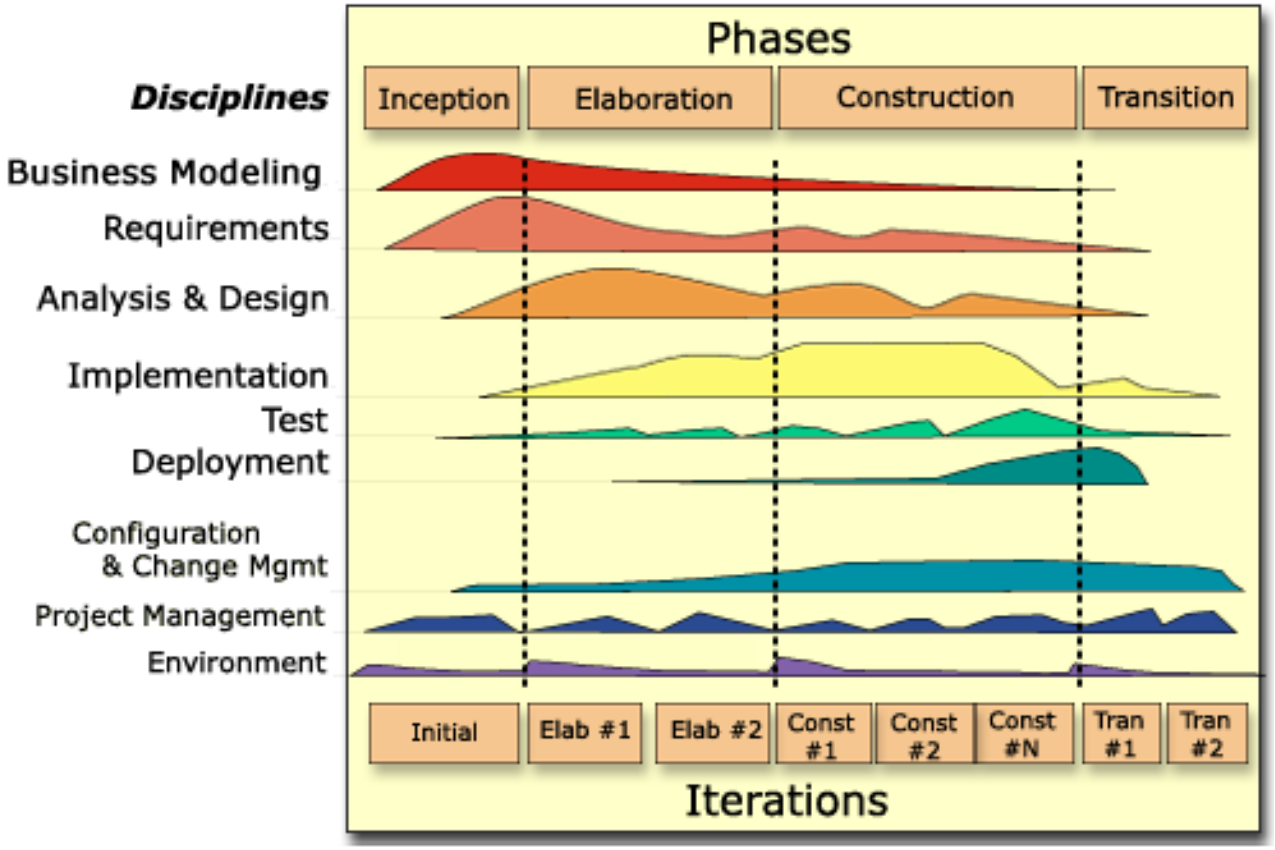
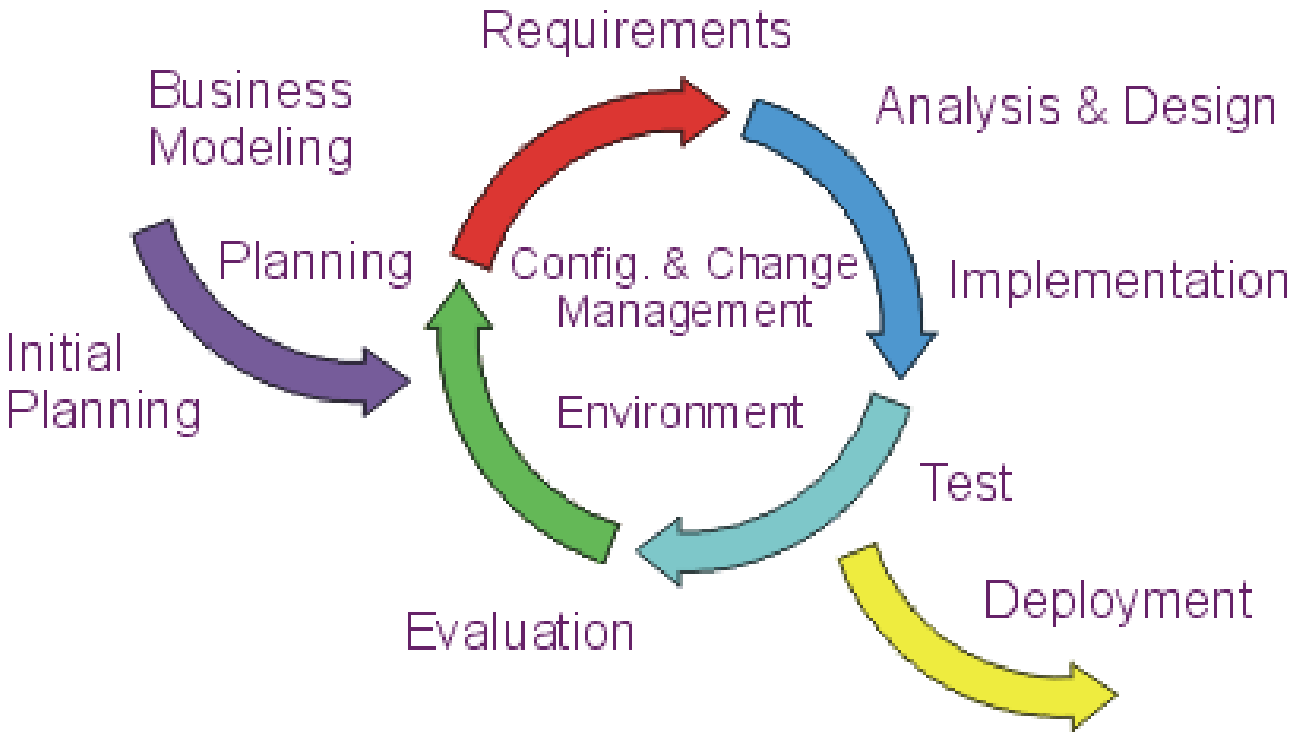
OOAD 案例分析之三
启元公司员工报帐系统

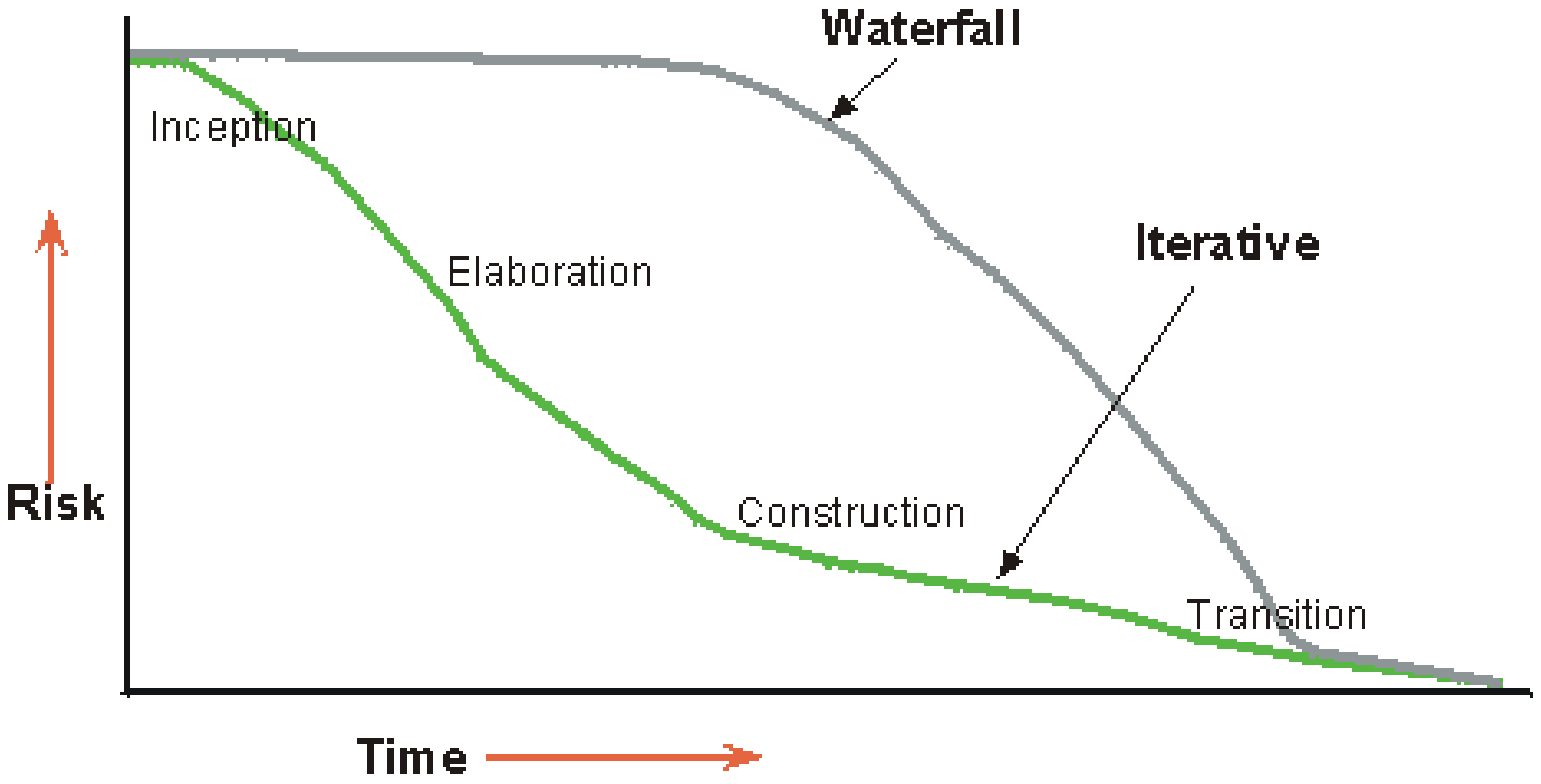
- RUP (Rational Unified Process)



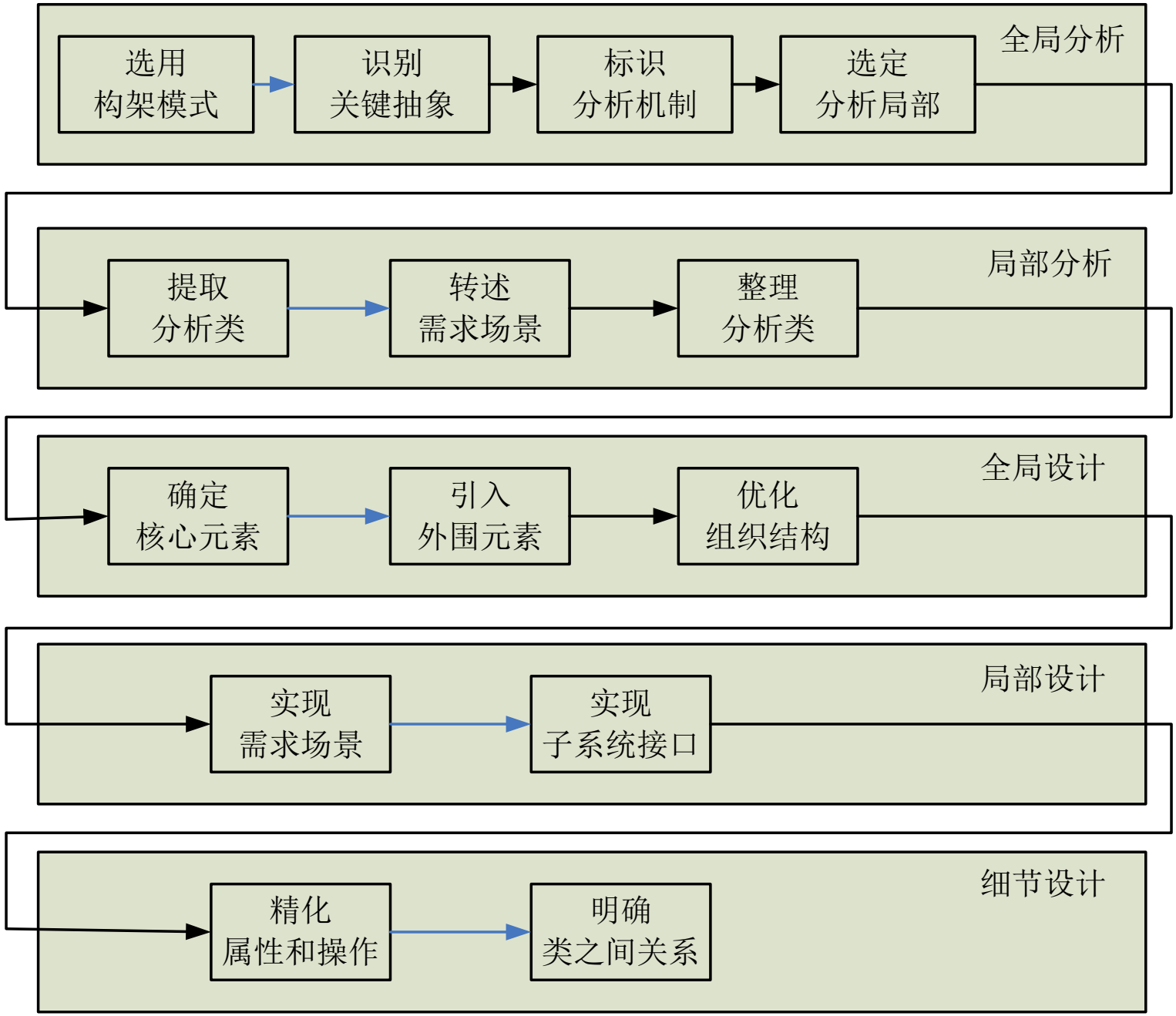
- RUP 工业界（推荐）使用的软件开发过程
- RUP 是一个迭代的软件开发过程



- 迭代的方法比传统的方法可降低软件开发中的风险



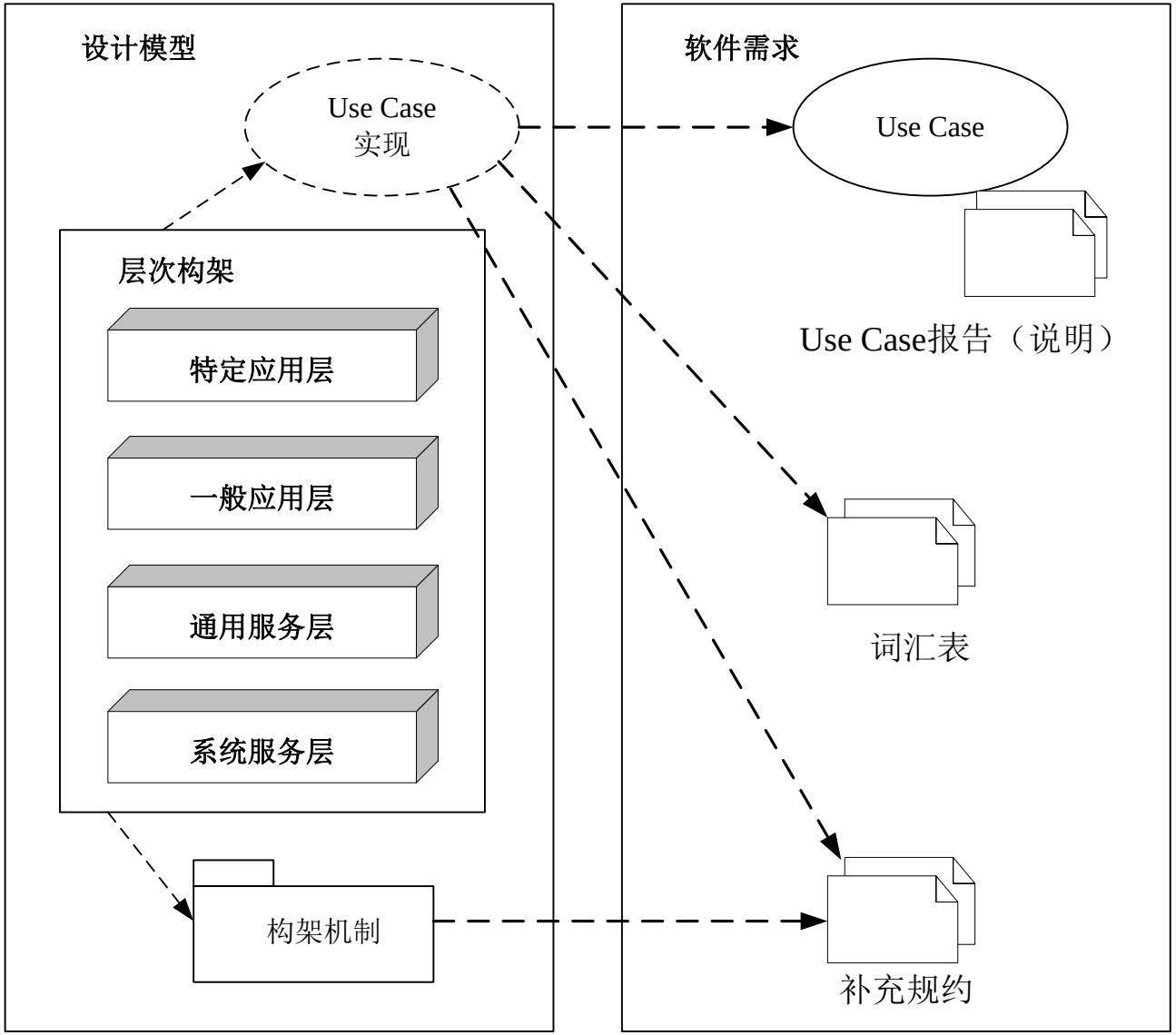
- RUP （Rational Unified Process）核心思想
  - Use Case 驱动
  - 体系架构为核心
  - 迭代化开发
- 本案例采用的是基于RUP的一个软件建模过程
  - 包含五项任务（14个活动，30个步骤）
    - 全局分析
    - 局部分析
    - 全局设计
    - 局部设计
    - 细节设计
- 本案例应用建模的目标
  - 有步骤，分层次地演进系统构架
  - 将软件需求琢渐转变为软件的设计方案
  - 保障软件的设计方案能够适应实施环境


全局分析
选用
构架模式
识别
关键抽象
标识
分析机制
选定
分析局部
局部分析
提取
分析类
转述
需求场景
整理
分析类
全局设计
确定
核心元素
引入
外围元素
优化
组织结构
局部设计
实现
需求场景
实现
子系统接口
细节设计
精化
属性和操作
明确
类之间关系

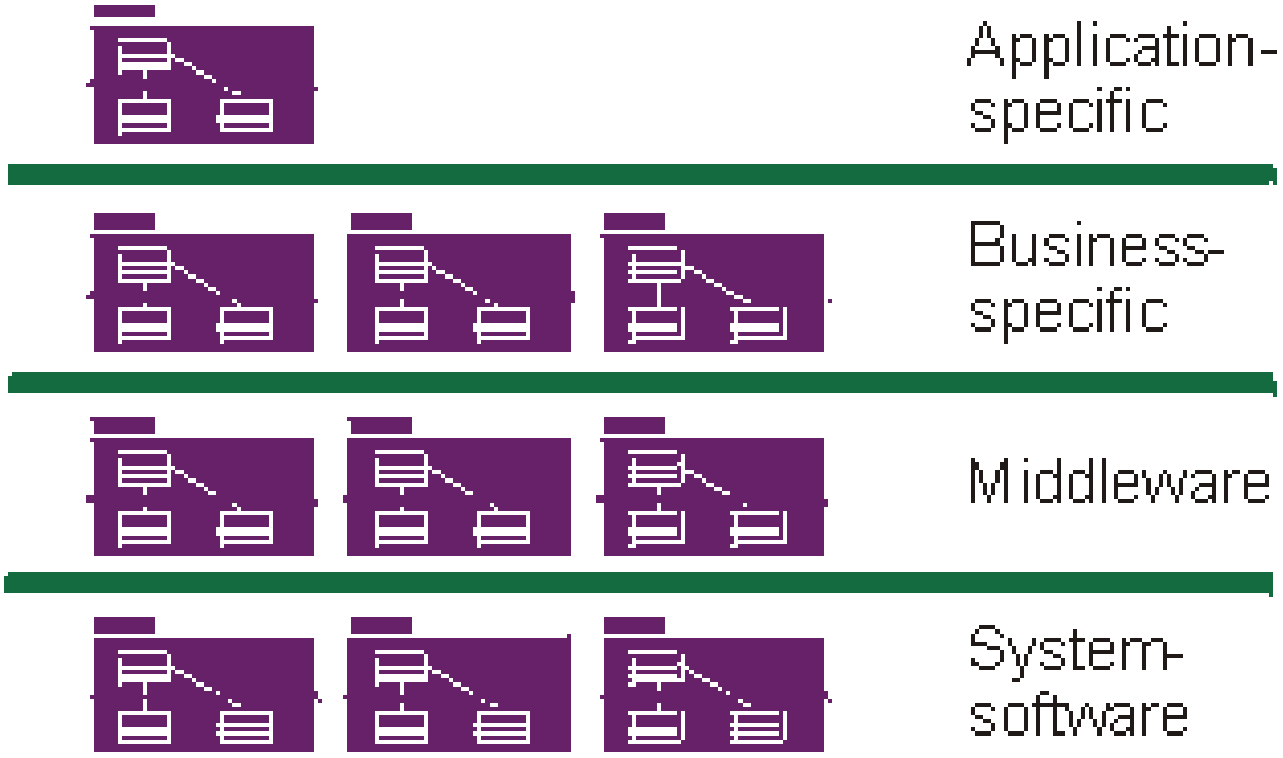
设计模型的内容和演进

- 设计模型的内容（本案例）
  - Use Case 实现（Use Case Realization）
  - 层次构架（Layered Architecture）
  - 构架机制（Architectural Mechanism）
- 层次构架是分析设计模型的核心
  - 层次构架的内容代表了分析和设计活动的实质结果
  - 将分析和设计的结果按照从特殊到一般的等级分组
  - 依赖于’Use Case实现’和‘构架机制’
- ‘Use Case 实现’
  - 通过一组协作（的对象）实现‘Use Case’描述的功能需求
  - 反映软件需求对设计内容的驱动
  - 是软件功能需求的内容过渡到层次构架的内容的桥梁
- 构架机制
  - 主要实现‘补充规约’中规定的非功能性需求
  - 是解决常见软件技术问题的设计经验描述，解决特定问题的模式

