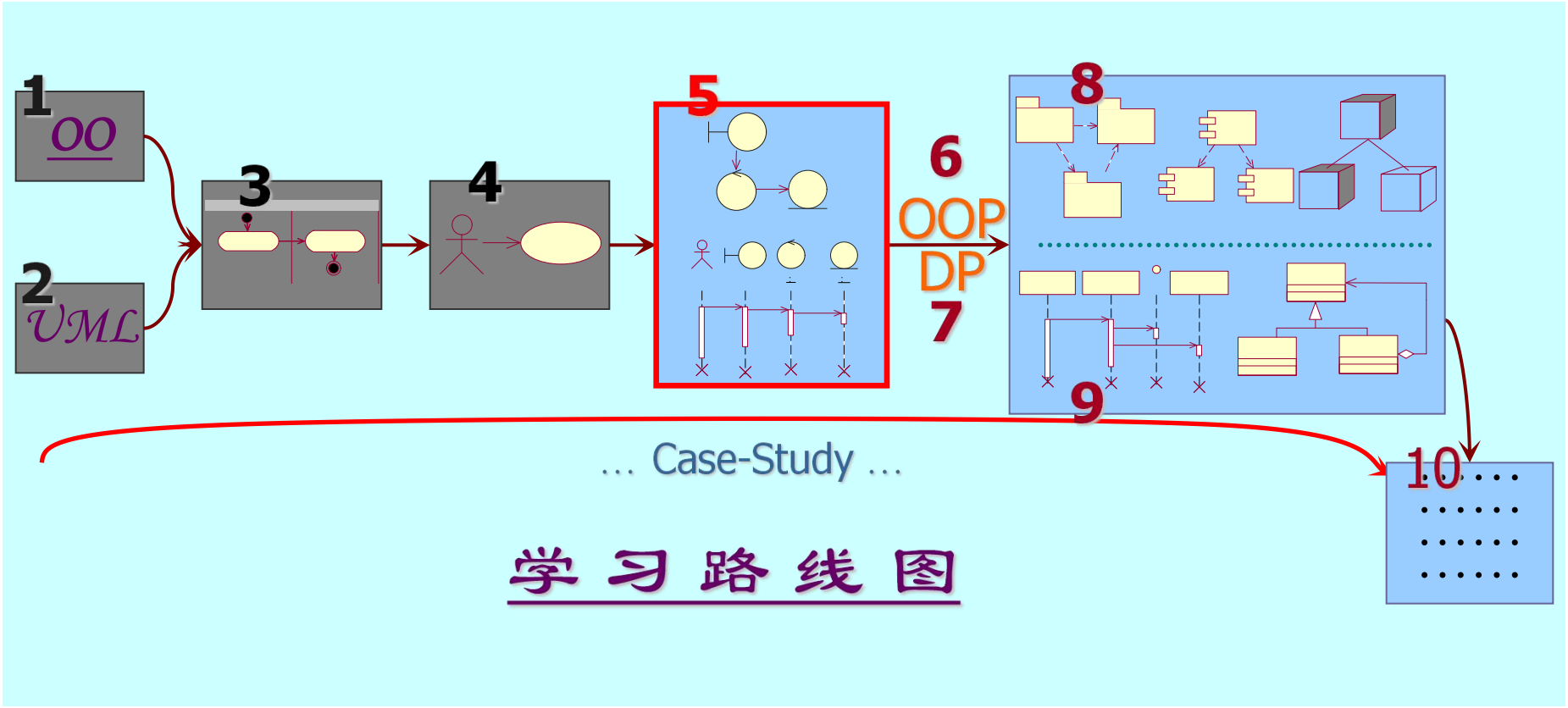
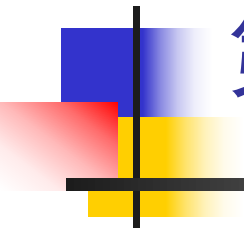
学习路线图

