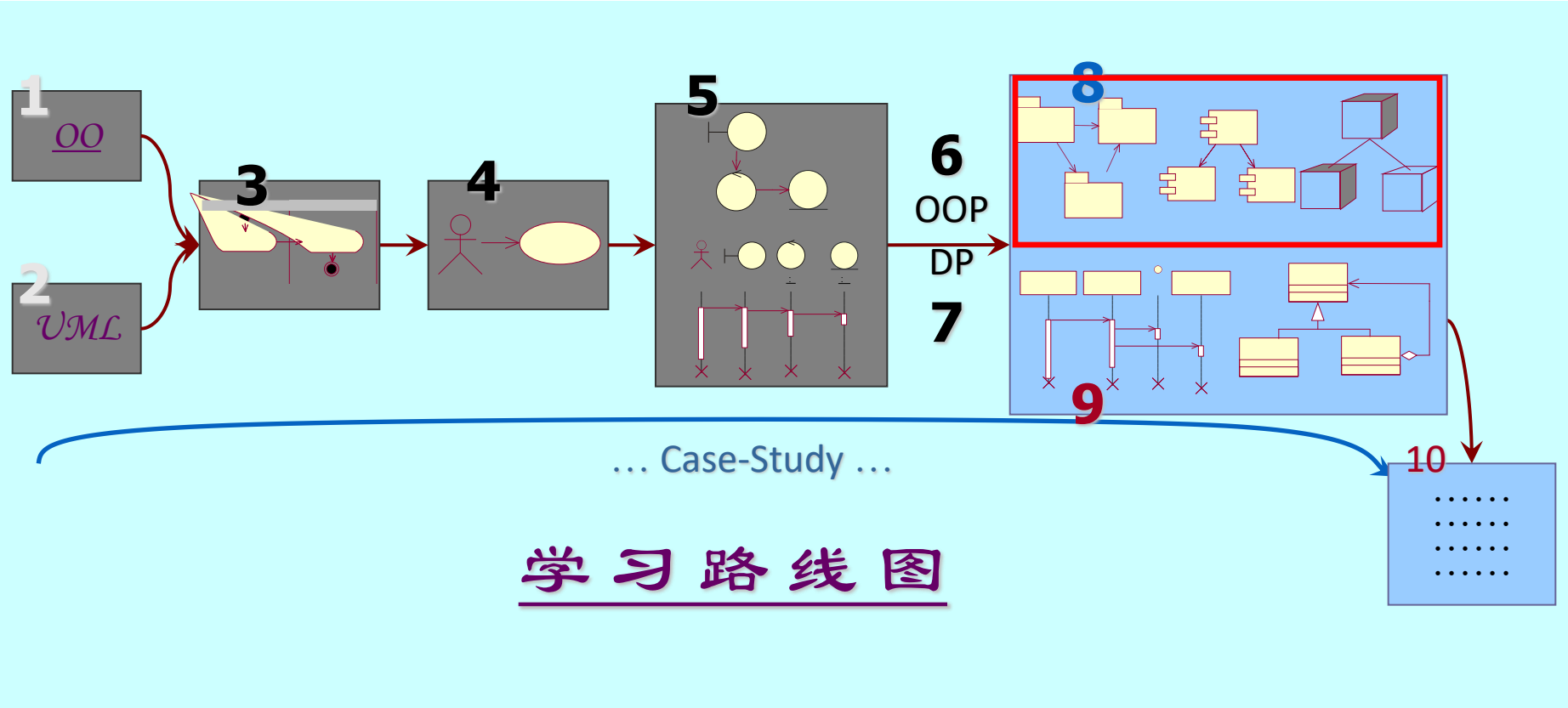
Lecture 14. Object-Oriented Design-Refine Architecture
面向对象的设计(架构设计)

Object Oriented Modeling Technology
面向对象建模技术

Professor: Yushan Sun

Fall 2022

学习路线图



内容安排

- 从分析过渡到设计
- 架构设计基础
- 确定设计元素
- 永久数据存储

从分析过渡到设计

软件设计

- **IEEE的定义**：设计是架构、构件、接口、以及系统其它特征定义的过程
- 软件设计（的结果）必须
  - 描述系统的架构（architecture）
    - 系统如何分解（decompose）和组织（organize）构件
  - 描述构件间的接口（interface）
  - 描述构件（component）：必须详细到可进一步构造的程度

软件设计知识域(Software Engineering Body of Knowledge) SWEBOK

3种不同的设计策略

- D-设计（分解设计，Decomposition design）
  - 将系统映射为构件片（component pieces），再对各个构件片进行内部设计；是应用广泛的设计策略。
- FP-设计（Family Pattern design）
  - 是一种探求一定范围的通用性设计策略，不从需求入手，而去探求问题的本质特征。主要用于通用产品设计，例如通用财务软件的设计。
- I-设计（Invention design）
  - 基于概念化原型作系统分析，定义系统以满足所发现的需要和需求

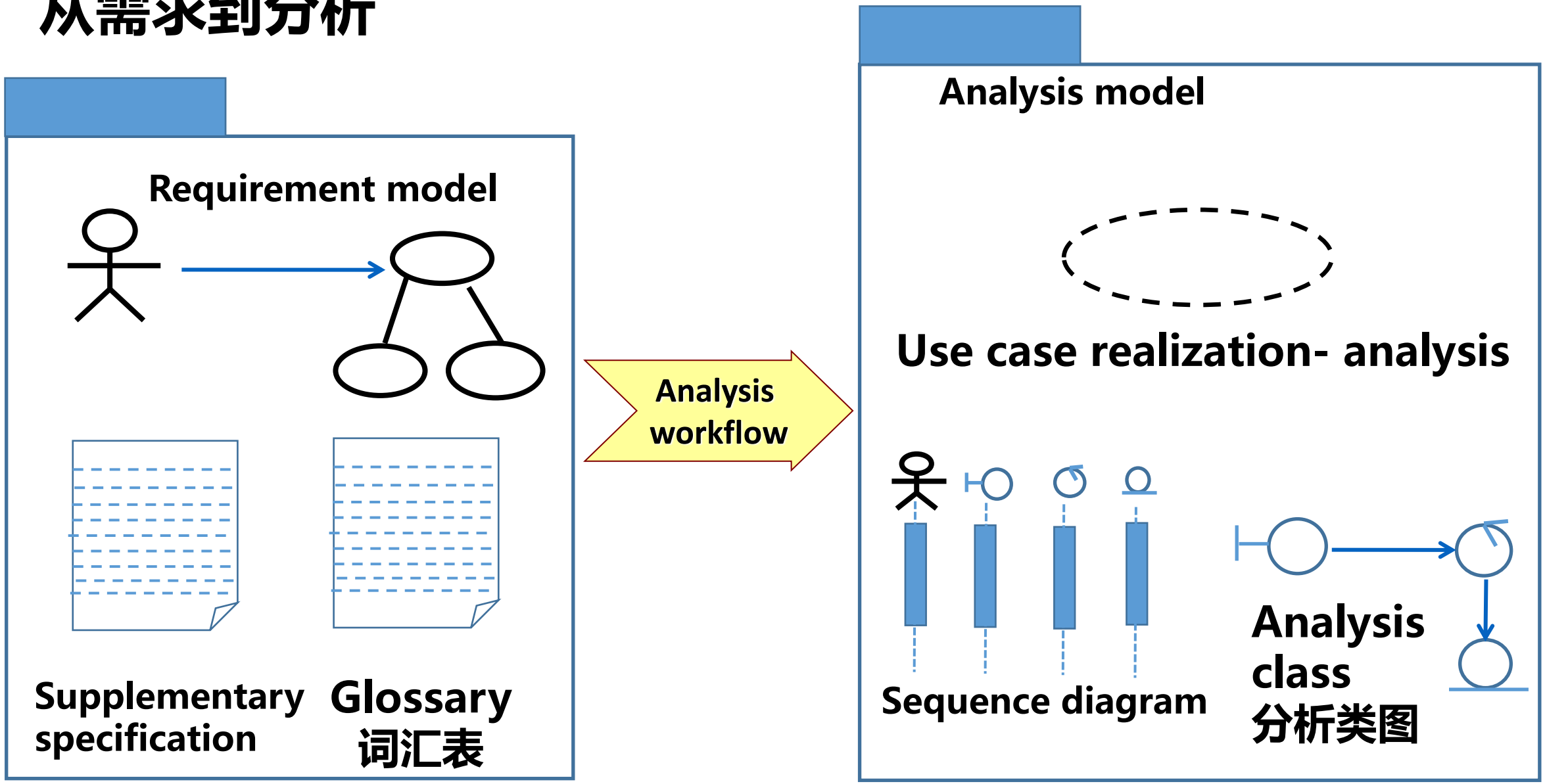
从分析到设计转变

- 面向对象分析是面向对象设计的输入，设计是分析的细化
  - 方法相同（都产生类图，分析类图，设计类图），但关注点不同
- **分析**：做什么(What)
  - 分析有效地确定了将要构建的内容
  - 分析重点关注业务(business)问题
- **设计**：怎么做(How)
  - 设计定义如何构建目标系统：采用何种技术、何种平台(如JavaEE，前后端分离开发)来实现分析模型
  - 设计关注于系统的技术(technical)和实现(implementation)细节

- **分析集中在系统需求，而**
- **设计集中在待生产的软件的结构．因此，在设计阶段，为了**
  - **软件的效率、**
  - **可扩展性等诸多方面**

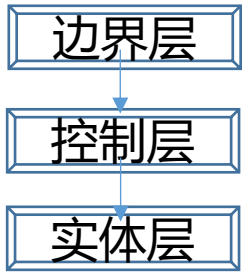
  的考虑，可能对分析阶段得到的类做修改。或者在设计的某一部分，使用软件设计模式。

从需求到分析

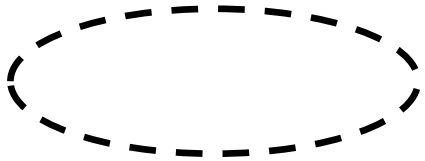

Requirement model
Supplementary specification
Glossary
词汇表
Analysis workflow
Analysis model
Use case realization- analysis
Sequence diagram
Analysis class
分析类图

从分析到设计

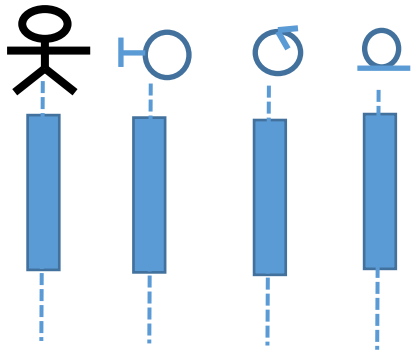
Analysis model (分析模型)



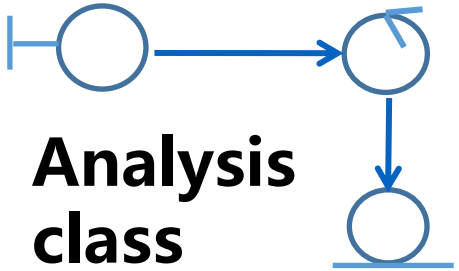
备选架构



Use case realization- analysis
分析阶段的用例实现



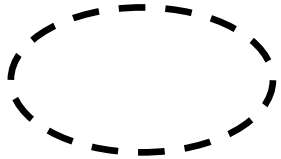
顺序图
Sequence diagram



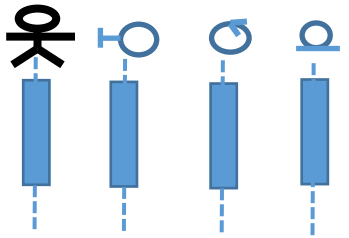
Analysis class
分析类图

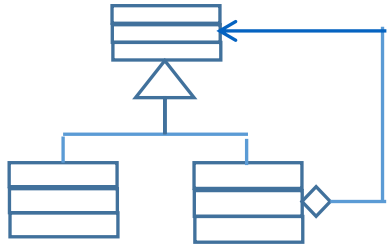
Design workflow

Design model (设计模型)

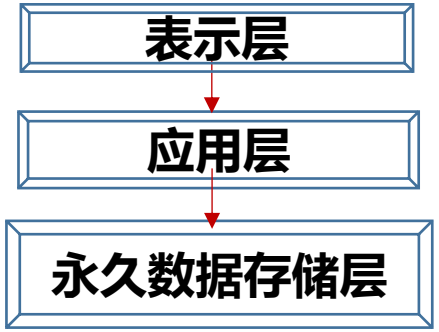


Use case realization-design
设计阶段的用例实现

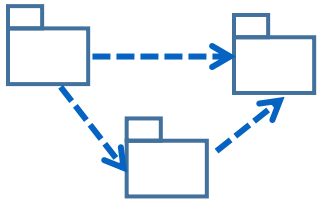




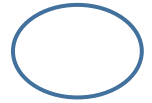
Design class
设计类图



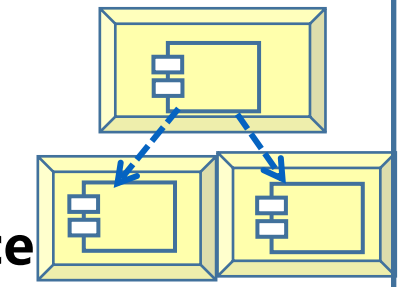
架构图



Subsystem
子系统



interface



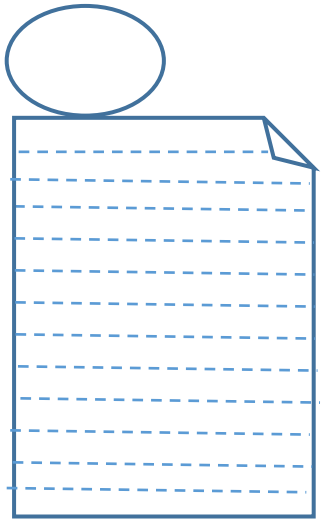
部署图

分析 VS. 设计

比较点	分析	设计
出发点	关注对业务问题的理解	关注解决方案的理解
关注点	侧重描述系统的功能需求	要全面考虑性能、可扩展性、可维护性等各类非功能需求
模型内容	一种理想化的设计，重点描述 • 系统的基本组成和 • 关键行为	要充分考虑操作、属性、对象生命周期等各个方面的问题
模型规模	一个较小的模型，只体现系统的核心元素	一个较大的模型，包括系统各个方面的细节

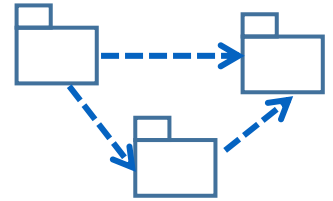
分析与设计

向上：也可以在分析结构上进一步抽象，通过提取接口或者相应的子系统，从而定义更高层的设计元素。

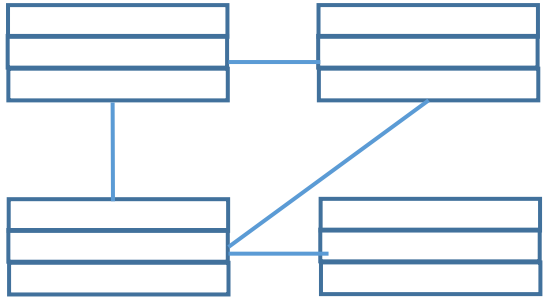


(可以认为：分析定义了一个中间层)

抽象



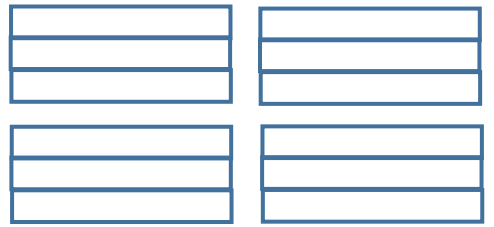
子系统



分析模型

具体化

向下：修改分析类，定义可以直接被实现的设计类



设计类

分析过程

设计过程

从分析模型到设计模型

- OO设计是对OO分析的进一步细化，其根本思想是：对分析模型中的分析类进行进一步的设计，添加实现细节，这些分析类最终转变成设计元素
- 面向对象的方法中，设计是分析的自然延续，在分析模型中添加特定的实现机制，得到可以实现的设计元素
    - 设计类(Design Classes)
    - 子系统(Subsystem)
    - 接口(Interface)
    - 主动类(Active Class)
- 设计过程会直接覆盖分析模型的成果。

保留分析模型

- 迭代开发中，保留分析模型是必要的
  - 分析模型可以保持从需求、分析到设计的可跟踪性
  - 下一迭代的分析也需要前一迭代的分析模型
  - 分析模型提供了系统核心业务场景，对于理解大规模系统的核心机制有非常重要的意义
- 需要采取一些手段来保留系统的分析模型
  - 在某个点冻结分析模型，保留一份历史拷贝；但这可能存在模型不一致性问题
  - 同时维护两个独立的分析模型和设计模型；但增加维护模型的成本

