软件详细设计

内容

1. 软件详细设计概述

- ✓任务
- ✓过程

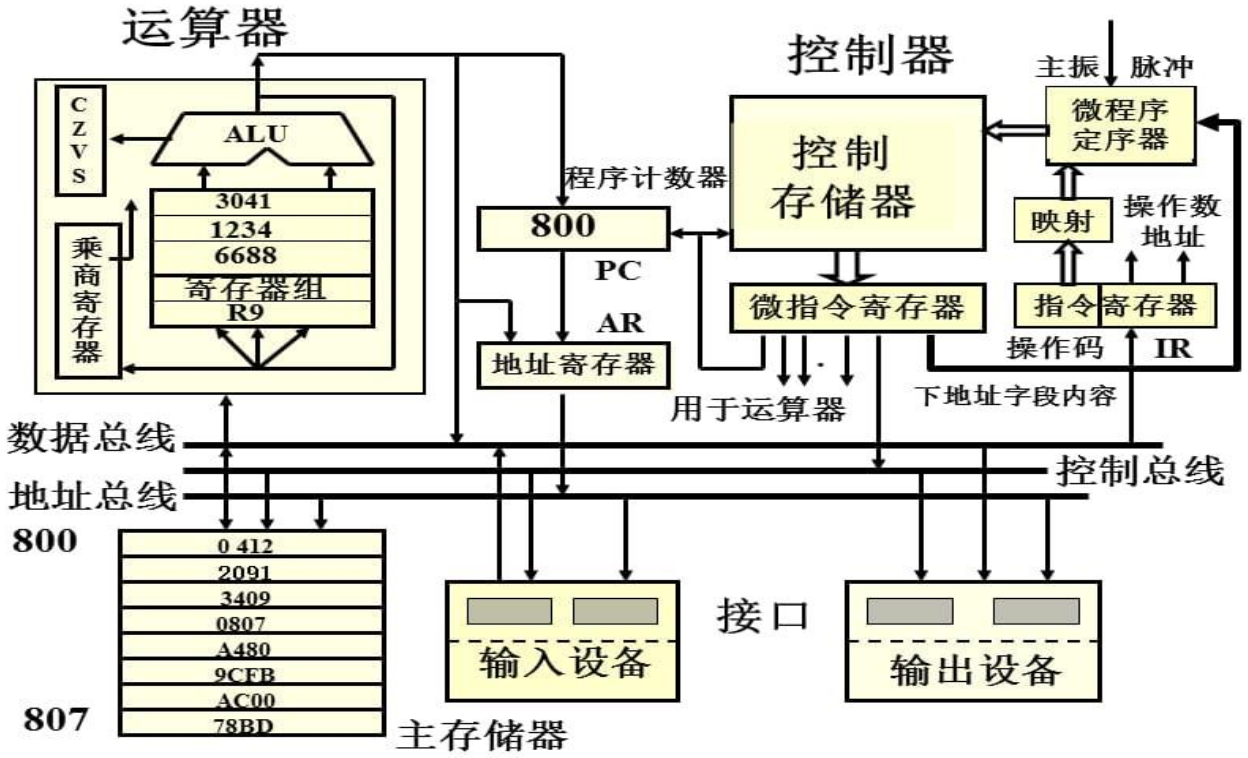
2. 软件详细设计活动

- ✓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 软件设计关注点的变化

- 详细设计是在体系结构设计的基础上，对构件内部细节的进一步设计
- 详细设计关注每个部件，如”运算器”的 内部电路和元件等。

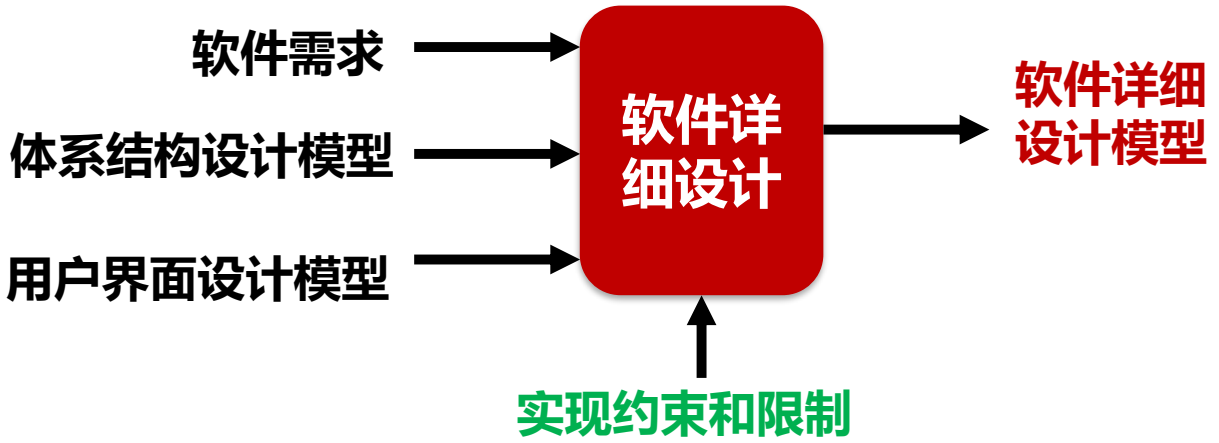


体系结构设计关注整体

详细设计关注细节

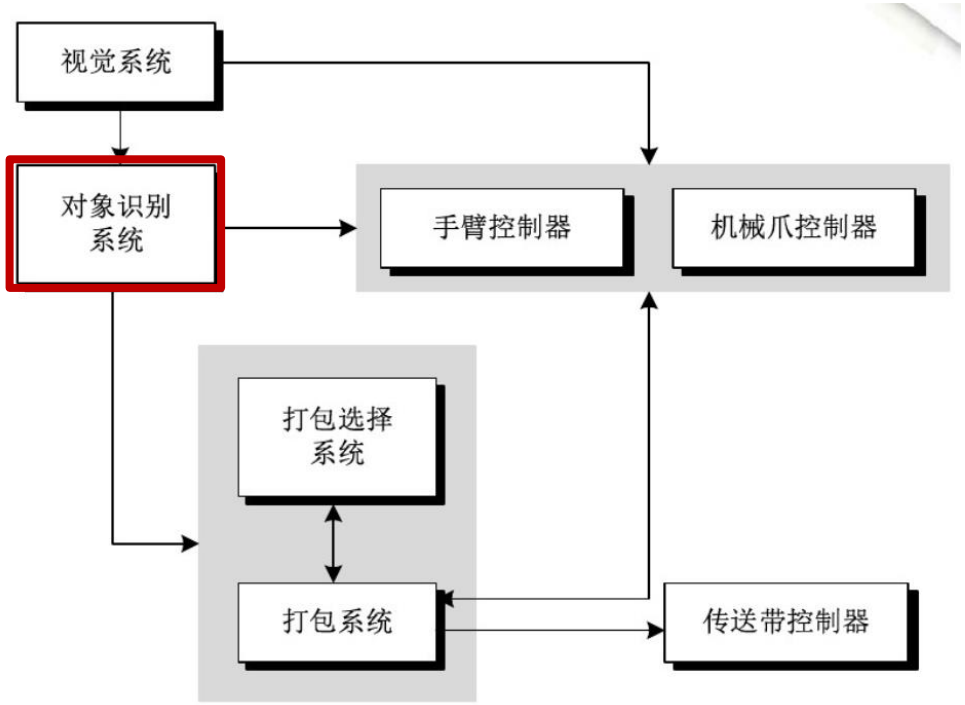
1.2 详细设计的任务

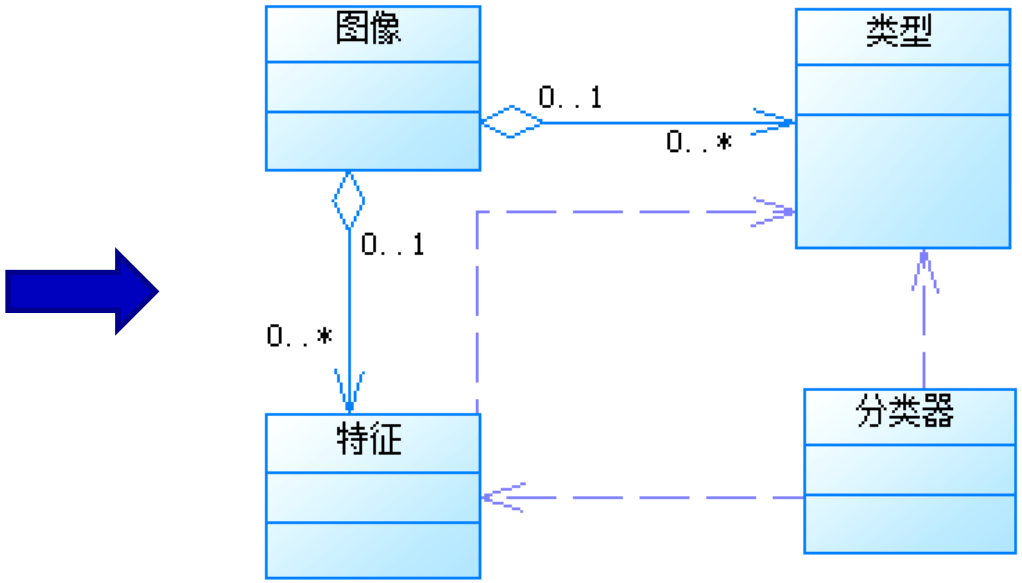
- **任务：对体系结构和界面设计成果进行细化，获得面向实现的设计模型**
  - ✓面向实现：设计成果能直接支持编码、实现
  - ✓关注约束：实现方案必须满足约束条件
  - ✓足够细化：
    - 构件中包括哪些设计类?
    - 每个设计类中包括的属性和方法?
    - 每个方法采用的算法逻辑?



详细设计是架构设计和实现间的桥梁

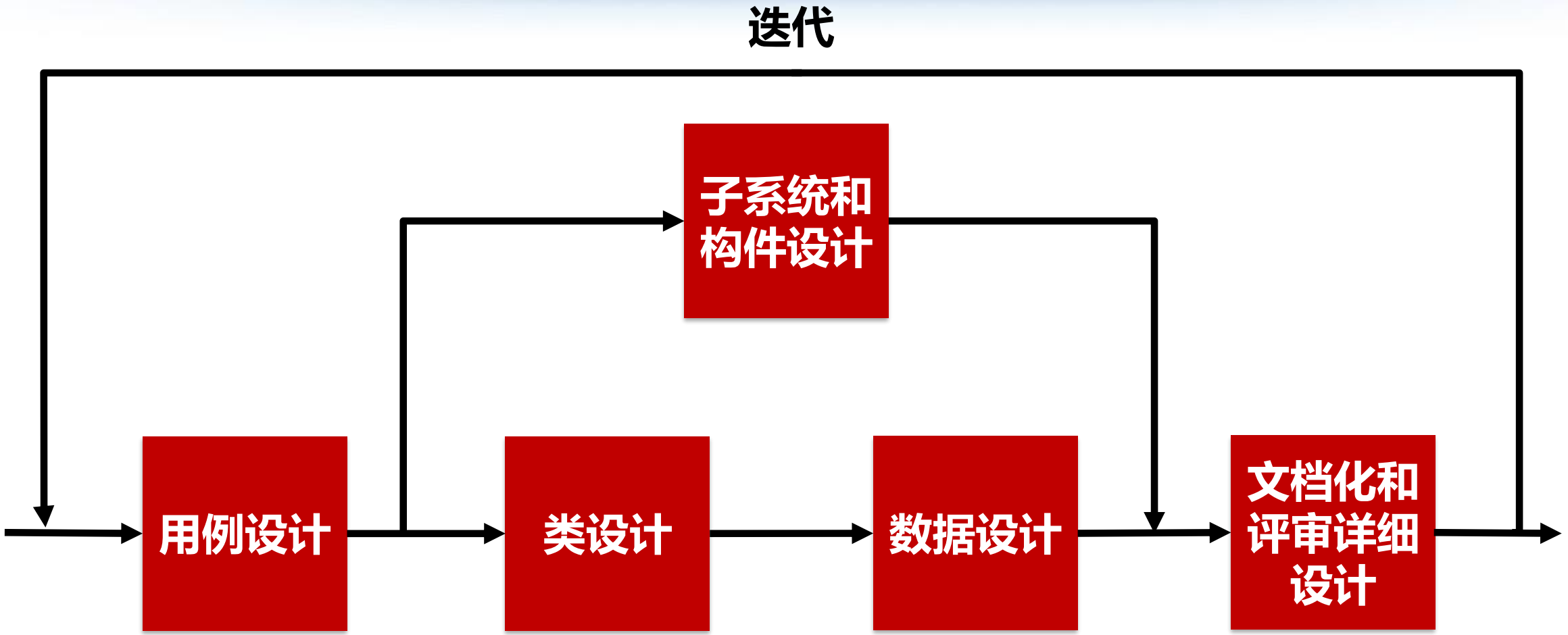
❑对“对象识别子系统”的详细设计





没有详细设计，开发者在编码时可能会面临很多不确定性，导致代码质量下降

1.3 详细设计过程



包含四个主要的详细设计活动

内容

1. 软件详细设计概述
   - ✓任务
   - ✓过程
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**

2.1 用例设计

❑任务

- ✓基于体系结构和界面设计给出的设计元素，设计每个**用例的实现方案**

❑用例的实现方案

- ✓用例如何通过各个设计元素（构件、设计类）之间的交互和协作来实现的

1. 设计用例实现方案

❑方法

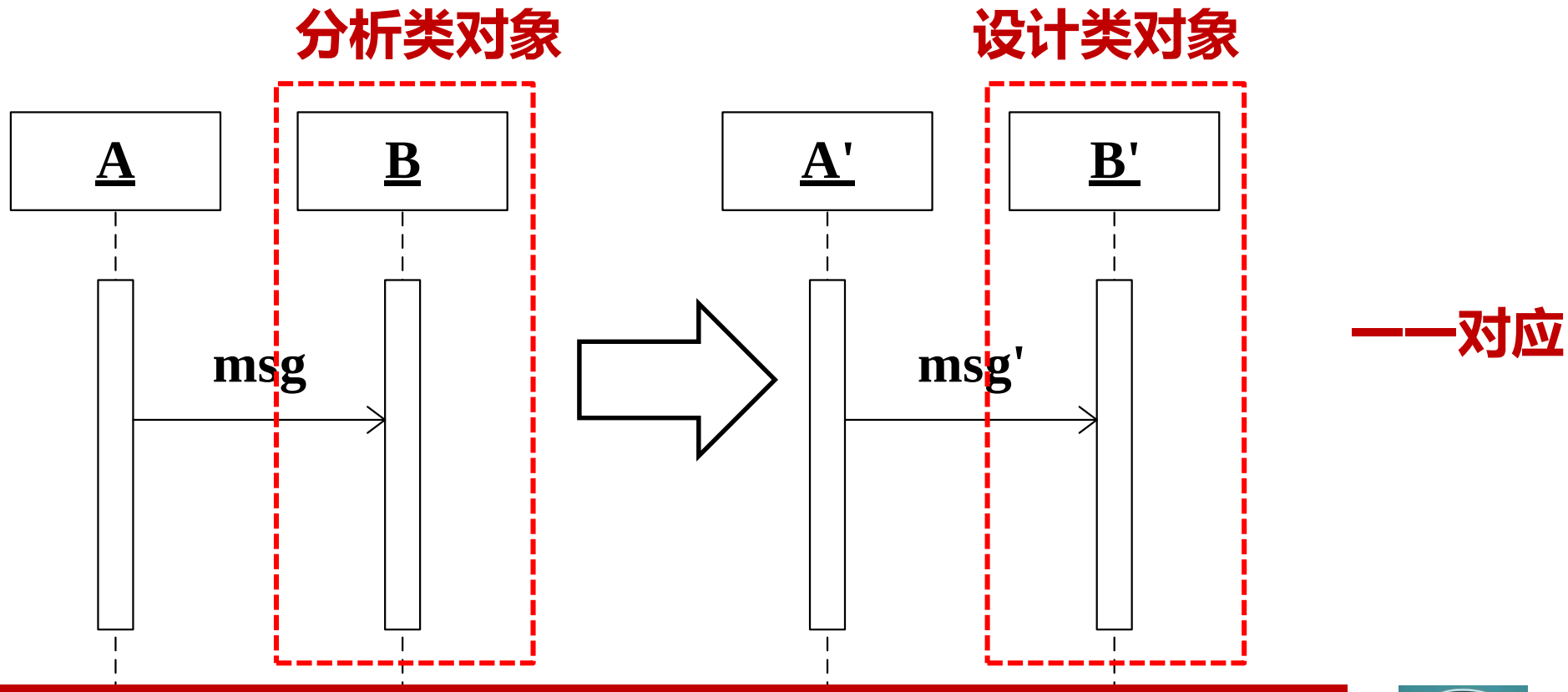
- ✓在分析阶段交互图的基础上，将交互图的**分析类**转化为用例实现的**设计类**，并结合体系结构设计、界面设计中的**设计元素**，产生用例实现的交互模型

❑目标

- ✓找到支撑用例实现的所有**设计类**

如何将分析类精化为设计类(1/3)