- 从‘应用逻辑牵引’和‘软件技术支撑’两个方面，‘Use Case实现’和‘架构机制’描述了从软件需求向设计方案的平滑过渡过程



- 层次构架是经典系统构架模式（Architecture Pattern）的一种
- RUP对层次构架（模式）的描述



- 示例软件的Problem Statement

PS - Problem Statement

启元公司的总部设在北京，它在上海，广州，成都，和西安都有分支机构。全公司有约700名员工。鉴于公司业务和员工团队的快速发展，为了提升整体的工作效率，启元公司准备开发一套员工报帐系统，用来取代现有的人工报帐系统。

报帐系统将支持员工记录（或预见）日常业务活动的开销，并自动结算每个月应该返还员工的补偿金额。补偿金额将自动存入员工的工资户中。

报帐系统应具有基于先进技术的图形用户界面，员工可以输入业务活动的种类和简短描述，活动开销的类别，选择不同的支付方式，并可以生成灵活的报表。

报帐系统应该有能力根据员工提供的信息和要求返还补偿金额，同时保存全部员工的报帐信息。员工可以通过他们自己的电脑来使用报帐系统。由于牵扯到财务信息，报帐系统必须提供可信的安全机制。

PS - Problem Statement（cont.）

启元公司现有一套基于微软 SQL Server 的人事管理系统，记录员工的基本信息和团队的组织结构。报帐系统将和现有的人事管理系统协同工作，需要引用人事管理数据库中的部分信息，但不会更改启内容。

通过报表系统，员工能够在出差前（提前两天）按照规定的额度向公司申请借款，相关的经理人员能够通过报帐系统批复或拒绝。报帐系统应在相关负责人批复之后通知该员工提取现金或确认相应款项已经划入指定信用卡（根据员工的要求）；员工可以通过报帐系统报销合理的业务活动费用。

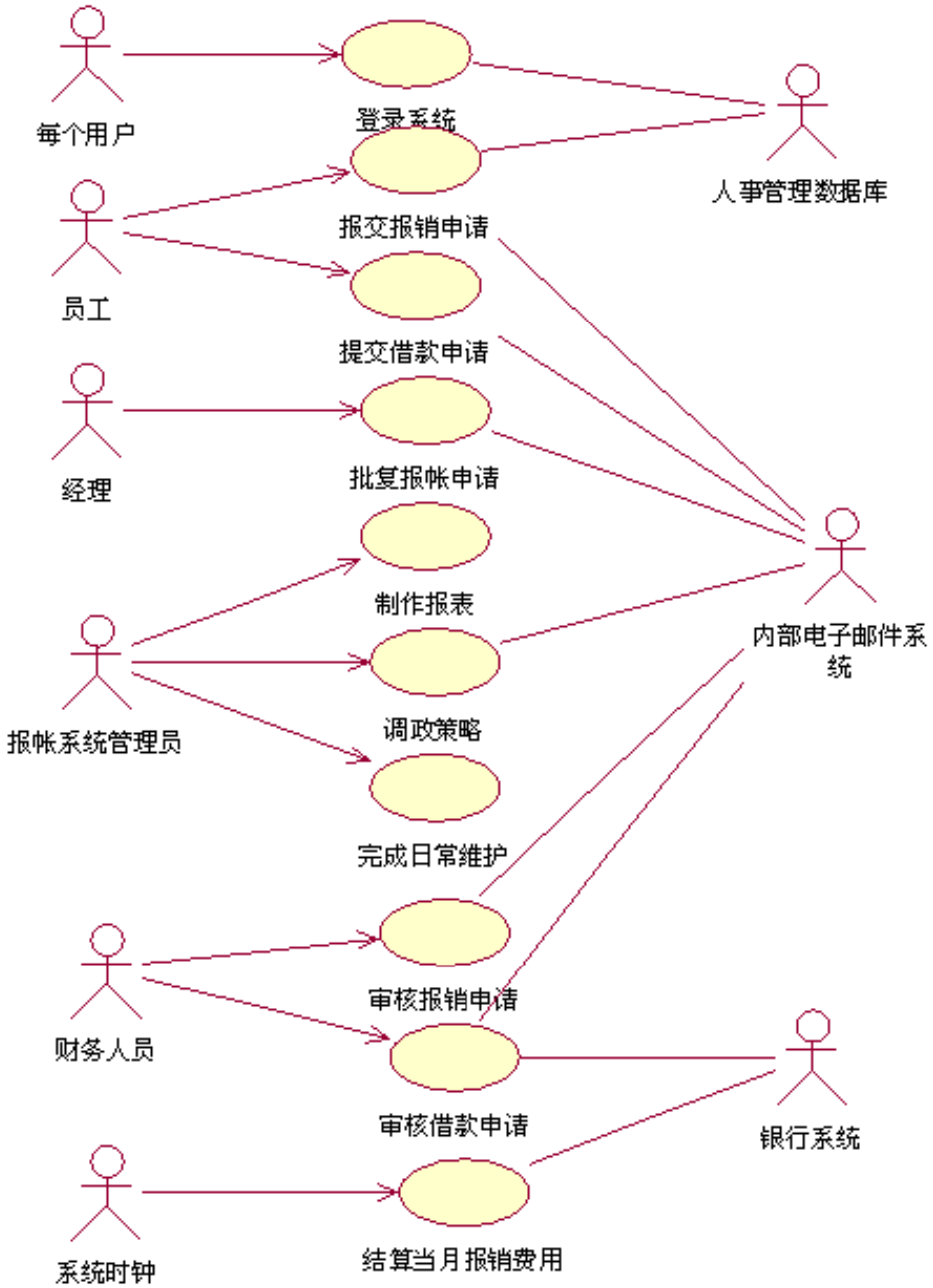
财务部门将指定一位报帐系统管理员监管拟建系统中的信息，负责初始设置和维护特定的分类额度准则，并能够定期或随机地向各部门负责人提交报帐情况和统计报告。

报帐系统在每月25日对通过审批的报帐申请自动作一次结算，并以电子邮件的方式通知应该得到补偿的员工，同时生成一份统计报告传送给财务部门的系统监管人员。

OOA – 需求获取活动

- Chapter 4， Requirement Elicitation
- 从 PS 入手，学习理解应用域有关概念和业务
  - 财务报帐知识入门
  - 人工报帐操作方式
  - 其它类似（软件）系统
- 建立启元公司员工报帐系统的用例模型
  - Use Case Diagram （see next slide）
  - Use Case （文字格式的描述）
    - 以“提交报销申请”为例

Use Case Diagram （Rational ROSE)



用例名称：　　提交报销申请

参与执行者：　公司员工

入口条件：　　员工成功登录系统，通过身份验证。被系统提示进入‘报销申请’或者‘借款申请’功能

事件流程:

1. 打开报销单

【员工】员工选择进入‘报销申请’功能

【系统】如果该员工当月报销单存在，系统将取出相应信息并展示给员工

2. 添加报销记录

【员工】员工要求添加一条报销记录

【系统】系统显示一条空白报销记录

3. 填写报销记录

【员工】员工开始填写报销记录，每条报销记录包括的信息有：业务活动发生的时间，地点，客户名称（可选），原因以及费用金额和种类（交通，餐饮，会议，通信和杂项）。

【系统】系统显示并记录员工输入的信息。为了让员工方便而准确地输入相关信息，除了客户名称，业务活动原因和金额之外，其它信息域提供相应的下拉式选择列表。

（重复以上针对每一条报销记录的活动，直至所有记录填写完毕）

4. 验证报销单

【员工】员工填写完所有的报销记录之后，要求系统验证这些记录的合理性

【系统】报销记录的初始状态为‘未验证’，当每一条报销记录被验证为合理，系统将该报销记录的状态设为‘已验证’，系统在验证所有报销记录（为‘已验证’）之后提示用户可以提交本月的报销单。验证为合理的记录必须满足几种条件：第一，不同种类的费用不超过相应的限额；第二，报销费用的类型要和员工的职能匹配

5. 提交报销单

【员工】所有报销记录经过验证后，员工提交当月的报销单。

【系统】系统保存这张报销单，将报销单的状态设置为‘已提交’并记录
提交日期，同时这张报销单被设为‘只读’。系统要从人事管理数据库中
获知该员工及其经理（负担该员工当月开销者）的电子邮件地址。为了及时通知相关人员，系统将自动生成一份以当前报销单为内容的电子邮件发送到该员工及其经理的信箱中。当邮件发送后，员工得到一个确认信息。

备选事件序列组（Alternative Flows）

A1. 创建当月报销单

【起始位置】基本事件序列中，员工进入报销申请程序并准备打开当月报销单。

【触发条件】该员工的当月报销单不存在。

【具体内容】系统为该员工创建一张当月报销单

【返回位置】基本事件序列中的’打开报销单‘步骤

A2. 删除报销记录

【起始位置】在提交报销单之前的任意时间点

【触发条件】员工希望删除某一条报销记录

【具体内容】系统删除由员工指定的报销记录

【返回位置】同’起始位置‘

A3. 更新报销记录

【起始位置】在提交报销单之前的任意时间点

【触发条件】员工希望更新某一条报销记录

【具体内容】系统根据员工重新输入的内容更新相应的一条报销记录。将该记录状态设置为’未验证‘。

【返回位置】同’起始位置‘

A4. 保存当月报销单

【起始位置】该 Use Case 允许员工在事件流中的任意时间点保存当月的报销单

【触发条件】员工希望将已录入的报销记录保存在报帐系统中

【具体内容】系统保存该员工的当月报销记录，并给出确认信息。员工可以在保存当月报销单之后直接退出系统

【返回位置】同’起始位置‘

A5. 报销记录不合理

【起始位置】基本事件序列中，’验证报销单‘步骤中对每一条报销记录

验证结束之后

【触发条件】报销记录未通过验证，由两种情形：第一，某报销记录的金额超出了其对应类型费用的上限，已知有三种：请客户用餐人均超过300元，出差式每天住宿费超过800元，移动电话费在无特殊说明情况下超过800元;第二，报销费用的类型和员工所处的部门及职能不匹配，已知的情形式业务部门的员工申请加班补助。

【具体内容】告知员工不合理的报销记录编号，以及未通过验证的原因

【返回位置】基本事件序列中的’填写报销单‘步骤，目的是更正有问题

的报销记录

A6. 人事管理数据库不可用

【起始位置】基本事件序列中，’提交报销单‘步骤的结尾

【触发条件】当报帐系统向人事管理数据库索取信息而该数据库没有正常的响应

【具体内容】告知员工’人事数据库不可用，报帐单没有提交成功‘

【返回位置】Use Case 执行结尾

A7. 邮件未及时发出

【起始位置】基本事件序列中，’提交报销单‘步骤的结尾，成功地从人

事数据库获得相关信息之后

【触发条件】报帐系统要求发送相关邮件时，邮件系统没有及时的响应

【具体内容】系统将以提示信息的方式告知员工，’邮件没有及时发出，但是报销单在系统内已经提交成功，待邮件系统恢复后，相关邮件回自动发出’

【返回位置】Use Case 执行结束

出口条件：　　如果该Use Case顺利执行，员工的报销申请单将被建立，更新，保存或者提交；否则，系统的状态应该保持和该Use Case执行之前相同

特殊需求：　　暂无

任务一： 全局分析

- 全局分析包括四项活动
  - 选用构架模式
  - 识别关键抽象
  - 表示分析机制
  - 选定分析局部

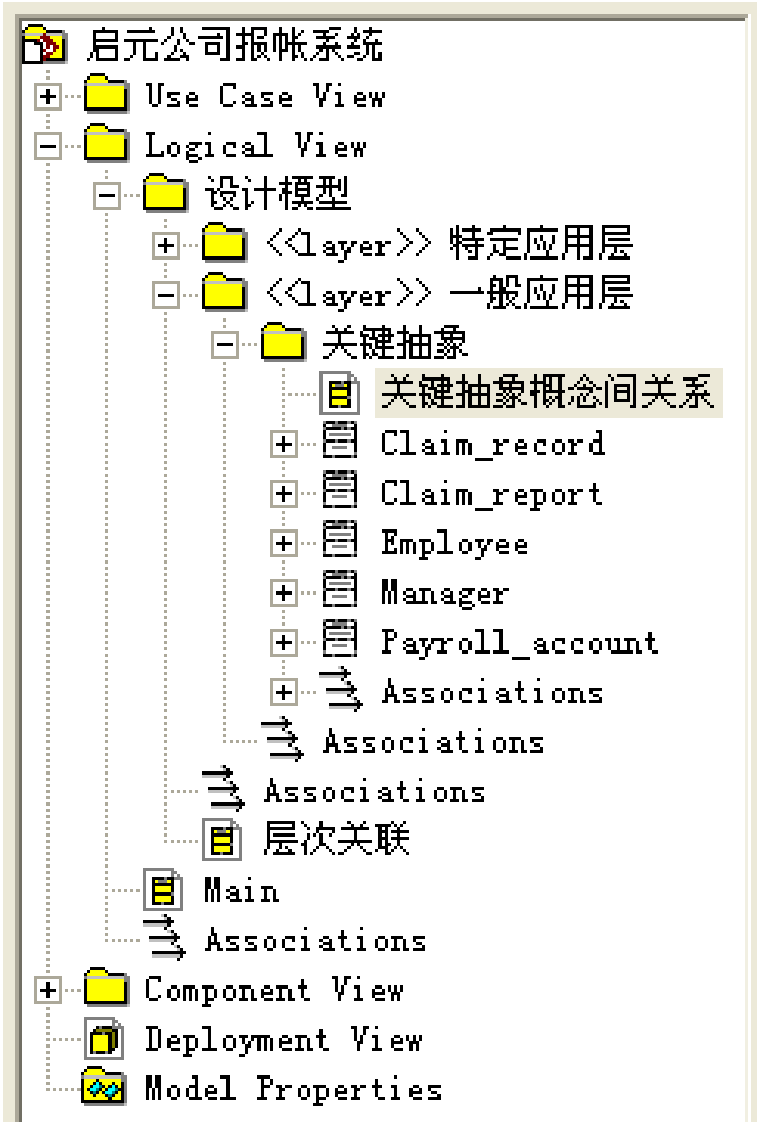
选用构架模式

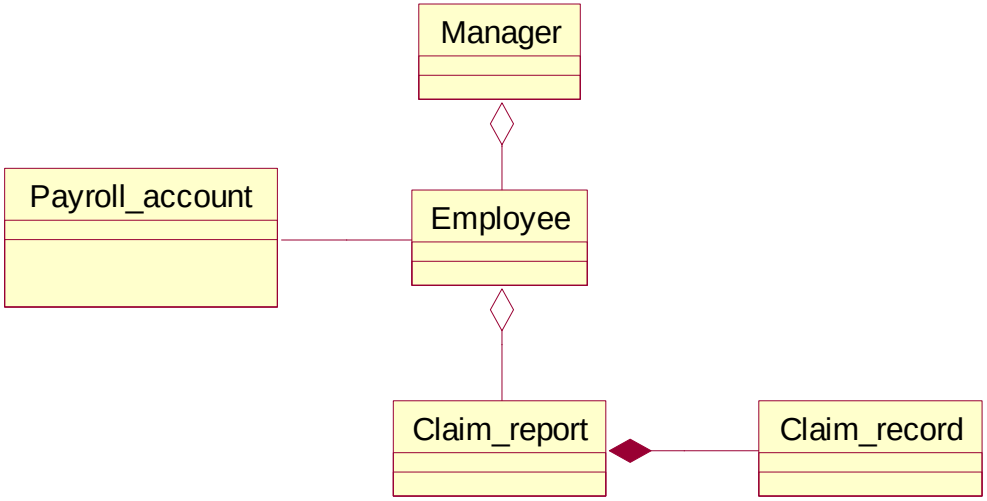
  - 选定一种可产生高内聚低耦合的系统分解方式
  - 系统设计中子系统分解经典模式（见课本‘系统设计’）
    - 分层的模式
    - 分区的模式
    - MVC模式
    - 管道和滤波器模式
  - 本例选择了分层的方式（层次架构模式）
  - 层次构架模式是一种普遍适用的构架模式，特别是对中，大型系统的面向对象的软件开发
  - 相应建模活动（见Rational ROSE 模型）

识别关键抽象

- 寻找那些无论在问题域还是方案域都具有普遍意义的概念点
- 目的是确定拟建系统必须处理的核心概念
- 根据一般经验，‘关键抽象’往往对应重要的实体信息

- ‘关键抽象’的来源
  - 词汇表（面向用户和软件投资者的，微观层面的，权威的术语定义集合）
  - Use Case Model
  - 反映领域知识的既往经验（业务模型，领域模型）
- 识别出的‘关键抽象’放入设计模型中‘一般应用层’的‘关键抽象’包中
  - 每个‘关键抽象’用一个类来表示
  - ‘关键抽象’包是一个过渡性的包，其中的类随着建模活动的展开将被移到构架中更恰当的位置
- 案例中的部分‘关键抽象’
  - 报销单（ClaimReport）
  - 报销记录（ClaimRecord）
  - 员工（Employee）
  - 经理（Manager）
  - 工资户头（Payroll_account）


启元公司报帐系统
Use Case View
Logical View
设计模型
<<layer>> 特定应用层
<<layer>> 一般应用层
关键抽象
关键抽象概念间关系
Claim_record
Claim_report
Employee
Manager
Payroll_account
Associations
Associations
Associations
层次关联
Main
Associations
Component View
Deployment View
Model Properties


Manager
Payroll_account
Employee
Claim_report
Claim_record

标识‘分析机制’

- ‘分析机制’是‘构架机制’在‘全局分析’阶段的表现形式
- ‘分析机制’概括地说明如何实现必须具备的软件功能
- ‘分析机制’的实例在系统构架中充当某些复杂行为的‘占位符’

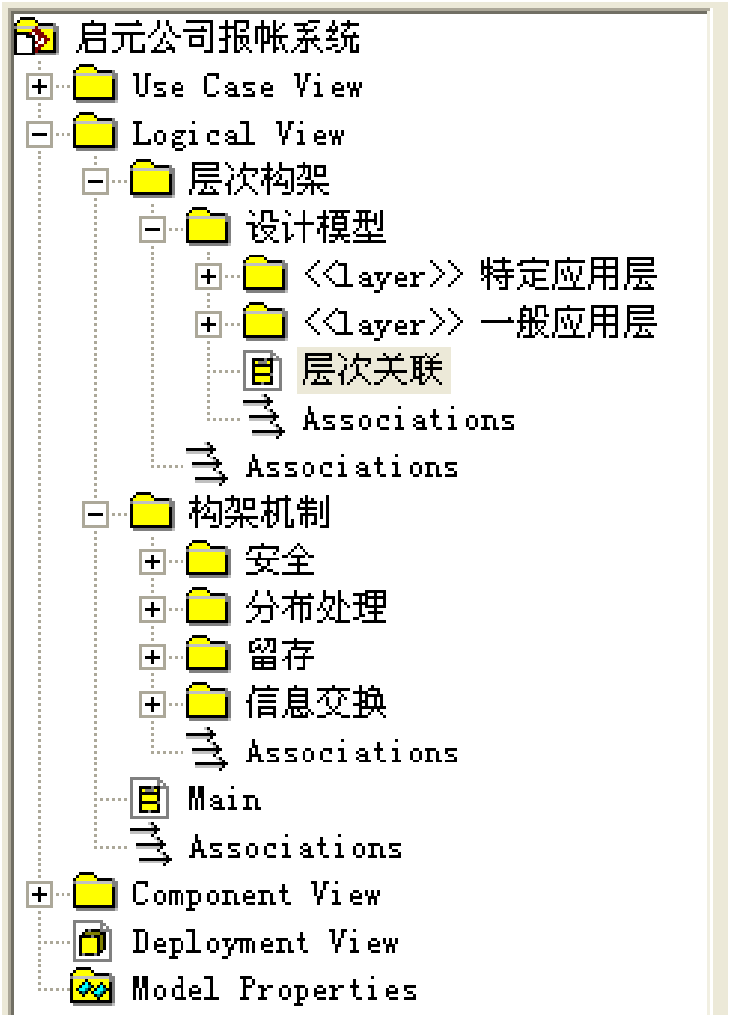
- 常见的分析机制（类似于教材系统设计中讨论的‘系统范围的问题’）
  - 留存
  - 分布式处理
  - 安全性
  - 进程间通信
  - 消息路由
  - 进程控制与同步
  - 交易事务管理
  - 信息交换
  - 信息的冗余
  - 错误检测，处理和报告
  - 数据格式转换

