Chapter 9

体系结构设计

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

第9章 体系结构设计

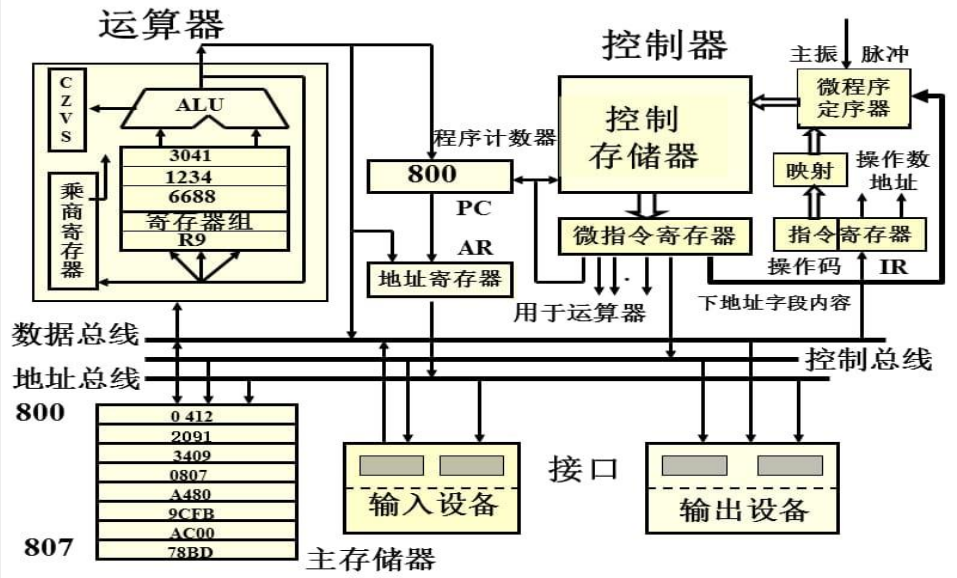
9.1 理解体系结构及设计

(1) 什么是软件体系结构

软件体系结构是软件的一种划分形式，从较高抽象层次定义系统的构件、构件之间的连接，以及由构件与构件连接形成的拓扑结构。

9.1 理解体系结构及设计

- 从计算机看“体系结构”
  - 主要硬件模块：控制器、运算器、内存储器、外存储器、输入设备.....
  - 硬件模块之间的连接关系：总线（控制总线、地址总线、数据总线）



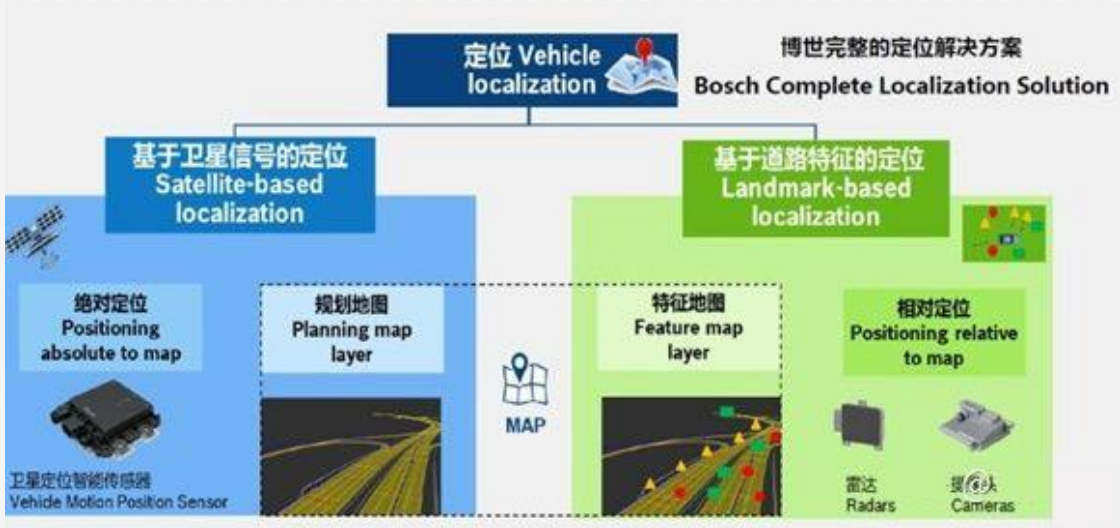
9.1 理解体系结构及设计

- 软件体系结构实例

9.1 理解体系结构及设计

(2) 体系结构设计的必要性

- 体系结构设计决定了系统的主体结构，结构优劣对软件质量具有重要影响。
- 针对安全需求，可以考虑采用分层结构，将重要资源置于内层保护。
- 针对可用性需求，可以考虑冗余设计。



9.1 理解体系结构及设计

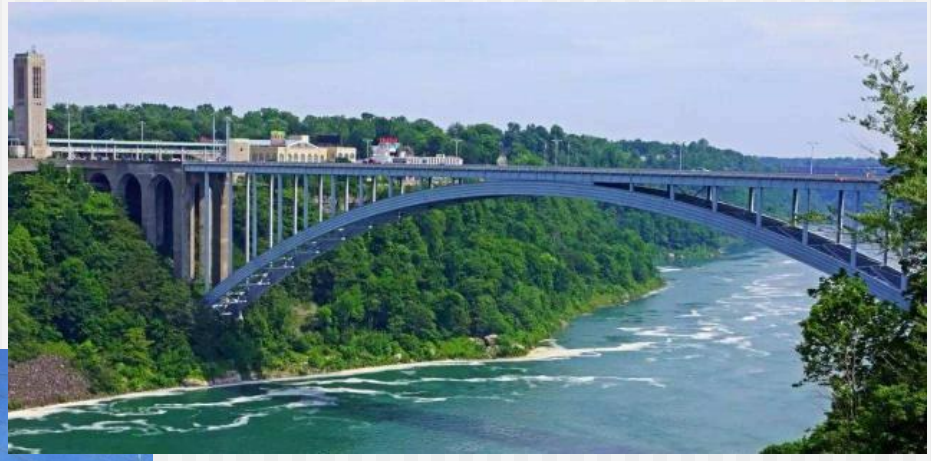
(3) 软件体系结构设计的关键决策

- 系统的基本构造单元是什么？
  - 图像获取 、图像矫正、图像去噪……
- 这些构造单元连接方式？
  - 函数调用 or 消息 or 通信中间件……
- 最终形成怎样的拓扑结构？
- 有哪些体系结构风格可供系统借鉴？
- 对于目标系统，如何基于一般性的体系结构进行适应性改造？
- 如何评估体系结构设计？

9.2 软件体系结构风格

- 体系结构风格

建筑风格：可区分建筑的一种模式，如外形、材料、工艺技术等。

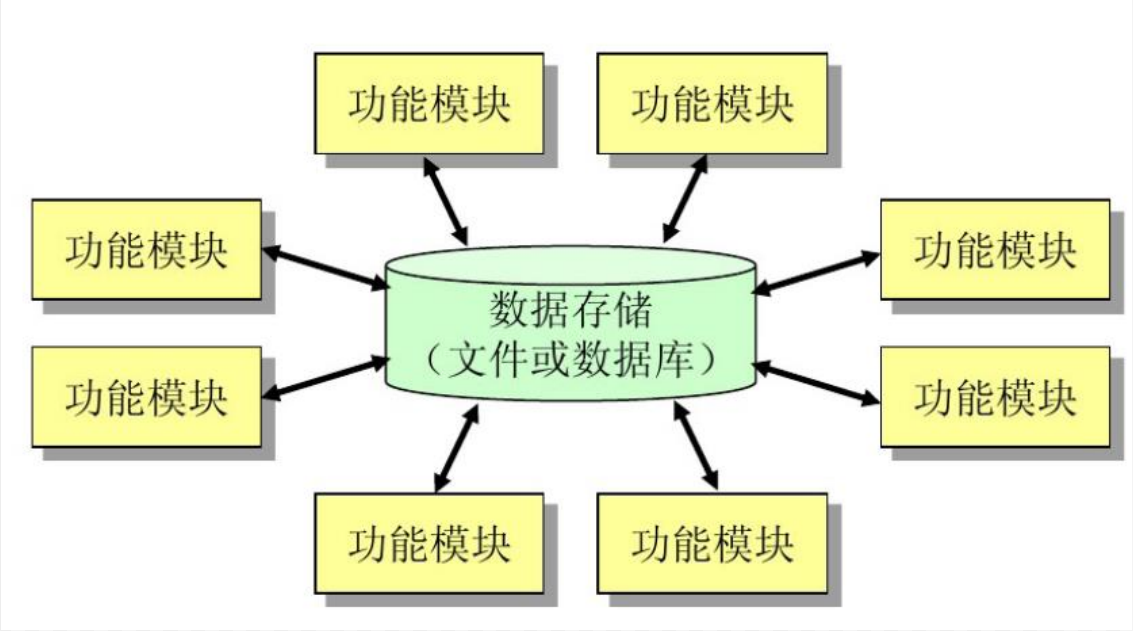


9.2 软件体系结构风格

- 计算机系统也有特定的“风格”。
- 众多系统所共有的结构和特性(如构件类型、交互机制、拓扑、约束)，可以指导同类型应用的体系结构设计。
  - 数据中心结构
  - 管道-过滤器结构
  - 调用返回结构
  - 消息总线结构
  - 面向对象结构
  - 层次化结构