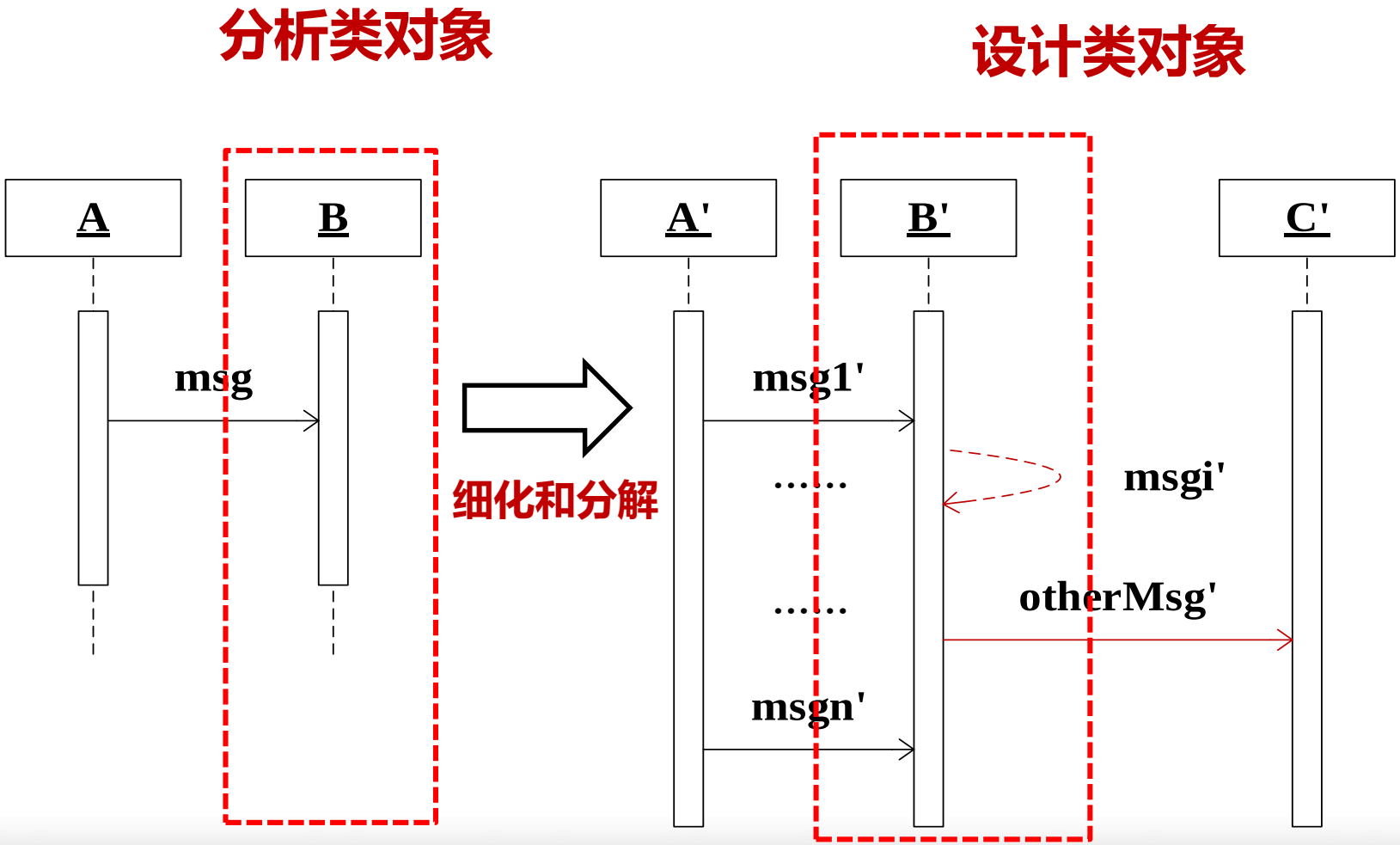
- 如果分析类的**职责较简单，可以由某个设计元素的单项操作实现**



分析类和设计类有何本质的区别?

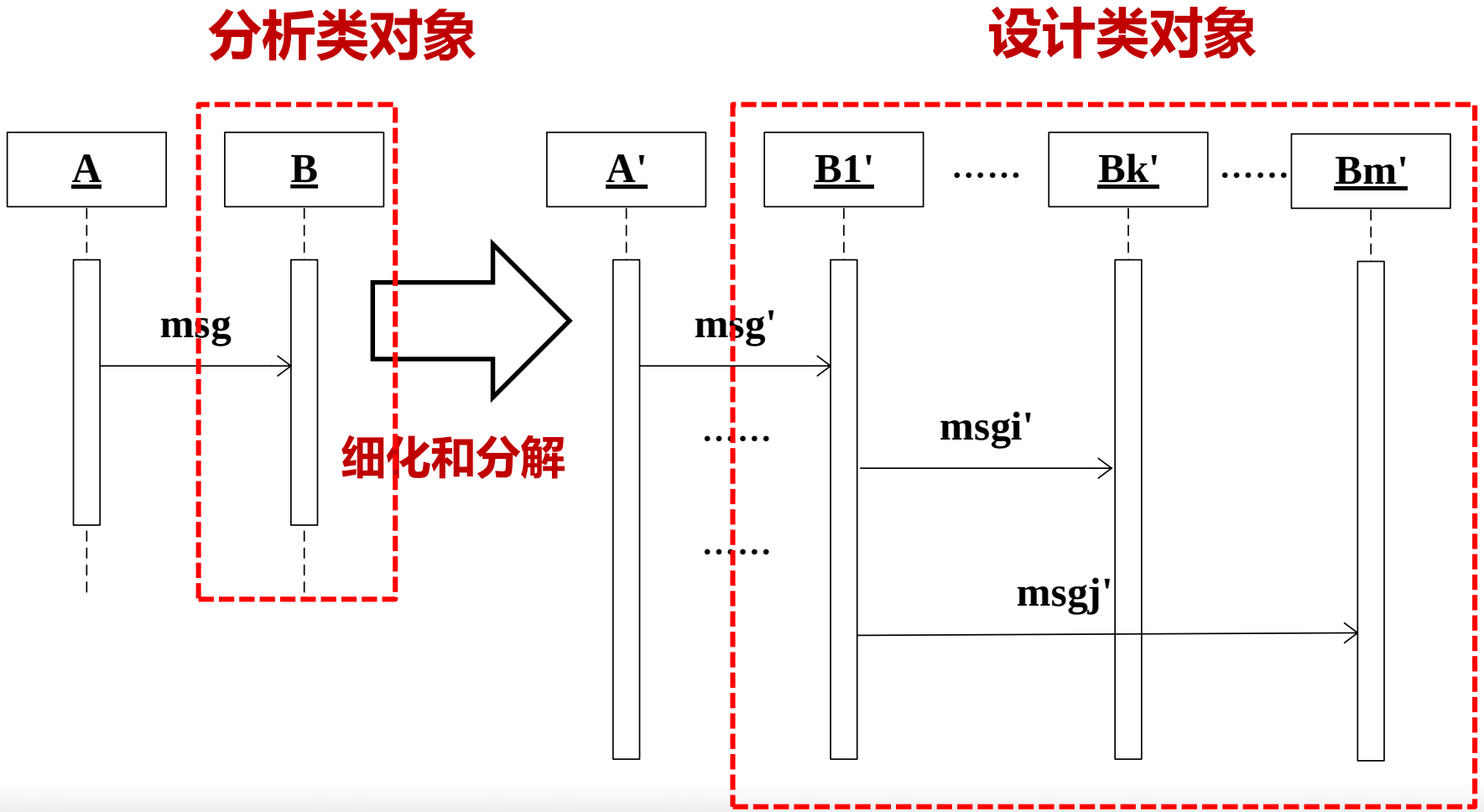
如何将分析类精化为设计类(2/3)

- 某个分析类的**一项职责**由某个设计元素的**多项操作**来实现

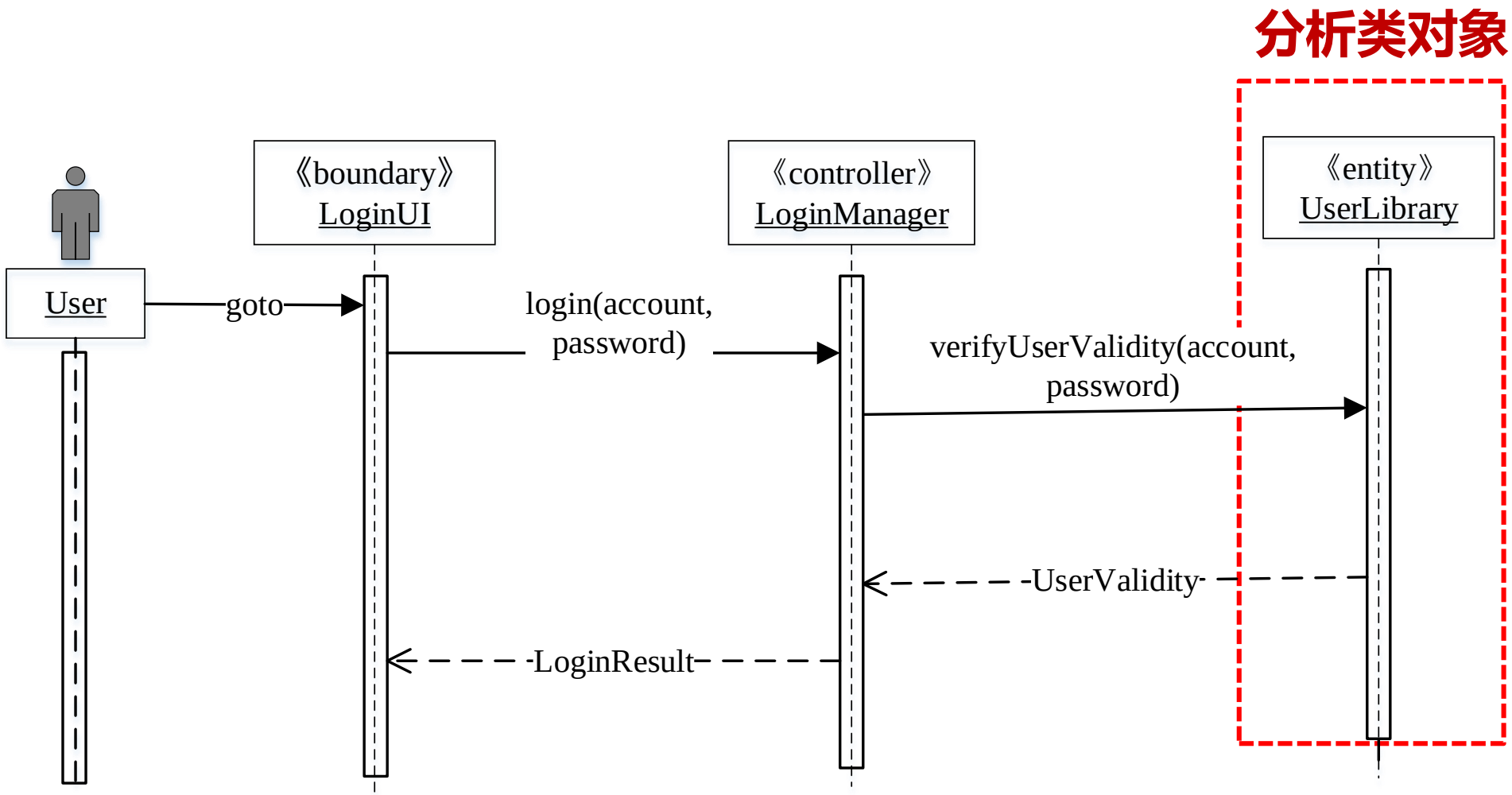


如何将分析类精化为设计类(3/3)

