《软件工程》

2023年春

第六讲 软件设计

内容

一．软件设计基础

二．软件体系结构设计

三．用户界面设计

四．详细设计

四、软件详细设计

实现类对象间的消息传递

- **OOP提供四种手段支持对象间实现消息传递**
  - ✓**引用全局对象**：obj1直接引用作为全局对象的obj2，依赖关系
  - ✓**通过参数传递**：obj2作为obj1的某项操作中的实在参数， 依赖关系
  - ✓**引用局部对象**：在obj1的某项操作的函数体中创建或获取obj2，依赖关系
  - ✓**通过类的成员变量**：obj2作为obj1所属类的属性的取值，聚合与组合关系

2.2.4 构造类对象的状态图

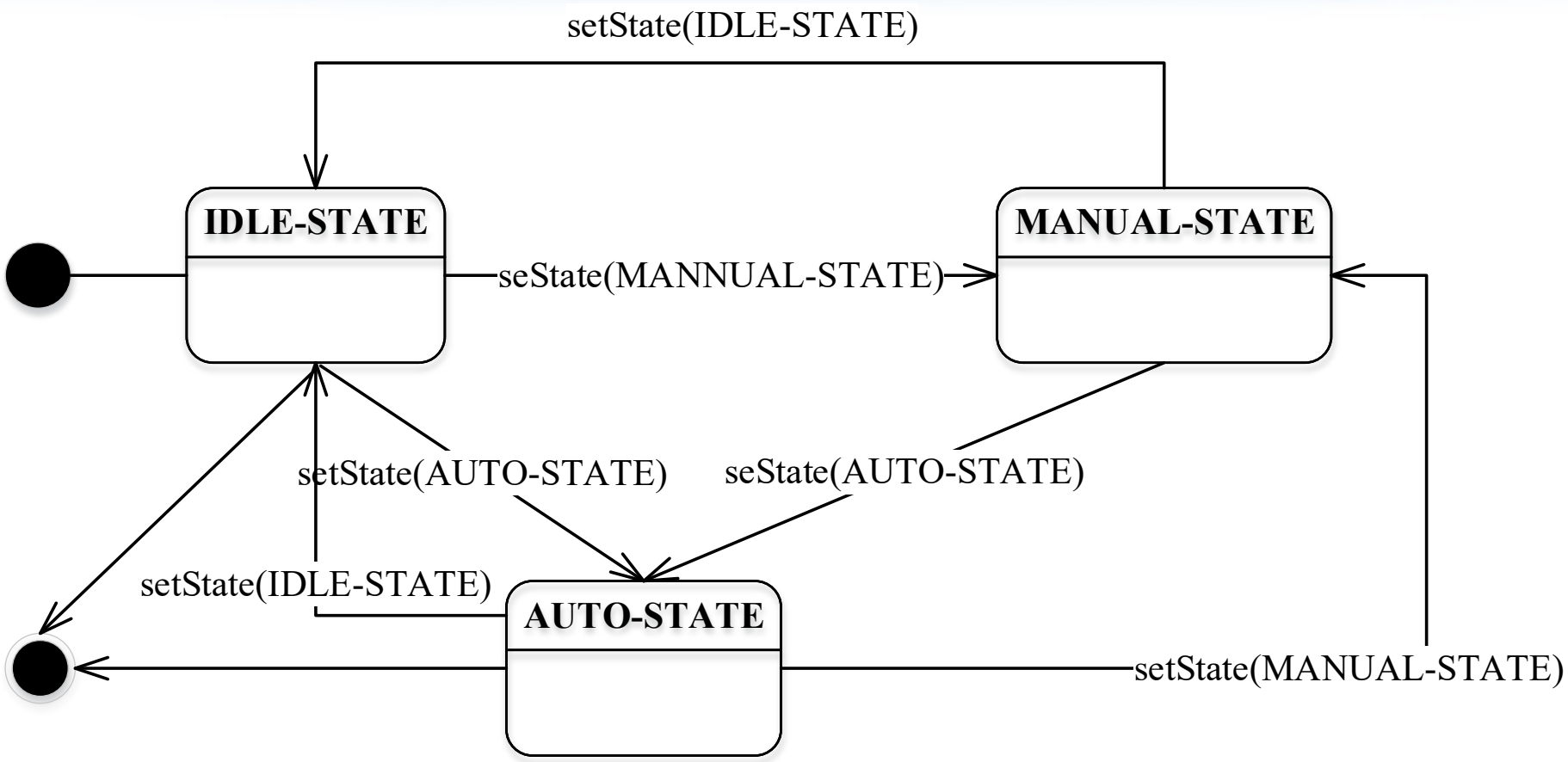
❑状态图

- ✓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

数据设计

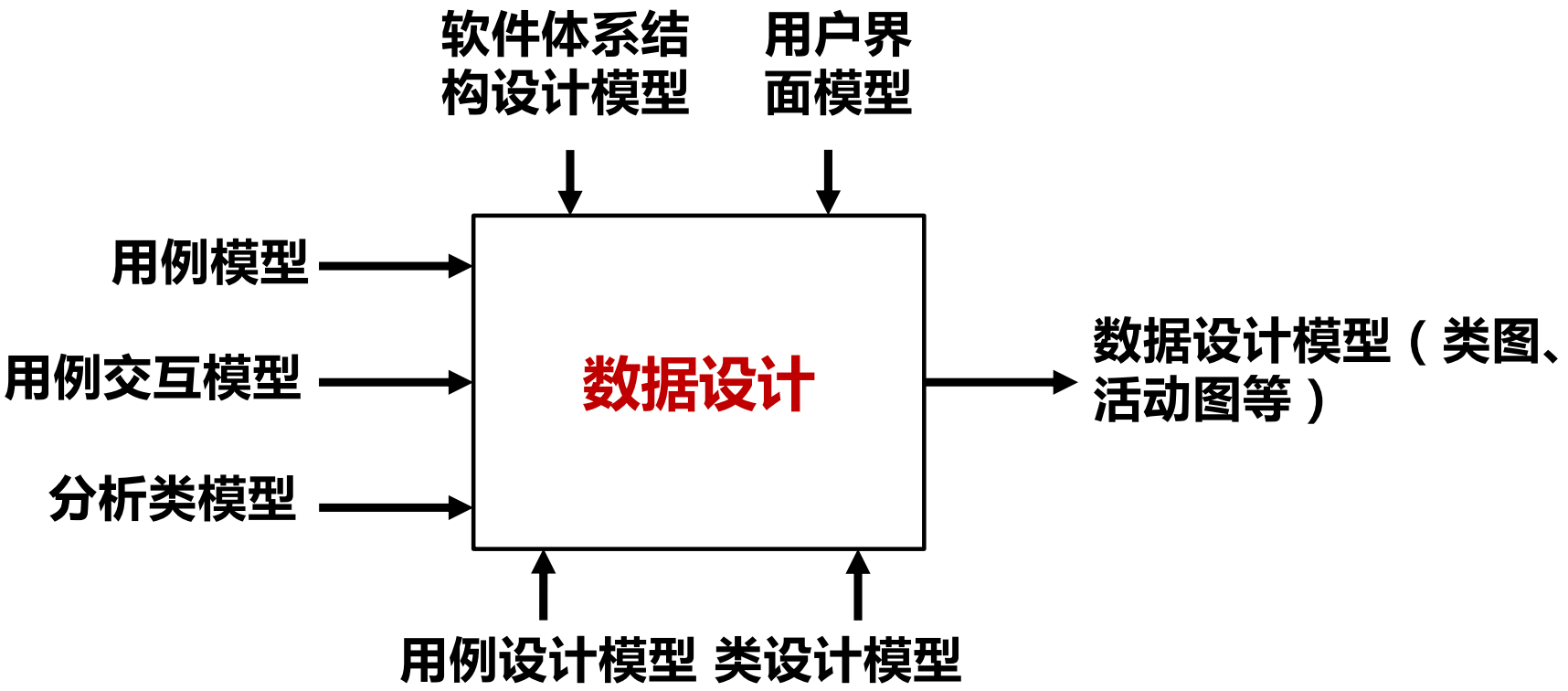
❑任务

- ✓设计需要持久保存的数据以及这些数据之间的关系
- ✓数据组织方式（例如关系数据库中的表、关键字、外键等）之间进行映射
- ✓为提高数据存储、操作性能而设计持久存储机制优化设施

