软件体系结构原理、方法与实践

从软件危机谈起 – 软件危机的原因

- 用户需求不明确 :[需求模糊，需求变更]
- 缺乏正确的理论指导:[缺乏工程化指导思想，依赖个人发挥]
- 软件规模越来越大:[团队大，沟通渠道增加，沟通不畅]
- 软件复杂度越来越高:[产生复杂问题]

从软件危机谈起 – 如何克服软件危机

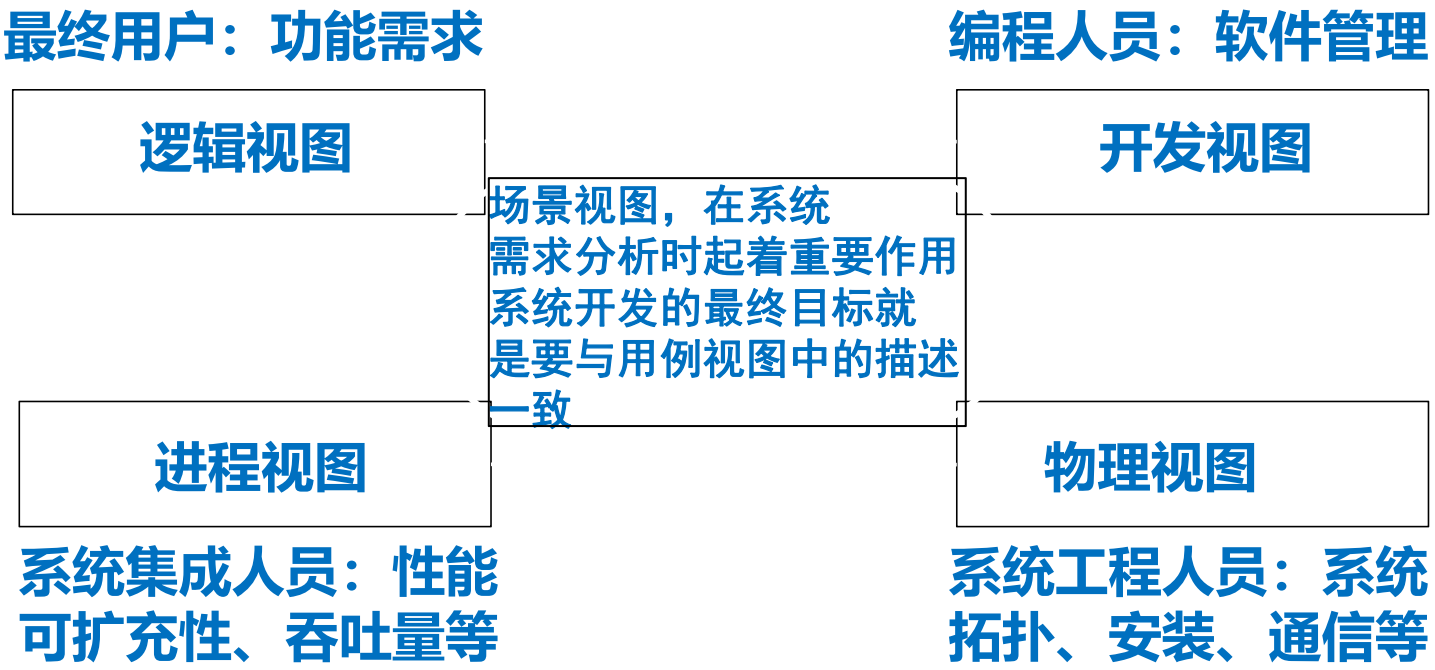
- ✓ 人们面临的不光是技术问题，更重要的是管理问题。管理不善必然导致失败 。[提高项目管理水平]
- ✓ 要提高软件开发效率，提高软件产品质量，必须采用工程化的开发方法与工业化的生产技术。[开发方法学，软件工具]
- ✓ 在技术上，应该采用基于重用的软件生产技术；在管理上，应该采用多维的工程管理模式。

1.2、构件与软件重用 – 构件获取方式

- 从现有构件中获得符合要求的构件，直接使用或作适应性修改，得到可重用的构件。
- 通过遗留工程，将具有潜在重用价值的构件提取出来，得到可重用的构件。
- 从市场上购买现成的商业构件，即商品化的产品或技术COTS（Commercial Off-The-Shell）构件。
- 开发新的符合要求的构件。

4+1模型 – 概述

- Kruchten在1995年提出了4+1的视图模型。



4+1模型 – 逻辑视图

- 逻辑视图主要支持系统的**功能需求**，即系统提供给**最终用户的服务**。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。
- 在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表逻辑视图，用类图来描述逻辑视图，逻辑视图使用面向对象的风格。

4+1模型 – 开发视图

- 开发视图也称模块视图，主要侧重于软件模块的组织和管理。
- 开发视图要考虑软件内部的需求，如软件开发的容易性、软件的重用和软件的通用性，要充分考虑由于具体开发工具的不同而带来的局限性。
- 开发视图通过系统输入输出关系的模型图和子系统图来描述。

4+1模型 – 进程视图

- 进程视图也称为并发视图，侧重于系统的运行特性，主要关注一些非功能性的需求。
- 进程视图强调并发性、分布性、系统集成性和容错能力，以及从逻辑视图中的主要抽象如何适合进程结构。它也定义逻辑视图中的各个类的操作具体是在哪一个线程中被执行的。
- 进程视图可以描述成多层抽象，每个级别分别关注不同的方面。在最高层抽象中，进程结构可以看作是构成一个执行单元的一组任务。它可看成一系列独立的，通过逻辑网络相互通信的程序。它们是分布的，通过总线或局域网、广域网等硬件资源连接起来。

4+1模型 – 物理视图

- 物理视图主要考虑如何把软件映射到硬件上，它通常要考虑到系统性能、规模、可靠性等。解决系统拓扑结构、系统安装、通讯等问题。
- 当软件运行于不同的节点上时，各视图中的构件都直接或间接地对应于系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其他视图的影响最小。

4+1模型 – 场景

- 场景可以看作是那些重要系统活动的抽象，它使四个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。在开发体系结构时，它可以帮助设计者找到体系结构的构件和它们之间的作用关系。同时，也可以用场景来分析一个特定的视图，或描述不同视图构件间是如何相互作用的。
- 场景可以用文本表示，也可以用图形表示。

软件体系结构风格与评估

- 学校要开发一个学生信息管理系统，系统功能有学生基础信息管理，学生事务审批，学生成绩统计分析。

(1) 请使用面向服务的方式设计该系统，并为该系统各功能模块选择合适的体系结构风格。

UML的结构

- 规格说明：对构造块的语法，语义进行文字性说明，是模型的核心。
- 结构事物：在模型中属于静态部分，代表该年少或物理上的元素；总共有七种结构事物
- 行为事物：是UML模型中的动态部分，代表时间和空间上的动作，主要有两种行为事物，分别是交互(消息)和状态机。

1、交互(内部活动)：由一组对象之间在特定上下文中，为达到特定目的而进行的一系列消息交互动作。

2、状态机：由一系列对象状态组成，状态的变化，如审核流程。

- 分组事物：是UML中的组织部分，可以把它看成一个盒子，模型可在其中进行分解，如包，子系统。
- 注释事物：模型的解释，说明，描述的元素，如注释。

- 公共分类
- 扩展机制

- 构件图
- 部署图
- 包图
- 组合结构图
- 制品图

UML2.0新增

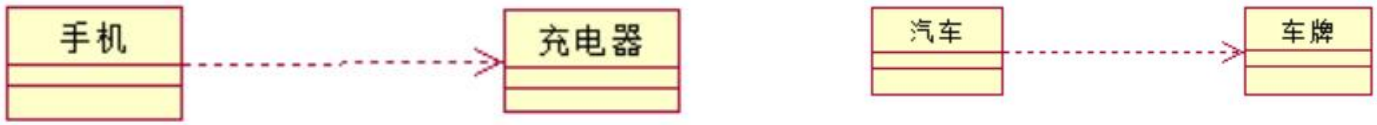
- 顺序图/序列图
- 通信图/协作图
- 状态图
- 活动图
- 定时图
- 交互概览图

UML2.0新增

种图，
1.0的
基础上新增了
5种图

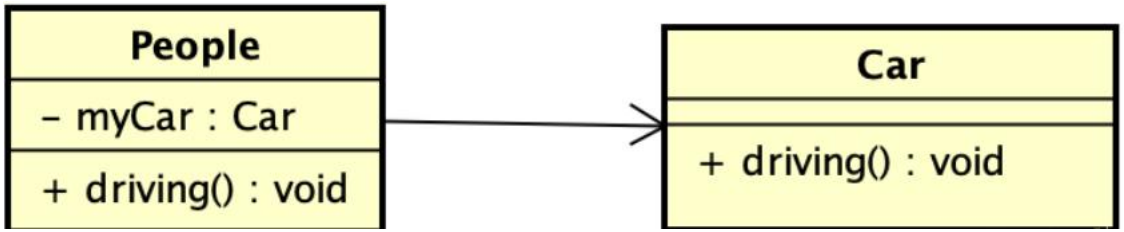
UML的结构-构造块中的关系-依赖关系

- **依赖关系:**也是类与类之间的连接. 表示一个类依赖于另一个类的定义. 依赖关系总是单向的 。可以简单的理解，就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、临时性的、非常弱的，但是B类的变化会影响到A，如:参数或变量；也就是说一个事务变化影响另外一个事务,
- **箭线表示：**带箭头的虚线，指向被使用者。
- **例如：**手机依赖于充电器，汽车依赖于车牌。