- 如果分析类的**职责**难以由单个设计类承担，则应该分解为**多个设计类**协同完成



示例1：“用户登录”用例顺序图（分析）

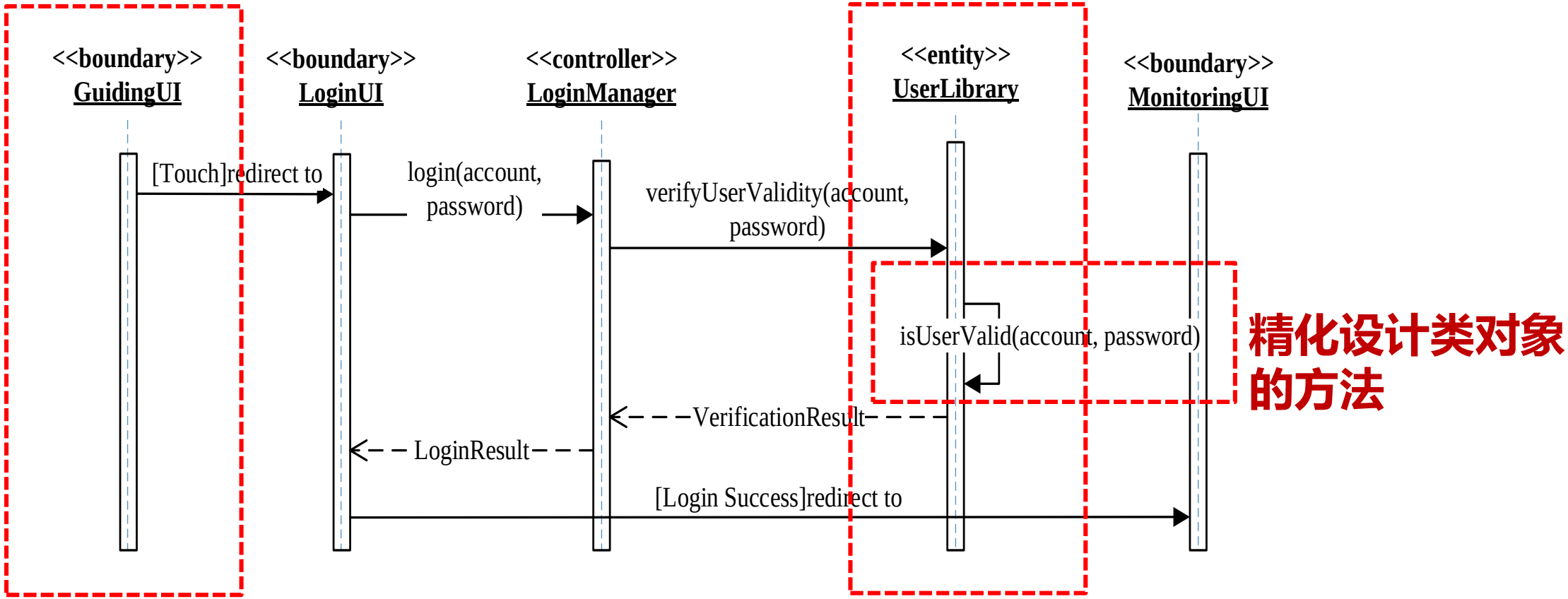


示例1：“用户登录”用例设计方案（设计）

界面设计类对象

设计类对象

界面设计类对象

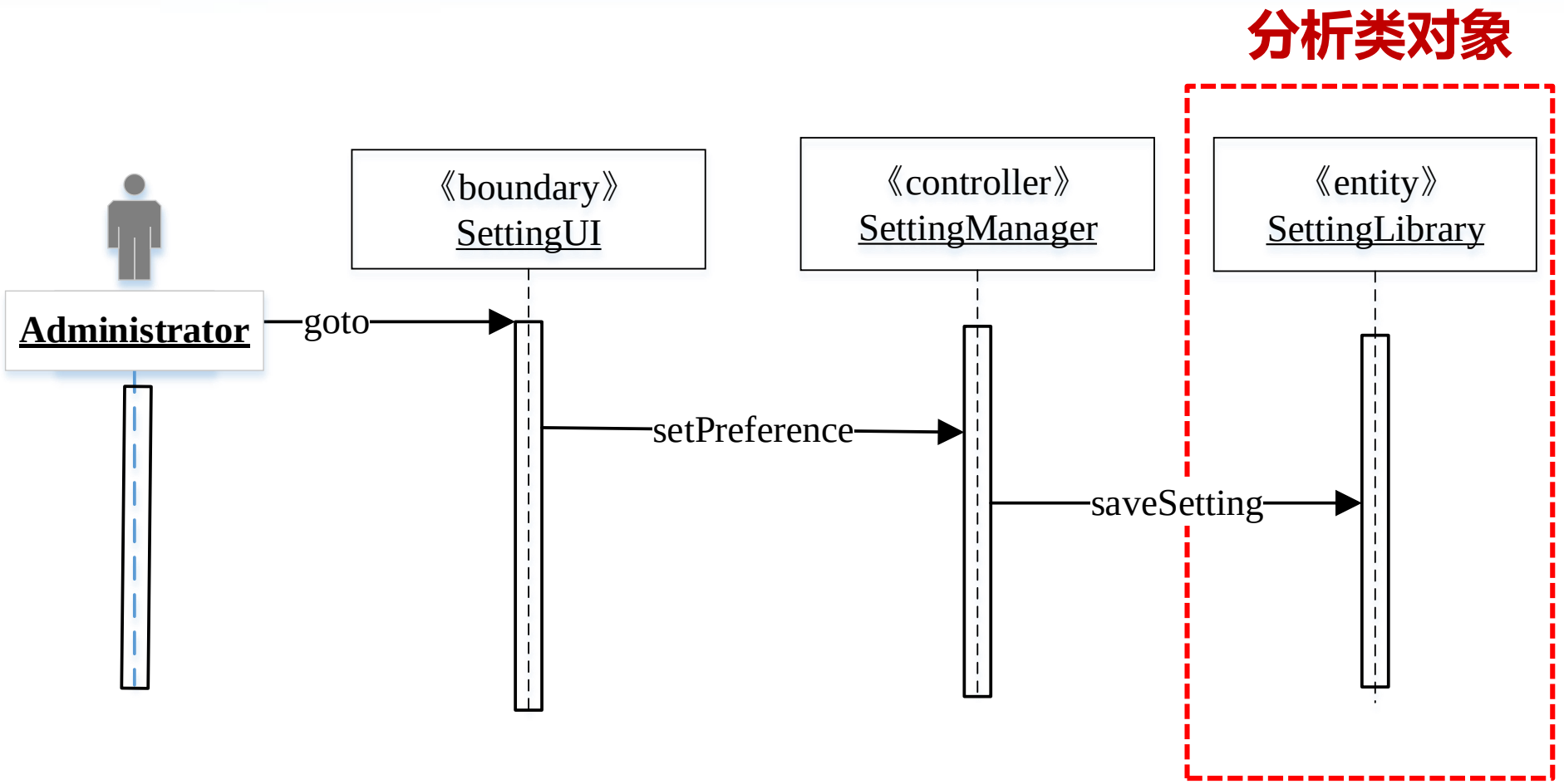


精化设计类对象的方法

设计元素及其交互

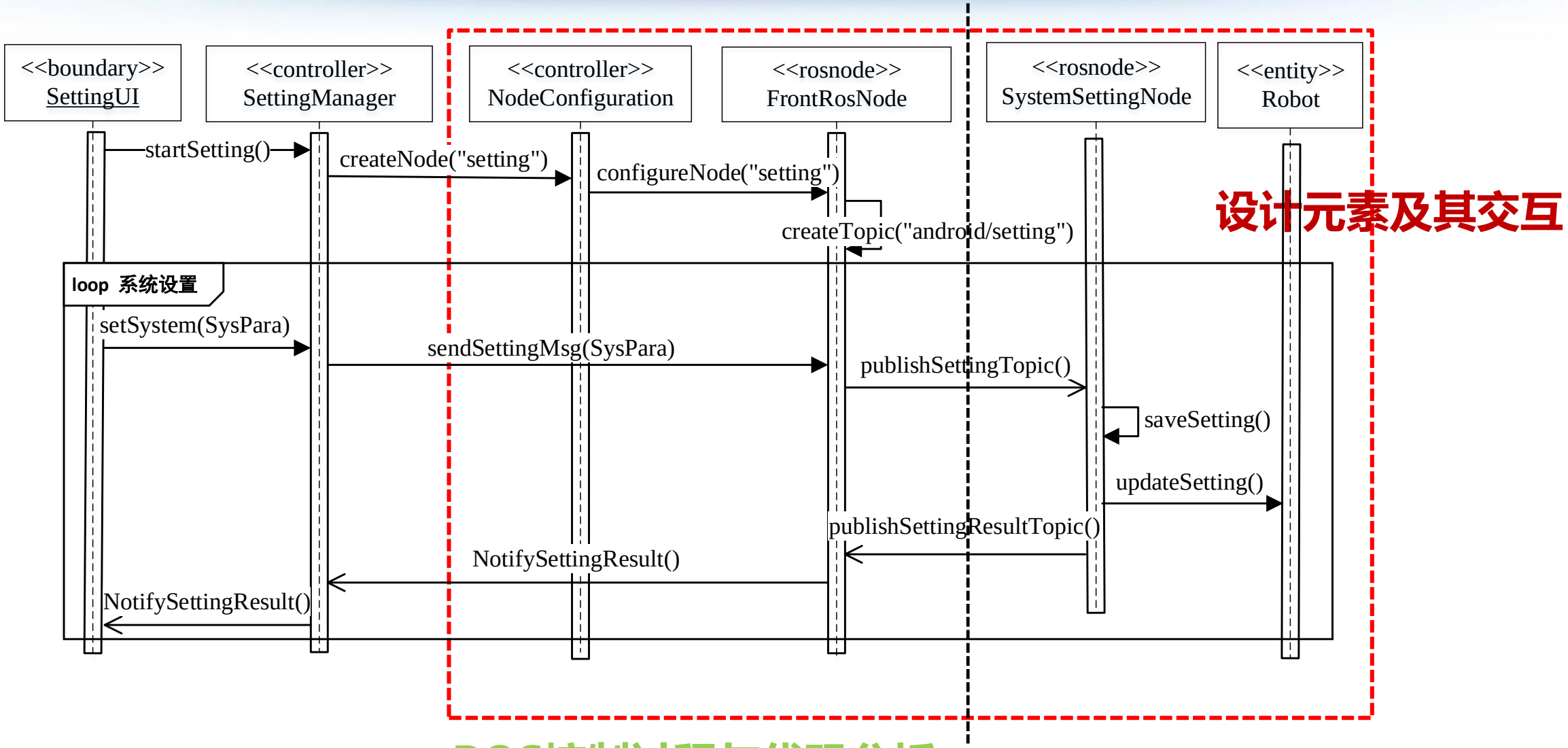
所有设计元素最终通过代码加以实现

示例2:“系统设置”用例的顺序图（需求）



设置系统参数，如机器人移动速度、安全距离大小等

示例2:“系统设置”用例设计方案（设计）

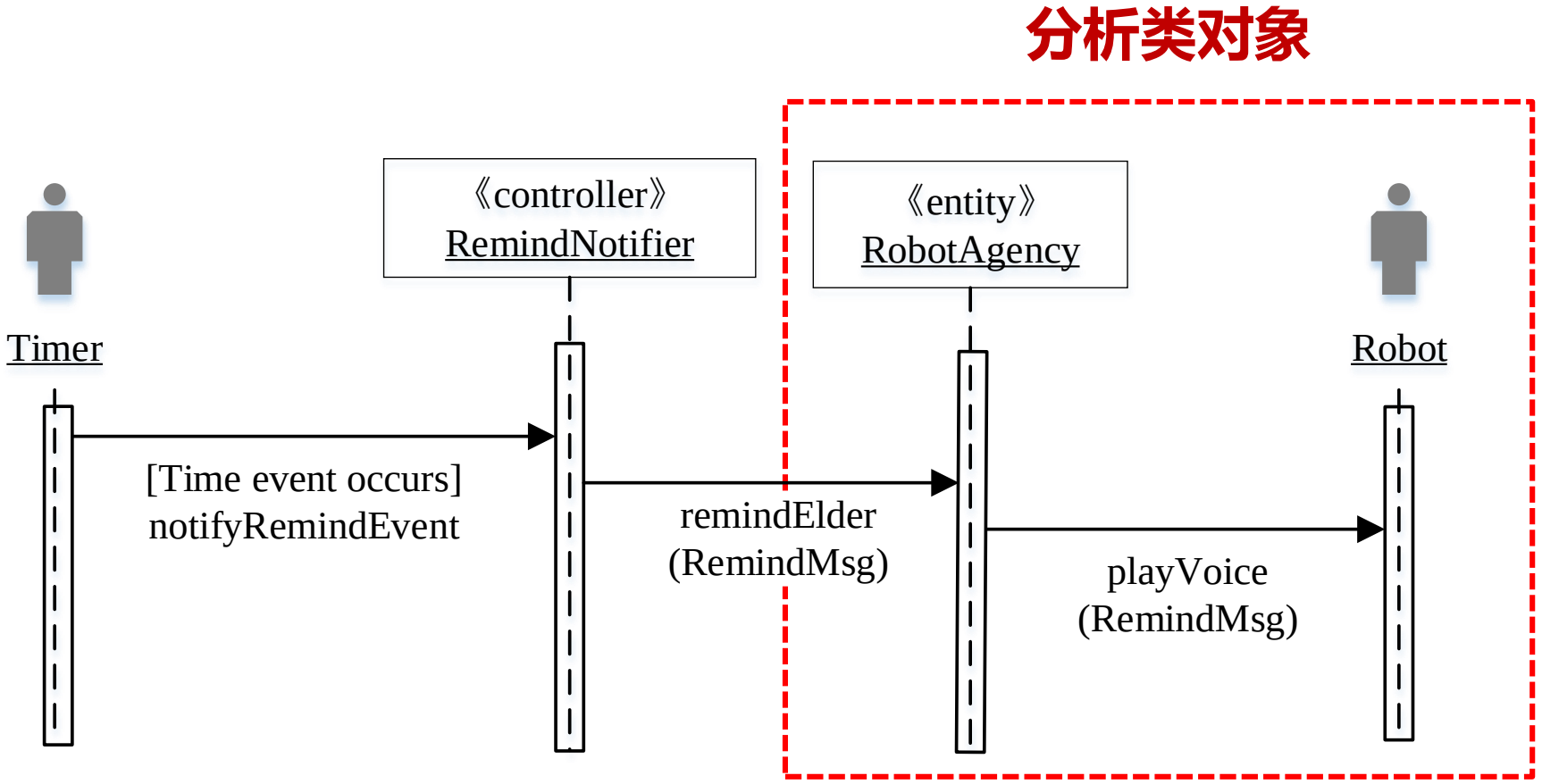


设计元素及其交互

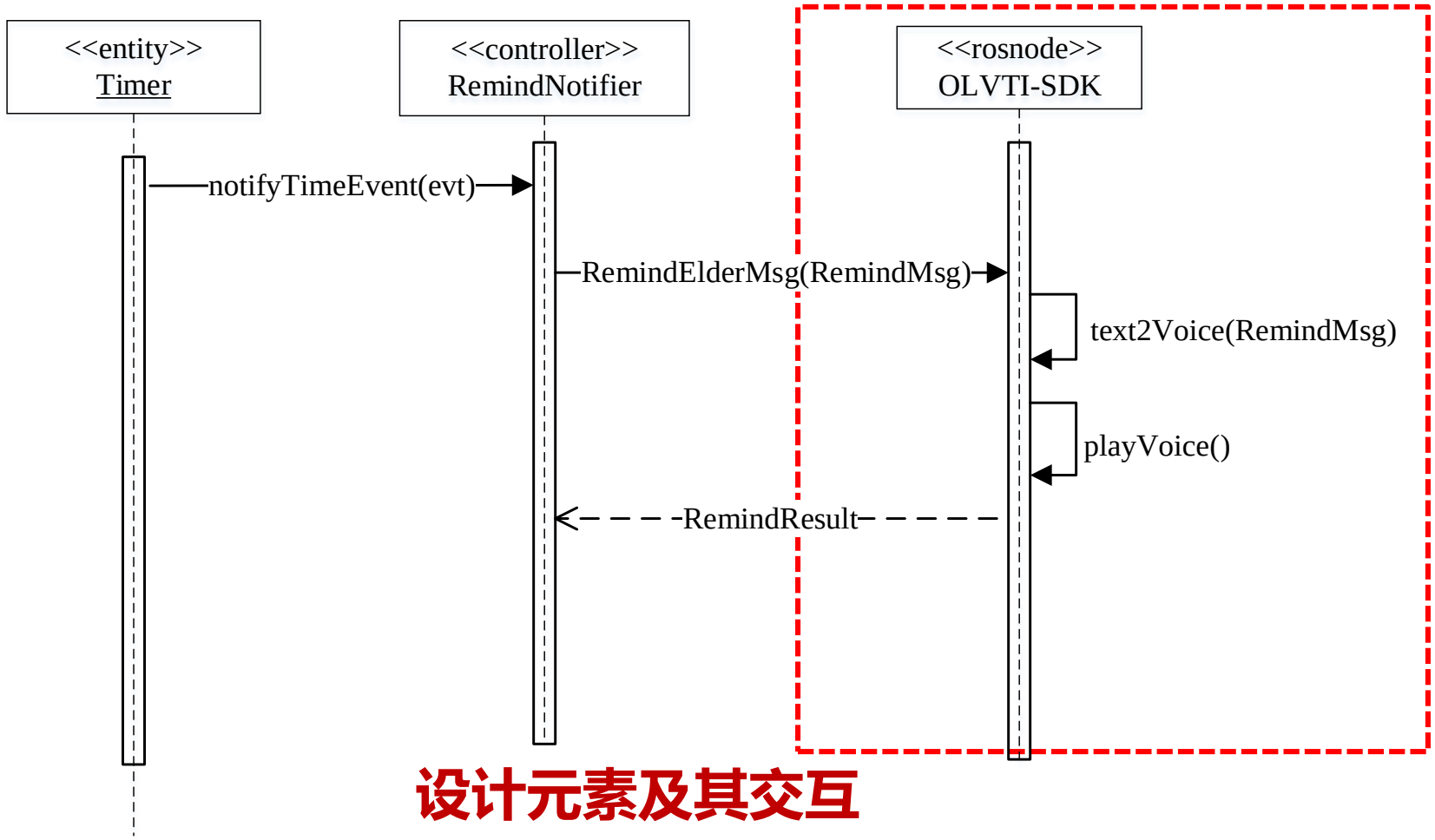
ROS控制过程与代码分析

所有设计元素最终通过代码加以实现

示例3:“提醒服务”用例的顺序图（需求）



示例3:“提醒服务”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

交互图对比

❑ 分析阶段的交互图

- ✓ **目的**：主要用于理解问题域和需求，以及系统如何响应外部事件或用户操作。
- ✓ **细节程度**：分析阶段的交互图较为抽象，重点在于描述分析类的职责和交互
- ✓ 如：网购商品用例中有分析类 “订单详情”

❑ 设计阶段的交互图

- ✓ **目的**：用于描述软件设计元素之间的交互，包括方法调用、消息传递等
- ✓ **细节程度**：设计阶段的交互图更加详细，更彻重于设计类和实际方法的调用与流程控制。
- ✓ 如：分析类“订单详情”需要细化为设计类“商品快照”、“商品详情”、“收货人信息”等。