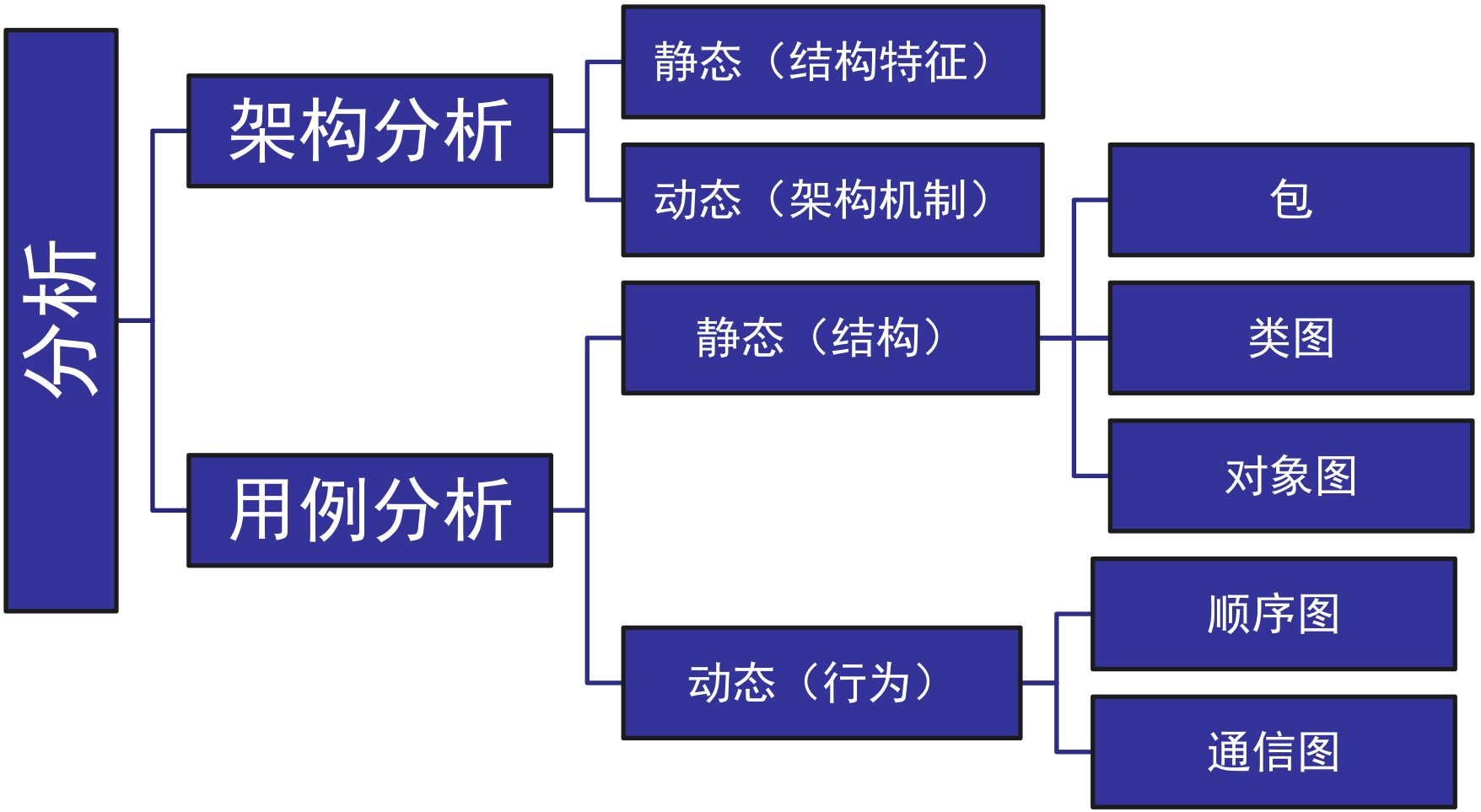




第05章 用例分析

Use Case Analysis

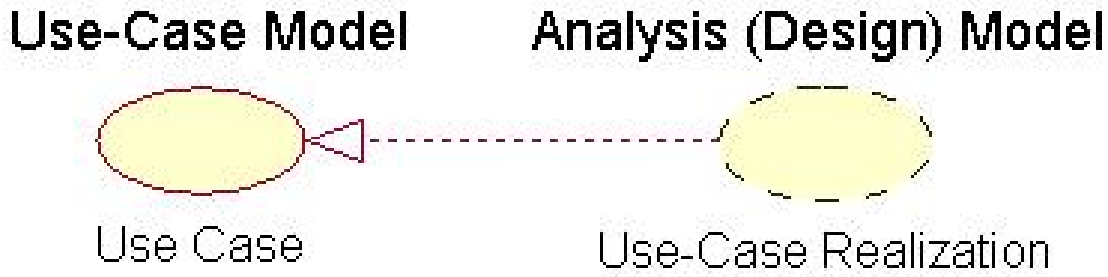
内容安排

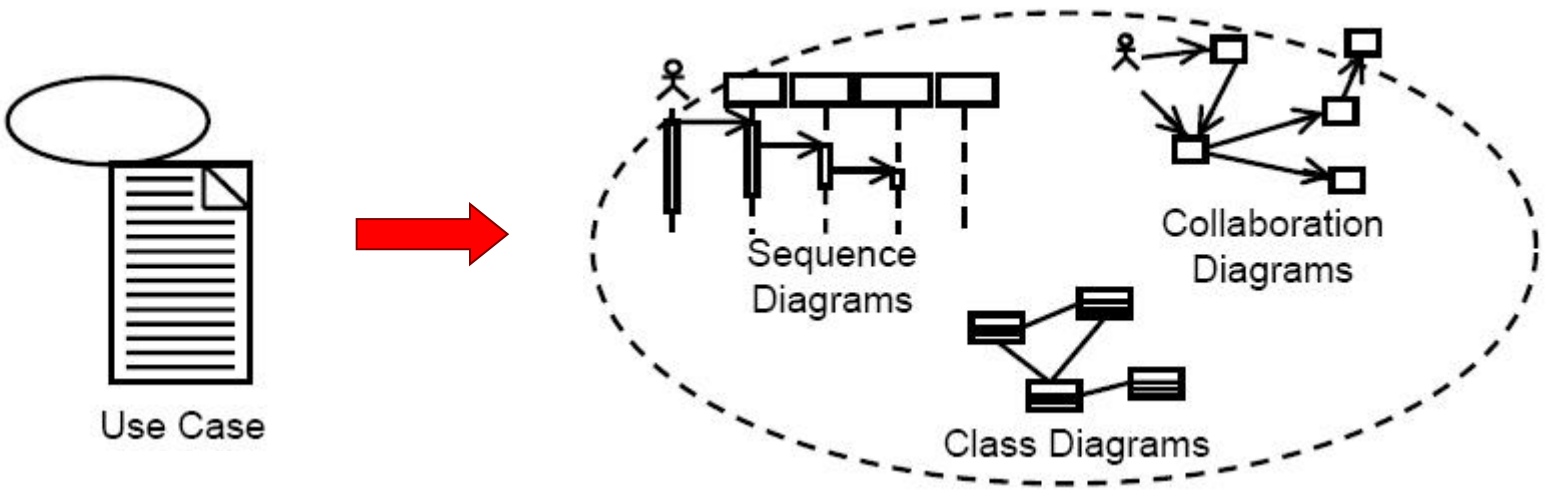


内容安排

- 构架分析
- 用例分析

创建用例实现


Use-Case Model
Analysis (Design) Model
Use Case
Use-Case Realization


Use Case
Sequence Diagrams
Collaboration Diagrams
Class Diagrams

什么是体系结构？

体系结构并非可运行的软件。确切地说，它是一种表达，使能够：

(1) 在满足既定的需求方面下，分析设计有效性；

(2) 在设计变更相对容易的阶段，考虑体系结构可能的选择方案；

(3) 降低与软件构造相关的风险

体系结构为什么重要？

- 软件体系结构的展示有助于对计算机系统开发感兴趣的所有利益相关者开展交流。
- 体系结构突出了早期的设计决策，这些决策对随后所有的软件工程工作有深远的影响。并且对于系统最终的成功，与运行实体同等重要。
- 体系结构“构建了一个相对小的、易于理解的模型，该模型描述了系统如何构成以及其构件如何一起工作”[BAS03]。

1 构架分析

- 构架分析的过程就是定义系统高层组织结构和核心构架机制的过程
  - 1.定义系统高层组织结构—备选构架
  - 2.确定系统通用构架机制—分析机制
  - 3.提取系统的关键抽象以揭示系统必须能够处理的核心概念—关键抽象
  - 4. 创建用例实现来启动用例分析—用例实现

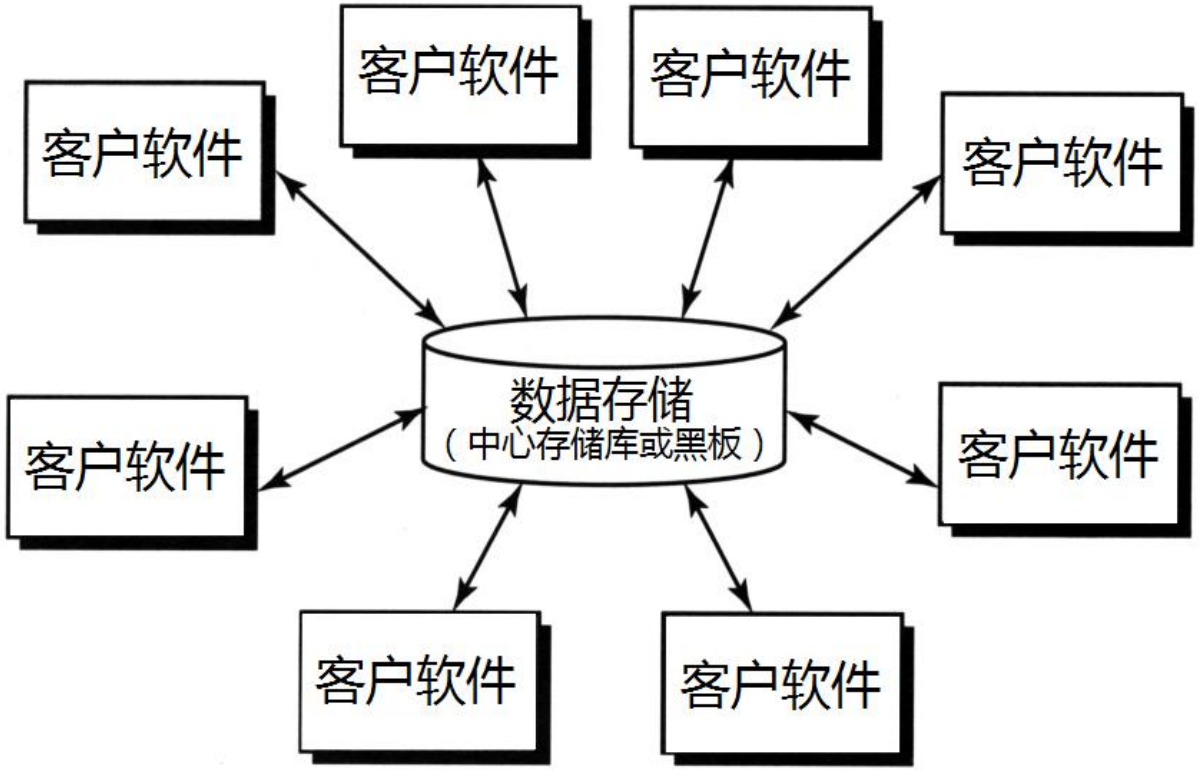
1. 1. 定义备选构架

- **定义备选构架**
  - 构架的初始草图
  - 初步定义系统的分层与组织
  - 初步定义一组在构架方面具有重要意义的元素，以作为分析的基础
  - 初步定义一组分析机制
  - 定义要在当前迭代中处理的用例实现

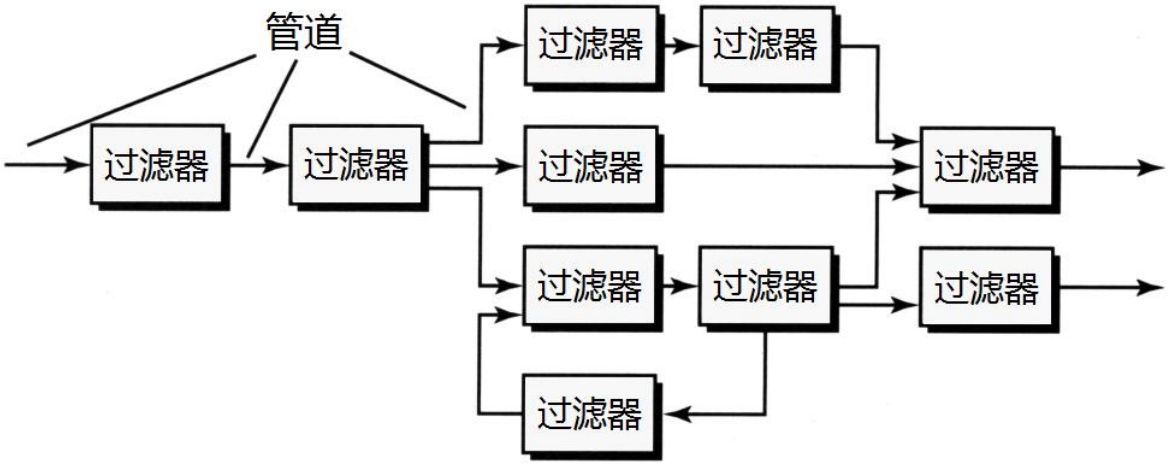
备选构架模式

- 每种风格描述一种系统类别，包括：（1）完成系统需要的某种功能的一组**构件**（例如，数据库、计算模块）；（2）能使构件间实现“通信、合作和协调”的一组**连接件**；（3）定义构件如何集成为**系统的约束**；（4）**语义模型**，能使设计者通过分析系统组成成分的已知属性来理解系统的整体性质
  - **以数据为中心的体系结构**
  - **数据流体系结构**
  - **调用和返回体系结构**
  - **面向对象体系结构**
  - **层次体系结构**