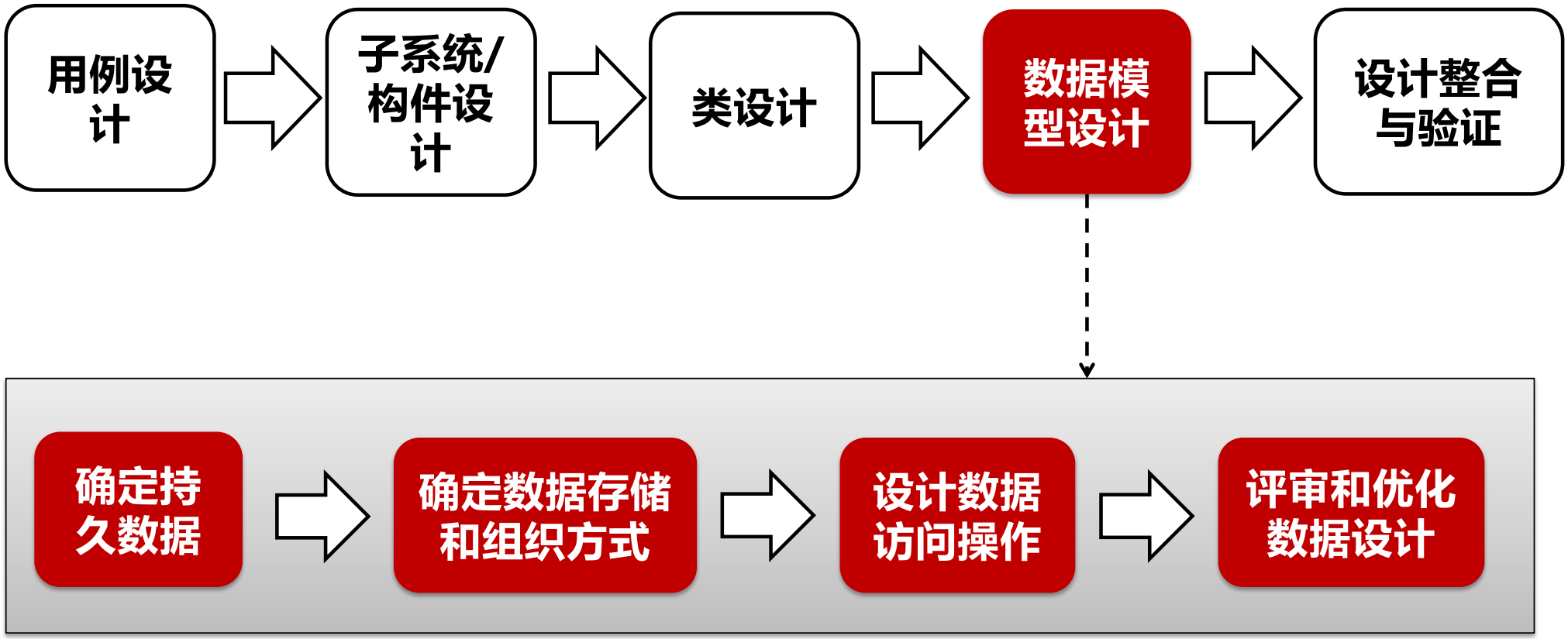
❑设计与建模

- ✓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

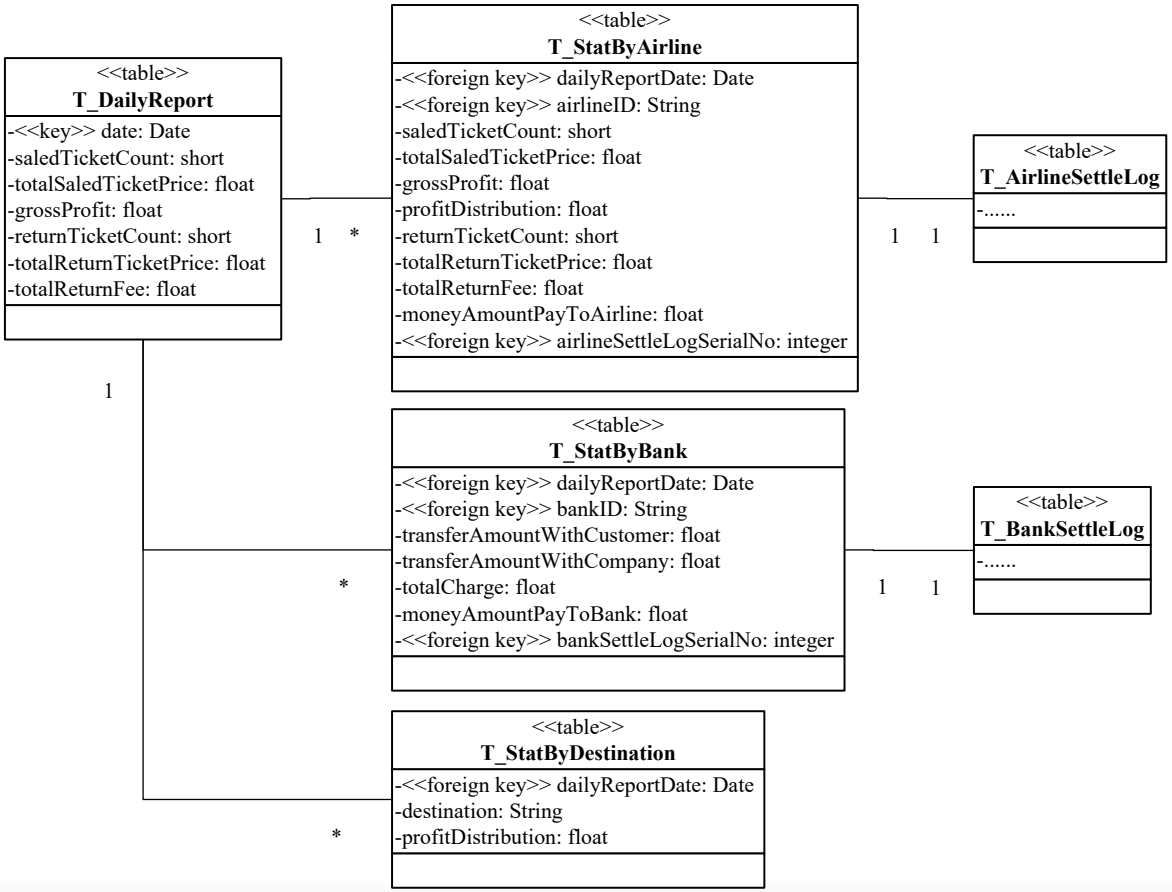
数据设计的任务


软件体系结构设计模型
用户界面模型
用例模型
用例交互模型
分析类模型
数据设计
数据设计模型（类图、活动图等）
用例设计模型
类设计模型

数据设计过程


用例设计
子系统/构件设计
类设计
数据模型设计
设计整合与验证
确定持久数据
确定数据存储和组织方式
设计数据访问操作
评审和优化数据设计

示例：数据模型设计


<<table>>
T_DailyReport
-<<key>> date: Date
-saledTicketCount: short
-totalSaledTicketPrice: float
-grossProfit: float
-returnTicketCount: short
-totalReturnTicketPrice: float
-totalReturnFee: float
<<table>>
T_StatByAirline
-<<foreign key>> dailyReportDate: Date
-<<foreign key>> airlineID: String
-saledTicketCount: short
-totalSaledTicketPrice: float
-grossProfit: float
-profitDistribution: float
-returnTicketCount: short
-totalReturnTicketPrice: float
-totalReturnFee: float
-moneyAmountPayToAirline: float
-<<foreign key>> airlineSettleLogSerialNo: integer
<<table>>