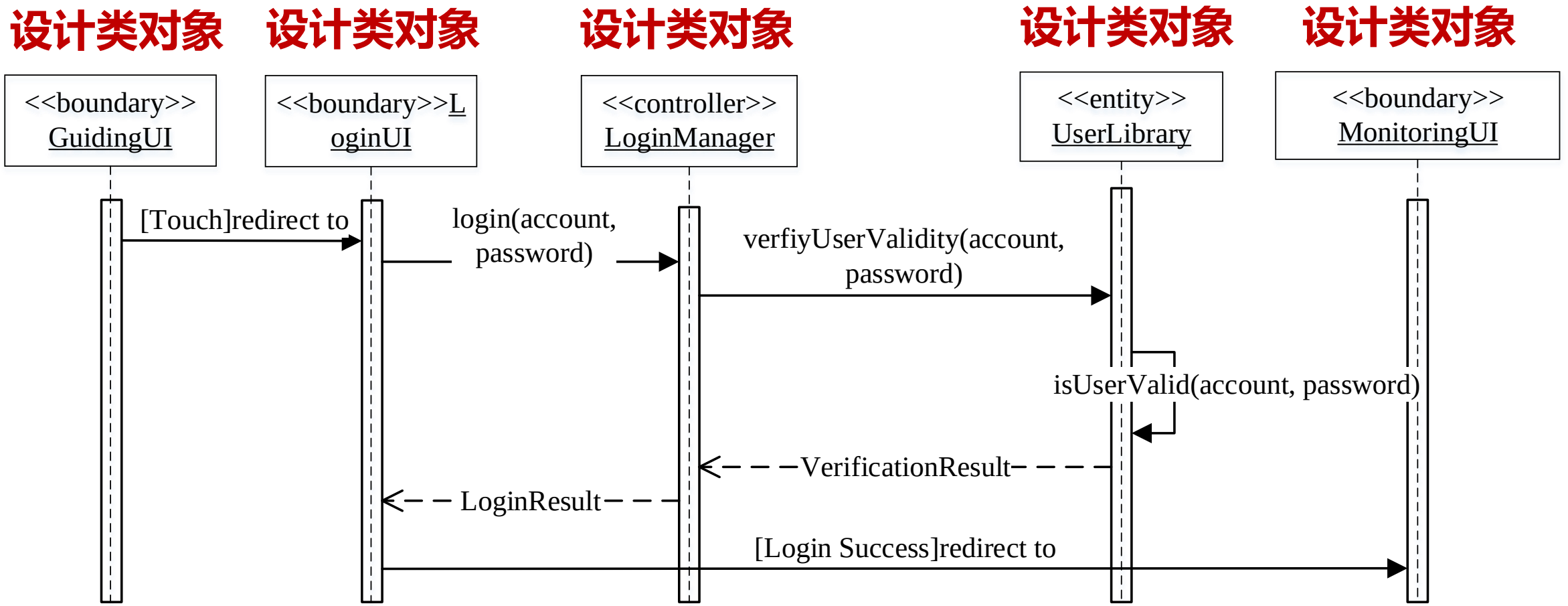
2. 构造设计类图

- 顺序图中对象所对应的类，将其抽象为**设计类图中的类**
- 如顺序图中对象a给b发送消息，则可以确定类B中应该具有相应的**职责和方法**，以处理该消息

- 根据顺序图中对象间消息来确定**设计类间的关系**
  - ✓如果对象a向对象b发消息，那么对应的类A与类B之间存在关联或者依赖关系

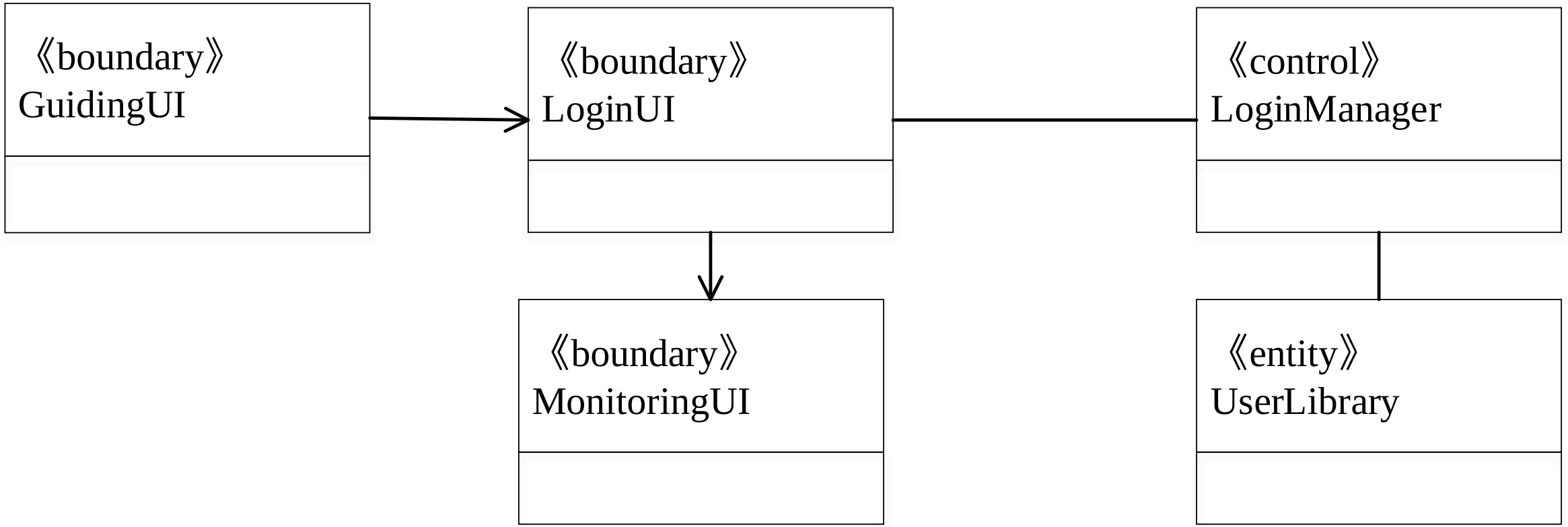
示例：“用户登录”用例实现的设计类图

❑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**



内容

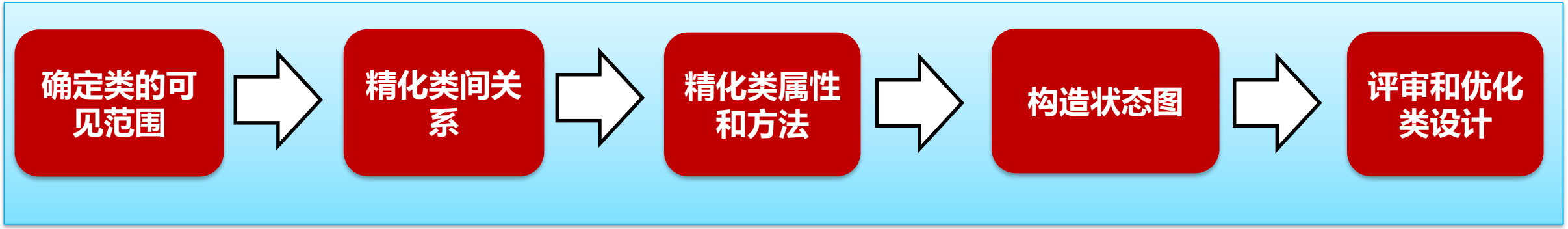
1. 软件详细设计概述
   - ✓任务
   - ✓过程
2. **软件详细设计活动**
   - ✓用例设计
   - **✓类设计**
   - ✓数据设计
   - ✓子系统和构件设计
3. **详细设计文档化和评审**

2.2 类设计

❑任务

- ✓对设计类图中的**界面类、设计类**等进行精化
- ✓明确设计类的内部实现细节

❑类设计过程



1. 确定类的可见范围

- 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

❑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

2. 精化类间的关系

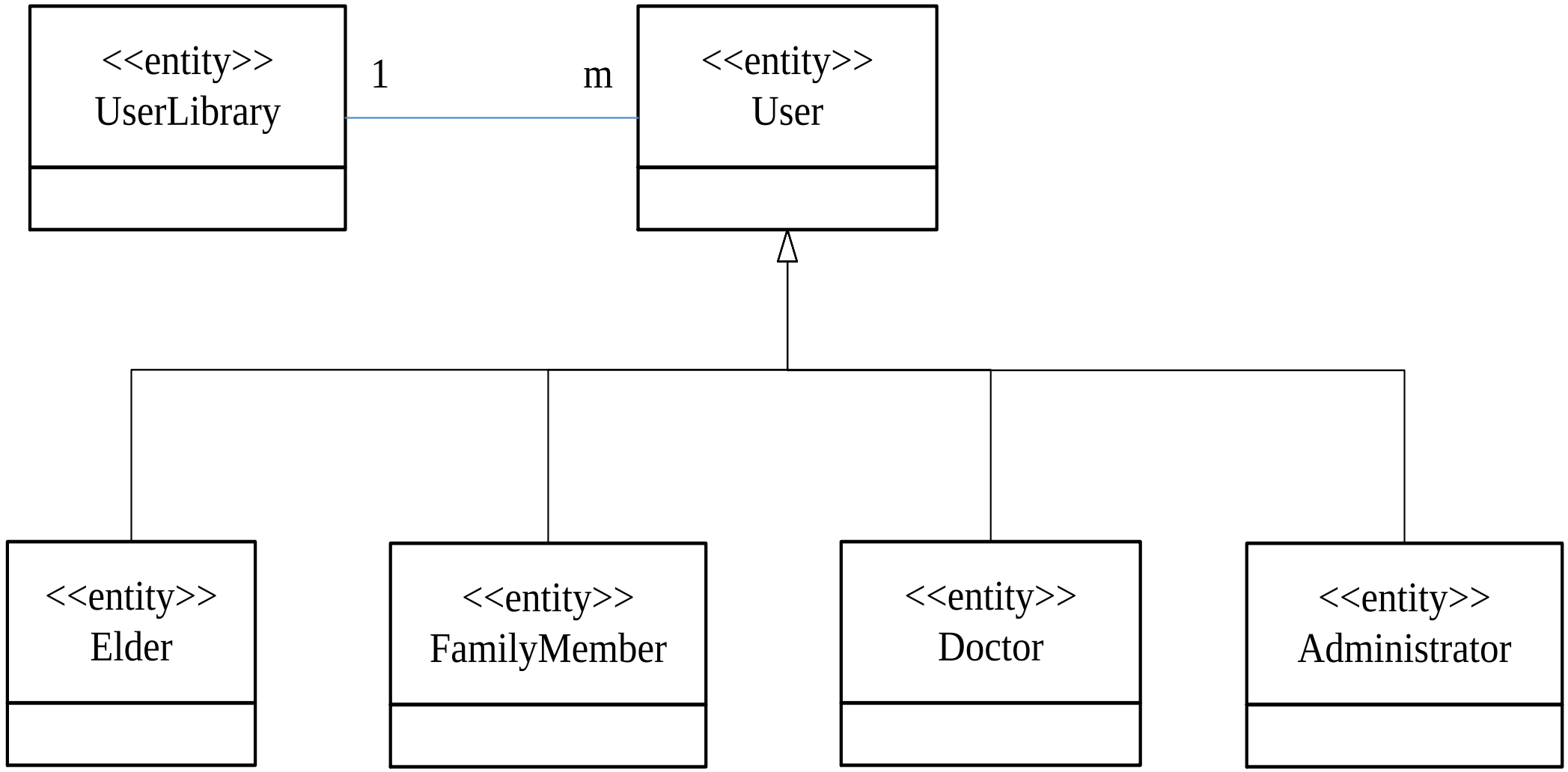
❑明确类间的语义关系

- ✓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**

❑如何区分类间关系

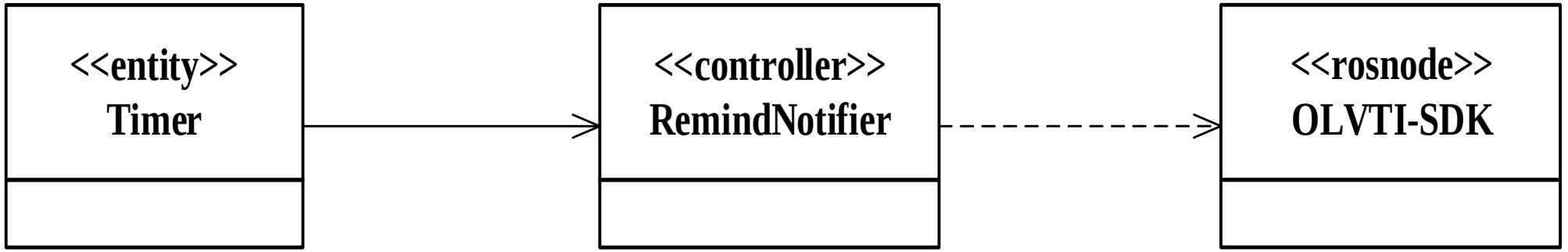
- ✓当一个类以成员变量形式出现在另一类中，两者是关联关系；
- ✓一个类以局部变量的形式出现在另一类中，二者是依赖关系。
- ✓如果类之间具有一般和特殊的关系，那么它们之间存在继承关系

示例1：精化类间的关系


<<entity>>
UserLibrary
1
m
<<entity>>
User
<<entity>>
Elder
<<entity>>
FamilyMember
<<entity>>
Doctor
<<entity>>
Administrator

示例2：精化类间的关系

❑精化关键设计类间的关系



精化“机器人感知与控制”子系统中类间的关系

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）类属性的设计

- 说明属性的名称、类型、可见范围、缺省值等
- 结合类关系来精化类属性设计(见第6章类图)

✓如果类A与类B间存在1:1关联或聚合关系，那么在A中设置类型为B的指针或引用（reference）的属性

✓如果类A到类B间存在1:n关联或聚合关系，那么在A中设置一个集合类型（如列表等）的属性，集合元素的类型为B的指针或引用

✓如果类A与类B间存在1:1的组合关系，那么在A中设置类型为B的属性

✓如果类A到类B间存在1:n的组合关系，那么在A中设置一个集合类型（如列表等）的属性，集合元素的类型为B

示例：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）类方法的设计

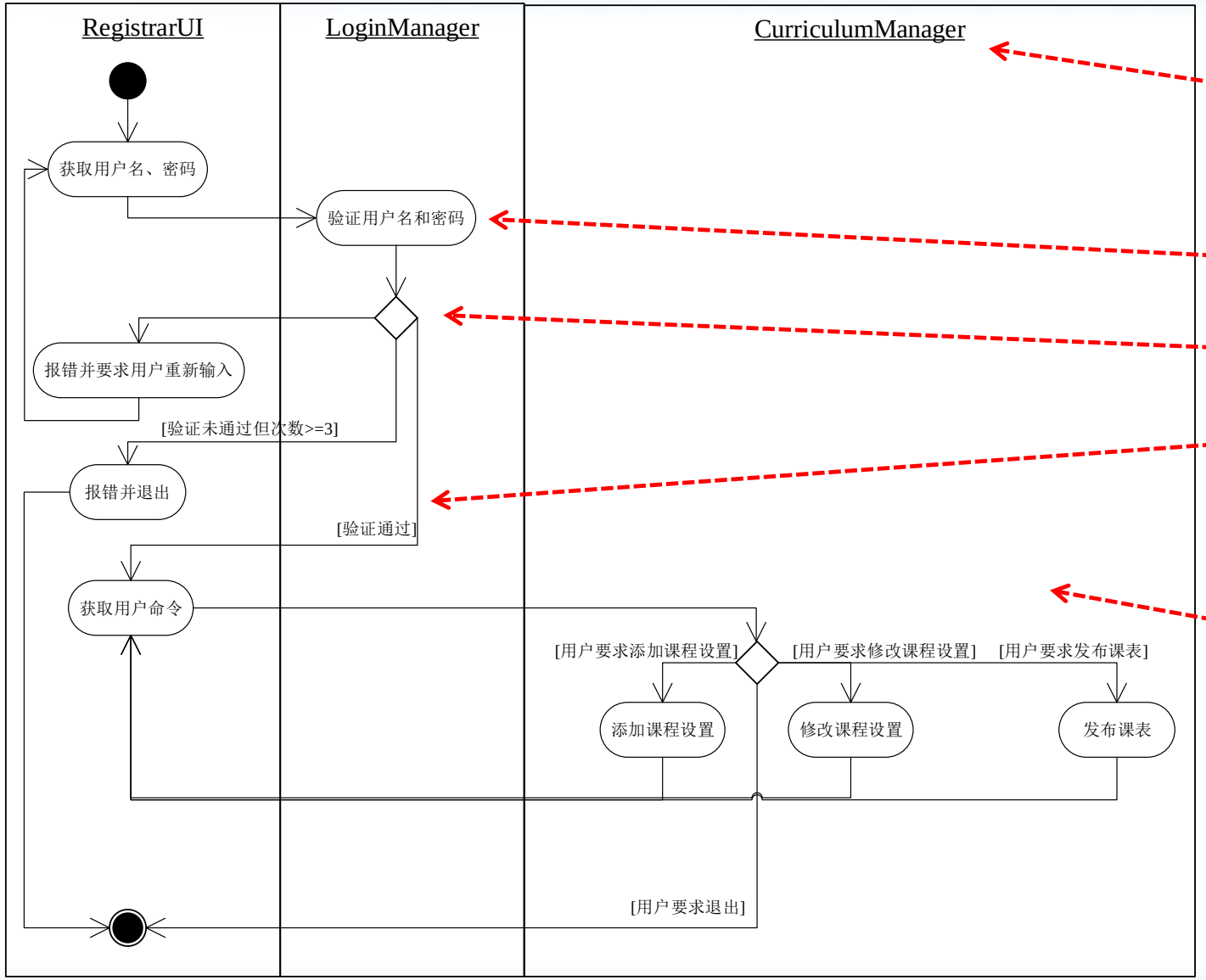
- 细化类中各个方法的描述
  - ✓方法名称
  - ✓参数表
  - ✓返回类型
  - ✓作用范围
  - ✓功能描述
  - ✓实现算法(UML活动图)
  - ✓前提条件（pre-condition）、出口断言（post-condition）等