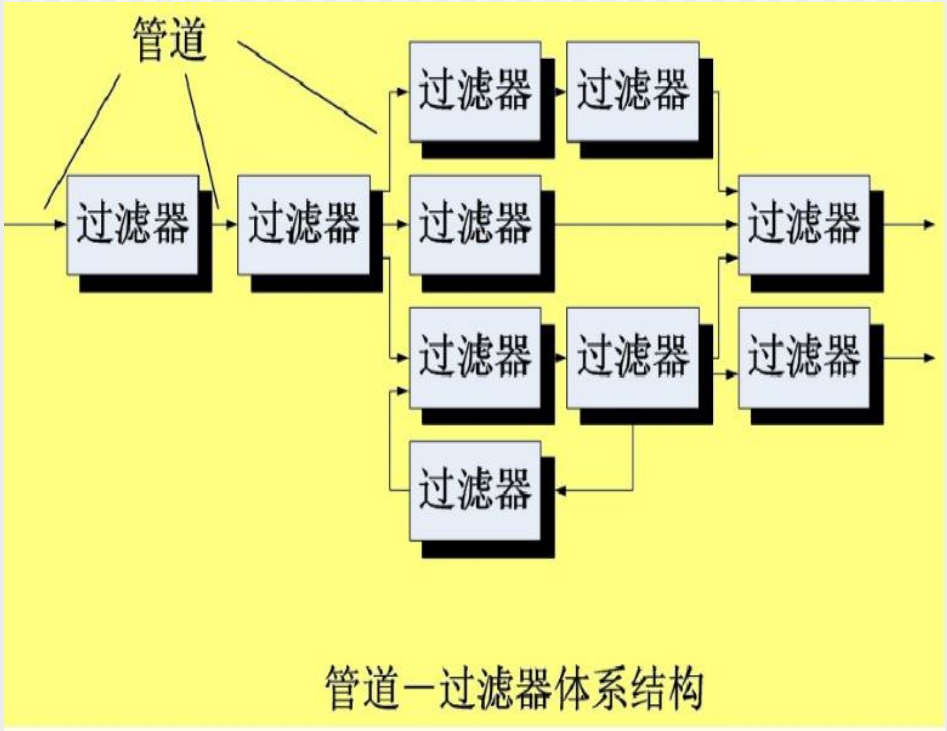
9.2.1 数据中心结构

- 该结构适合于：数据由一个构件(模块)产生，而由其它构件使用的情形。
- 缺点：一旦中心数据存储出现问题会影响整个系统。
- 如电子白板系统、管理信息系统（MIS） 。



9.2.2 管道-过滤器结构

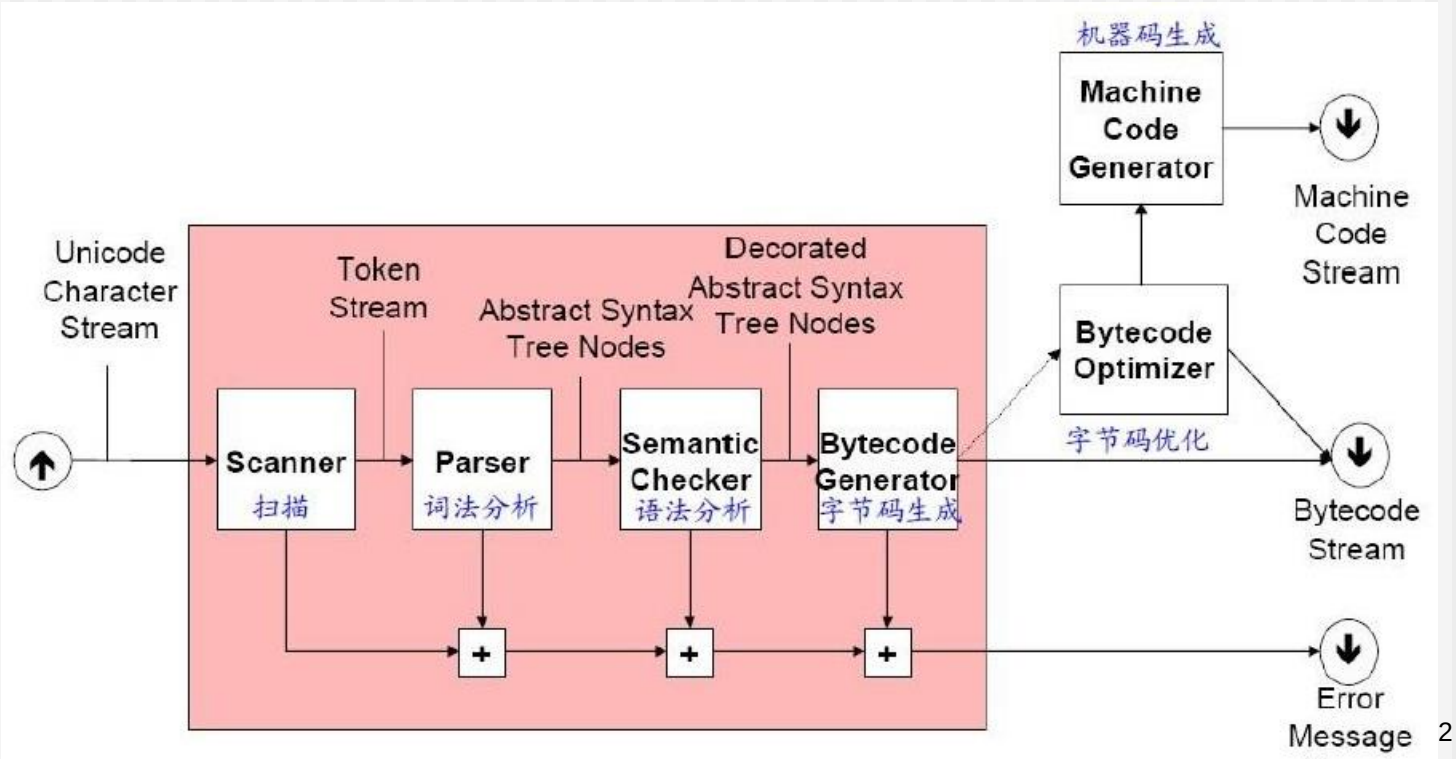
- 系统由若干构件(过滤器)按一定顺序组合而成，一个构件的输出是另一个构件的输入。
- 适合于批处理系统，不适用于交互式应用系统。



