北京大學
PEKING UNIVERSITY

Thinking in UML
第3章
UML核心元素

孙鹏晖

Mar, 2017

Outline

- 衍型
- 参与者
- 用况
- 边界
- 业务实体
- 包
- 分析类 & 设计类
- 关系
- 构件 & 节点

衍型

- 衍型也称为类型，构造型，是对UML元素基础定义的扩展，在同一个元素基础定义之上赋予特别含义.
- 例如:
- 用况：业务用况，业务用况实现等.
- 类：接口，边界类，实体类，控制类.

- 可以自己定义新的衍型.

参与者 —基本概念

- 参与者(actor):在系统之外与系统交互的某人或某物.
- 参与者只能存在于边界之外，边界之内的所有人和事物都不是参与者.
- 参与者也叫作主角.

- 参与者可以非人：每天自动统计网页访问量，生成统计报表，并发送至管理员信箱.
- 参与者是谁?->计时器.

参与者—如何发现

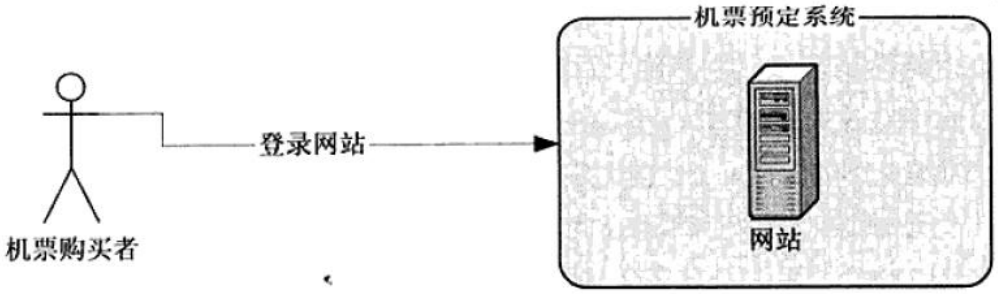


图 3.2　参与者情况一

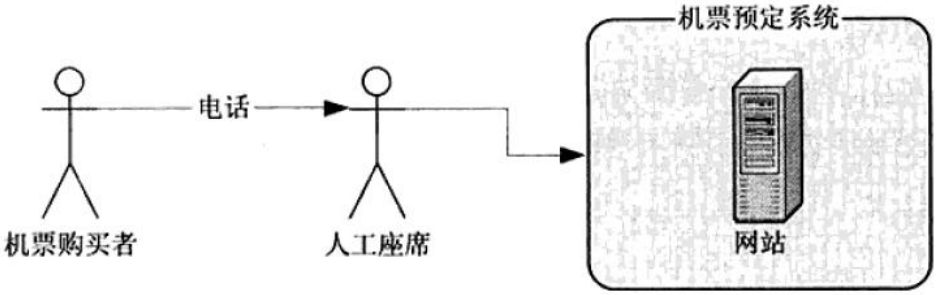


图 3.3　参与者情况二

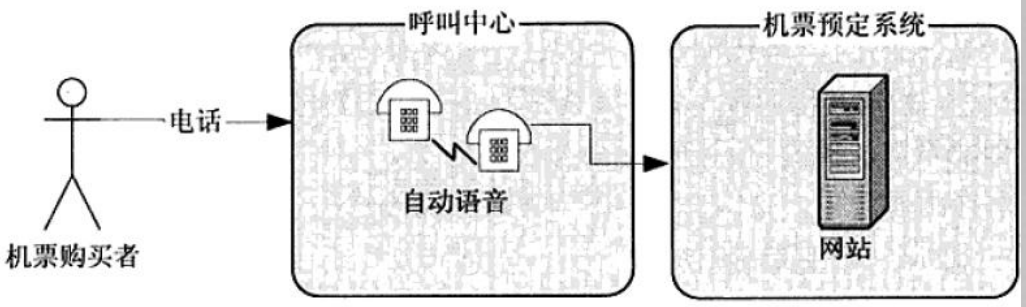


图 3.4　参与者情况三

参与者—业务主角

- 业务主角是参与者的一个衍型，特别用于定义业务的参与者，在需求阶段使用.