何为活动图

- 描述实体为完成某项功能而执行的操作序列

- 可用于用例执行流程（作用类似交互图） 或 算法的描述

示例：活动图



对象

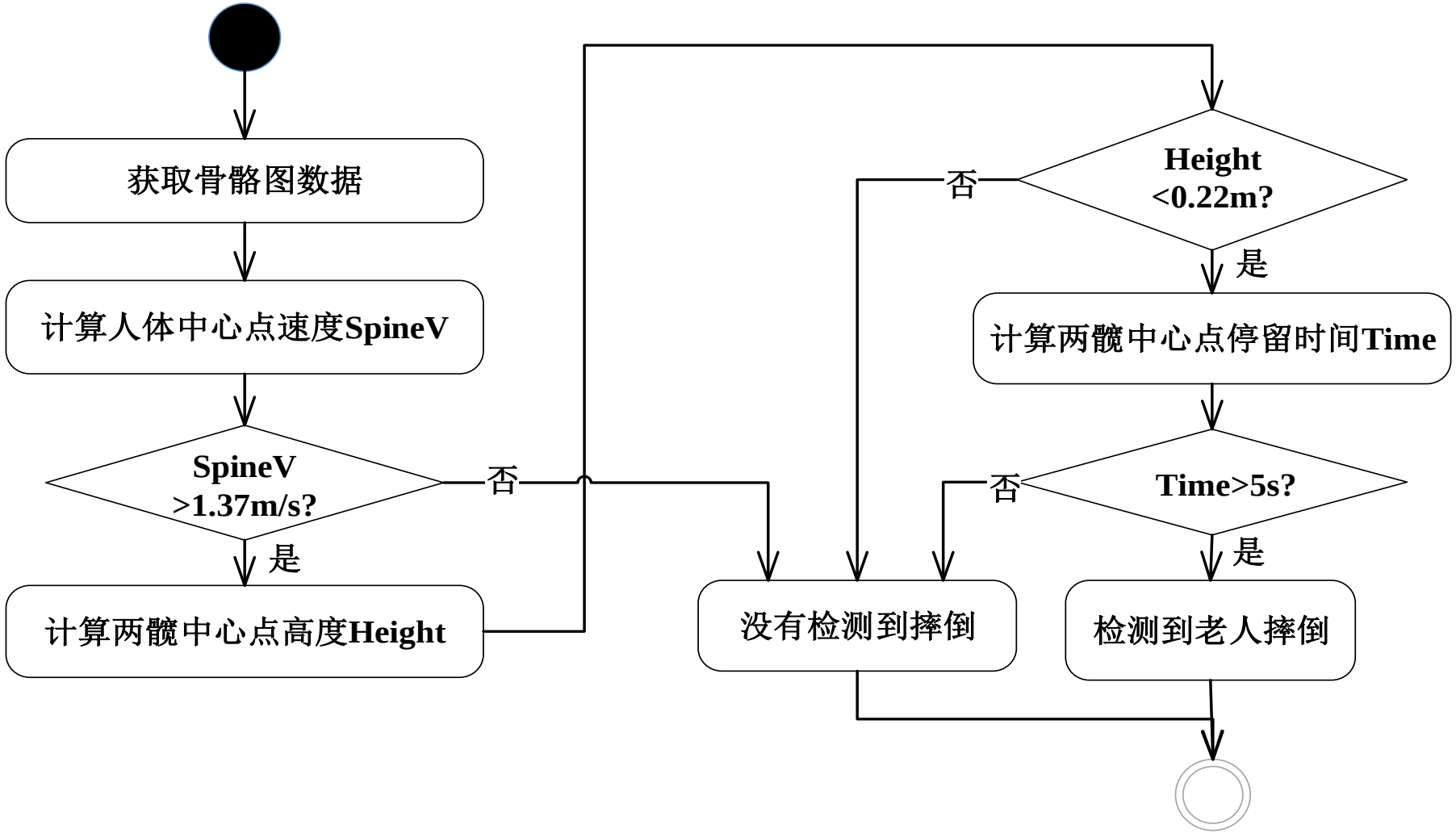
✓ 活动

✓ 决策点

✓ 信息流和控制流

✓ 泳道

示例：精化detectFallDown()方法的详细设计

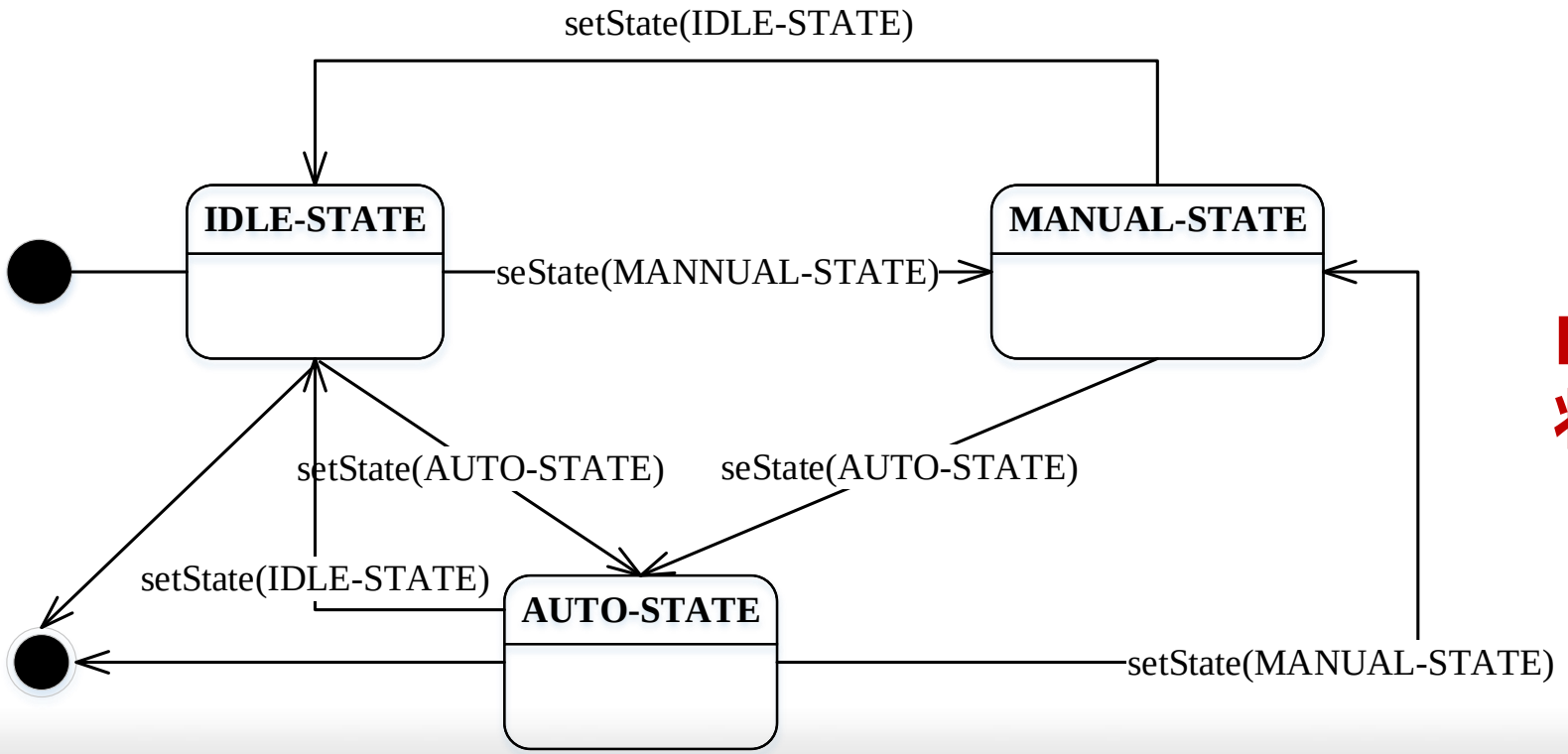


ElderInfoAnalyzer中方法的详细算法设计

4. 构造类对象的状态图

❑状态图

- ✓如果一个类的对象具有较复杂的状态，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，可以考虑构造类的状态图



Robot类的状态图

5. 评审和优化类设计

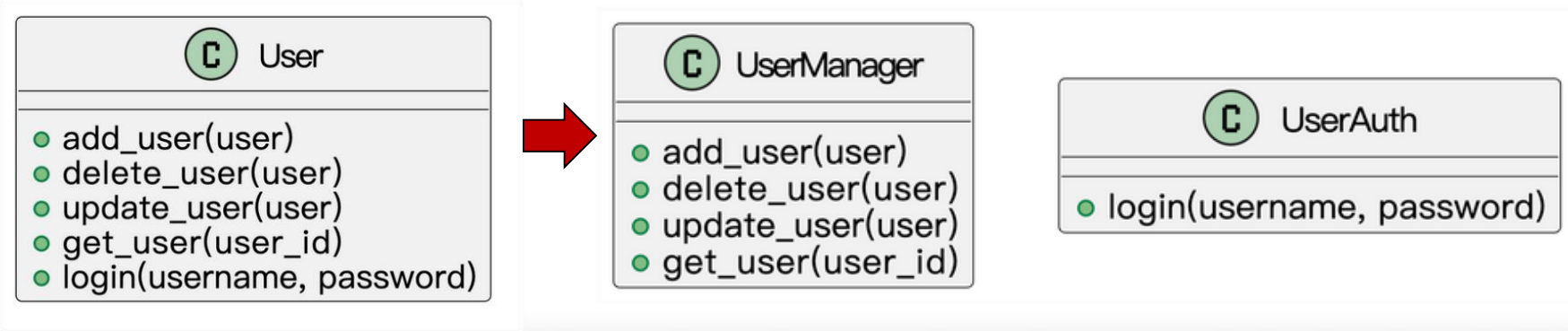
- 根据“高内聚、松耦合”的原则，判断设计的质量，必要时可以应用**面向对象设计原则**进行结构优化

- 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
- 按照简单性、自然性等原则，评判类间的关系是否恰当地反映了类之间的逻辑关系
- 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。

面向对象设计原则

- 为了满足高内聚（类内成员关系紧密）、低耦合（类间关系松散）的目标，可以根据面向对象六大设计原则进行类的设计和优化。

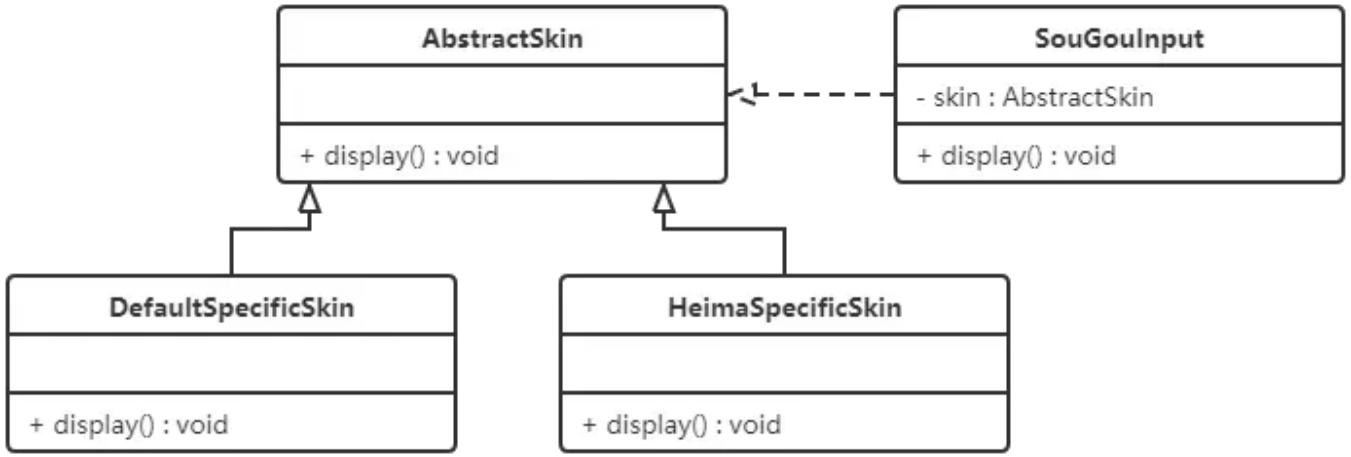
- （1）单一职责原则
  - ✓核心思想： 一个类只应承担一类职责。此原则的核心就是解耦和增强内聚性。
  - ✓下列User 类既负责用户的增删改查，又负责用户的登录验证，这违背了单一职责原则。



面向对象设计原则

（2）开闭原则

- ✓核心思想： 每个类应该对扩展开放，对修改封闭。
- ✓这意味着应该通过扩展已有类来实现新功能，而不是通过修改已有类。这样可以保护已有代码的稳定性。
- ✓实现途径：用抽象构建框架，用继承/实现扩展细节



输入法可以使用不同皮肤，每种皮肤都是AbstractSkin的子类，用户可以任意增加或更改皮肤(子类)，而不需要修改原代码。

面向对象设计原则

（3）里式替换原则

- ✓核心思想：子类对象必须能替换其父类对象，且替换后不会影响程序的正确性。
- ✓这一原则要求子类在继承父类时，不能破坏父类的行为。
- ✓通俗来讲就是：子类继承父类时，除添加新的方法完成新增功能外，尽量不要重写父类的（非抽象）方法。
- ✓该规则可以帮助我们判断什么时候应该使用继承，什么时候不应该使用继承。

【例】从里式替换原则看，鸵鸟不是鸟，不能建立为继承关系，因为鸟都有“飞”的功能，鸵鸟不具备，因此无法替换父类（鸟）对象。

面向对象设计原则

- （3）里式替换原则

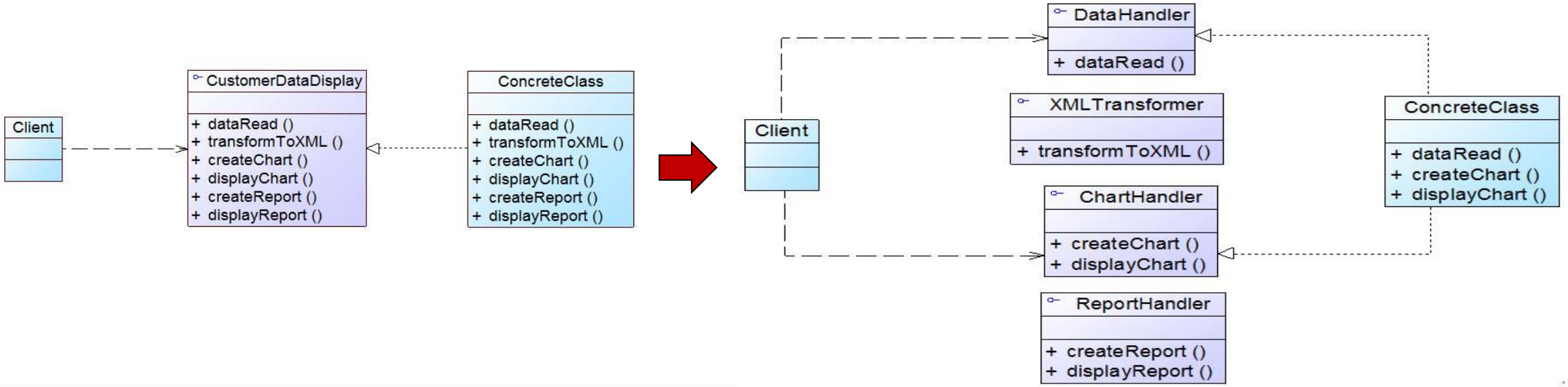
```
class Bird {
     public void fly() ;
}

public class Ostrich extends Bird { //鸵鸟
       public void fly() { …….. }
      //这里事实上已经改变了fly，因为鸵鸟不能飞，与父类fly的行为不同
}
public class Test {
   public static void showFly(Bird bird) {   //子类对象能替换父类对象
          bird.fly();
    }
   public static void main(String[] args) {
          showFly(new Ostrich());
   }
}
```

面向对象设计原则

（4）接口隔离原则

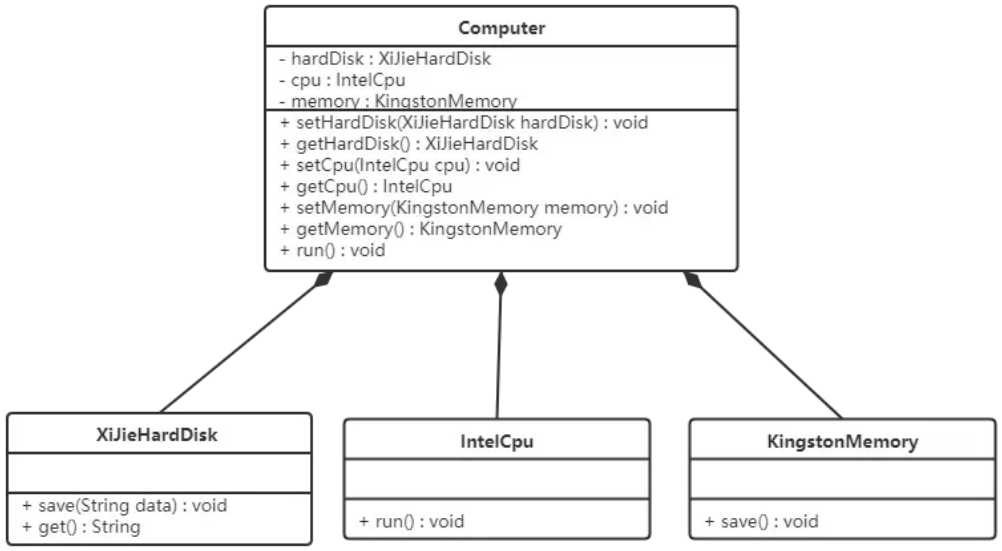
- ✓核心思想：接口应该尽可能小，以解开耦合。
- ✓这一原则要求将庞大的接口拆分成更小的、更具体的接口，每个接口只包含客户端需要的方法。
- ✓这样做可以减少接口的冗余和复杂性，提高系统的可维护性和灵活性。

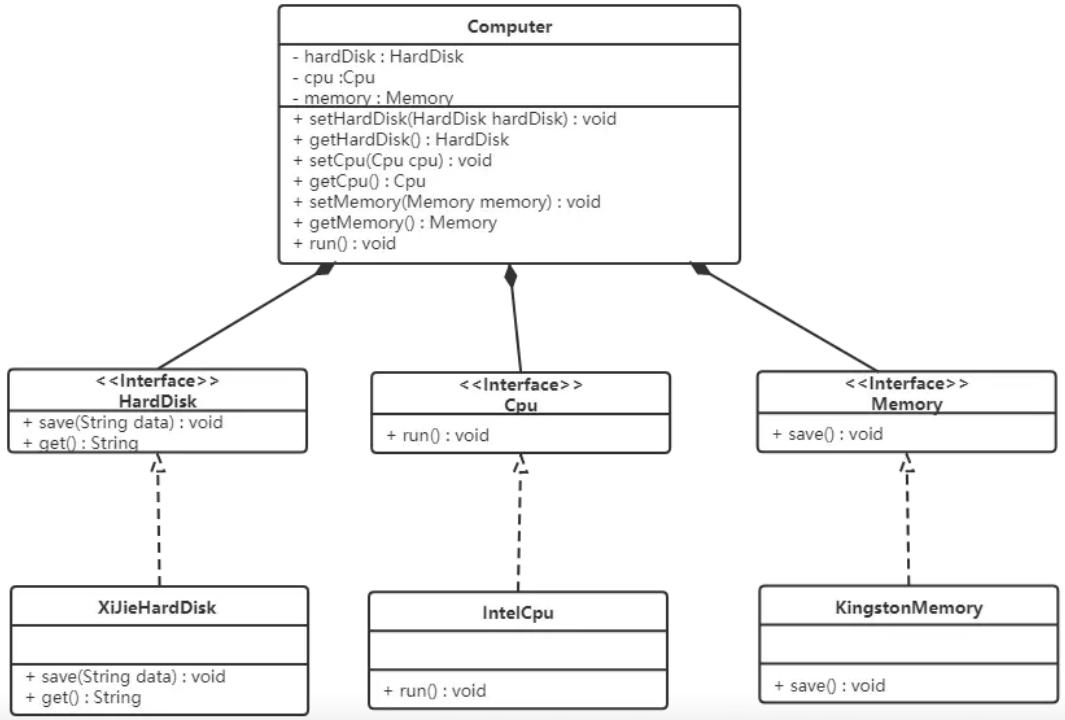


面向对象设计原则

（5）依赖倒置原则

- ✓核心思想：高层模块不应该依赖低层模块，它们都应该依赖于抽象。
- ✓依赖于抽象，而非具体实现。
- ✓这样做可以降低模块之间的耦合度，提高系统的灵活性和可扩展性