T_AirlineSettleLog
-.....
<<table>>
T_StatByBank
-<<foreign key>> dailyReportDate: Date
-<<foreign key>> bankID: String
-transferAmountWithCustomer: float
-transferAmountWithCompany: float
-totalCharge: float
-moneyAmountPayToBank: float
-<<foreign key>> bankSettleLogSerialNo: integer
<<table>>
T_BankSettleLog
-.....
<<table>>
T_StatByDestination
-<<foreign key>> dailyReportDate: Date
-destination: String
-profitDistribution: float
1
*
1
1
1
*
1
1
1
*

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

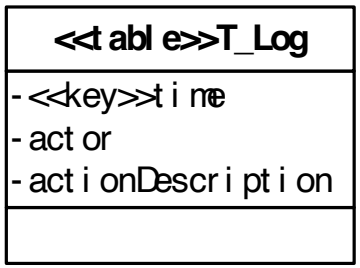
- **根据对需求的理解来确定哪些数据需要永久保存**
  - ✓如用户的账号和密码
  - ✓系统设置信息

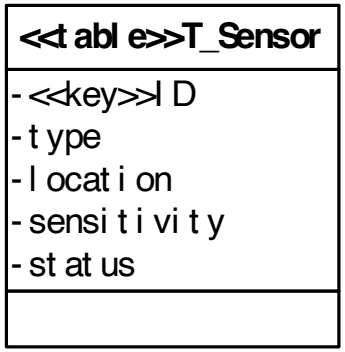
2.3.2 确定持久数据的存储和组织方式

- **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

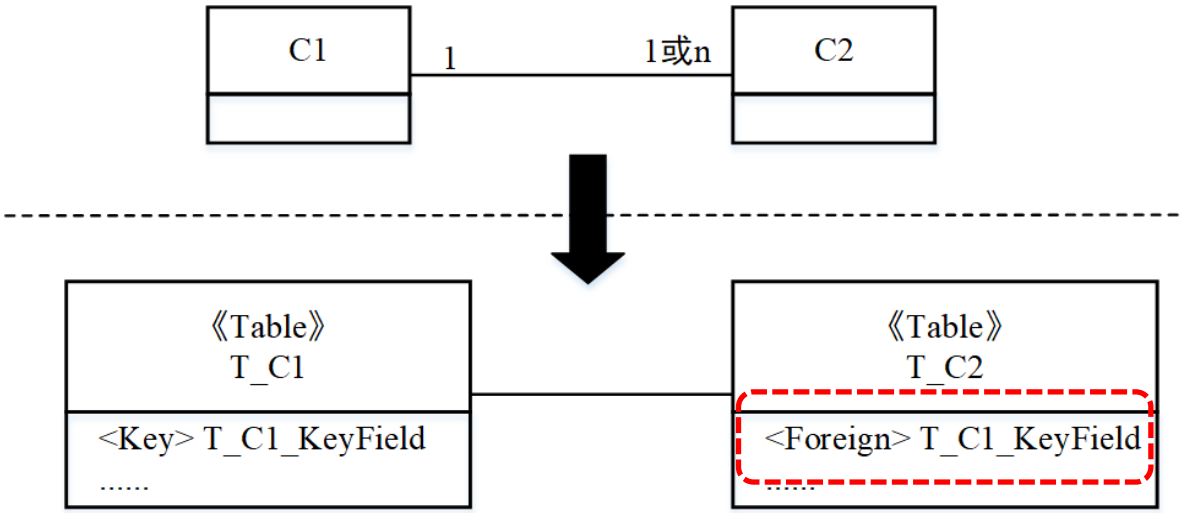
- 确定设计模型中需要持久保存的类的对象及其属性
- 面向对象设计模型与关系数据库模型的对应关系
  - ✓类对应于“**表格**”（table）
  - ✓对象对应于“**记录**”（record）
  - ✓属性对应于表格中的“**字段**”（field）