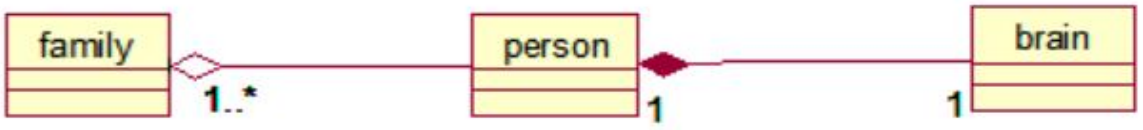


UML的结构-构造块中的关系-关联关系

- **关联关系:**关联关系式一种结构化的关系，是指一种对象和另一种对象有联系。给定关联的两个类。能够从当中的一个类的对象访问到另外一个类的相关对象
  箭线表示：带普通箭头的实线，指向被拥有者。双向的关联可以有两个箭头，或者没有箭头，单向的关联有一个箭头。如图：人是人，车是车，没有整体与部分的关系。

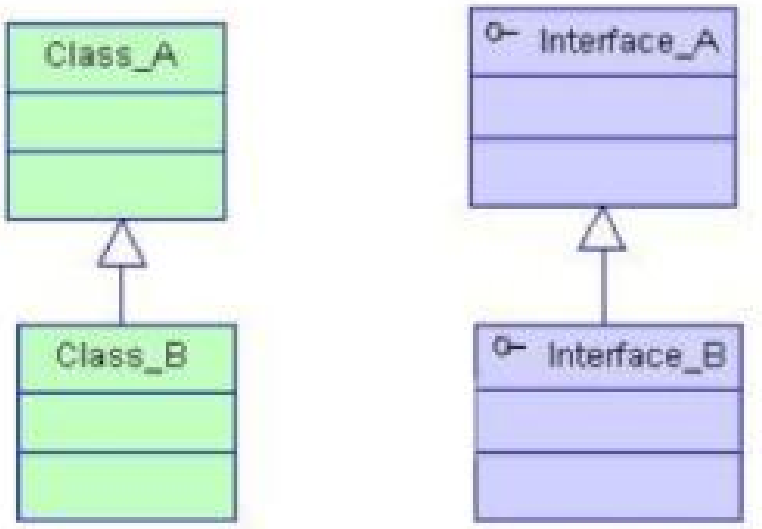


- **聚合关系:**总体和部分的关系，Has-a关系。总体和部分能够分离，总体与部分生命周期不同；如：计算机和主板，家庭和孩子
- **组合关系：**是一种包含关系，Is-a 关系。总体与部分不可分割，总体与部分的生命周期相同；如：桌子与桌腿的关系，人和大脑



UML的结构-构造块中的关系-泛化关系

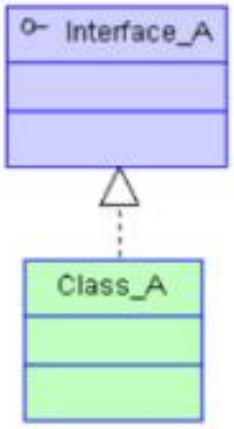
- **泛化关系:**一个类/接口继承另外一个类/接口的关系，子类还能添加自己的功能；子类与父类的关系，也称为特殊和一般的关系，例如：动物类与老虎类
- **箭线表示:**用实线空心箭头表示
- **如图：**B类继承A类，B接口继承A接口均为泛化关系



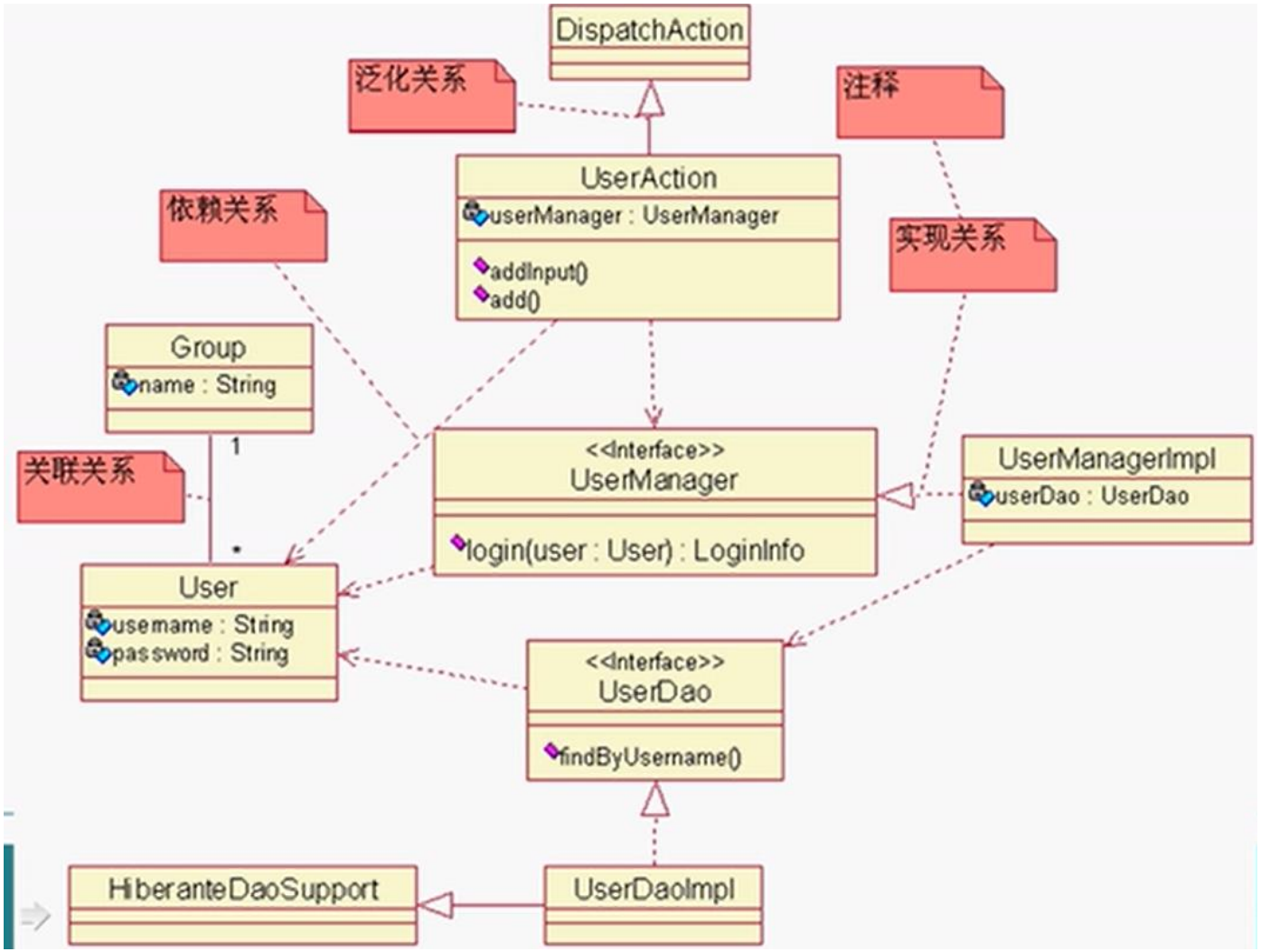
UML的结构-构造块中的关系-实现关系

- **实现关系:**指一个类实现接口的关系。动物类实现移动的接口
- **箭线表示:**虚线空心箭头表示

如图：A类实现A接口均为泛化关系



UML的结构-关系图形表示实例



UML的结构-五个系统视图

逻辑视图:用于描述用例视图中提出的系统功能实现，关注系统内部，既描述系统的静态结构(类，对象以及他们之间的关系)，也描述系统内部的动态协作关系。
组成:静态(类图和对象图)，动态(状态图，序列图，协作图，活动图)。
使用者：系统分析人员，设计人员
干系人关注点:类和对象