管道—过滤器体系结构

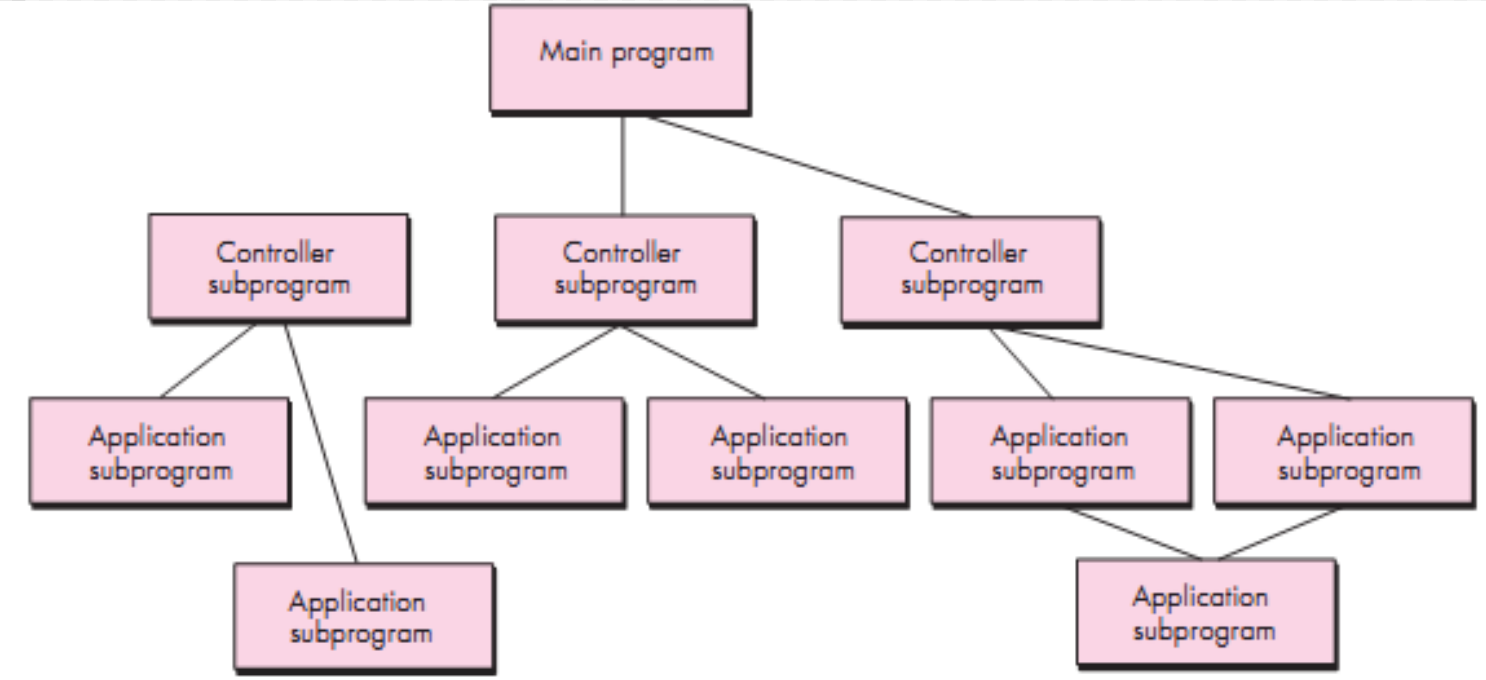
9.2.2 管道-过滤器结构

- 编译器就是一种管道-过滤器系统，包括词法分析、语法分析、字节码生成等。
- 图像处理系统等也大多采用该风格。



9.2.3 调用返回结构

- (1) 主程序/子程序结构

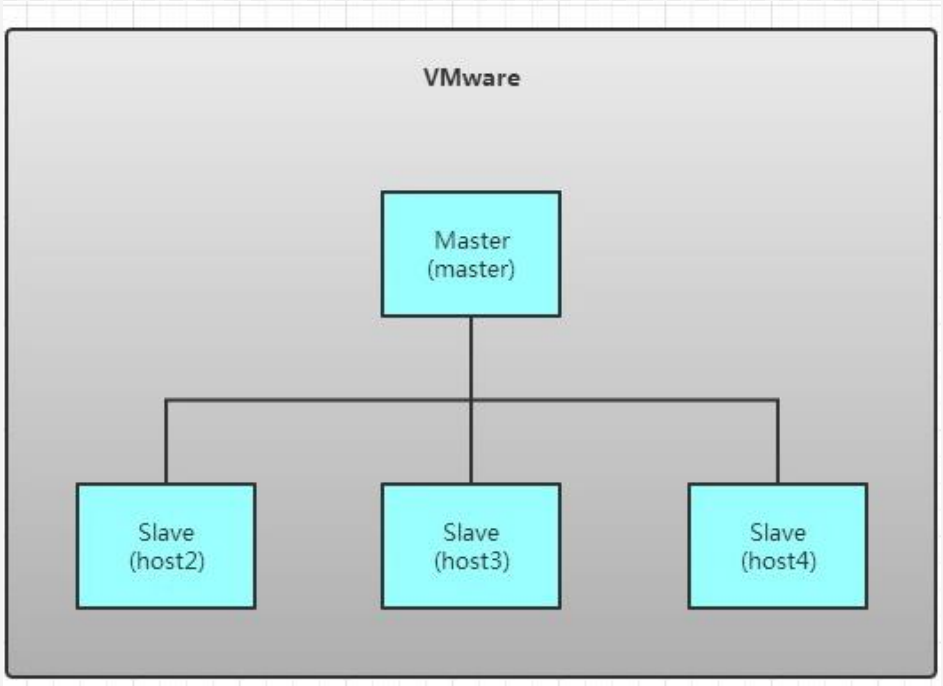


9.2.3 调用返回结构

- (2)远程过程调用结构

主程序/子程序中的构件分布在网络中的多台计算机。

如：如分布式计算系统Hadoop。

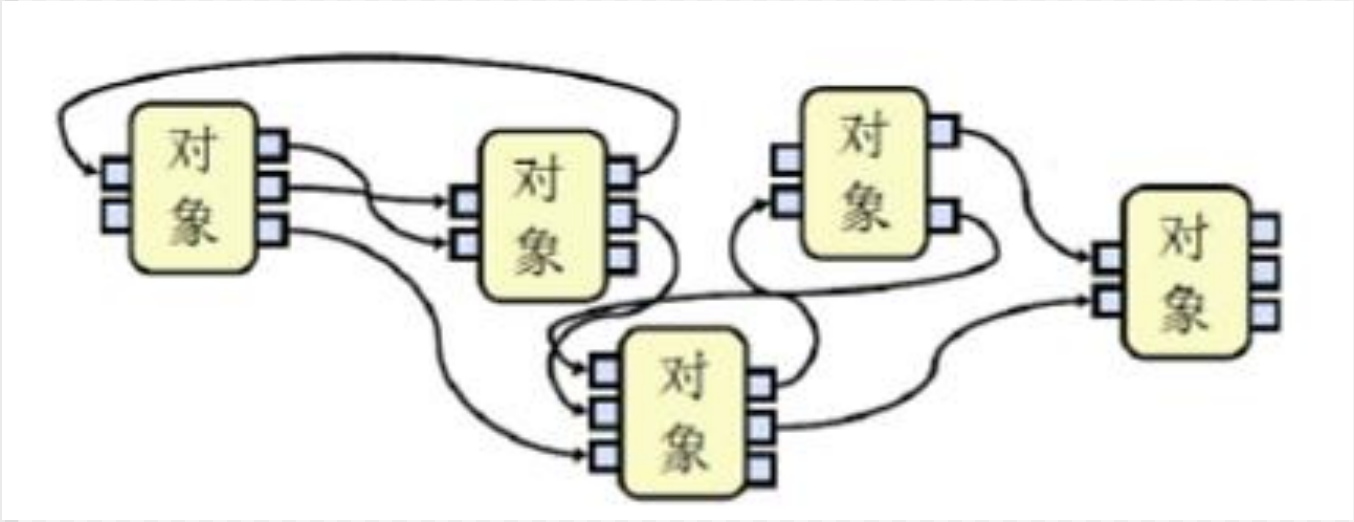


9.2.3 调用返回结构

- 调用返回结构缺点
  - 主构件需要预先知道被调子构件的接口标识。
  - 构件接口标识的更改会影响所有调用它的其它构件，而且这种现象还会进行传递，导致更大的负面影响。

9.2.4 面向对象结构

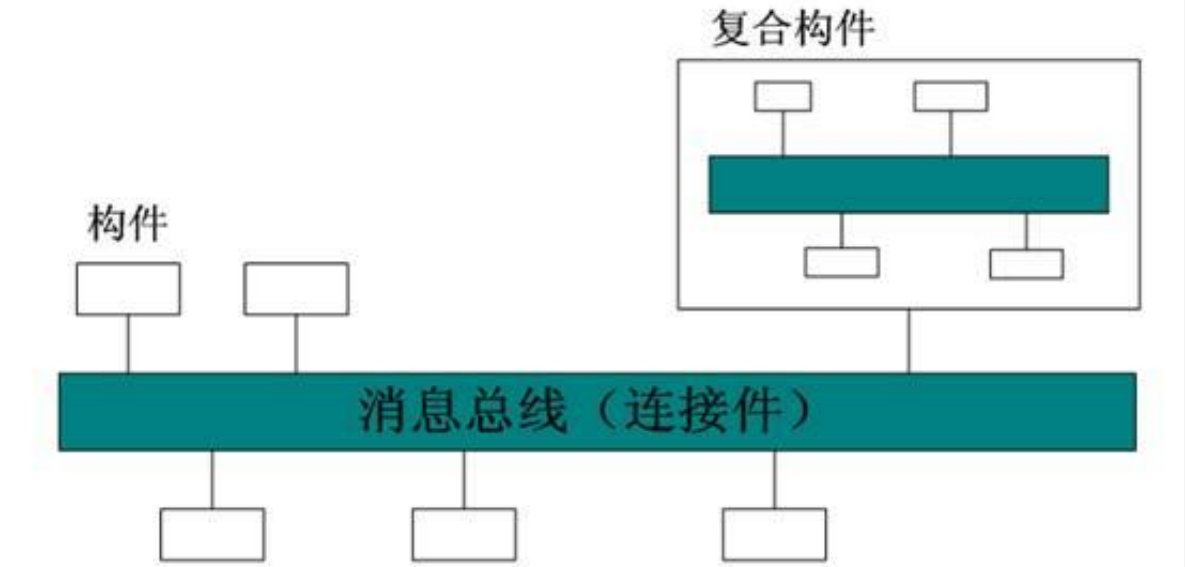
- 构造单元：类和对象
- 连接：基于消息机制。
- 拓扑结构：不同对象之间是平级的，没有主次之分。



如：自主足球机器人系统

9.2.5 消息总线结构

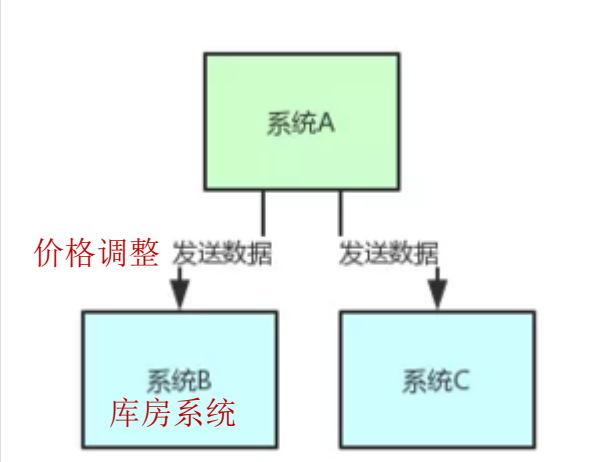
- 构件不直接调用其它构件，而是广播一个或多个事件。
- 系统中的构件注册(监听)一个或多个事件源；
- 当一个事件被广播，系统自动调用该事件中注册的构件。