Return

架构设计基础

架构和架构设计

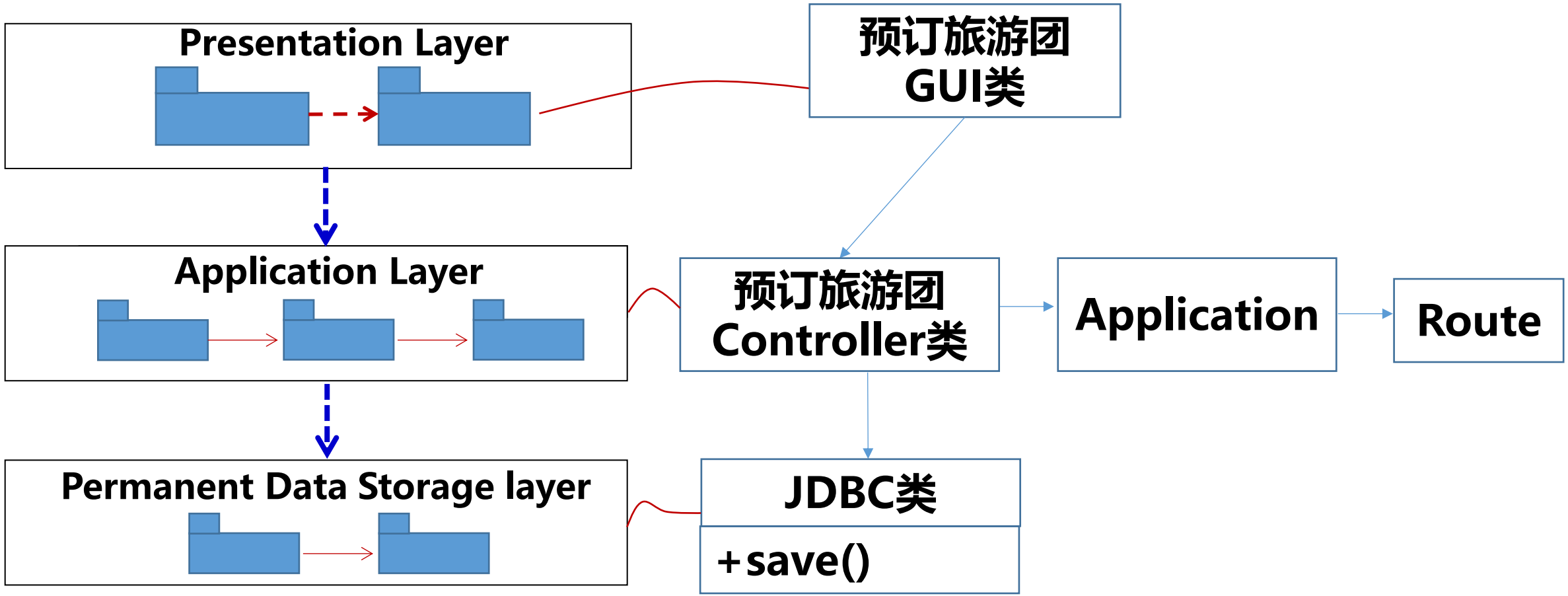
- 架构是一个系统的组织结构，包括系统分解成的各个部分、它们的连接性、交互机制和指导系统设计的相关规则
  - 架构设计的活动在分析阶段就已经开始，分析阶段主要关注基础架构的选型和并确定核心的分析机制
  - 设计阶段，要针对分析阶段的备选架构的各个方面进行详细的定义，以设计出符合特定系统的架构

架构设计的主要工作

- 架构设计内容
  - **确定核心元素**：在架构的中高层，以“分析类”为出发点，确定相应的“核心**设计元素**”（设计类，接口，子系统等）
  - **优化组织结构**：按照高内聚、低耦合的基本原则，整理并逐渐充实架构的层次和内容（在架构的哪些层，具体地包含哪些包，在哪些包中，包含哪些类）
  - **定义设计后的组织结构**：架构设计还应该考虑设计完成后系统**实现、运行以及部署**等阶段的组织结构（JavaEE, 前后端分离开发，服务器，数据存储，软件部署）

UML和架构设计

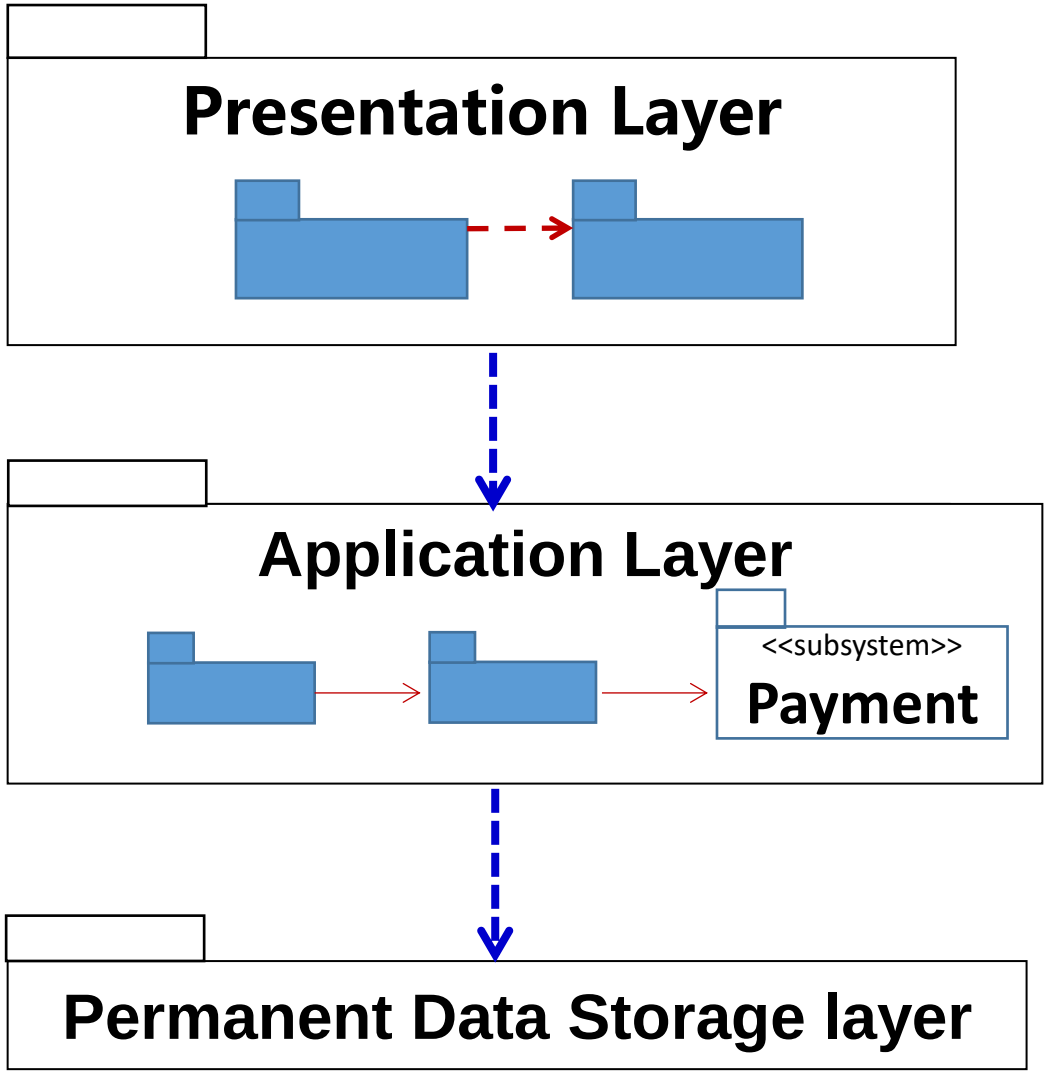
- **架构的全部内容就是复杂性管理**：将解决方案划分成多个小的组成部分，再将这些小的部分结合起来，构成更大的、更加一致的结构



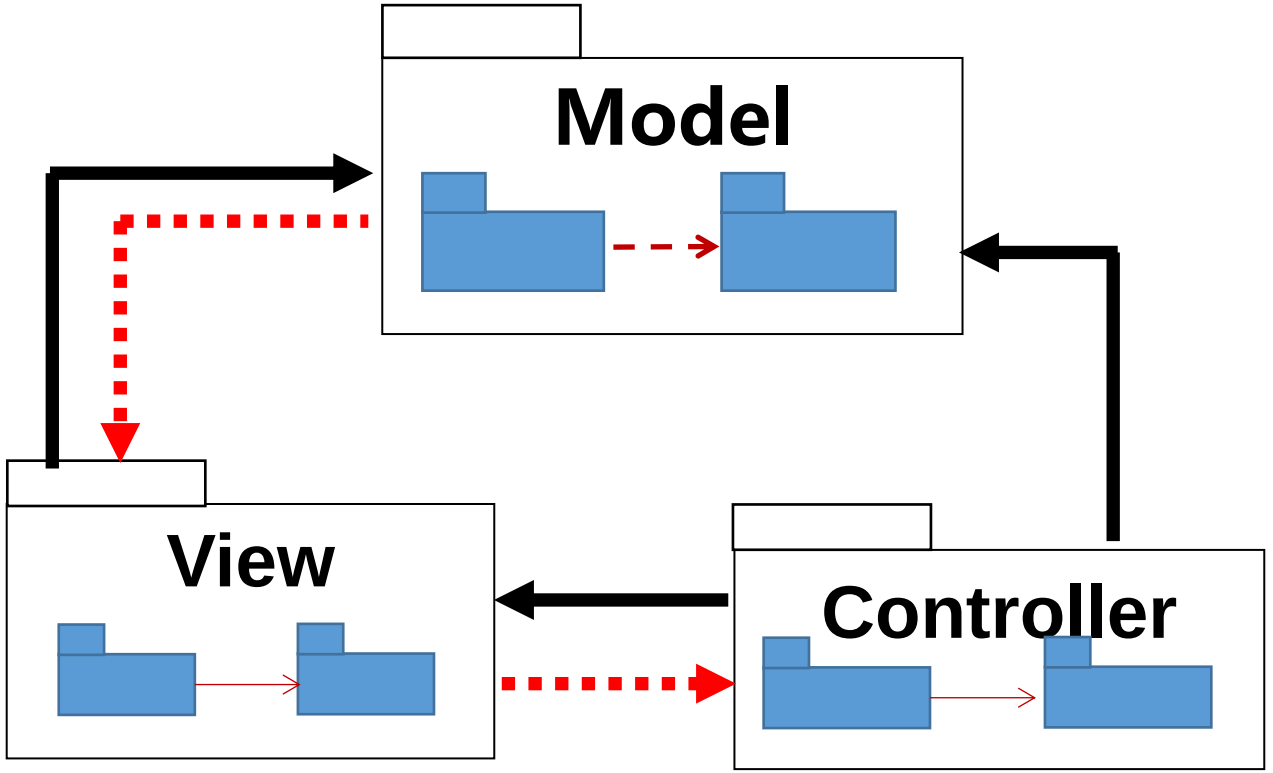
- **包图的概念：**
- 包图是一种将模型元素分组的机制
- 包主要用于
  - 组织模型元素
  - 配置管理单元

- **分包的原则**：职责相似，将一组职责相似、但以不同方式实现的类归为一组有意义的包；如java类库中的javax.swing.border
- **协作关系**：包图包含了各种不同类型的类，它们之间通过相互协作实现一个意义重大的职责

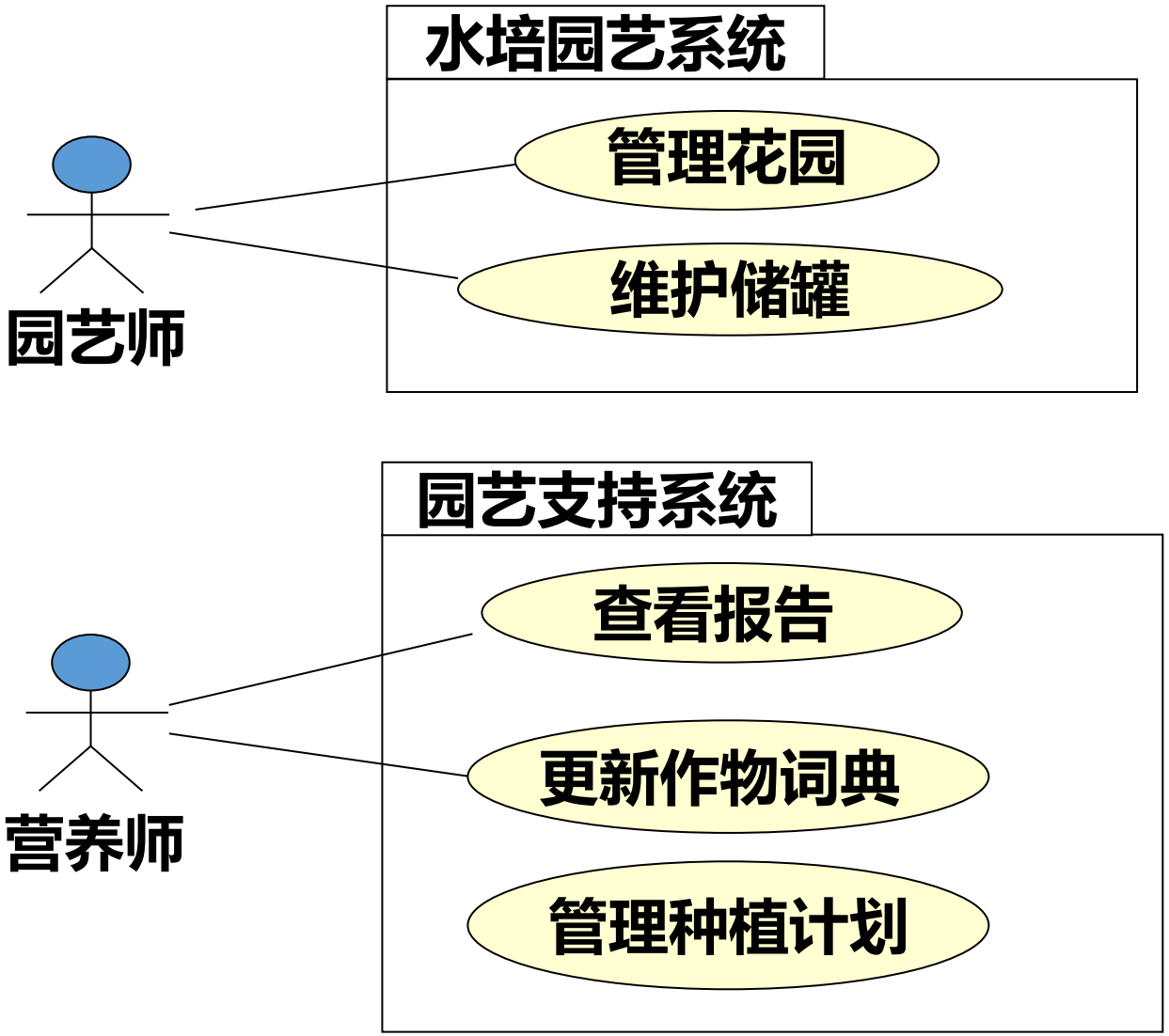
- 可以使用包记号表示大的系统架构与局部的结构
- 表示架构，可以
  - 依据架构层
  - 依据用户功能



三层层次架构



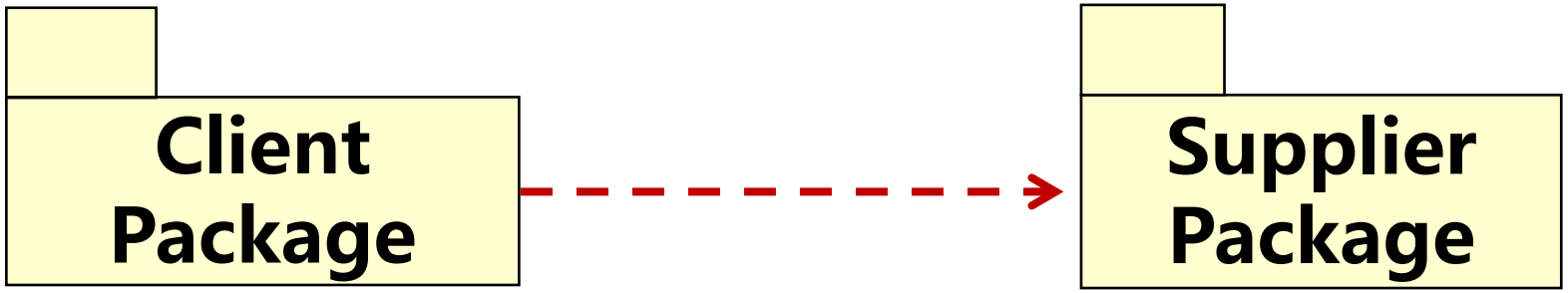
MVC架构



依据用户功能组织包

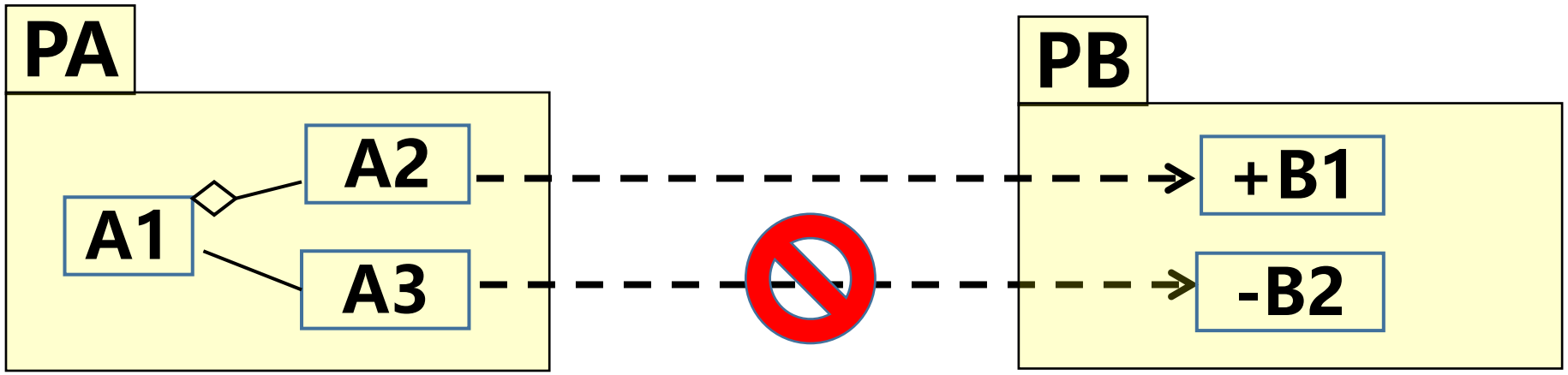
依赖关系

- 包之间存在着依赖关系



- 上图说明Client包依赖于Supplier包：
  - ✓ Supplier包的改变将影响到Client包
  - ✓ Client包不能够独立的复用，因为它依赖于Supplier包

