

Lecture 10. 系统分析(System analysis) 用例分析(Use Case Analysis) II

**Object Oriented Modeling Technology
面向对象建模技术**

**Professor: Yushan Sun
Fall 2022**

内容安排

1. 用例实现-将用例行为分配给类

2. 完成参与类类图

3. 定义分析类

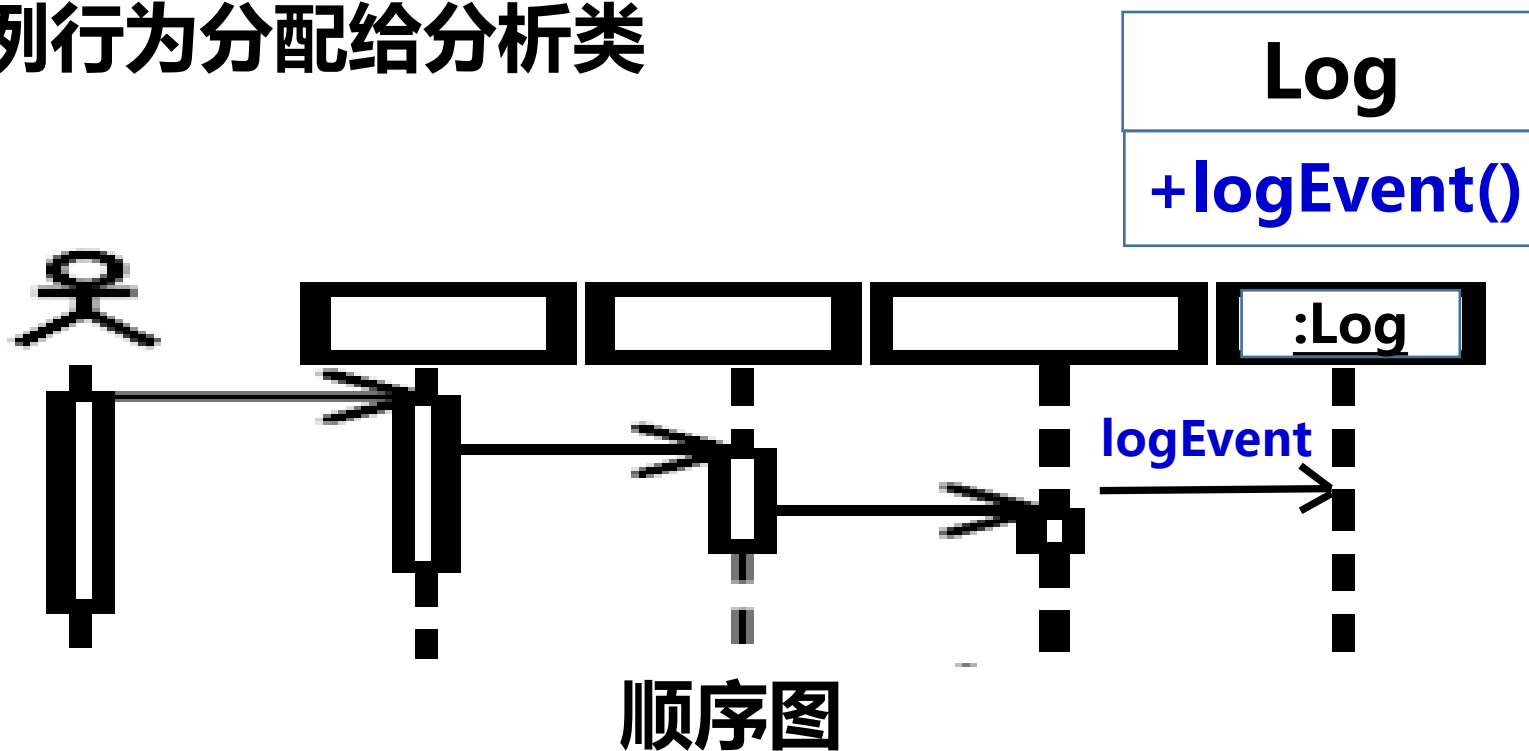
本节目的：

- 1) 给分析类增加方法
- 2) 给分析类增加属性

用例实现-将用例行为分配给类

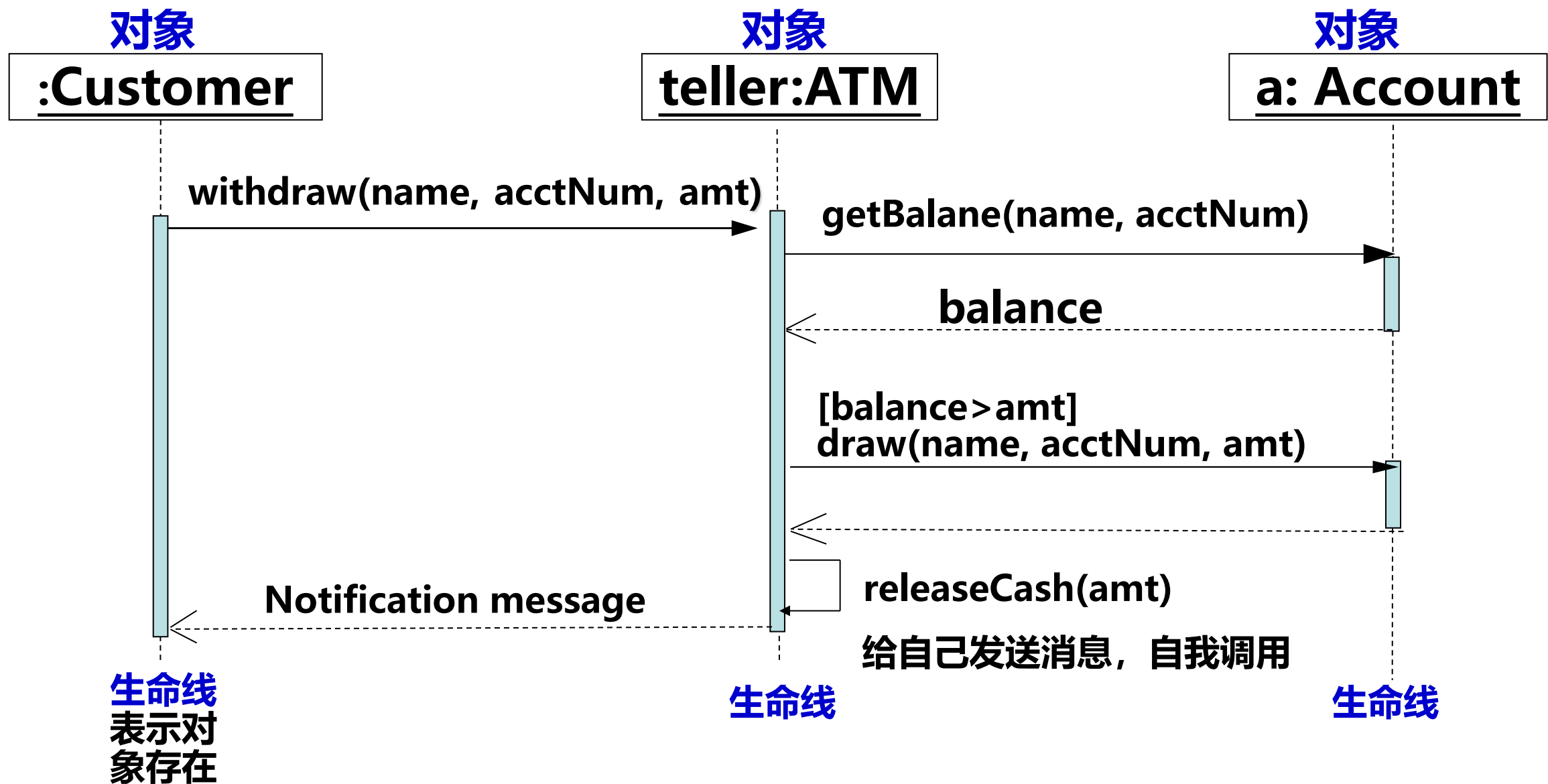
3. 将用例行为分配给类

- 面向对象系统是通过对象间的协作实现需求(用例)的
 - 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(顺序图)
 - 利用交互图将用例行为分配给分析类



用例文档(包括用例图)

例1：ATM机“取款”顺序图



发送消息：箭头指向哪个对象，则该对象所对应的类就包含对应消息的方法。

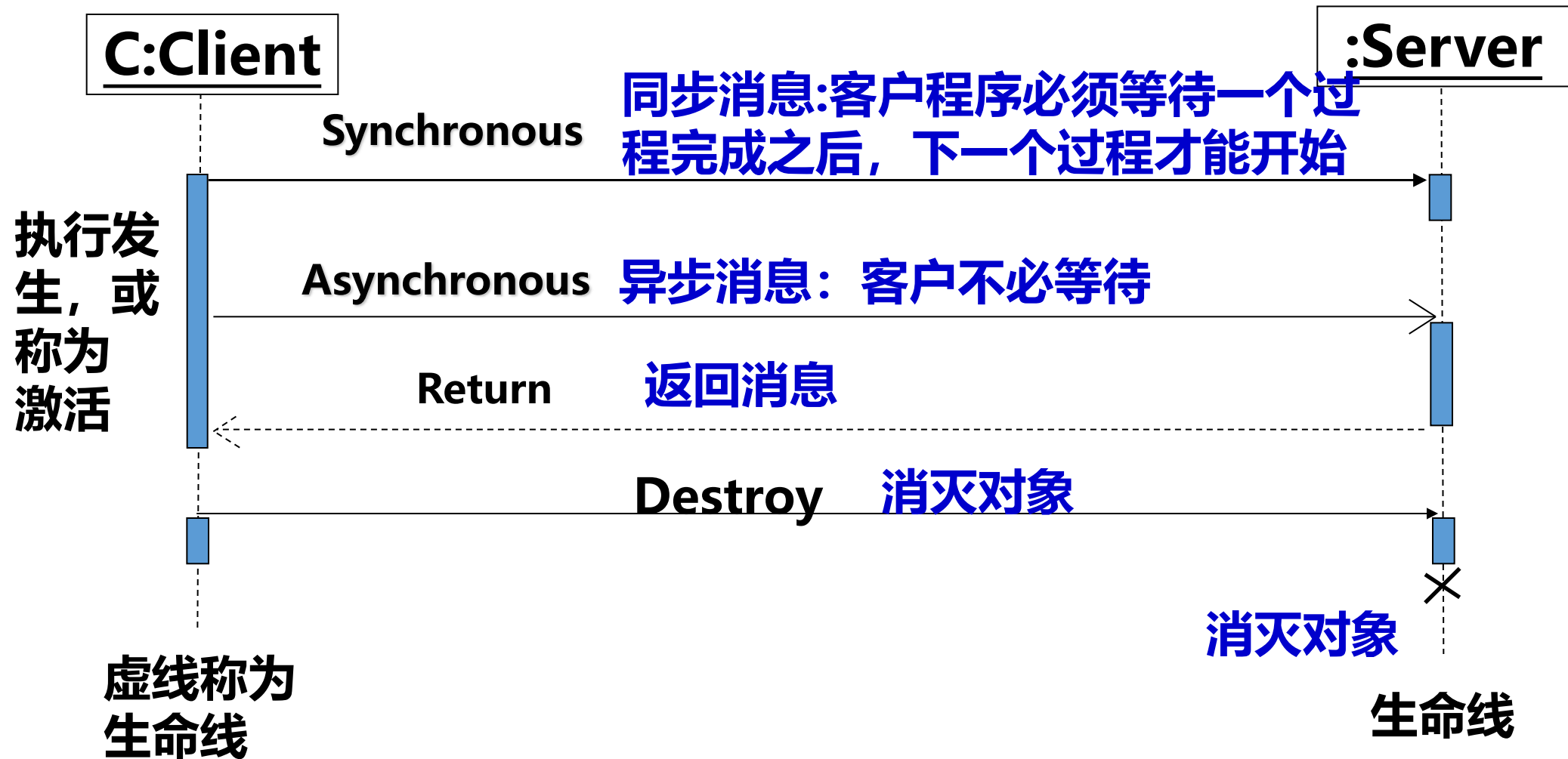
```
public class Account{  
    public double getBalane(name: String, acctNum: String);  
    public void draw(name: String, acctNum: String, amt: double);  
}
```

```
public class ATM{  
    public void withdraw(name, acctNum, amt);  
    public void releaseCash(amt)  
}
```