实现视图：组件是不同类型的代码模块，构造应用的软件单元，该视图描述系统的实现模块和他们之间的依赖关系。
组成：构件图

使用者：开发人员
干系人关注点:物理代码文件和组件

用例视图:描述系统应具备的功能，用例是系统的一个功能单元，可以被描述为系统与参与者的一次交互，是情景描述。
组成:用例图，使用者：用户
干系人关注:需求分析模型

进程视图：考虑资源的有效利用，代码的并行执行以及系统环境中异步事件的处理

组成：状态图，协作图，活动图

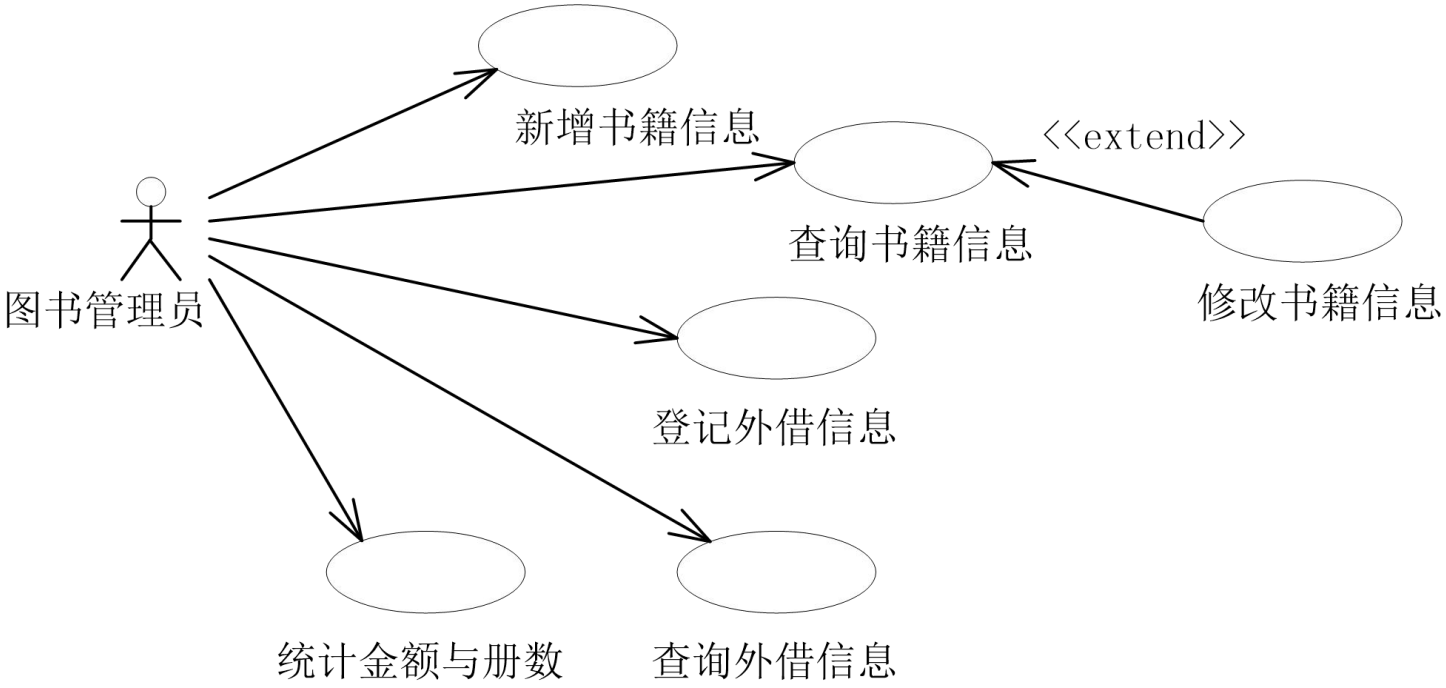
使用者：开发人员，系统集成人员
干系人关注点:线程，进程，并发

物理视图:显示系统的物理部署，描述节点上运行实例的部署情况。

组成：部署图

使用者：开发人员，系统集成人员，测试
干系人关注点:软件到硬件的映射

用例图



新的开始，新的起点，让我们一起为梦想而努力。

类图

类图:一切面向对象的核心建模工具,描述了系统中对象的类型以及他们之间存在的各种静态关系。

类图中的各种关系:

