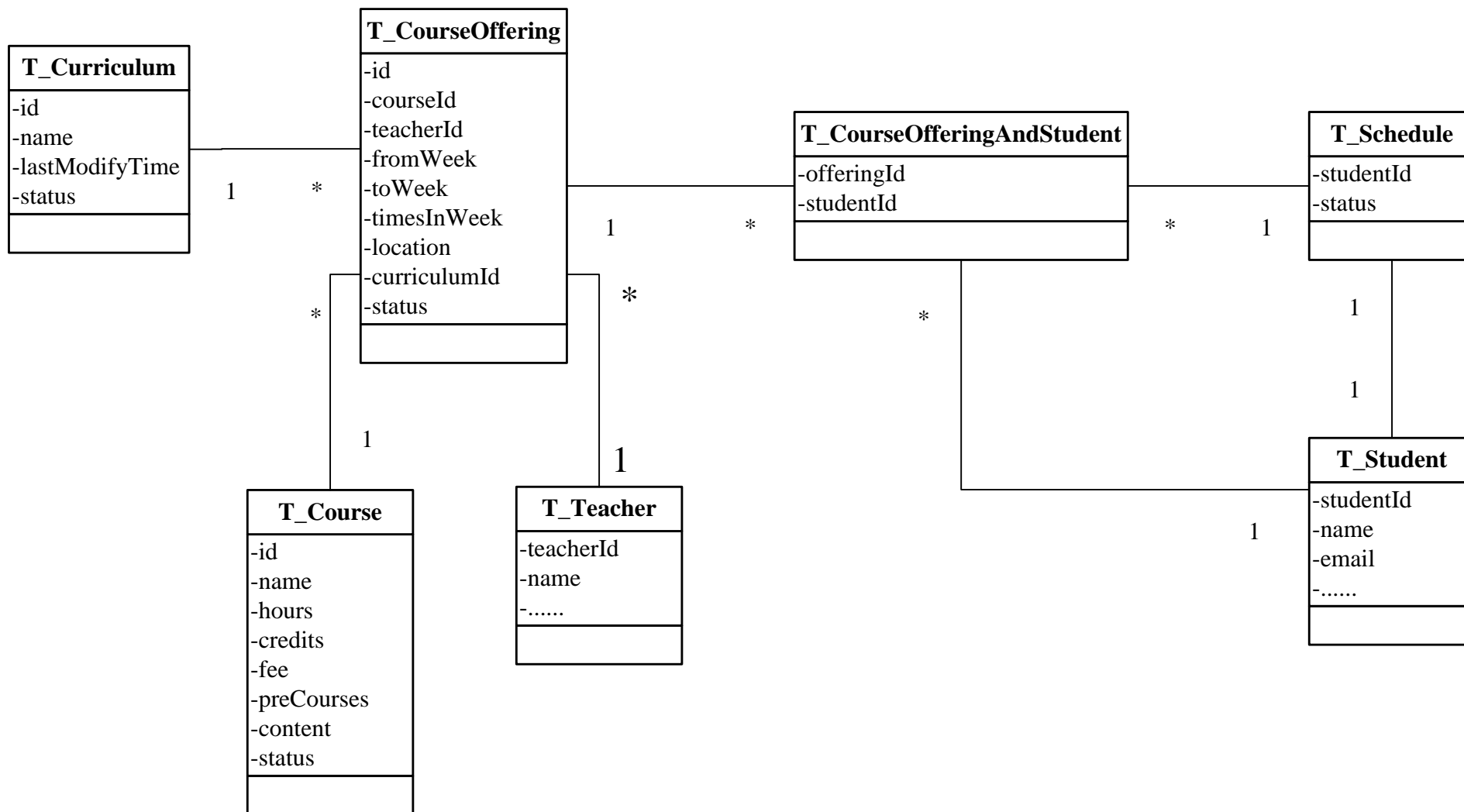
name string[10]

mobile string[12]

type int

保存 “User” 类对象的数据库表 “T_User”

示例：设计持久数据



2.3.3 设计数据操作

□ 写入、查询、更新和删除四类基本操作以及由它们复合而成的业务数据操作

- ✓ **写入操作** 将数据从运行时的软件系统保存至数据库
- ✓ **查询操作** 按照特定的选择准则从数据库提取部分数据置入运行时软件系统中的指定对象
- ✓ **更新操作** 以运行时软件系统中的（新）数据替换数据库中符合特定准则的（旧）数据
- ✓ **删除操作** 将符合特定准则的数据从数据库中删除

□ 数据验证操作该操作

- ✓ **验证操作** 负责验证数据的完整性、相关性、一致性等等

示例：设计永久数据的操作

- **boolean insertUser(User)**
- **boolean deleteUser(User)**
- **boolean updateUser(User)**
- **User getUserByAccount(account)**
- **boolean verifyUserValidity(account, password)**
- **UserLibrary()**
- **~UserLibrary()**
- **void openDatabase()**
- **void closeDatabase()**

<<table>>
T_User

<<key>>account string[50]
password string[6]
name string[10]
mobile string[12]
type int