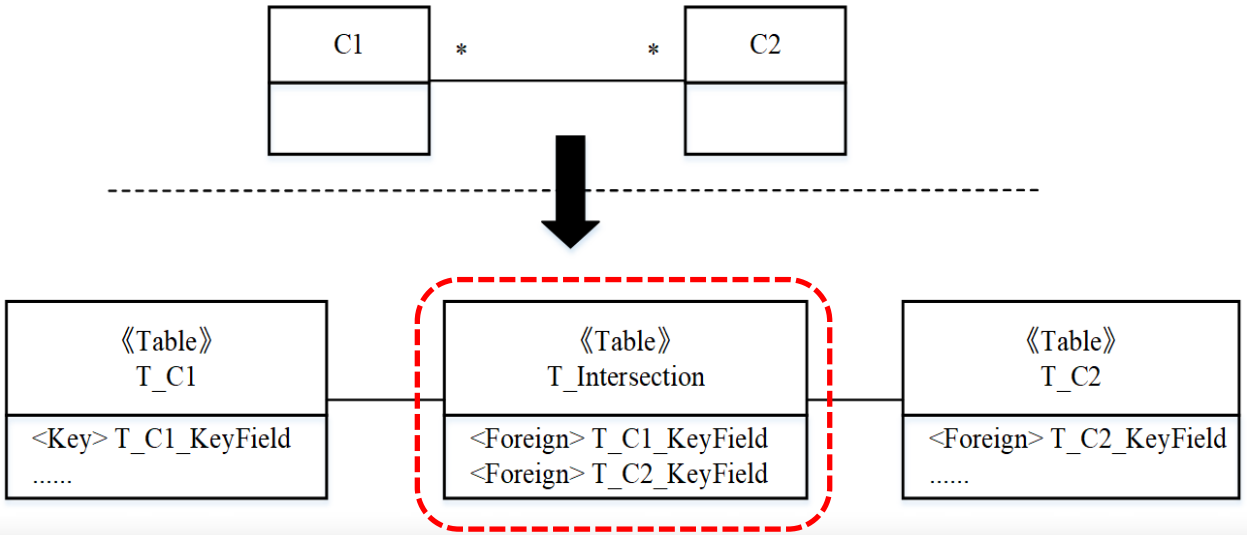
1:1、1:n关联关系的映射

- 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

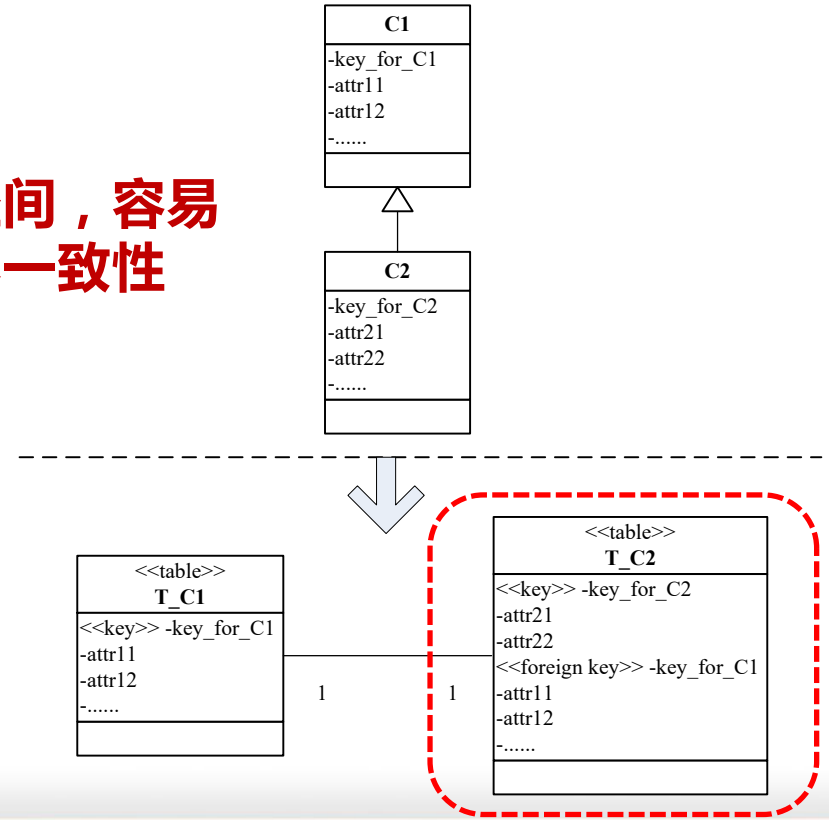
- 在T_C1、T_C2间引进新交叉表格T_Intersection，将T_C1、T_C2关键字段纳入T_Intersection中作为外键，在T_C1与T_Intersection之间、T_C2与T_Intersection之间建立一对多关系



继承关系的数据库表设计(1/2)

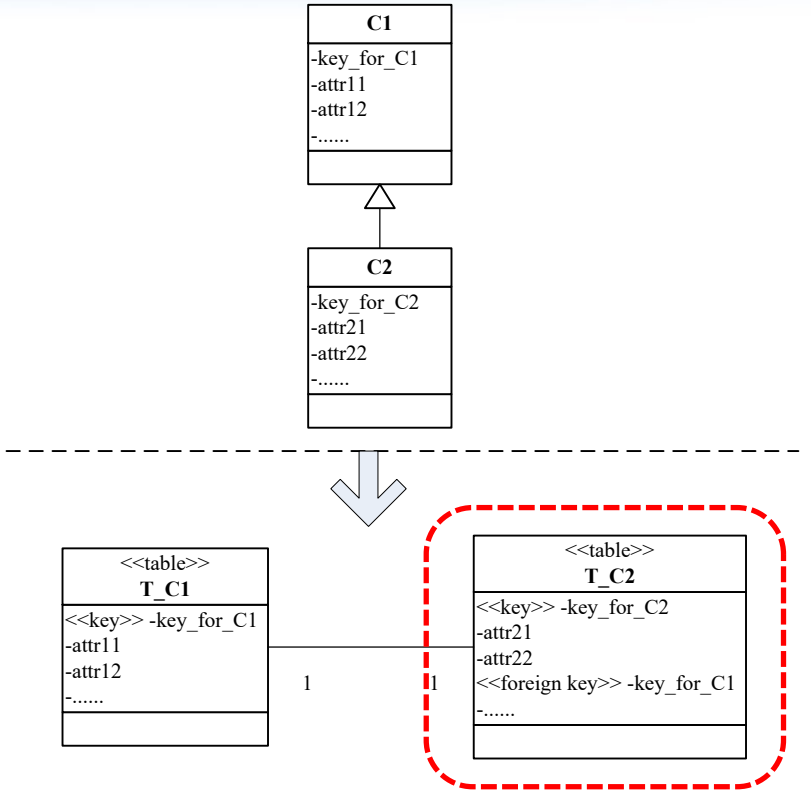
- 假设C1是C2的父类，将T_C1中的所有字段全部引入至T_C2

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

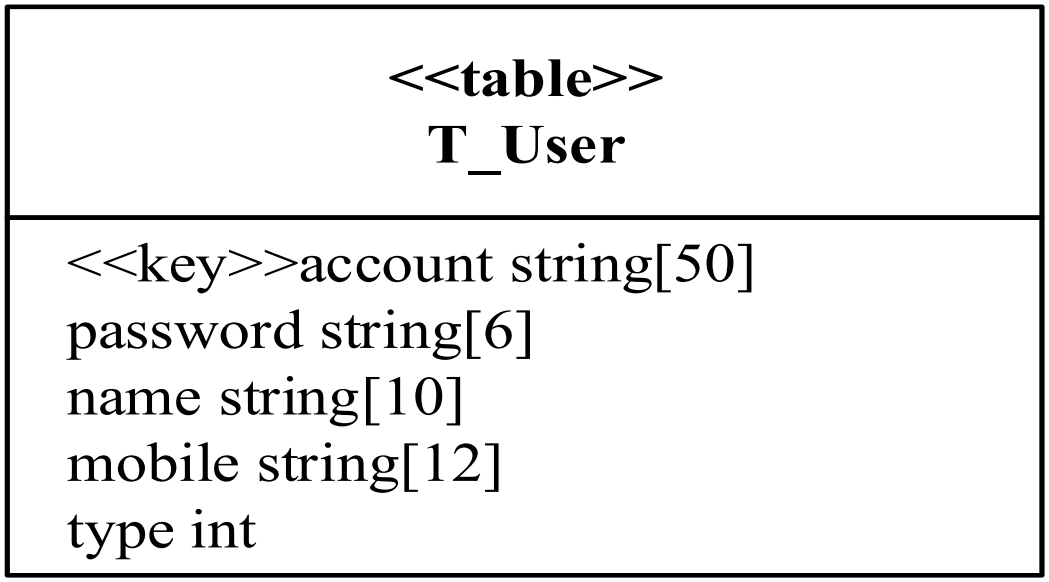


继承关系的数据库表设计(2/2)