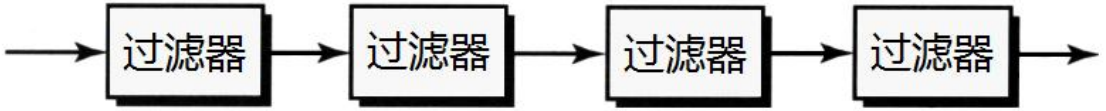
以数据为中心的体系结构



数据流体系结构

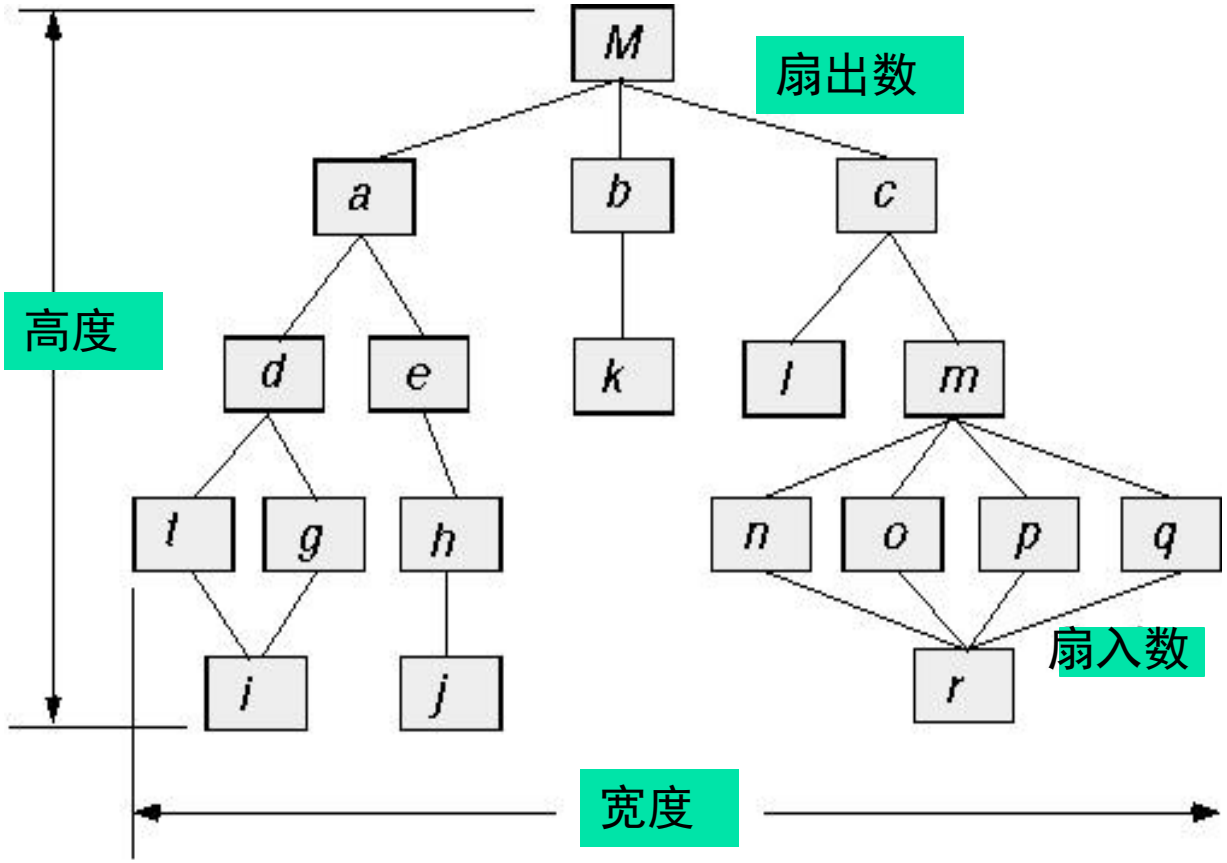


(a) 管道和过滤器

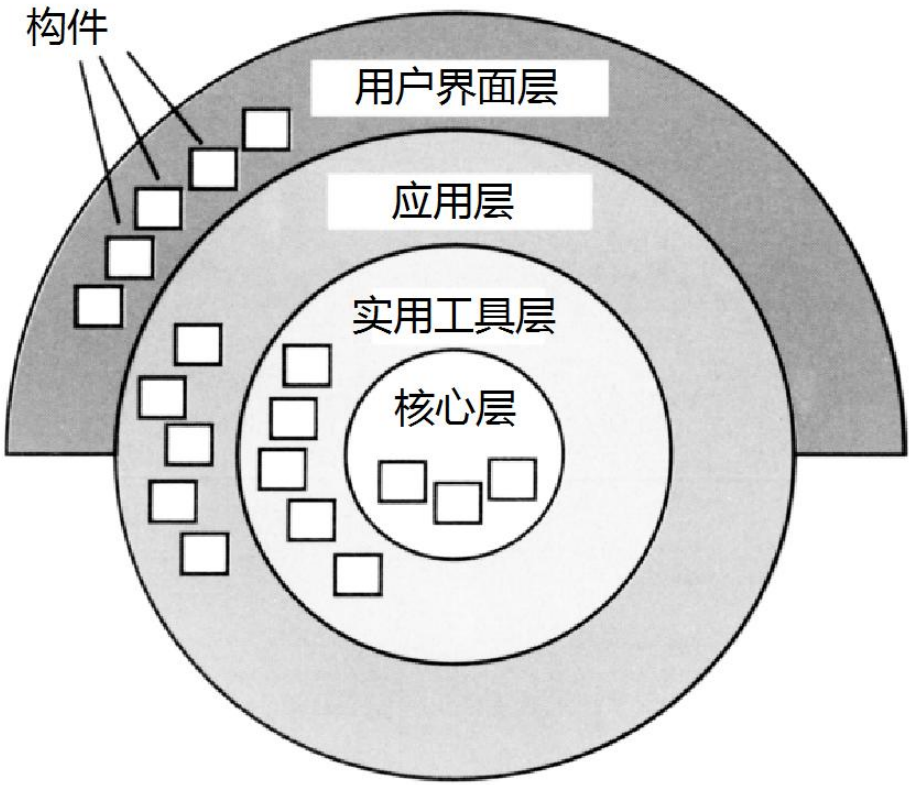


(b) 批序列

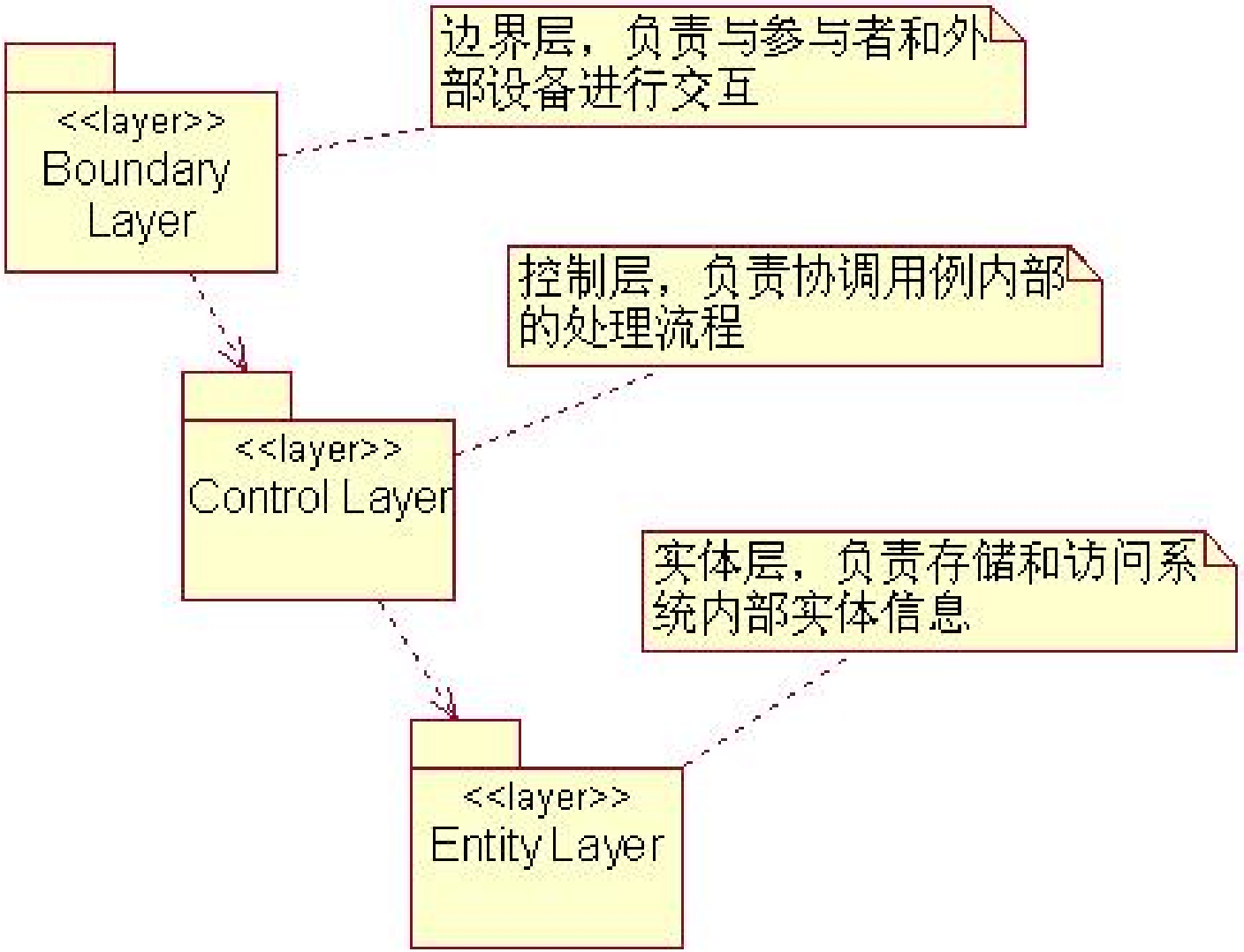
调用和返回体系结构



层次体系结构



分析阶段备选分层构架B-C-E



B-C-E三层构架解析

- 以构造型<<layer>>表示系统不同层次
- 以B-C-E三层划分系统三类处理逻辑
  - 边界层(Boundary)负责系统与参与者之间的交互
  - 控制层(Control)处理系统的控制逻辑
  - 实体层(Entity)管理系统使用的信息
- 层之间建立依赖关系