定义依赖关系-包元素的可见性

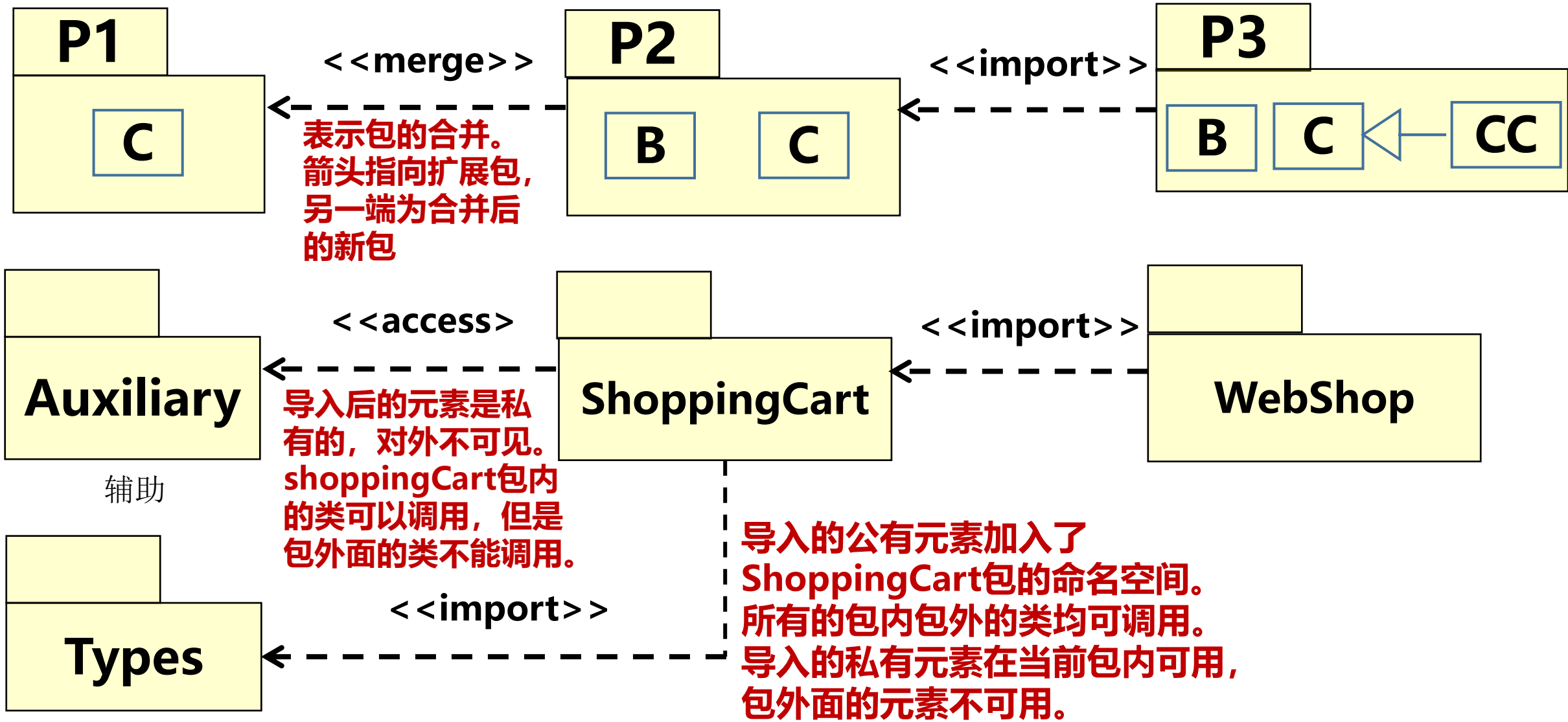


B1是公有类：包内外的元素都可以访问
B2是私有类：只有包PB内的元素可以访问

高级依赖关系

- 合并关系（merge）
  - 定义一个包的内容如何被另一个包扩展
- 包导入关系允许一个包可以不需要通过包名直接访问被导入包中的元素
  - 公有导入（导入关系（import））
    - 导入后的元素在当前包内的可见性不变
  - 私有导入（访问关系（access））
    - 导入后的元素是私有的，对外不可见
    - 等同于Java中的import关键词

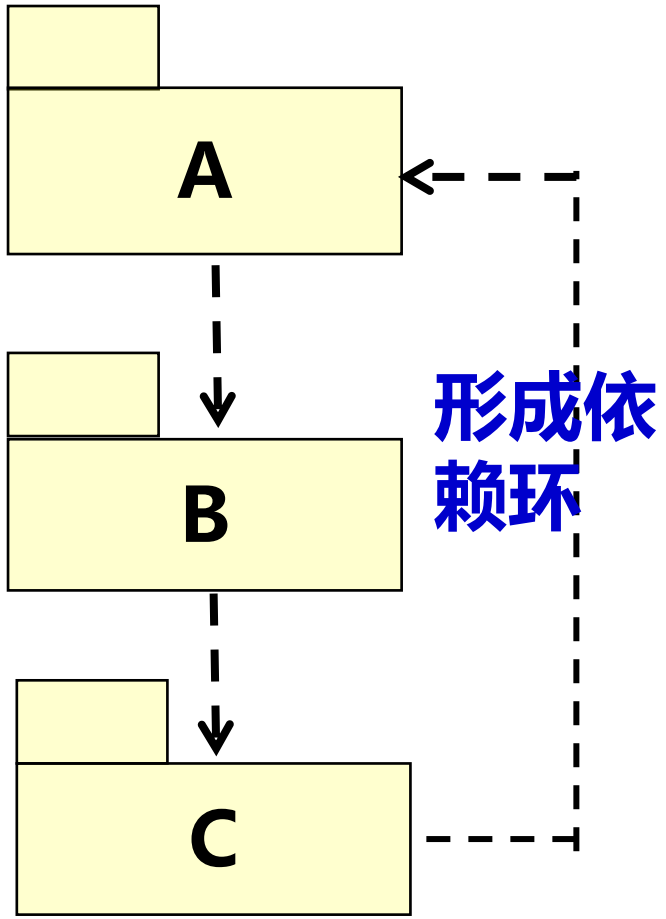
示例：高级依赖关系


P1
C
<<merge>>
P2
B
C
表示包的合并。
箭头指向扩展包，
另一端为合并后
的新包
<<import>>
P3
B
C
CC
Auxiliary
辅助
<<access>>
ShoppingCart
导入后的元素是私
有的，对外不可见。
shoppingCart包内
的类可以调用，但是
包外面的类不能调用。
<<import>>
WebShop
Types
<<import>>
导入的公有元素加入了
ShoppingCart包的命名空间。
所有的包内包外的类均可调用。
导入的私有元素在当前包内可用，
包外面的元素不可用。

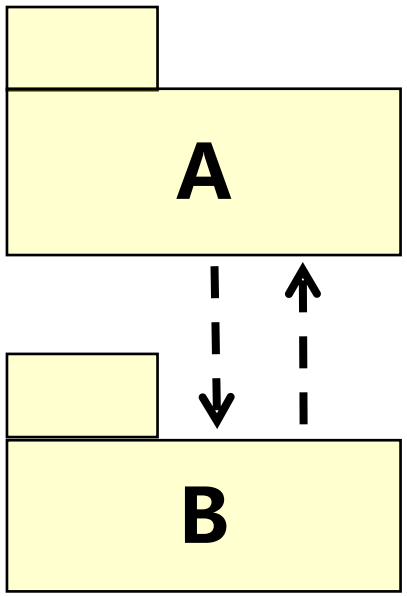
包设计原则

- 包内聚性原则
  - **复用发布等价原则**（REP）：包内的公有元素要么都是可复用的，要么都是不可复用的
  - **共同复用原则**（CRP）：包中所有的类应该是共同复用的；如果复用了一个类，就要复用包中所有的类
  - **共同封闭原则**（CCP）：一个变化会对一些类产生影响，而不会对其它类产生影响；将那些受影响的类放在一个包里面。
- 包耦合性原则
  - **无环依赖原则（ADP）**：包图中的依赖关系不允许存在环
  - **稳定依赖原则（SDP）**：朝着稳定的方向依赖，如果一个包被许多包所依赖，那么它就是稳定的
  - **稳定抽象原则（SAP）**：稳定的包应该包含抽象类(接口)，以便支持扩展（因此稳定）

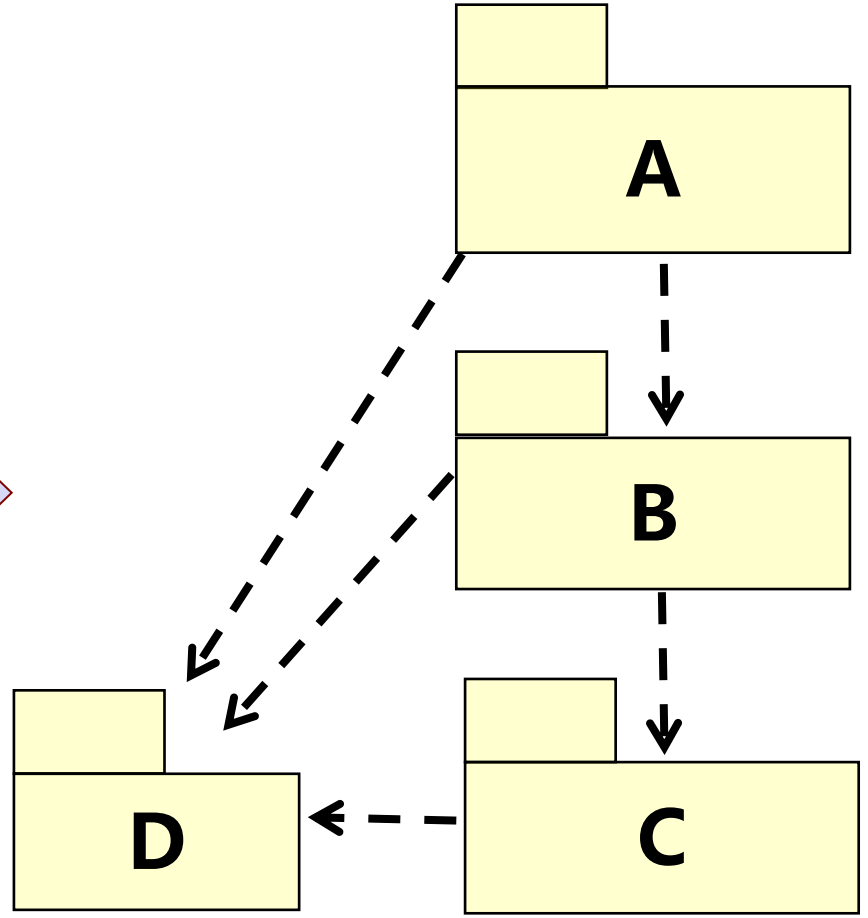
包设计原则：无环依赖原则



形成依赖环



形成依赖环



无依赖环

依赖环使得任何一个包都不能独立的重用，修改任何一个包都会引起所有的包的变化。

Return

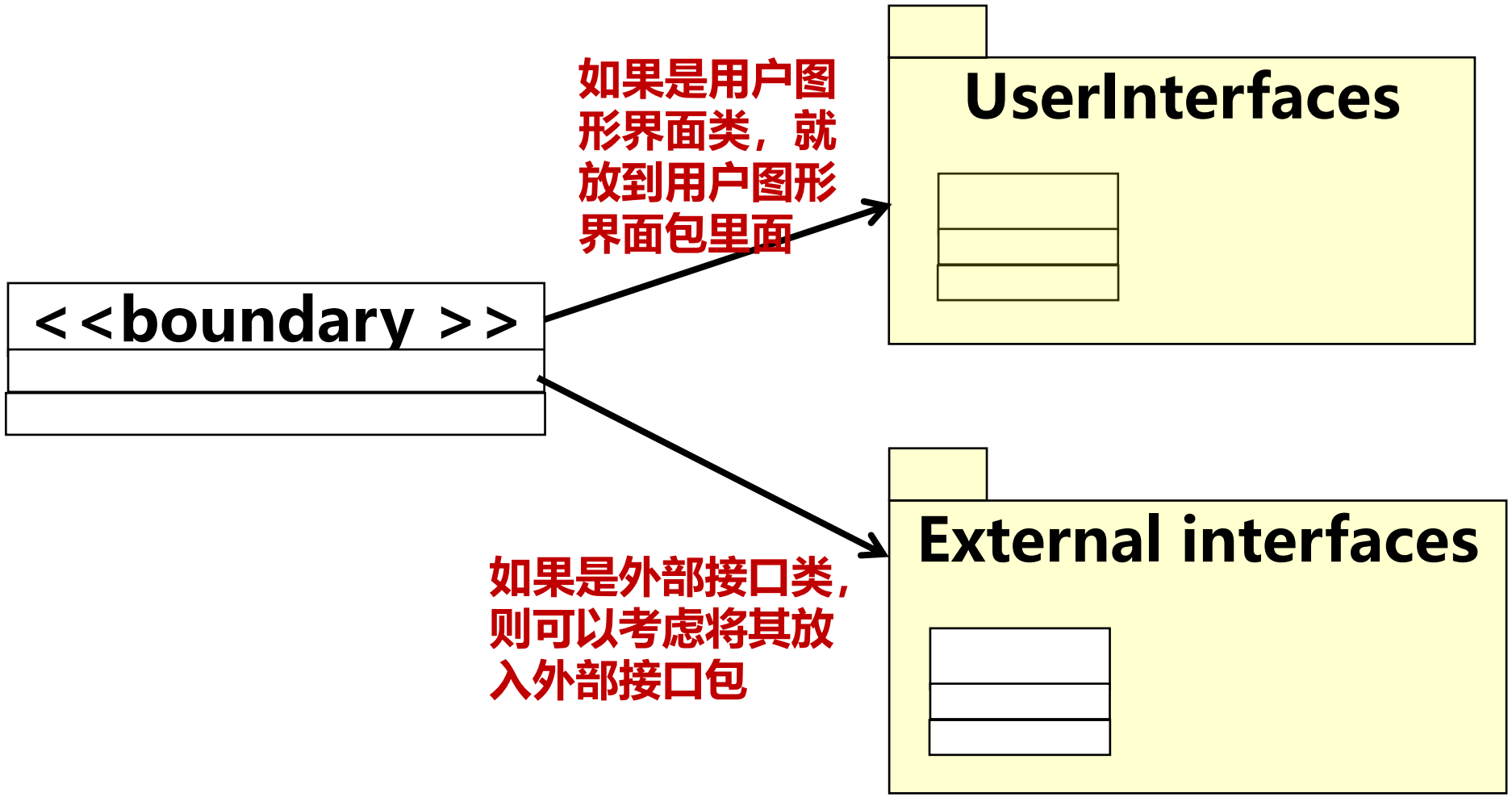
确定设计元素

设计元素

- 设计元素(Design Elements)是指能够直接用于实现(编码)的模型要素
  - 包(Package)
  - 设计类(Design Classes)
  - 子系统(Subsystem)
  - 接口(Interface)
  - 主动类(Active Class)，例如用户图形界面类
- 确定设计元素的目的是改进(调整)分析类，使之成为适当的设计模型元素