顺序图Sequence Diagram

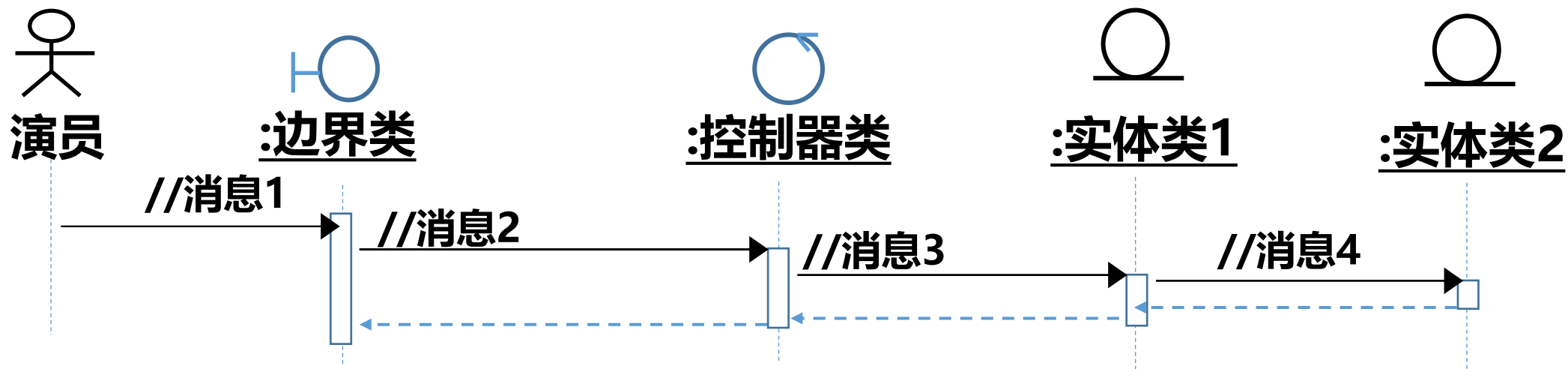
- 顺序图是一种交互图，描述对象之间的**动态交互**关系，着重体现对象间消息传递的时间顺序
 - 对象(Object)：对象、对象的生命线、对象的执行发生和对象的删除
 - 消息(Message)：简单消息、同步消息、异步消息、返回消息

消息的类型:



利用顺序图进行职责分配

- 以边界类-控制类-实体类(Boundary-Control-Entity)的方式绘制顺序图, 并以Control类将控制逻辑隐藏起来



- 绘制顺序图所需要的对象从哪里来呢?
 - ✓ 通常是在分析类图中找出顺序图中所需的对象 (类)
 - ✓ 若分析类图中真没有合适的类(对象), 则可以定义一个**新类**, 然后将**新类**添加到分析类图上

指南：将职责分配到分析类

- 以分析类的构造型作为分配标准
 - **边界类**：承担与参与者(演员，角色)进行通信的职责
 - **控制类**：接收参与者由用户界面传递来的请求，然后，根据请求的类型，决定调用哪些实体类。承担协调用例参与者与系统之间交互的职责
 - **实体类**：实体类代表系统的核心概念，来自于业务中实体对象的归纳与抽象，用于记录系统所维护的数据和对数据的处理行为。承担对被封装的内部数据进行操作的职责

例2：机票类的例子

AirTicket

-airline: String
-departCity: String
-destnCity: String
-departTime: Time
-destnTime: Time
-price: float

+setAirline(a: String): void
+setDepartCity(city: String): void
+setDestnCity(city: String): void
+setDepartTime(t: Time): void
+setDestnTime(t: Time): void
+setPrice(price: float): void
+getAirline(): String
+getDeparture(): String
+getDestination(): String
+getDepartTime(): Time
+getDestnTime(): Time
+getPrice(): float

Java 代码:

```
AirTicket a = new AirTicket();  
a.setAirline( "MU3015" );  
a.setDepartCity( "北京首都国际机场T2" );  
a.setDestnCity( "广州白云机场" );  
a.setDepartTime(10: 30);  
a.setDestnTime(13: 55);  
a.setPrice(650);
```

```
AirTicket a = new AirTicket( "MU3015" ,  
                                "北京首都国际机场T2" ,  
                                "广州白云机场" );
```

```
a.setDepartTime(10: 30);  
a.setDestnTime(13: 55);  
a.setPrice(650);
```

实体类:

- 用于维护AirTicket的数据和
- 对数据的处理行为

例3：银行账户类的例子

在该Account类的设计中，出来gets和sets方法以外，还有银行的主要业务操作

- deposit(a: float): void
- withdraw(a: float): void
- transfer(a: float): void

```
Account a = new Account( "Mike Lee" ,  
                          "738699231800" ,  
                          "230208200108250931" );
```

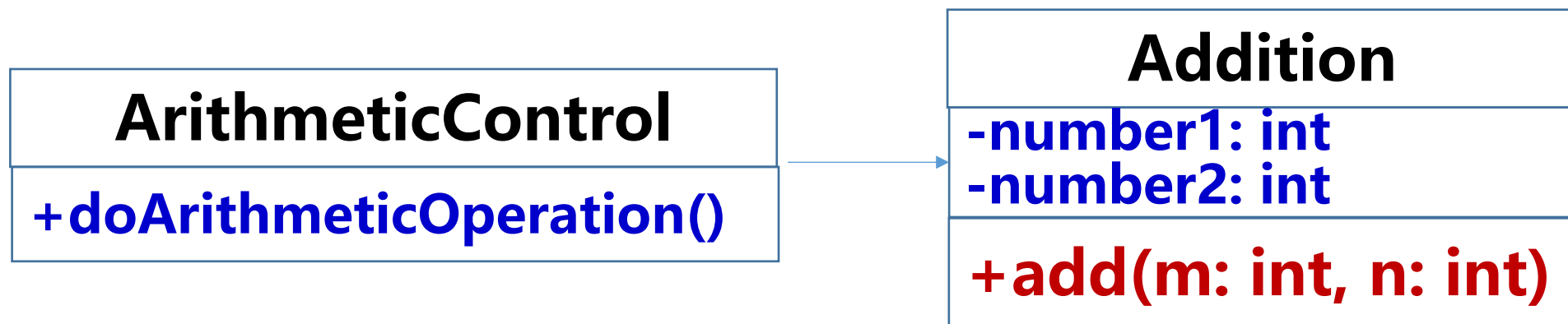
```
a.setPassword( "tiger1993black" );  
a.setBalance(3008);  
a.deposit(20000);
```

Account
<ul style="list-style-type: none">-ownerName: String-acctNum: String-ownerIdNum: String-balance: float-password: String
<ul style="list-style-type: none">+getOwnerName(): String+getAcctNum(): String+getownerIdNum(): String+getBalance(): float+getPassword(): String+setBalance(): void+setPassword(): void+validateAcct(): boolean+deposit(a: float): void+withdraw(a: float): void+transfer(a: float): void

指南：怎样将职责分配到分析类(续)

- **专家模式**：将职责分配给具有当前职责所需要的数据的类
 - 如果一个类有这个数据，就将职责分配给这个类

例4：考虑一个算数程序片段如下；

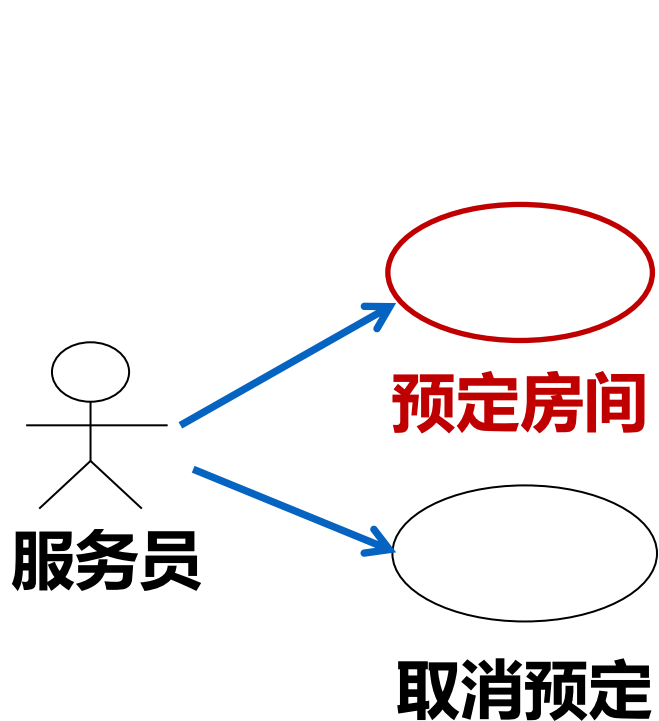


现在想增加一个加法方法

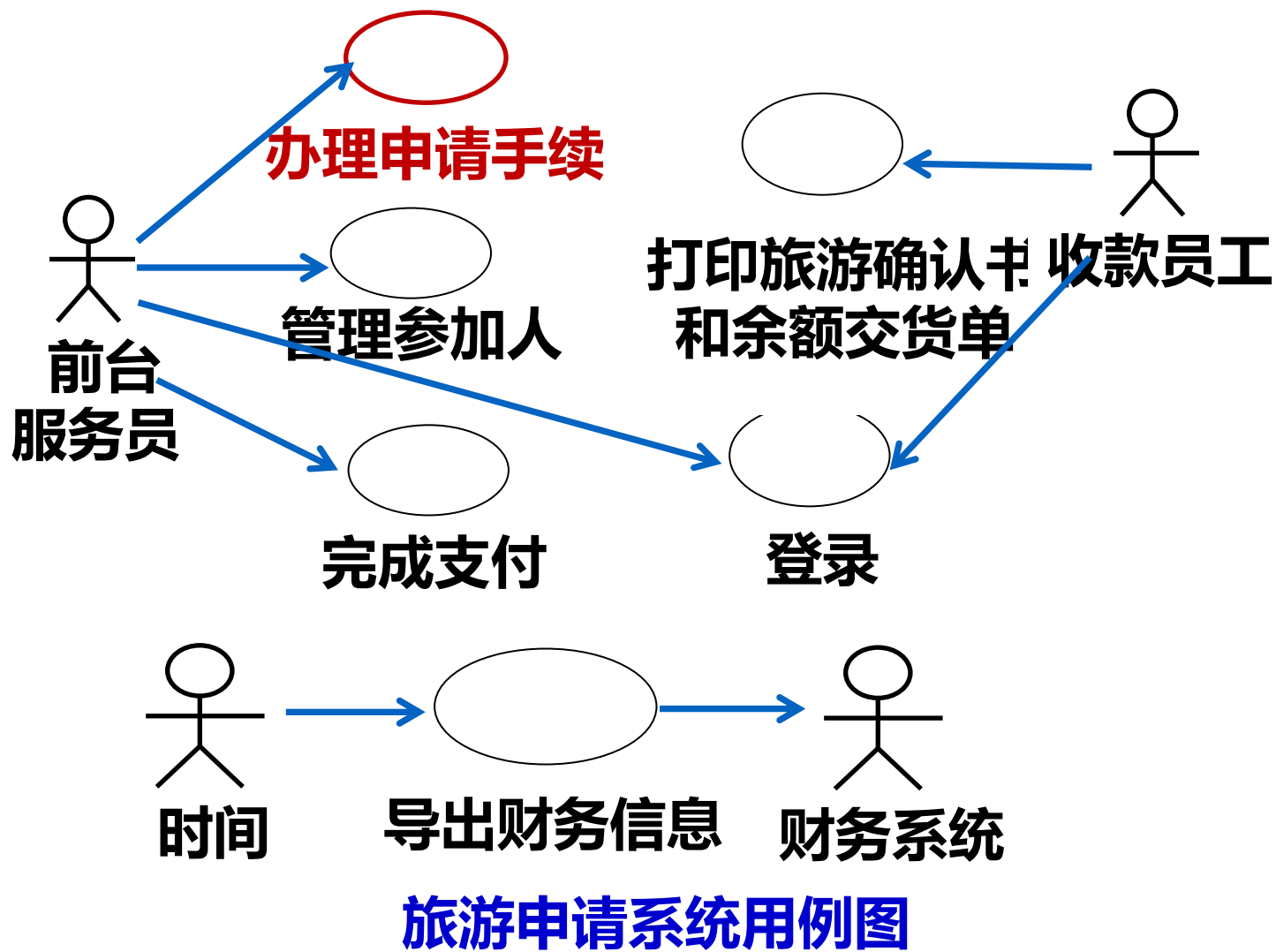
`add(m: int, n: int)`

计算这两个数m和n的和。问题：这个方法应该放在那个类里面。

- 由于课时所限，本讲稿只给出
 - 1) 旅店预订系统的“预订房间”用例实现
 - 2) 旅游申请系统的“办理申请手续”用例实现



旅店预订系统用例图



旅游申请系统用例图

实例：旅店“预定房间”用例的实现

- 根据“预定房间”用例描述的步骤，绘制顺序图
- 即，将“预定房间”用例描述（参与者-系统）“翻译”成顺序图(系统内部对象之间的交互)
- 顺序图中的某个消息message接受者对象的类将会包含message作为方法