1. 2. 构架机制

- 构架机制是对通用问题的决策、方针和实践
  - 构架机制描述了针对一个经常发生的问题的一种通用解决方案
  - 作为系统构架的一部分，构架机制常常集中和定位在系统的非功能需求上
- 三类构架机制
  - 分析机制（概念）
  - 设计机制（具体）
  - 实现机制（实际）

构架机制示例

分析机制	设计机制	实现机制
持久性	关系型数据库	JDBC
	面向对象数据库	Object Store
分布	远程方法调用	JavaSE6 RMI
安全性	RBAC身份认证机制	Shiro

常见的分析机制

- 持久性(Persistency)
- 遗留接口(Legacy Interface)
- 分布(Distribution)
- 安全性(Security)
- 事务管理(Transaction Management)
- 进程控制和同步(Process Control and Synchronization)
- 通信(Communication)
- ……

1.3. 关键抽象

- 关键抽象是一个通常在需求上被揭示的概念，系统必须能够对其处理
  - 来源于业务，体现了业务的核心价值，即业务需要处理哪些信息
  - 这些信息所构成的实体即可作为初步的实体分析类
  - 具体来源有需求描述、词汇表、特别是业务对象模型
  - 通过一个或多个类图来展示关键抽象，并为其编写简要的说明

旅店预订系统中的关键抽象

关键抽象	含义	相关联的关键抽象
旅客	预订房间或入住的人，姓名，联系方式（手机）	
房间	旅店提供旅客住宿的地方，类型（单间，标间，豪华标间。。。，单价）	
预订	记录旅客在指定时间段入住指定类型房间、押金支付信息、预订状态，流水号。。。	旅客、房间、支付
支付	（支付信息，包括支付方式，支付时间，支付金额等）。。	

内容安排

- 构架分析
- **用例分析**

2. 用例分析

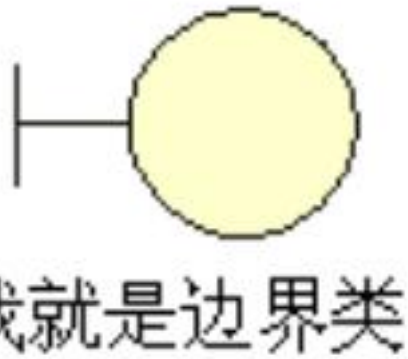
- 用例分析是分析最核心的工作
  - 获得实现用例行为所必须的分析类
  - 利用这些分析类来描述其实现逻辑
- 针对每一个用例实现：
  - 1. 完善用例文档
  - 2. 识别分析类
    - 边界类、控制类和实体类
  - 3. 分析交互
    - 交互图分析用例实现
  - 4. 完成参与类类图
    - 参与用例实现的类的类图
  - 5. 处理用例关系

2.1什么是分析类

- 分析类代表了“系统中必须具备职责和行为的事物”的早期概念模型
- 分析类处理主要的功能需求，模型化问题域对象
- 根据备选构架定义三类分析类
  - 边界类：系统及其参与者的边界
  - 控制类：系统的控制逻辑
  - 实体类：系统使用的信息

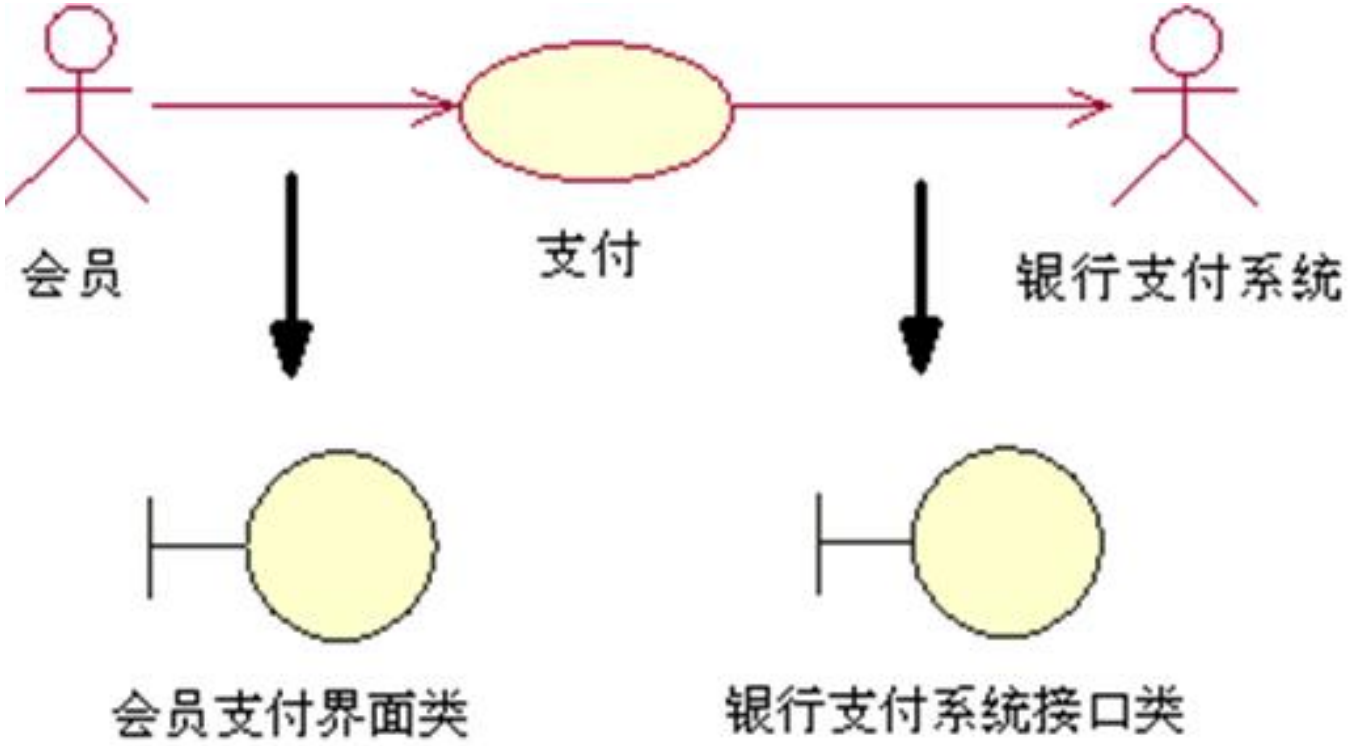
边界类

- 边界类表示系统与参与者之间的边界
  - 代表系统与环境的交互
  - 是接口和外部事物的中间体
  - 构造型<<boundary>>
- 两类边界类
  - 用户界面类
  - 系统和设备接口类



示例：识别边界类

- 每对参与者/用例定义一个边界类

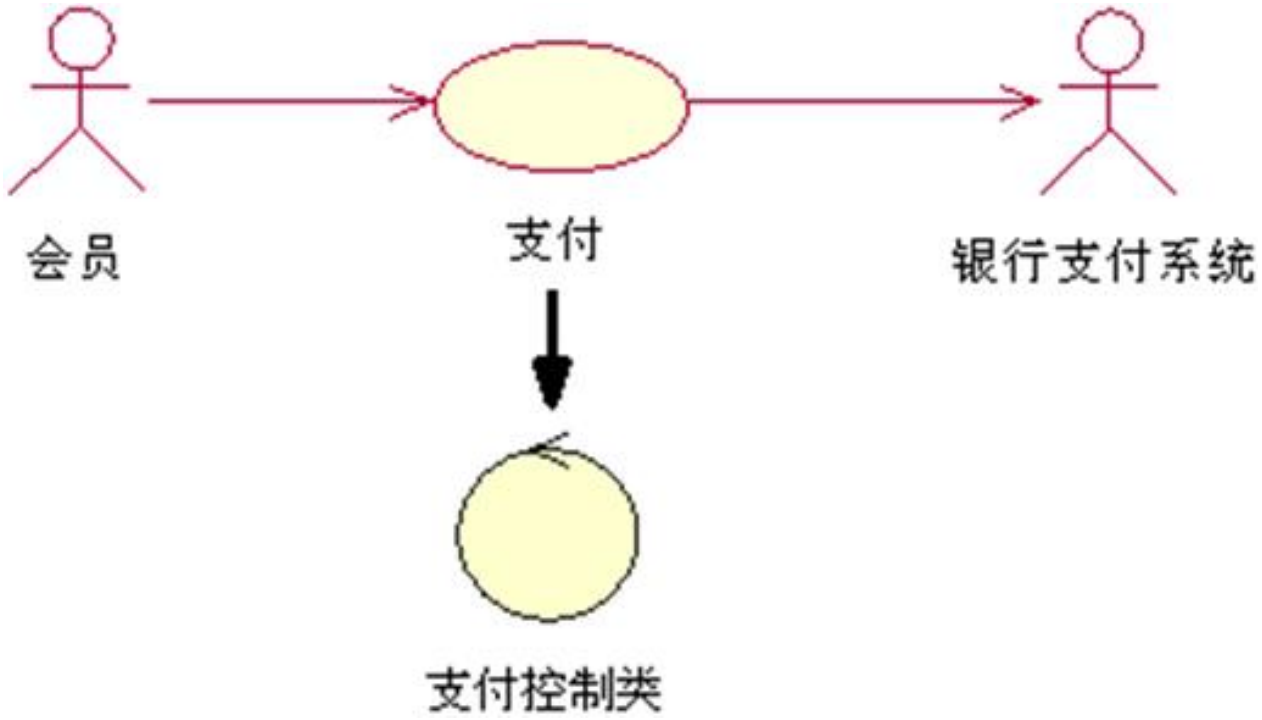


控制类

- 控制类表示系统的控制逻辑
  - 系统行为的协调器
  - 构造型<<control>>
- 识别控制类
  - 在系统开发早期，为一个用例定义一个控制类，负责该用例的控制逻辑
  - 针对复杂用例，可为备选路径分别定义不同控制类