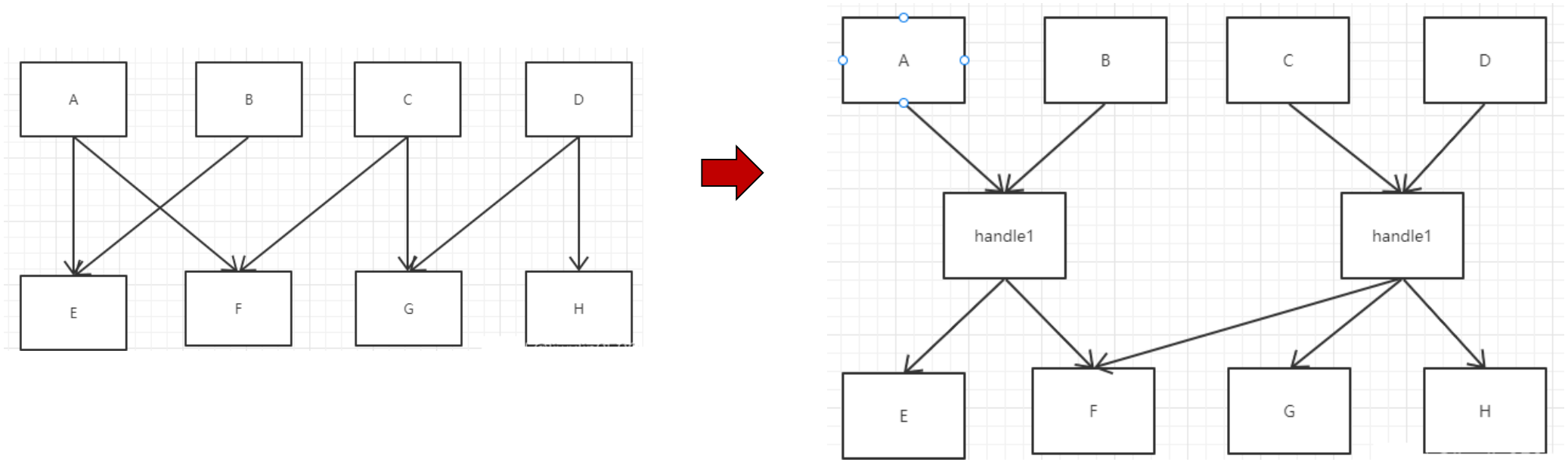


一台电脑由希捷硬盘、intelCPU、金斯顿内存条组成
如果换成其它品牌硬盘，CPU, 怎么办？

面向对象设计原则

（6）最少知识原则

- ✓核心思想：即一个对象应该对其它对象有最少的了解，从而尽量避免与不相关的对象交互。
- ✓这一原则有助于减少类之间的耦合，从而提高系统的可维护性。
- ✓通过增加“中间类”将直接耦合的类隔开，后期只要修改中间类即可。



类设计的输出

1. 详细的类属性、方法和类间关系设计的**类图**
2. 描述类方法实现算法的**活动图**
3. 必要的**状态图**（可选）

内容

1. 软件详细设计概述
   - ✓任务
   - ✓过程
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**

为什么要进行数据设计

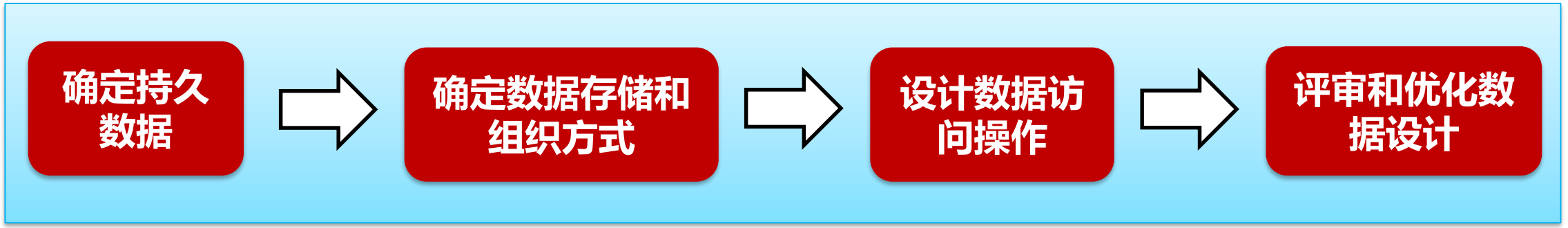
- **软件系统有些数据需要持久保存**
  - ✓对此，需要开展数据设计，以支持信息的抽象、组织、存储和读取

- **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

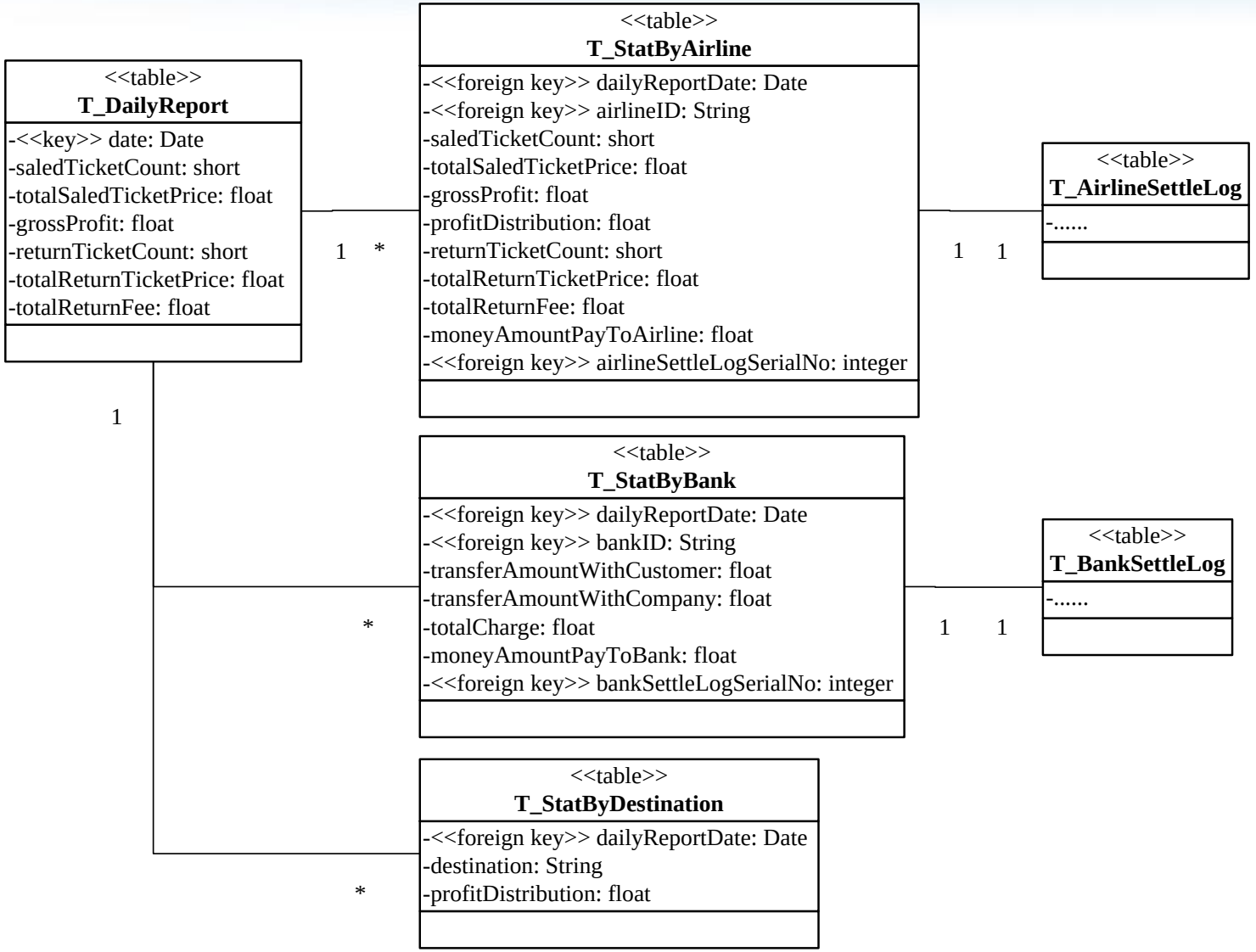
2.3 数据设计

❑任务

✓针对UML类图中需要持久保存的数据，将其映射为数据库模型，即**数据表**以及这些**数据表之间的关系**



示例：数据模型设计



数据库的表以及表之间的关系

1. 确定永久数据

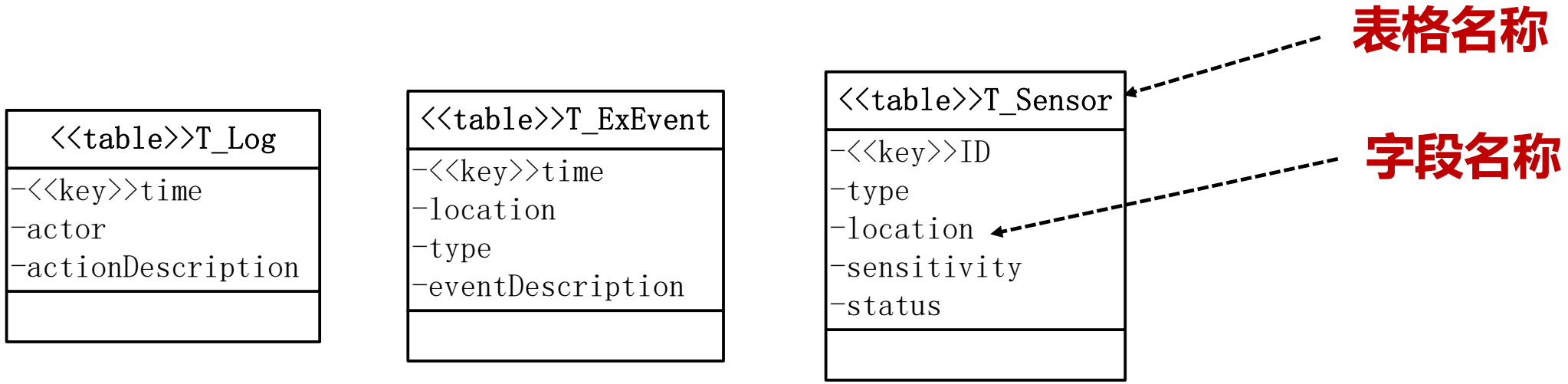
- **根据对需求确定哪些数据需要永久保存**
  - ✓如用户的账号和密码
  - ✓系统设置信息

2. 确定持久数据的存储和组织方式

- **将数据存储在数据文件中**
  - ✓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
- **将数据存储在数据库中**
  - ✓设计支持数据存储的数据库表

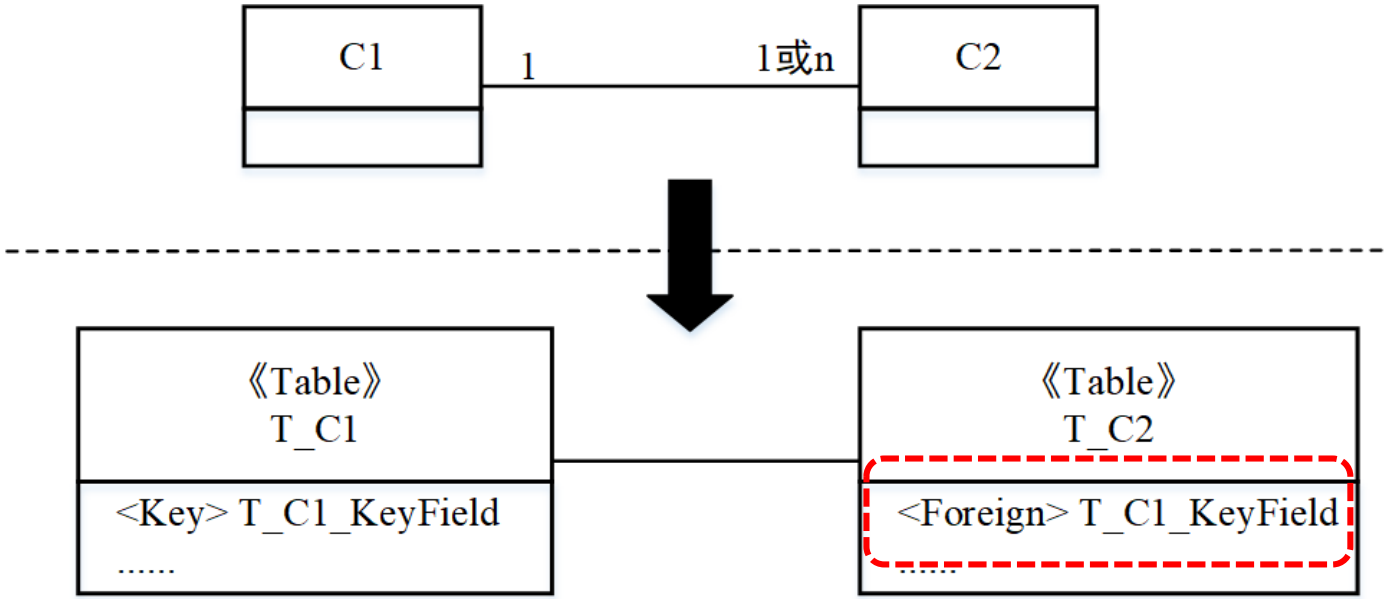
确定持久数据条目

- 利用设计模型与数据库关系模型的对应关系
  - ✓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）



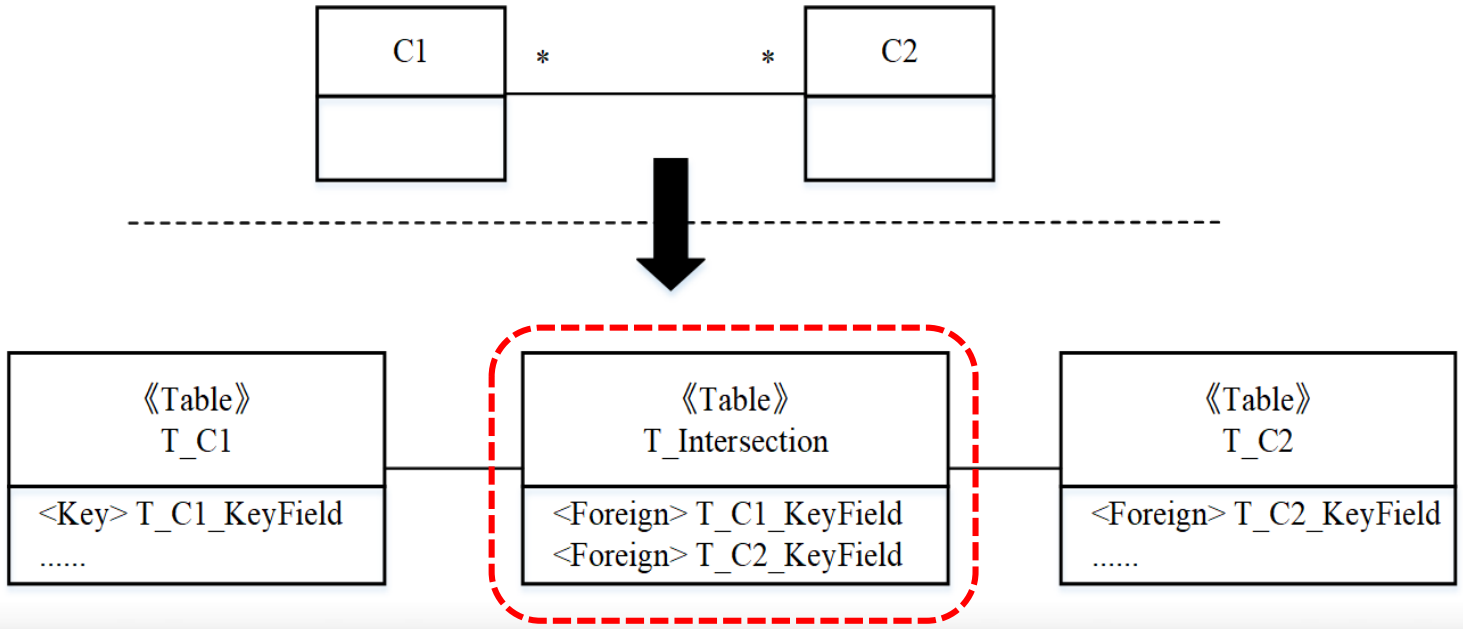
1:1、1:n关联关系的映射

- 假设类C1、C2对应的表分别为T_C1、T_C2
- 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

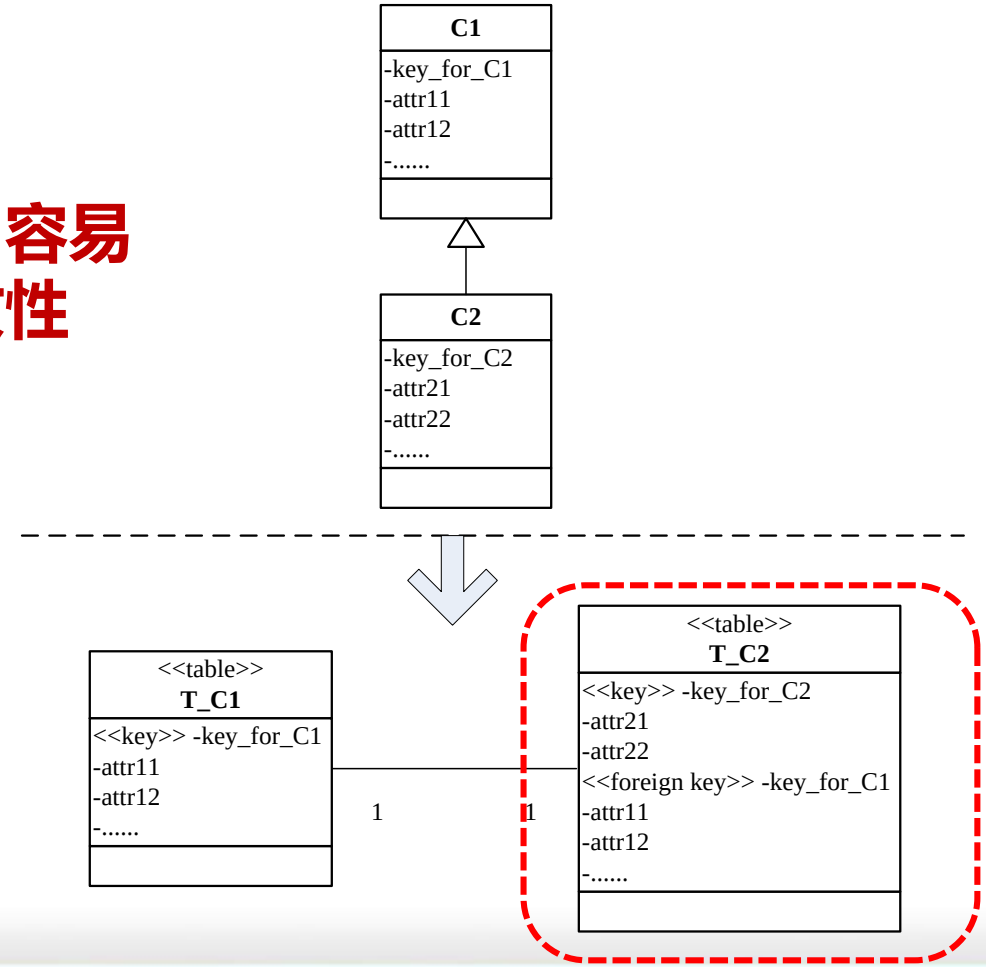
- 在T_C1、T_C2间引进新交叉表T_Intersection，将T_C1、T_C2关键字段纳入T_Intersection中作为外键，
- 将多对多转化为2个一对多关系