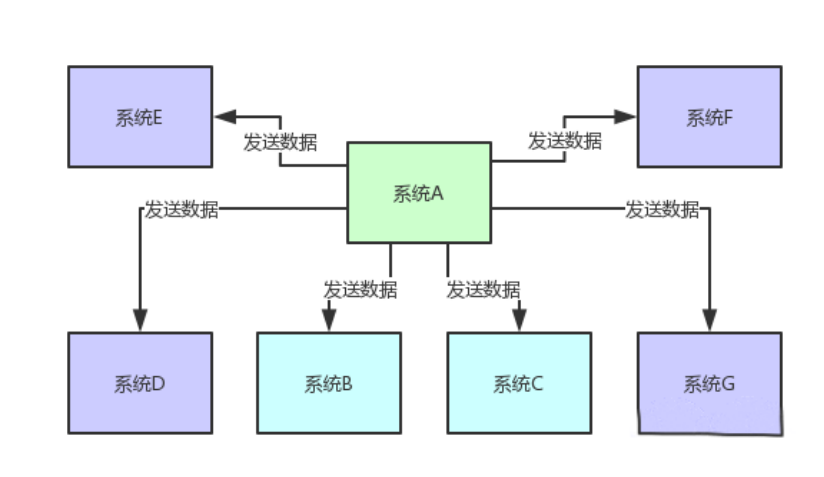


9.2.5 消息总线结构

- 消息总线风格逐渐演变为消息中间件。
- 消息中间件应用场景：系统解耦、异步通讯、削峰。

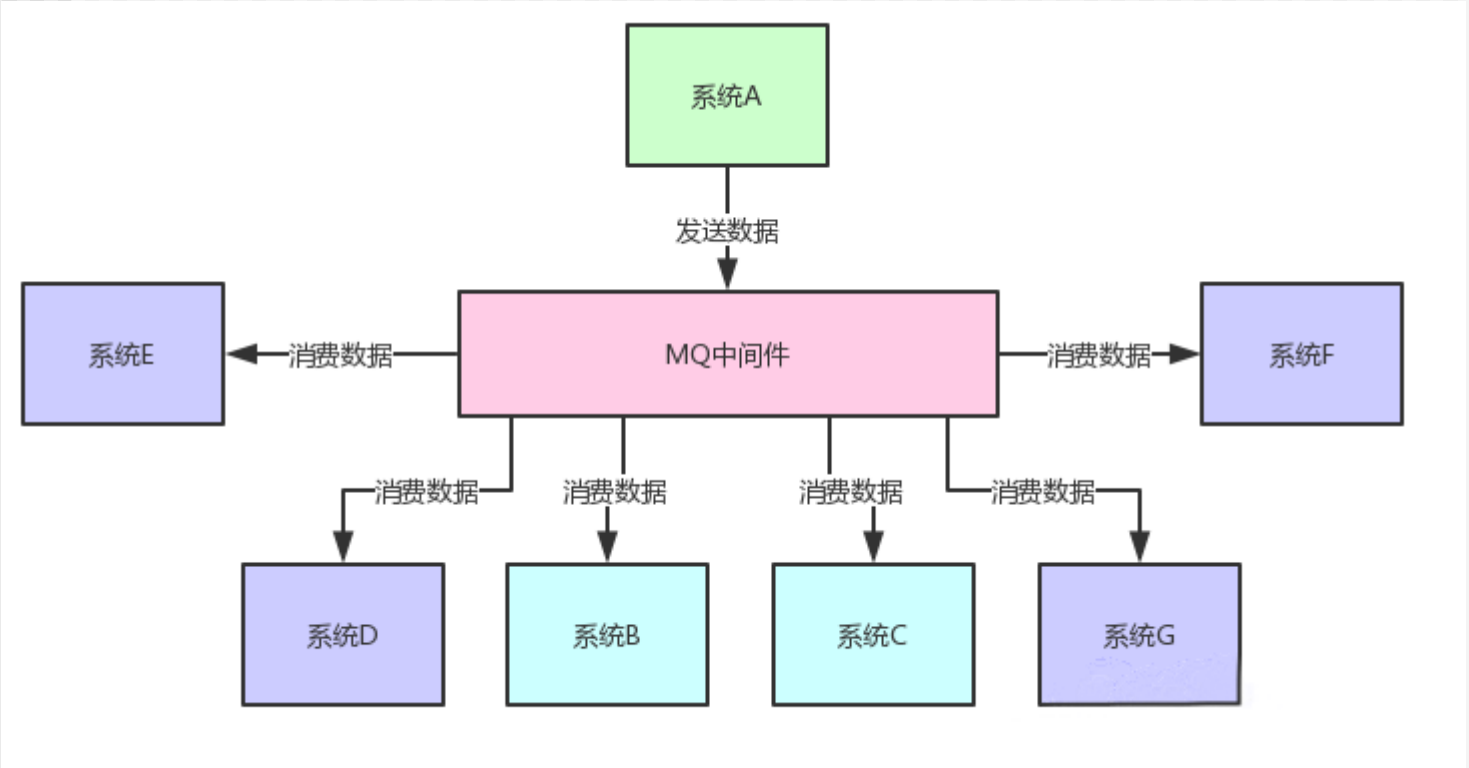
下图为传统方式下系统B、C与A之间的通信连接，随着新的D、E、F、G系统的增加，必然导致巨大的维护成本（加入或退出均需要维护A）。





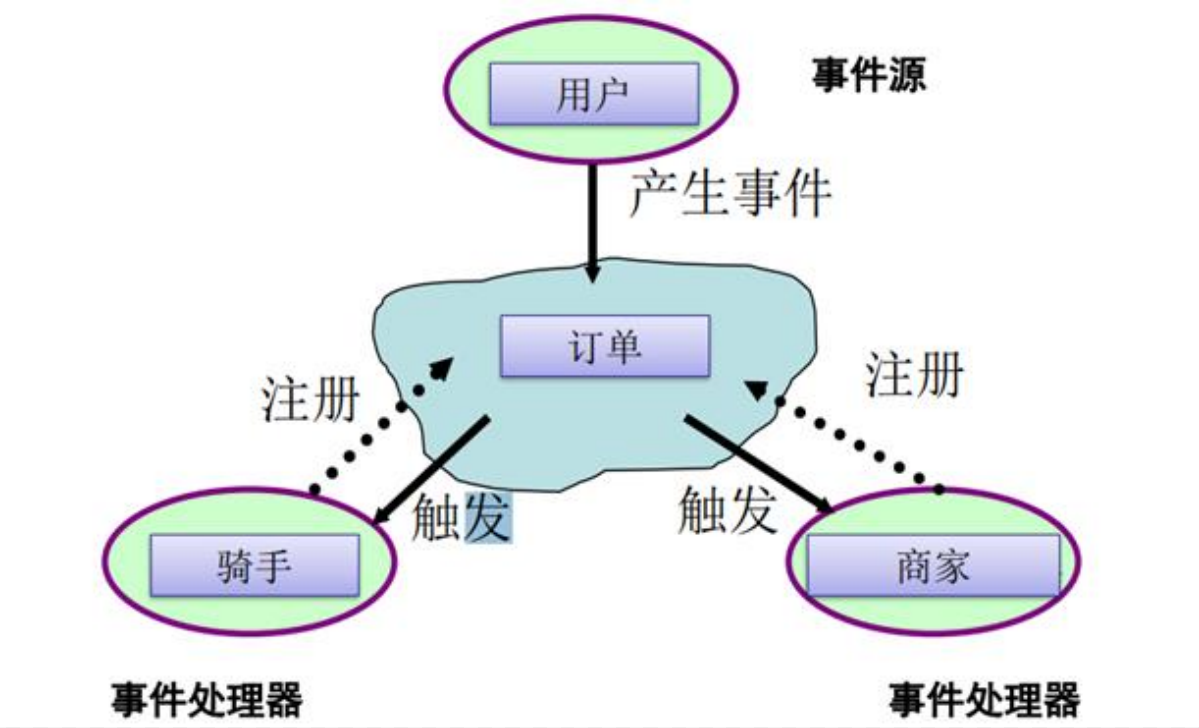
9.2.5 消息总线结构

- 优点：扩展简单。系统无须知道其他子系统接口名、位置的情况下，就可以激活其它子系统。
- 消息中间件产品：ActiveMQ、RabbitMQ



9.2.5 消息总线结构

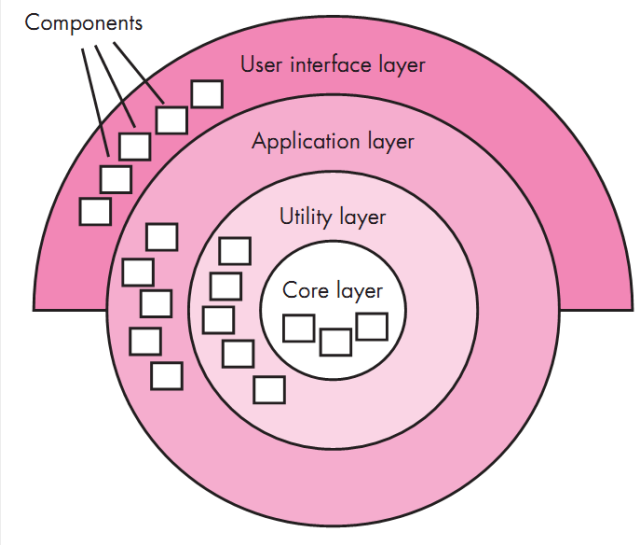
- 应用案例：美团平台



系统中用户下单构件、骑手接单构件、商家接单构件之间的耦合度大大降低。

9.2.6 层次化结构

- 系统被组织成若干个层次，每层都由一系列构件组成。
- 同层内构件可相互调用。
- 层之间存在调用接口，但只限于相邻层，下层可为上层服务，并作为再下一层客户。



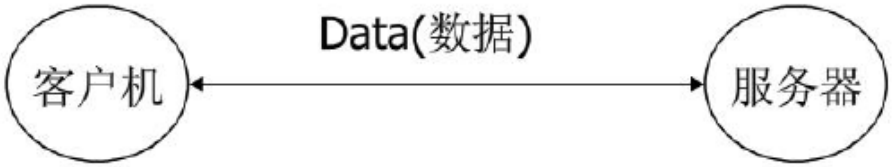
分层体系结构

9.2.7 客户-服务器结构

- 源于计算任务和计算资源的能力差异。
- 构件分布在两台或多台计算环境中。

客户机(前端，front-end)：界面逻辑、与服务器通讯的接口；

服务器(后端：back-end)：与客户机通讯的接口、业务逻辑、数据管理。



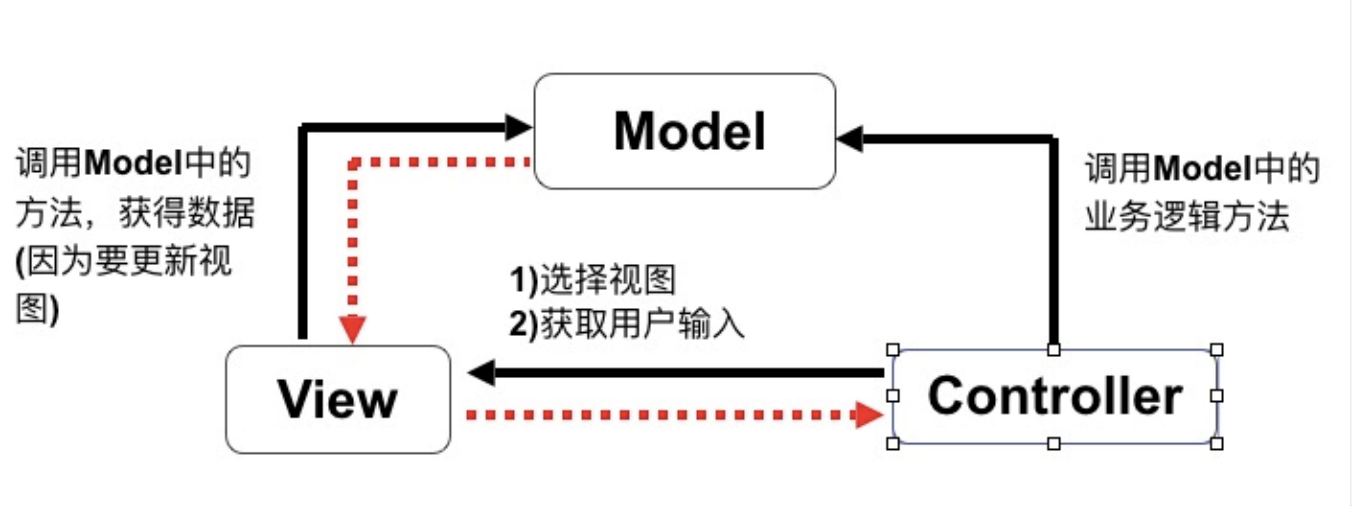
- 如银行ATM系统，远程管理调度系统
- 演变：三层，多层C/S，B/S体系结构