- 根据以往经验和‘补充规约’内容确定要用到的构架机制
- 案例中用到的四种‘构架机制’
  - 安全：
  - 留存：
  - 分布处理
  - 信息交换

‘关键抽象’和‘分析机制’的映射关系

	安全	留存	分布处理	信息交换
报销单	X	X	X	
报销记录	X	X	X	
员工		X		
经理		X		
工资户头	X	X		X

- ‘构架机制’在模型中的位置
  - 在设计模型中添一个‘构架机制’包
  - 在‘构架机制’包中为每一个机制建立一　　相应的包



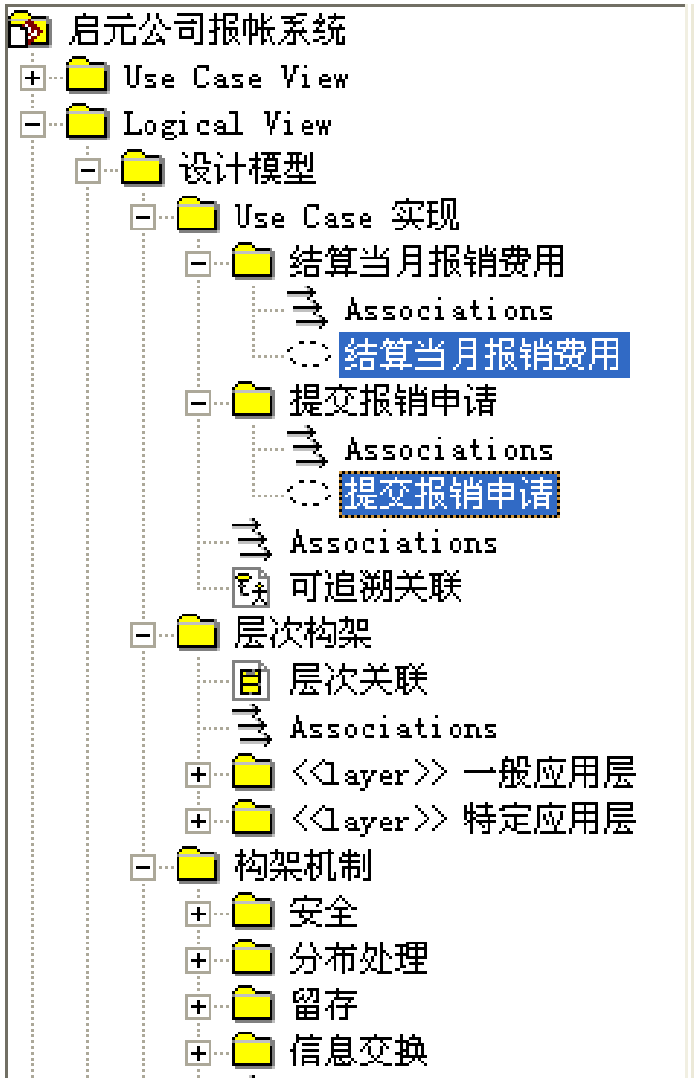
- 在各个‘构架机制’包中对相应机制的主要技术特征加以描述
- 安全：用户名要求，密码要求和权限要求
- 留存：粒度，容量，留存的持续时间
- 分布处理：分布的模式（Tiers 结构），时延，同步，消息粒度，协议等
- 信息交换：信息记录的格式，编码转换

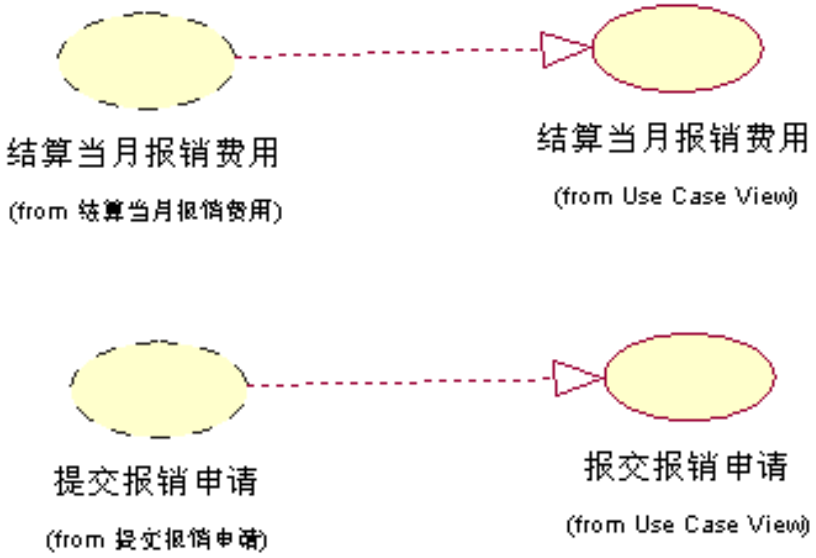
选定分析局部

- RUP 是一种迭代化的过程，每个迭代解决系统中选定的一个部分
- 迭代选择的单位是Use Case
- 风险是安排迭代内容的依据，先实现风险优先级高的 Use Case

- 建立Use Case和风险的对应表（see next slide）
- 选定当前的待分析局部
  - 在覆盖主要风险的前提下，选择尽可能少的 Use Case
    - 并非所有的Use Case都会影响拟建系统构架的关键部分
    - 众多非重点问题会分散对核心问题的注意力
  - Use Case‘提交报销申请’和‘结算当月报销费用’几乎覆盖了全部风险，并且它们只占全部用例数的很小一部分（2/9），故将其选为本次迭代的‘分析局部’
- 在设计模型中，建立名为‘Use Case实现’的包
  - 建立以当前迭代中命名的两个包，在每个包内建立相应的‘Use Case’实现
  - 建立‘可追溯关联’图

优先级 风险 \ Use Case		调整策略	结算当月报销费用	批复报帐申请	审核报帐申请	提交报销申请	提交贷款申请	完成日常维护	制作报表	登录系统
1	数据库的响应速度			X	X	X	X			
2	数据库的容量					X	X			
3	与内部邮件系统连接	X	X	X	X	X	X			
4	与银行系统连接		X		X					
5	与人事管理系统连接					X	X			X
6	内部网络有足够带宽			X	X	X	X			X
7	数据备份							X		


启元公司报帐系统
Use Case View
Logical View
设计模型
Use Case 实现
结算当月报销费用
Associations
结算当月报销费用
提交报销申请
Associations
提交报销申请
Associations
可追溯关联
层次构架
层次关联
Associations
<<layer>> 一般应用层
<<layer>> 特定应用层
构架机制
安全
分布处理
留存
信息交换


结算当月报销费用
(from 结算当月报销费用)
结算当月报销费用
(from Use Case View)
提交报销申请
(from 提交报销申请)
报交报销申请
(from Use Case View)

任务二： 局部分析

- ‘局部分析’以选定的Use Case为研究对象，以相对粗大的颗粒，用面向对象的概念和方法对问题进行转述，为后续以相对细小的颗粒做进一步的设计活动提供必要的基础
- 局部分析包括三项活动（‘点，线，面’）
  - 提取分析类：以‘关键抽象’和‘Use Case’的文字内容为基础，从三个维度提取能够协作完成该‘Use Case’行为的分析元素（分析类）
  - 转述需求场景：用‘分析类’的实例为载体，通过消息传递的方式实现对‘Use Case’的分解
  - 整理分析类：根据‘分析类’的实例在‘Use Case’需求场景中的消息传递关系，归纳‘分析类’所承担的‘责任’和分析类‘之间的关系

提取分析类

  - ’分析类‘是概念层面上的内容，与应用逻辑直接相关
  - ’分析类‘直接针对软件的功能需求，其行为来自Use Case的内容
  - ‘分析类’是用于描述拟建系统中那些较高层次的对象，不考虑与应用逻辑不直接相关的细节

- ’分析类‘的类型划分
  - 实践经验表明，软件系统往往容易在三个维度发生变化
    - 拟建系统和外部要素之间交互的边界
    - 拟建系统在运行中的控制逻辑
    - 拟建系统要记录和维护的信息
  - 按照这三个维度进行’分析类‘的类型划分，可将未来的变化限制在较小的范围，形成较稳定的系统结构
- 边界类的特征
  - 描述三种类型的内容，用户界面，与外部系统的接口，和与设备的接口
  - 与外部系统的接口主要关注通信协议
  - 用户界面主要关注交互的内容（不是具体窗口构建）
- 控制类的特征
  - 控制类常用来描述一个Use Case所特有的事件流控制能力
  - 控制类将Use Case所特有的行为进行封装
- 实体类的特征
  - 用于描述系统必须存储的信息
  - 同时描述与此信息相关的行为
    - Information Expert 模式（在对象间分配行为时可供参考的一个模式）

- 提取分析类的步骤
  - 第一步：充实Use Case内容：如果Use Case没有明确定义系统必须做什么才能响应外部的要求
  - 第二步：提取分析类
- 通常一个Actor和Use Case之间的通信关联对应一个边界类
- 通常一个Use Case对应一个控制类
- 实体类的主要来源
  - ’关键抽象‘
  - 当前Use Case的文字描述
  - 有些Actor也对应系统内部的实体类
- 分析类在模型中的（安放）位置
  - 控制类 ＝》特定应用层
  - 边界类（用户界面） ＝》特定应用层
  - 边界类（外部系统接口） ＝》一般应用层
  - 实体类（分析局部中得到的）＝》特定应用层
- 从案例的Use Case中提取分析类

用例名称：　提交报销申请【控制，SubmitClaim】

参与执行者：　公司员工

入口条件：　员工成功登录系统，通过身份验证。被系统提示进入‘报销申请’或者‘借款申请’功能

事件流程：

1. 打开报销单

【员工】员工【实体，关键抽象，Employee】选择进入‘报销申请’　【边界，SubmitClaimForm】功能

【系统】如果该员工当月报销单存在，系统将取出响应信息并展示给员工

2. 添加报销记录

【员工】员工要求添加一条报销记录【实体，关键抽象，ClaimRecord】

【系统】系统显示一条空白报销记录

3. 填写报销记录

【员工】员工开始填写报销记录，每条报销记录包括的信息有：业务活动发生的时间，地点，客户名称，（可选），原因以及费用金额和种类（交通，餐饮，会议，通信和杂项）。

【系统】系统显示并记录员工输入的信息。为了让员工方便而准确地输入相关信息，除了客户名称，业务活动原因和金额之外，其它信息域提供相应的下拉式选择列表。

（重复以上针对每一条报销记录的活动，直至所有记录填写完毕）

4. 验证报销单

【员工】员工填写完所有的报销记录之后，要求系统验证这些记录的合理性【实体，Valid_Rule】

【系统】报销记录的初始状态为‘未验证’，当每一条报销记录被验证为合理，系统将该报销记录的状态设为‘已验证’，系统在验证所有报销记录（为‘已验证’）之后提示用户可以提交本月的报销单。验证为合理的记录必须满足几种条件：第一，不同种类的费用不超过相应的限额；第二，报销费用的类型要和员工的职能匹配

5. 提交报销单

【员工】所有报销记录经过验证后，员工提交当月的报销单。

【系统】系统保存这张报销单，将报销单的状态设置为‘已提交’并记录

提交日期，同时这张报销单被设为‘只读’。系统要从人事管理数据库【边界，HRDatabase】中获知该员工及其经理（负担该员工当月开销者）的电子邮件地址。为了及时通知相关人员，系统将自动生成一份以当前报销单为内容的电子邮件发送到该员工及其经理的信箱中（邮件系统【边界，MailSystem】 ）。当邮件发后，员工得到一个确认信息。

备选事件序列组（Alternative Flows）

A1. 创建当月报销单

【起始位置】基本事件序列中，员工进入报销申请程序并准备打开当月报销单。

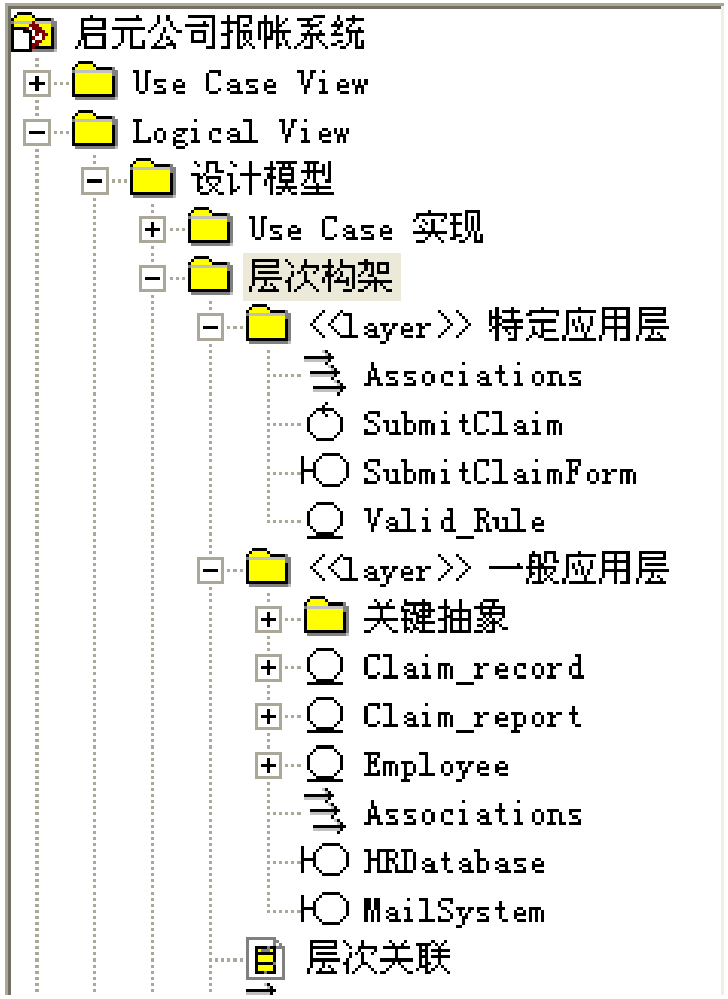
【触发条件】该员工的当月报销单不存在。

【具体内容】系统为该员工创建一张当月报销单

【返回位置】基本事件序列中的’打开报销单‘步骤

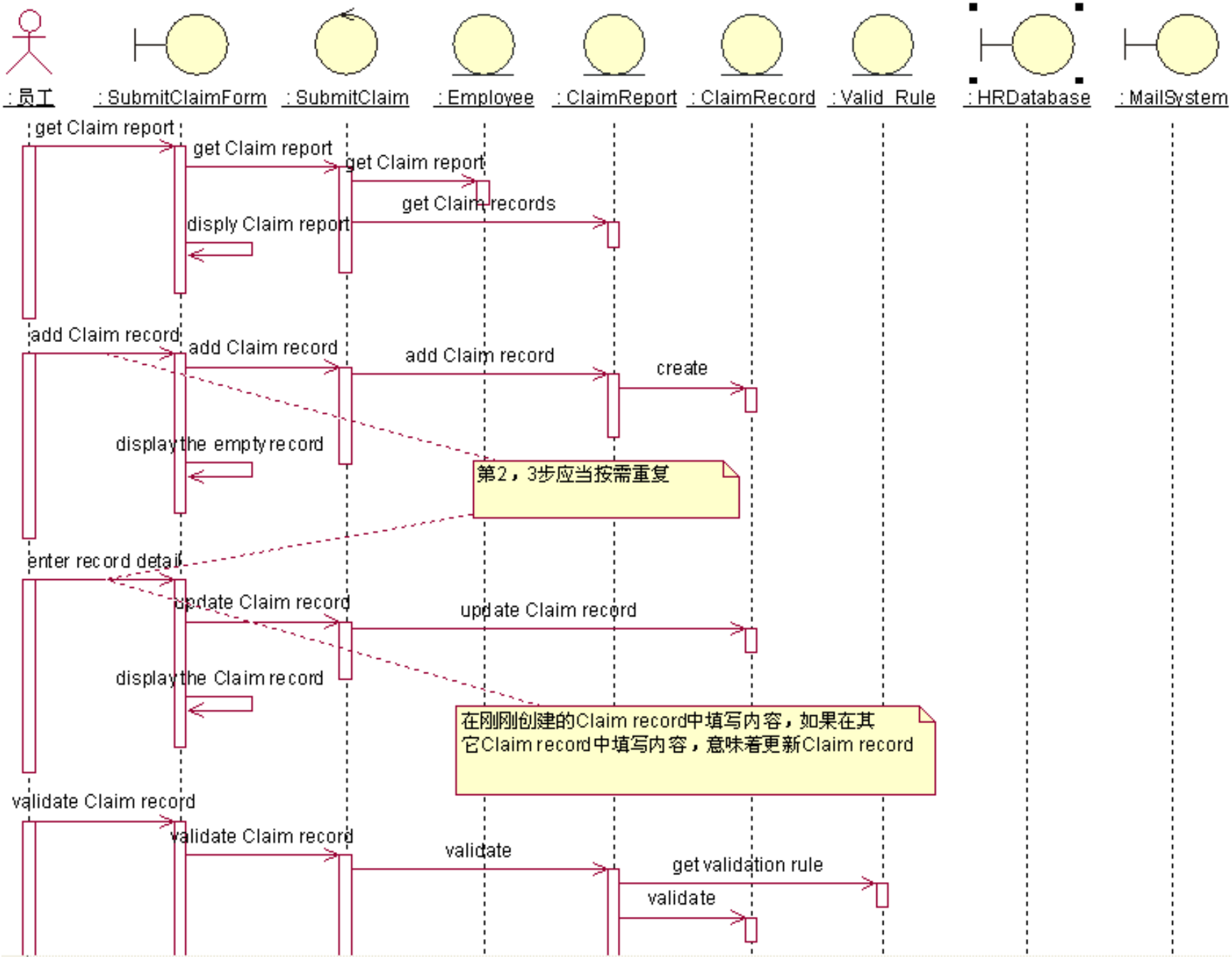
… （略）

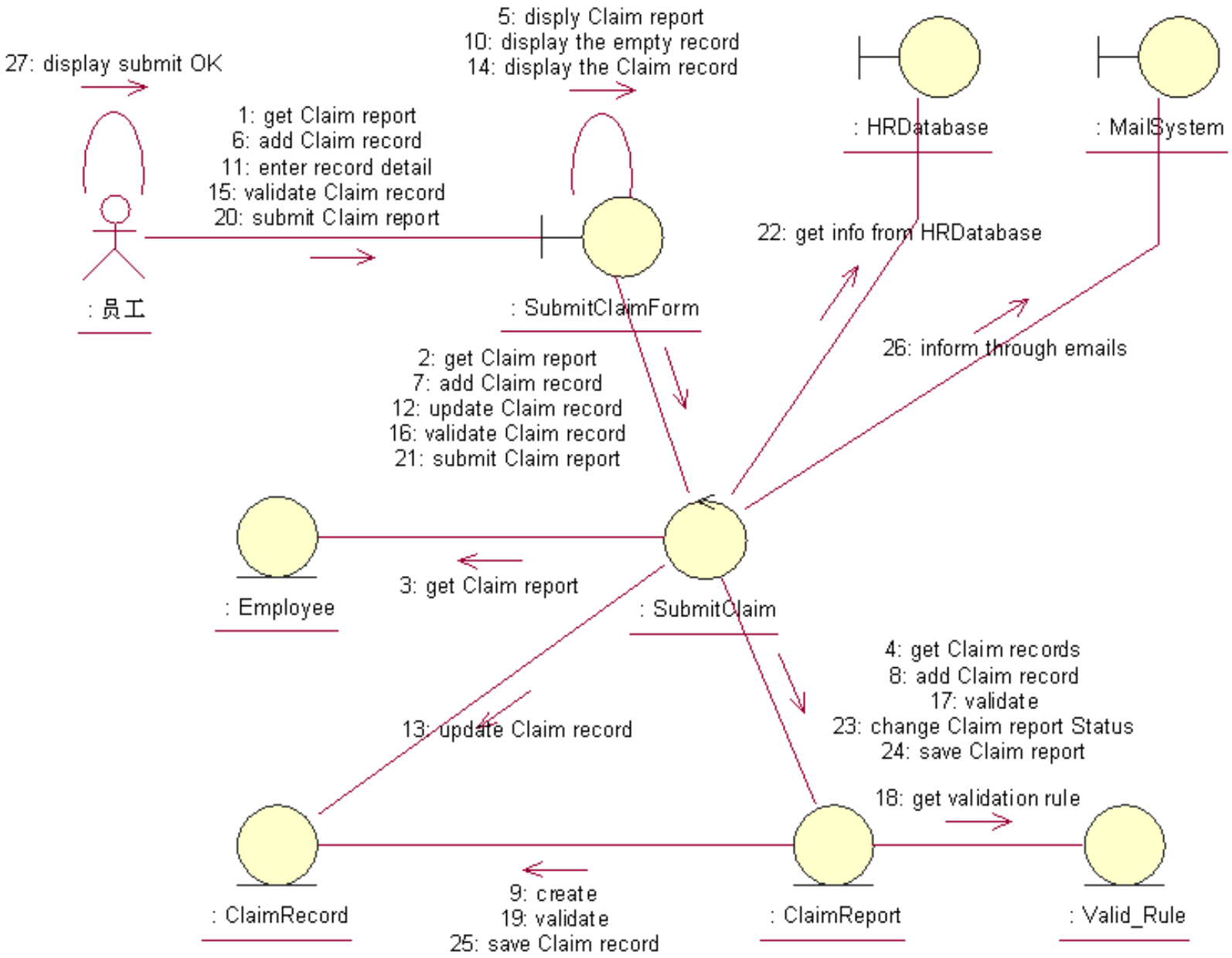
- 将提取的分析类放入层次结构中的适当层中
  - 一般应用层：外部接口边界类
  - 特定应用层：控制类，用户界面的边界类，从特定’分析局部‘中识别出的实体类