- 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

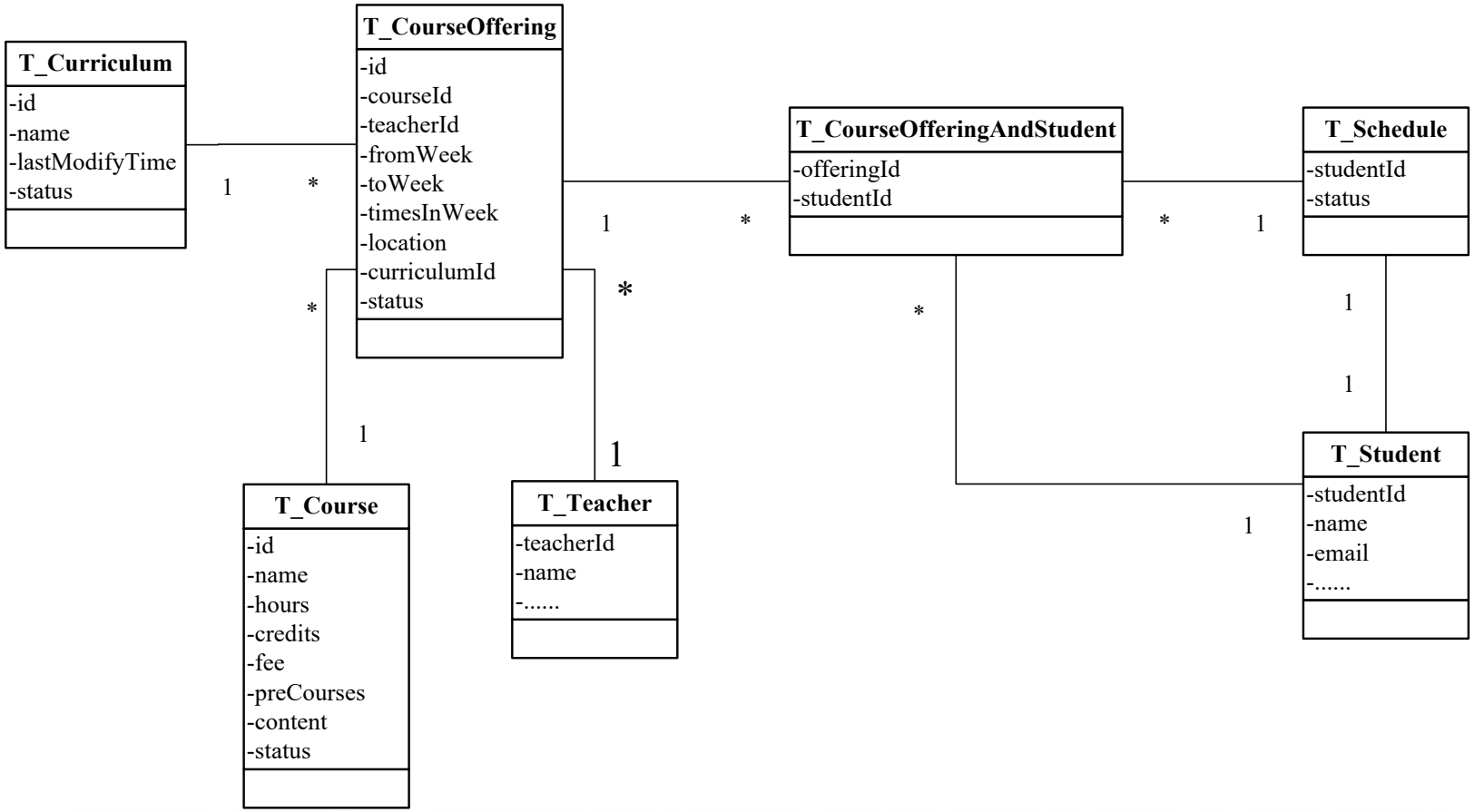


示例：设计持久数据



保存“User”类对象的数据库表“T_User”

示例：设计持久数据

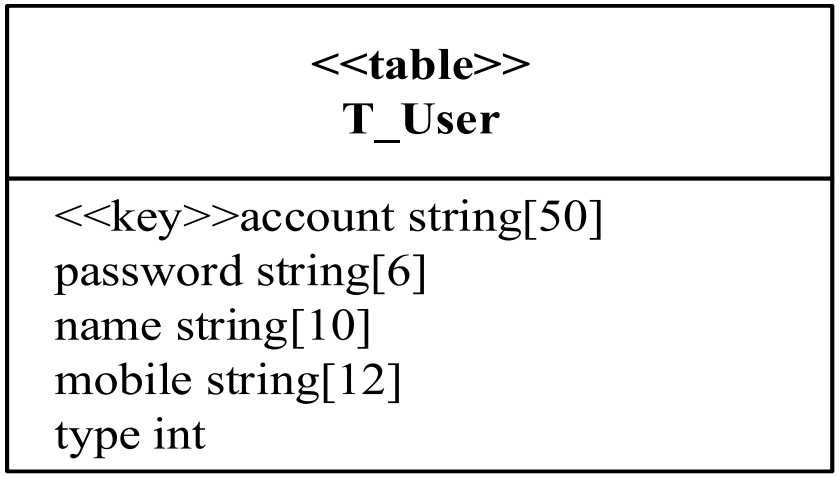

T_Curriculum
-id
-name
-lastModifyTime
-status
T_CourseOffering
-id
-courseId
-teacherId
-fromWeek
-toWeek
-timesInWeek
-location
-curriculumId
-status
T_CourseOfferingAndStudent
-offeringId
-studentId
T_Schedule
-studentId
-status
T_Course
-id
-name
-hours
-credits
-fee
-preCourses
-content
-status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

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()