Analysis classes

Design elements



组织设计类

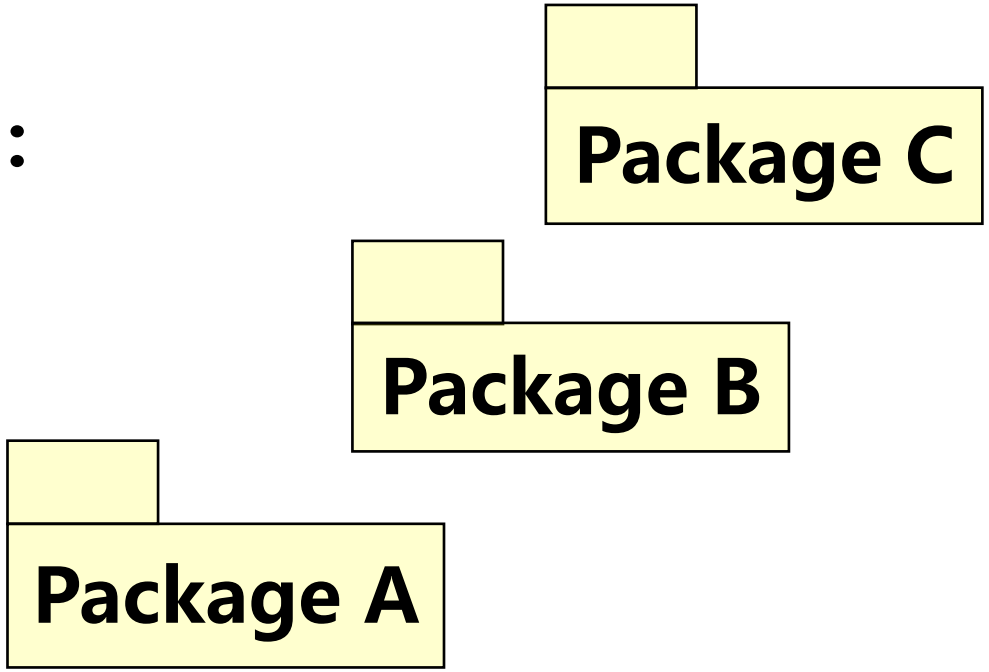
- 简单的分析类被直接映射到设计类，如果：
  - 若一个分析类是一个简单类（表示一个简单逻辑抽象）
  - 则将此分析类增加详细内容（属性，方法的参数，返回值），从而成为设计类
- 更复杂的分析类可能
  - 被分成多个设计类
  - 或者设计为一个包
  - 或设计为一个接口和子系统

分析类到设计元素的映射

- 一个分析类，在设计阶段，可能成为
  - 一个简单的设计类
  - 一个设计类的一部分（被某个其它的类聚合）
  - 一个聚合类（改为一个类聚合另外一个类的情况）
  - 同一个类继承而来的一组类（shape, circle, rectangle三个单独的类可以设计成层次类）
  - 一组功能相关的类(如，一个包)
  - 一个子系统
- 分析类间的一个关系可能成为设计中的一个类(关联类)

打包设计类

- 在分析阶段利用B-C-E的备选架构对分析类进行分组，而设计时，由于大量设计元素的引入，因此需要定义更合理的分组(封装)机制
- 封装标准可以基于多种不同的因素：
  - 配置单元
  - 开发团队中的资源分配
  - 反映用户类型
  - 表示已有产品和服务

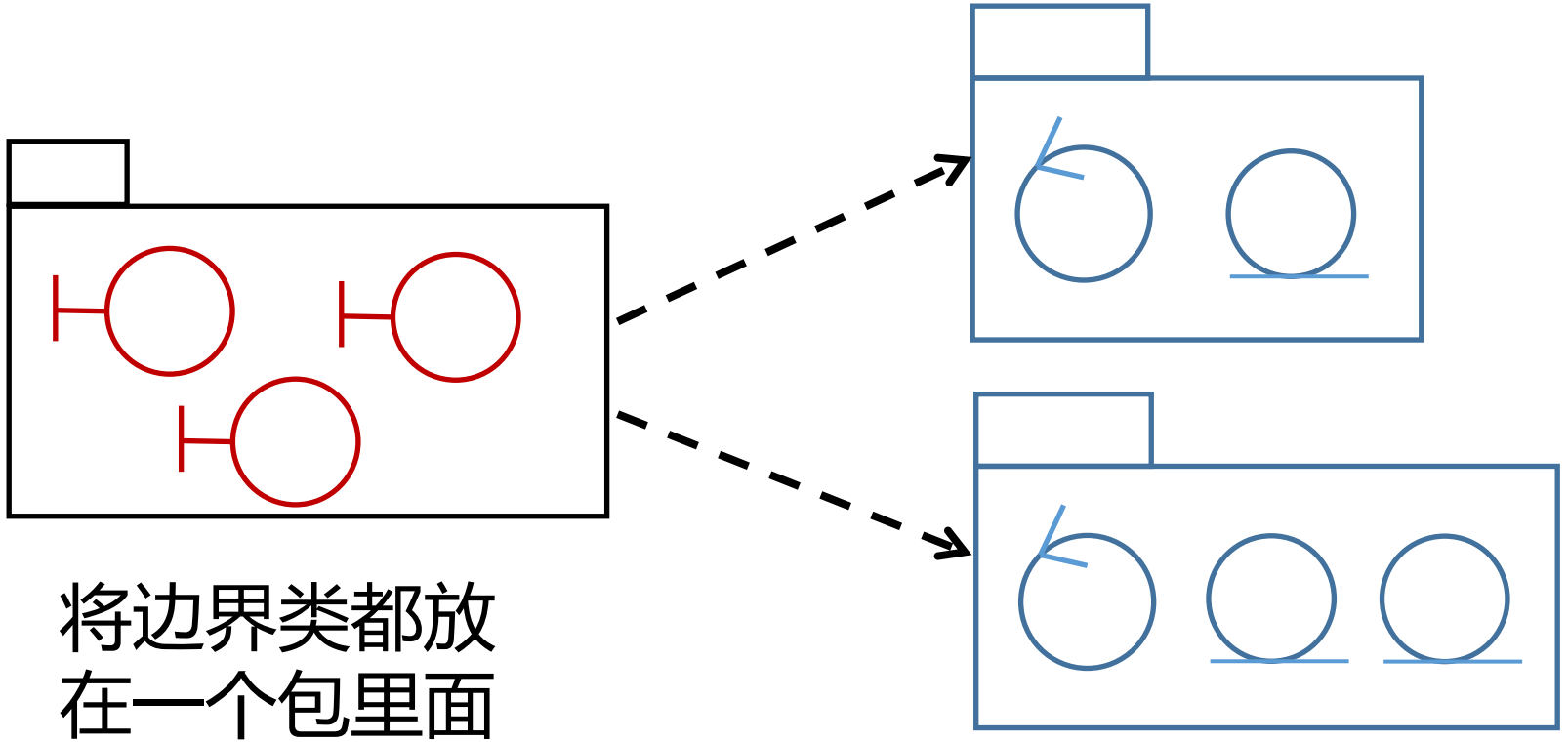


封装技巧：功能相关的类

- 确定两个类在功能上是否相关的线索：
  - 如果某个边界类的功能是显示一个特定的实体类，它就可能在功能上与该实体类相关
  - 如果两个类与同一个参与者进行交互，或受到对同一个参与者更改的影响，则这两个类可能相关
  - 若类A的行为或结构的变化导致另一个类B也必须相应地变化，则A和B类相关
  - 如果一个类A的删除影响其它类B，则A和B类相关
  - 如果两个对象存在消息交互，则这两个对象所对应的类相关
  - 两个类之间存在某些关系，则这两个类相关
  - 一个类A创建另一个类B的实例，则这两个类相关

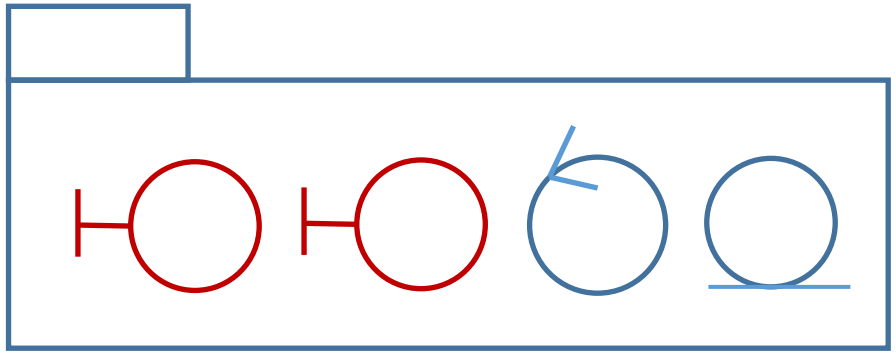
封装技巧：边界类(1)

- 如果系统边界(用户界面、系统接口)可能进行相当大的更改
  - 则边界类应被单独地放置在一个（或者几个）单独的包里面
  - 目的：使得边界类的更改不影响其它的控制器类和实体类



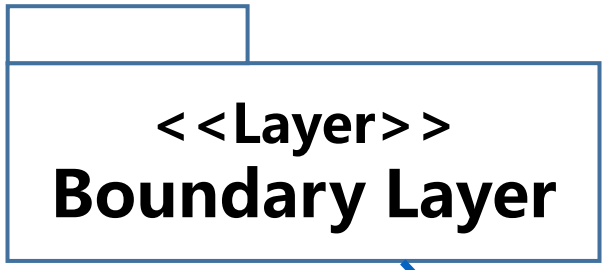
封装技巧：边界类(2)