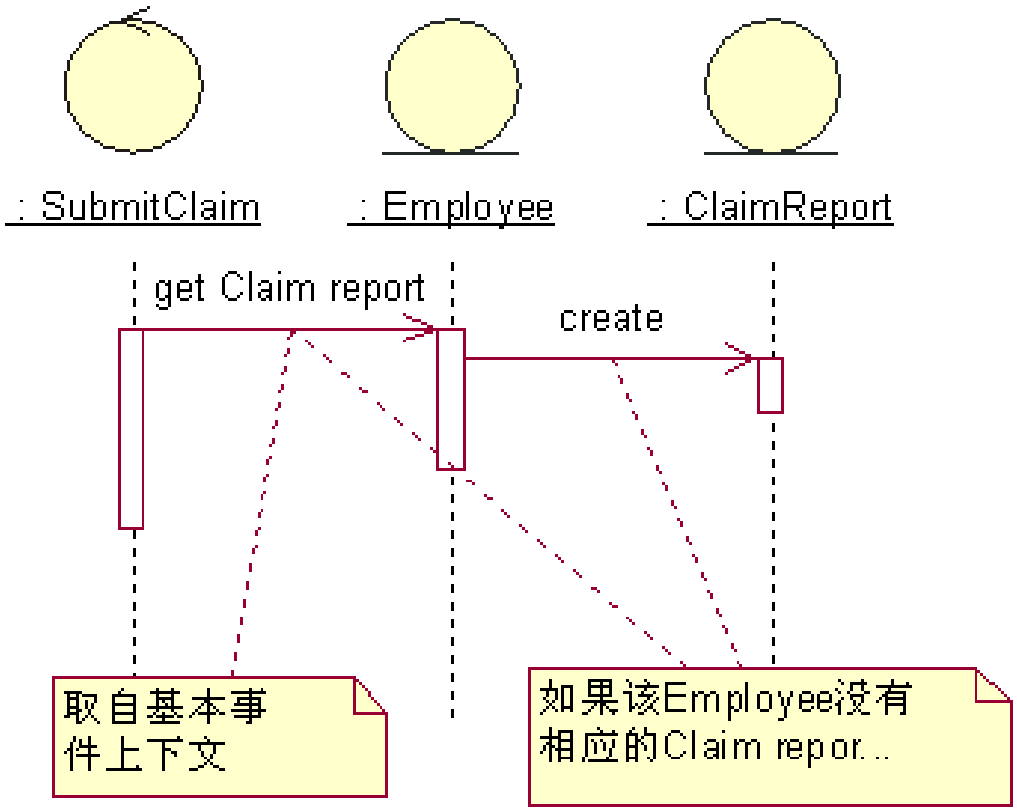


转述需求场景

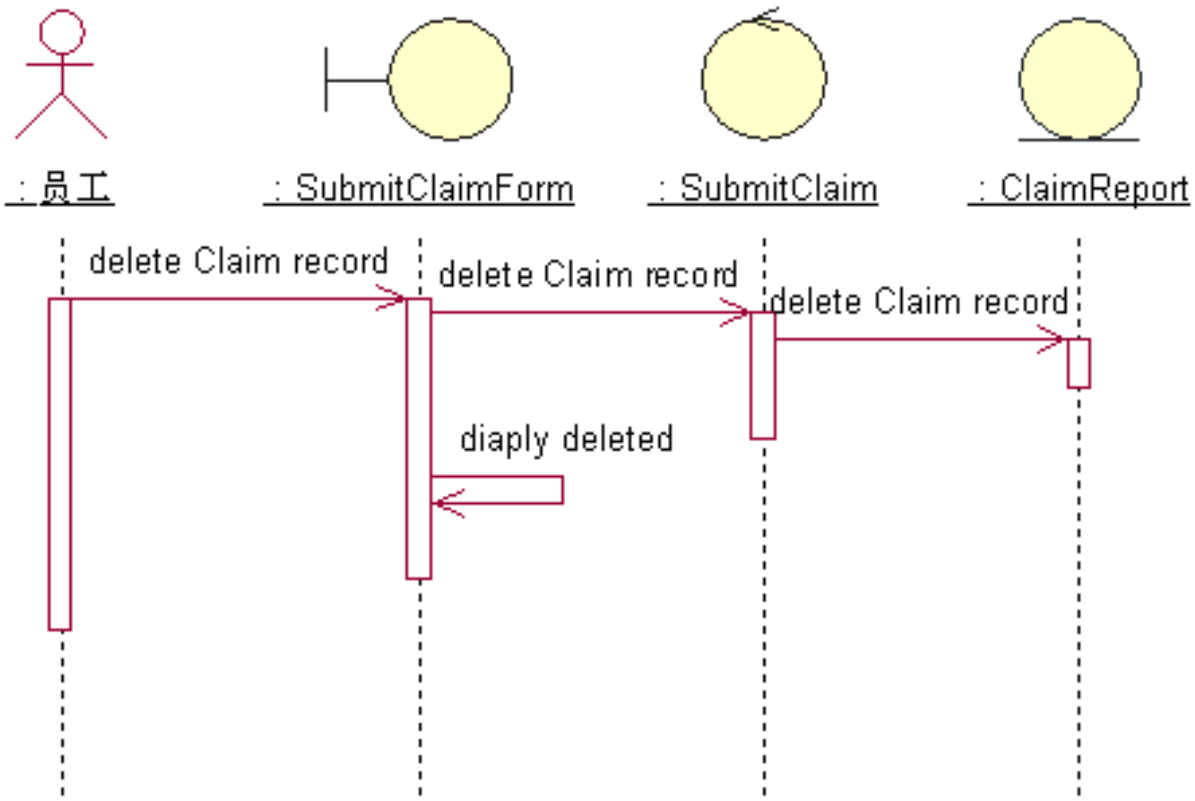
- 将Use Case描述的功能交给一组识别出的分析类去完成
- 通过发消息的方法在相关分析类间分配责任
- 步骤一：绘制序列图
  - 在对象间用消息传递的方式将事件序列的内容复述出来
- 步骤二：找出对象传递消息的通道
  - 两对象可传递消息的前提是它们之间存在一个连接
  - 对象间的连接隐含这对应类之间的关联关系


:员工
: SubmitClaimForm
: SubmitClaim
: Employee
: ClaimReport
: ClaimRecord
: Valid_Rule
: HRDatabase
: MailSystem
get Claim report
get Claim report
get Claim report
get Claim records
disply Claim report
add Claim record
add Claim record
add Claim record
create
display the empty record
第2，3步应当按需重复
enter record detail
update Claim record
update Claim record
display the Claim record
在刚刚创建的Claim record中填写内容，如果在其
它Claim record中填写内容，意味着更新Claim record
validate Claim record
validate Claim record
validate
get validation rule
validate

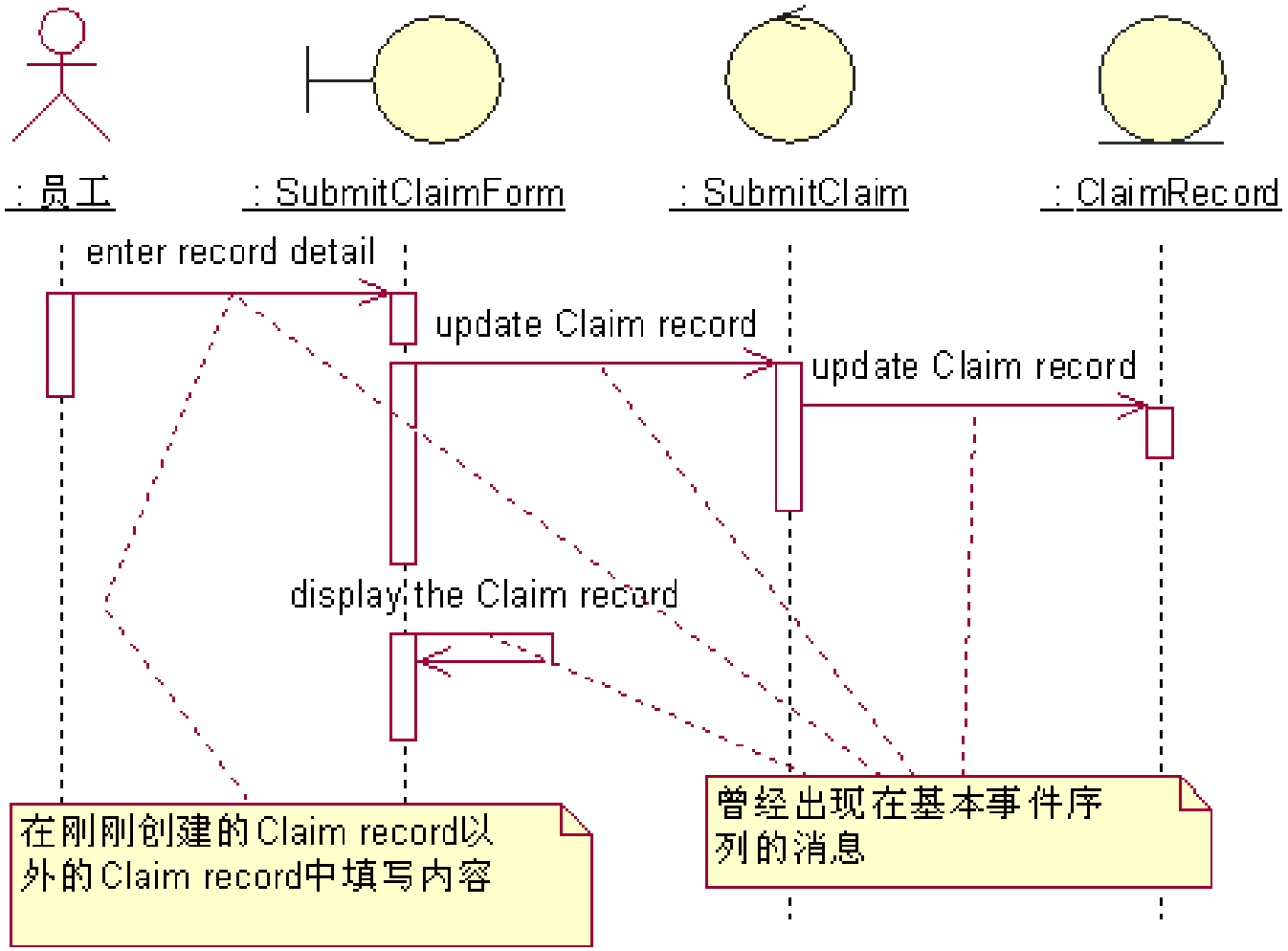

27: display submit OK
5: disply Claim report
10: display the empty record
14: display the Claim record
: HRDatabase
: MailSystem
1: get Claim report
6: add Claim record
11: enter record detail
15: validate Claim record
20: submit Claim report
22: get info from HRDatabase
: 员工
: SubmitClaimForm
26: inform through emails
2: get Claim report
7: add Claim record
12: update Claim record
16: validate Claim record
21: submit Claim report
: Employee
3: get Claim report
: SubmitClaim
4: get Claim records
8: add Claim record
17: validate
23: change Claim report Status
24: save Claim report
13: update Claim record
18: get validation rule
: ClaimRecord
9: create
19: validate
25: save Claim record
: ClaimReport
: Valid_Rule