示例：识别控制类

- 通常，每个用例定义一个控制类
  - 随着分析的继续，一个复杂用例的控制类可以发展为多个



实体类

- 实体代表了待开发系统的**核心概念**
- 实体类提供了另一个理解系统的观点
  - 显示了系统的**逻辑数据结构**
  - 传统的面向对象方法就是从这个角度进行分析和设计
- 使用构造型<<entity>>
- 可以从以下中找到实体类
  - 用例事件流(需求)、业务模型(业务建模)、词汇表(需求)

识别实体类

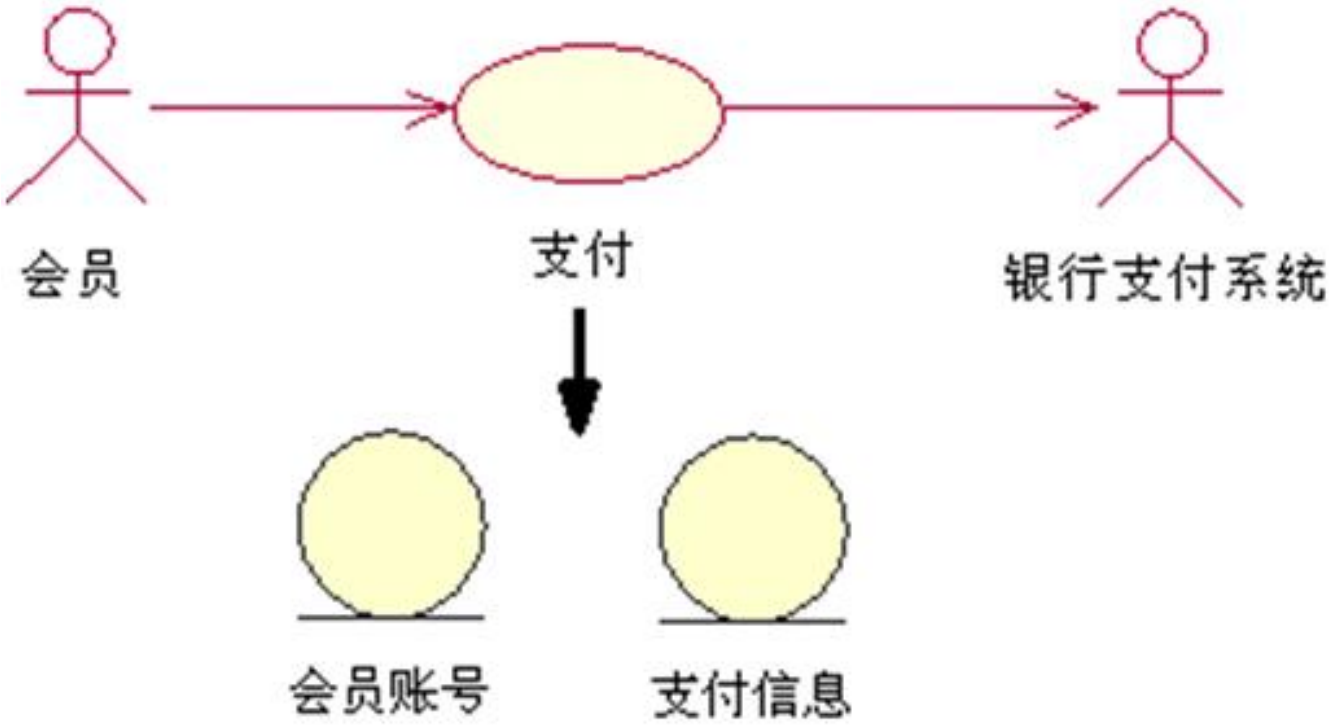
- 分析用例事件流中的名词、名词短语找出系统所需的实体对象，这些名词可能是：
  - 对象、对象的特征和状态
  - 参与者、描述信息、系统之外的
- 从这些名词、名词短语中进行筛选，抽取出系统对象，并抽象成类
  - 综合考虑在系统中的意义、作用和职责
- 对于所识别的类进行命名

指南：名词筛选法识别实体类

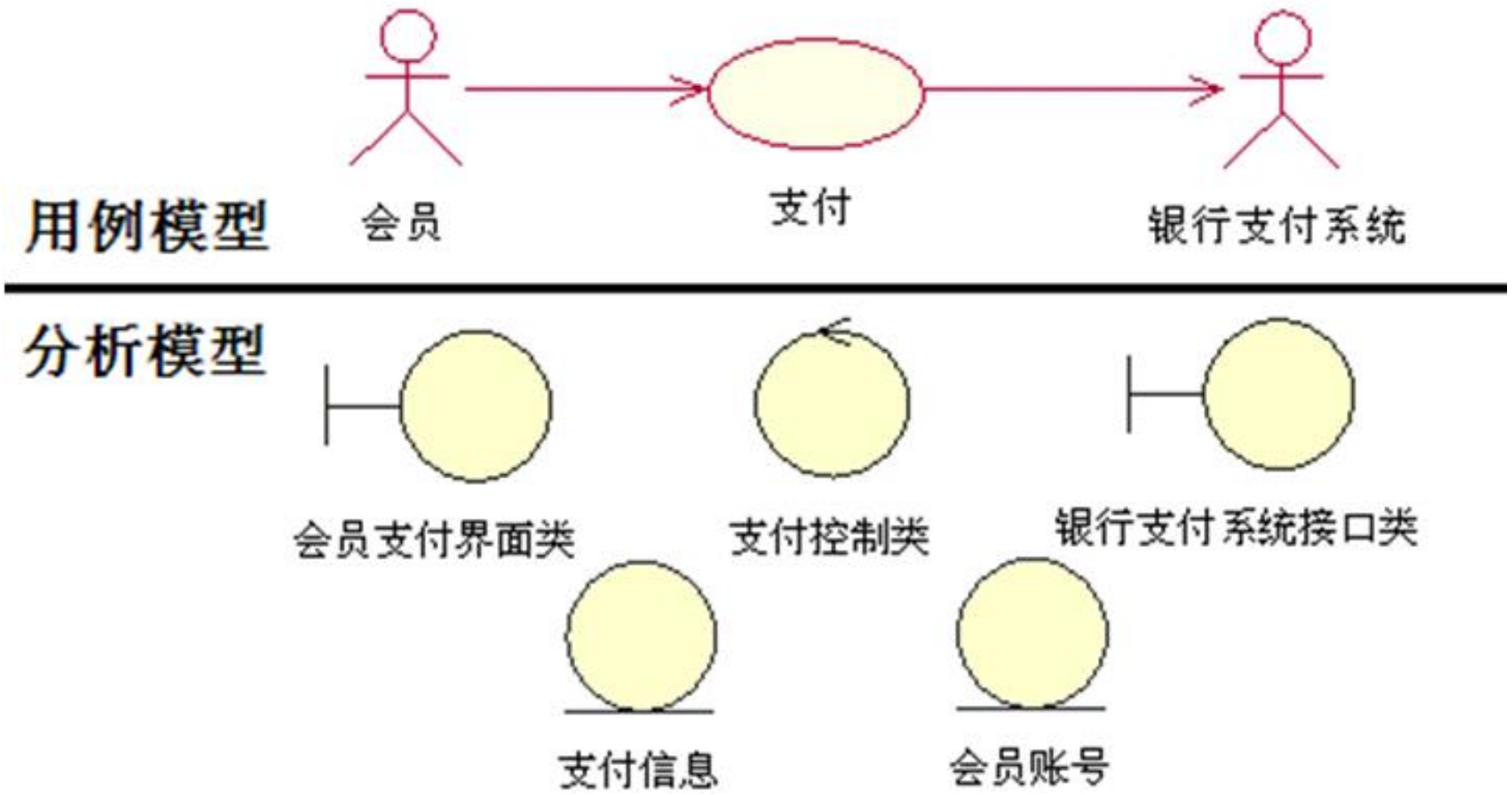
- 名词筛选法识别实体类的基本思路：
  - 将用例文档作为输入，找出文档中的名词或名词性短语，形成了实体类初始候选列表
  - 合并那些含义相同的名词
  - 删除那些系统不需要处理的名词
  - 删除作为参与者的名词
  - 删除与实现相关的名词
  - 删除那些作为其它实体类属性的名词
  - 对剩余的名词，综合考虑它在当前用例以及整个系统中的含义、作用以及职责，并基于此确定合适的名字，作为初始实体类存在