- 如果系统边界(用户界面、 系统接口)不太可能进行大的更改
  - 则**可以**将边界类和在功能上与它们相关的类打包到一起

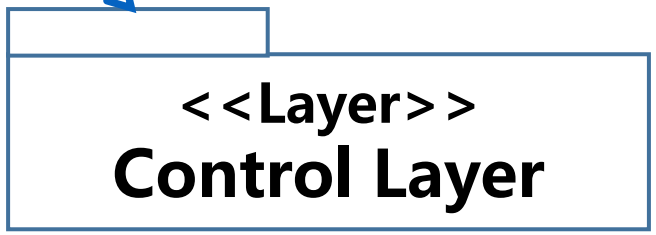


实例：设计阶段旅游申请系统分包考虑

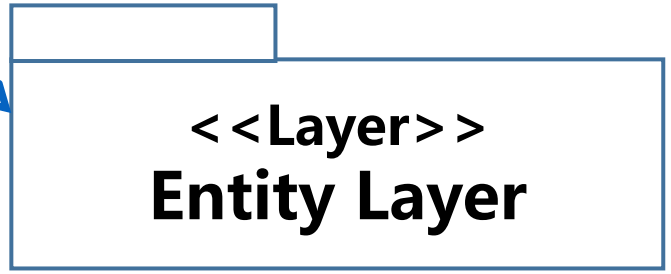
分析阶段产生的B-C-E三层备选架构抄写如下：



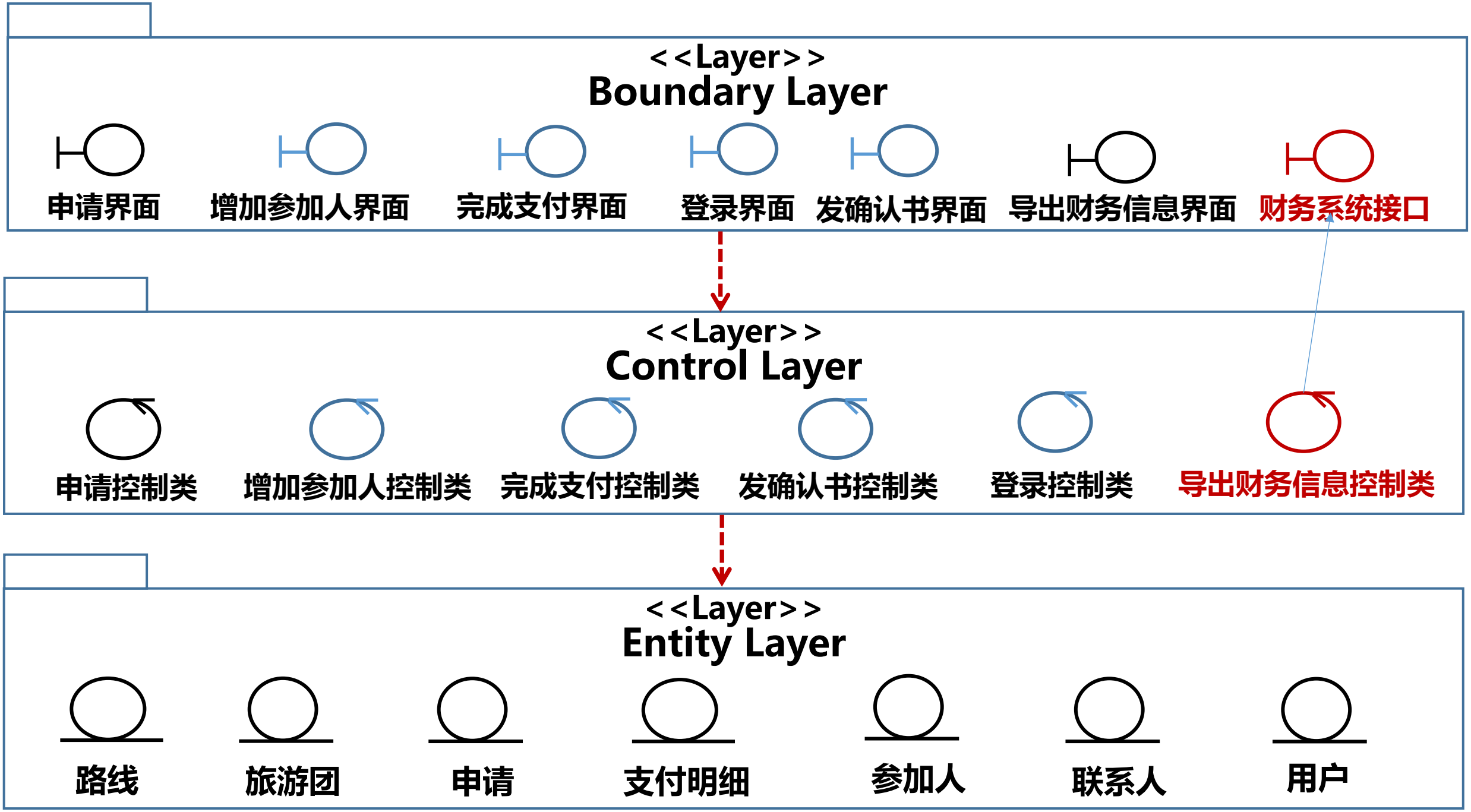
边界层：负责与参与者和外部设备进行交互



控制层：负责协调用例内部的处理流程

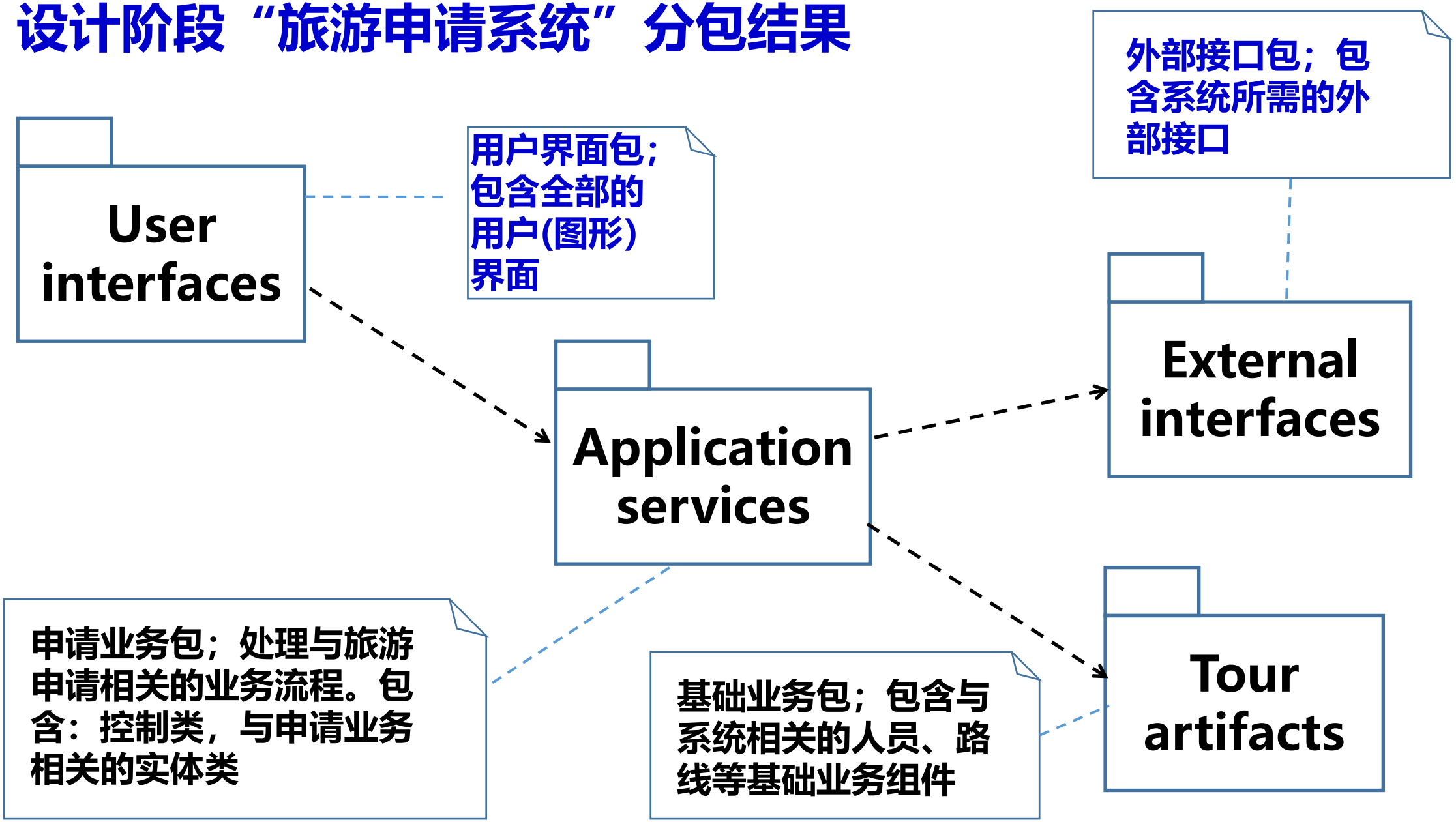


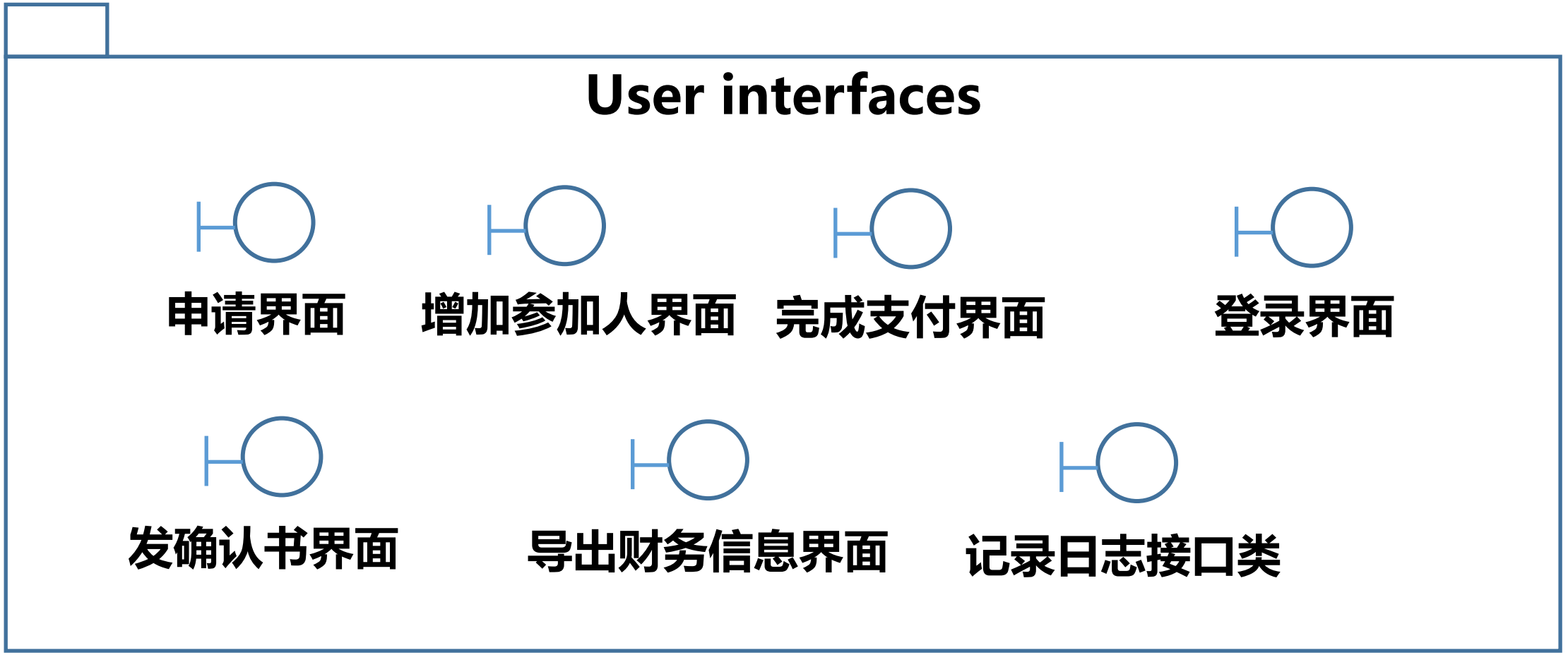
实体层：负责存储和访问系统内部实体信息。

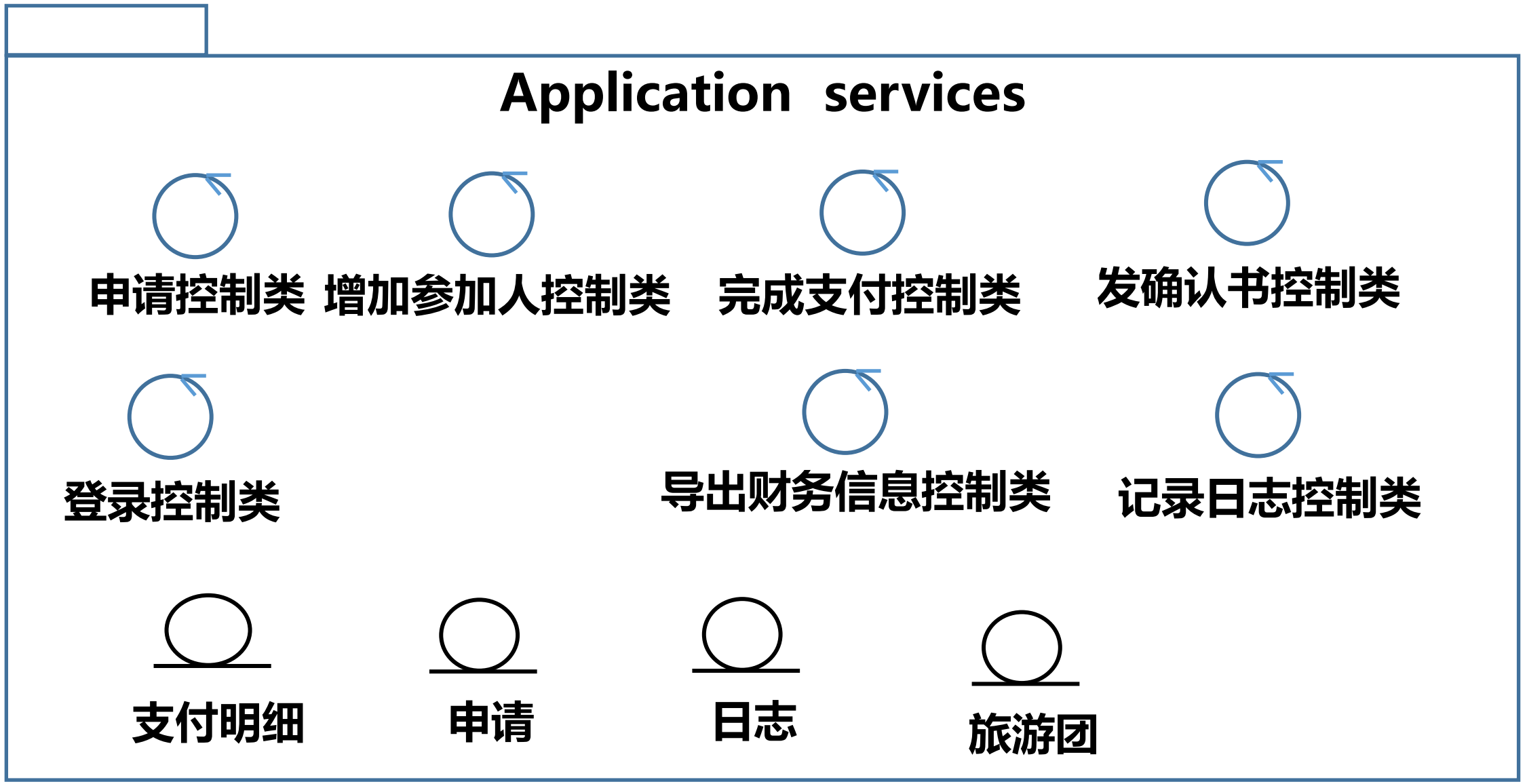

<<Layer>>
Boundary Layer
申请界面
增加参加人界面
完成支付界面
登录界面
发确认书界面
导出财务信息界面
财务系统接口
<<Layer>>
Control Layer
申请控制类
增加参加人控制类
完成支付控制类
发确认书控制类
登录控制类
导出财务信息控制类
<<Layer>>
Entity Layer
路线
旅游团
申请
支付明细
参加人
联系人
用户

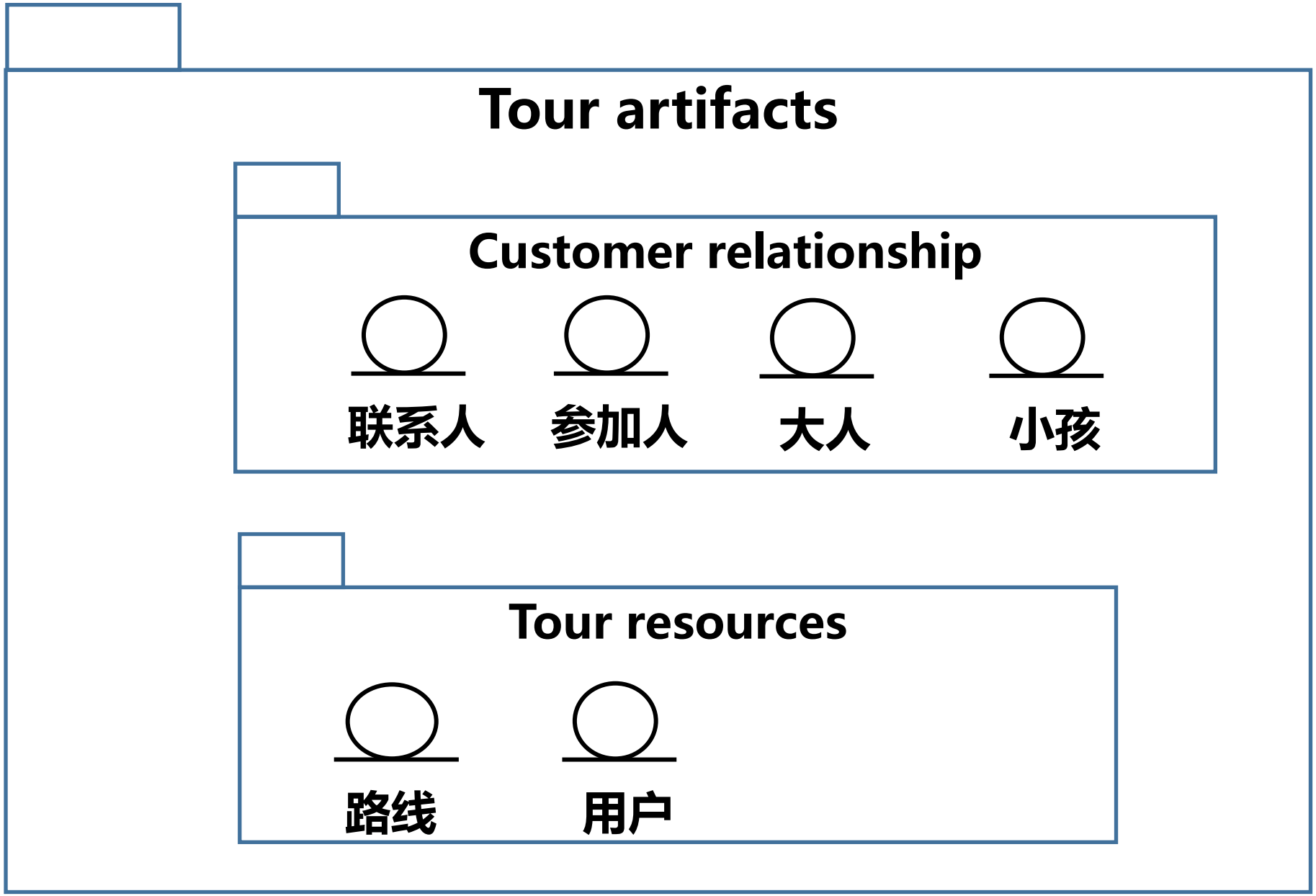
- 考虑的几个要素
  - 注意到“导出财务信息控制类“需要访问“财务系统接口”类，因此产生了边界包和控制包之间的依赖环。
  - 为消除依赖环，将边界包中的接口类独立出来，建立新的外部接口包(External Interfaces)，剩余的用户界面保留为单独的界面包(User Interface)
  - 控制类和申请业务相关的实体类打包为申请业务包（Application Services），负责与前端进行交互，并处理与申请相关的业务
  - 与参与者相关业务可以考虑和其它系统的复用(如与CRM系统)，与路线管理相关的业务也存在一定的复用性，这些均可放在单独的包中，作为该旅游企业基础的业务服务单元(Tour Artifacts)

设计阶段“旅游申请系统”分包结果

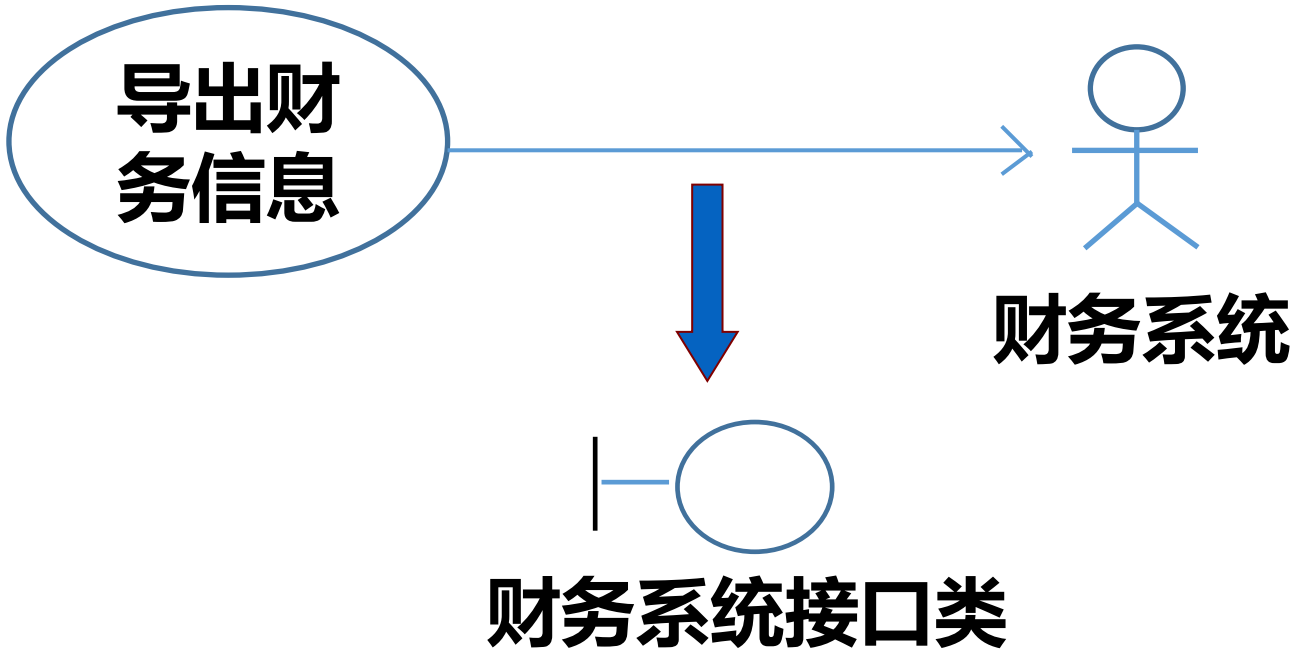



User interfaces
申请界面
增加参加人界面
完成支付界面
登录界面
发确认书界面
导出财务信息界面
记录日志接口类


Application services
申请控制类
增加参加人控制类
完成支付控制类
发确认书控制类
登录控制类
导出财务信息控制类
记录日志控制类
支付明细
申请
日志
旅游团


Tour artifacts
Customer relationship
联系人
参加人
大人
小孩
Tour resources
路线
用户

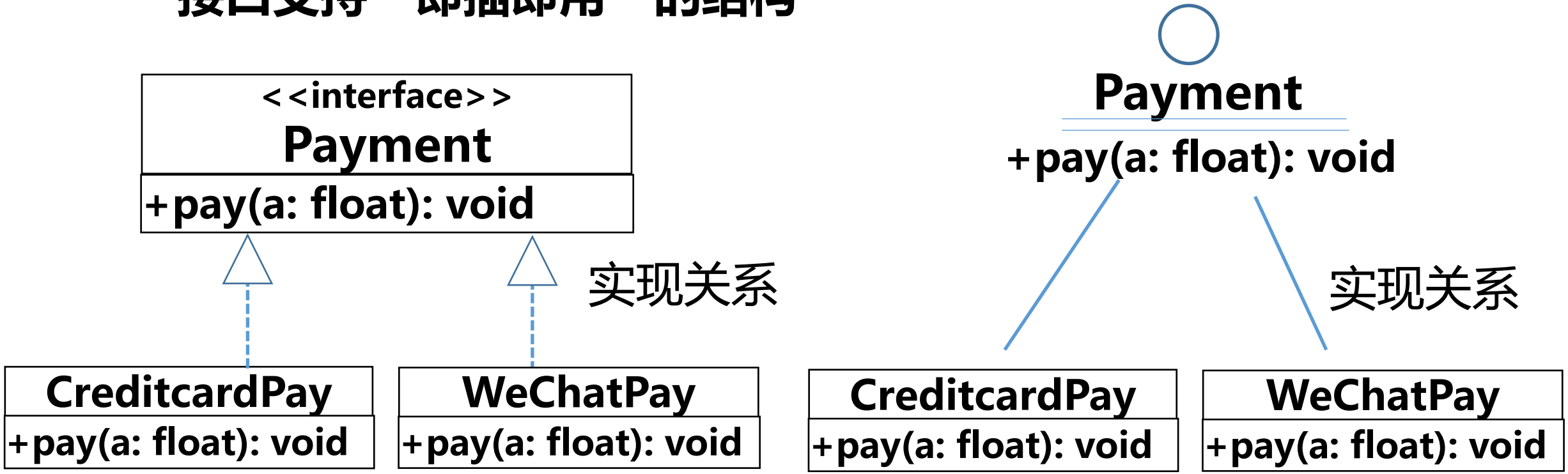
External interfaces



之后会处理财务系统接口类

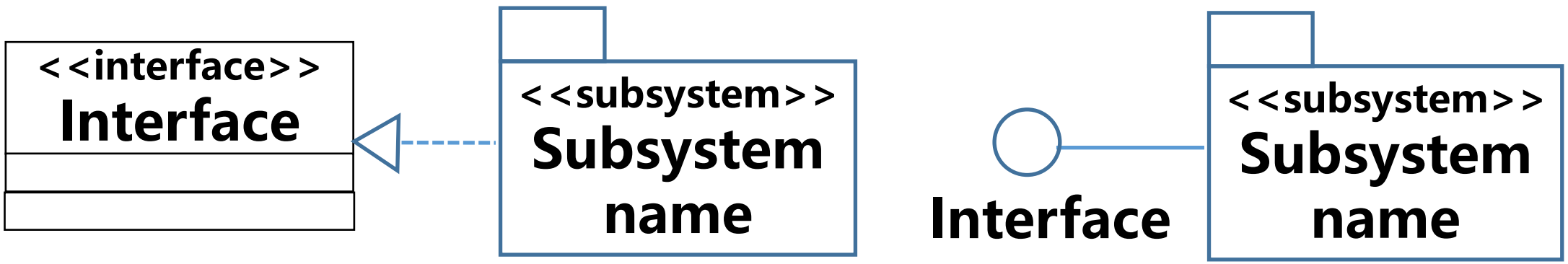
接口

- 接口(Interface)是类、子系统或构件提供的操作的集合
  - 接口允许用户以公开的方式定义多态，并且和实现没有直接联系
  - 接口支持“即插即用”的结构