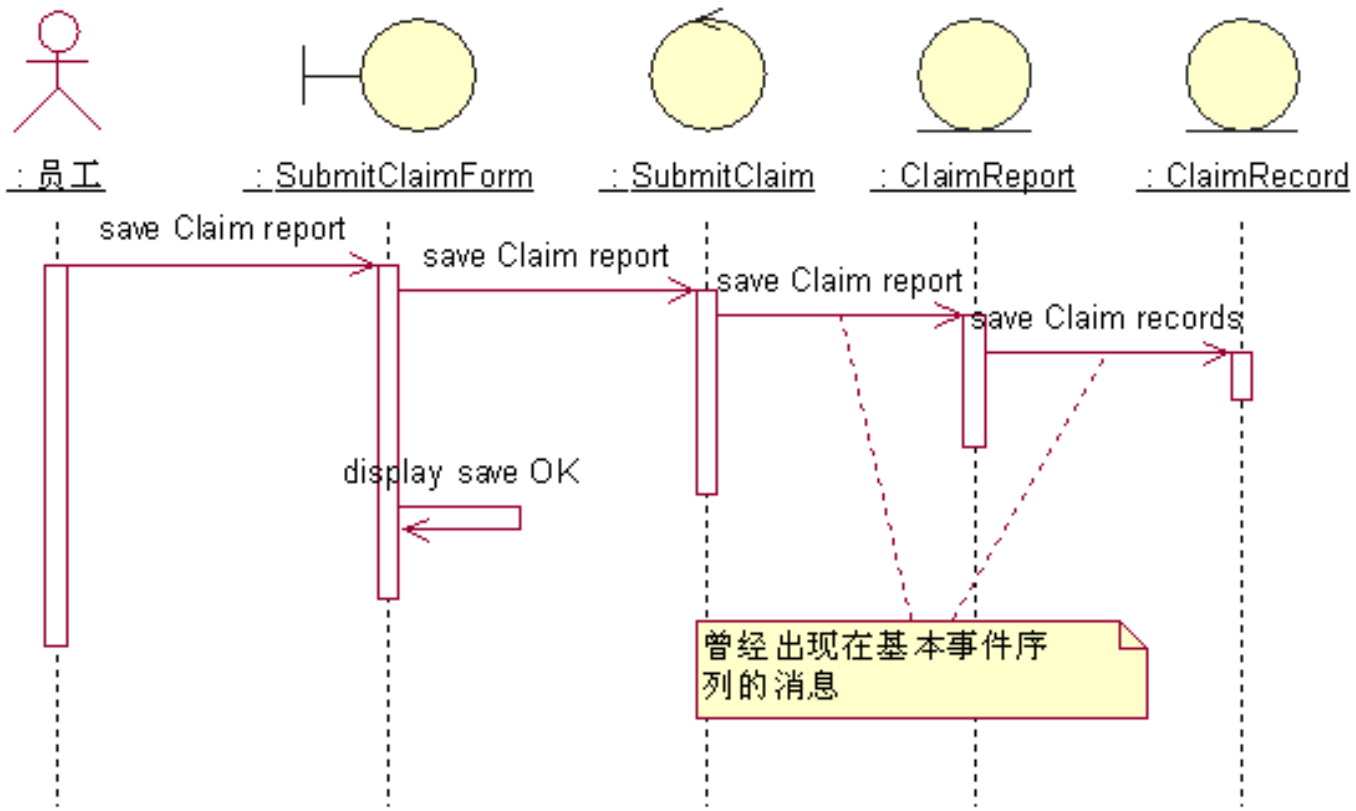
A1. 备选事件序列（建立ClaimReport）



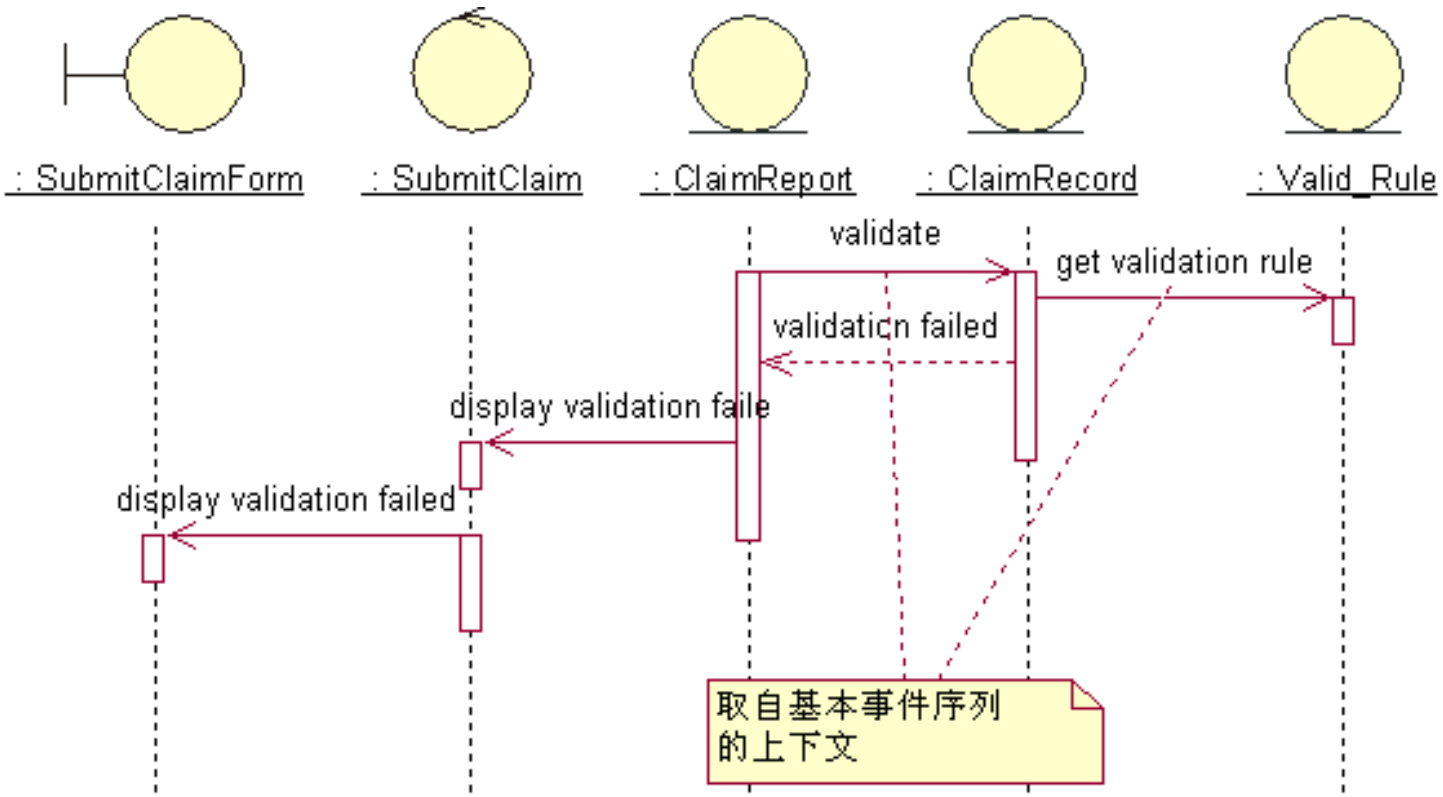
A2. 备选事件序列（删除ClaimReport）



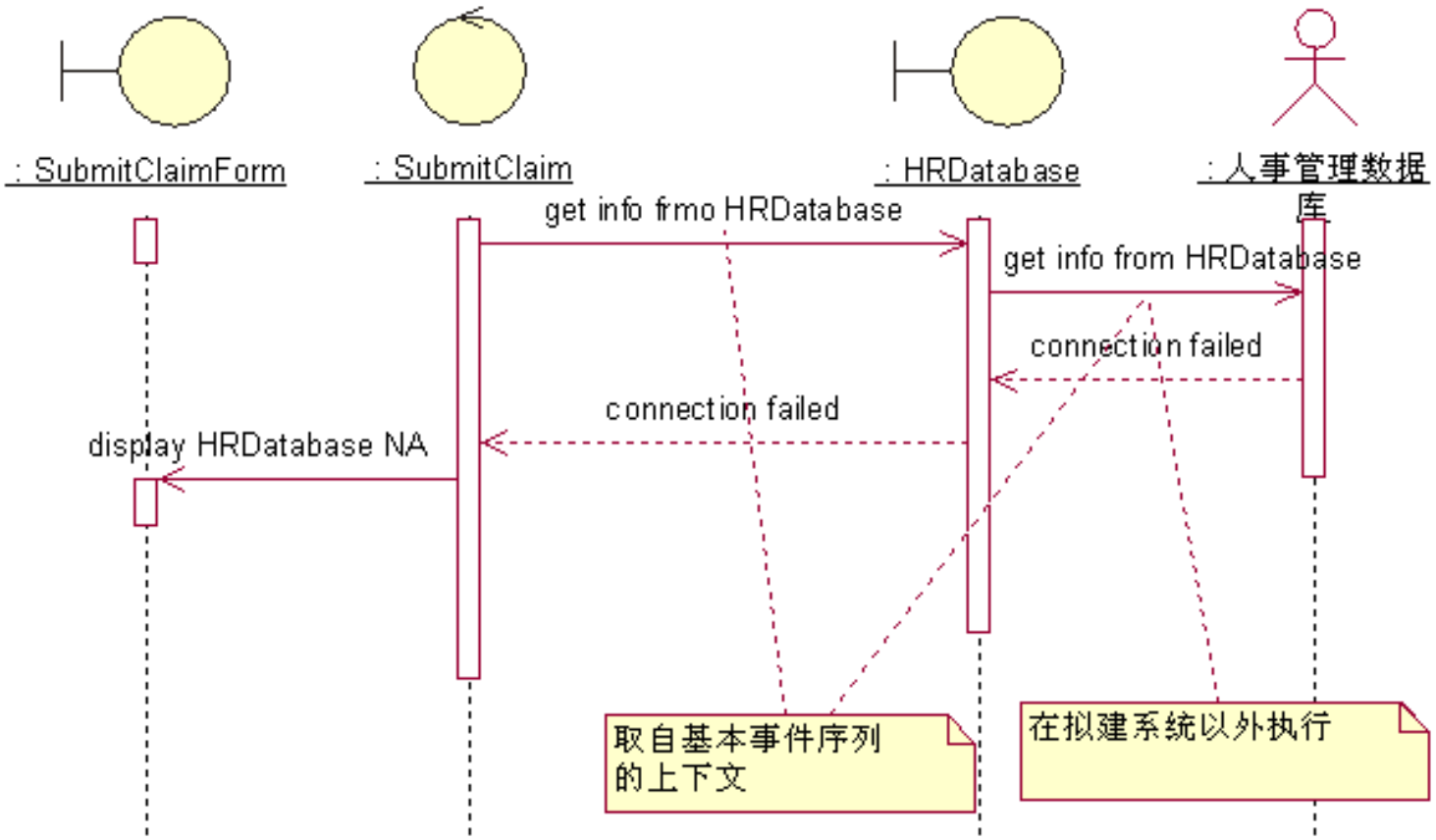
A3. 备选事件序列（更新ClaimReport）



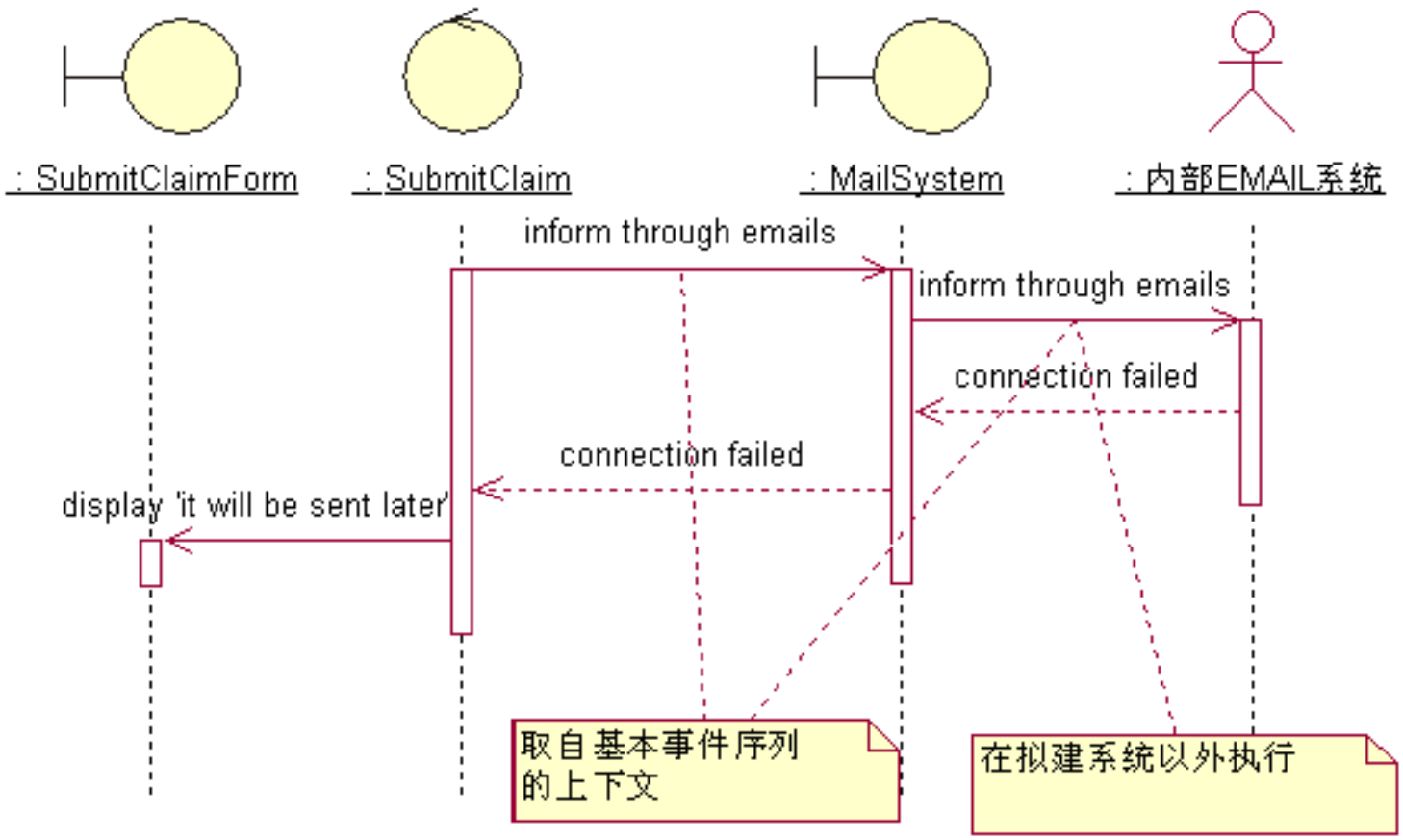
A4. 备选事件序列（保存ClaimReport）



A5. 备选事件序列（ClaimRecord不合理）



A6. 备选事件序列（HRDatabase不可用）



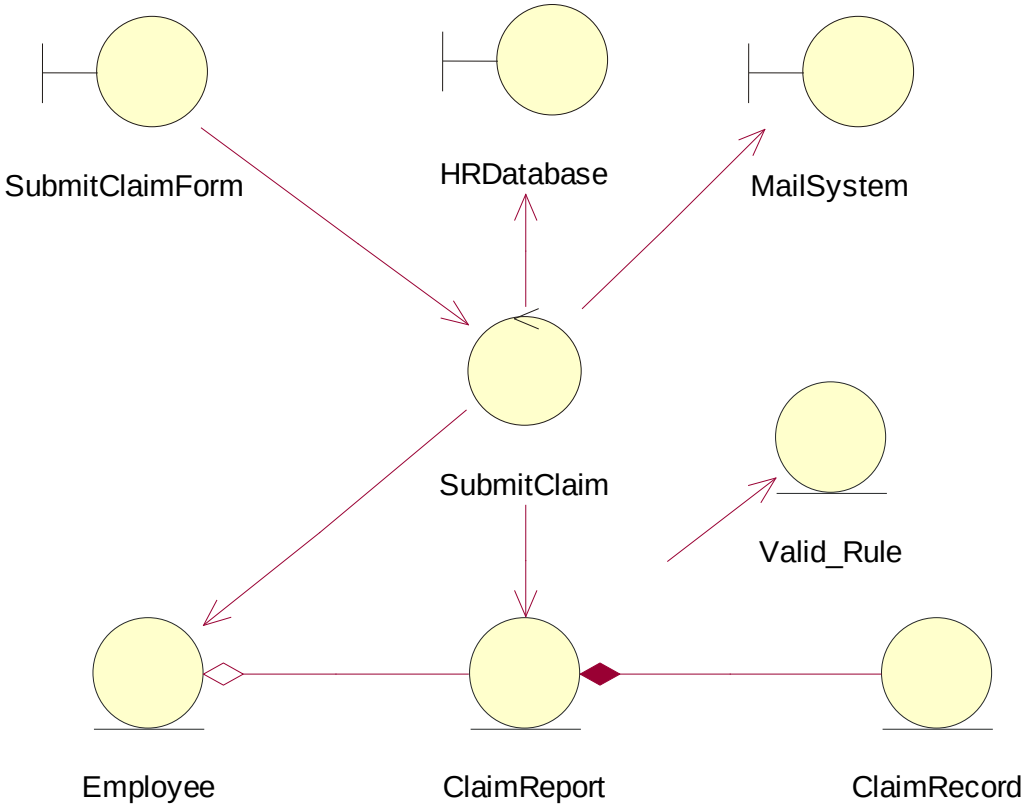
A7. 备选事件序列（MailSystem不可用）

- 对以上得到的初步交互图组的改进
  - A1备选事件序列（很简单）可以并入基本事件序列（附以相应说明）
  - A3备选事件序列虽然较A1复杂，但其事件序列与基本事件序列重合，亦可并入基本事件序列（附以相应说明）
  - A3序列中SubmitClaim 实例和 ClaimRecord实例之间的连接并不是必要的，利用ClaimReport接力可降低对象间的耦合
  - 根据‘在情况允许时，尽量使序列图中的消息指向右方’的建议，对A5，A6，和A7进行调整

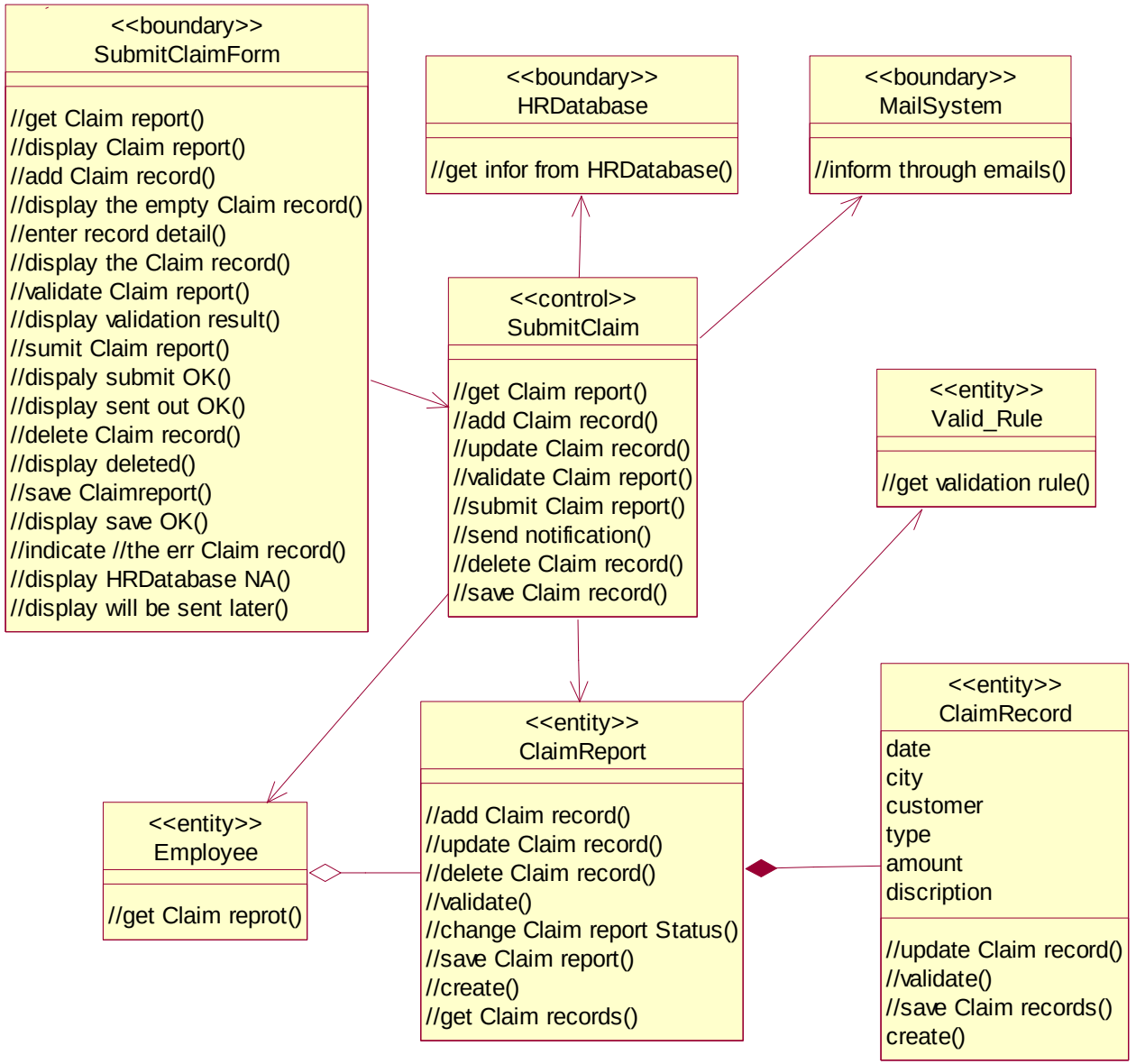
整理分析类

- 相关概念
  - 分析类的责任和关联关系：消息被映射为分析类的责任（责任是响应消息的能力），消息的传递路径被映射成分析类之间的关联关系
  - 分析类的属性：描述分析类为履行责任而应具备的知识
  - 参与类图：由参与特定Use Case的对象对应的类及其关联关系组成，一般由顺序图导出

- 步骤一：确定分析类的责任
  - 根据消息和责任的对应关系确定分析类的责任
- 步骤二：确定分析类间的关联关系
  - 在特定的Use Case实现中添加参与类图
  - 将所有参与该Use Case实现的类加到图中
  - 将从分析类实例交互图中识别的关联关系加到图中
- 步骤三：确定分析类的属性
  - 基本来源是 Use Case 的事件流描述



提交报销申请（Use Case）参与类图（简略）



提交报销申请（Use Case）参与类图（详细）

任务三： 全局设计

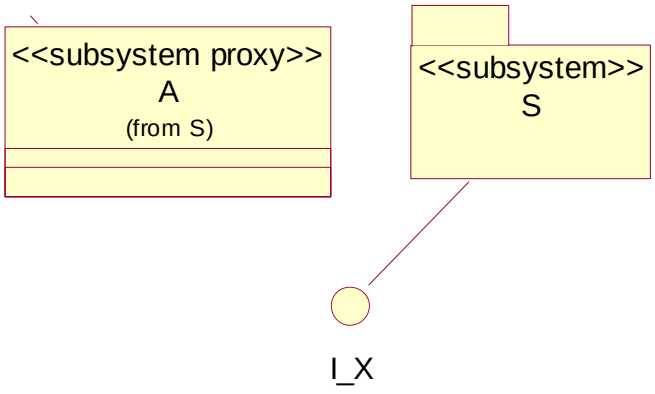
- 全局设计包括三项活动
  - 确定核心元素：以分析类为出发点，确定相应的设计元素（设计类，子系统接口）
  - 引入外围元素：以分析机制为出发点，确定能满足相关分析类要求的设计机制
  - 优化组织结构：按高内聚，低耦合的原则，整理充实起来的层次构架内容
- 相关概念
  - 核心设计元素：分设计类，和子系统接口两种
    - 设计类承担那些对应于独立构件的分析类之责任集合
    - 子系统接口定义了那些对应于复合构件（即子系统）的分析类之责任集合
    - 分析类解决‘要做什么事情（what）’
    - 设计类解决‘怎么做才切合实际（how）’

确定核心元素

- 步骤一：映射分析类到设计元素
  - 如果一个分析类比较简单，可直接映射为设计类
    - 例如：主导 Actor 对应的边界类，控制类，和一般的实体类

- 如果一个设计类的责任比较复杂，应将其映射为子系统接口
- 被动 Actor 对应的边界类，一般被映射为子系统接口

- 步骤二：定义子系统接口
  - 命名和简述子系统接口
    - 例：名字加前缀 ‘I_’，设计类用《Interface》构造型标记
  - 描述拟建子系统的行为
    - 准确定义操作的集合
    - 操作的描述：名字，返回值，参数及类型，和操作应该作什么（的描述）
  - 子系统，子系统接口，和子系统代理
    - 子系统接口是子系统使用者和实现者之间的一个合约（只有逻辑意义）
    - 子系统物理上实现子系统接口
    - 子系统代理（Façade 模式）物理上实现了子系统接口，并是子系统的一部分
    - 有些子系统是基本是由第三方软件（通常是经过调整后）实现的



案例中分析类到设计元素的映射

构造型	分析类	设计元素
<<Interface>>	SumnitClaimForm	’设计类‘ SumnitClaimForm
	HRDatabase	’子系统接口‘ I_ HRDatabase
	MailSystem	’子系统接口‘ I_ MailSystem
	…	…
<<Control>>	SubmitClaim	’设计类‘ SubmitClaim
	…	
<<Entity>>	Employee	’设计类‘ Employee
	ClaimReport	’设计类‘ ClaimReport
	ClaimRecord	’设计类‘ ClaimRecord
	ValidRule	’设计类‘ ValidRule
	…	…

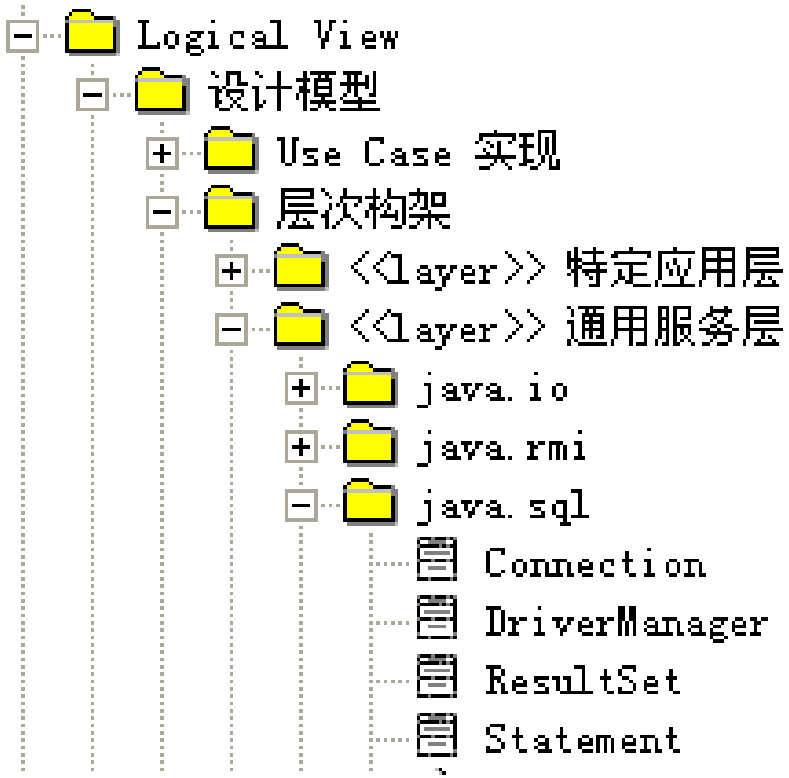
引入外围元素

- 相关概念
  - 基础设计元素：预定义好的，存在于各种类库中的设计元素
  - 衔接设计元素：不是预定义好的，但在不同应用场合的形式和内容总是大同小异的设计元素
  - 外围设计元素：与拟建系统的应用逻辑不直接相关，辅助核心设计元素具体地利用分析机制所概括的基本服务。外围设计元素基本由基础设计元素和衔接设计元素组成
  - 设计机制与实施机制：设计机制用于实现分析机制，在分析机制的框架中添加具体内容。实施机制运用特定的实施技术实现相应的设计机制。设计机制描述了如何利用基础设计元素和衔接设计元素实现核心设计元素所需要的那些与应用逻辑不直接相关的必要服务。设计机制的关键内容是衔接设计元素的使用方法
- 步骤一：分析机制向设计机制映射
  - 在全局分析中得到，关键抽象和分析机制的关联表
  - 扩展后可得到核心设计元素和分析机制的关联表
  - 一种分析机制可能会映射成几种设计机制
    - 例：‘留存机制’ ＝》内存，文件，或数据库

- 步骤二：落实设计机制的具体内容
  - 引入特定设计机制中涉及的基础设计元素，放入系统层次构架的通用服务层中
  - 为每个设计机制建立一个独立的包，在包中建立相关核心设计元素的适配器以及衔接元素的‘模子’
  - 用参与类图和序列图描述设计机制的静态和动态协作关系

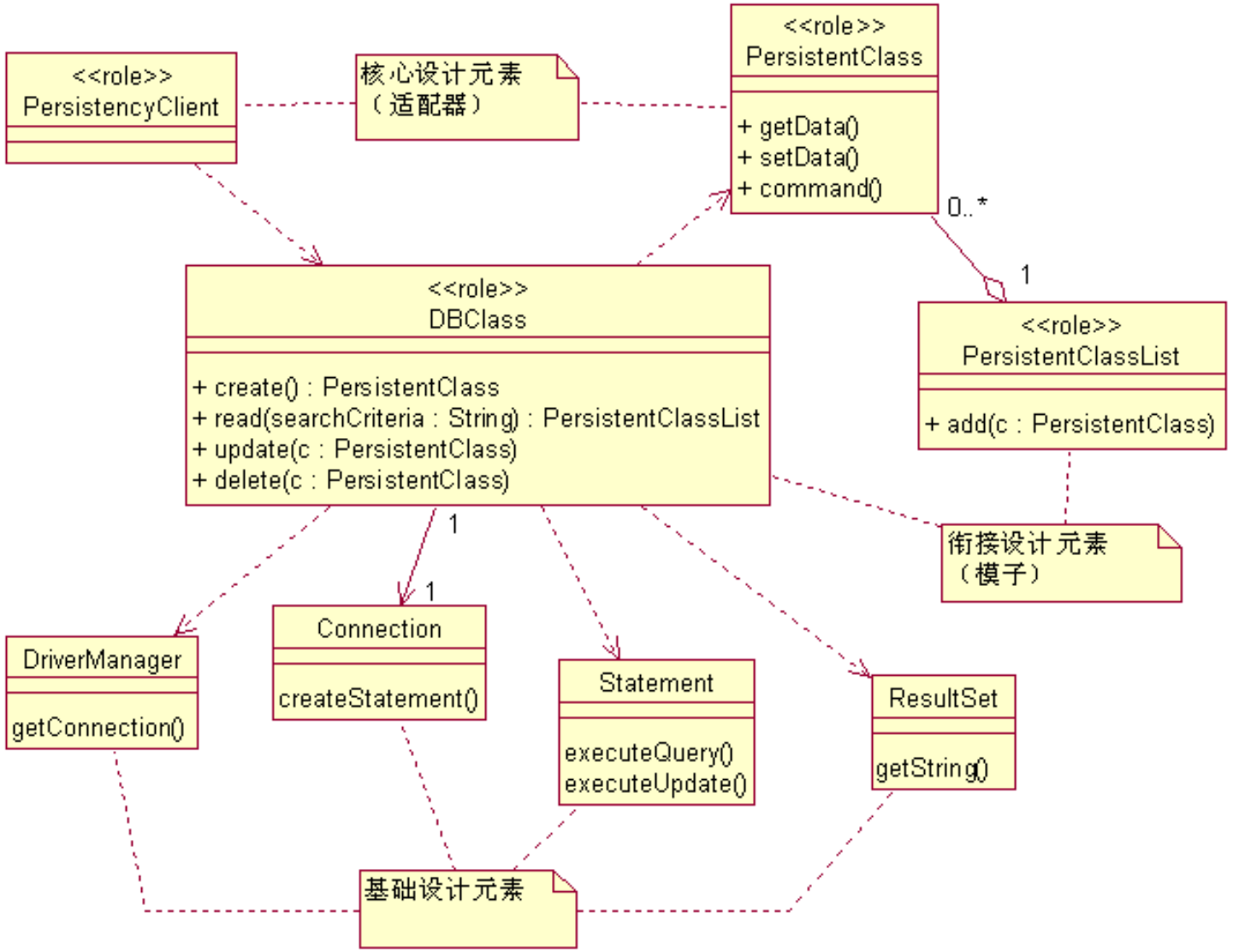
分析机制	设计机制	实施机制	使用者（核心设计元素）
留存	RDBMS	JDBC	HRDatabase Employee ClaimReport ClaimRecord ValidRule …
	…	…	…
分布处理	RMI	Java 1.3	SubmitClaim …
	…	…	…
…	…	…	…