示例：候选实体类

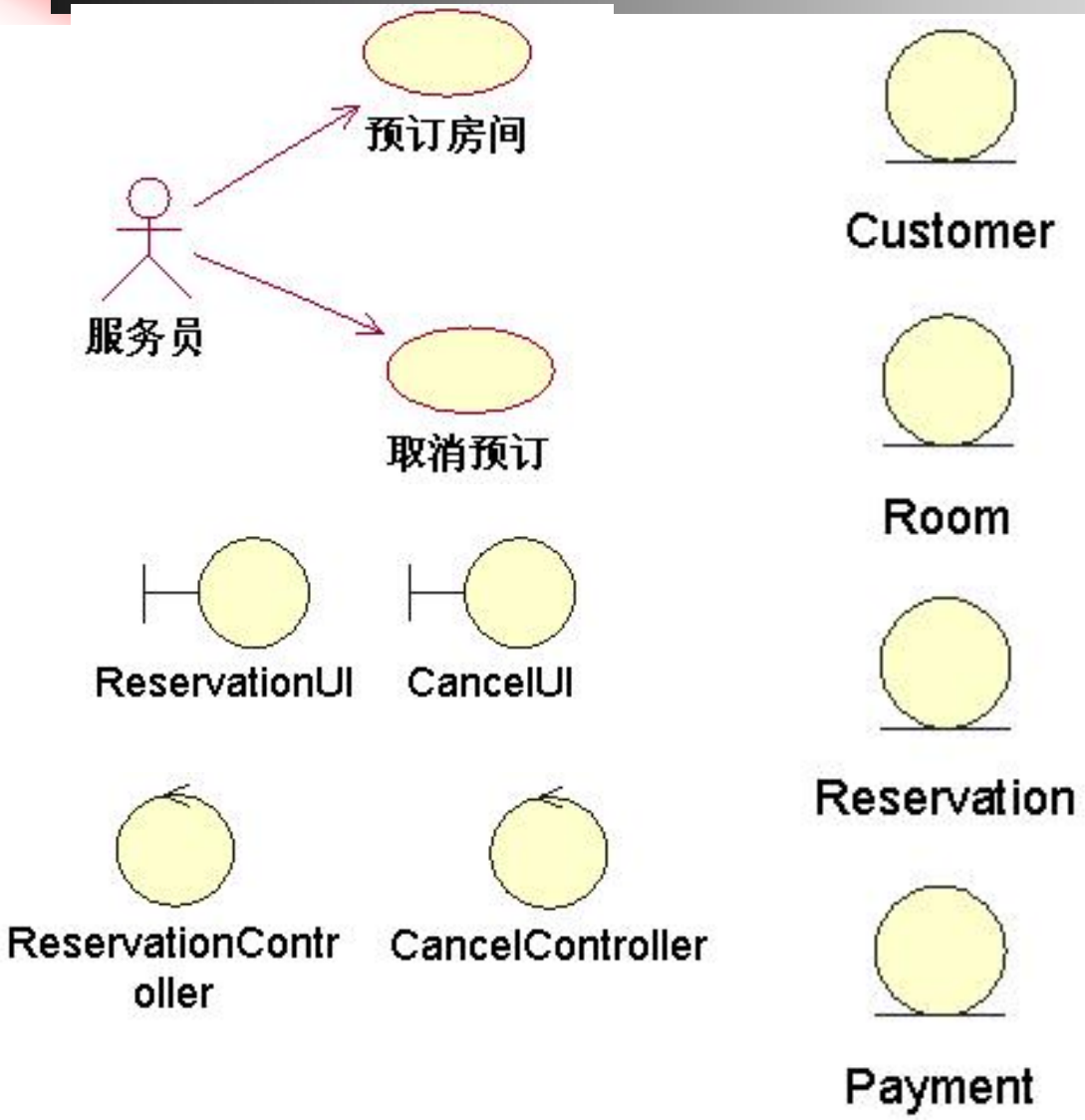
- “支付”用例基本路径中的候选实体类

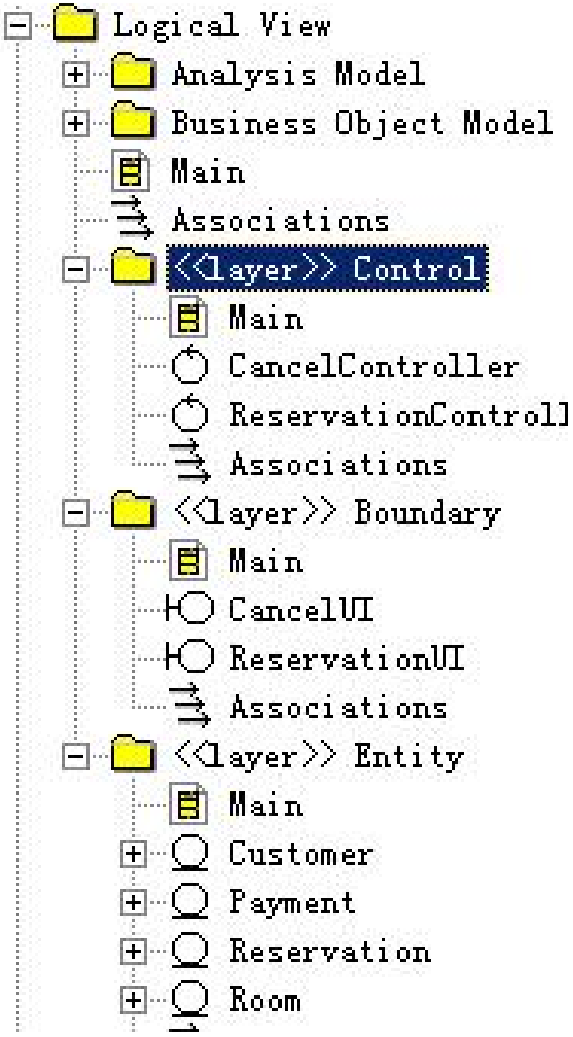


示例：总结-分析类



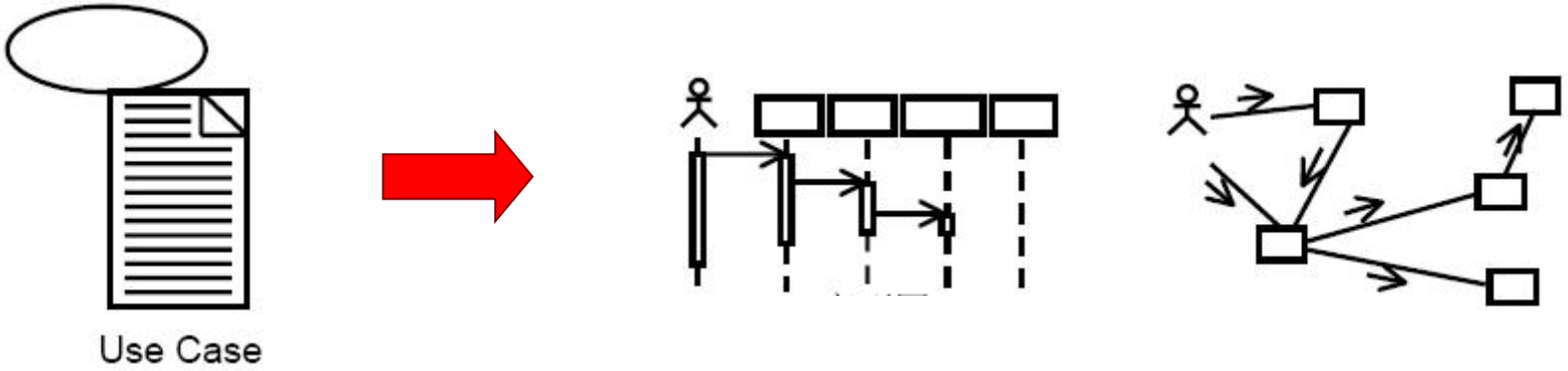
实例-旅店预订系统中识别分析类





2.2 将用例行为分配给类

- 面向对象系统是通过对象间的协作实现需求
  - 需求阶段通过自然语言描述
  - 分析设计阶段采用图形化方式描述协作过程
  - 利用交互图将用例行为分配给分析类