子系统与接口

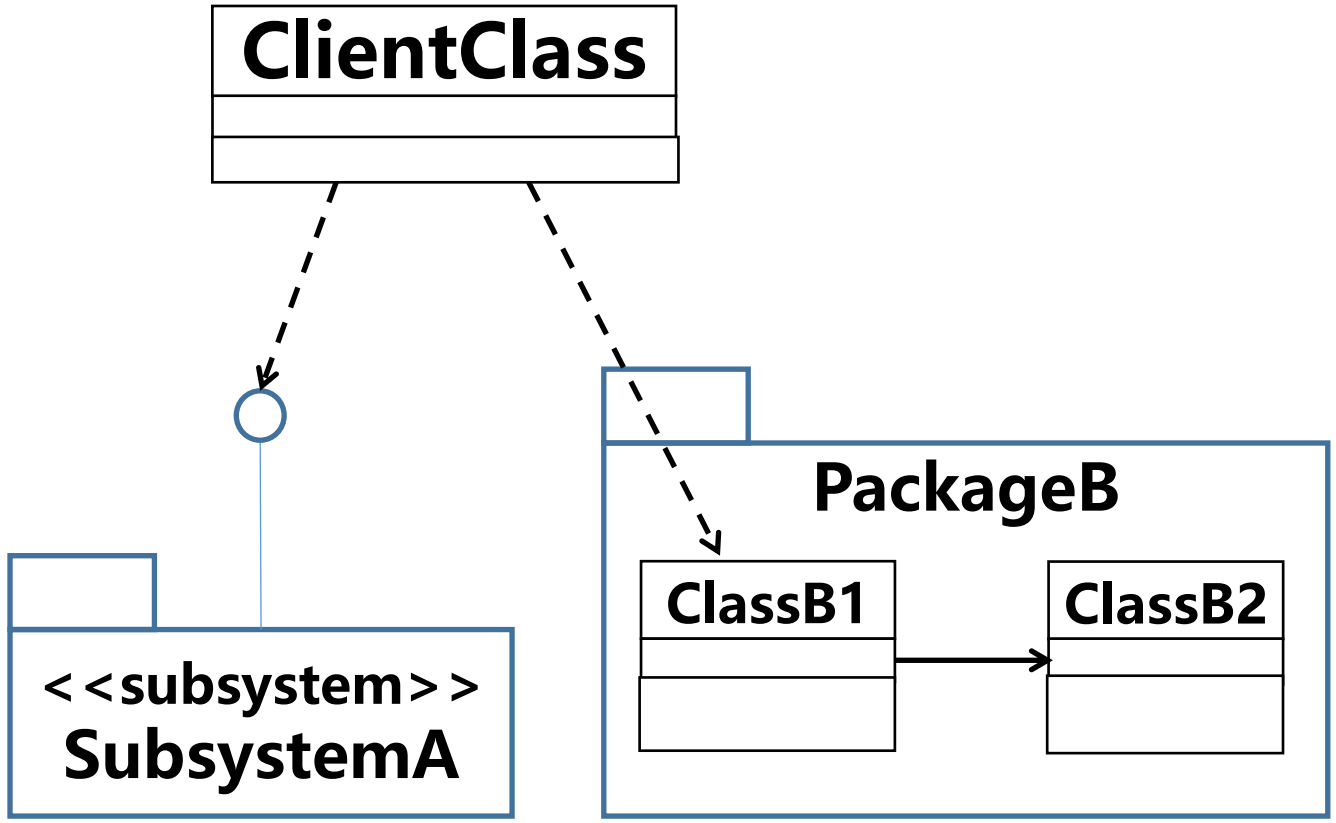
- 子系统(Subsystem)是一种介于包和类之间的一种设计机制，它实现一个或多个接口所定义的行为
  - 具有包的语义：能够包含其它模型元素
  - 具有类的语义：具有行为



利用子系统实现一个接口

子系统 VS. 包

- 子系统：
  - 提供行为
  - 完全封装实现细节
  - 容易替换
- 包：
  - 不提供行为
  - 不完全封装实现细节
  - 难以替换



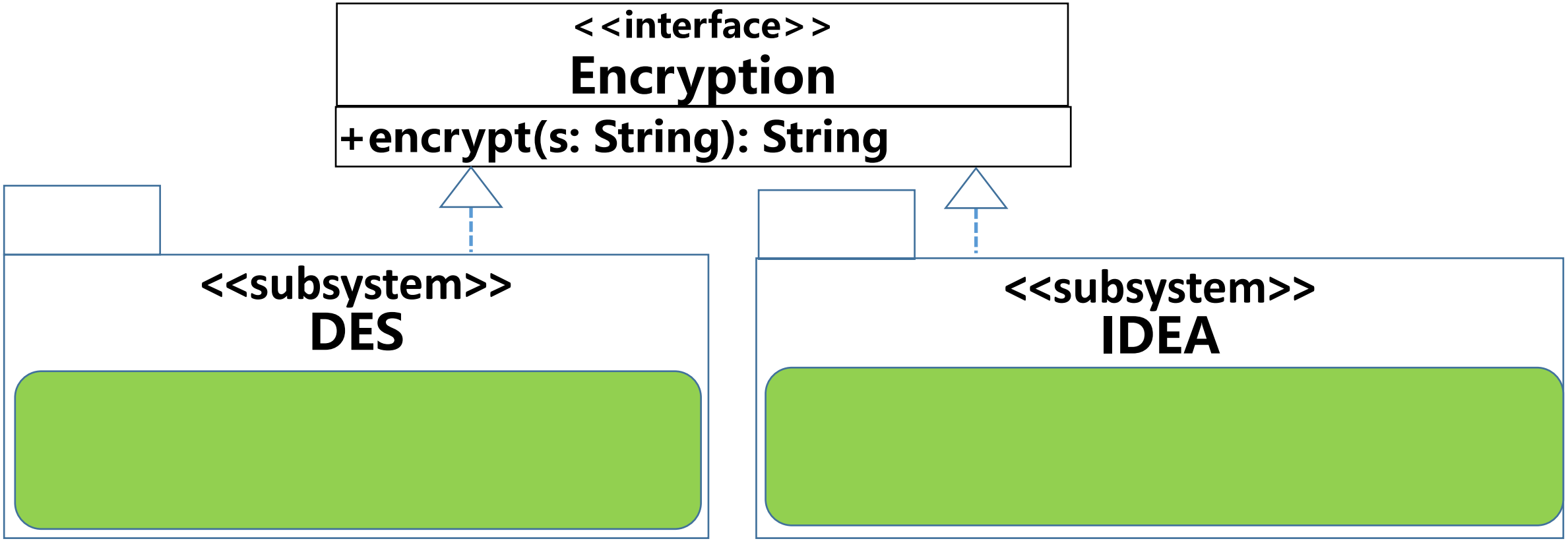
关键在于封装

子系统的主要用途

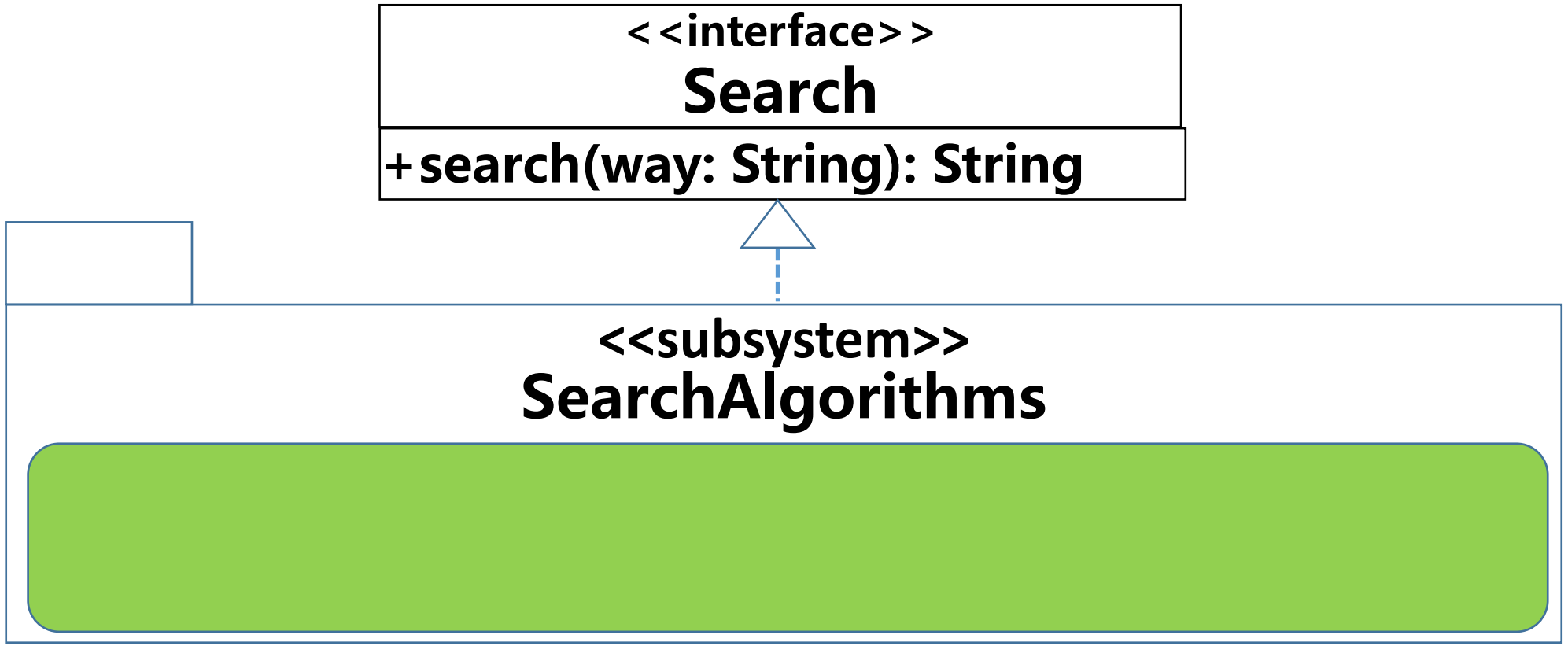
- 子系统可以将系统划分成独立的部分，将被实现为独立的构件，这些构件在保持结构不变的情况下，可以
  - 独立地开发和部署，
  - 适应变更，而不影响到其它系统
- 子系统也可用于
  - 将系统划分成若干单元
  - 表示设计中的既存产品或外部系统

什么情况应该使用子接口-子系统进行设计?

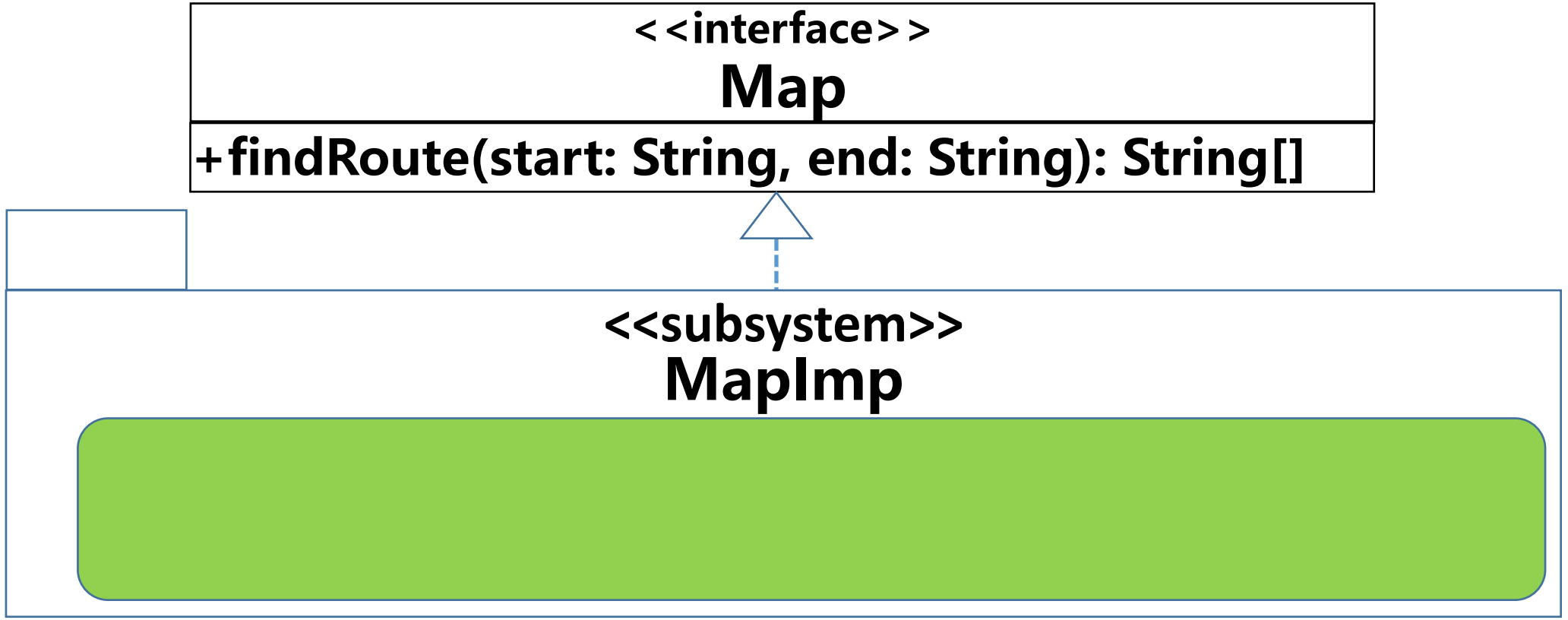
- **第一种情况**：提供复杂服务或通用功能的类。因为分析类所封装的业务复杂，很难通过单一的设计类实现；在设计阶段可以将其分解为接口-子系统



- **第二种情况**：提供公共服务的类，有很多类使用它，此时，为保证系统的稳定性，应该按照**开-闭原则**将其定义为接口和其相应的子系统（也可以是一个类）来实现。目的，利用子系统可替换型，以保证系统的稳定性



第三种情况：边界类中的外部接口类。这些外部接口用来描述设计外部既存产品或外部系统（**财务系统**，微信支付，支付宝支付，百度地图，某计算引擎，等等）。该外部接口的实现不仅与本系统的设计有关，且严重依赖外部环境。此情况下，只有通过将其业务封装在子系统内部，才能目标系统的稳定性。