泛化关系:继承关系。

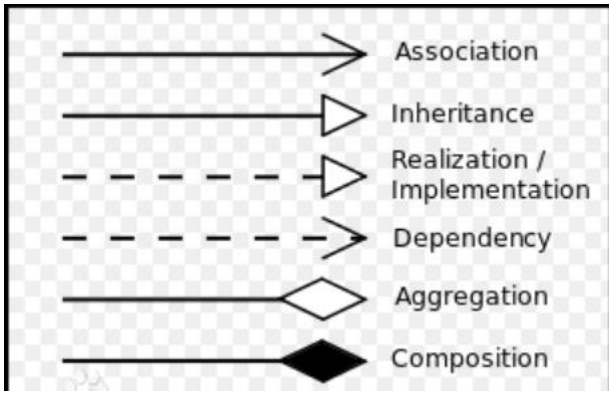
实现关系:接口与实现类关系

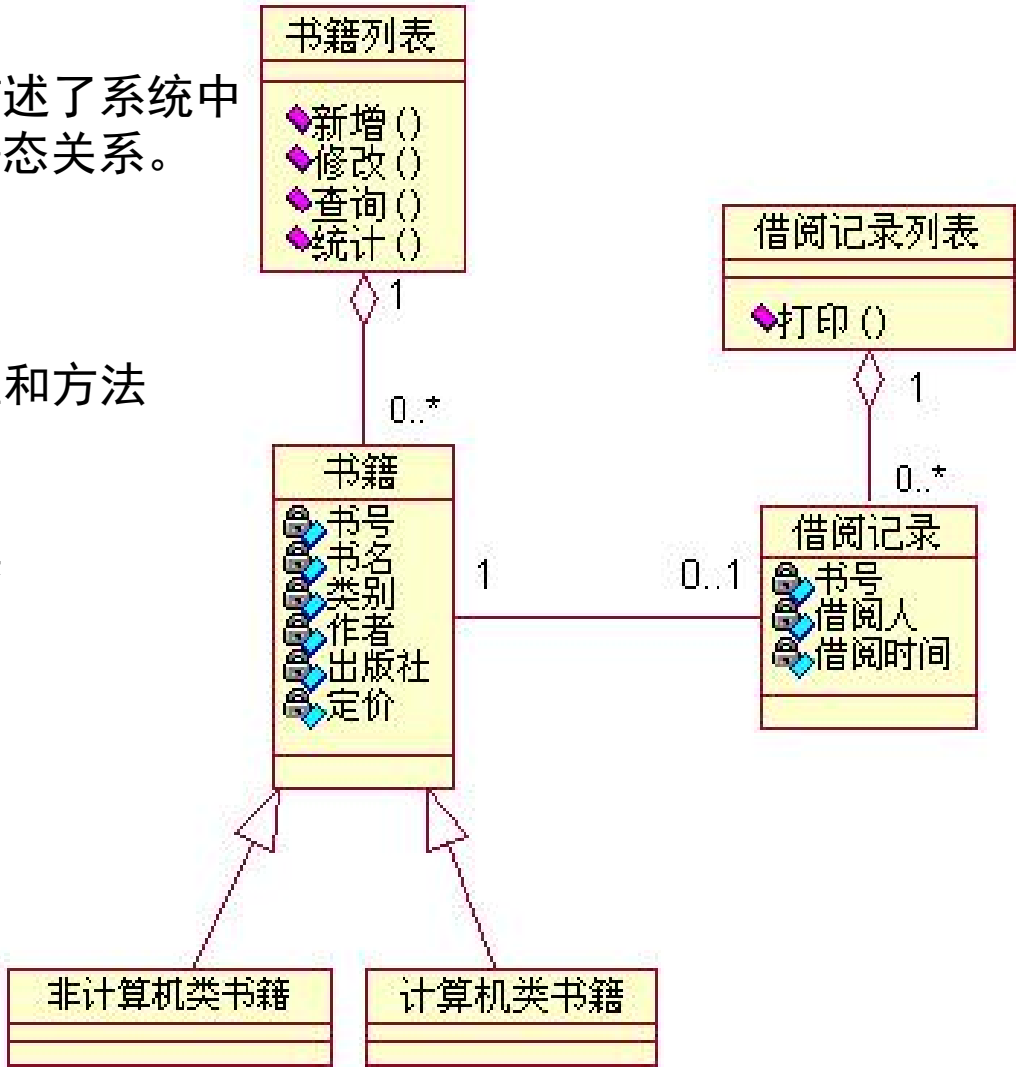
关联关系:一个类知道另一个类的属性和方法

聚合关系:整体与部分生命周期不同

组合关系:整体与部分生命周期相同

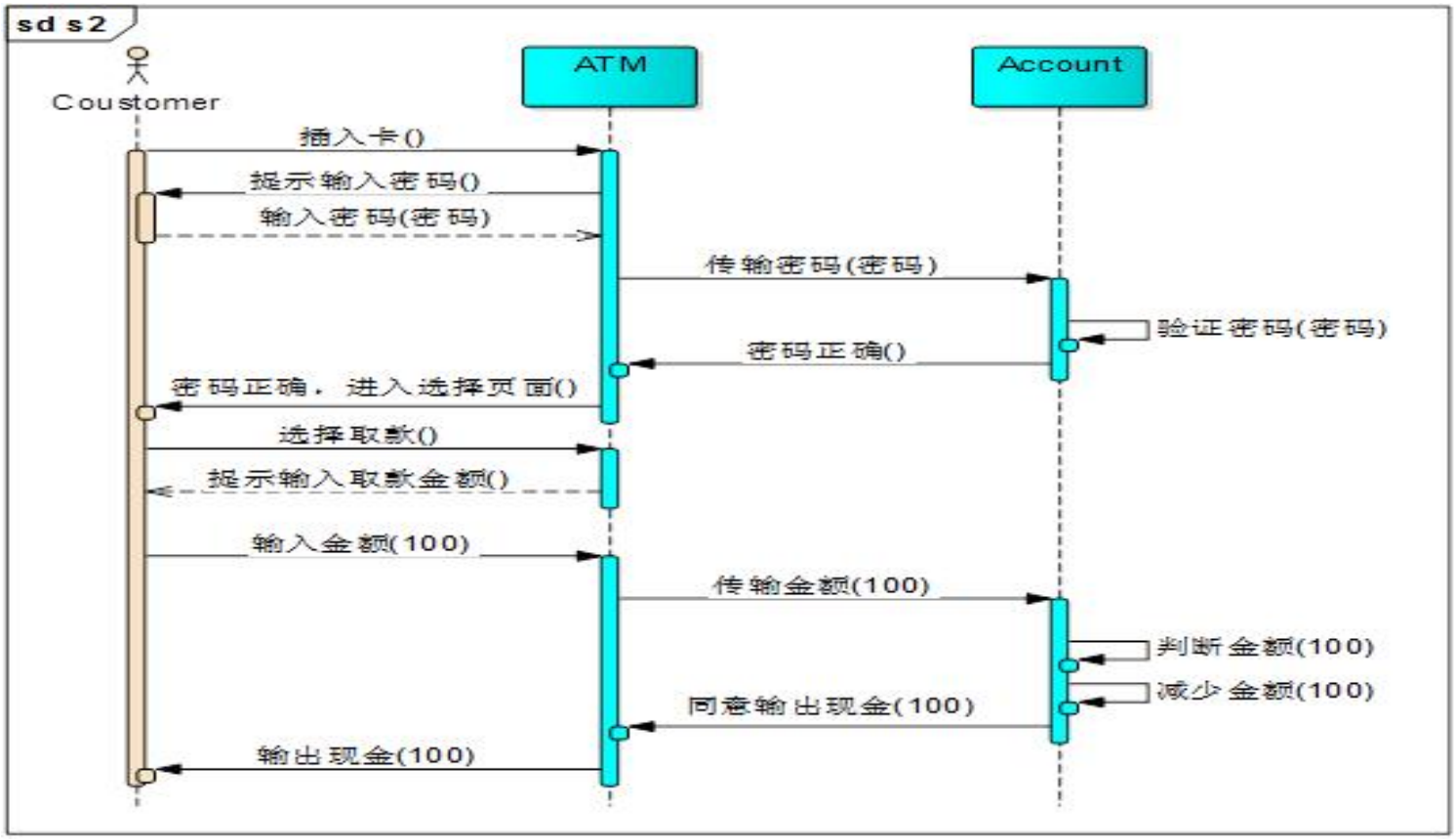
依赖关系:一个类变化影响另外一个类





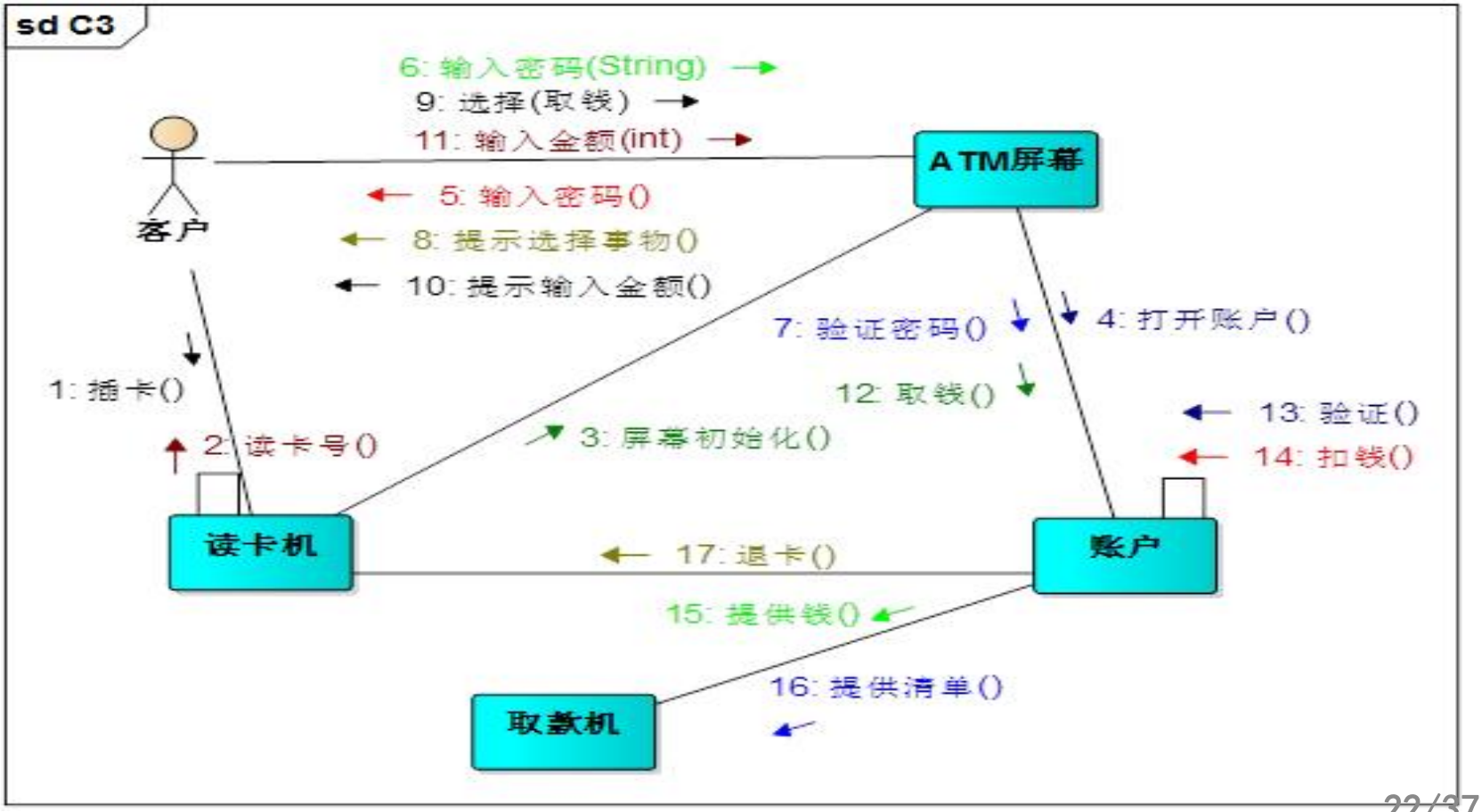
交互图-顺序图

顺序图:描述按时间的先后顺序对象之间的交互过程，纵向是时间轴，时间沿竖线向下延伸。顺序图对象，消息，生命线(激活)构成。



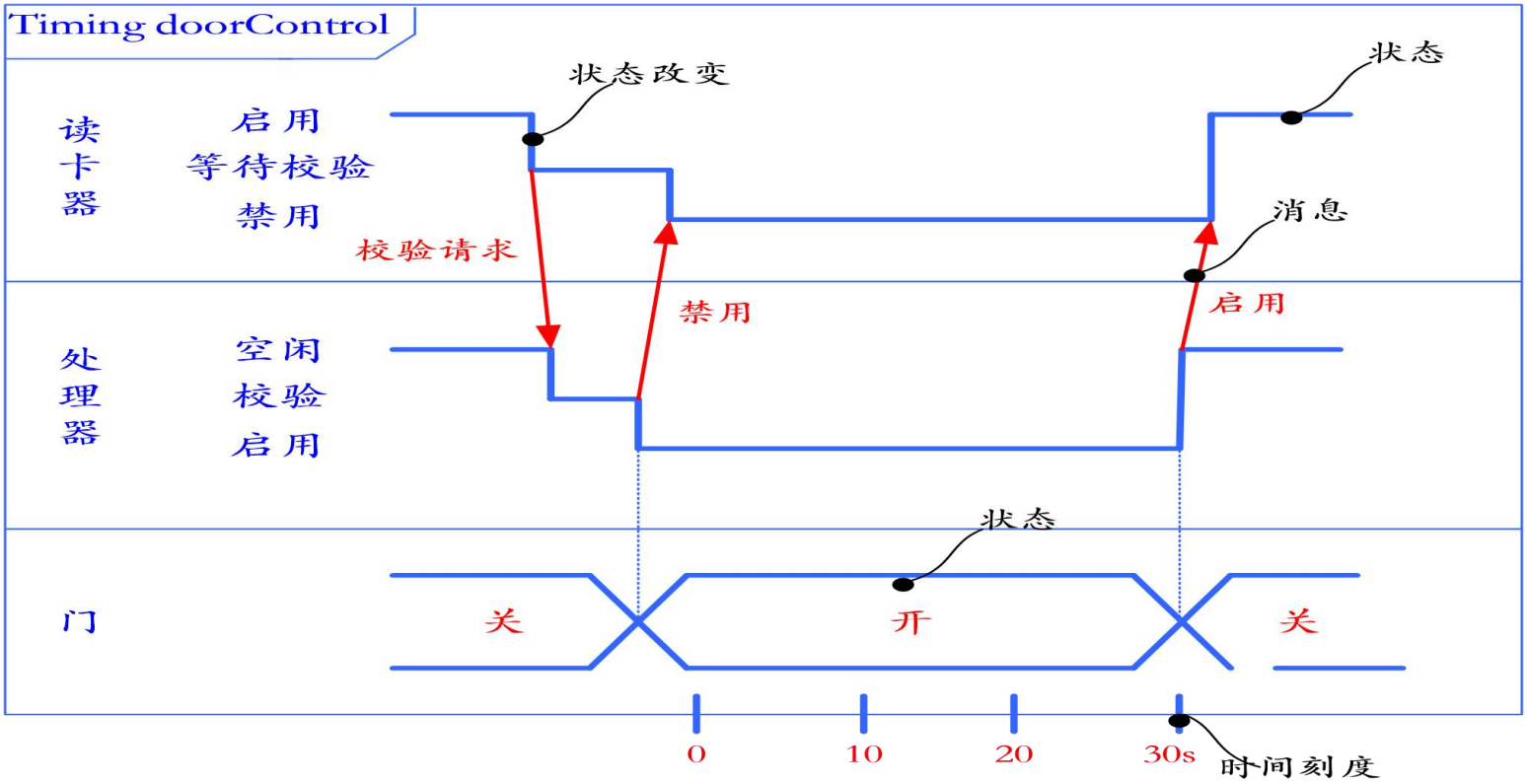
交互图-通信图

又称协作图，是表示对象交互关系的图，表示多个对象在达到共同目标的过程中互相通信的情况。强调收发消息的对象或参与者的结构组织关系。



交互图-定时图

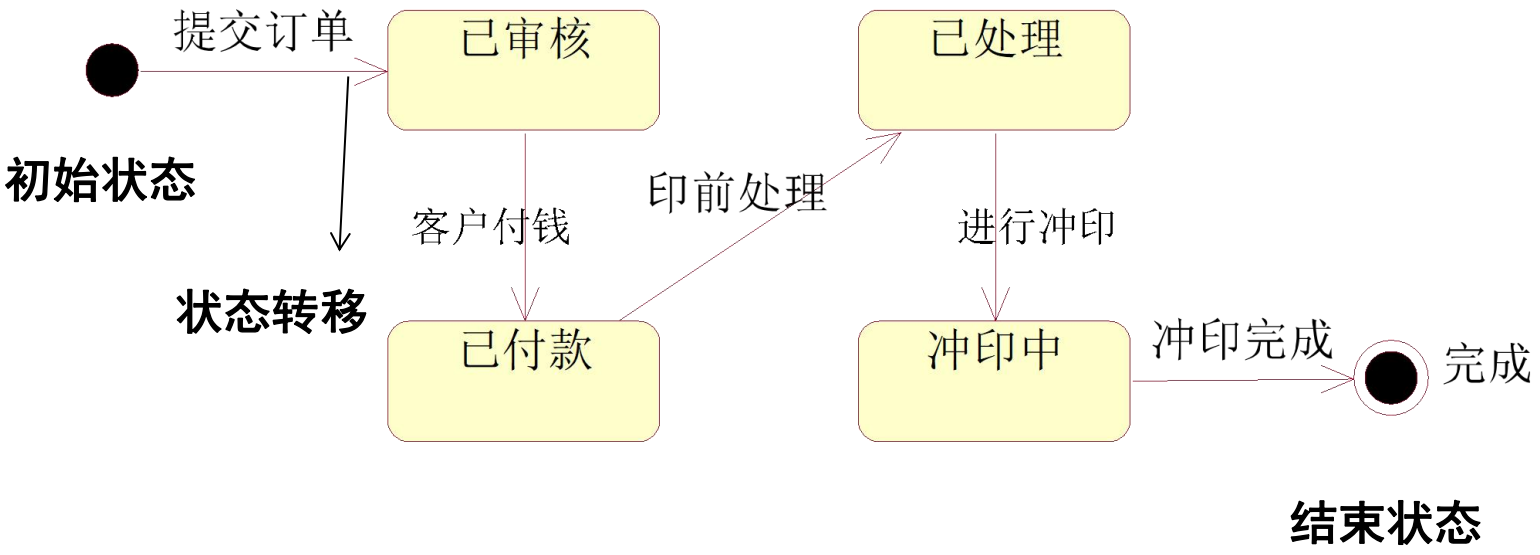
定时图:采用一种带数字刻度的时间轴来精确地描述消息的顺序，允许可视化地表示每条生命线的状态变化，适用于实时事件的建模。