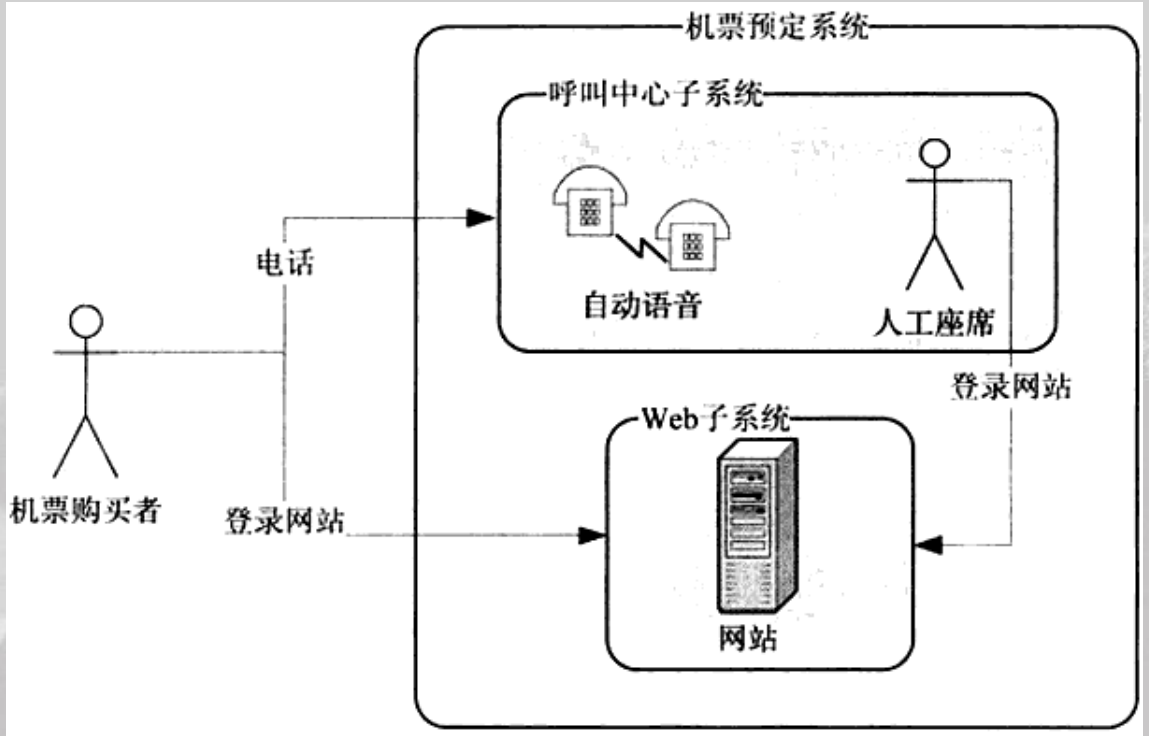


图 3.5　参与者情况四

参与者—业务工人

- 他们不是参与者!!!