9.2.8 MVC

- Model-View-Controller
  - 常用于Web应用系统。
  - 将系统分解为应用逻辑、用户界面、控制逻辑等不同构件类型。
  - 优势：

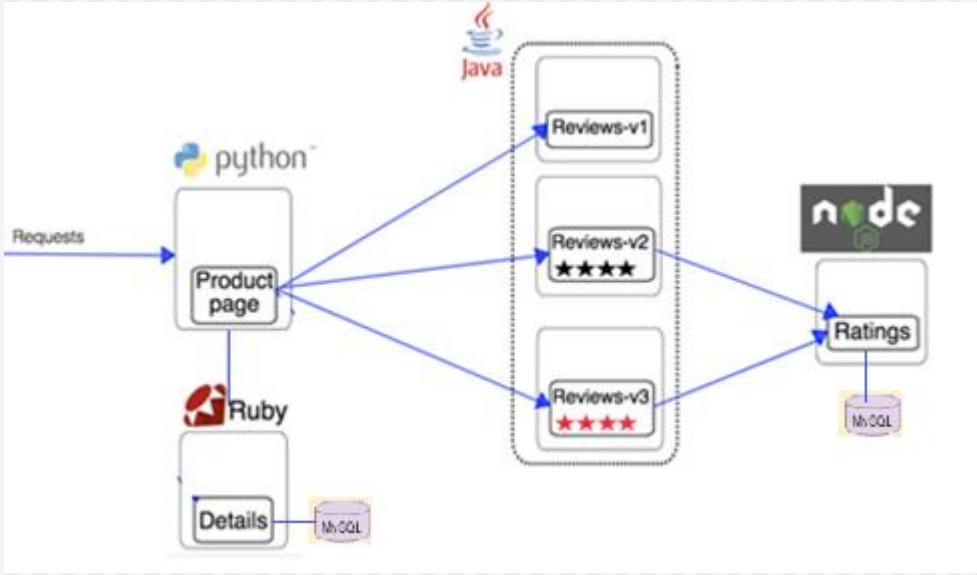
    任何一种构件的改变都不会对其它构件造成影响。

    一个模型能为多个视图提供数据。



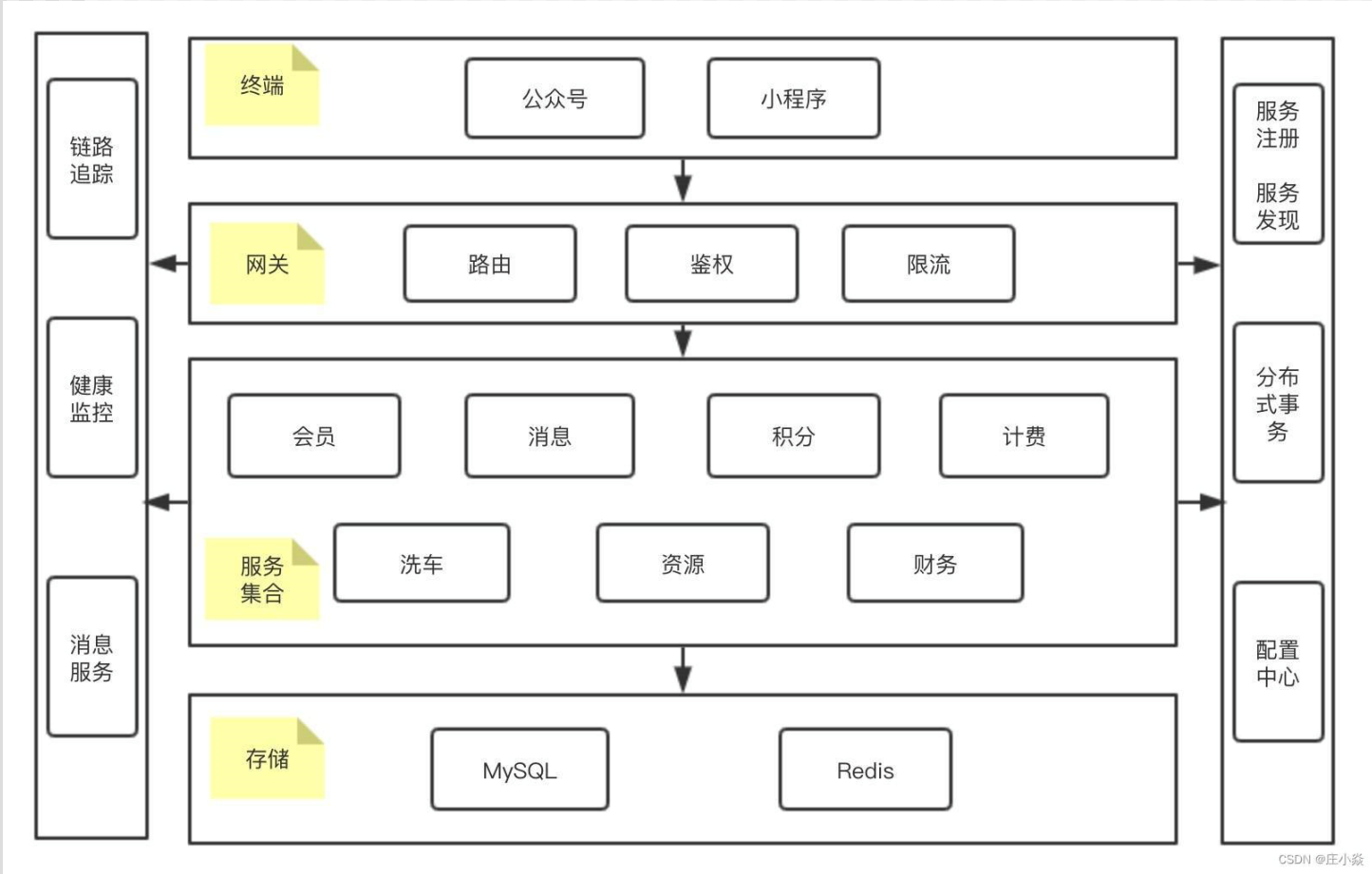
9.2.9 微服务架构

- 单体架构：应用打包(如war)部署为一个独立的单元，通过一个进程方式运行。
- 微服务架构：
  - 应用(功能)被分割成更小的、独立的服务，各个微服务之间的关联通过RPC(同步调用)或MQ(异步调用)实现。
  - 独立的微服务不需要部署在同一个虚拟机或应用服务器中。