继承关系的数据库表设计(1/2)

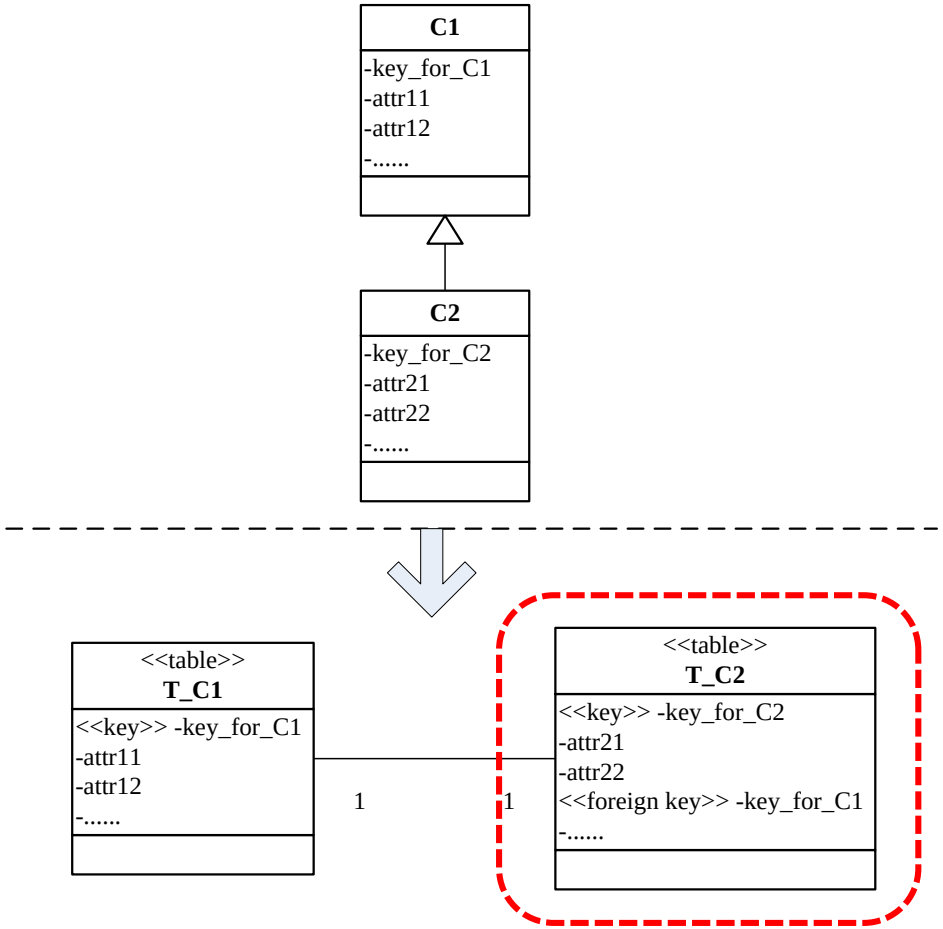
- 假设C1是C2的父类，将T_C1中的所有字段全部引入至T_C2

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



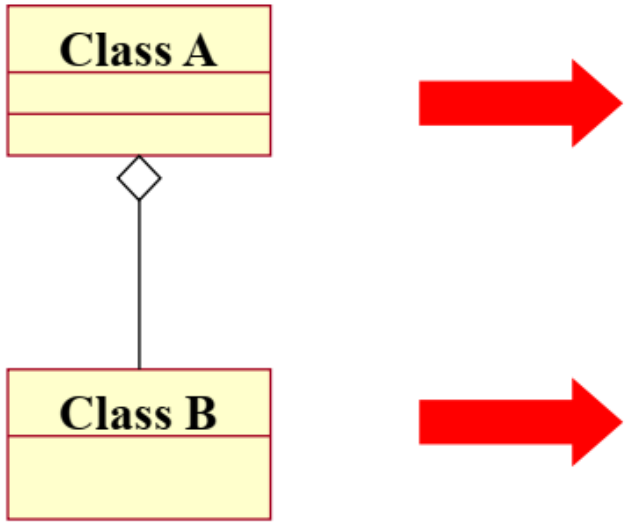
继承关系的数据库表设计(2/2)

- 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
- 避免数据冗余，但在读取C2对象时性能不如前种方法
- 如果要获取C2对象的全部属性，需要联合T_C2和T_C1



组合/聚合关系的数据库表设计

- 类似于1:n的关联关系
- 部分表中，其关键字由两个表的关键字组合而成



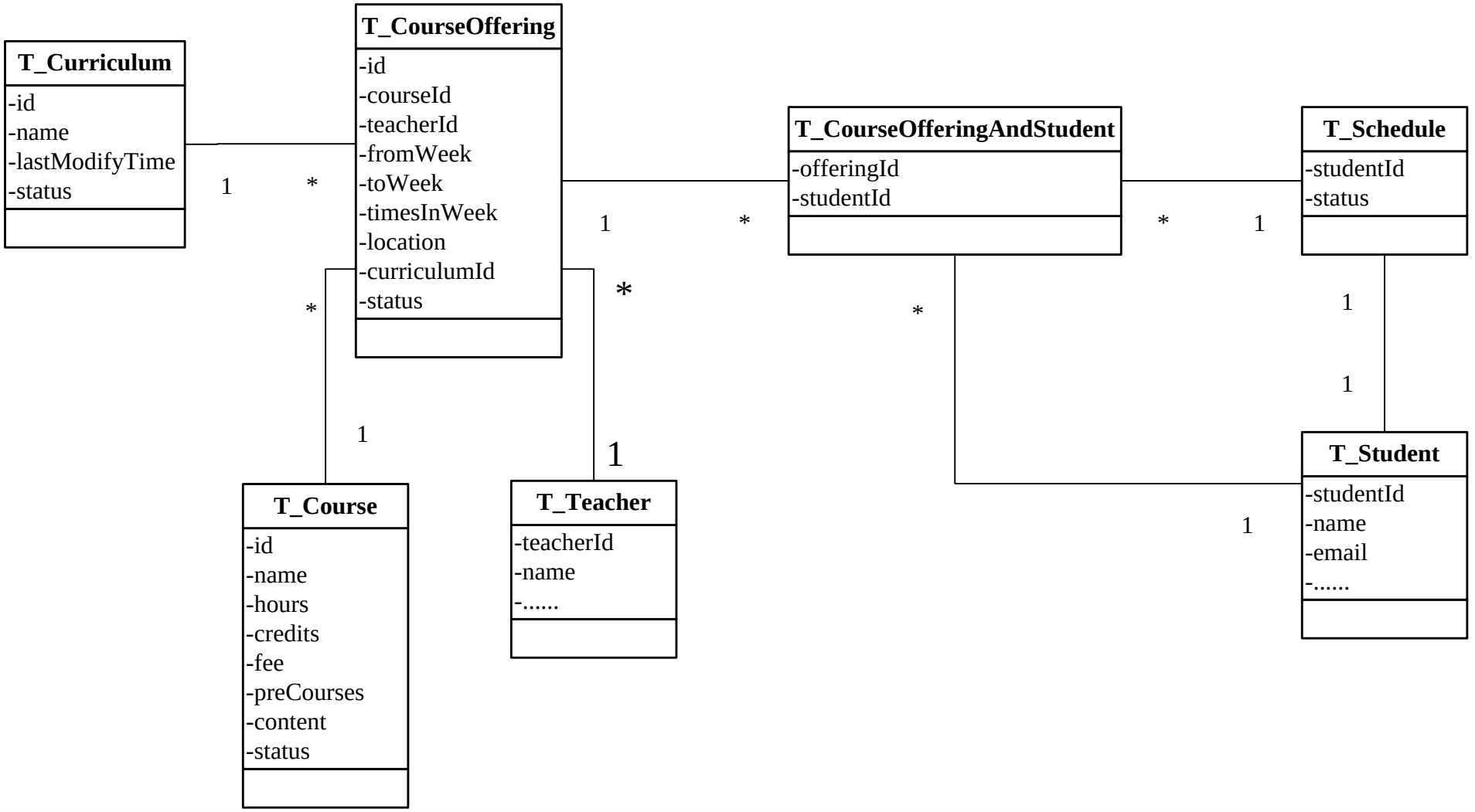
“整体”表

A-ID	X	Y
:	:	:

“部分”表

A-ID	B-ID	x	y
:	:	:	:

示例：设计持久数据

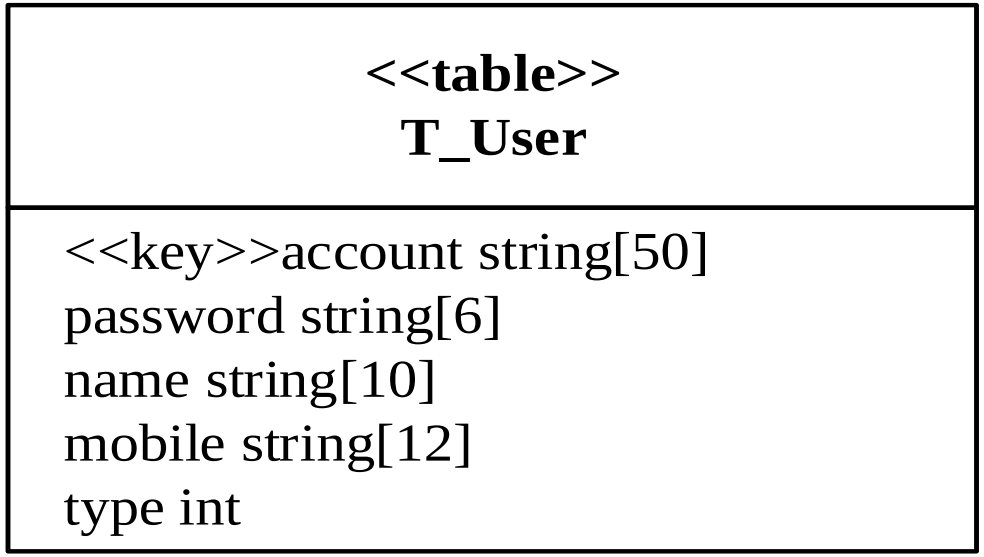

T_Curriculum
-id
-name
-lastModifyTime
-status
T_CourseOffering
-id
-courseId
-teacherId
-fromWeek
-toWeek
-timesInWeek
-location
-curriculumId
-status
T_CourseOfferingAndStudent
-offeringId
-studentId
T_Schedule
-studentId
-status
T_Course
-id
-name
-hours
-credits
-fee
-preCourses
-content
-status
T_Teacher
-teacherId
-name
-.....
T_Student
-studentId
-name
-email
-.....
1
*

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组合而成的其它操作**
  - ✓**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
  - ✓**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()



对象关系映射

❑对象关系映射（Object Relational Mapping,ORM）

- ✓用于在关系数据库和对象程序语言之间转换数据。
- ✓ORM 框架允许开发者以面向对象的方式来操作数据库，而不需要编写原生的 SQL 语句。

❑常见的 ORM 框架

- ✓ Hibernate、mybatis（Java）
- ✓ Entity Framework（.NET）
- ✓ Django ORM（Python/Django）

对象关系映射

❑基于Hibernate实现数据表操作

```
public static void main(String[] args) {
  Configuration cfg=new Configuration();
  cfg.configure(“hibernate.cfg.xml）
  SessionFactory sf = cfg.buildSessionFactory（）
  Session session = sf.openSession();
  Transaction ts= session.beginTransaction();
  User    user=new User(11, “张三");
  session.save(user);
  ts.commit();
}
```

数据库操作代码

```
<hibernate-mapping>
   <class name="test.User" table="user">
      <id name="id" type="java.lang.Integer"
column="id">
      <property name="userName"
type="java.lang.String" column="user_name"/>
   </class>
</hibernate-mapping>
```

Hibernate xml映射文件

内容

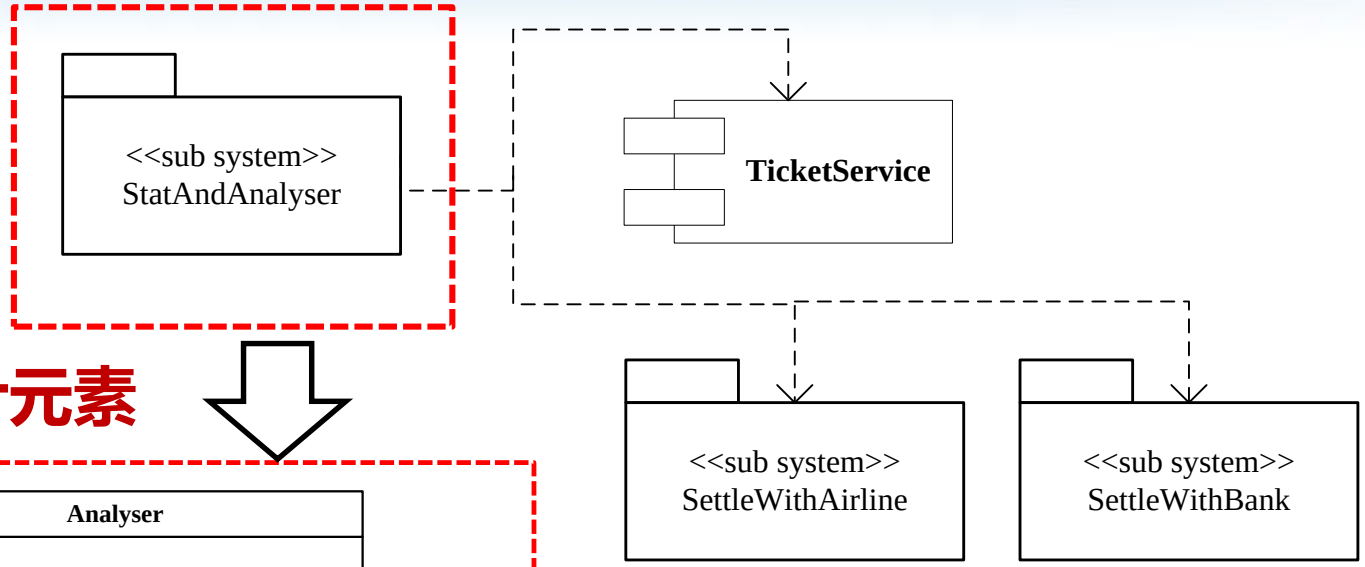
1. 软件详细设计概述
   - ✓任务
   - ✓过程
2. **软件详细设计活动**
   - ✓用例设计
   - ✓类设计
   - ✓数据设计
   - ✓子系统和构件设计
3. **详细设计文档化和评审**

2.4 子系统/构件设计

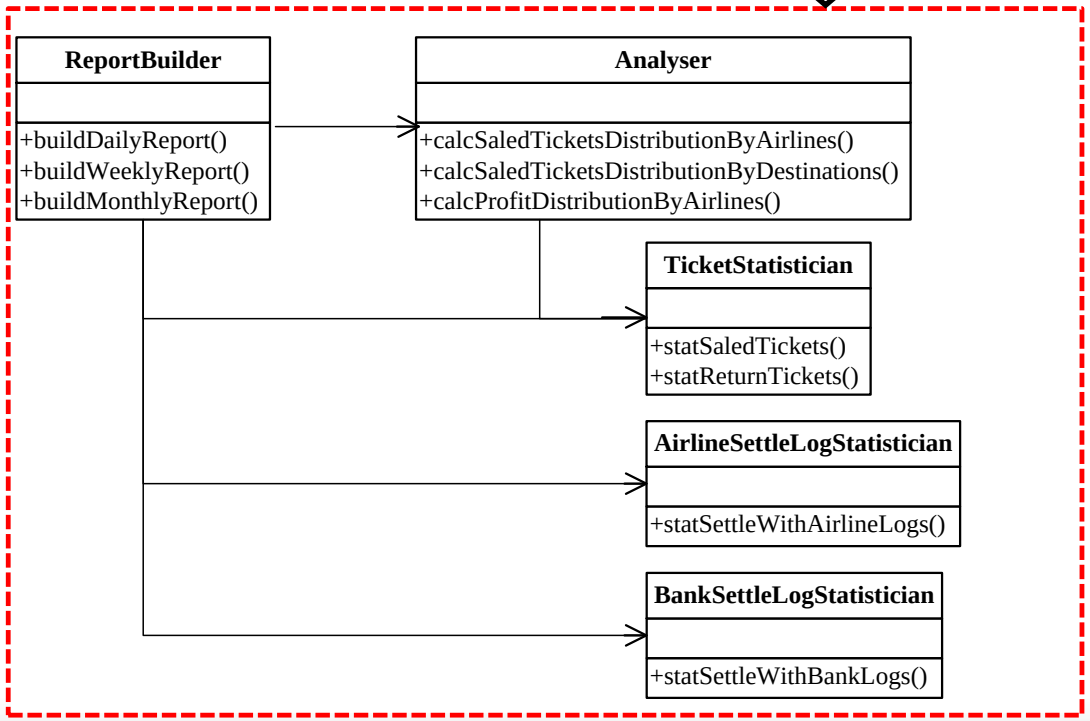
❑任务

- ✓细化子系统/构件的**内部结构**，找到其中更小粒度的**子系统、构件和设计类**，明确它们之间的**协作关系**
- ✓设计子系统/构件中的**类及接口**