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

设计输出

- 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

子系统设计

❑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

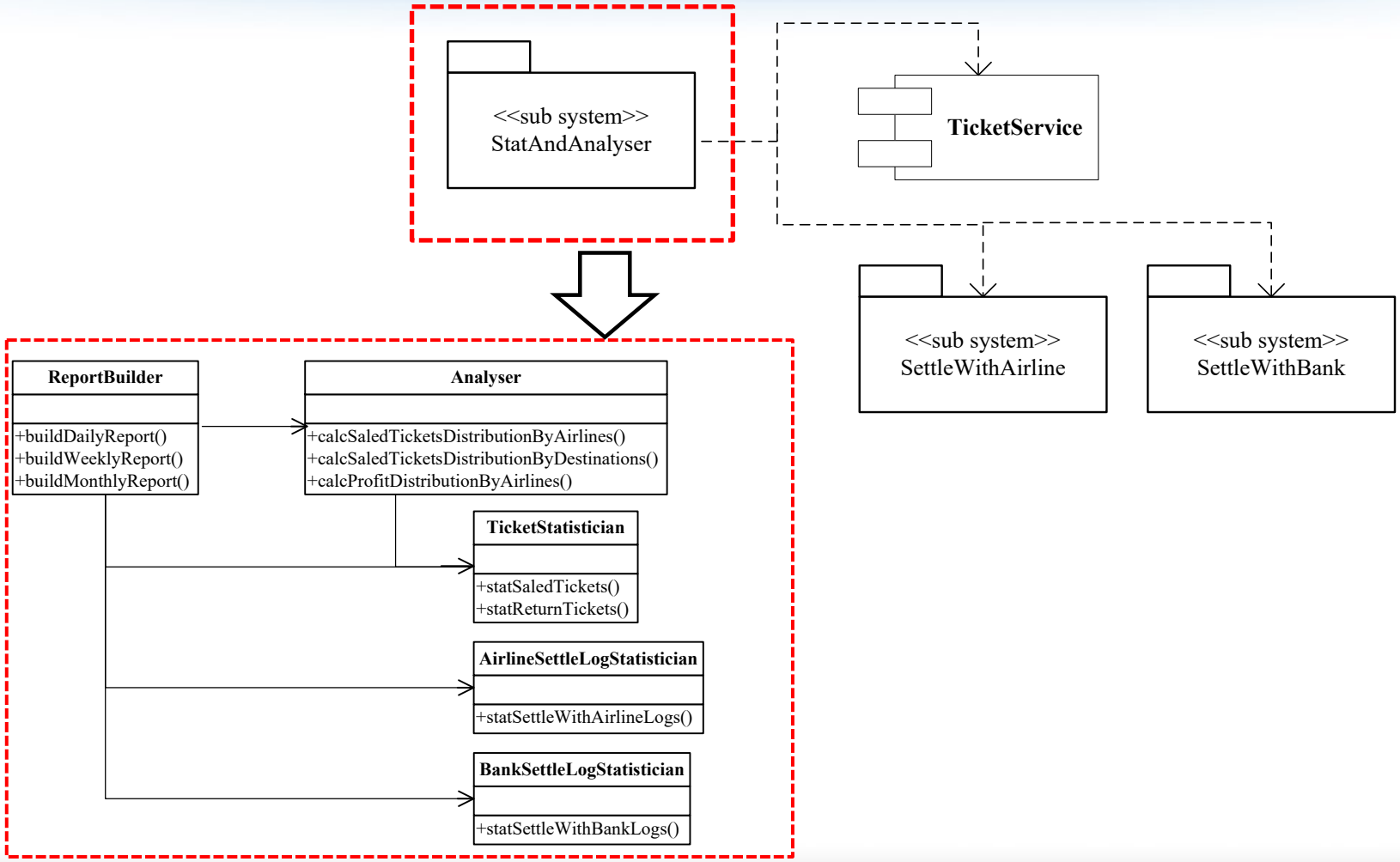
❑设计和建模

- ✓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

子系统设计的示例

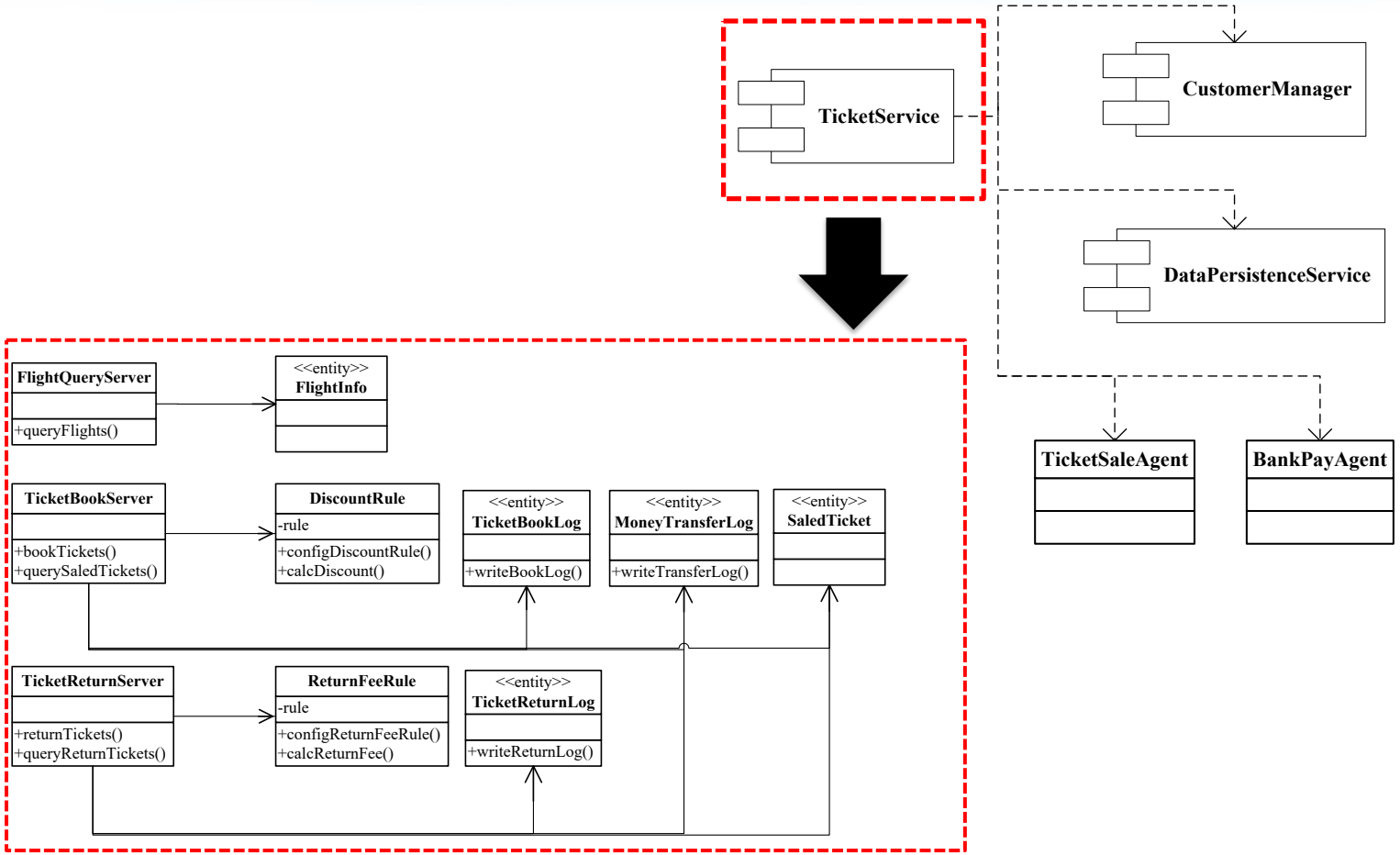

<<sub system>>
StatAndAnalyser
TicketService
<<sub system>>
SettleWithAirline
<<sub system>>
SettleWithBank
ReportBuilder
+buildDailyReport()
+buildWeeklyReport()
+buildMonthlyReport()
Analyser
+calcSaledTicketsDistributionByAirlines()
+calcSaledTicketsDistributionByDestinations()
+calcProfitDistributionByAirlines()
TicketStatistician
+statSaledTickets()
+statReturnTickets()
AirlineSettleLogStatistician
+statSettleWithAirlineLogs()
BankSettleLogStatistician
+statSettleWithBankLogs()

构件设计

- **任务**
  - ✓定义构件内部的**设计元素**及其**协作方法**
  - ✓内部设计元素可以是**子构件**，也可以是粒度更细的**类**
- **设计与建模**
  - ✓细化构件的内部细节，如子构件、类等
  - ✓对构件内部的结构进行建模
  - ✓对构件内部各个设计元素之间的协作进行建模
- **结果**
  - ✓构件图、类图、顺序图、活动图等