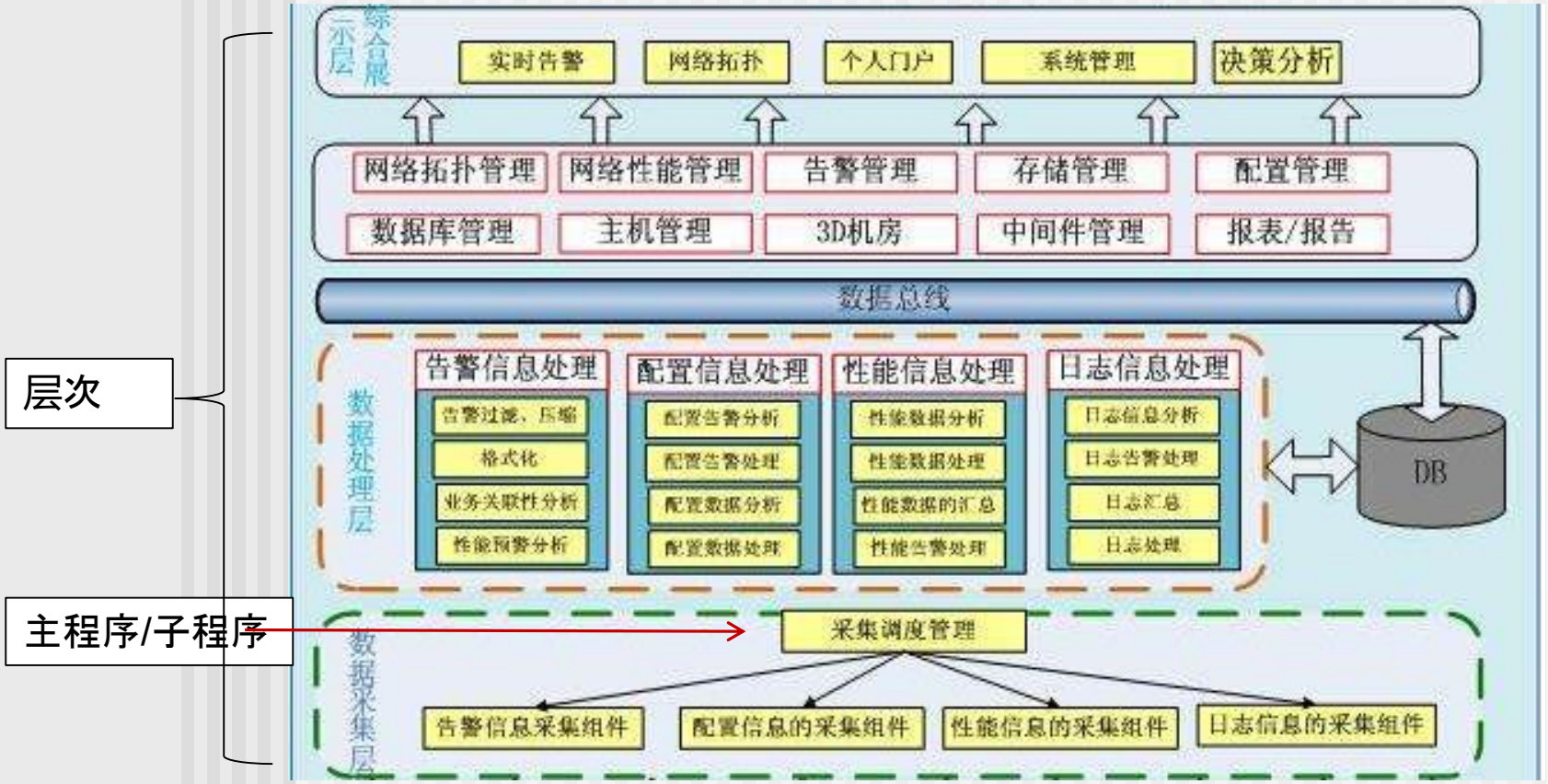
9.2.9 微服务架构

- 微服务架构示例



9.2.10 混合风格

- 大规模系统往往包括多种体系结构风格



9.3 体系结构描述

- 关注点：为谁描述哪些内容？用什么符号体系或描述语言？
- 程序员说：体系结构就是要决定需要编写哪些类、使用哪些现成框架。程序经理笑了；
- 程序经理说：体系结构就是模块的划分和接口的定义。数据库工程师笑了；
- 数据库工程师说：体系结构规定了持久化数据的结构，其他不过是对数据的操作而已。用户笑了；
- 用户说：体系结构就是决定一个个功能子系统如何划分。程序员又笑了；

这些描述我们都需要。每个视角代表一类利益相关者，反应了系统结构的一个侧面。

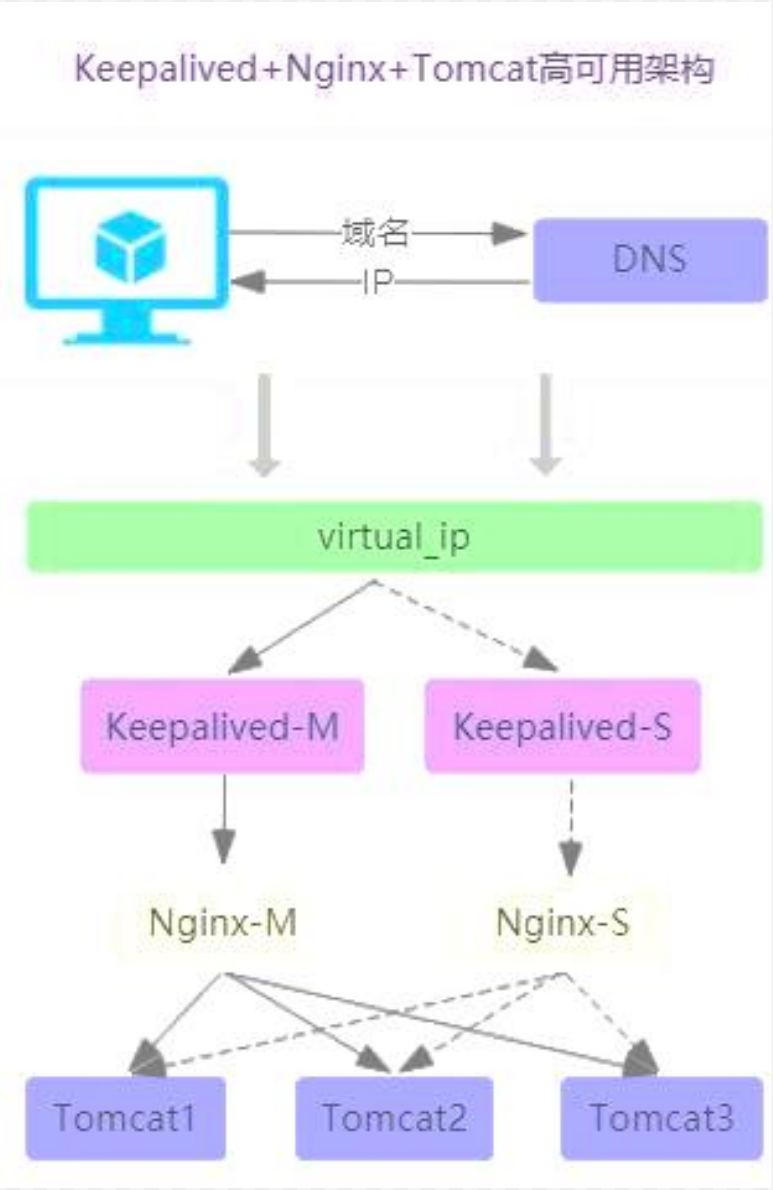
9.3 体系结构描述

- 系统体系结构多从逻辑架构、物理架构和开发架构等视角描述。
- 逻辑架构关注的是功能构件，包含用户可见的功能构件，以及不可见的功能构件。
- 物理架构关注网络、服务器等基础设施及其拓扑结构。
- 开发架构更关注程序包，除自己写的程序外，还包括应用程序依赖的SDK、第三方类库等。

9.3 体系结构描述

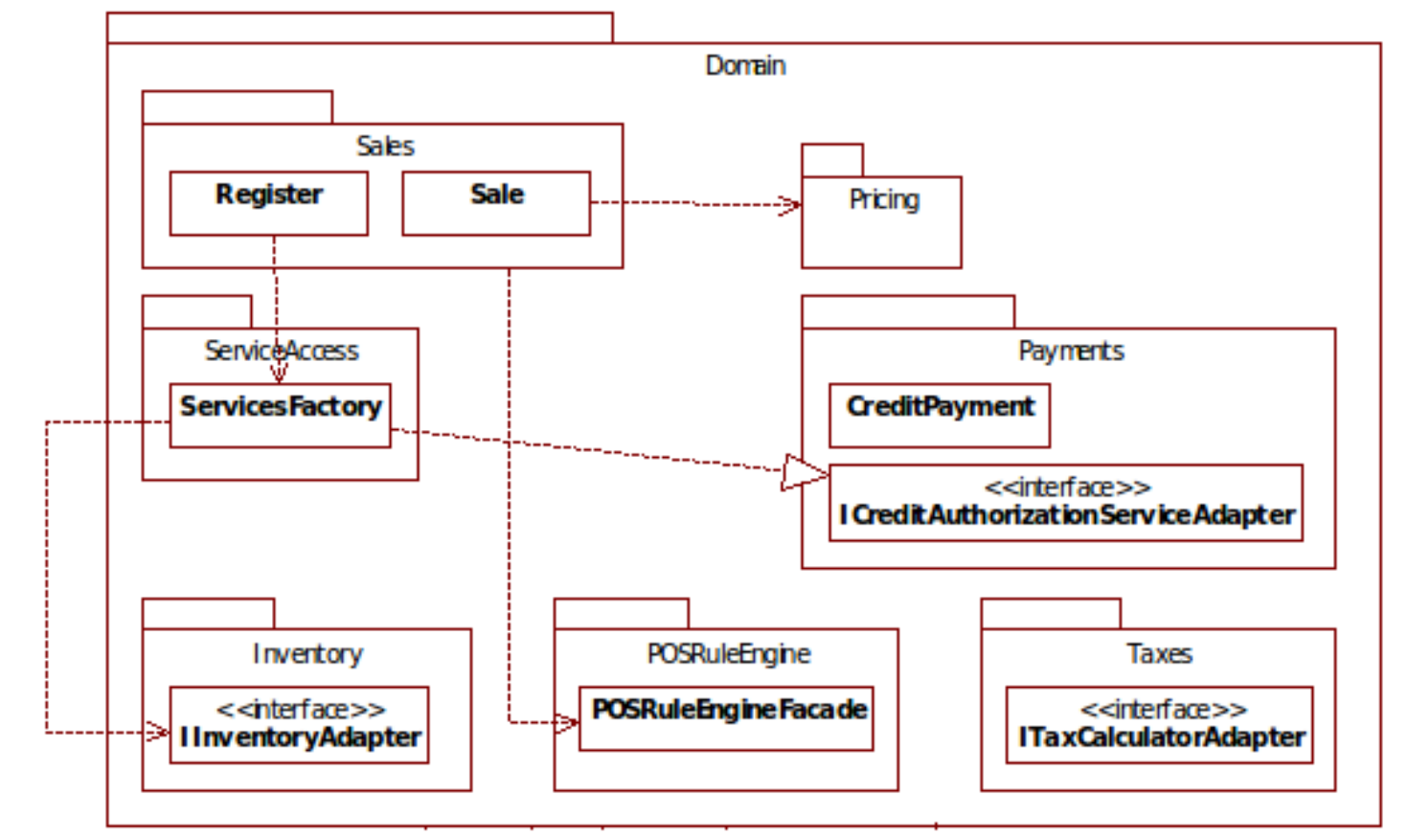
物理架构实例

- 如何保障网站能支持同时10W人在线、7*24小时提供服务?
- Nginx实现负载均衡，将请求路由到对应的Tomcat
- Keepalived用于监控系统中服务节点（Nginx）的状态。