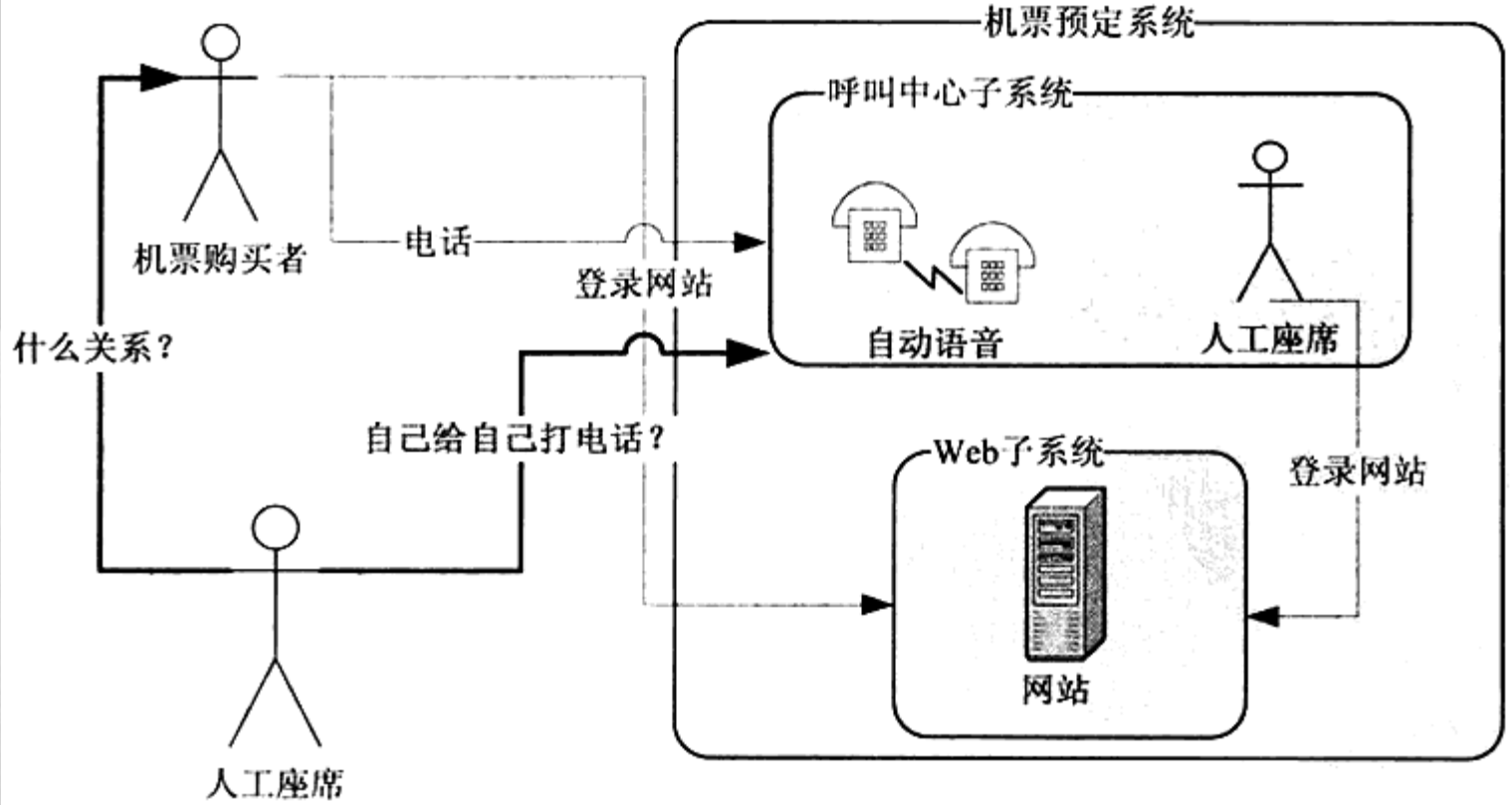


图 3.6　业务工人的尴尬

用况 —基本概念

- 用况是一种把现实世界的需求捕获下来的方法.
- 用况定义了一组用况实例, 其中每个实例都是系统所执行的一系列操作, 这些操作生成特定主角可以观测的值.