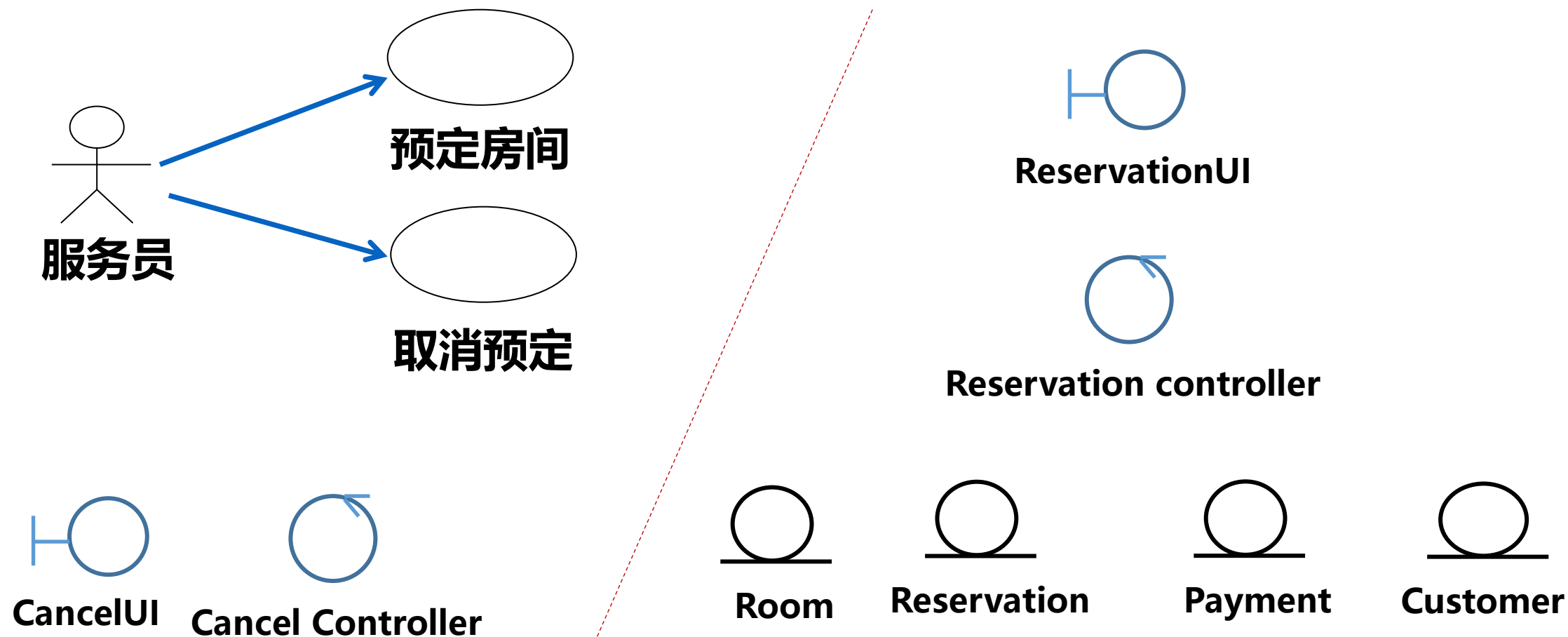
“预定房间” 用例描述

用例描述

基本事件流

1. 用例起始于旅客现场需要预订房间
2. 服务员按照旅客要求设定查询条件(D1)来查询可预订的房间信息 (D2)
3. 系统显示所有可以预订的房间列表(A1)(B1)
4. 服务员为旅客选定所需要的房间(A2)，并输入预订的时间和天数
5. 系统计算所需的总费用和预付定金金额(B2)
6. 旅客现场用现金支付所需的定金(A3)
7. 服务员将支付信息记录到系统中，并进行预订操作
8. 系统保存本次预订信息(D3)，显示预订成功消息 (A4)
9. 系统打印预订收据后，用例结束

实例-旅店预订系统中识别分析类



请注意观察，思考这7个类的对象是怎样协作以实现“预定房间”用例的



:服务员



:ReservationUI



:ReservationController



:Room



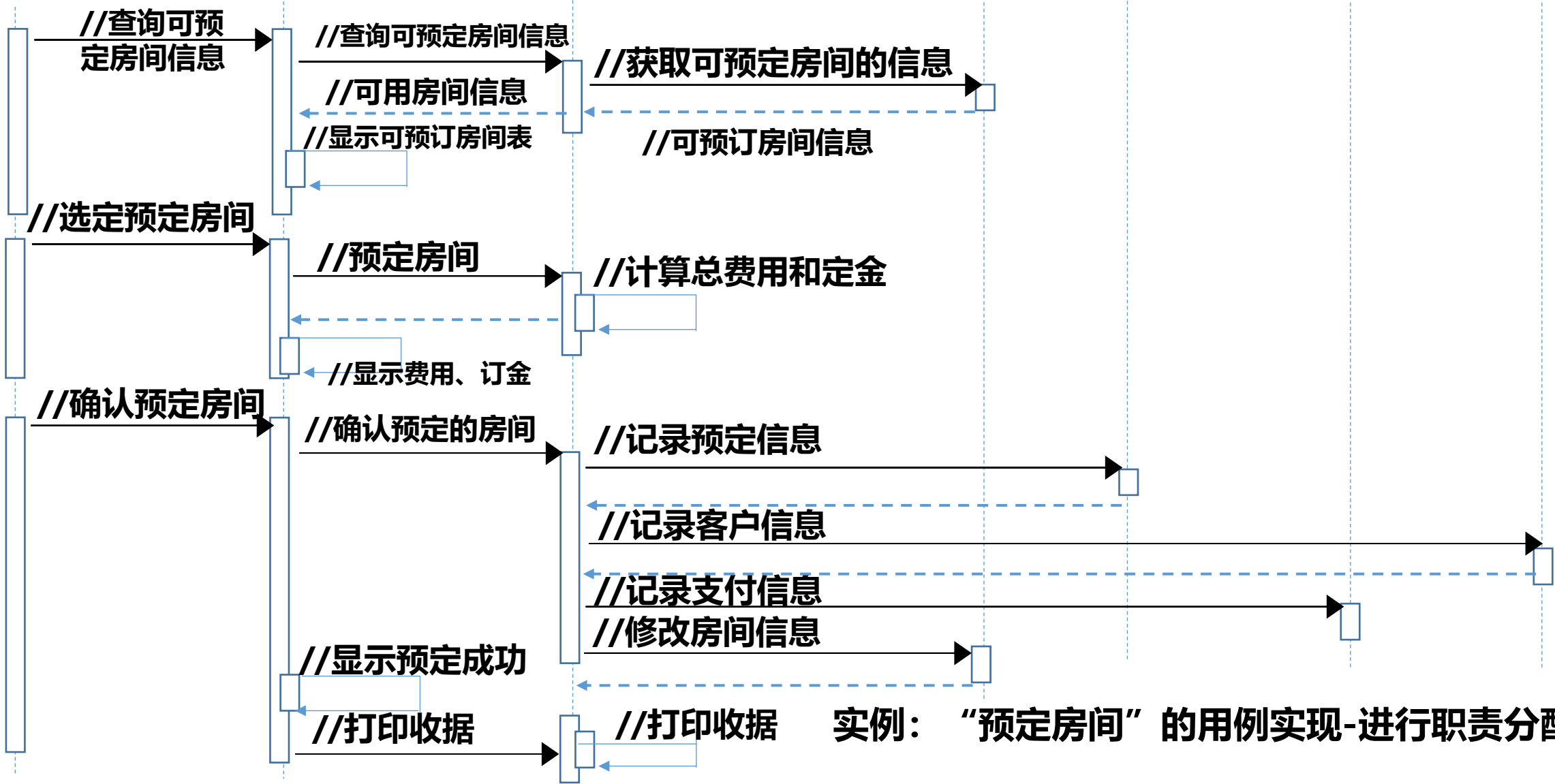
:Reservation



:Payment



:Customer



实例：旅游团“办理申请手续”用例的实现

- **根据“办理申请手续”用例描述，绘制顺序图**
- **将“办理申请手续”用例描述（参与者-系统）翻译成顺序图(系统内部对象之间的交互)**
- **顺序图中的某个消息message接受者对象的类将会包含message作为方法**