9.3 体系结构描述

开发架构实例



9.4 体系结构设计方法

- 基本设计思路

- 包括：
  - 面向对象体系结构设计
  - 传统结构化设计

9.4.1 面向对象设计方法

- (1) 定义系统交互的外部环境和交互特性
  - 目的是识别系统的外部实体与接口,定义接口构件
- (2) 识别一组结构原型(archetypes)
  - 原型是系统中最核心的抽象类或模式，目标系统的体系结构可由这些原型组成。
  - 原型(archetypes)是系统中最稳定的元素，可以基于系统行为以不同方式对这些元素实例化。
  - 多数情况下，原型可以通过分析类来获得。
- (3) 对每个结构原型进一步细化，以完善系统结构。

9.4.1 面向对象设计方法

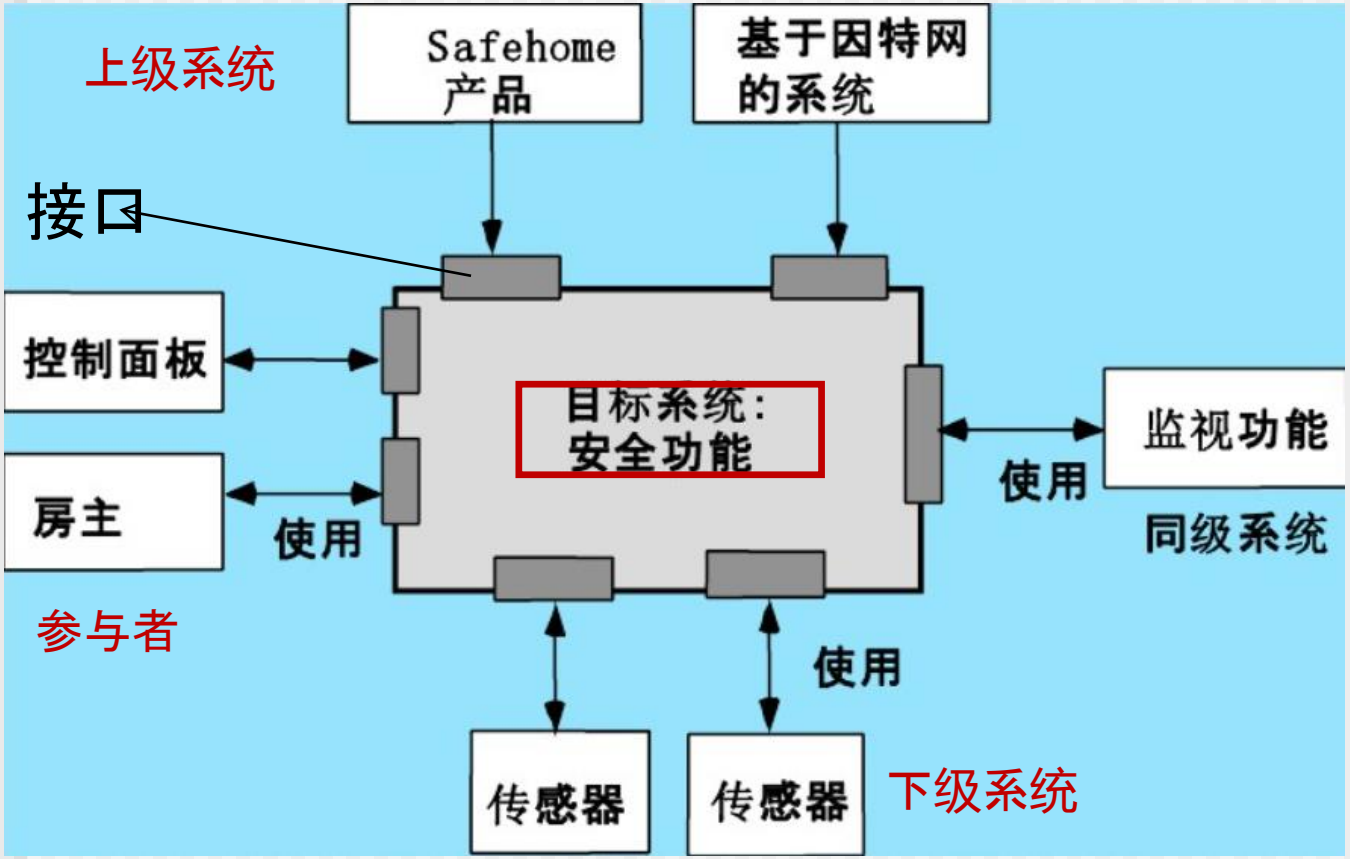
案例1：safehome系统中的“安全功能”子系统。

完整的safehome系统包括安全功能、监视功能、住宅管理功能等多个子系统。

- “安全功能”子系统的需求：　该子系统涉及哪些用例？？
  - 1）允许户主通过控制面板 解除或开启 安全功能。
  - 2）对传感器信号分析，检测是否达到报警条件。
  - 3）一旦检测到报警信号，即通过电话、告警器等方式报警。

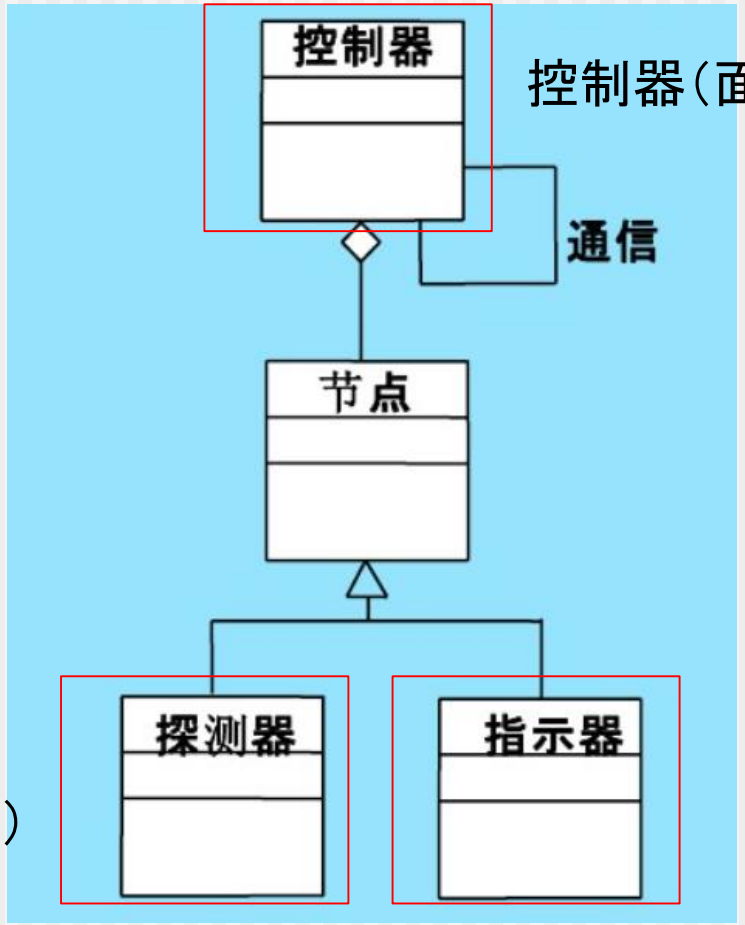
9.4.1 面向对象设计方法

(1)识别系统外部环境



9.4.1 面向对象设计方法

(2) 定义原型



控制器(面板)