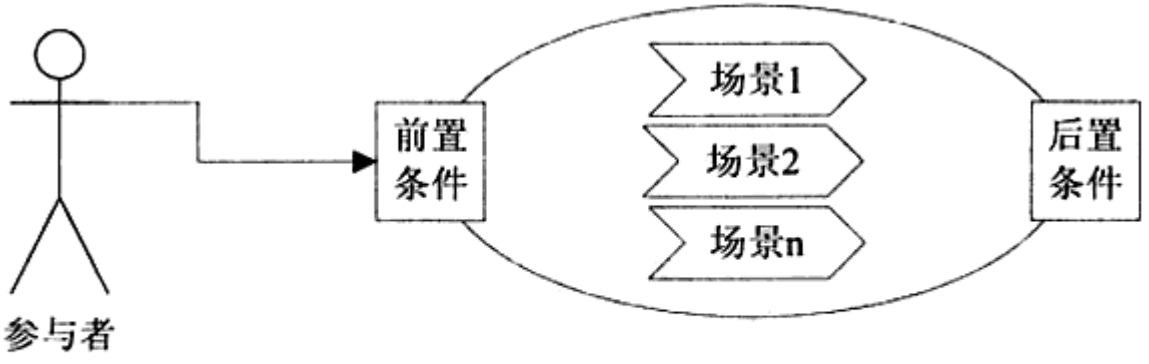


图 3.8 用例的构成

用况—特征

- 用况是相对独立的；

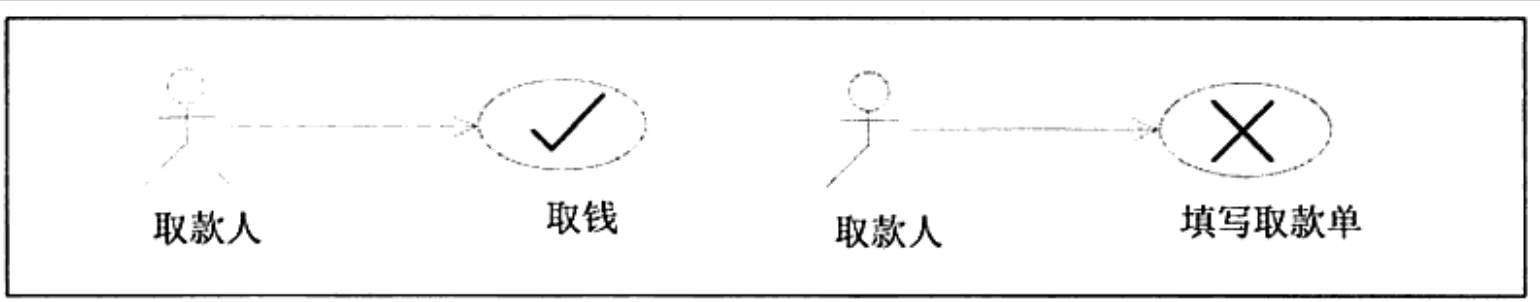


图 3.9　填写取款单不是取款人的目的，因此不是用例

- 用况的结果对于参与者是可观测且有意义的；

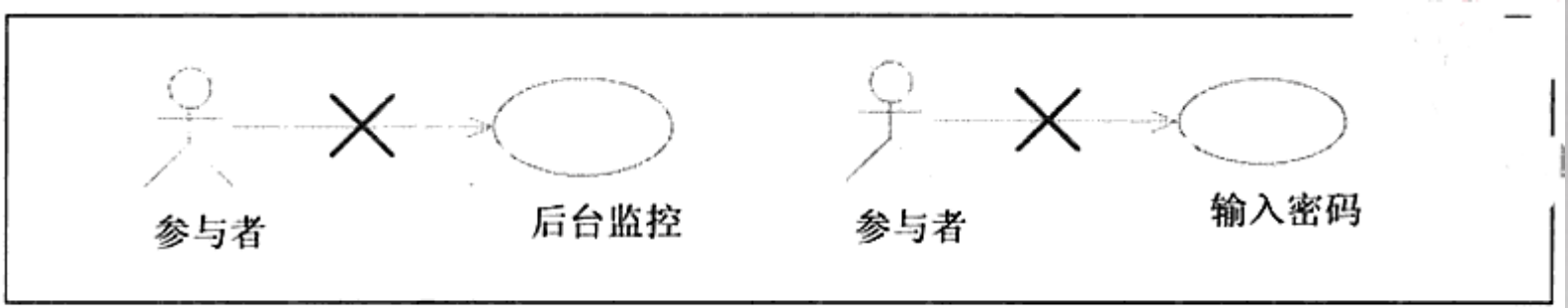


图 3.10　后台监控和输入密码对参与者是没有意义的，因此不是用例

用况—特征

- 用况必须由参与者发起，用况不能自启动，也不能启动其他用况；