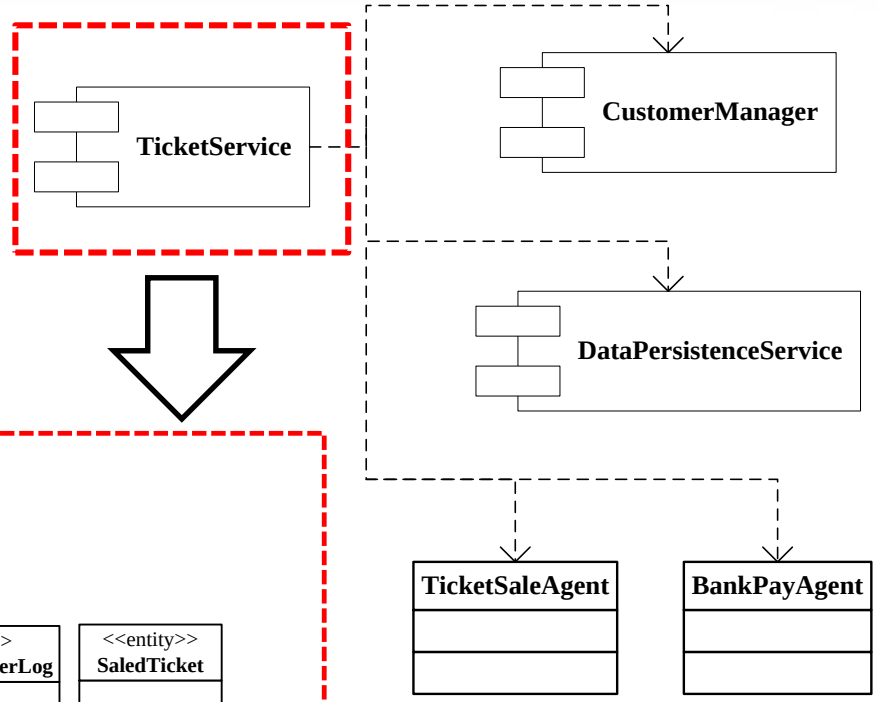
子系统设计的示例



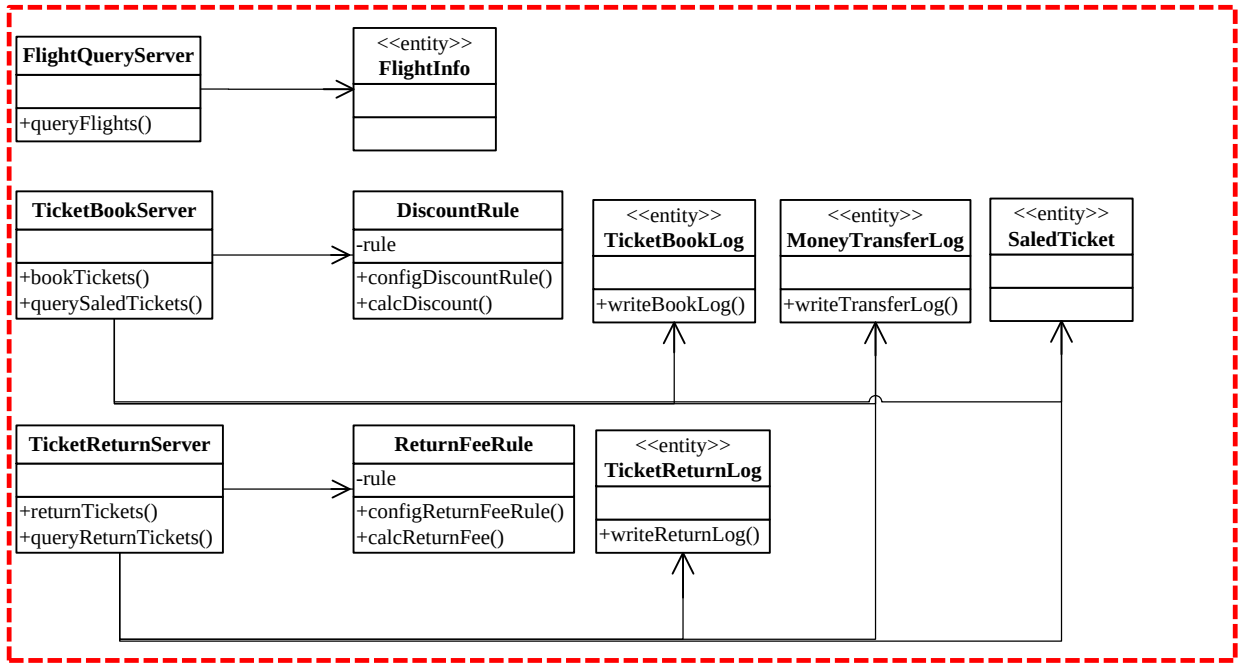
设计子系统内部的设计元素



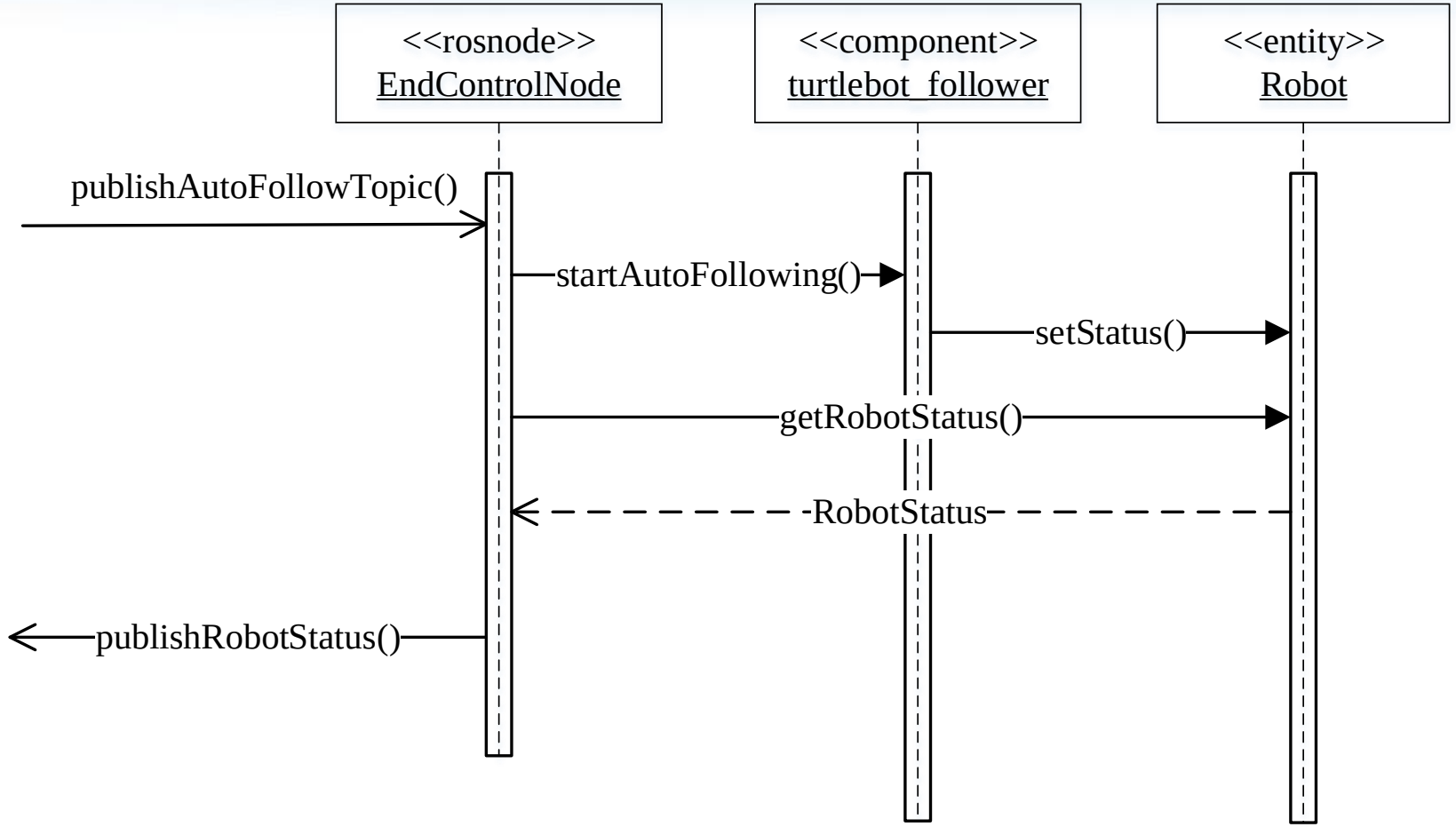
构件设计的示例



设计构件内部的设计元素

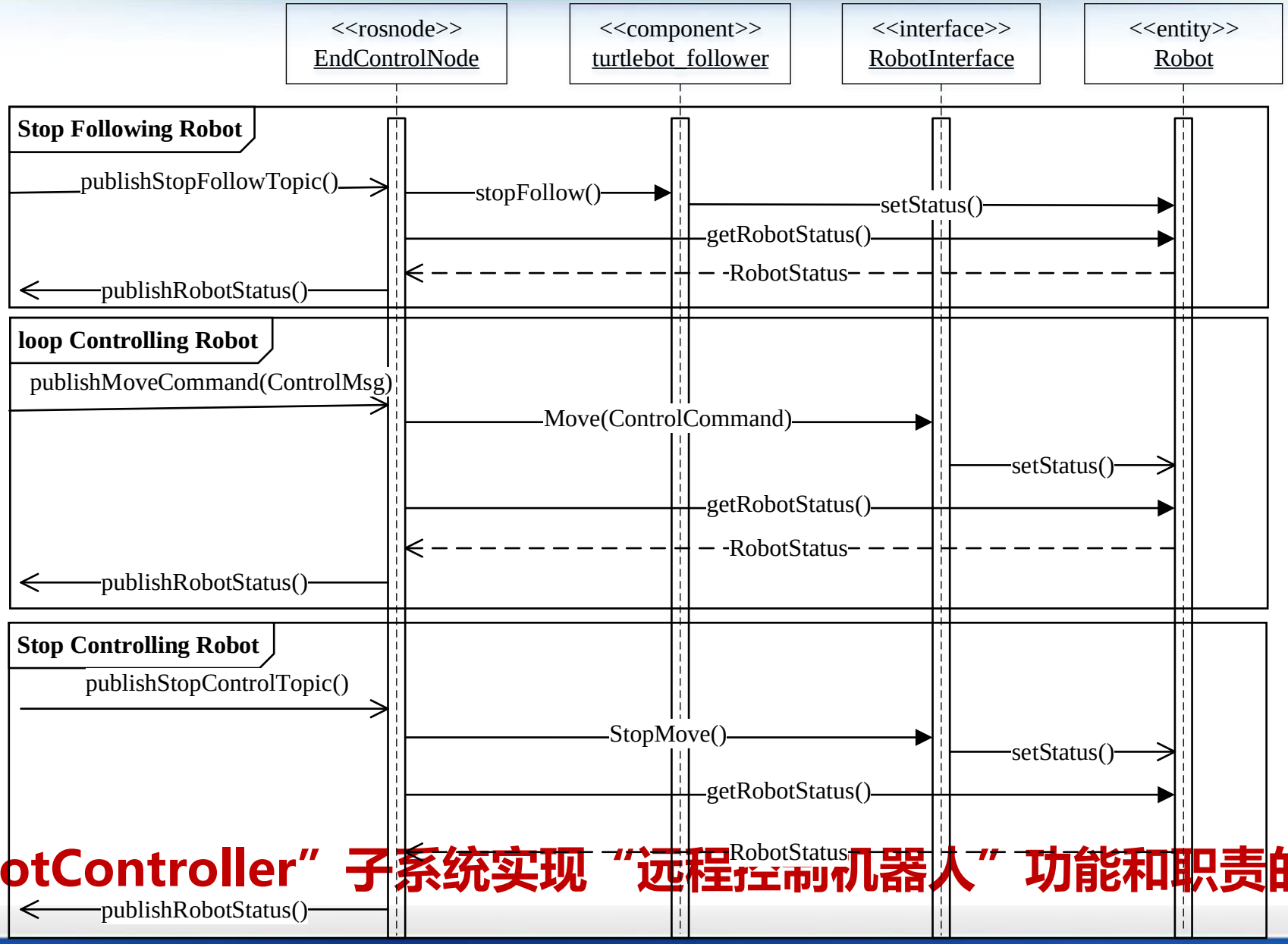


示例：精化“RobotController”子系统的设计元素



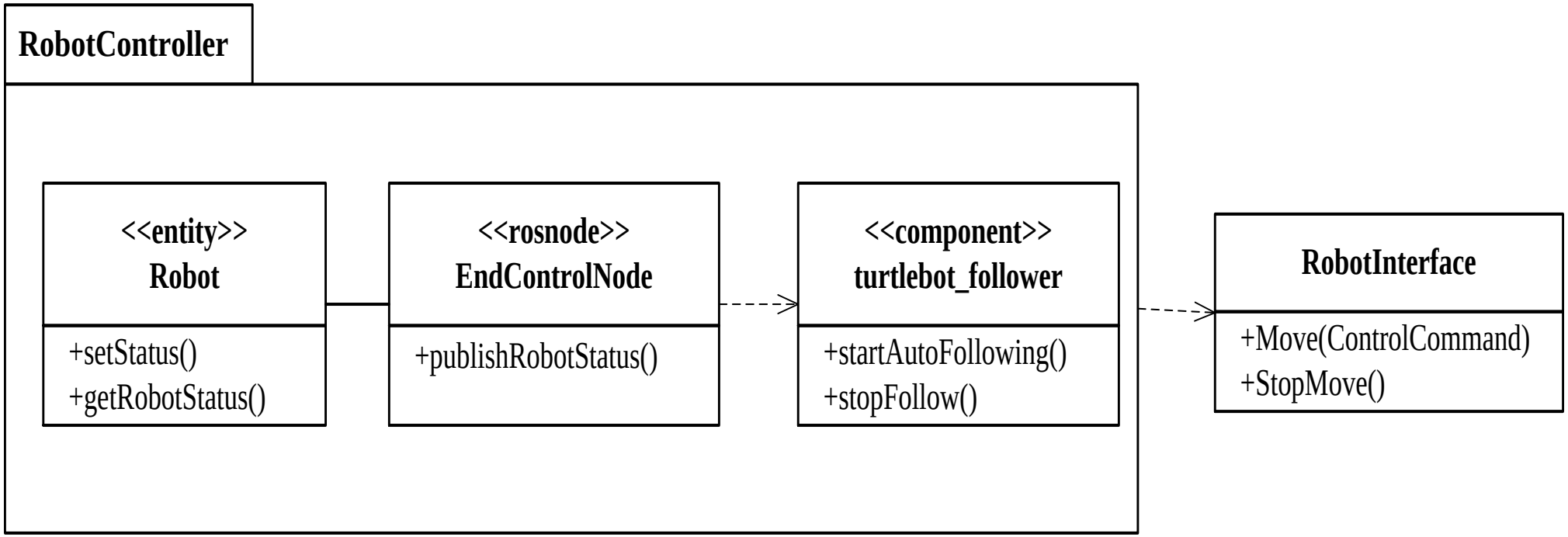
“RobotController”子系统实现“自主跟随老人”功能和职责的顺序图

示例：精化“RobotController”子系统的设计元素



“RobotController”子系统实现“远程控制机器人”功能和职责的顺序图

示例:“RobotController”子系统的设计类图



“RobotController”子系统的类图

内容

1. 软件详细设计概述
   - ✓任务
   - ✓过程
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 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 用户界面设计
- 3.3 用例设计
- 3.4 子系统/构件设计
- 3.5 类设计
- 3.6 数据设计
- 3.7 部署设计

4. 实施指南

3.3 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

评审设计文档(2)

- **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

小结

- **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

综合实践一

- **任务：开源软件的详细设计**
- **方法**
  - ✓针对开源软件新增加的软件需求，考虑软件的体系结构设计和用户界面设计，对开源软件进行详细设计，以实现开源软件新功能
- **要求**
  - ✓基于开源软件新构思的软件需求，结合体系结构设计和用户界面设计的成果，要详细到足以支持编码
- **结果：用类图、顺序图、活动图、状态图等描述的详细设计模型**

综合实践二

- **任务：软件详细设计**
- **方法**
  - ✓基于软件系统的用例模型、用例交互模型和分析类图，开展用例设计、类设计、数据设计、子系统/软构件设计，产生软件详细设计模型
- **要求**
  - ✓基于软件需求分析、体系结构设计、用户界面设计的具体成果，所产生的详细设计成果要详实到足以支持编码
- **结果：用类图、顺序图、活动图、状态图等描述的详细设计模型**

问题和讨论