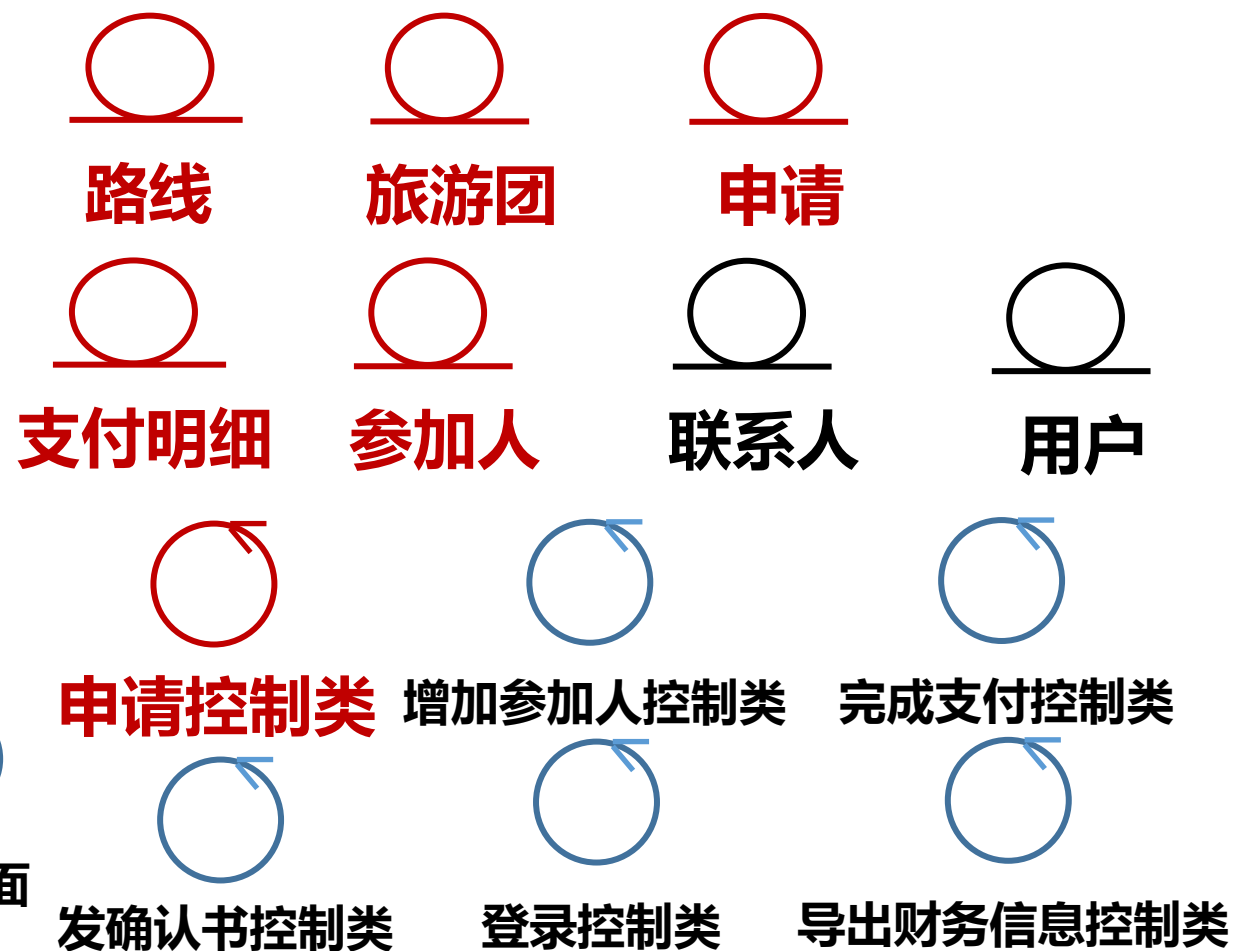
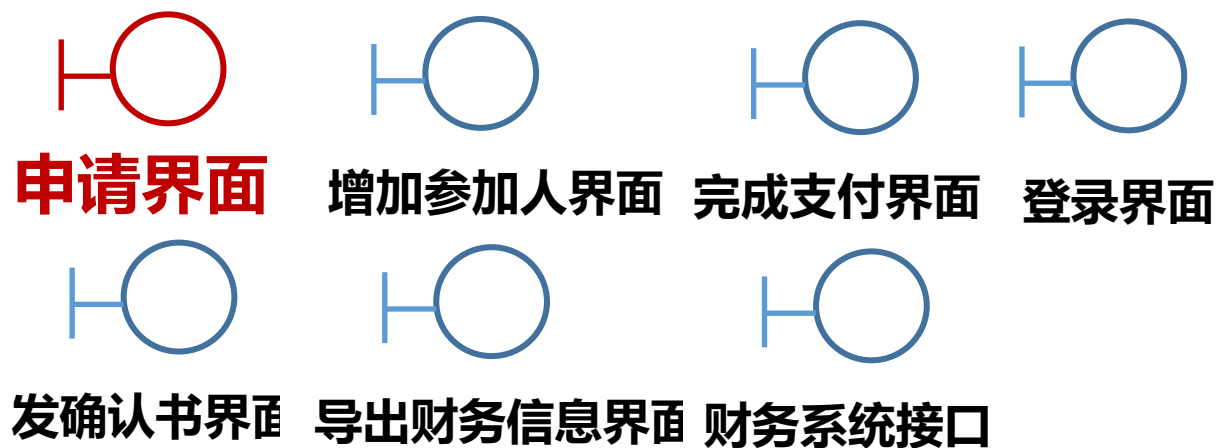
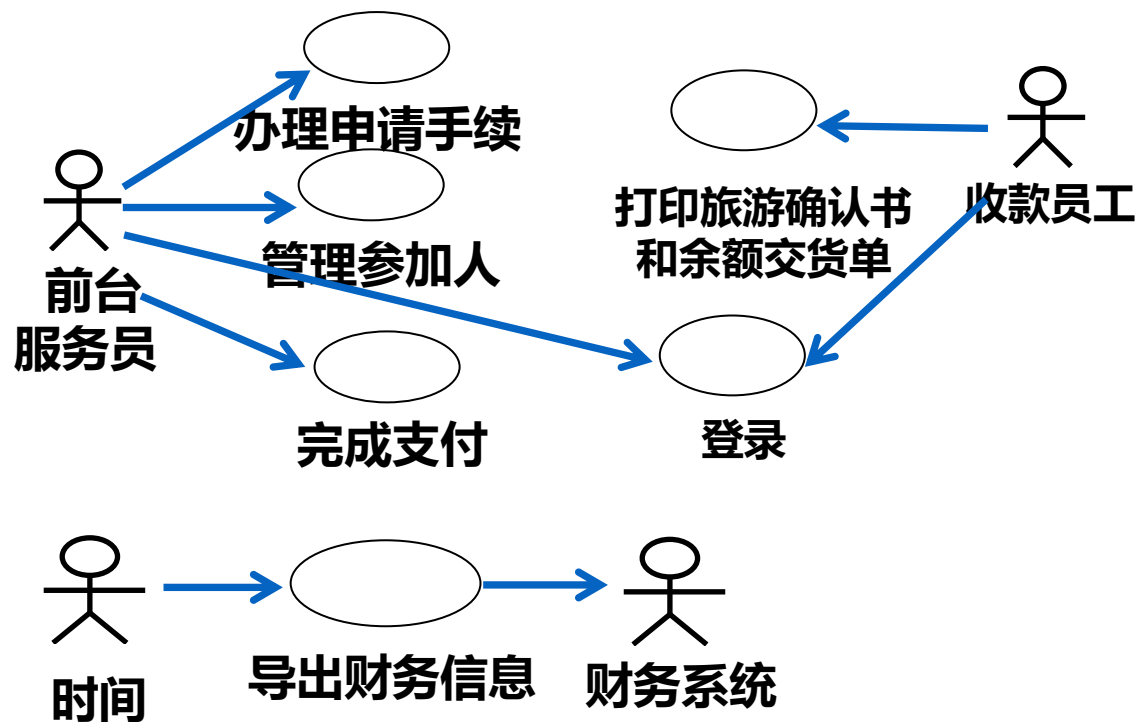
表4-12 “办理申请手续” 用例文档(续表)

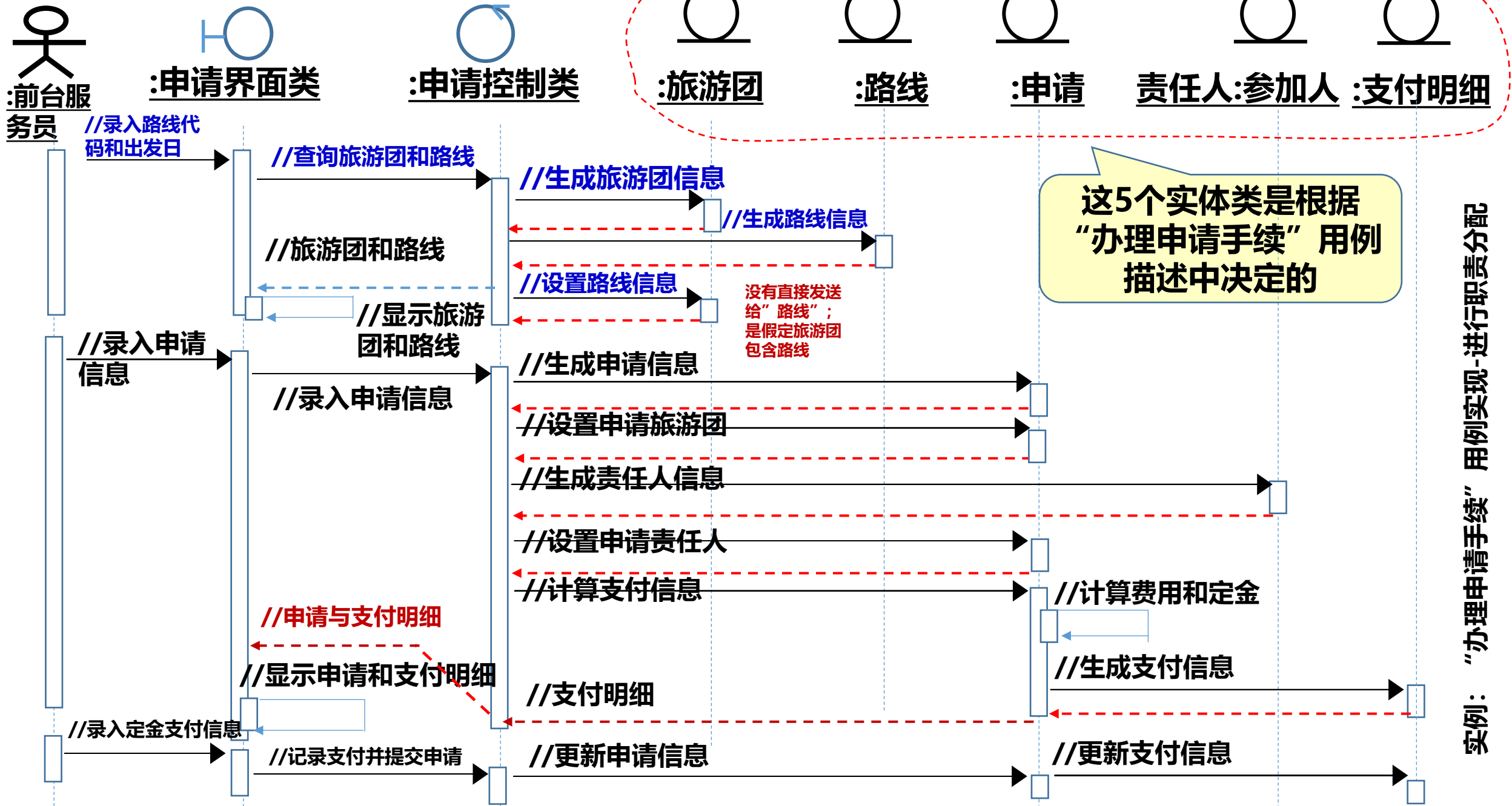
用例描述

基本事件流

- 1. 用例起始于旅客需要办理申请手续
- 2. 前台服务员录入要申请的旅游团旅行路线代码和出发日期
- 3. 系统查询要申请的旅游团信息(A-1)
- 4. 系统显示查询到的旅游团和相关路线信息 (D-1) , (A-2, A-3)
- 5. 前台服务员录入本次申请信息(D-2)
- 6. 系统计算并显示旅行费用的总额和申请定金金额
- 7. 申请责任人缴纳定金, 前台服务员录入定金信息, 提交本次申请信息
- 8. 系统保存该申请信息 (A-4) ,用例结束