构件是可独立部署和运行的设计元素

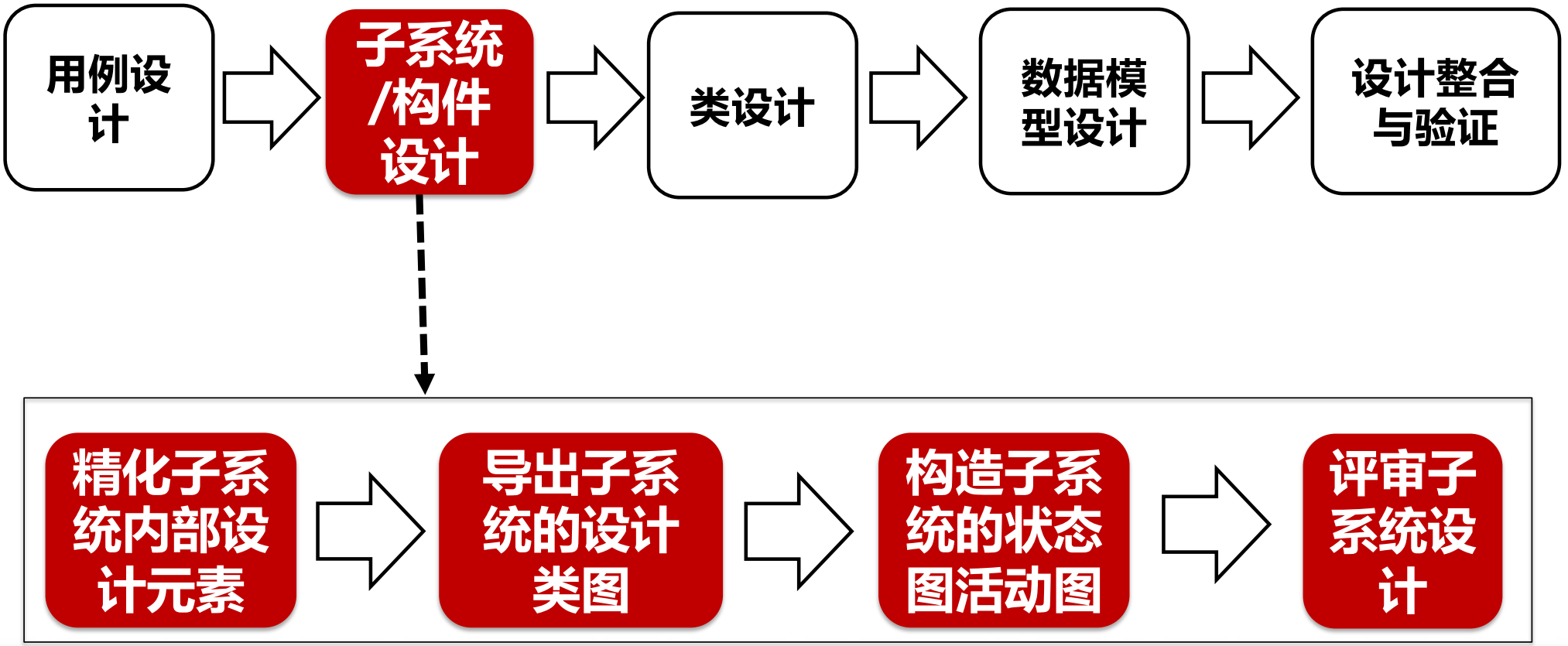
构件设计的示例


TicketService
CustomerManager
DataPersistenceService
TicketSaleAgent
BankPayAgent
FlightQueryServer
+queryFlights()
<<entity>>
FlightInfo
TicketBookServer
+bookTickets()
+querySaledTickets()
DiscountRule
-rule
+configDiscountRule()
+calcDiscount()
<<entity>>
TicketBookLog
+writeBookLog()
<<entity>>
MoneyTransferLog
+writeTransferLog()
<<entity>>
SaledTicket
TicketReturnServer
+returnTickets()
+queryReturnTickets()
ReturnFeeRule
-rule
+configReturnFeeRule()
+calcReturnFee()
<<entity>>
TicketReturnLog
+writeReturnLog()

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


用例设计
子系统/构件设计
类设计
数据模型设计
设计整合与验证
精化子系统内部设计元素
导出子系统的设计类图
构造子系统的状态图活动图
评审子系统设计

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

2.4.1 精化子系统内部设计元素

- **在子系统中设置哪些设计元素**
  - ✓构件、设计类或子系统
- **它们各自的职责是什么**
  - ✓提供什么功能和服务
- **它们间如何协作**
  - ✓实现子系统的职责、接口和功能

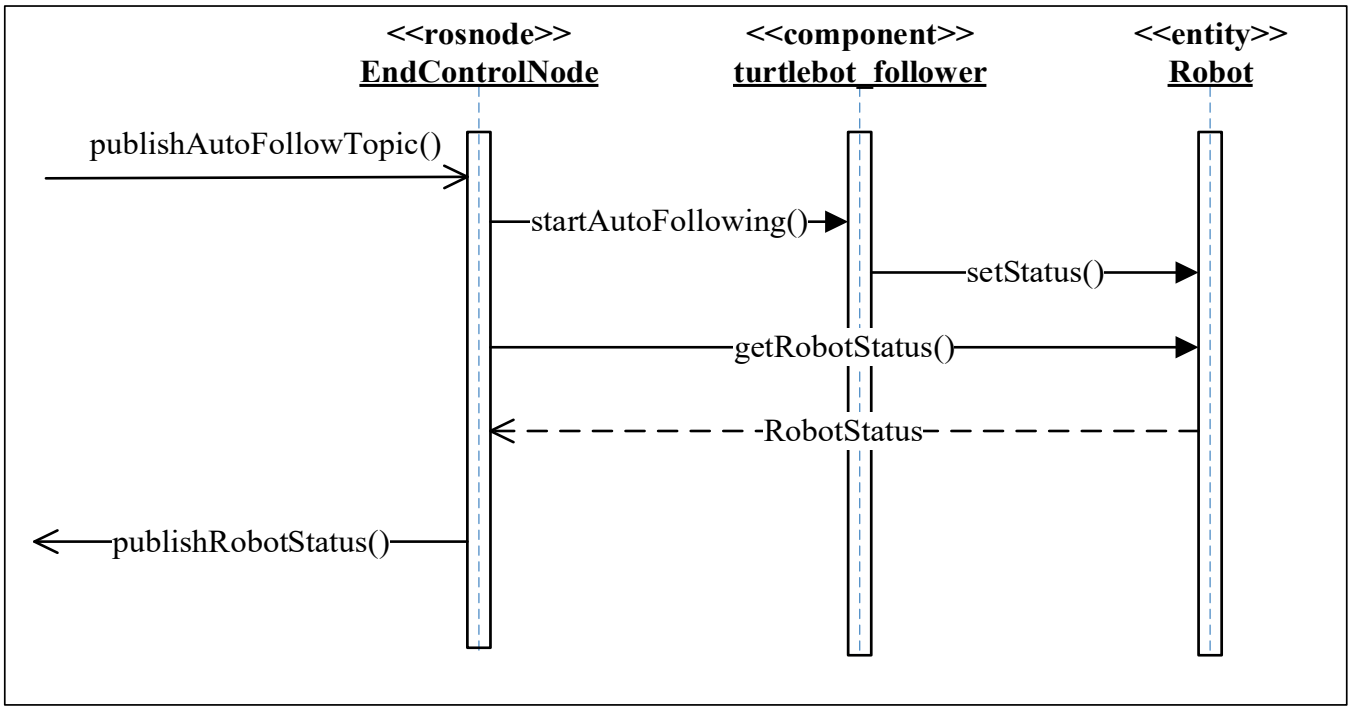
子系统内部设计的方法

- **理解和分解子系统的职责**
  - ✓通过一系列交互图来进一步分析子系统的职责
- **采用自顶向下和自底向上相结合的方式**
  - ✓将子系统职责交由一组相对独立的设计元素（如设计类、构件等）来完成
- **重用已有的软件资产（如开源软件、遗留系统）**
  - ✓如果它们能够承担部分职责，那么将相关的软件资产作为构成子系统的成分之一
- **绘制一系列UML交互图**
  - ✓刻画子系统中软件元素如何通过交互来实现子系统的职责
- **选择合适的设计模式有助于重用和优化子系统设计**
  - ✓重用一些有效的问题求解和职责实现方式