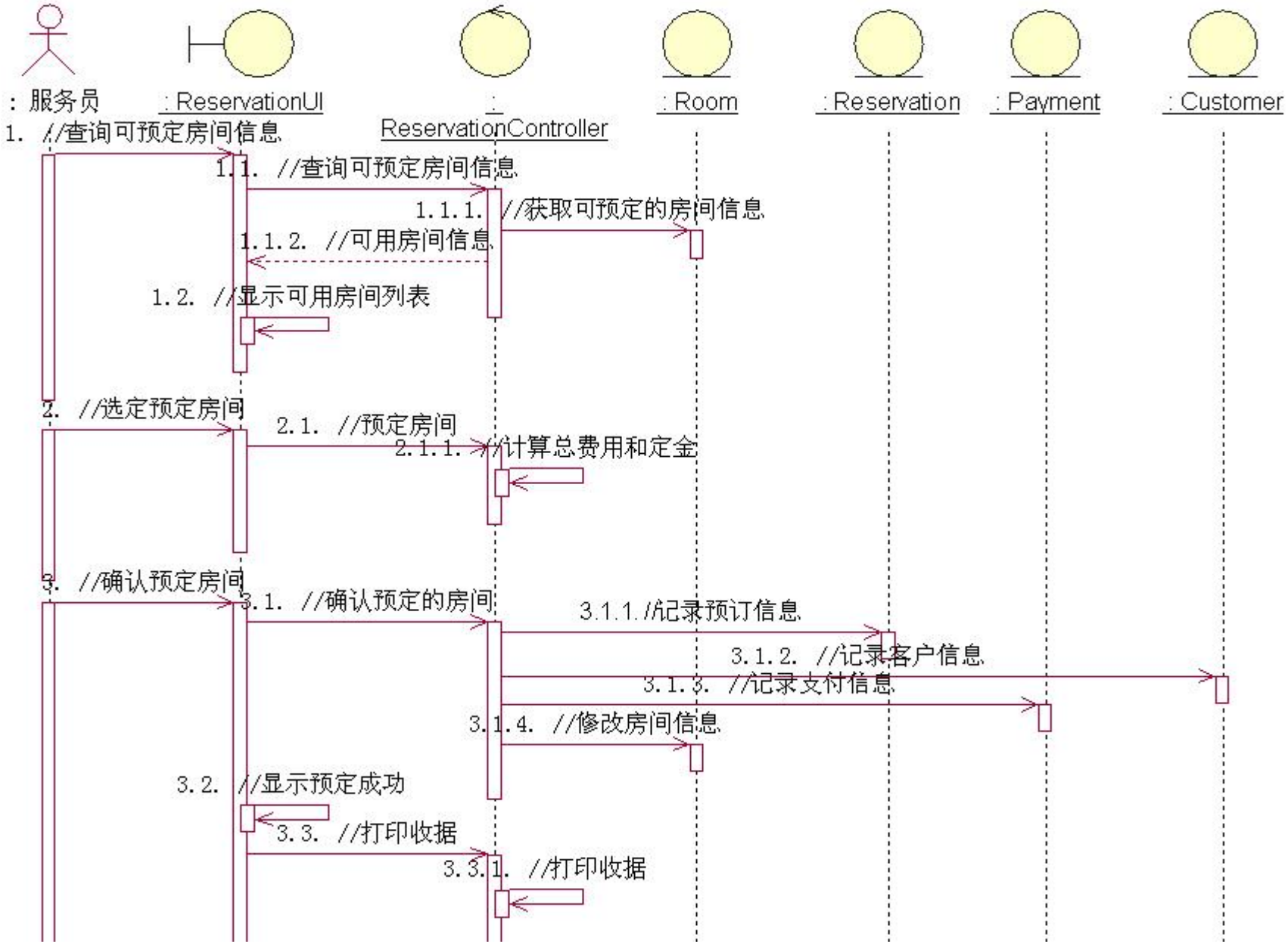


顺序图 Sequence Diagram

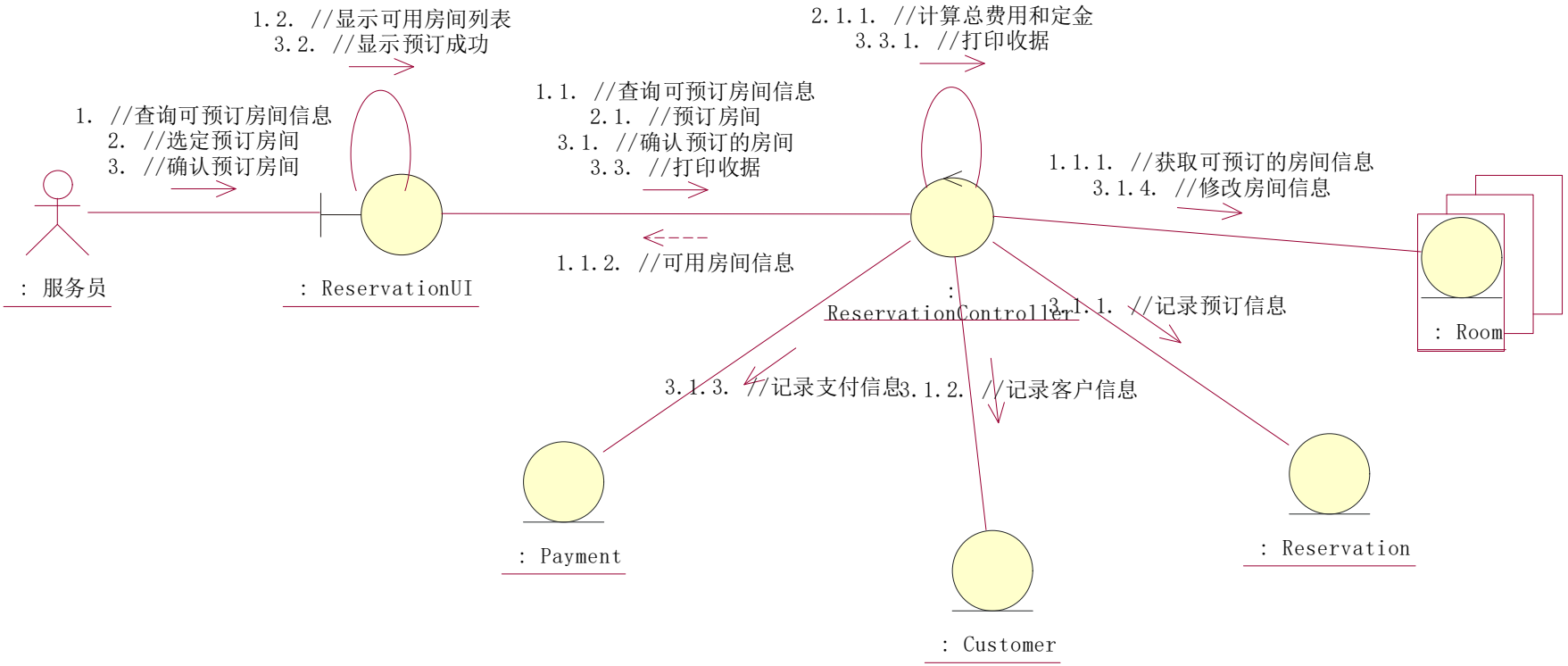
- 顺序图是一种交互图，描述对象之间的动态交互关系，着重体现对象间消息传递的时间顺序
  - 对象(Object)：对象、对象的生命线、对象的控制焦点和对象的删除
  - 消息(Message)：简单消息、同步消息、异步消息、返回消息
  - 交互片段(Interaction Frame)：分支、循环

利用顺序图进行职责分配

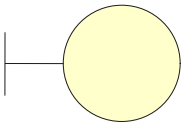
- 以B-C-E的方式绘制顺序图，并以Control类将控制逻辑隐藏起来
- 可以将对象之间的信息传递以“//”的方式标记，表明初步进行类的职责分配，该项信息尚未编写完全。
- 可以利用白话的方式将信息进行的说明，在信息长度不够时，可以加上UML的注释来做说明
- 分析阶段顺序图中所找出的对象可以放置到分析阶段的类图上，反之，也可以由分析阶段的类图上找出顺序图中所需的对象


：服务员
：ReservationUI
：
ReservationController
：Room
：Reservation
：Payment
：Customer
1. //查询可预定房间信息
1.1. //查询可预定房间信息
1.1.1. //获取可预定的房间信息
1.1.2. //可用房间信息
1.2. //显示可用房间列表
2. //选定预定房间
2.1. //预定房间
2.1.1. //计算总费用和定金
3. //确认预定房间
3.1. //确认预定的房间
3.1.1. //记录预订信息
3.1.2. //记录客户信息
3.1.3. //记录支付信息
3.1.4. //修改房间信息
3.2. //显示预定成功
3.3. //打印收据
3.3.1. //打印收据

协作图/通信图


1.2. //显示可用房间列表
3.2. //显示预订成功
2.1.1. //计算总费用和定金
3.3.1. //打印收据
1. //查询可预订房间信息
2. //选定预订房间
3. //确认预订房间
1.1. //查询可预订房间信息
2.1. //预订房间
3.1. //确认预订的房间
3.3. //打印收据
1.1.1. //获取可预订的房间信息
3.1.4. //修改房间信息
1.1.2. //可用房间信息
: 服务员
: ReservationUI
: ReservationController
: Room
3.1.1. //记录预订信息
3.1.3. //记录支付信息
3.1.2. //记录客户信息
: Payment
: Customer
: Reservation

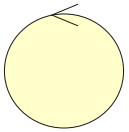
分析类职责分配



ReservationUI

(from Boundary Layer)

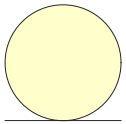
+ 显示可用房间列表()
+ 显示预订成功()
+ 查询可预订房间信息()
+ 选定预订房间()
+ 确认预订房间()



ReservationController

(from Control Layer)

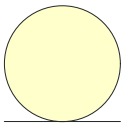
+ 查询可预订房间信息()
+ 预订房间()
+ 确认预订的房间()
+ 打印收据()
+ 可用房间信息()
+ 计算总费用和订金()



Customer

(from Entity Layer)

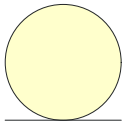
+ 记录旅客信息()



Payment

(from Entity Layer)

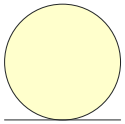
+ 记录支付信息()



Reservation

(from Entity Layer)

+ 记录预订信息()



Room

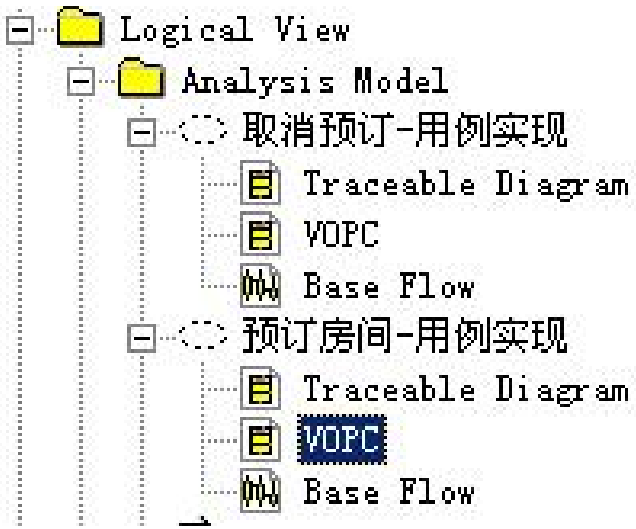
(from Entity Layer)