- **第四种情况**：为了应对系统的经常变化以及系统的可扩展性，可以建立一个接口-子系统机制。

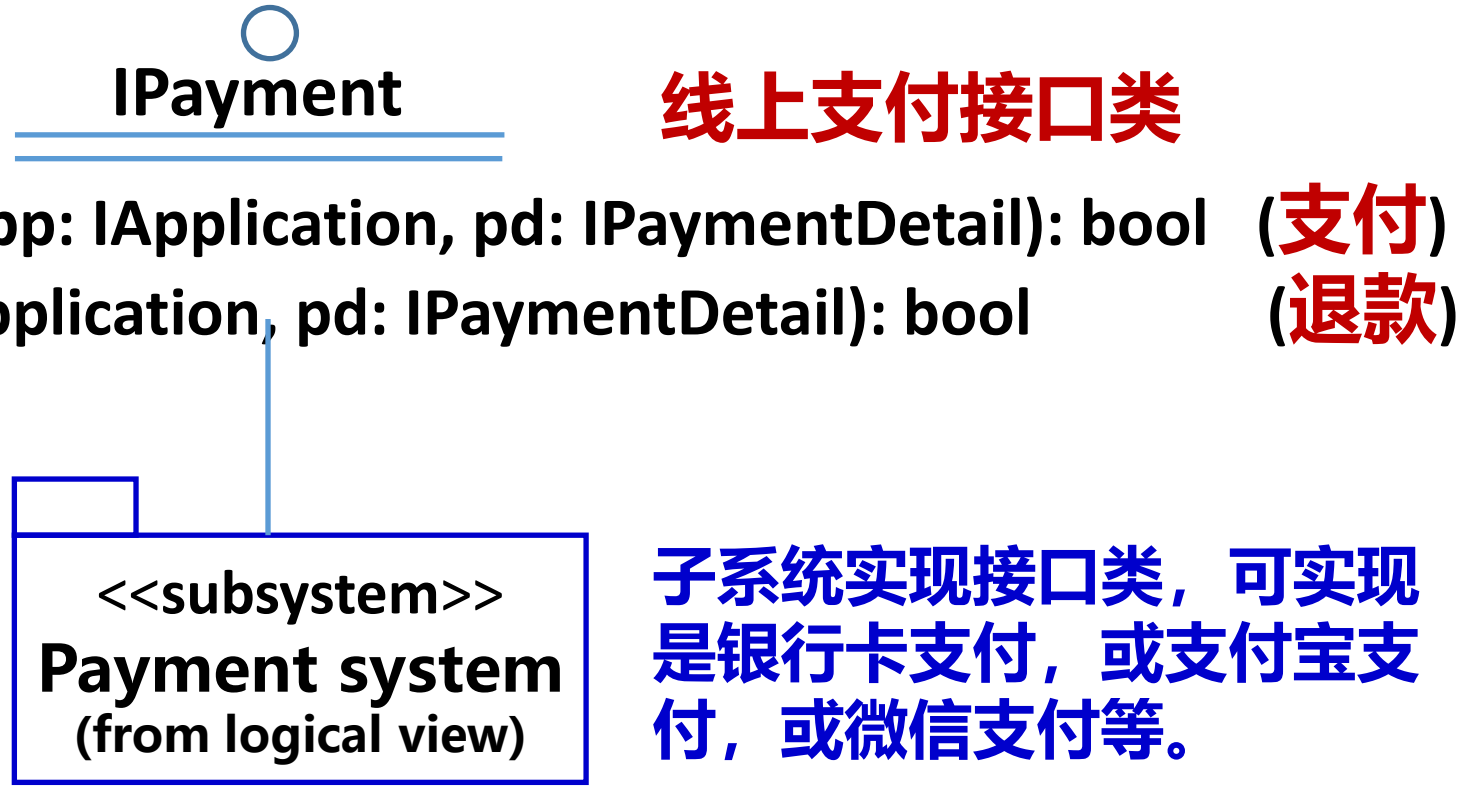
确定子系统的接口

- 目的
  - 基于子系统的职责确定其接口
- 步骤
  - 为每个子系统确定一个备选接口集
  - 寻找接口之间的相似点
  - 定义接口依赖关系
  - 定义接口所指定的行为
  - 将接口打包

- **关于打包**
- 在描述设计元素以及其分组和打包机制时，还需要充分考虑复用问题
- 可复用的元素来自两个方面
  - **内部：**其一是待开发系统内部：分析那些包/子系统内部的频繁被引用的元素，如果有复用的可能，按照**复用发布原则和共同复用原则，**将那些可以复用元素单独发布，建立可复用包
  - **外部：**另一类可复用的元素来自于待开发系统外部，可以复用第三方商业组件或以前开发系统的一些公共组件

实例1：确定子系统和接口-应对支付手段的变化与扩展

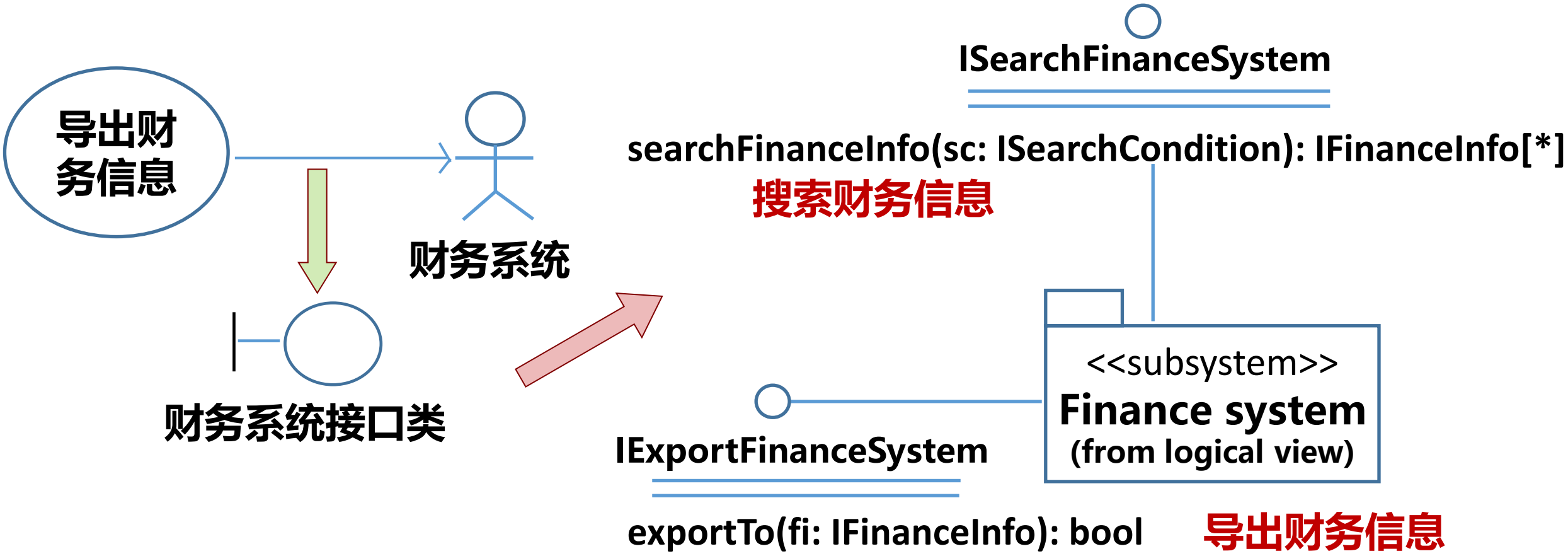
- **考虑关于支付的扩展**：现在的旅游团申请系统需求是现金支付，但考虑到线上支功能需要扩展(银行卡支付，微信支付)等多种形式，所以可以将支付的功能进行扩展。采用子系统的方式扩展如下。