2.3.4 评审和优化数据设计

□正确性

- ✓数据设计是否满足软件需求

□一致性

- ✓数据设计尤其是数据的组织是否与相关的类设计相一致

□时空效率

- ✓分析数据设计的空间利用率，以此来优化数据的组织
- ✓根据数据操作的响应时间来分析数据操作的时效性，优化数据库以及数据访问操作

□可扩展性

- ✓数据设计是否考虑和支持将来的数据持续保存的可能扩展

- 描述数据设计的类图
- 描述数据操作的活动图



内容

1. 软件详细设计概述

- ✓任务、过程和原则
- ✓详细设计的UML模型

2. 软件详细设计活动

- ✓2.1 用例设计
- ✓2.2 类设计
- ✓2.3 数据设计
- ✓**2.4 子系统和构件设计**

3. 详细设计文档化和评审



为什么需要子系统/构件设计

- 体系结构设计和用例设计引入了子系统或者构件
- 从软件封装和重用的角度需要将设计元素重组为子系统或者构件
- 尚未对子系统/构件进行深入的设计

□任务

- ✓确定子系统**内部结构**，设置包含于其中的更小粒度**子系统、构件和设计类**，明确它们之间的**协作关系**
- ✓确保它们能够协同实现子系统接口规定的所有功能和行为