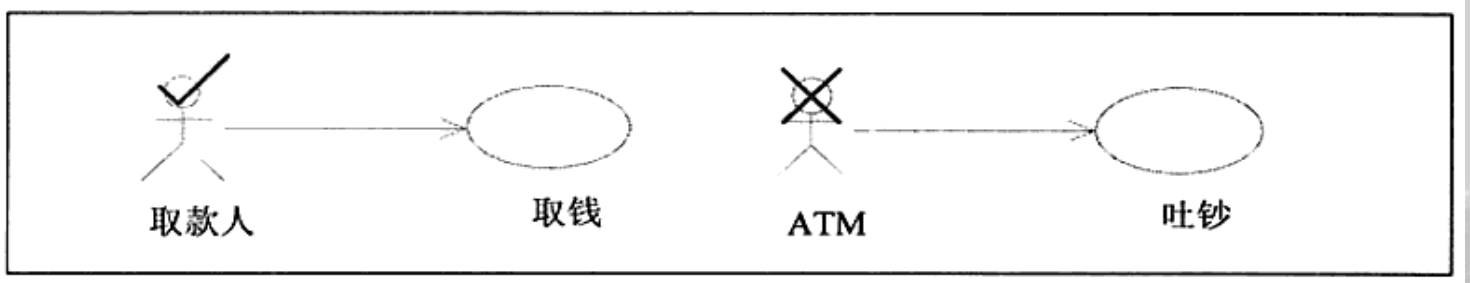


图 3.11　ATM 是没有吐钞的愿望的，因此不能驱动用例

- 用况要以动宾短语形式出现；

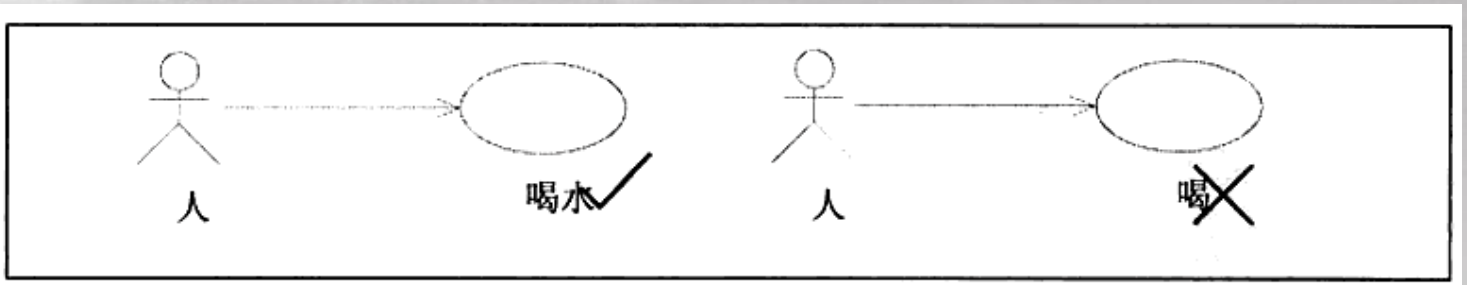


图 3.12　喝不能构成一个完整的事件，因此不能用来命名用例

用况—特征

- 一个用况就是一个需求单元，分析单元，设计单元，开发单元，测试单元，甚至部署单元.

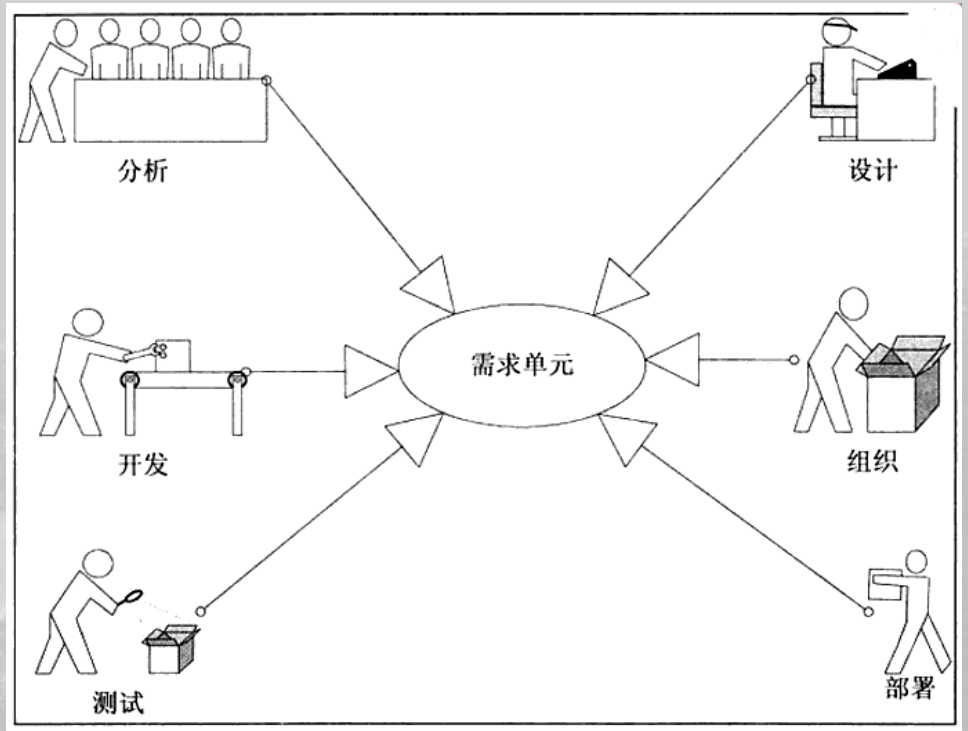


图 3.13　用例驱动

用况—粒度

- 这个问题没有一个标准的规则!
- 在项目过程中根据阶段不同，使用不同的粒度.