- 留存机制的描述
  - 引入基础元素

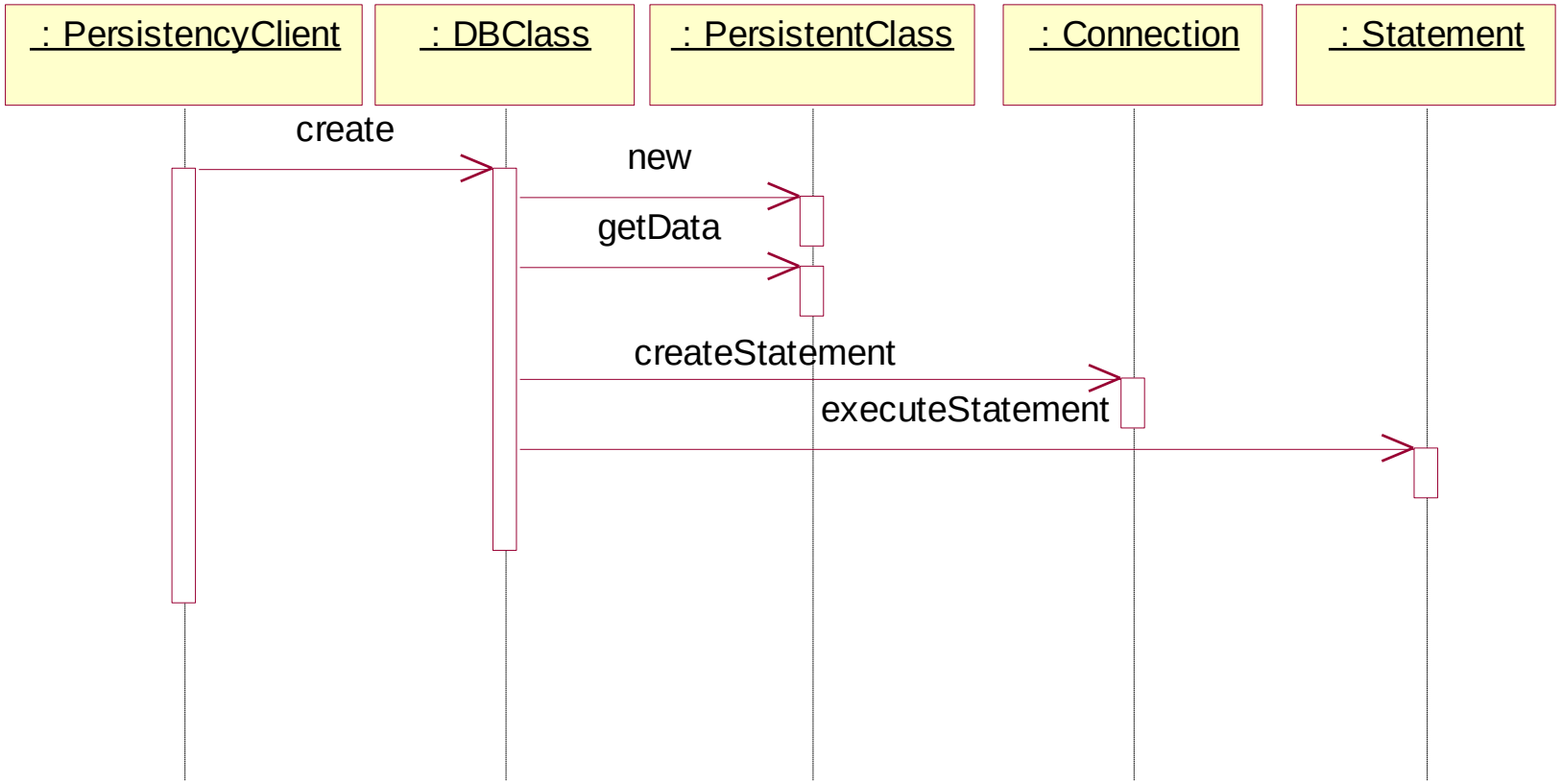


- 留存机制的描述
  - 在构架机制包种的留存包里，建立名为 RDBMS-JDBC的 包

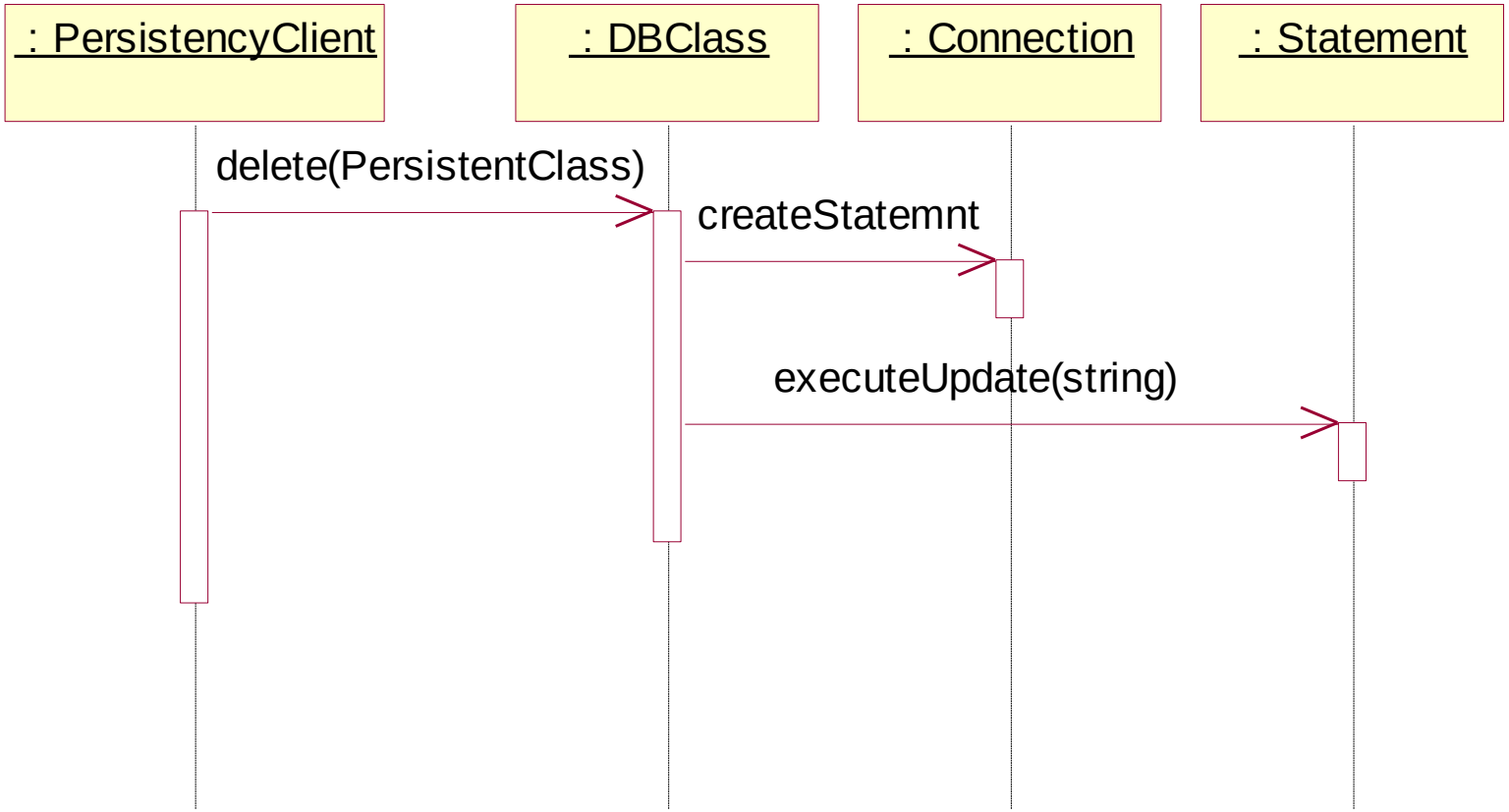
- 留存机制的描述
  - 静态机制的描述



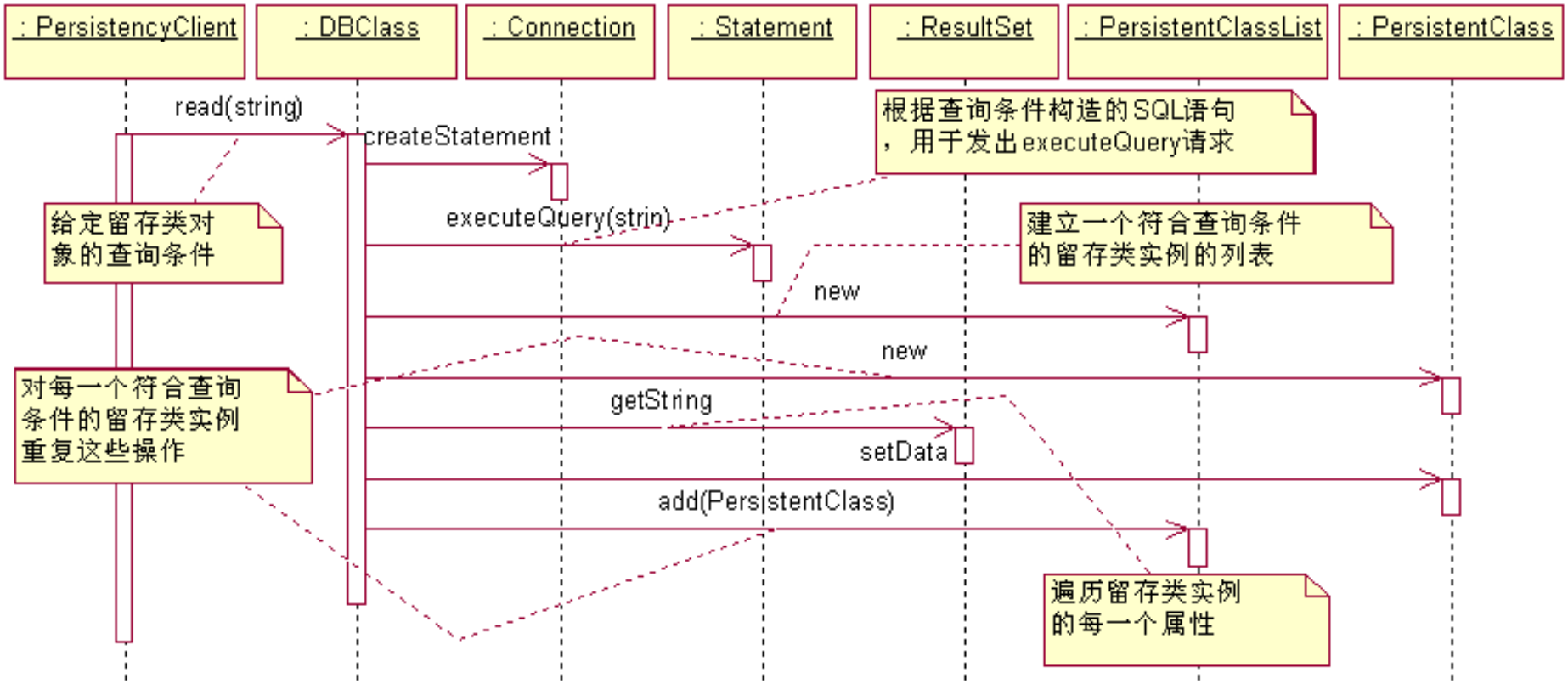
- 典型应用场景
  - 创建留存类的实例



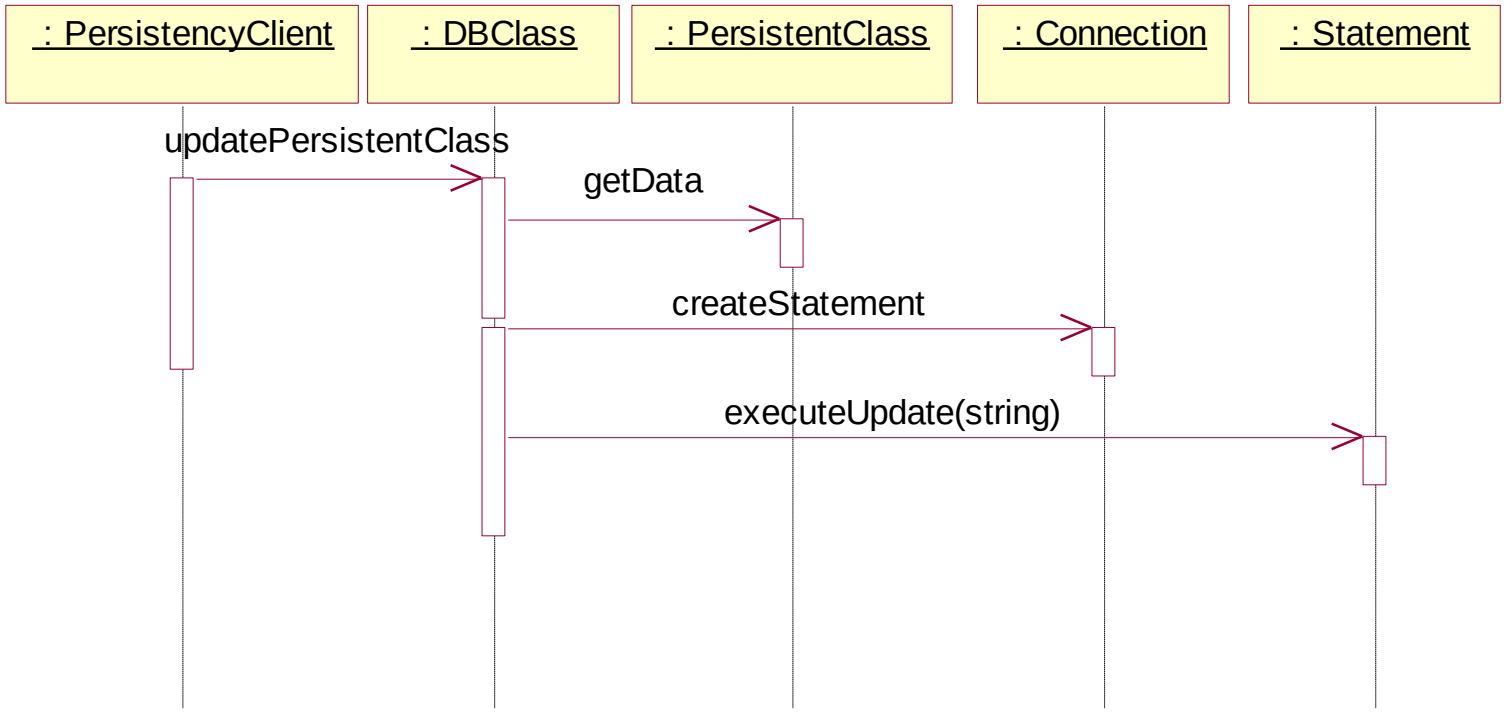
- 典型应用场景
  - 删除留存类的实例



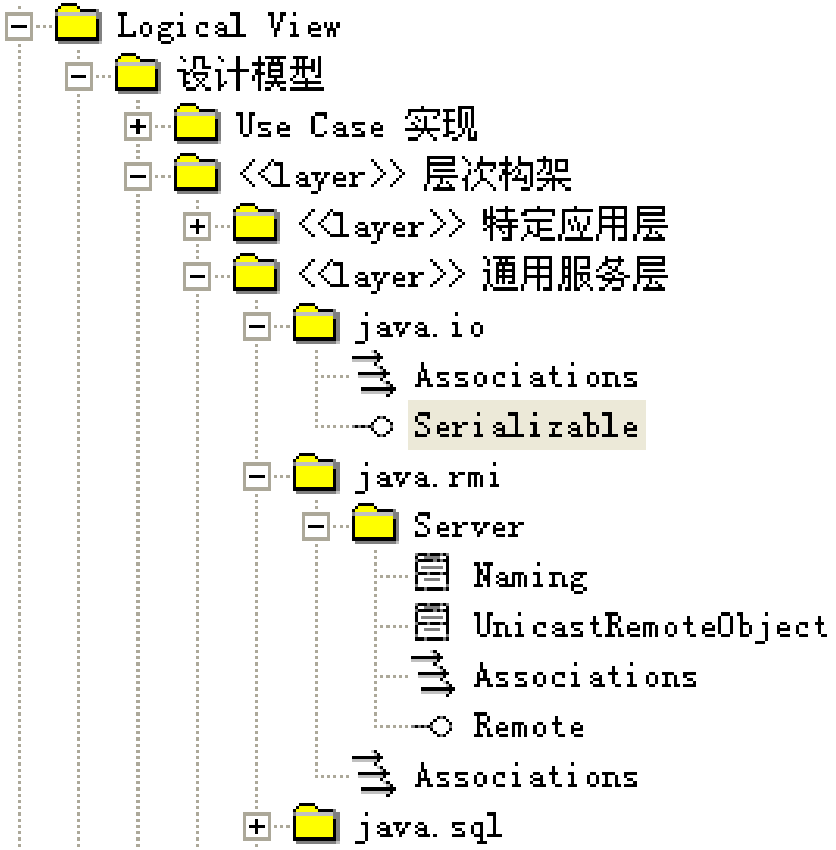
- 典型应用场景
  - 读取留存类的实例



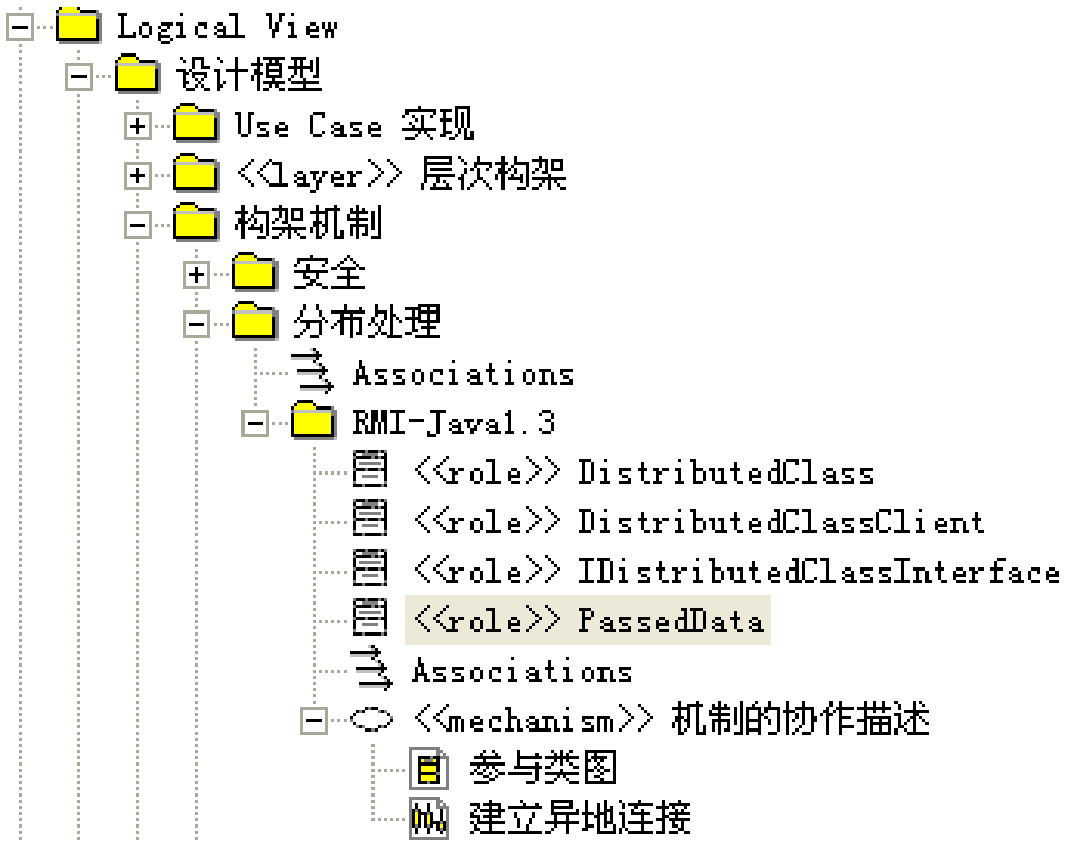
- 典型应用场景
  - 更新留存类的实例



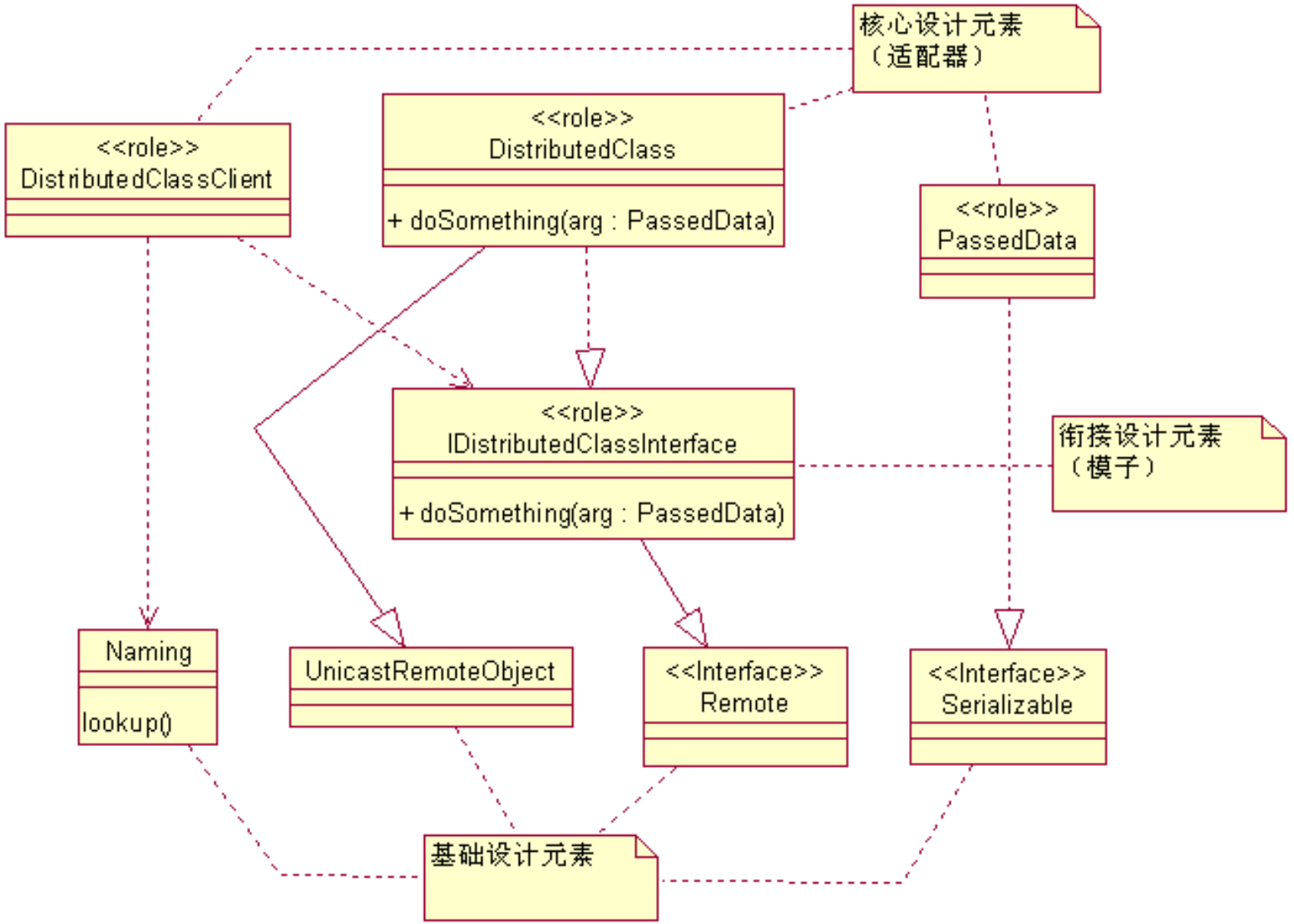
- 分布处理机制的描述
  - 引入基础元素



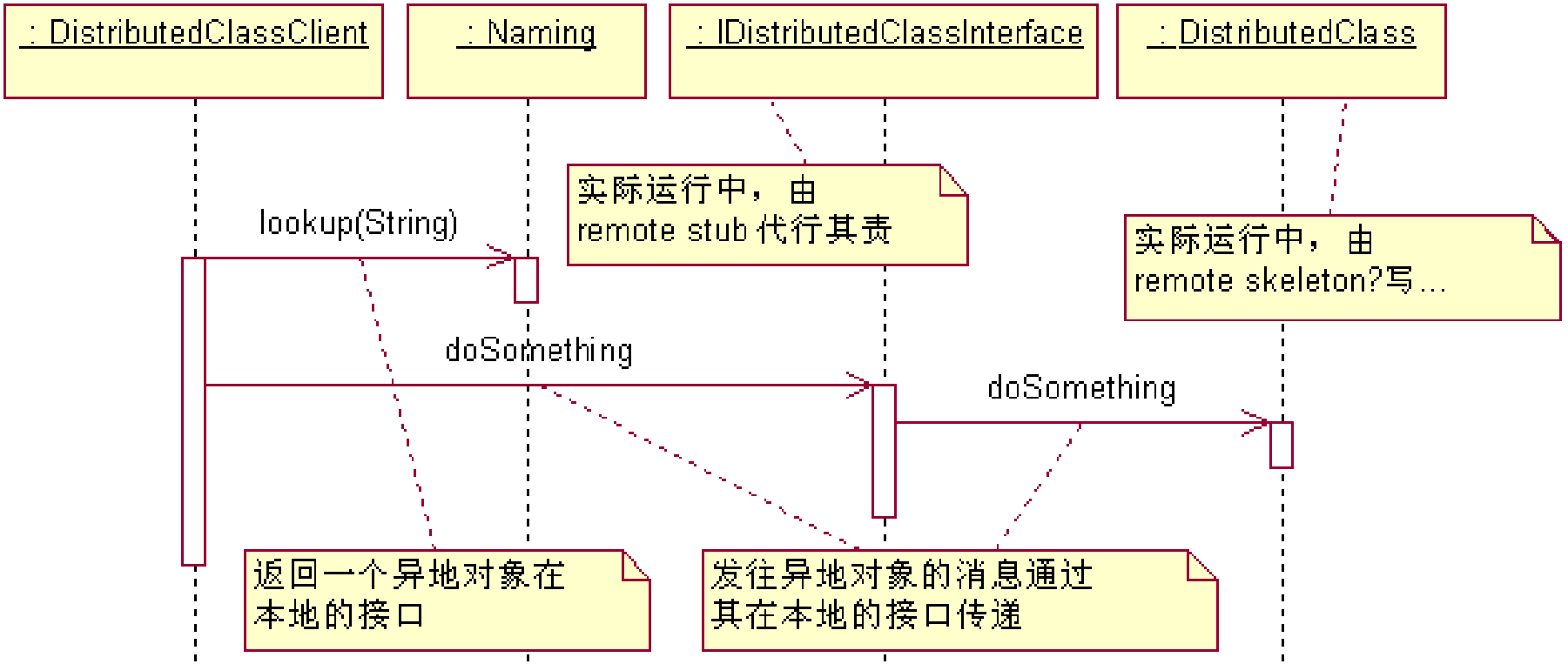
- 分布处理机制的描述
  - 在构架机制包的分布处理包中，建立一个名为 RMI-Java1.3 的包
  - 建立相关核心设计元素的适配器以及衔接元素的模子