新的开始，新的起点，让我们一起为梦想而努力。

状态图

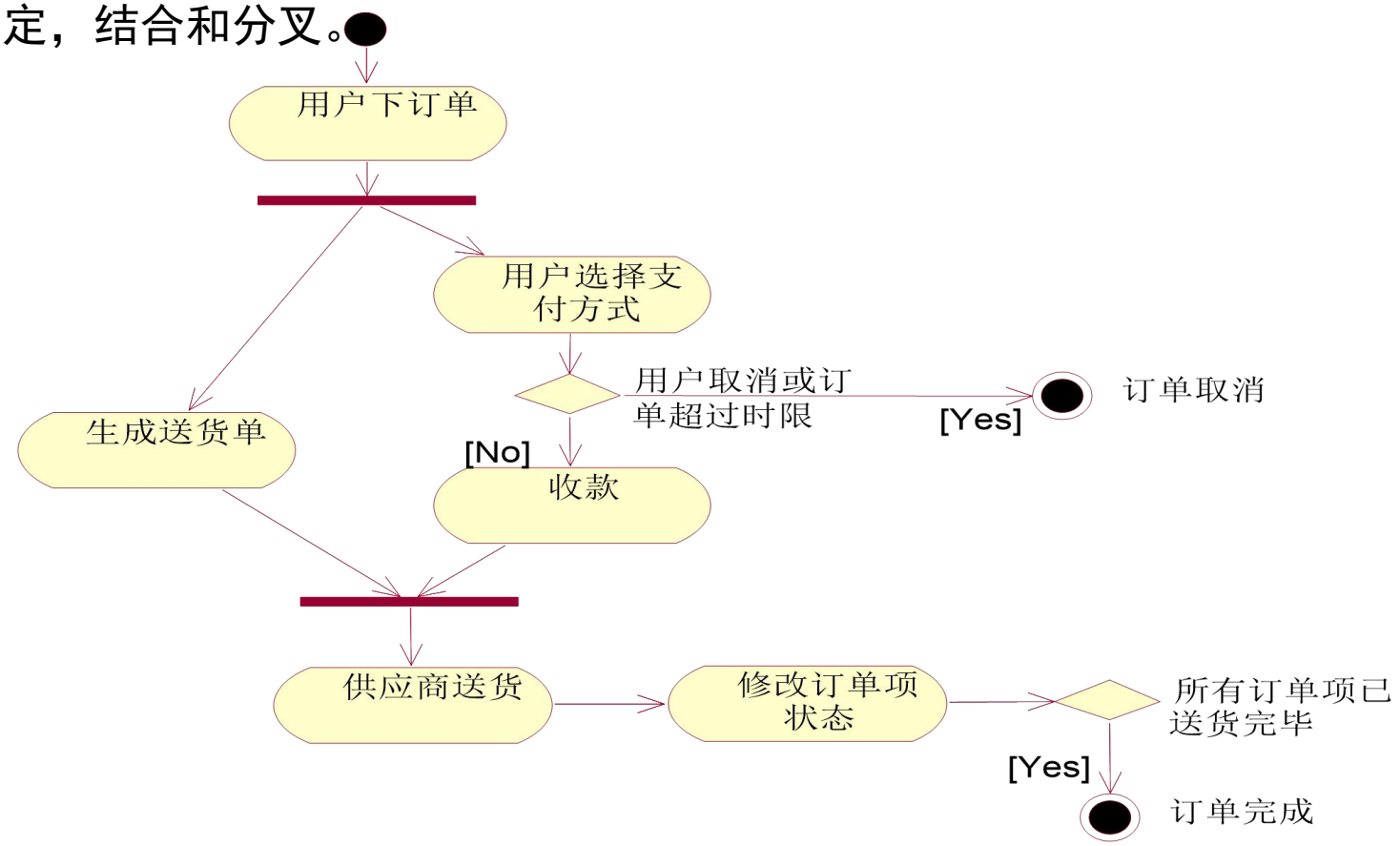
状态图:描述对象在生命周期中的各种状态及状态的转换。
如图：描述一个订单对象的状态变化过程。



状态图适合用于表述在不同用例之间的对象行为，但并不适合于表述包括若干协作的对象行为。通常不会需要对系统中的每一个类绘制相应的状态图，而通常会在业务流程、控制对象、用户界面的设计方面使用状态图。