- 添加接口和子系统后，软件架构也会相应的调整

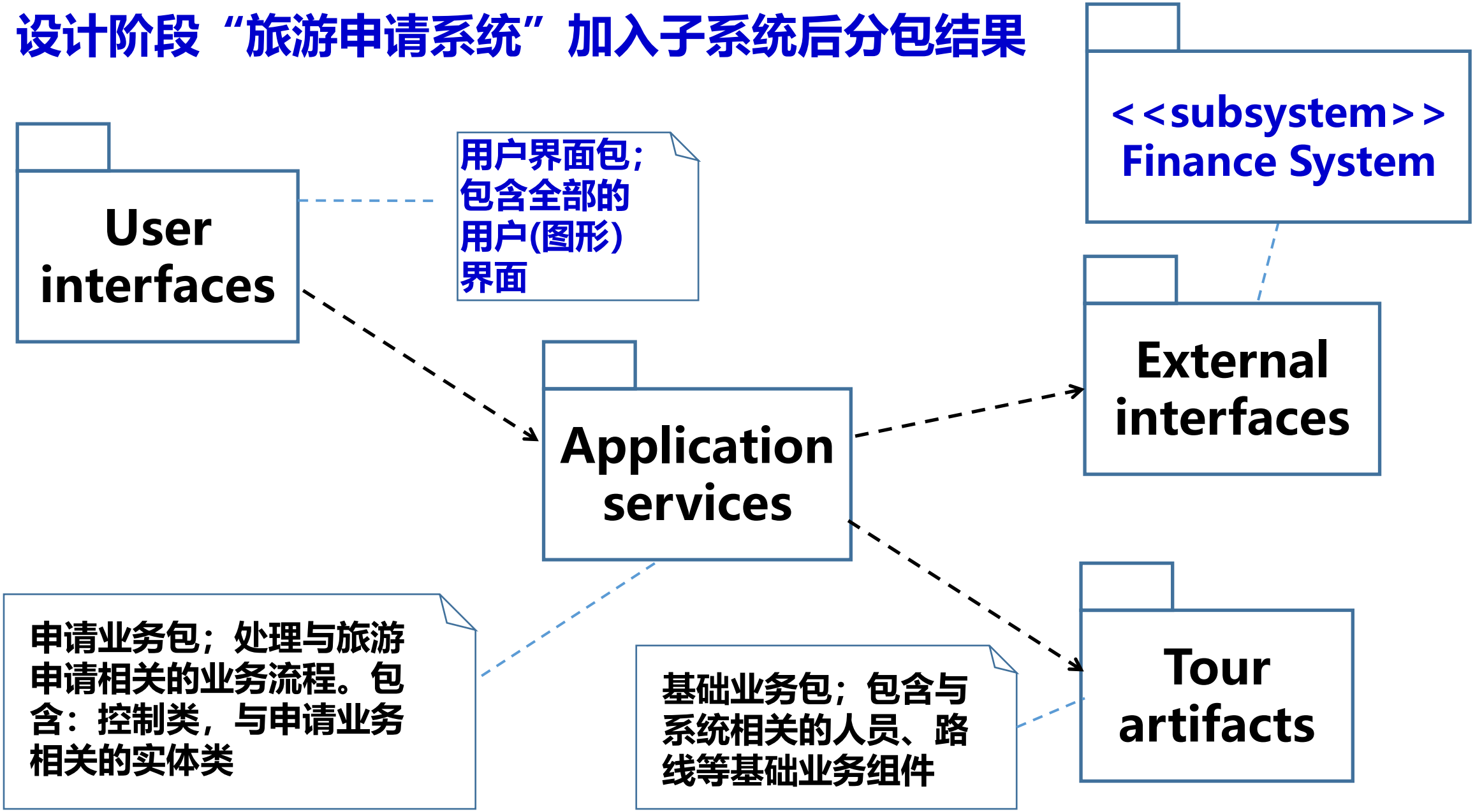
实例2：确定子系统和接口-将财务系统接口类转变为一个子系统

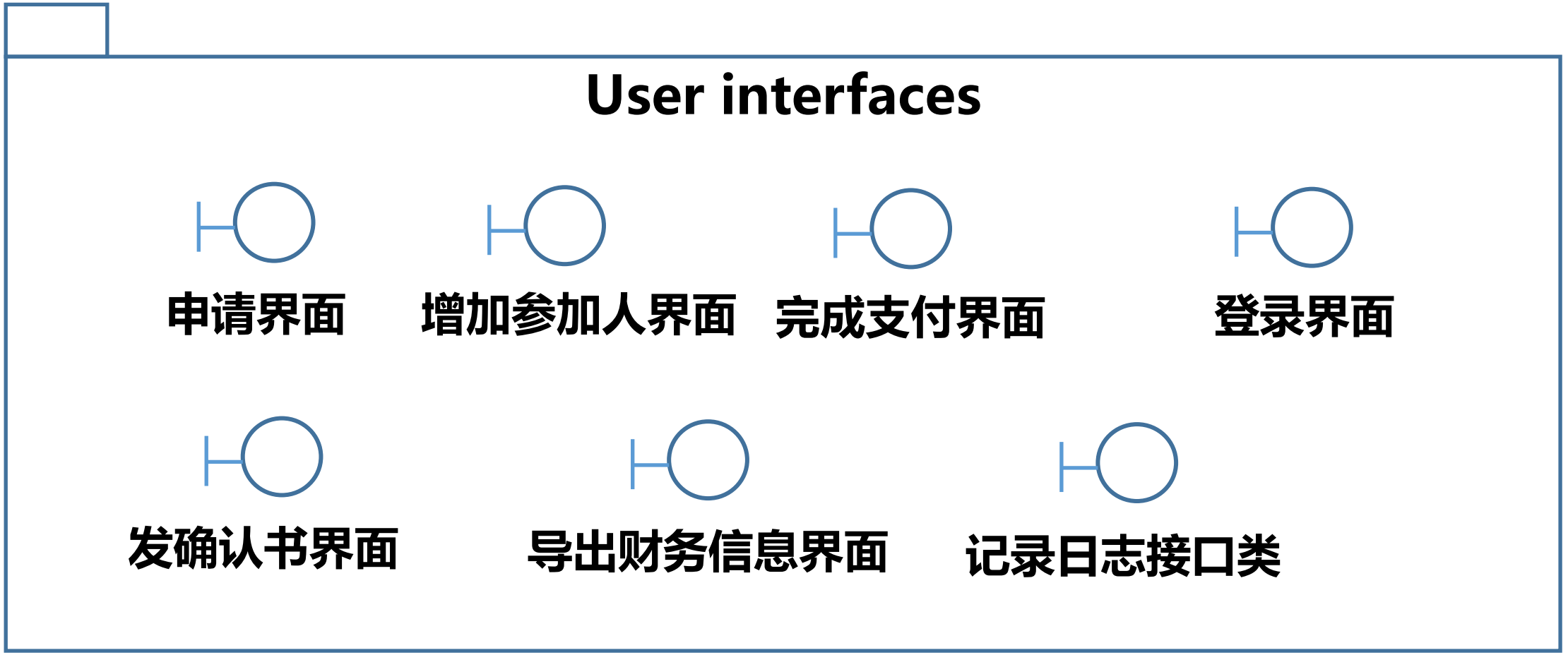
- 考虑“旅游业务申请系统”中的下列用例模型的分析和设计策略

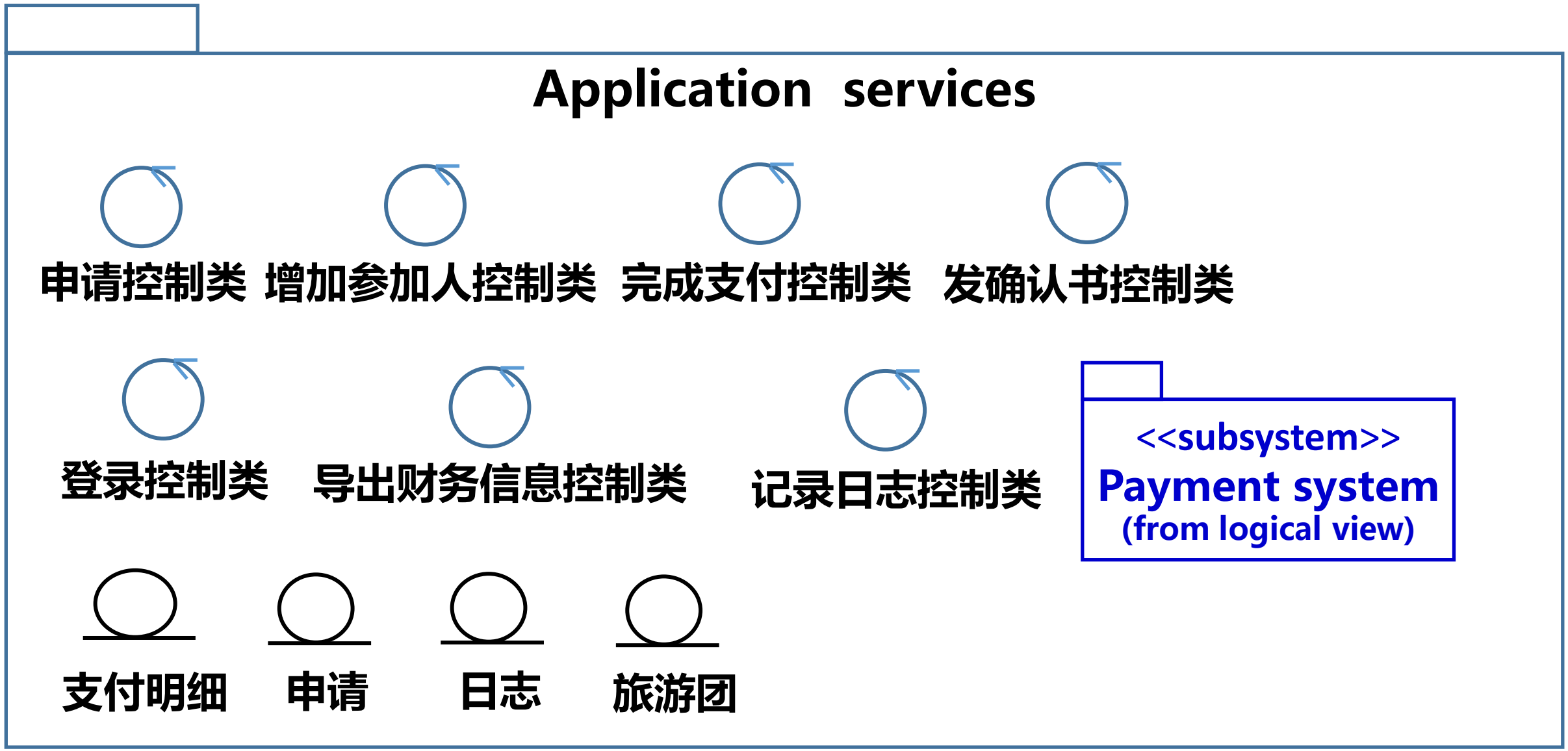


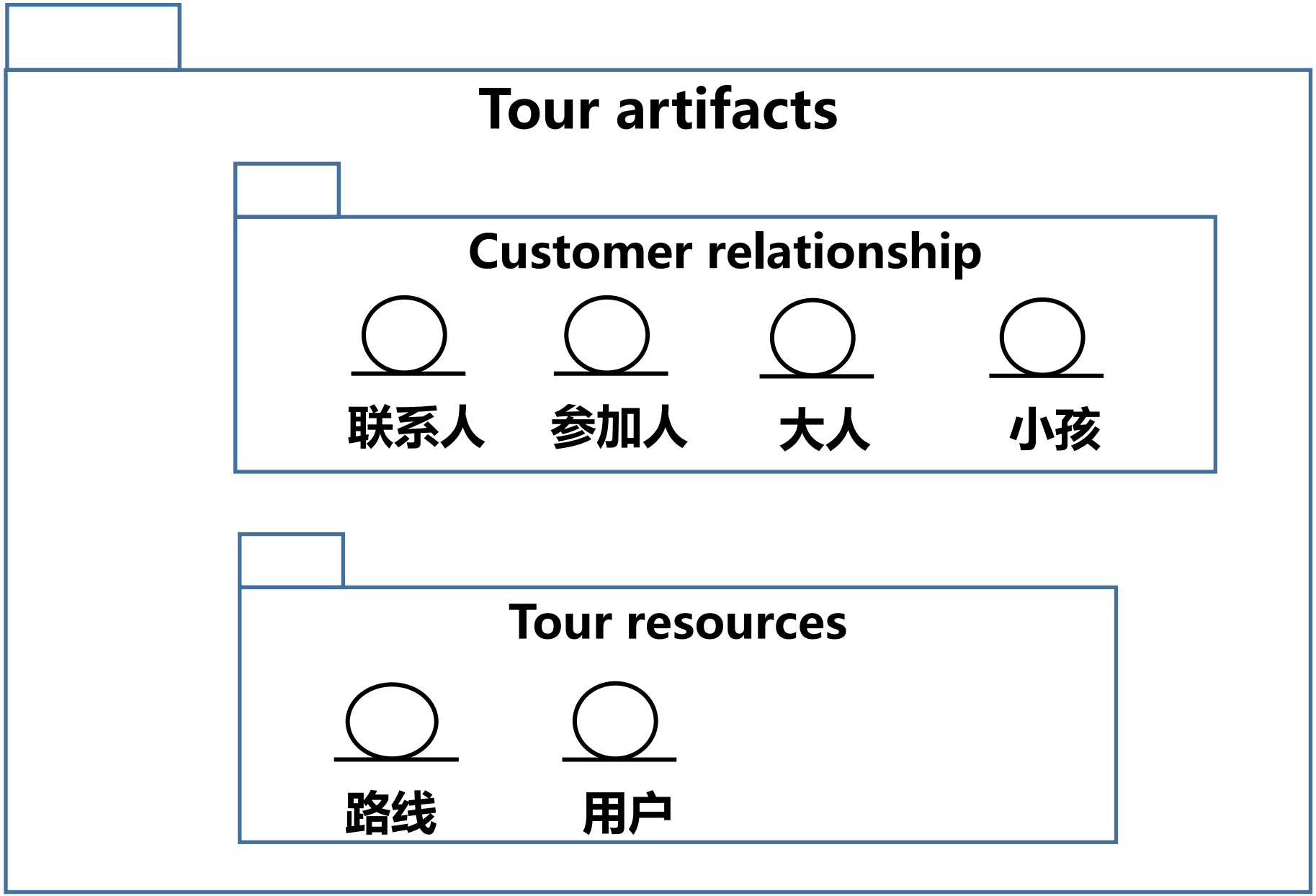
- 添加接口和子系统后，软件架构也会相应的调整

设计阶段“旅游申请系统”加入子系统后分包结果




User interfaces
申请界面
增加参加人界面
完成支付界面
登录界面
发确认书界面
导出财务信息界面
记录日志接口类


Application services
申请控制类
增加参加人控制类
完成支付控制类
发确认书控制类
登录控制类
导出财务信息控制类
记录日志控制类
<<subsystem>>
Payment system
(from logical view)
支付明细
申请
日志
旅游团


Tour artifacts
Customer relationship
联系人
参加人
大人
小孩
Tour resources
路线
用户

External interfaces

IPayment

线上支付接口类

makePayment(app: IApplication, pd: IPaymentDetail): bool 支付

withdraw(app: Ipplication, pd: IPaymentDetail): bool 退款

ISearchFinanceSystem

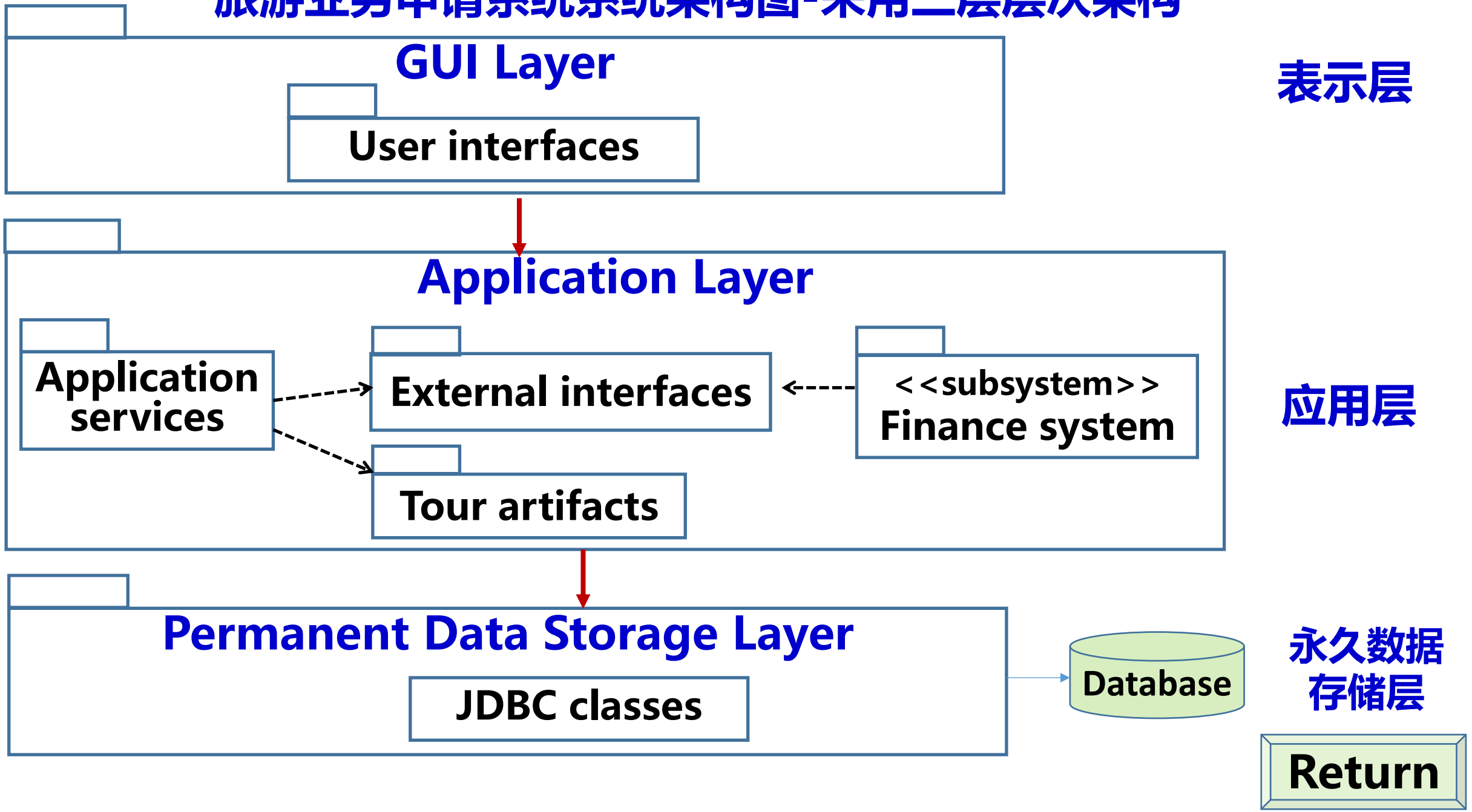
财务系统接口类

searchFinanceInfo(sc: ISearchCondition): IFinanceInfo[*] 搜索

IExportFinanceSystem

exportTo(fi: IFinanceInfo): bool 导出

旅游业务申请系统系统架构图-采用三层层次架构

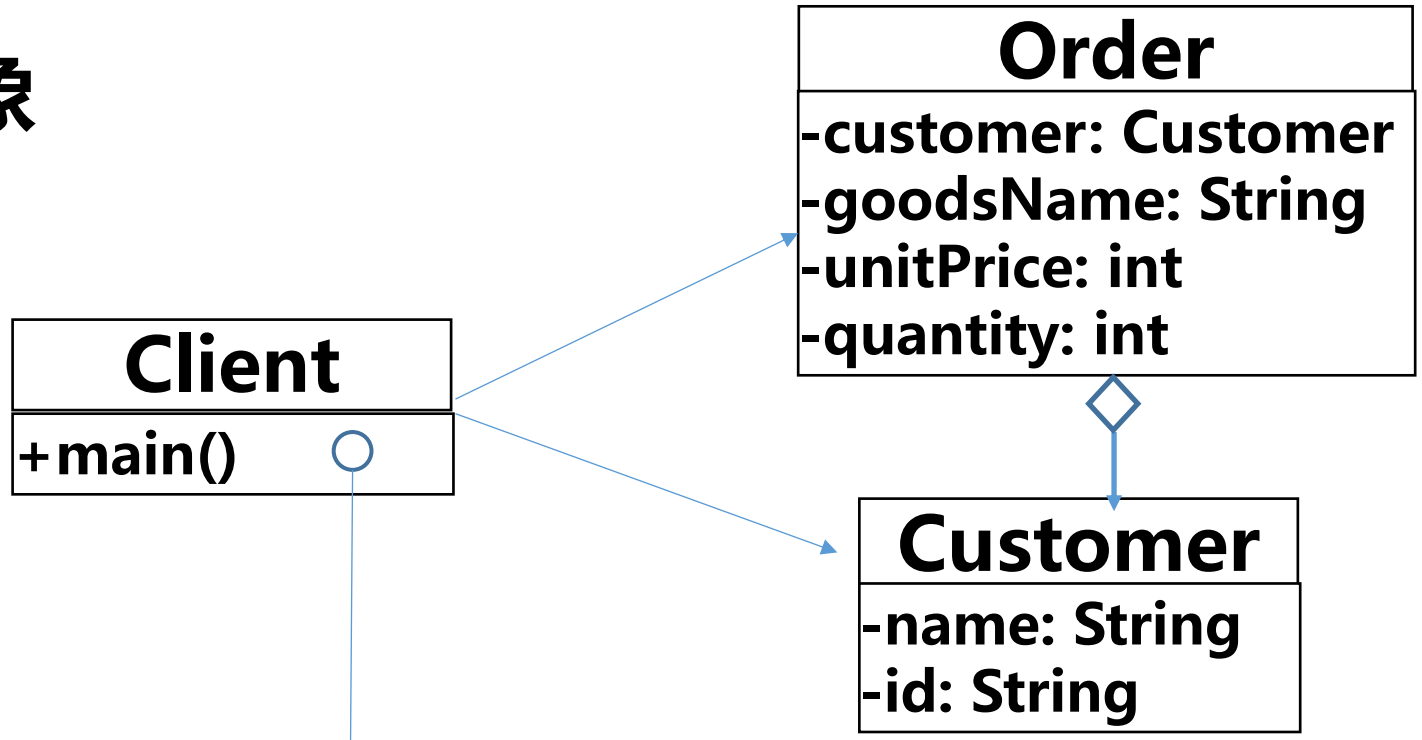


永久数据存储

存储：对象的持久化问题(即，对象中的数据存储问题)

- 关系数据库（RDBMS）（SQL server, Oracle, MySQL Server）
- 面向对象数据库（OODBMS）

用关系数据库来存储对象



Client类的代码：

```
Customer c = new Customer( “Qi Huli”, 12365);
Order o = new Order(c, “TV”,” Haixin” 1, 3500 )
```

怎样将预订数据存入数据库?

关系型数据库简单介绍，next PPT

关系型数据库用表(table)存储数据

Orders

Order1	PC	Apple	1	8600	8600	c3
Order2	TV	Haixin	2	3200	6400	c1
Order3	TV	Haier	1	1500	1500	c2

订单表

主键

外键

Customers

c1	Li Ming	12365	
c2	Wu mi	32176	
c3	Qi Huli	12365	

客户表

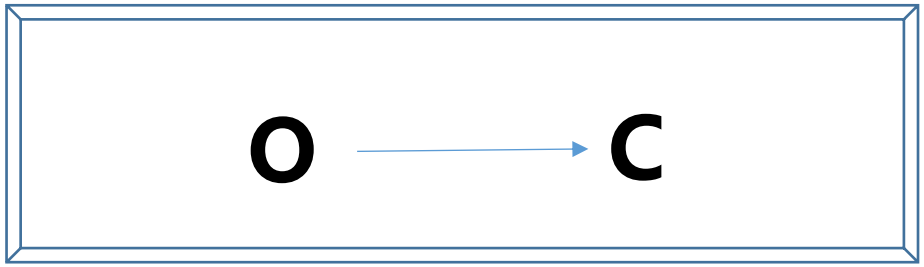
主键

面向对象数据库的情况

Client类的代码：

```
Customer c = new Customer(“Qi Huli”, 12365);
Order o = new Order(c, “TV”,” Haixin” 1, 3500 )
```

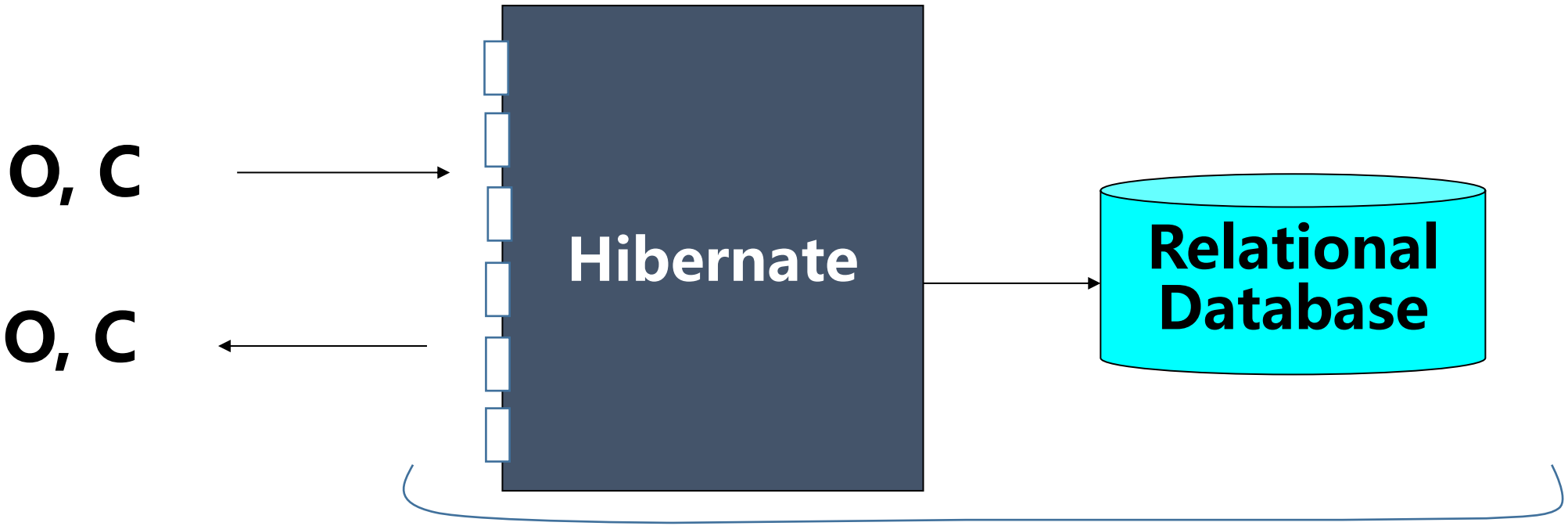
对象数据库：将整个对象存入数据库；存入对象O（连同数据）与C（连同数据），连同他们的关系也一起存入。



- 需要的时候，可以取出对象O（连同数据）与C（连同数据）。真好!
- 事实上：没有商业上适用的面向对象数据库。

- 使用O-R mapping 软件，例如，开源软件Hibernate，构建一个“面向对象数据库”。

```
Customer c = new Customer( “Qi Huli”, 12365);
Order o = new Order(c, “TV”,” Haixin” 1, 3500 )
```



存的是那个对象，取出的还是那个对象。

- 对于数据库操作，其应用场景主要有：
  - 初始化，建立数据库连接
  - 插入数据(Create)
  - 读取数据(Read)
  - 更新数据(Update)
  - 删除数据(Delete)

关于旅游申请系统中的什么数据数据会存到数据库中，见下一个PPT讲稿。

Return