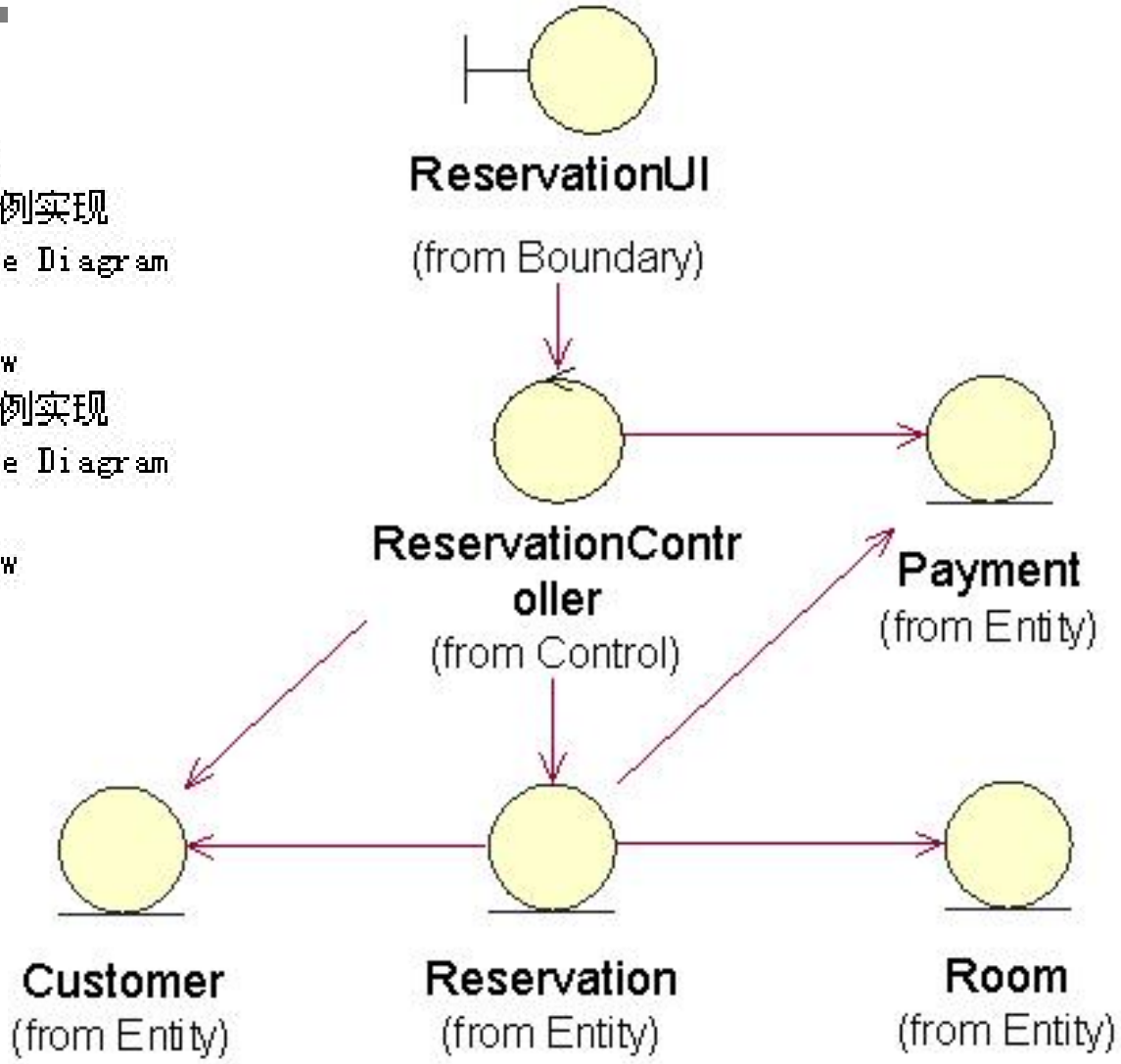
+ 获取可预订的房间信息()
+ 修改房间信息()

2. 3. VOPC图

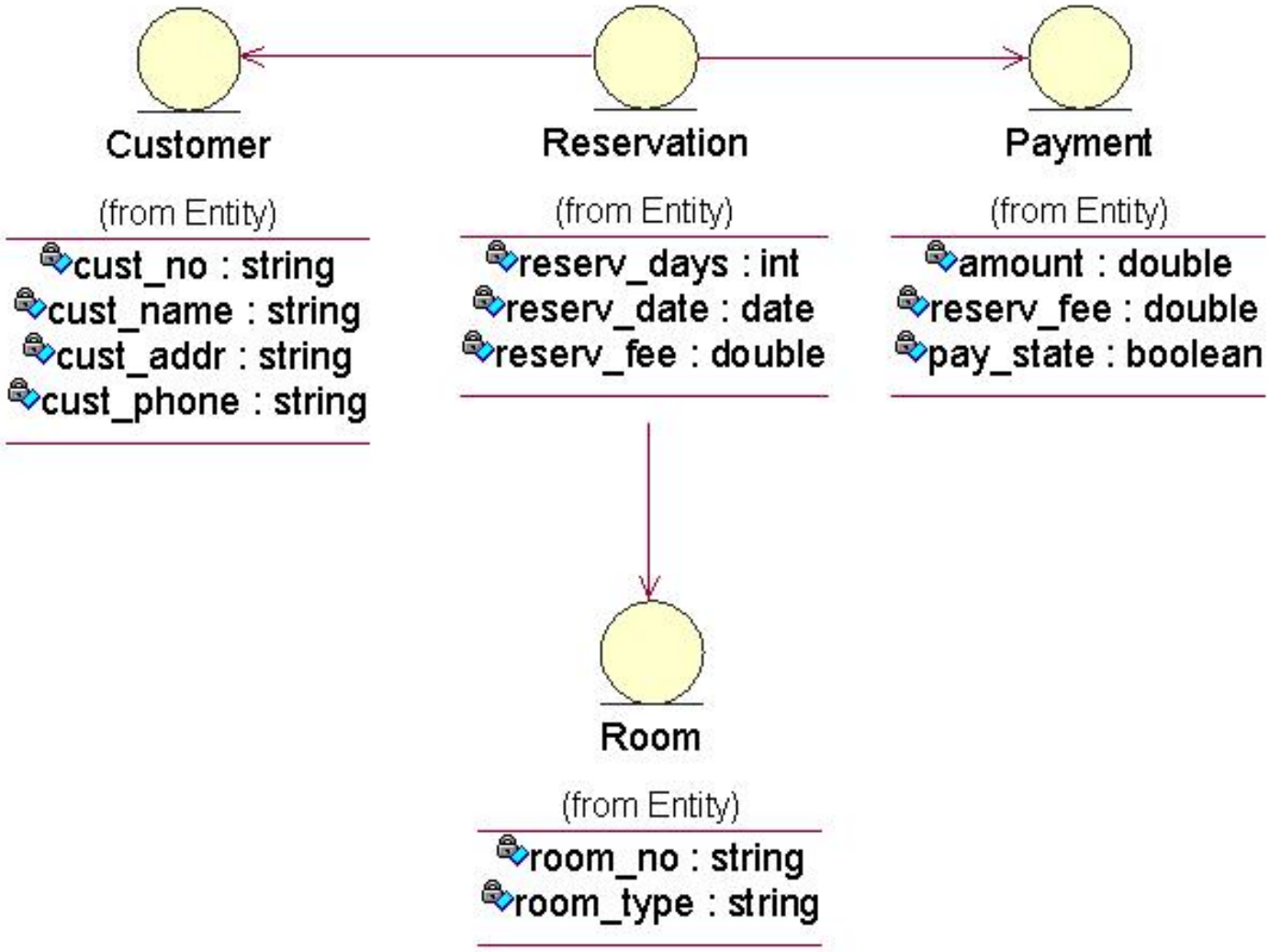
- 对于每个“用例实现”都存在若干张交互图进行描述，而这些交互图中会使用到各种分析类的对象
- 对于每一个“用例实现”，需要绘制与之相关的类图，即VOPC图
  - 参与类类图(VOPC, View Of Participating Classes Class Diagram)
  - 类图中的元素来自于交互图中的对象
  - 类图中的关系来自于交互图中的消息(和业务对象模型)，分析阶段主要使用关联关系，也可根据业务模型引入泛化、聚合等关系

实例：绘制VOPC类图


Logical View
Analysis Model
取消预订-用例实现
Traceable Diagram
VOPC
Base Flow
预订房间-用例实现
Traceable Diagram
VOPC
Base Flow


ReservationUI
(from Boundary)
ReservationContr
oller
(from Control)
Payment
(from Entity)
Customer
(from Entity)
Reservation
(from Entity)
Room
(from Entity)

实例：旅店预订系统实体类类图



2.4. 限定分析机制

- 建立分析类和分析机制的对应图

分析类	分析机制	说明
房间	持久性	房间信息需存储在数据库中
预订	持久性	预订相关信息需存储在数据库中
支付	持久性、安全性、	支付明细信息需存储在数据库中，并不允许随意修改
旅客	持久性	旅客信息需存储在数据库中