- 业务建模阶段—每个用况能够说明一件完整的事情.
- 例如:取钱，报装电话，借书．不要细致到填写申请单，查找书目等步骤.

用况—粒度

- 用况分析阶段—每个用况能描述完整的事件流.
- 例如:申报宽带业务可以分解为提供申请材料，受理业务,现场安装等.

- 系统建模阶段-用况能够描述作者与计算机的一次完整交互.
- 例如:填写申请单，审核申请单等.

边界

- 边界可大可小，由建模者主观臆定.
- 边界是无形的，与其说是一个元素，不如说是一个分析方法.

业务实体 —概念

- 业务实体是类的一种衍型，特别用于在业务建模阶段建立领域模型.
- 业务实体代表业务角色执行业务用况时所处理或使用的事物.

业务实体 —特征

- 业务实体来自现实世界.
- 例如:饭店中业务实体有菜单, 饮料等.

- 业务实体是在分析业务流程时发现的.
- 例如:商场卖衣服, 衣服是业务实体, 衣架不是.

- 业务实体具有属性和方法.

包

- 包将信息分类，形成逻辑单元，整合复杂信息.
- 包的关系只有依赖关系，好的分包高内聚，低耦合.

- 同一个包内部相互联系紧密. —高内聚
- 不同包之间尽量不要依赖. —低耦合
- 避免双向依赖和循环依赖.

分析类—概念

- 分析类用于获取系统中主要的“职责簇”. 代表系统的原型类.
- 在统一过程中是一个过渡类型，不是强制过程.

分析类—边界类

- 边界类是一种用于对系统外部环境与其内部运作之间的交互进行建模的类，任何两个有交互的关键对象之间都应当考虑建立边界类.

- 例如:
- 边界类实例可以是窗口，传感器，终端，驱动程序等.

分析类—控制类

- 控制类用于对一个或几个用况所特有的控制行为进行建模, 具有协调性质.

- 例如:
- 寄信人到邮局寄信，其中写上地址，称重，计算邮资，邮寄信件等都可以是控制类的来源.
- 控制类多位于业务逻辑层.

分析类—实体类

- 实体类:用于对必须存储的信息和相关行为建模的类.
- 实体类通常都是永久性的.

- 例如:
- 人到邮局寄信的业务实体可以转化为信,信封,邮票等实体类.

设计类 —基本概念

- 设计类是系统实施中一个或多个对象的抽象，设计类所对应的对象取决于实施语言.

图 3.27　设计类的版型

设计类—可见性

- 公有
- 保护
- 私有
- 实施:属性和方法只在类本身内部是可视的(取决于具体的语言)，实施可见性最具限制性，当只有类本身才可以使用操作时,使用这种可见性,是私有可见性的变体.

关系 —概念&类型

- 抽象出对象之间的联系，让对象构成某个特定的结构.

- 关联关系：描述不同类对象之间的结构关系，是一种静态关系，与运行状态无关，是一种强关联关系.
- 例如：公司与员工之间的关系，车票与乘车人之间的关系等

关系—类型

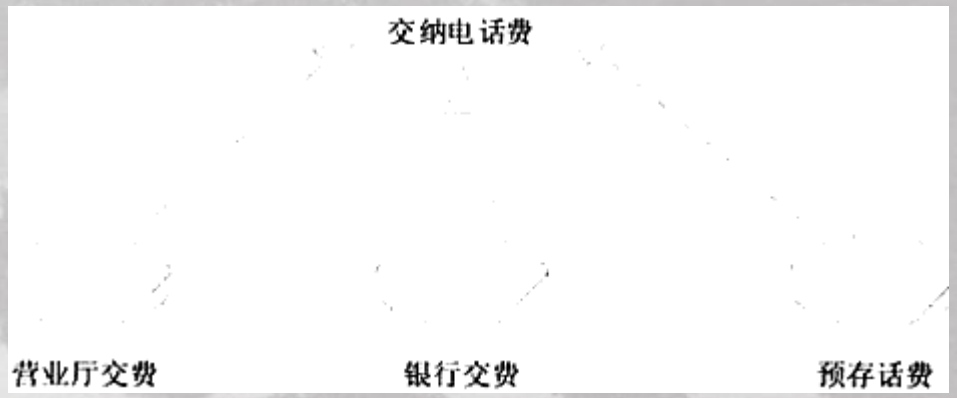
- 依赖关系：描述对象在运行时使用到另一个对象的关系，是一种临时性的关系.
- 例如：人与船的关系，人与刀的关系等.