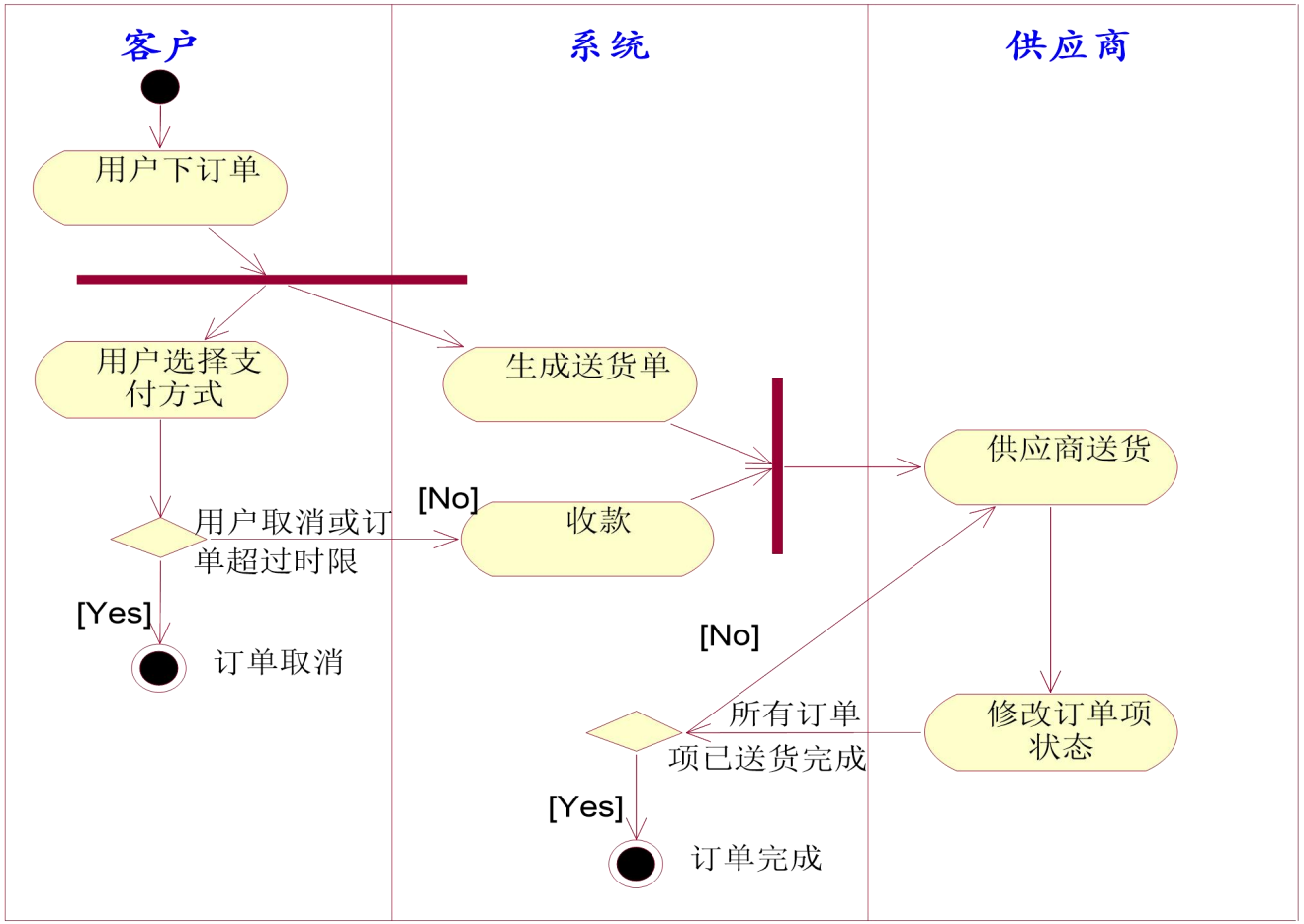
基本活动图

活动图：描述一系列具体动态过程的执行逻辑，展现活动和活动之间转移的控制流，并且它采用一种着重逻辑过程的方式来叙述。相比状态图增加了判定，结合和分叉。



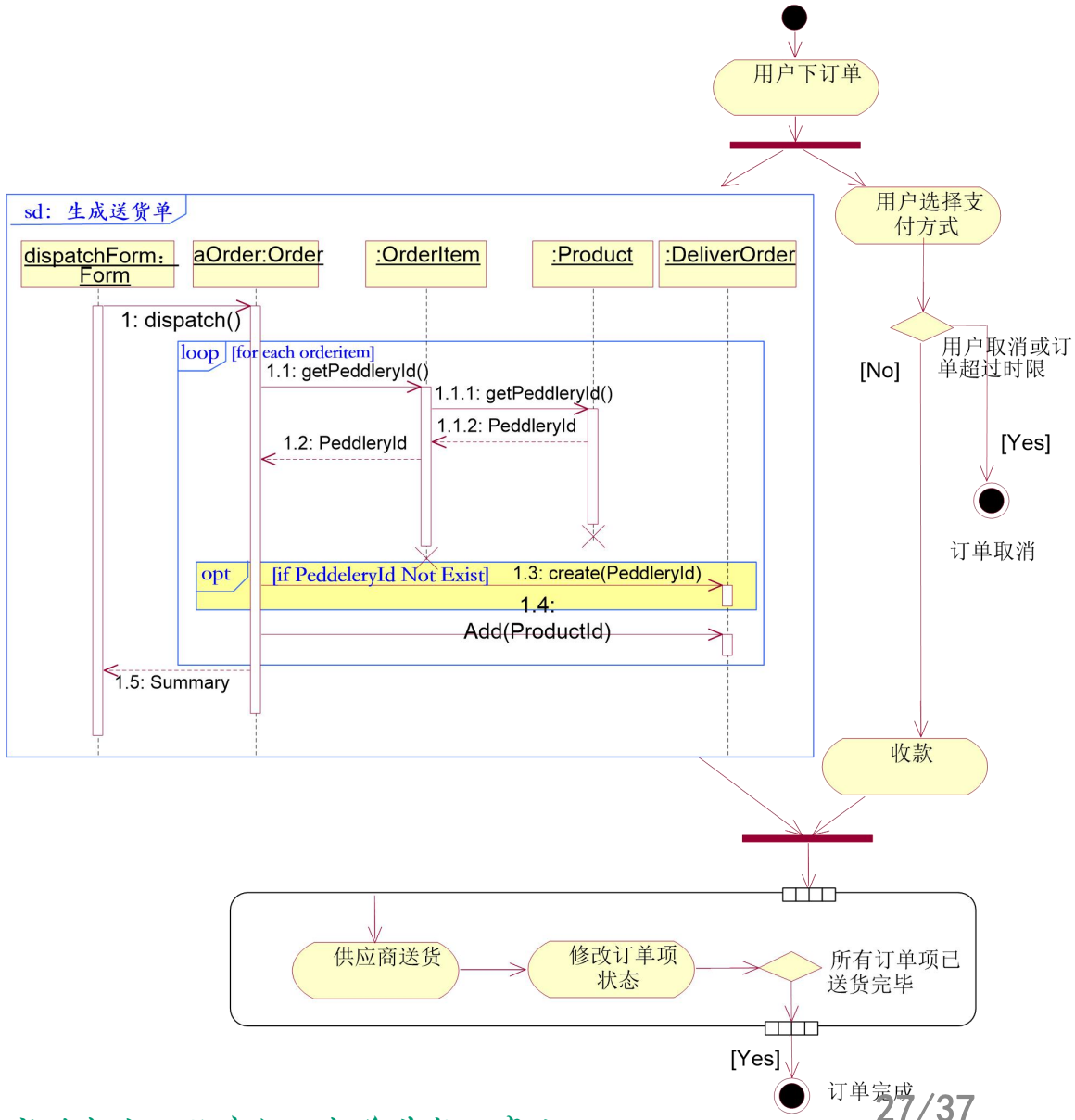
带泳道的活动图

相比活动图更能够说明完成活动的对象



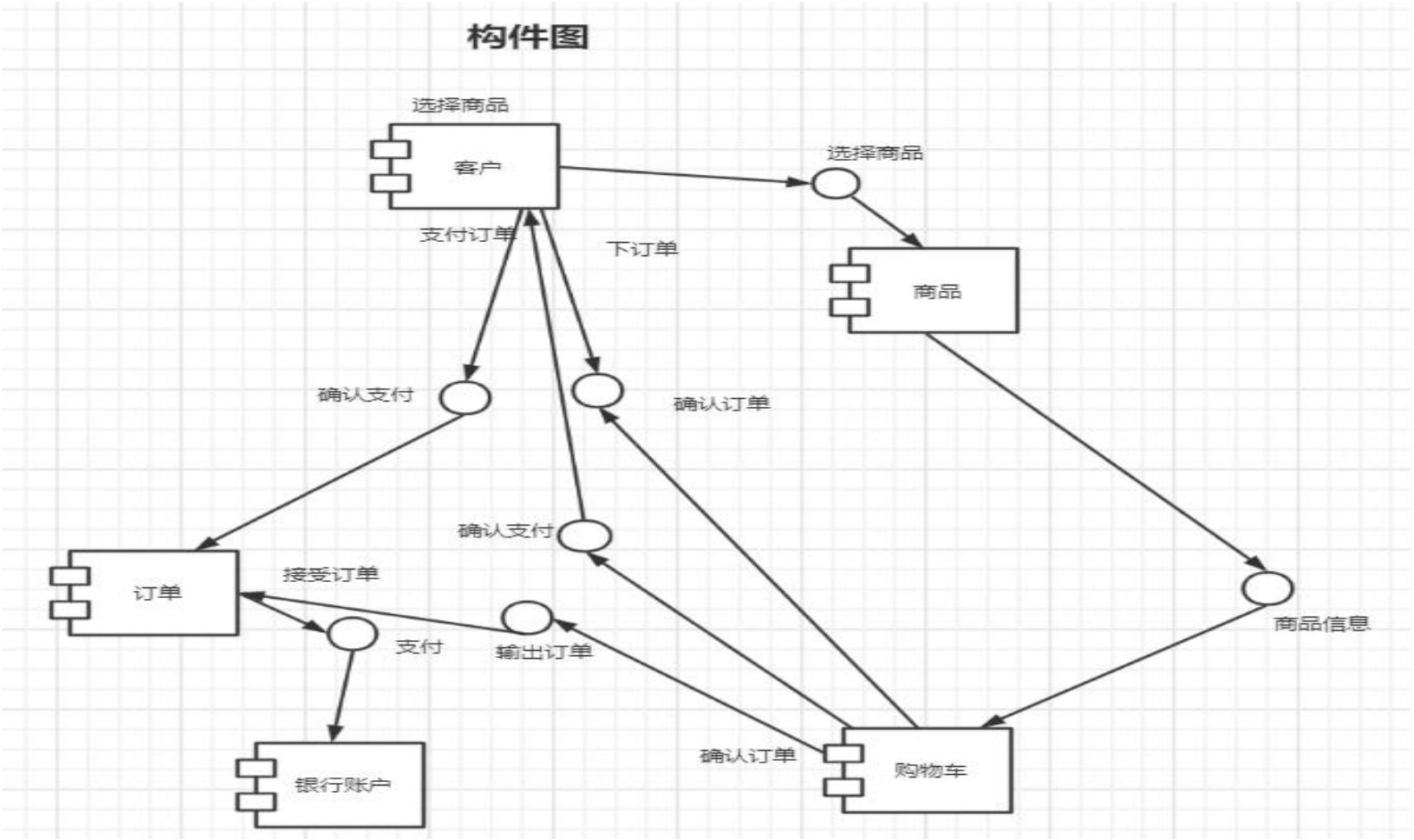
交互图-交互概览图

交互概览图是活动图的一种形式，其中节点表示交互图(顺序图，状态图)，交互概览图的大多数符号与活动图相同。



构件图

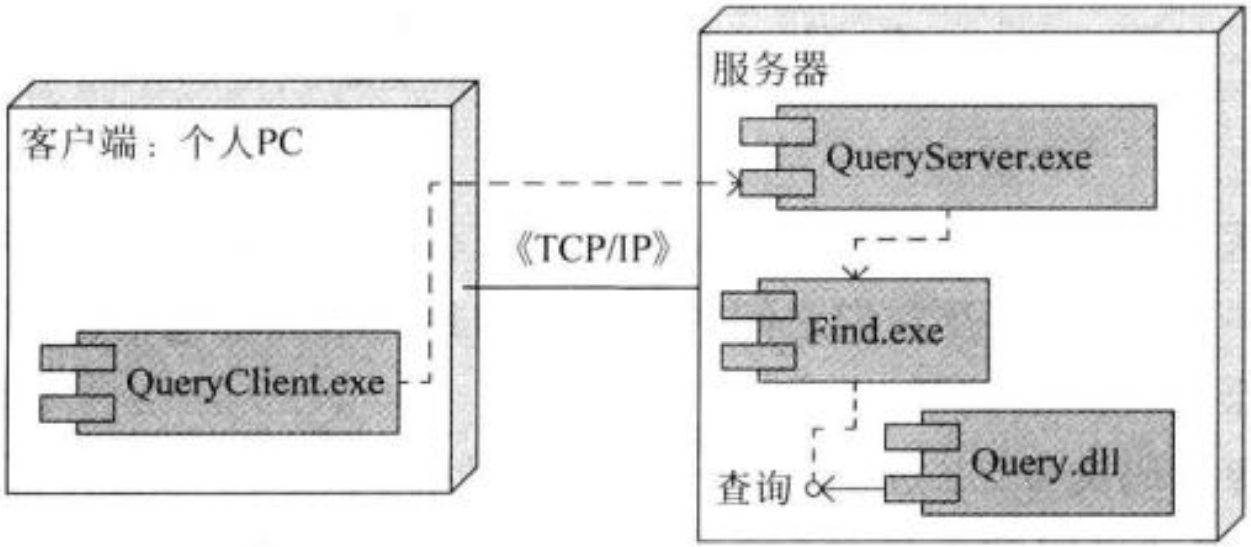
各构件之间相互依赖关系的图，如客户下单构件图：



新的开始，新的起点，让我们一起为梦想而努力。

部署图/实施图

描述处理器、硬件设备和软件构件在运行时的架构，它显示系统硬件的物理拓扑结构及在此结构上执行的软件



XML文档

```
<?xml version="1.0"?>
<visit>
        <to>alluser</to>
        <from>educity.cn</from>
        <heading>welcome</heading>
        <body>Welcome to the best online education website!</body>
</visit>
```

(1) 文档以XML定义<?xml version="1.0"?>开始。

(2) 有一个包含所有其它内容的根元素，如上面例子中的<visit>和</visit>标记符。

(3) 所有元素必须合理地嵌套，不允许交叉嵌套。

XML概述

XML的功能：

1、数据存储:(各类配置文件)
2、数据传输:(各类接口)