实例-旅游申请系统中的分析类





实例：“办理申请手续”用例实现-进行职责分配

说明：实际绘制时序图的时候

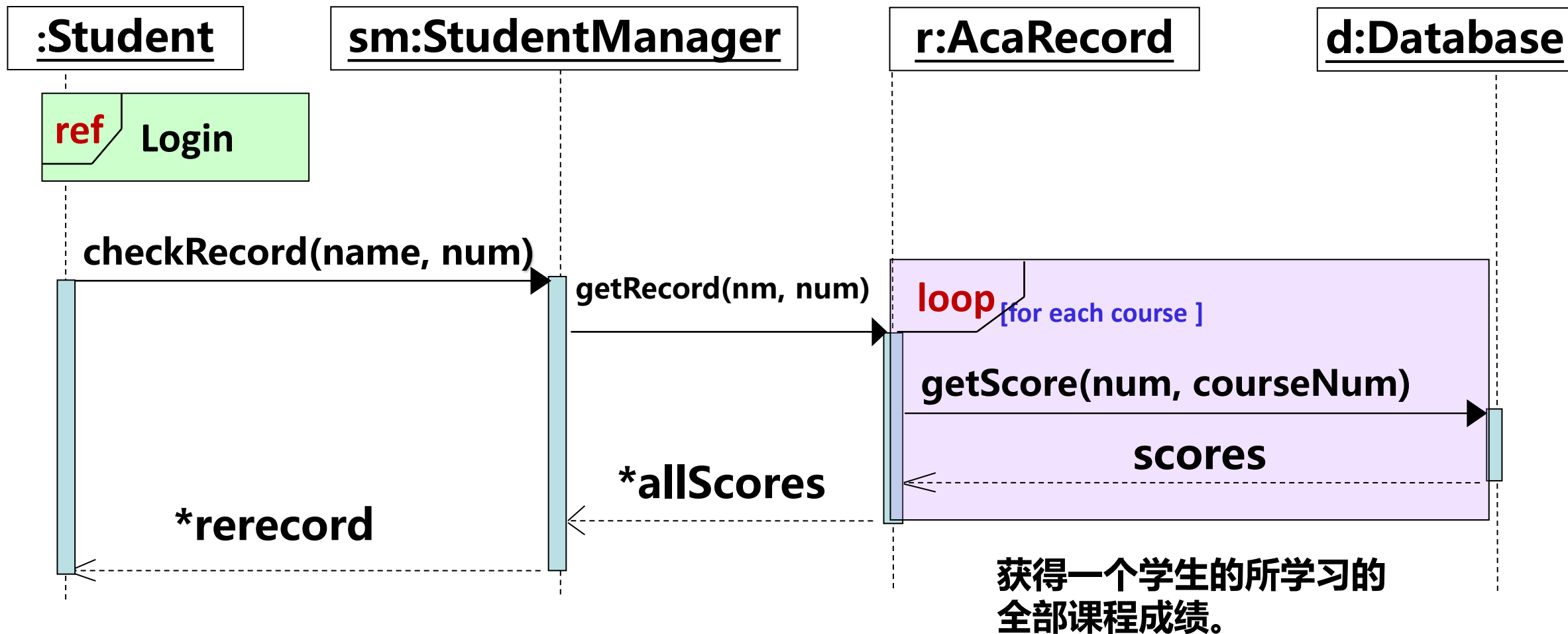
- 如果依据以上的5个实体类对象所绘制的顺序图能实现了“办理申请手续”的话，那么说明这5个实体类是合适的
- 如果以上的5个实体类还不能够实现用例的话，则可能还需要添加新的实体类；
- 如果觉得某个实体类不合适的话，也可以改变该实体类的名称
- 如果以上的某4个实体类就可以实现用例的话，说明那个没有被使用的实体类可能是多余的
- 在实际定稿之前总是可以反反复复改动的；这对新人是考验
- 智商，情商，第6感，**经验**，...
- 科学、设计

顺序图中的保护条件和循环

- 顺序图主要用于描述顺序的执行流程，对于简单分支或循环，可直接描述
 - 可以使用保护条件，用[]括起来描述，表示条件为真时才执行，否则不执行
 - 循环条件要在条件前加上 “*” 来描述，表示条件为真时重复执行
 - 其他约束用{ }括起来

顺序图中的交互使用

- UML 2.0之后，允许利用关键词ref，重用在其它地方定义的时序图，其目的是简化复杂的时序图
- **交互使用 (Interaction Use)**：欲复用在其它地方定义的交互
 - 使用ref记号表示复用（参考）在其它地方定义的时序图
- **循环(Loop)**：循环控制结构体,可用在时序图中
- 使用Loop记号表示一个循环体(见例子)



带有控制结构(loop)的学生成绩查询时序图，查询所有课程的成绩

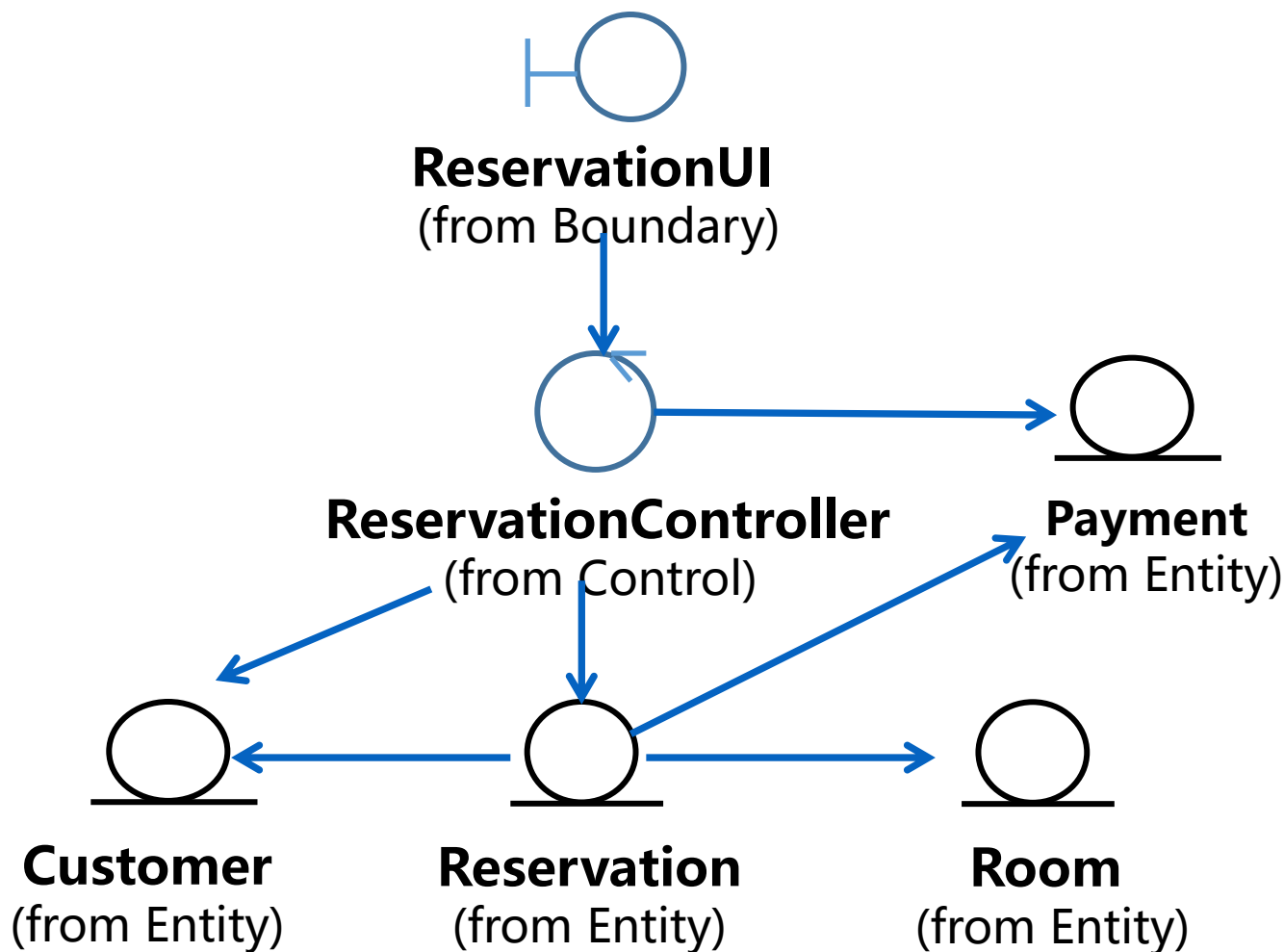
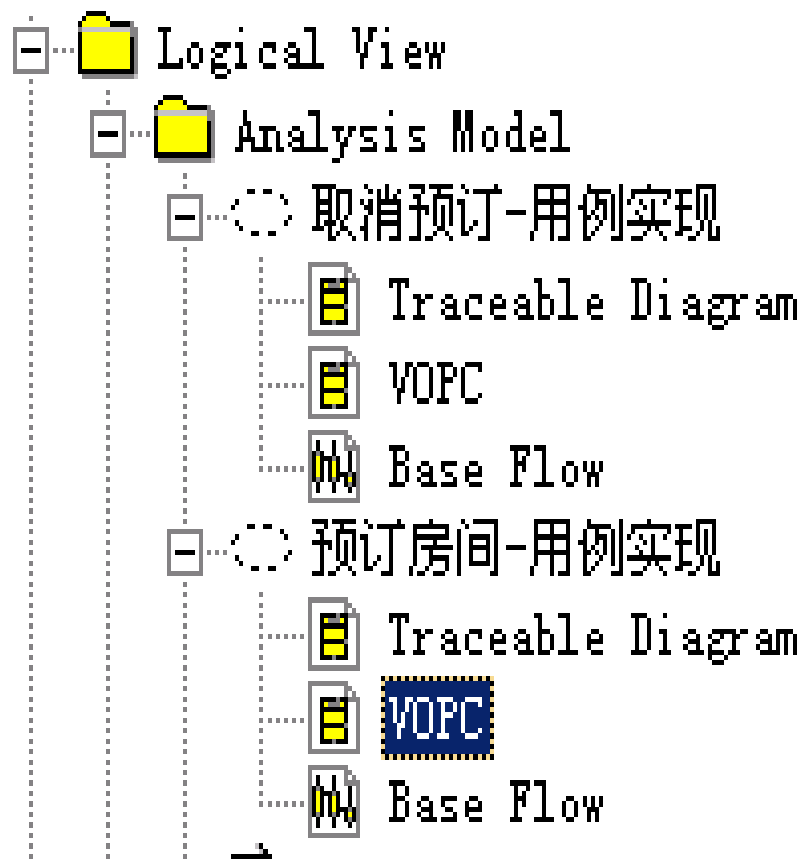
Return

完成参与类类图(VOPC图)

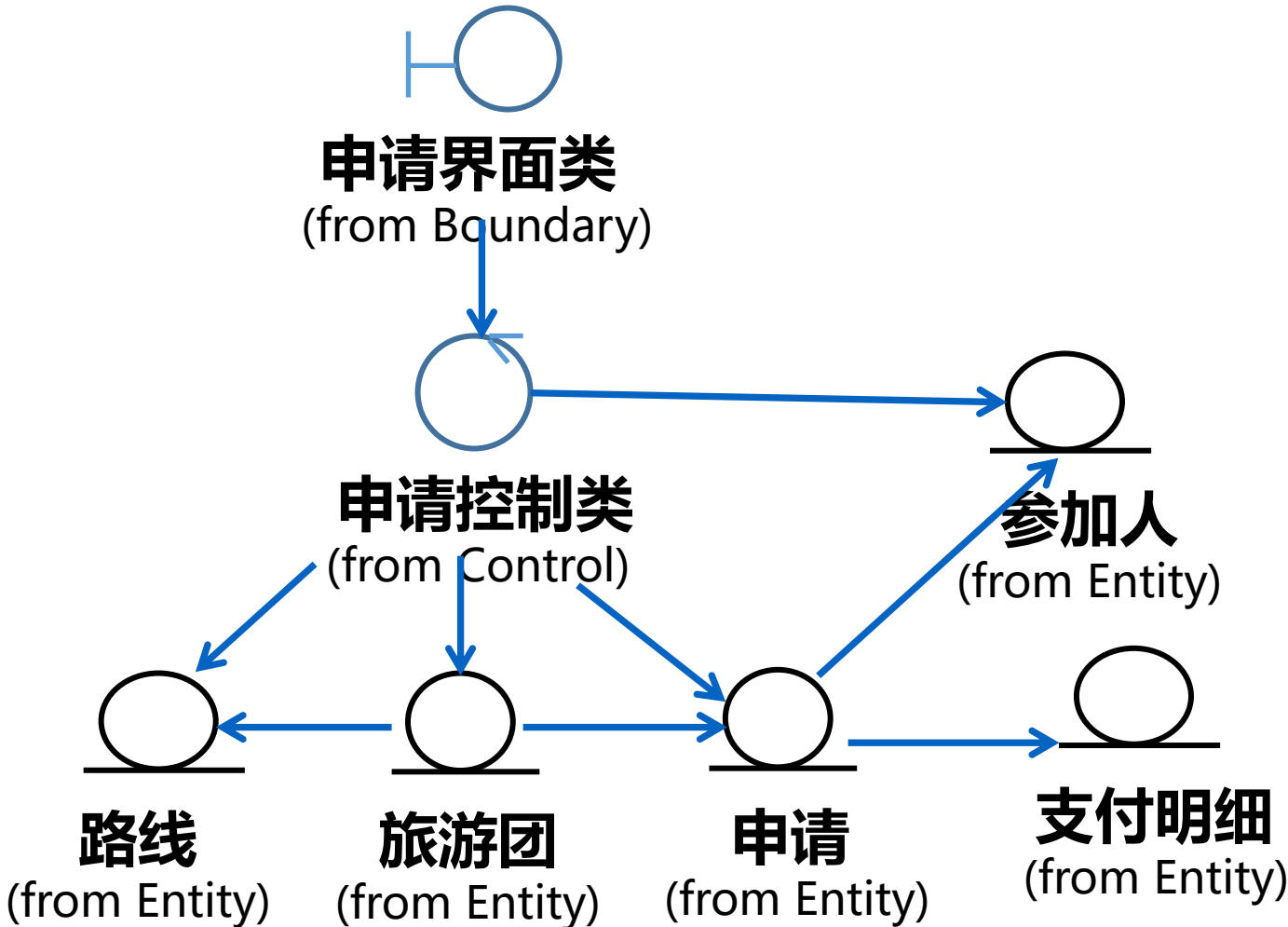
4. VOPC图(参与类类图)