- 分布处理机制的描述
  - RMI实现分布处理机制的静态结构



- 分布处理机制的描述
  - RMI实现分布处理机制的动态结构，呼叫异地对象



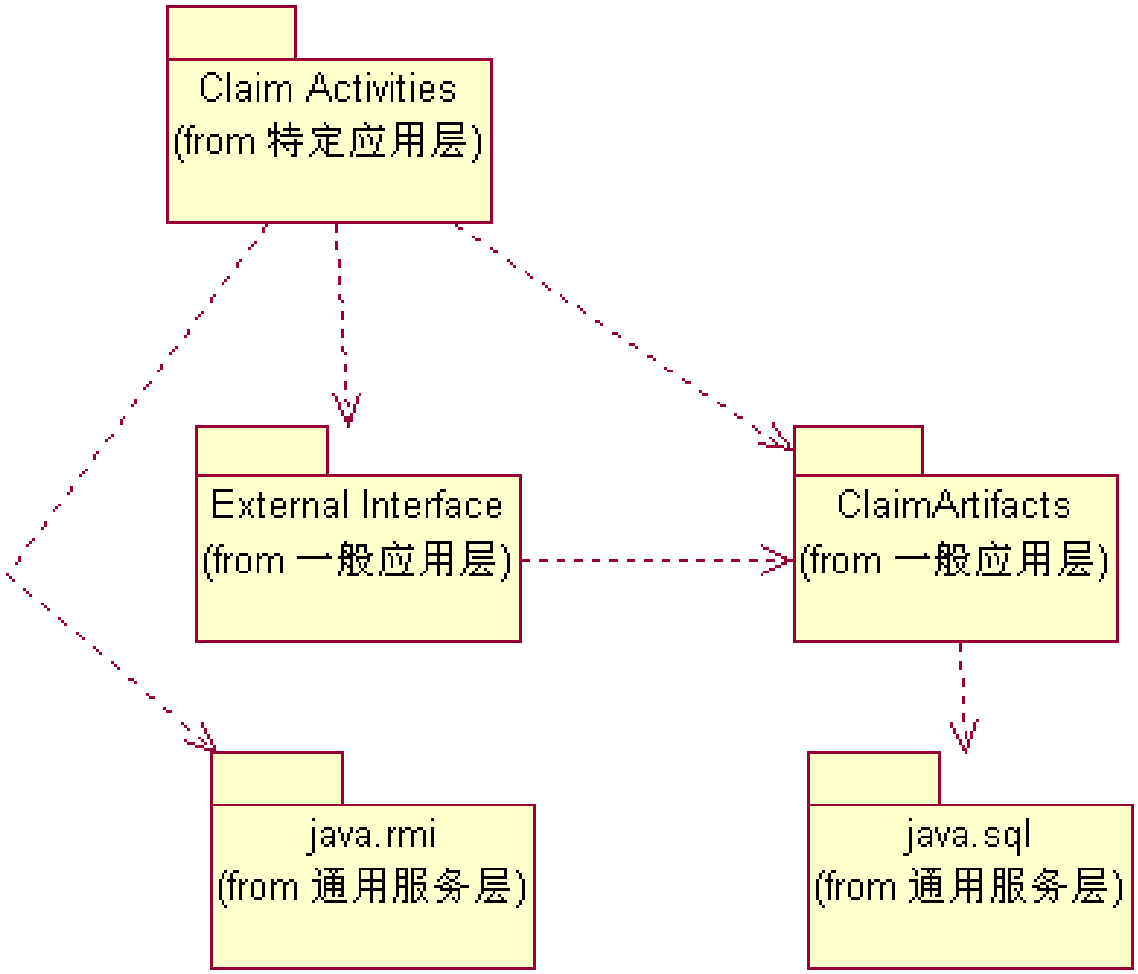
优化组织结构

- 步骤1：分包组织设计元素
  - 为了便于理解，可以在层次结构的基础上划分更细的组织单元（包）
  - 相当与教材上‘系统设计’一章的子系统分解，但是在分层体系结构基础上的进一步分解
  - 分包的原则（得到高内聚，低耦合分解的具体做法）
    - 将直接和特定类型用户（主导Actor）关联的边界类所对应的设计元素放在同一个包中（针对这种类型用户的用户界面）。
    - 将功能相关性比较强的设计元素放在同一包中
    - 将（针对被动Actor的）边界类相关的设计元素放在同一包中（后台系统接口）
- 步骤2：描述包之间的依赖关系

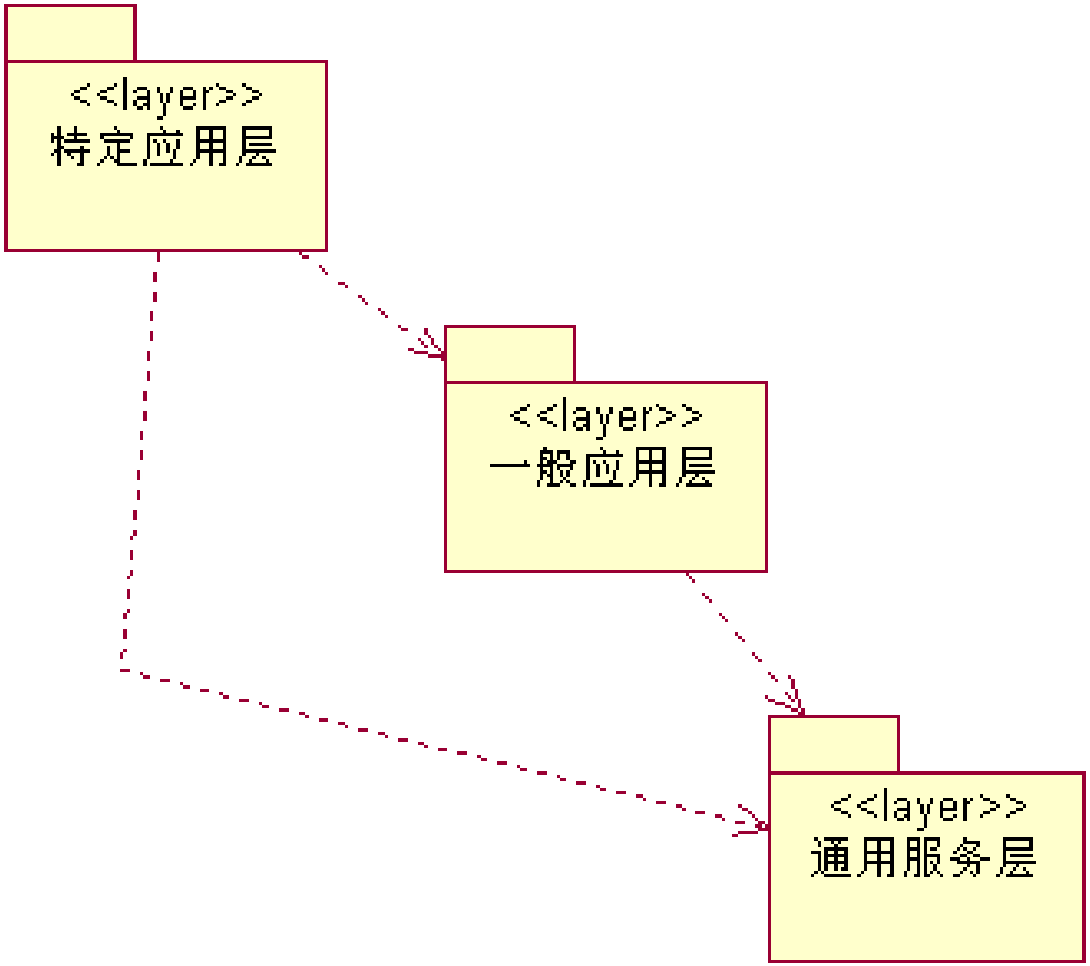
- 优化组织结构前的层次架构

- 优化组织结构后的层次架构

- 包之间的依赖关系


Claim Activities
(from 特定应用层)
External Interface
(from 一般应用层)
ClaimArtifacts
(from 一般应用层)
java.rmi
(from 通用服务层)
java.sql
(from 通用服务层)

- 层之间的依赖关系



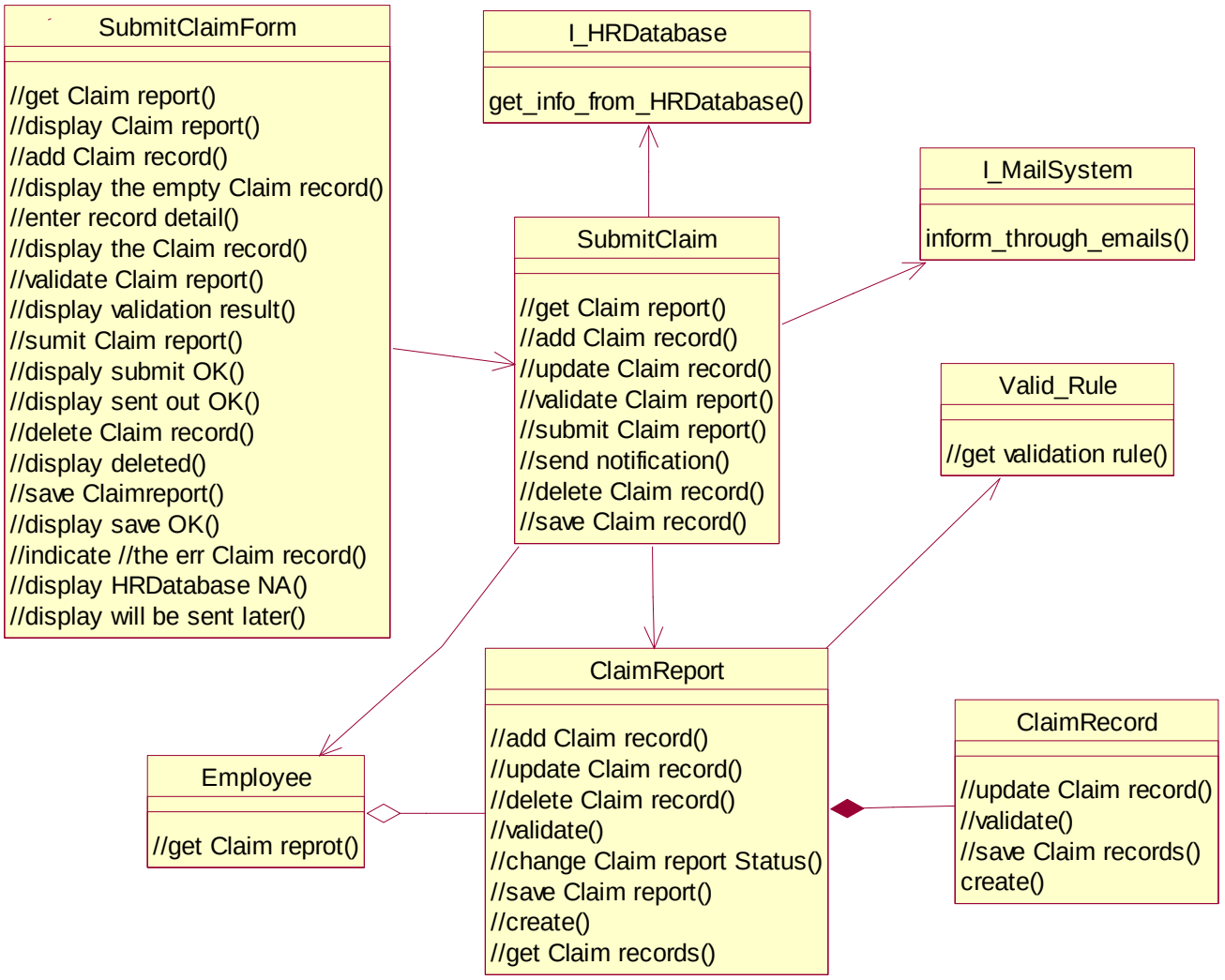
任务四： 局部设计

- 局部设计包括二项活动
  - 实现需求场景：用设计元素来描述Use Case实现
    - 用核心设计元素取代分析类
    - 落实外围设计元素对核心设计元素的支撑作用
  - 实现子系统接口：实现子系统接口规定的应为要求

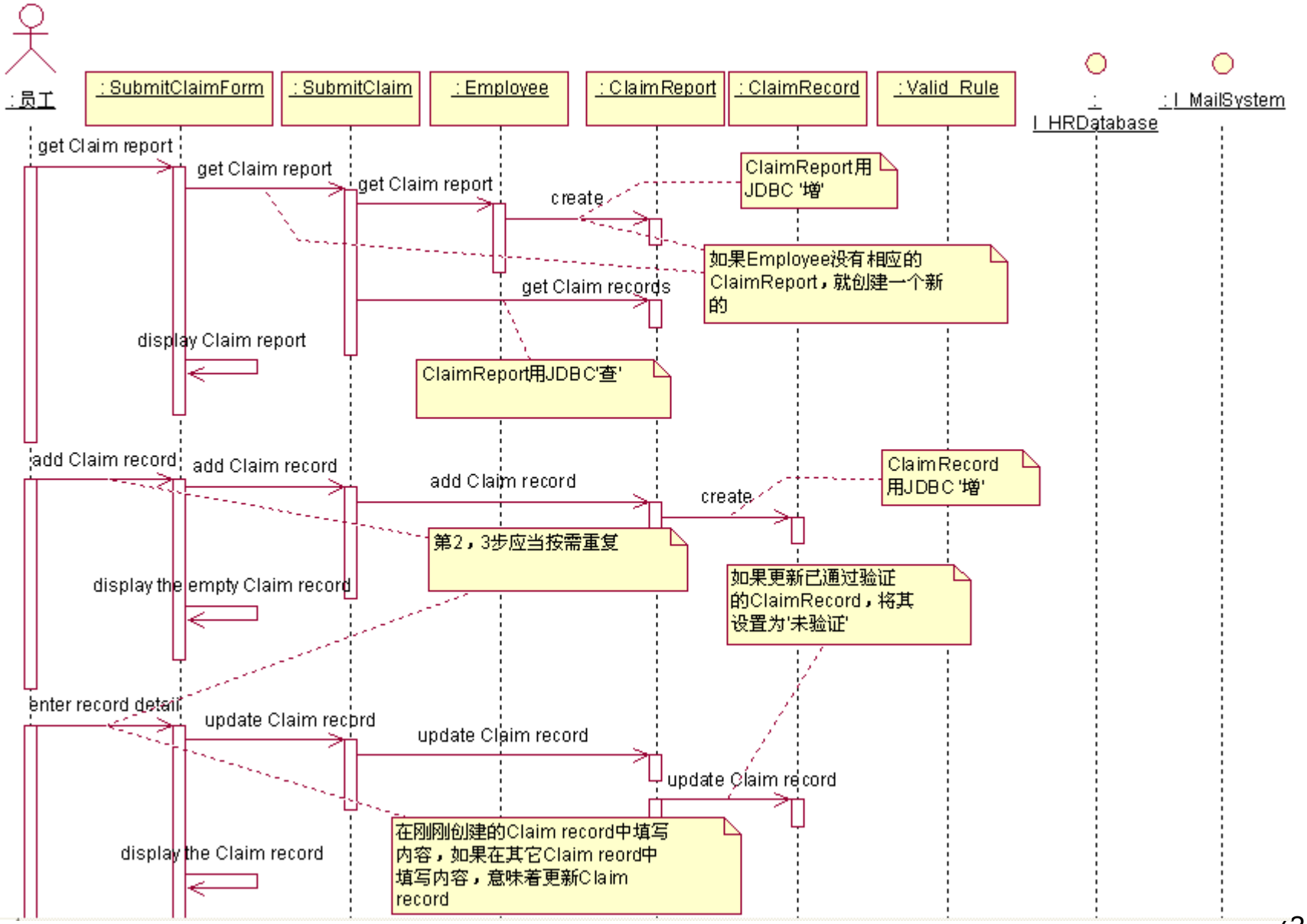
实现需求场景

- 步骤1：用核心设计元素替换分析类
  - 子系统接口类（分析类）的替换。若只接收消息，可用子系统接口的实例替换（表述）。若有可能向外发消息，用《subsystem proxy》的实例替换
  - 替换直接对应的设计类和分析类
- 步骤2：落实构架机制的支撑作用
  - 对使用构架机制的设计类，添加必要的操作和属性去绑定构架机制；建立并衔接必要的外围设计元素
  - 用序列图展开描述构架机制在Use Case实现中的应用

- 在参与类图中用核心设计元素替换分析类



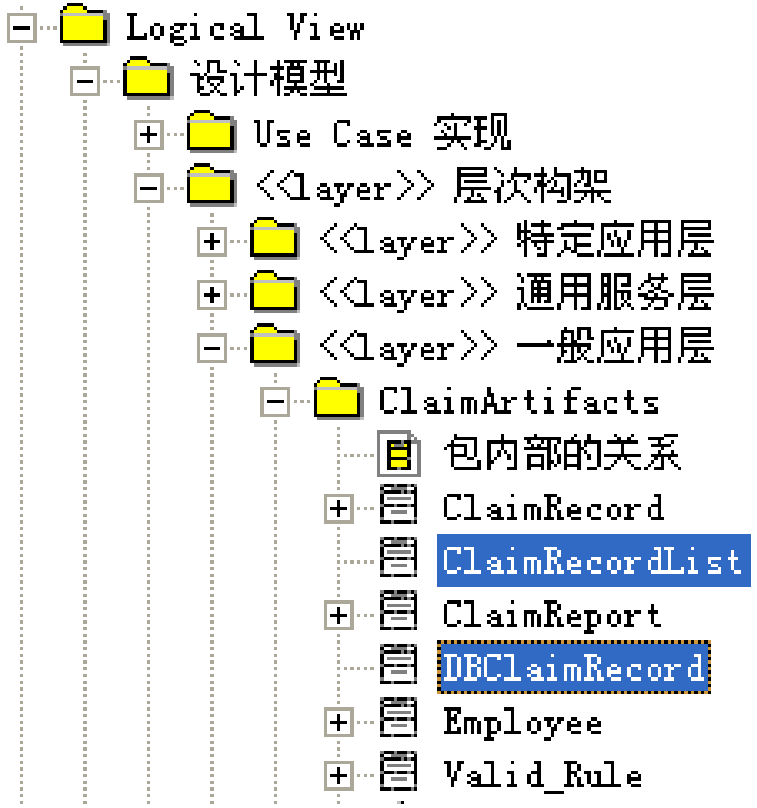
- 在序列图中用核心设计元素替换分析类



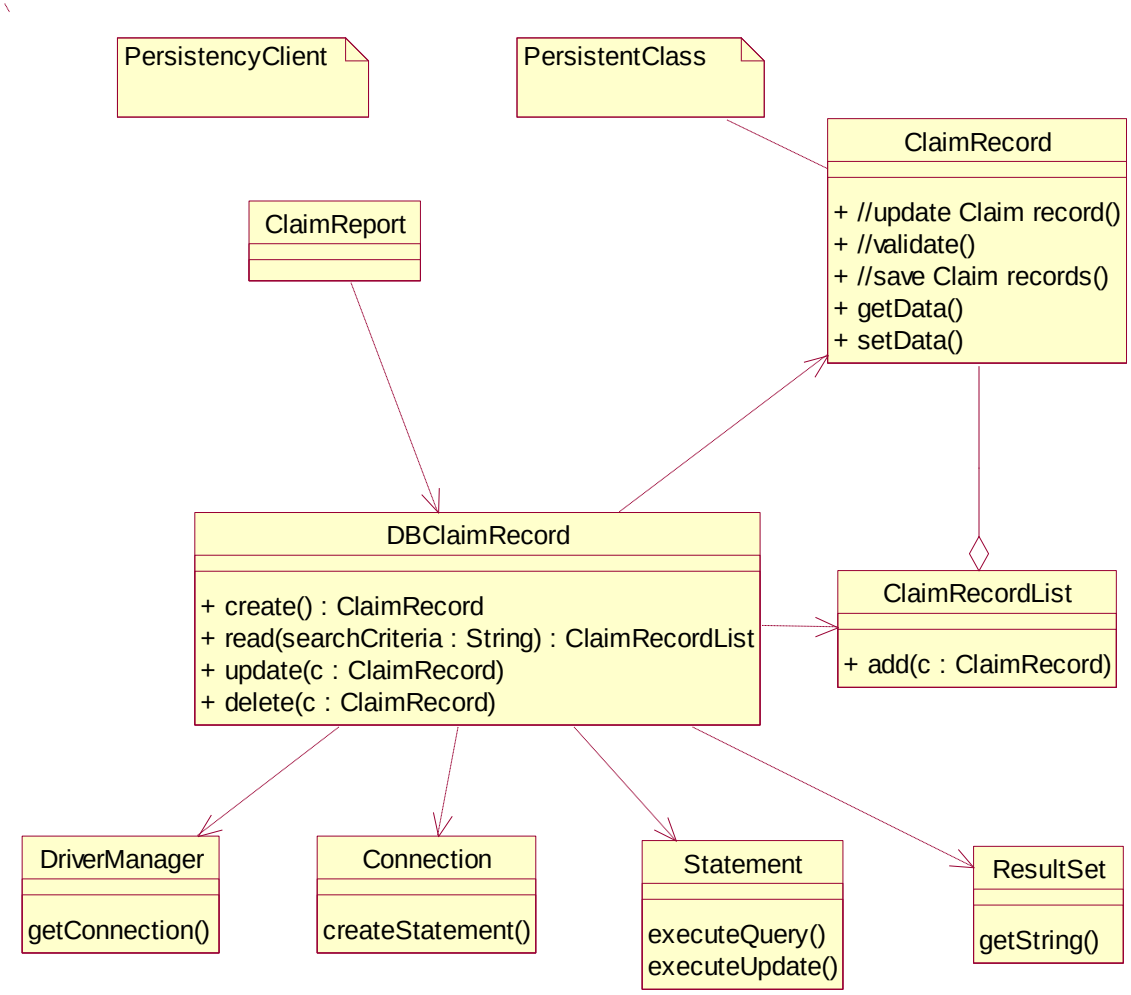
- 落实构架机制的支撑作用
- 例：ClaimRecord 应用留存机制时，构造型《role》所对应的具体核心设计元素和衔接设计元素