XML与html的区别：

1、XML标签都是自定的，HTML标签是预定义的
2、XML语法严格，HTML语法松散
3、XML主要用于数据存储和传输，HTML用于数据显示

RIA的概念 – 传统WEB程序的缺点

操作复杂性

由于受传统Web应用程序的局限性，当进行一个多步骤或多选项的事务时，用户要么会看到一份很长的、笨拙的页面，要么就得通过反复翻转若干网页、执行多步操作。

数据复杂性

高效率地表达复杂的数据，是现有Web应用程序所面临的巨大挑战。理想的图形工具应该能够既操作简便，又能生动明了地展示各种错综复杂的数据信息。

交互复杂性

互动性需求的应用程序使得交互的问题变得日益突出，用户的耐心变得越来越少，他们的要求是要向桌面应用程序的速度靠齐。

RIA的概念 – “富”的含义

丰富的数据模型

丰富的数据意味着客户端的用户界面能表现和应对更多更复杂的数据模式，这样才能处理客户端的运算以及异步发送、接收数据。为了达到高度复杂的数据模式,客户端允许用户构建一个高响应、交互式的应用程序。

丰富的用户界面

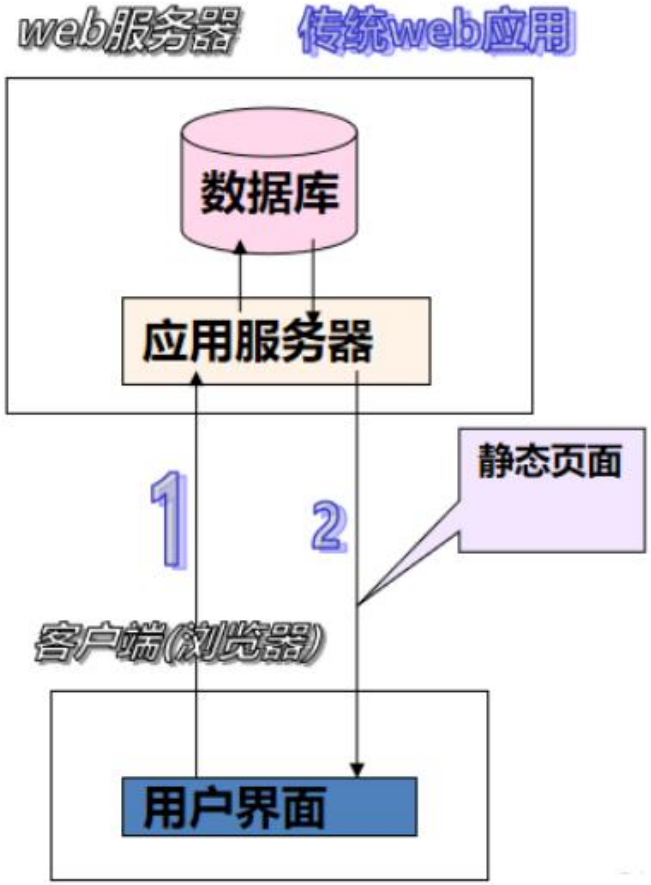
HTML只能为用户的界面控制提供有限的功能，反之，RIA允许一些富有创造性的界面控制，巧妙地与数据模式相合。伴随着丰富的用户界面，用户可以从早期的服务器响应影响整个界面的运作模式，迁移到只对发出请求的特定区域进行改变的模式上来。本质上，意味着界面将会被分解为由单独个体组成，来适应局部改变、服务器交互，以及客户端内部构件的通信。

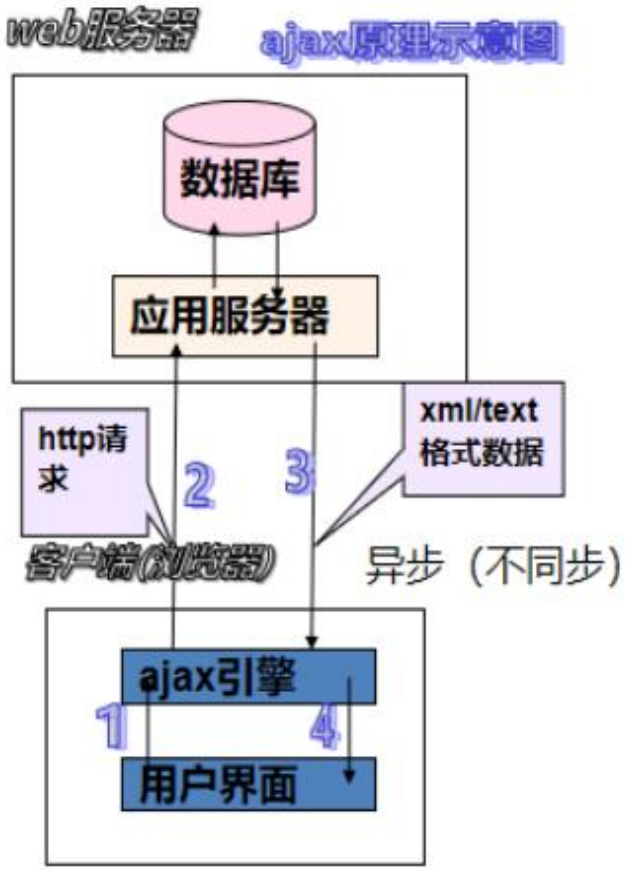
RIA客户端开发技术 – AJAX

Asynchronous JavaScript And XML

- 基于XHTML和CSS标准的表示
- 使用DOM进行动态显示和交互
- 使用 XML 和 XSLT 进行数据交换及相关操作
- 使用XMLHttpRequest与服务器进行异步通信
- 使用JavaScript绑定一切

RIA客户端开发技术 – AJAX





RIA客户端开发技术 – AJAX

优点：

异步交互，局部更新。

改进的传统web程序数据复杂性，交互复杂性，操作复杂性的缺点

缺点：

破坏浏览器“后退”按钮的正常行为；

使用动态页面更新时，用户难以将某个特定的状态保存到收藏夹中。

使用AJAX开发时要慎重考虑网络延迟

质量属性 – 性能

- 性能是指系统的响应能力，即要经过多长时间才能对某个事件做出响应，或者在某段事件内系统所能处理的事件的个数。
- 经常用单位时间内所处理事务的数量或系统完成某个事务处理所需的时间来对性能进行定量的表示。
- 性能测试经常要使用基准测试程序（用以测量性能指标的特定事务集或工作量环境）。

质量属性 – 可靠性

可靠性是软件系统在应用或系统错误面前，在意外或错误使用的情况下维持软件系统的功能特性的基本能力。可靠性通常用平均失效等待时间（MTTF）和平均失效间隔时间（MTBF）来衡量。在失效率为常数和修复时间很短的情况下，MTTF和MTBF几乎相等。

- **容错。**其目的是在错误发生时确保系统正确的行为，并进行内部“修复”。例如，在一个分布式软件系统中失去了一个与远程构件的连接，接下来恢复了连接。在修复这样的错误之后，软件系统可以重新或重复执行进程间的操作直到错误再次发生。
- **健壮性。**这里说的是保护应用程序不受错误使用和错误输入的影响，在遇到意外错误事件时确保应用系统处于已经定义好的状态。

❁ 质量属性 – 可用性

可用性是**系统能够正常运行的时间比例**。经常用两次故障之间的时间长度或在出现故障时系统能够恢复正常的速度来表示。

质量属性 – 安全性

- 安全性是指系统在向合法用户提供服务的同时能够阻止非授权用户使用的企图或拒绝服务的能力。安全性是根据系统可能受到的安全威胁的类型来分类的。
- 安全性又可划分为机密性、完整性、不可否认性及可控性等特性。其中，机密性保证信息不泄露给未授权的用户、实体或过程；完整性保证信息的完整和准确，防止信息被非法修改；可控性保证对信息的传播及内容具有控制的能力，防止为非法者所用。

质量属性 – 可修改性

- 可维护性。在错误发生后“修复”软件系统。为可维护性做好准备的软件体系结构往往能做局部性的修改并能使对其他构件的负面影响最小化。
- 可扩展性。使用新特性来扩展软件系统，以及使用改进版本来替换构件并删除不需要或不必要的特性和构件。
- 结构重组。重新组织软件系统的构件及构件间的关系，例如通过将构件移动到一个不同的子系统而改变它的位置。
- 可移植性。可移植性使软件系统适用于多种硬件平台、用户界面、操作系统、编程语言或编译器。

质量属性 – 功能性

功能性是系统所能完成所期望的工作的能力。一项任务的完成需要系统中许多或大多数构件的相互协作。

质量属性 – 可变性

- 可变性是指**体系结构经扩充或变更而成为新体系结构的能力**。这种新体系结构应该符合预先定义的规则，在某些具体方面不同于原有的体系结构。
- 当要将某个体系结构作为一系列相关产品（例如，软件产品线）的基础时，可变性是很重要的。

质量属性 – 可集成性

可集成性是指系统能与其他系统协作的程度。

质量属性 – 互操作性

作为系统组成部分的软件不是独立存在的，经常与其他系统或自身环境相互作用。为了支持互操作性，软件体系结构必须为外部可视的功能特性和数据结构提供精心设计的软件入口。程序和用其他编程语言编写的软件系统的交互作用就是互操作性的问题，这种互操作性也影响应用的软件体系结构。

基本概念 – 敏感点和权衡点

- 敏感点是一个或多个构件（和/或构件之间的关系）的特性。研究敏感点可使设计人员或分析员明确在搞清楚如何实现质量目标时应注意什么。
- 权衡点是影响多个质量属性的特性，是多个质量属性的敏感点。例如，改变加密级别可能会对安全性和性能产生非常重要的影响。提高加密级别可以提高安全性，但可能要耗费更多的处理时间，影响系统性能。如果某个机密消息的处理有严格的时间延迟要求，则加密级别可能就会成为一个权衡点。