- 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传递方向，**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

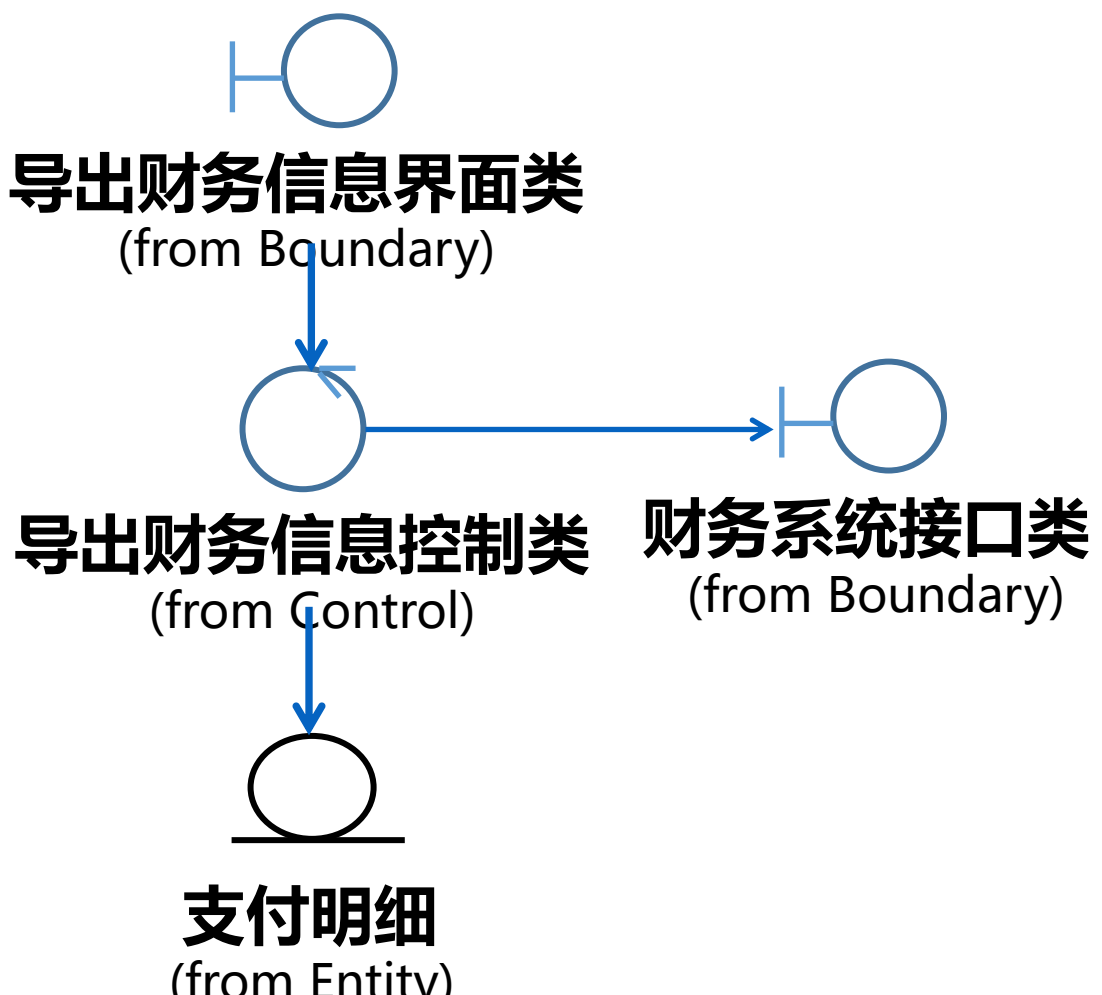
实例：绘制旅店“预订房间”的VOPC类图



实例：绘制旅游申请系统的2个用例的VOPC类图



“办理申请手续”的VOPC类图



“导出财务信息”的VOPC类图

当所有的用例的VOPC图都画完了以后，就由此可以推导出分析类图



定义分析类

定义分析类

- 最终目标是从系统的角度明确说明每一个分析类的
 - 职责和
 - 属性以及
 - 类之间的关系，从而构造系统的分析类视图。
- 根据这些视图来描述和理解目标系统，为后续的设计提供基本的素材。
- **局限**：在构造**用例实现**的过程中已经获得了**分析类的基本定义**，但那是在**单个用例实现**的基础上完成的，主要关注的是用例事件流的交互过程，而对**单个类自身的特征和行为缺少统一的考虑**

定义分析类的过程

- 从单个分析类入手，完成如下工作：
 - 1. 定义职责
 - 2. 定义属性
 - 3. 定义关系
 - 3.1 关联关系
 - 3.2 聚合关系
 - 3.3 泛化关系
 - 4. 统一分析类

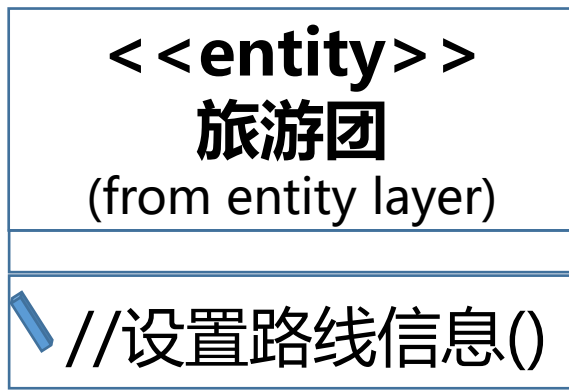
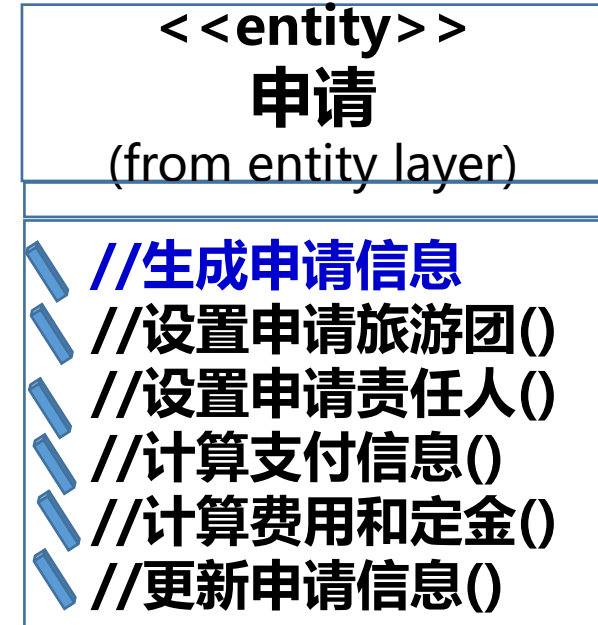
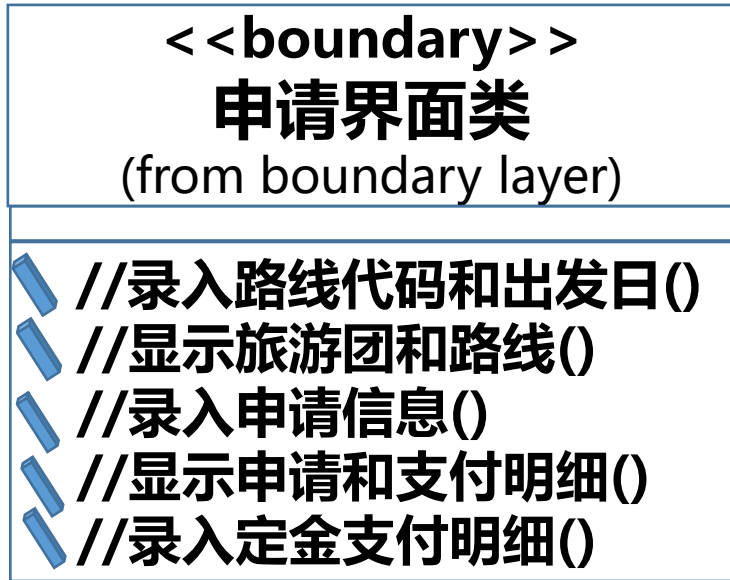
1. 定义分析类的职责

- 职责是要求某个类的对象所要履行的行为契约，在设计中将演化为类的操作(一个或多个)
- 获取类的职责
 - 从交互图中的消息：使用一个表格，收集所有的用例描述（顺序图）产生的消息
 - 消息指向哪个对象（类），哪个类就将有这个操作
 - 经过整理之后，就得到每个类应该包含的操作（C++函数，Java方法）

用例描述名称	消息
办理申请手续-用例实现	//生成申请信息 //设置申请旅游团 //设置申请责任人 //计算支付信息 //更新申请信息 //计算费用和定金
增加参加人-用例实现	
完成支付-用例实现	
打印旅游确认书和余额催款单-用例实现	
导出财务信息-用例实现	

与旅游团申请系统的 “申请类” 相关的消息列表

实例：定义分析类的职责



**用例实现(顺序图)中的消息
箭头指向对象的类将此消息
作为方法。**

保持类职责的一致性

- 从每个用例实现中发现类的职责，之后，要从系统整体角度定义类的职责
 - 当一个类中包含了互不相干职责，可以将此类分成2个或者以上不同的类；并且更新交互图
 - 当两个(或多个)不同的类有很相似的职责，合并这些类；并且重新整理出职责
 - 没有职责的类，可以考虑直接删除即可
 - 行为的更佳分配方案，可以返回之前的交互图并且重新并且采用新的方案重新分配职责-重新绘制顺序图

2. 定义分析类的属性

- 属性(Attribute)是类的已命名属性，用来存储对象的数据信息，是没有职责的原子事物
 - 属性名是一个名词，清楚地表达了属性保留的信息
 - 可利用文字详细说明属性中将要存储的相关信息
 - 属性类型应来自业务领域，与具体的编程语言无关

应该怎样定义类的属性呢？

- 从以下几个方面来定义属性：
 - 识别分析类的过程中，也可同时发现类的属性，包括：接在所有格后面的名词或形容词（即某某的属性）、**不能成为类的名词以及字段列表中所描述的数据需求**
 - 作为一般业务常识，是否有从**类职责范围**考虑所应包括的属性，例如发送消息“输入个人消息”给类实体类A，则要考虑类A可能包含属性：身份证号码
 - 该业务领域的专家意见以及
 - 以往所设计或看到的类似系统

例5， 银行的账户类应该包含的属性

设计如右图所示：

- 这是一般业务常识
- 领域专家也会给你几乎同样的建议
- 缺少了一个属性，则某个方法就无法实现
- 假如，如果缺少了属性balance则方法
 - **setBalance():**
 - **deposit(a: float): void**
 - **withdraw(a: float): void**
 - **transfer(a: float): void**都无法实现

Account
<ul style="list-style-type: none">-ownerName: String-acctNum: String-ownerIdNum: String-balance: float-password: String
<ul style="list-style-type: none">+getOwnerName(): String+getAcctNum(): String+getownerIdNum(): String+getBalance(): float+getPassword(): String+setBalance(): void+setPassword(): void+validateAcct(): boolean+deposit(a: float): void+withdraw(a: float): void+transfer(a: float): void

实例：为分析类添加属性

<<entity>>
路线
(from entity layer)

 代码
 名称: String
 天数: int





<<entity>>
旅游团
(from entity layer)

 出发日期: Date
 截止日期: Date
 可申请人数: int
 大人价格: Fee
 小孩价格: Fee

<<entity>>
申请
(from entity layer)