RDBMS-JDBC中的《role》	核心设计元素与衔接设计元素
《role》PersistentClass	ClaimRecord
《role》PersistencyClient	ClaimReport
《role》PersistentClassList	ClaimRecordList
《role》DBClass	DBClaimRecord

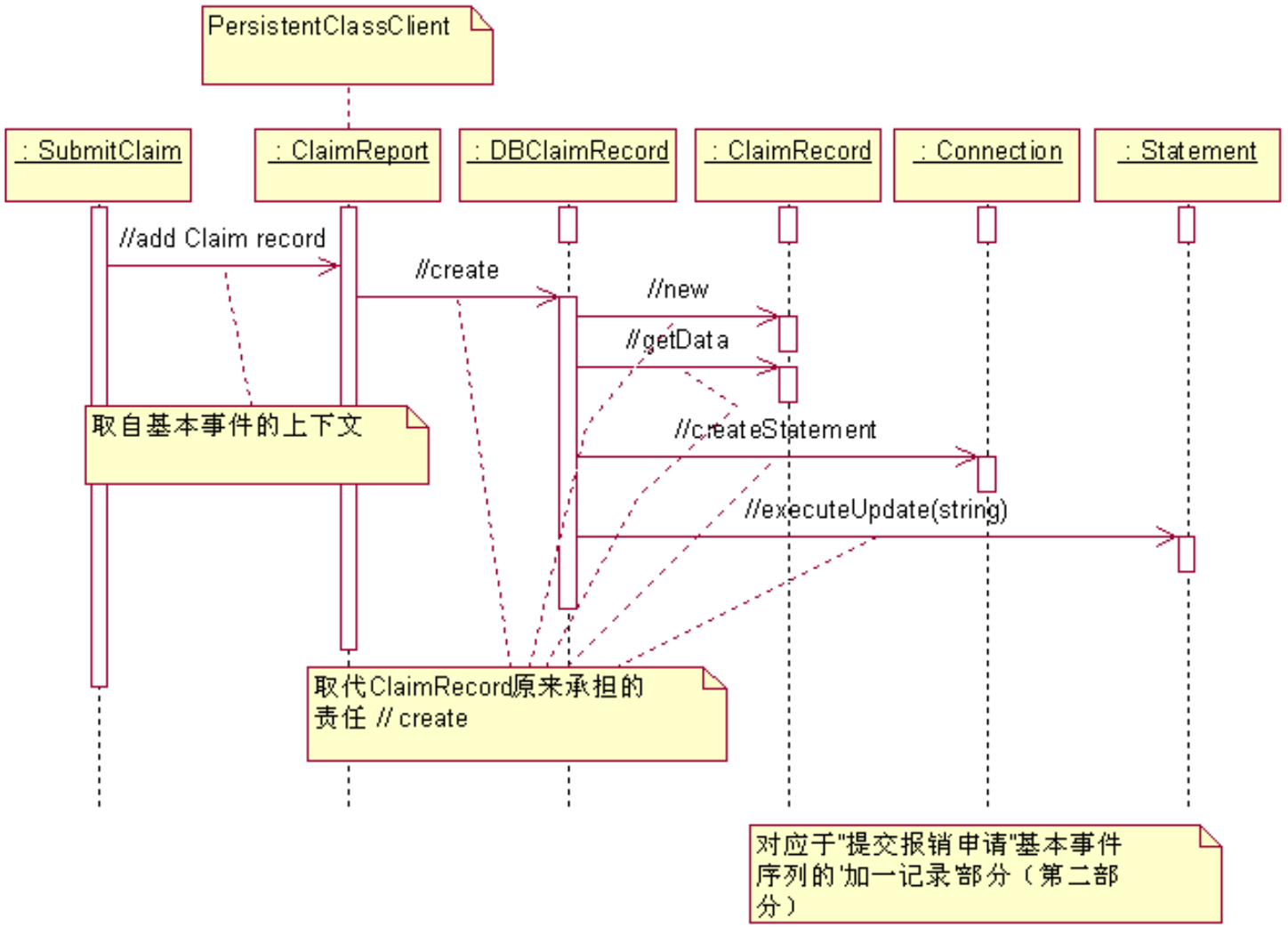
- 为ClaimRecord应用留存机制而创建的衔接设计元素



- ClaimRecord使用RDBMS-JDBC的静态结构



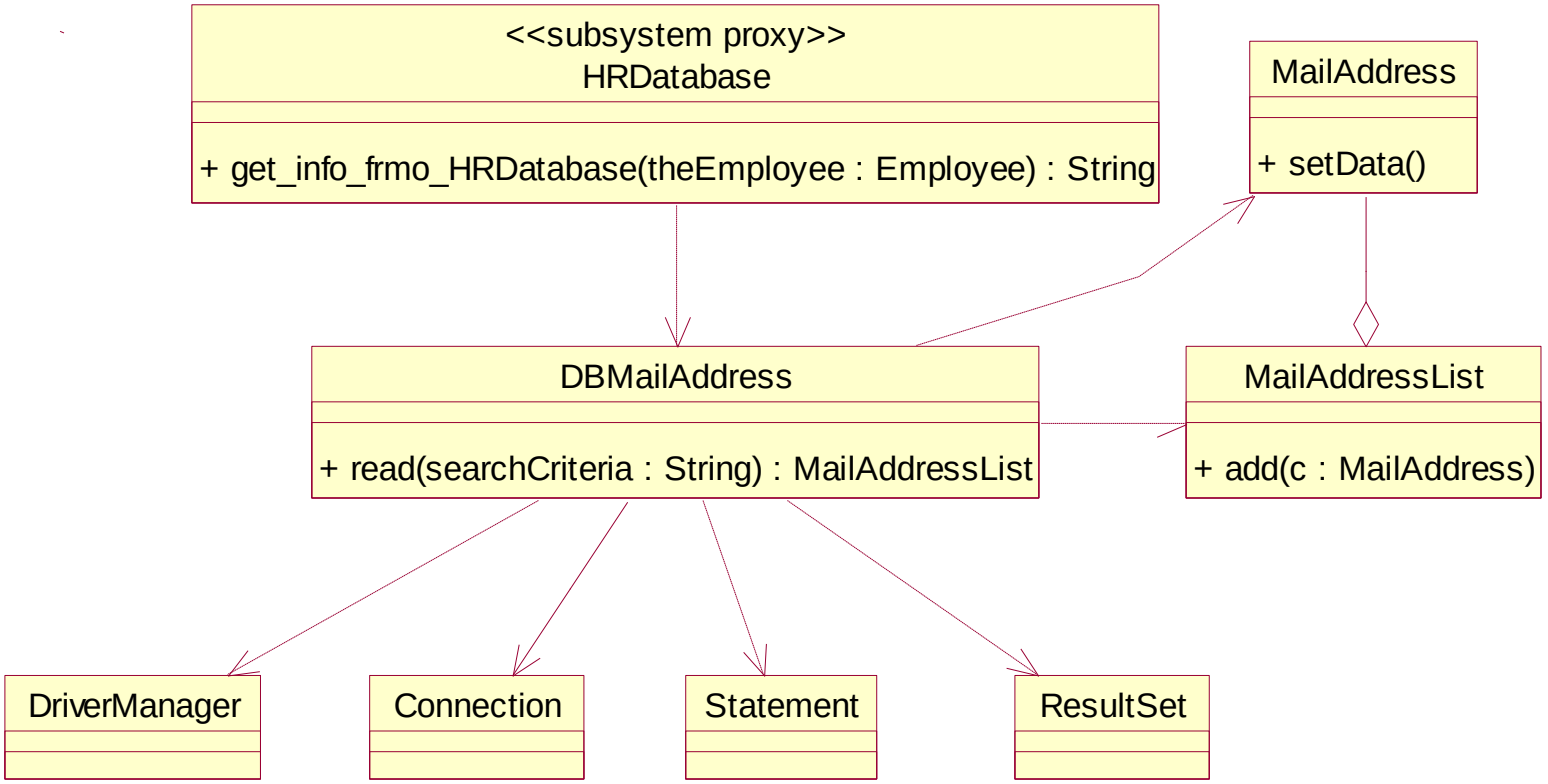
- ClaimRecord使用RDBMS-JDBC的’增‘的场景



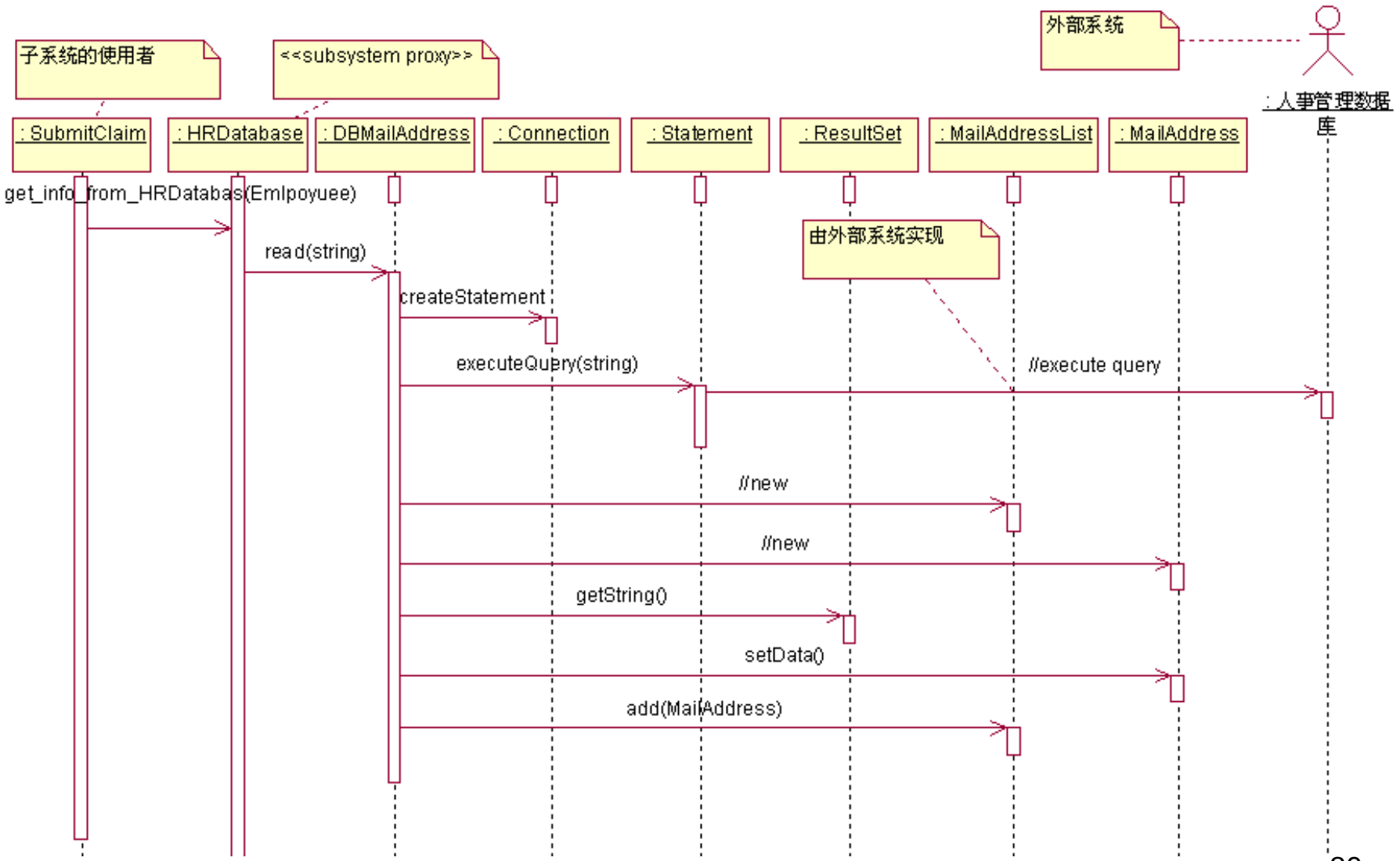
实现子系统接口

- 子系统接口定义的行为通常要由一组对象（子系统）来实现
- 步骤1：实现子系统接口定义的行为
  - 确定参与实现子系统行为的设计元素，将它们放入与该子系统对应的包内
  - 描述子系统的动态场景和静态结构
- 步骤2：明确子系统与外部设计元素的关系
  - 如果参与实现子系统的设计元素位于子系统所在的包之外，则子系统与其它设计元素的关系需要进一步说明
- 例：子系统接口I_HRDatabase 的实现

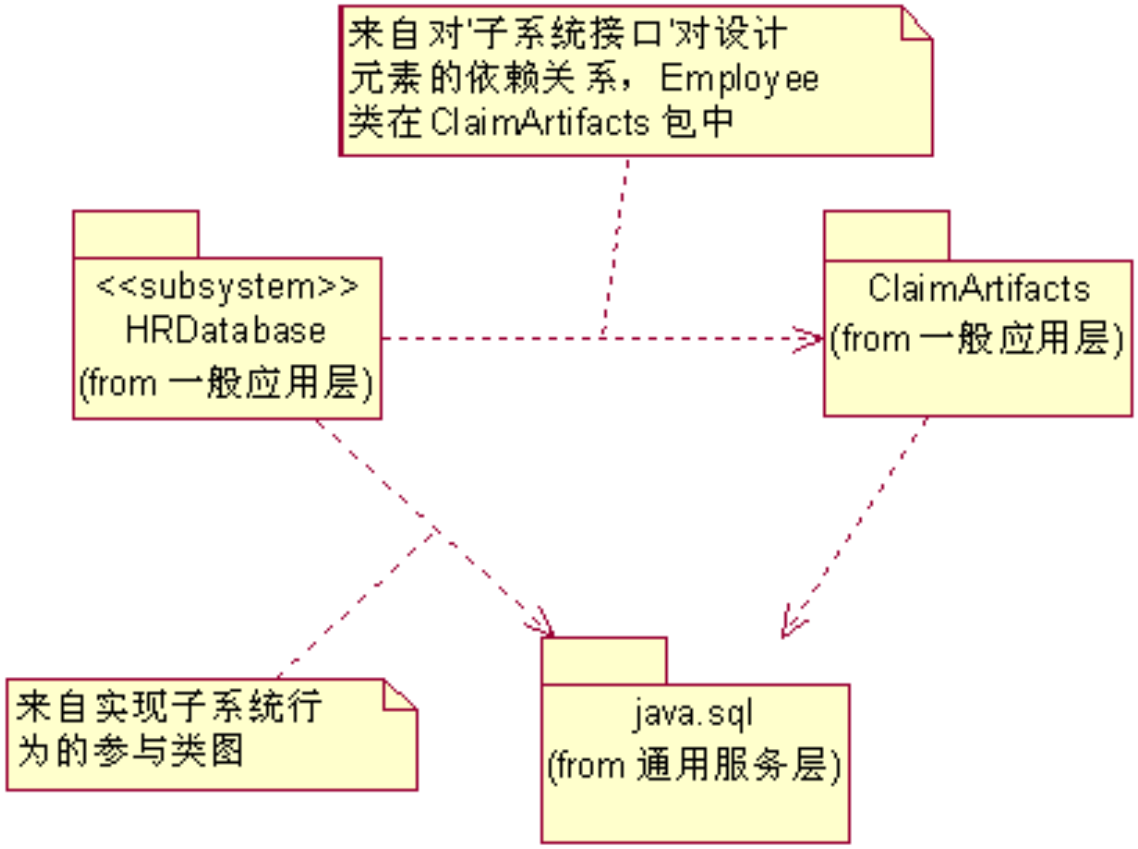
- HRDatabase 子系统的参与类图



- HRDatabase 子系统的顺序图



- HRDatabase 对外部的依赖关系



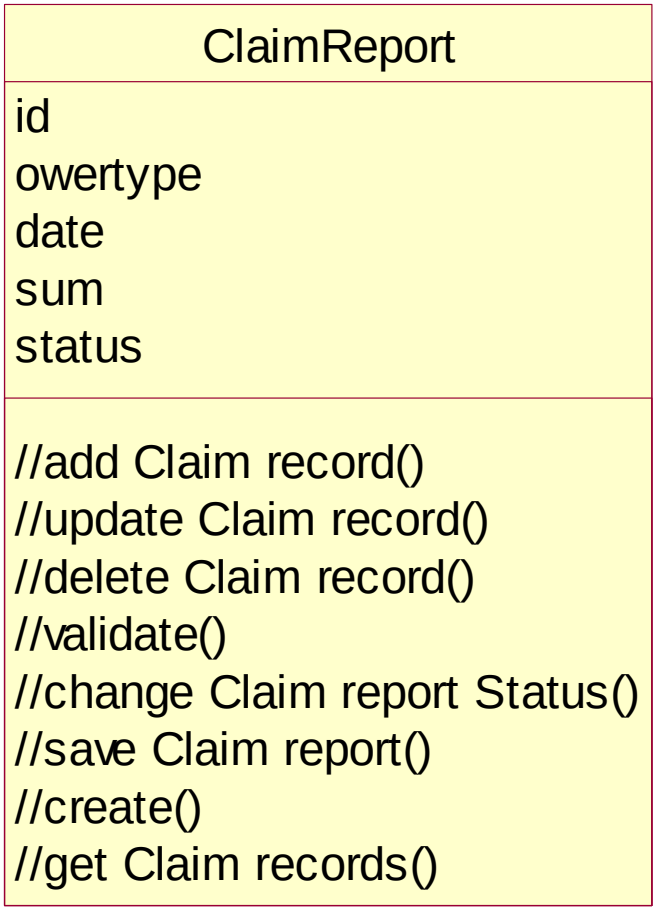
任务五： 细节设计

- 细节设计包括二项活动
  - 精化属性和操作：明确定义操作的参数和基本的实现逻辑，明确定义属性的类型和用途
  - 明确类之间关系：根据类之间通信的实际情况，明确类之间的关系
- 相关概念
  - 需要精化的类
    - 不是所有的类都需要在这里精化（比如GUI类不需在这里细化）
    - 重点在核心设计元素，主要时由控制类和实体类演化而来的类

精化属性和操作

- 步骤一：明确操作的定义
  - 找出满足基本逻辑要求的操作。操作的原始依据是类的责任及其相关的上下文信息
  - 补充Use Case实现没有覆盖的必要的操作，例如实例的初始化等
  - 给操作以清晰的表述
- 步骤二：明确属性的定义
  - 名称，类型，缺省值，可见度，适用范围，及简短文字说明

- ‘精化操作和属性’前的 ClaimReport



- ‘精化操作和属性’后的 ClaimReport

ClaimReport
id : int owertype : String date : Date sum : float status : int recordNumber : int <<class>> nextID : int
addClaimRecord() : ClaimRecord updateClaimRecord(theContent : ClaimRecord) deleteClaimRecord(theUselessRecord : ClaimRecord) validate() : boolean changeClaimReportStatus() : boolean getClaimRecords() : ClaimRecordList saveClaimRecord() <<class>> getNextID() : int getData() setData()

明确类之间关系

- 类之间可由三种关系
  - 关联关系（Association）
  - 依赖关系（Dependency）
  - 泛化关系（Generalization）
- 步骤一：明确依赖关系
  - 考查在分析任务中识别的核心设计元素之间的关联关系是否仍然成立。
  - 考查操作的表示（Signature）中涉及到的类是否引入了新的依赖关系
- 步骤二：细化关联关系
  - 对于保留下的关联关系做进一步的细化
  - 是否可从关联关系转化为聚合（Aggregation）甚至组合关系（Composite）
  - 在满足逻辑要求的条件下，尽可能避免双重的访问方向
  - 有必要的话，标注关系的重数，和类在关系中的角色
- 步骤三：构造泛化关系
  - 从特殊到一般，或从一般到特殊