(2) 根据以下质量属性绘制质量效用树：

a. 在正常负载下系统应在0.3秒内对用户的界面操作做出响应

b. 系统应具备完整的安全防护措施，支持对恶意攻击行为的检测与报警

c. 系统在一年内因各种故障导致系统不能正常使用的时间不得超过24小时

d. 系统需要对用户输入的手机号进行校验如果输入的手机号格式错误应给与提示

e. 系统每次发生故障后恢复的时间不能超过30分钟

f. 系统支持横向存储扩展，要求在2人天内完成所有的扩展与测试工

g. 系统在调用支付接口的时候传递的参数需要进行加密

h. 系统能够支持200用户的并发访问量

i. 支持对系统的外观进行调整和配置，调整工作需要在4人天内完成

j. 系统需要对用户输入的金额与系统中的约进行对比，如果输入的金额比余额小则提示余额不足 。

设计模式

- 单例模式:这个类提供 了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。
- 工厂方法模式:定义一个用于创建对象的接口，让子类决定实例化哪个产品类对象。工厂方法使一个产品类的实例化延 迟到其工厂的子类。
- 抽象工厂模式:是一种为访问类提供一个创建一组相关或相互依赖对象的接口，且访问类无须指定所要产品的具体类就 能得到同族的不同等级的产品的模式结构。
- 原型模式:用一个已经创建的实例作为原型，通过复制该原型对象来创建一个和原型对象相同的新对象。
- 建造者模式:将一个复杂对象的构建与表示分离，使得同样的构建过程可以创建不同的表示

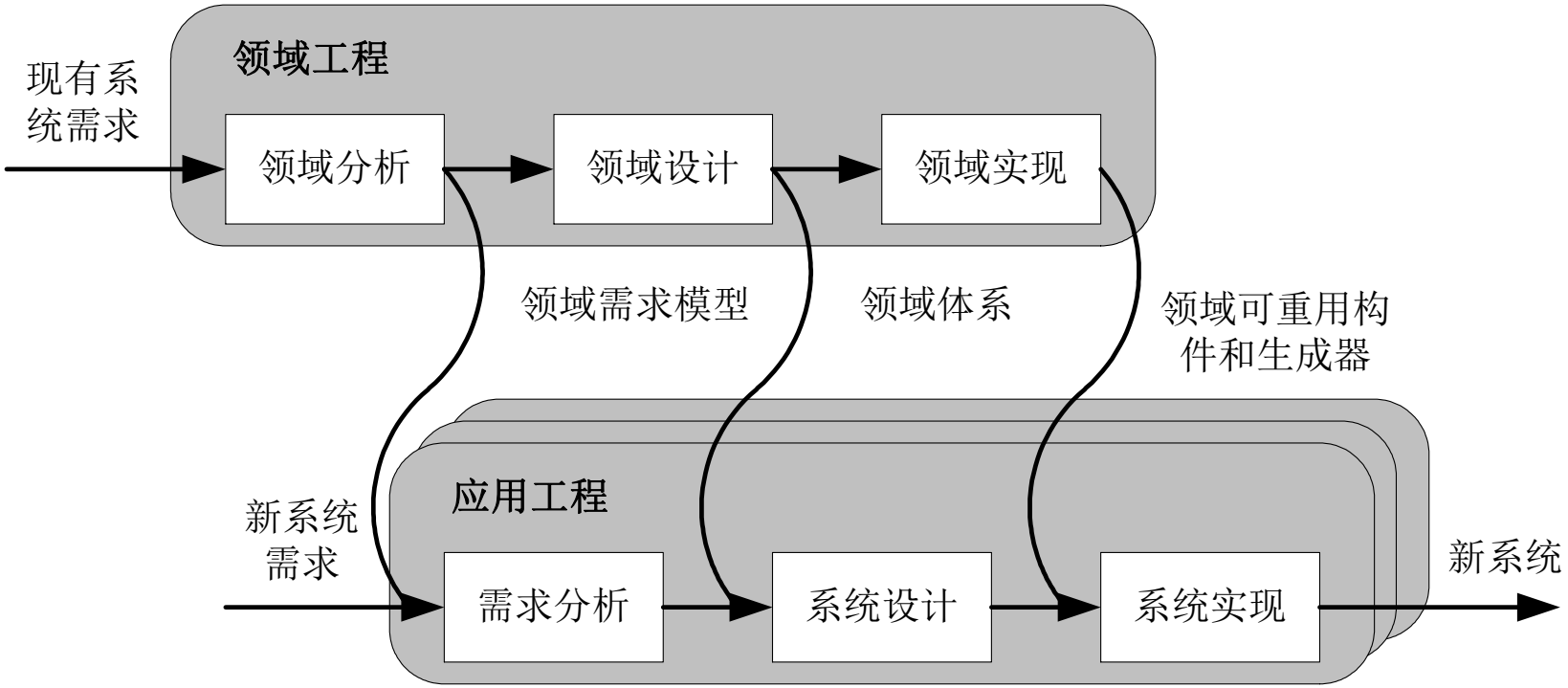
设计模式

- 代理模式：由于某些原因需要给某对象提供一个代理以控制对该对象的访问。这时，访问对象不适合或者不能直接 引用目标对象，代理对象作为访问对象和目标对象之间的中介。
- 适配器模式：将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类能 一起工作。
- 装饰者模式：指在不改变现有对象结构的情况下，动态地给该对象增加一些职责(即增加其额外功能)的模式
- 策略模式:该模式定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的变化不会影响使用 算法的客户。

设计模式

- 观察者模式:又被称为发布-订阅(Publish/Subscribe)模式，它定义了一种一对多的依赖关系，让多个观察者 对象同时监听某一个主题对象。这个主题对象在状态变化时，会通知所有的观察者对象，使他们能够自动更新自己。
- 责任链模式:为了避免请求发送者与多个请求处理者耦合在一起，将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止。

软件产品线的过程模型 – 双生命周期模型



软件产品线的过程模型 – 双生命周期模型

领域工程阶段的主要任务：

领域分析：利用现有系统的设计、体系结构和需求建立领域模型。

领域设计：用领域模型确定领域产品线的共性和可变性，为产品线设计体系结构。

领域实现：基于领域体系结构开发领域可重用的资源(构件，文档，代码生成器等)

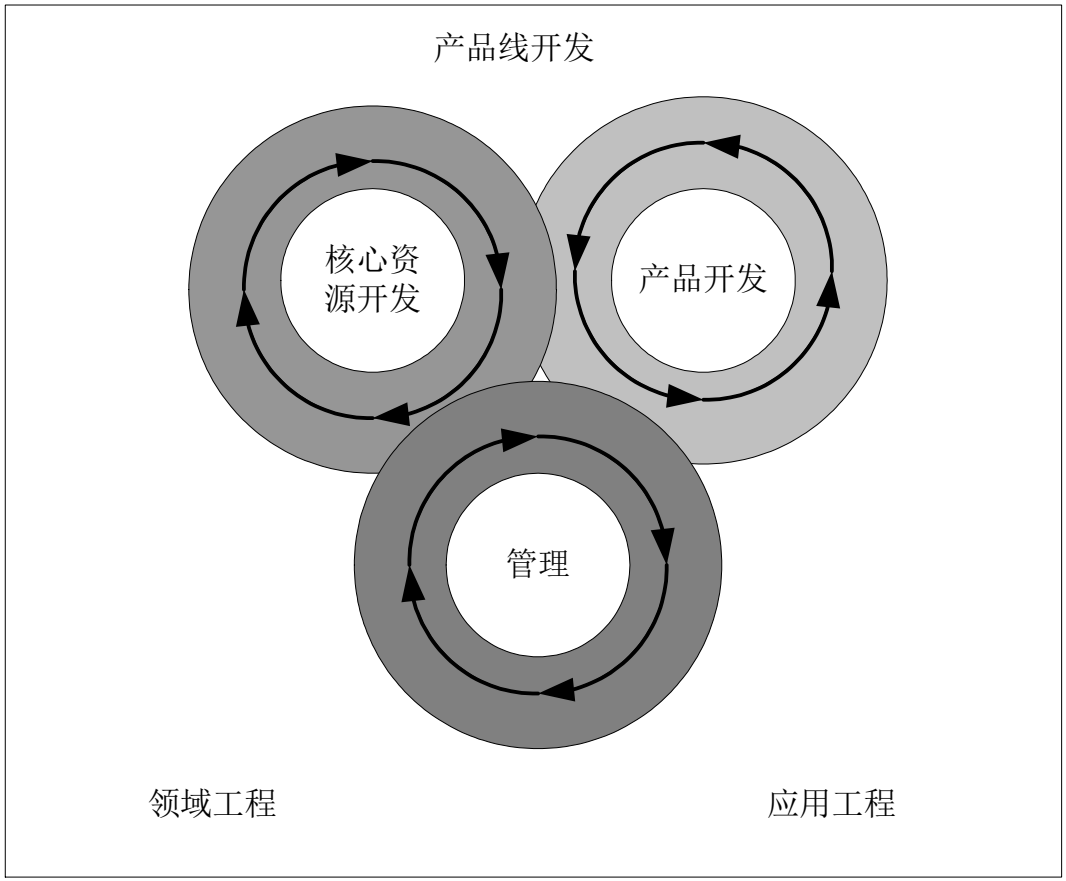
应用工程生成新产品阶段

需求分析：将系统需求与领域需求比较，划分出领域公共需求和独特需求，得出系统说明书。

系统设计：在领域体系结构基础上，结合系统独特需求设计应用的软件体系结构。

系统实现：遵照应用体系结构，用领域可重用资源实现领域公共需求，用定制开发的构件满足系统独特的需求，构件新系统。

软件产品线的过程模型 – SEI模型



- 循环重复是产品线开发过程的特征，也是核心资源开发、产品线开发以及核心资源和产品之间协作的特征；
- 核心资源开发和产品开发没有先后之分；
- 管理活动协调整个产品线开发过程的各个活动，对产品线的成败负责；
- 核心资源开发和产品开发是两个互动的过程，三个活动和整个产品线开发之间也是双向互动的。

软件产品线的过程模型 – 三生命周期模型