 申请编号:
 大人人数: int
 小孩人数: int
 申请状态:
 申请日期: Date

<<entity>>
支付明细
(from entity layer)

 金额: Fee
 截止日期: Date
 支付日期: Date
 支付状态:

<<entity>>
参加人
(from entity layer)

 姓名: String
 性别:
 生日: Date
 电话号码: String
 联系地址: String
 邮编: String
 email: String

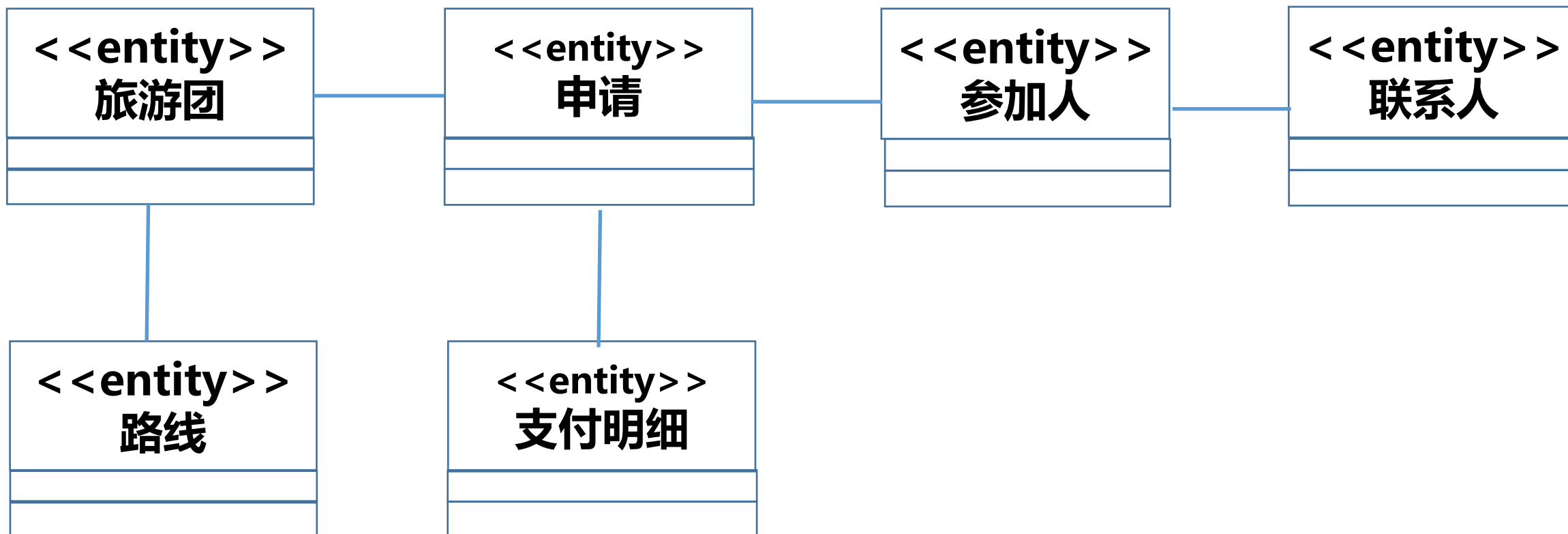
3. 定义分析类的关系

- **对象不能孤立地存在，它们之间需要频繁地通过消息进行交互从而执行有用的工作，并达到用例的目标**
- **为此，相应的类之间也应该存在特定的关系来支持这种交互过程**
 - **3.1 关联关系：协作关系**
 - **3.2 聚合关系：整体-部分**
 - **3.3 泛化关系：抽象-具体**

3.1 关联关系

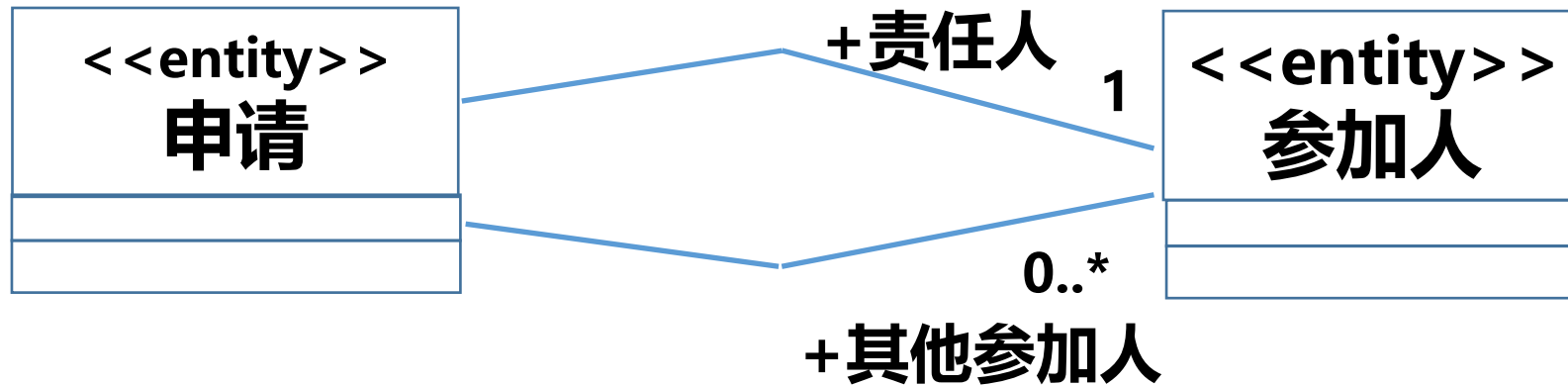
- 关联是类之间的一种结构化关系，是类之间的语义联系
 - 表明类的对象之间存在着链接
 - 对象是类的实例，而链接是关联的实例
- 识别关联的基本思路
 - 从交互模型中发现对象之间的链接，从而在相应的类上建立关联关系：如VOPC图中关联关系
 - 从业务领域出发，分析领域中所存在的实体类之间的语义联系，为那些存在语义联系的类之间建立关联关系：如实体类之间的各种语义联系

实例：实体类之间的关联关系



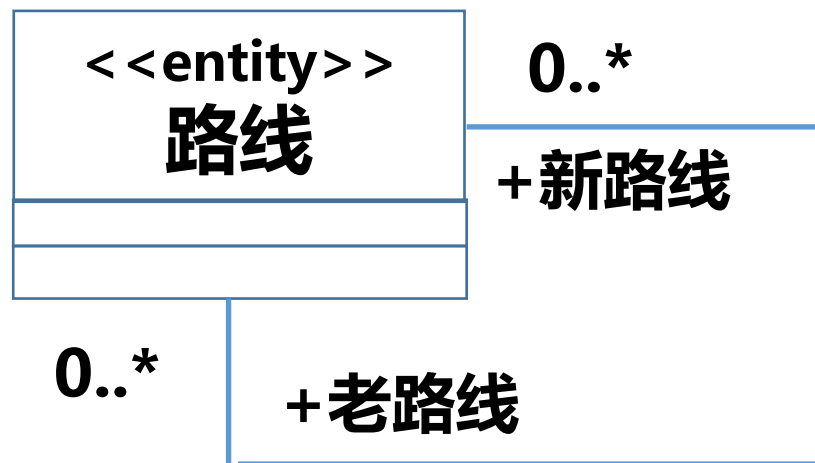
细化关联关系

- 关联具有：名称、端点和角色名、多重性
 - 名称：动词短语
 - 端点和端点名
 - 多重性表达式：*, 1..*, 1-40, 5, 3, 5, 8, ...



自反关联

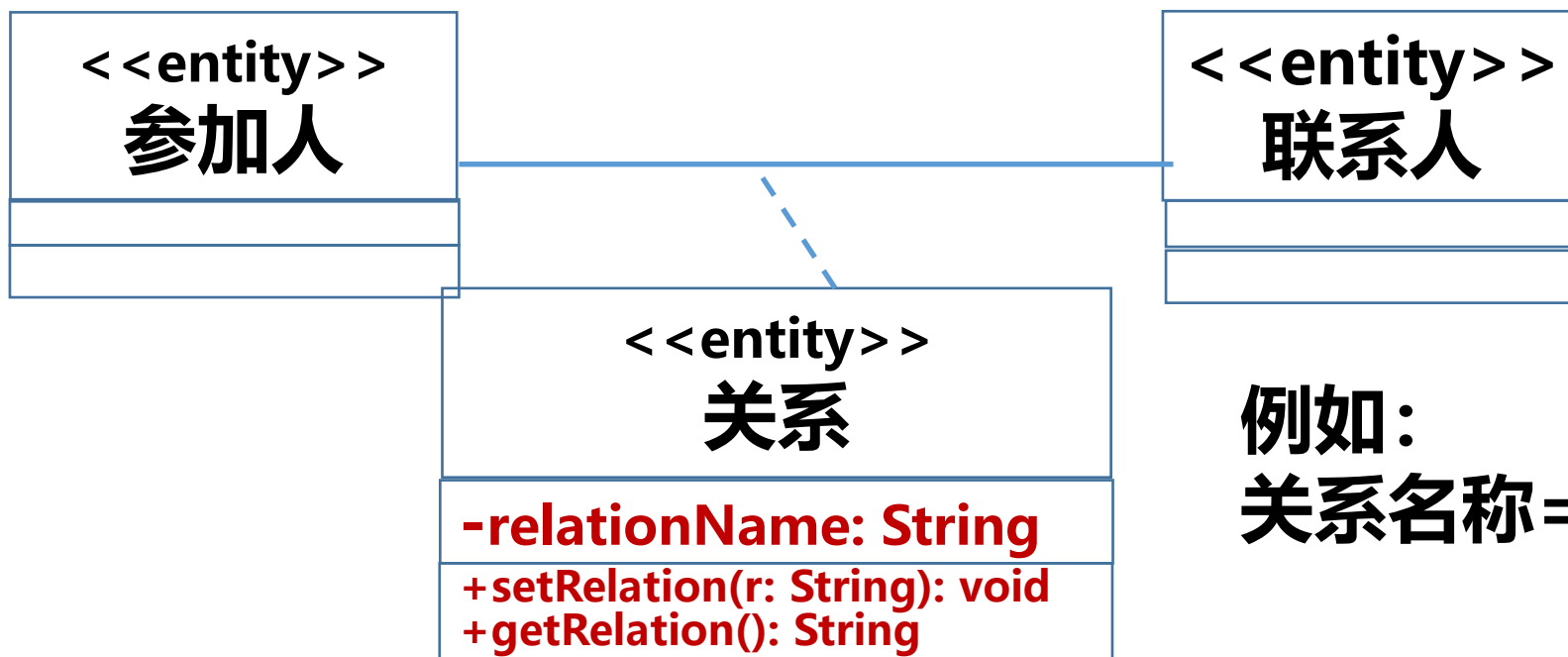
- 自反关联是指一个类自身之间存在关联，它表明同一个类的不同对象之间存在链接



关联类

- 关联类(Association Class)

- 是一种被附加到关联上的类，用来描述该关联自身所拥有的属性和行为
- 当某些属于**关联自身的特征信息**无法被附加到**关联两端的类**时，就需要为该关联定义**关联类**

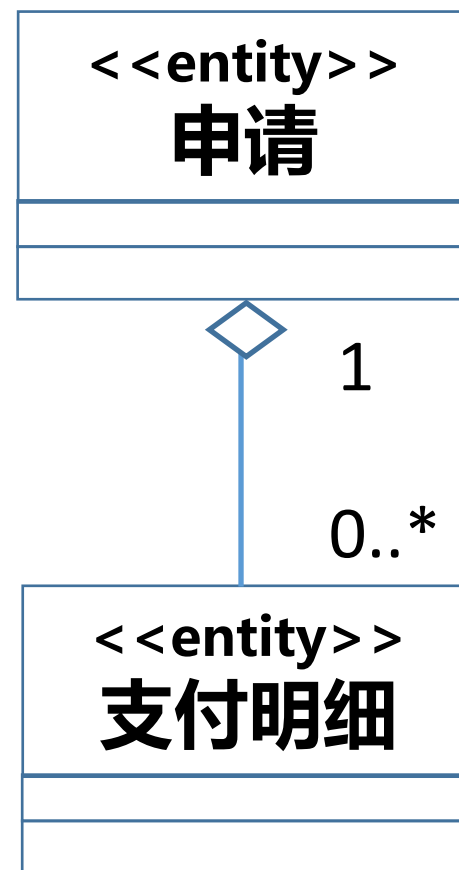


例如：
关系名称=母女

3.2 聚合关系

- 聚合(Aggregation)关系是一种特殊的关联关系
 - 除了拥有关联关系所有的基本特征之外
 - 两个关联的类还分别代表“整体”和“部分”，意味着整体包含部分
- 可以在已有的关联关系基础上，通过分析两个关联的类之间是否存在如何语义来识别聚合关系
 - A（整体）由B（部分）构成
 - B（部分）是A（整体）的一部分