□设计和建模

- ✓细化子系统内部的细节，如设计元素、关联和交互
- ✓对子系统内部的结构进行建模
- ✓对子系统内部各个设计元素之间的协作进行建模

□结果

- ✓包图、构件图、顺序图、活动图、类图

子系统设计的任务

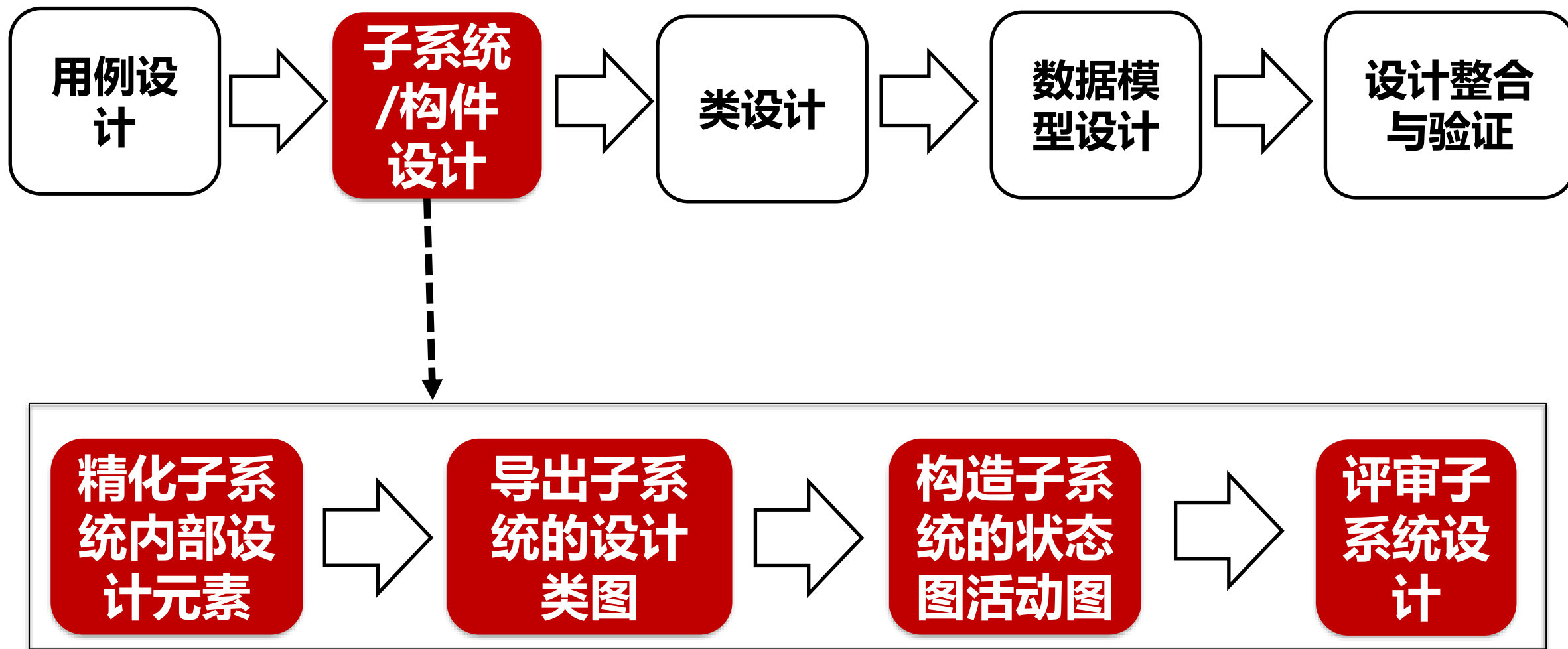
□确定子系统内部的结构

- ✓设计包含于其中的、粒度更小的子系统、构件和设计类
- ✓设计它们之间的接口和协作关系

□确保它们能够协同实现体系结构模型中该子系统的服务提供接口所规定的全部功能和行为



子系统设计过程



子系统设计的原则

- 将分析模型中一个或一些较复杂、职责粒度较大的分析类抽象为一个子系统，并对此进行单独设计
- 考虑软件非功能性需求，思考实现非功能需求的方法
- 确保将子系统的职责分解到各个设计元素之中
- 确保子系统设计元素能够完整地实现整个子系统职责
- 不仅将注意力集中在子系统内部元素的设计上，还要思考所设计的子系统如何通过接口与其外部的设计元素（如构件、设计类、其他子系统等）进行交互和协作。
- 结合已有软件资产、考虑实现约束等因素来进行子系统的设计，尽可能地通过重用开源软件、集成遗留系统

□完整性

- ✓ 子系统内部各设计元素所承担的职责完整覆盖了子系统的职责

□设计质量

- ✓ 是否体现了软件工程的基本原则
- ✓ 各个设计元素的职责划分是否合理、功能和接口封装是否恰当

□可满足性

- ✓ 子系统设计是否实现了所赋予子系统的软件需求，是否存在多余的设计元素，是否引入了不必要的软件资产

□正确性

- ✓ 是否正确地使用UML图符和模型来描述子系统设计的软件制品

□一致性

- ✓ 多个软件制品间是否一致，模型和文字表述是否一致



内容

1. 软件详细设计概述

- ✓任务、过程和原则
- ✓详细设计的UML模型

2. 软件详细设计活动

- ✓用例设计
- ✓类设计
- ✓数据设计
- ✓子系统和构件设计

3. 详细设计文档化和评审



3.1 软件详细设计的输出

□模型

- ✓用UML类图、构件图、包图、状态图、顺序图等描述的详细设计模型

□文档

- ✓软件详细设计规格说明书

3.2 设计整合

□ 汇总迄今获得的所有设计模型

✓ 包括体系结构模型、界面设计模型、用例设计模型、子系统/构件/类设计模型、数据模型等

□ 形成系统、完整的软件设计方案

3.3 设计验证

- 验证整个设计的正确性、优化性和充分性等
- 验证设计模型之间的不一致性、冗余性等
- 发现设计方案中的问题并进行整改

3.4 撰写设计文档

1、引言

- 1.1 编写目的
- 1.2 读者对象
- 1.3 软件系统概述
- 1.4 文档概述
- 1.5 定义
- 1.6 参考资料

2、软件设计约束和原则

- 2.1 软件设计约束
- 2.2 软件设计原则

3. 软件设计方案

- 3.1 体系结构设计
- 3.2 部署设计
- 3.3 用户界面设计
- 3.4 用例设计
- 3.5 子系统/构件设计
- 3.6 类设计
- 3.7 数据设计

4. 实施指南

3.5 设计评审人员

□ 用户（客户）

- ✓ 评估和分析软件设计是否正确地实现了他们所提出的软件需求

□ 软件设计人员

- ✓ 根据评审的意见来修改设计方案

□ 程序员

- ✓ 能否正确理解设计文档、是否提供足够详细的设计方案以指导编码

□ 软件需求分析人员

- ✓ 是否实现了他们所定义的软件需求

□ 质量保证人员

- ✓ 发现软件设计模型和文档中的质量问题，并进行质量保证

□ 测试工程师

- ✓ 以软件设计文档为依据，设计软件测试用例，开展软件测试

□ 配置管理工程师

- ✓ 对软件设计规格说明书和设计模型进行配置管理

3.6 评审设计文档(1)

□规范性

- ✓ 是否遵循文档规范，是否按规范的要求和方式来撰写文档

□简练性

- ✓ 语言表述是否简洁不啰嗦、易于理解。

□正确性

- ✓ 设计方案是否正确实现了软件功能性需求和非功能性需求

□可实施性

- ✓ 设计元素是否已充分细化和精化，模型是否易于理解，所选定的程序设计语言是否可以实现该设计模型

□可追踪性

- ✓各项需求是否在设计文档中都可找到相应的实现方案，设计文档中的设计内容是否对应于需求条目

□一致性

- ✓设计模型间、文档不同段落间、文档的文字表达与设计模型间是否一致。

□高质量

- ✓是否充分考虑了软件设计原则，设计模型是否具有良好的质量属性，如有效性、可靠性、可扩展性、可修改性等

- 阅读和汇报软件设计规格说明书
- 收集和整理问题
- 讨论和达成一致，并进行修改
- 纳入配置

□详细设计是要给出可指导编码的详细设计方案

- ✓依据软件需求、体系结构和用户界面设计模型

□详细设计的任务

- ✓用例设计、子系统设计、构件设计、数据设计、类设计

□详细设计的描述和输出

- ✓UML的顺序图、类图、状态图、活动图、构件图等
- ✓软件设计规格说明书

□软件设计的整合、验证与评审

- ✓形成系统的软件设计方案
- ✓发现和修改方案中存在的问题