输出

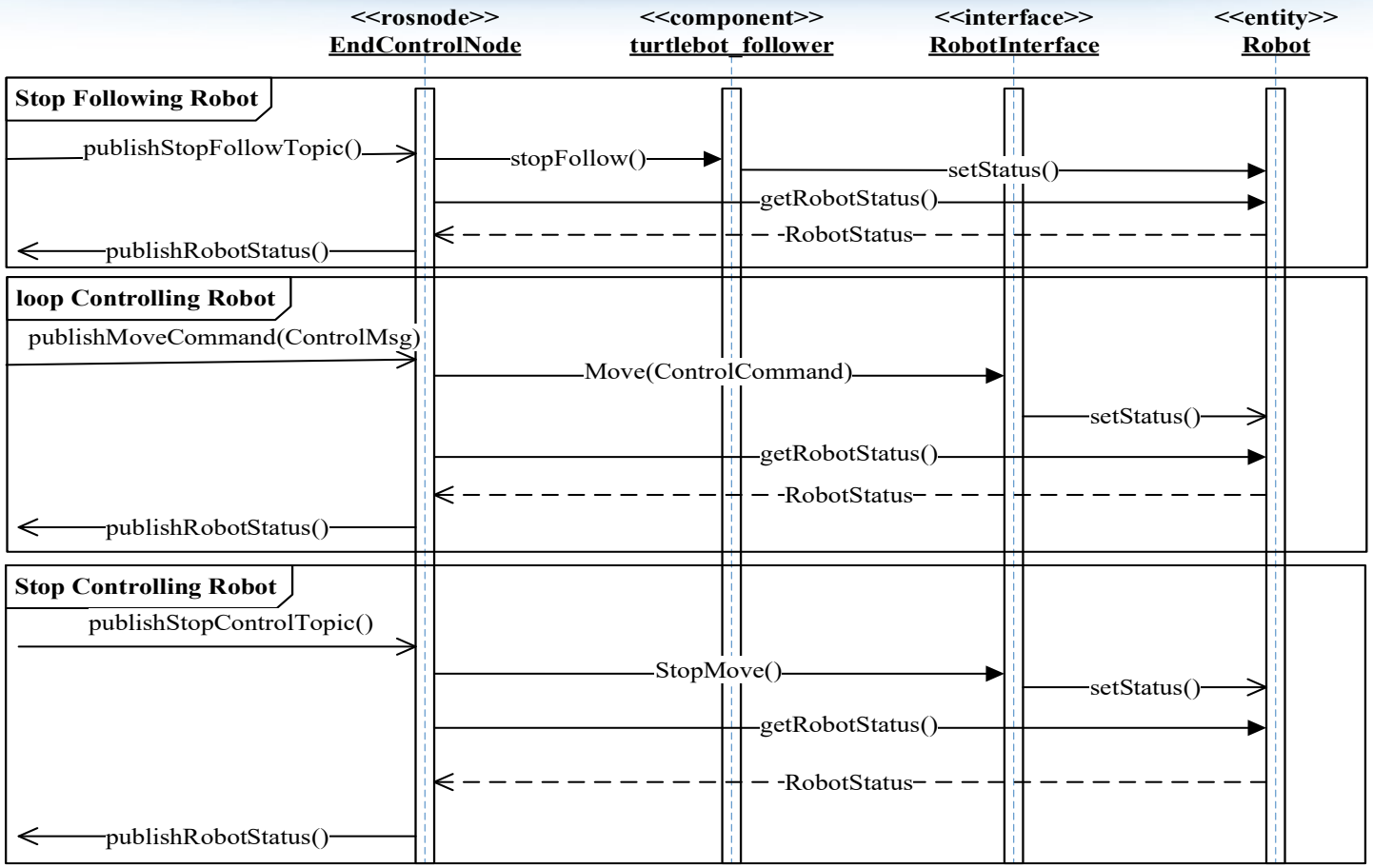
- 组描述子系统内部设计元素交互的UML顺序图

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

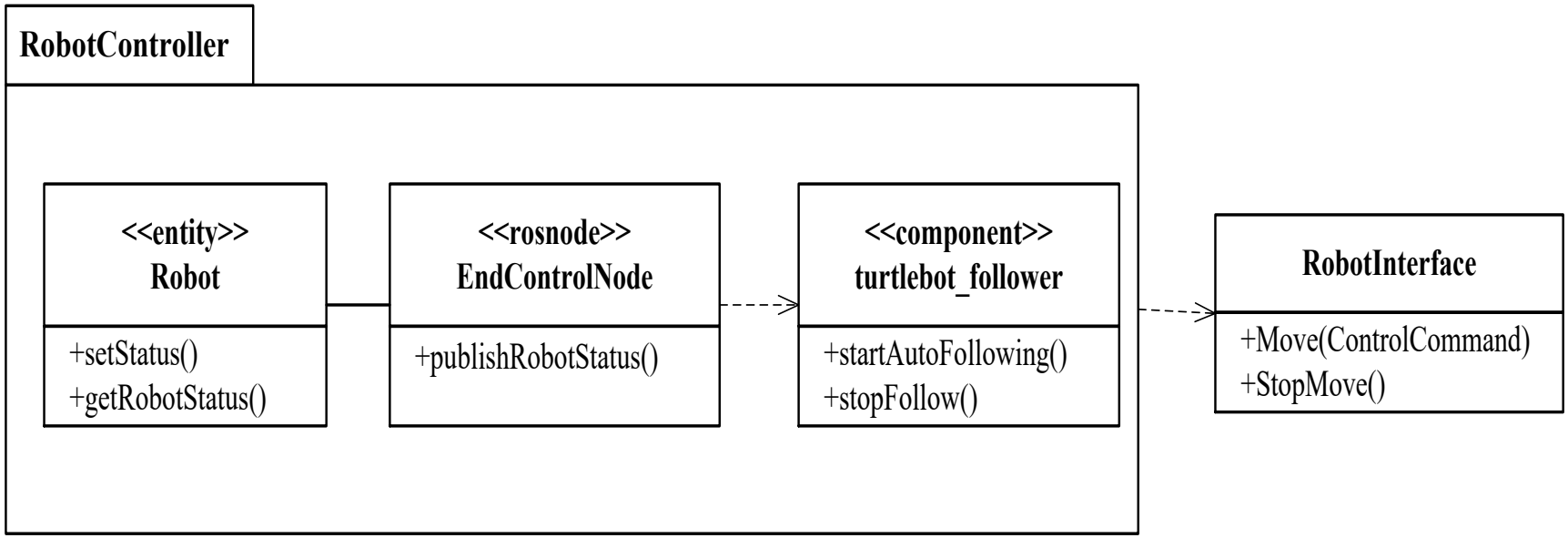
2.4.2 构造子系统的设计类图

- **基于子系统设计的UML交互图**
  - ✓详细描述了子系统功能和职责的实现方式
- **推导出子系统的设计类图**
  - ✓显式区分子系统内部的设计元素与位于子系统之外、为子系统提供服务的其他设计元素

推导子系统设计类图的方法

- **针对顺序图中对象所对应的类，将其抽象为设计类图中的类**
- **如果顺序图中对象a给对象b发消息m并附带参数p**
  - ✓目标对象对应的类具有相应的职责和方法，以处理消息m
  - ✓目标对象对应的类具有相应的属性以存储p
- **根据顺序图中对象间消息来确定设计类间的关系**
  - ✓如果一个对象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
  - ✓如果子系统外的设计元素通过子系统的接口与子系统进行交互，那么这些设计元素与子系统之间存在依赖关系
  - ✓如果多个设计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:“RobotController”子系统的设计类图



“RobotController”子系统的类图

2.4.3 构造子系统的状态图和活动图

- **如果子系统或其内部设计元素具有明显状态特征，那么绘制和分析其UML状态图**
  - ✓状态及其变化

- **构造子系统及其设计元素的活动图来理解和分析子系统是如何实现的**
  - ✓子系统内部的设计元素协同完成子系统的某些功能
  - ✓子系统与外部设计元素协同完成更大范围内的功能

子系统设计的输出制品

- 子系统设计方案
  - ✓交互图
  - ✓设计类图
  - ✓可能的状态图、活动图

评审子系统设计

- **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 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

3.5 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

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

评审步骤

- 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

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

问题和讨论