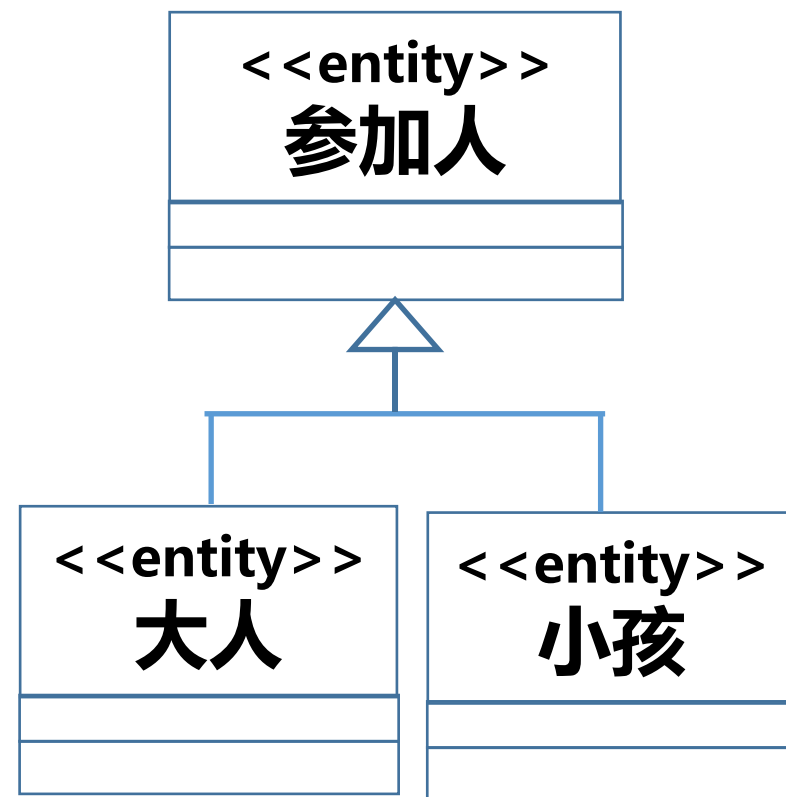
例子



3.3 泛化关系

- 泛化是指类间的结构关系、亲子关系
 - 子类继承父类所具有的属性、操作和关联
- 分析阶段的泛化关系主要来自与业务对象模型，针对实体类，结合业务领域的需求，从两个方面来提取泛化关系：
 - 是否有类似的结构和行为的类，从而可以抽取通用的结构和行为构成父类
 - 单个实体类是否存在一些不同类别的结构和行为，从而可以将这些不同类别的结构抽取出来构成不同的子类

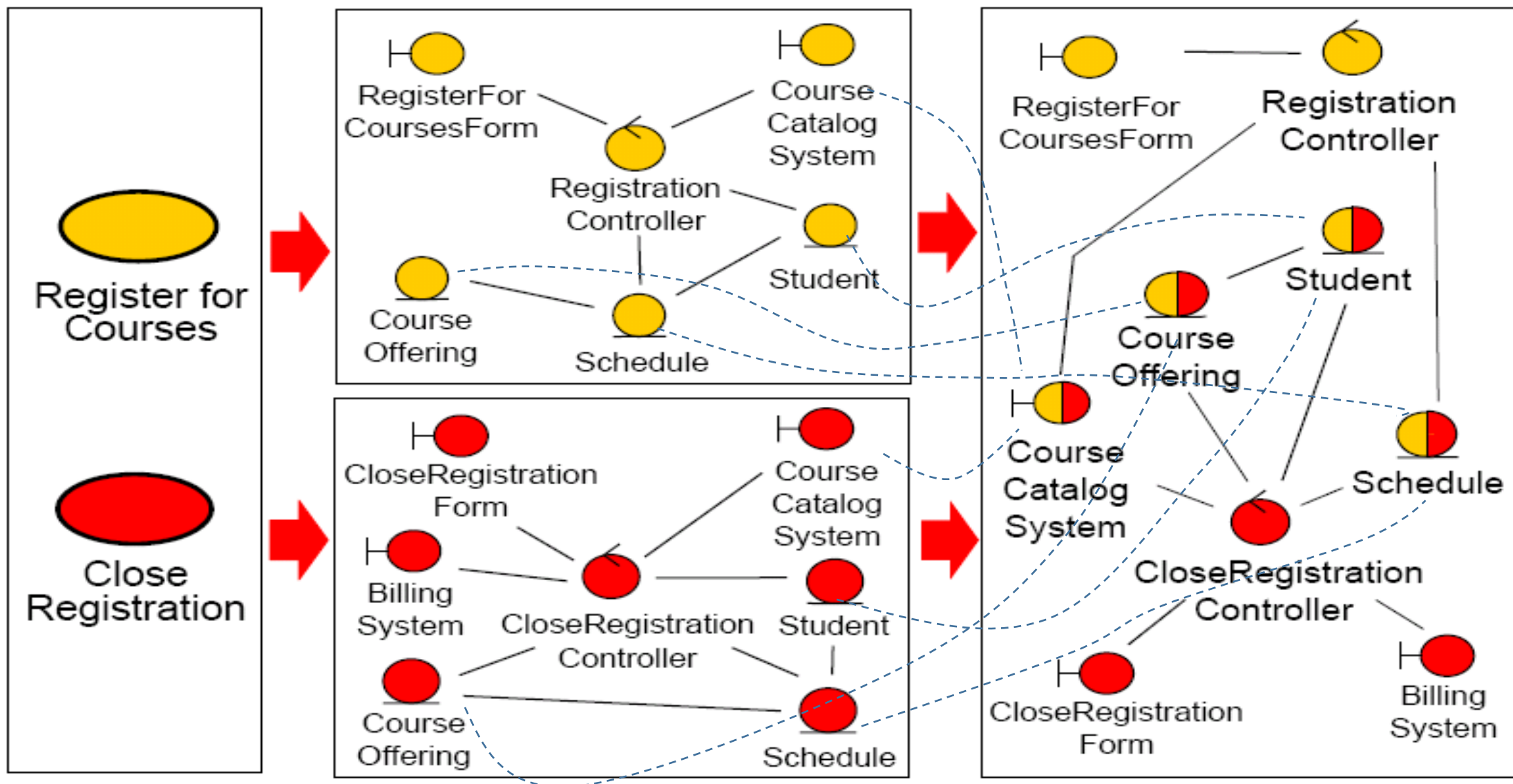
例子



4.统一分析类

- 类体现了系统的静态结构，通过分析类图体现软件静态结构
- 统一分析类的目的是确保每个分析类表示一个**单一的明确定义的概念，而不会职责重叠**
- 在分析工作完成之前，需要过滤分析类以确保创建最小数量的新概念

示例：统一分析类，删除各个用例的参与类类图的重叠部分

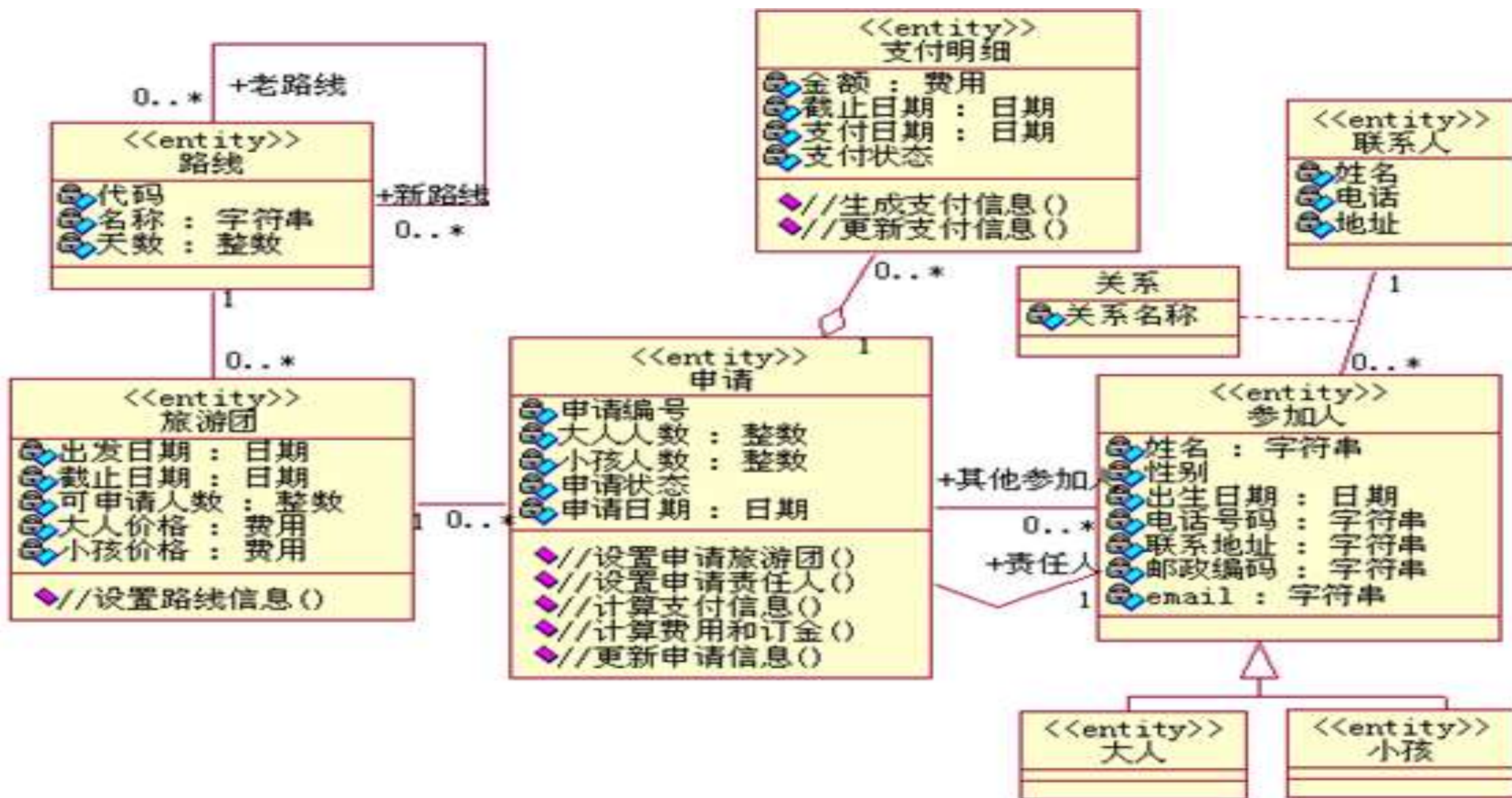


Student, CourseOffering, CourseCatalog System, Schedule 重复了，现在应该合并在一起。

再说分析类图

- **分析阶段的重点在于找出体现系统核心业务所需数据的实体类，而界面和业务逻辑细节分别由边界类和控制类隐藏**
- **在很多UML模型中，分析阶段的工作就是找到这些实体类**
- **这些实体类组成系统概念模型(分析类图)**
- **通过比较各个用例的VOPC图，删除重复的与那些没有引用的实体类，即可得到由实体类组成的分析类图，这些是分析的关键**

实例：旅游申请系统实体类类图



实例：旅游申请系统实体类类图