- 扩展关系：特别用于在用况模型中说明向基本用况中的某个扩展点插入扩展用况.
- 例如：打电话时接通另一个呼叫，保留当前通话是打电话用况的扩展用况.

关系—类型

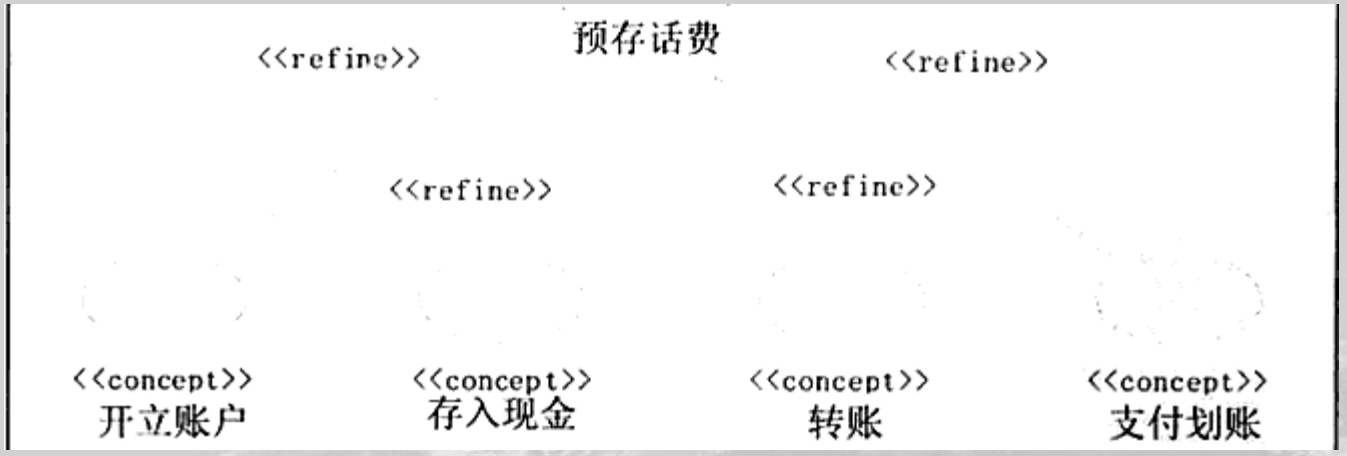
- 包含关系：特别用于用况模型，说明在执行基本用况实例过程中插入的行为段.
- 例如：去银行取钱，转账或者修改密码都要先核对账号，那么核对账号就是一个包含用况，是必须的.

- 实现关系：用于用况和用况实现，说明基本用况的实现方式.



关系—类型

- 精化关系:用于基本用况分解出许多更小的精化用况.



- 泛化关系:表明对象之间的继承关系. 作者不赞同在用况中使用泛化关系.

关系—类型

- 聚合关系:用于类图表明整体由部分构成.
- 例如:一个部门有很多人员.

- 组合关系:表示一个母对象由子对象组合而成.
- 例如,母公司由很多子公司组合而成,母公司不存在则子公司也不存在了.

构件—基本概念

- 构件是系统中实际存在的可更换部分，实现特定功能，符合一套接口标准并实现一组接口.
- UML中把构件定义为任何的逻辑代码模块.
- 作者认为，一个构件是独立的业务模块，有完备的功能，可独立部署，是一个完备的服务.

构件—特性(作者观点)

- 完备性:能够完成一项或一组特定的业务目标(功能).
- 独立性:构件应当不依赖于其他构件,可独立部署.
- 逻辑性:构件通过软件设计的逻辑进行划分.
- 透明性:构件的修改应当只涉及构件的定义和类组合的修改,而不应该导致类的修改.

节点—概念

- 节点是带有至少一个处理器，内存以及可能还带有其他设备的处理元素.
- 实际工作中，一般来说服务器工作站或客户机都可以称为一个节点.

Q & A

Thanks !