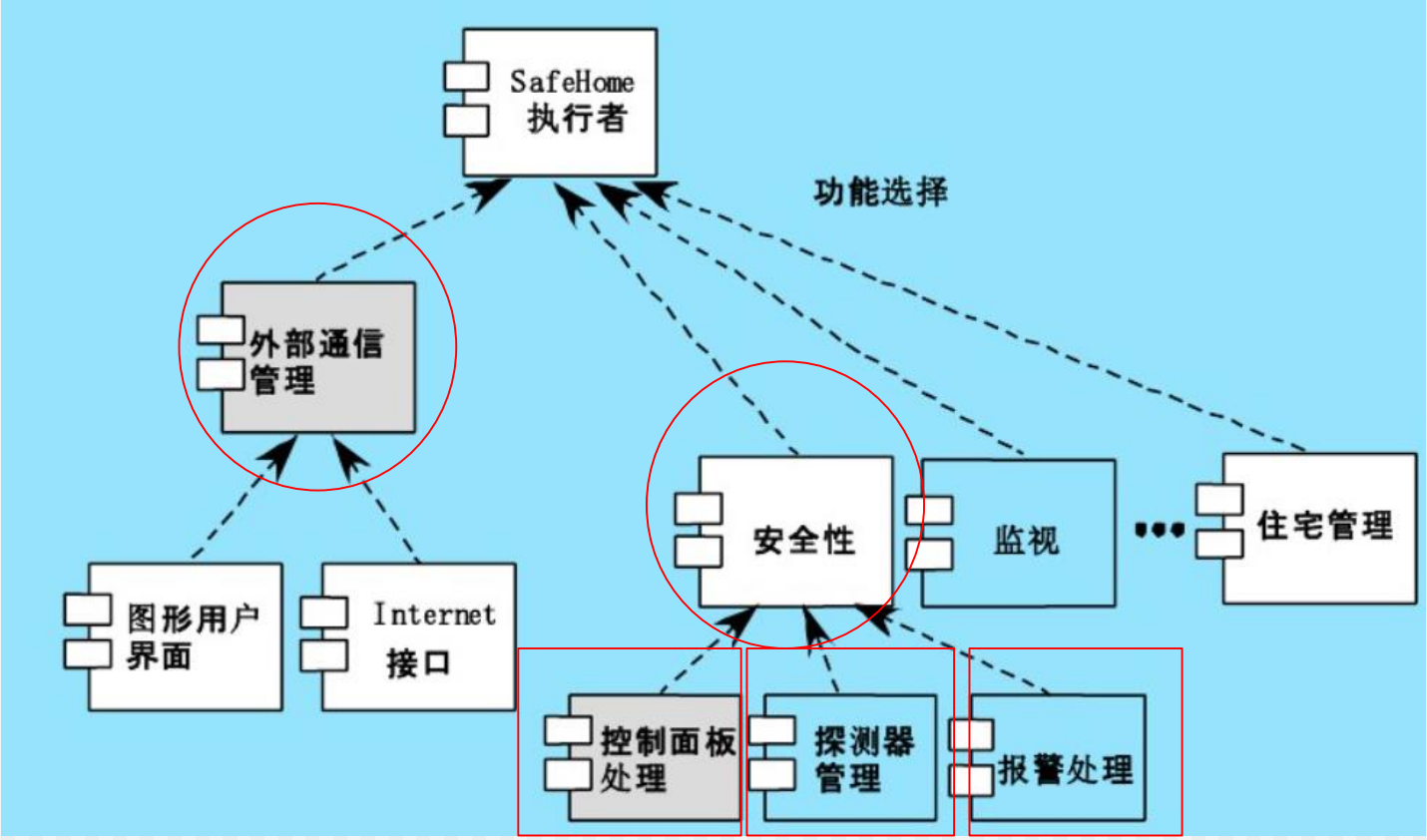
探测器（是对多种传感装置的抽象）

指示器 （是对实际报警装置如响铃、闪灯等的抽象）

9.4.1 面向对象设计方法

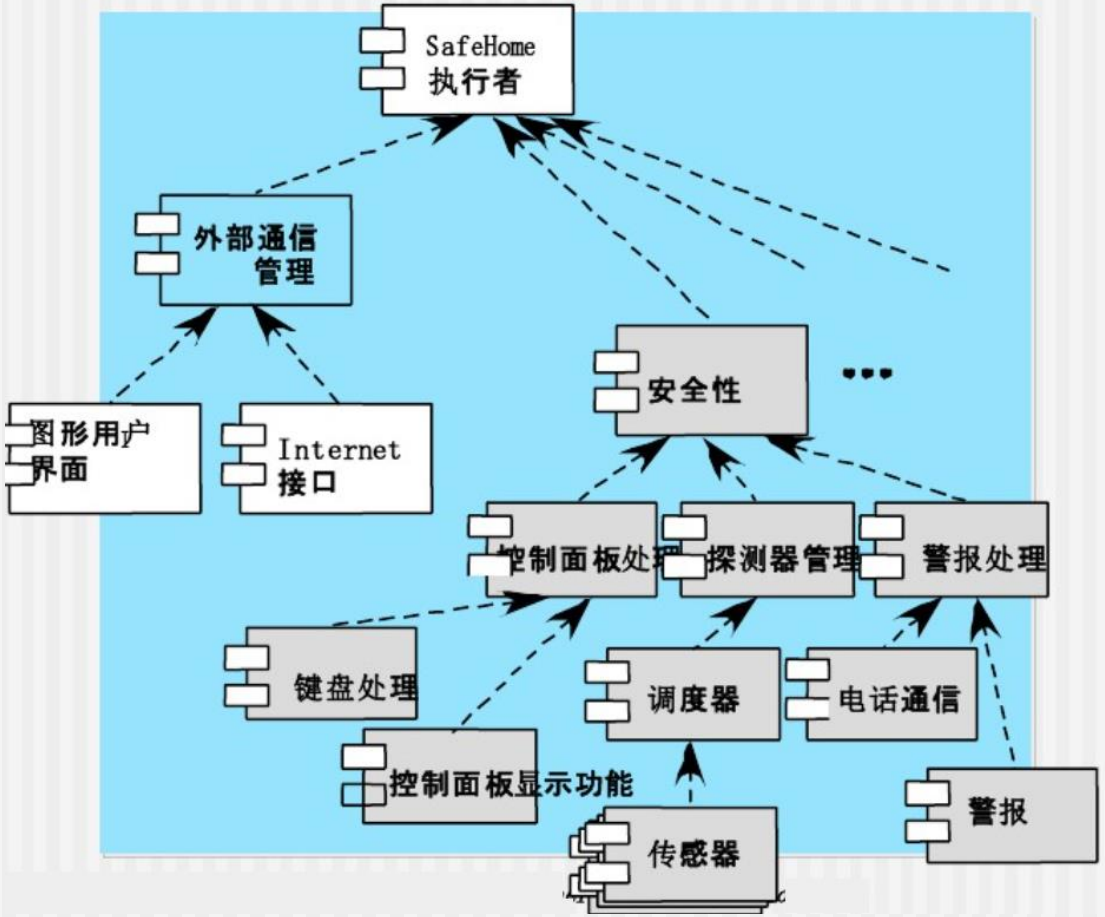
(3) 设计顶层系统结构

包括外部通信管理、控制面板处理、探测器管理、报警处理



9.4.1 面向对象设计方法

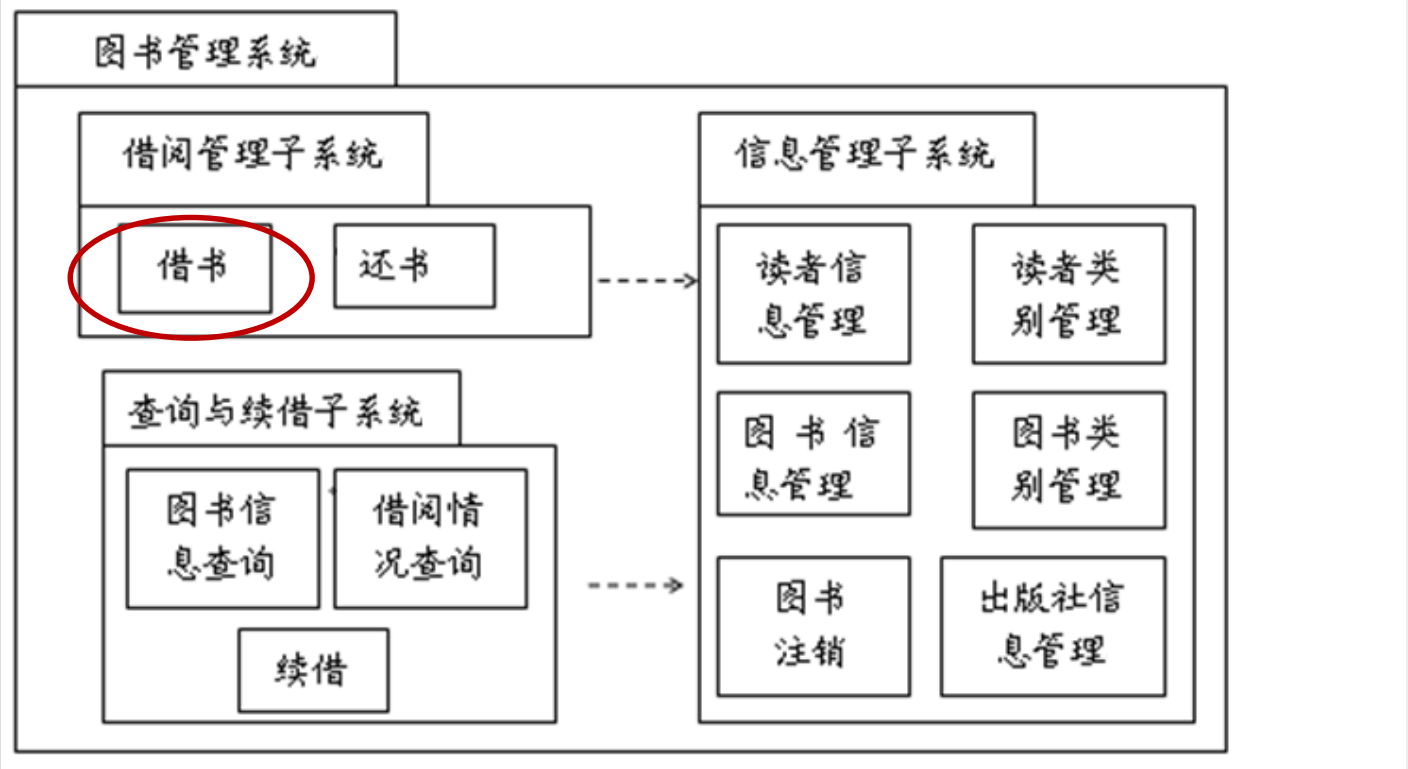
(4) 系统结构细化



9.4.2 面向对象设计案例

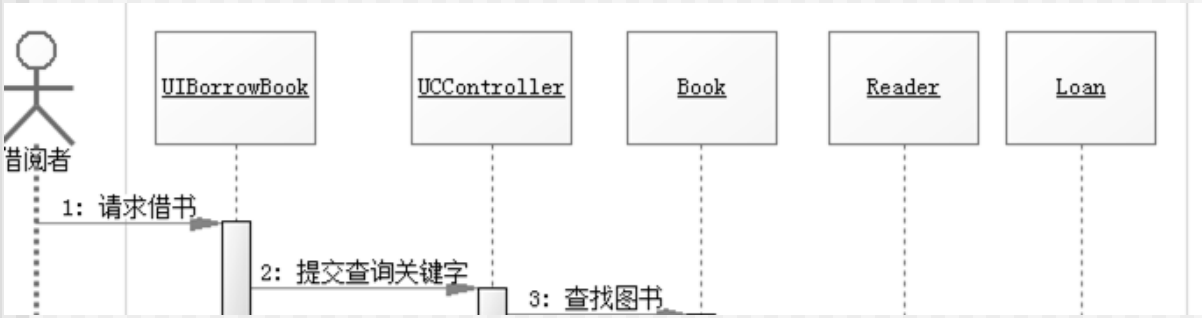
- 案例2 对图书管理系统体系结构设计

  (1) 先采用“包图”划分子系统。

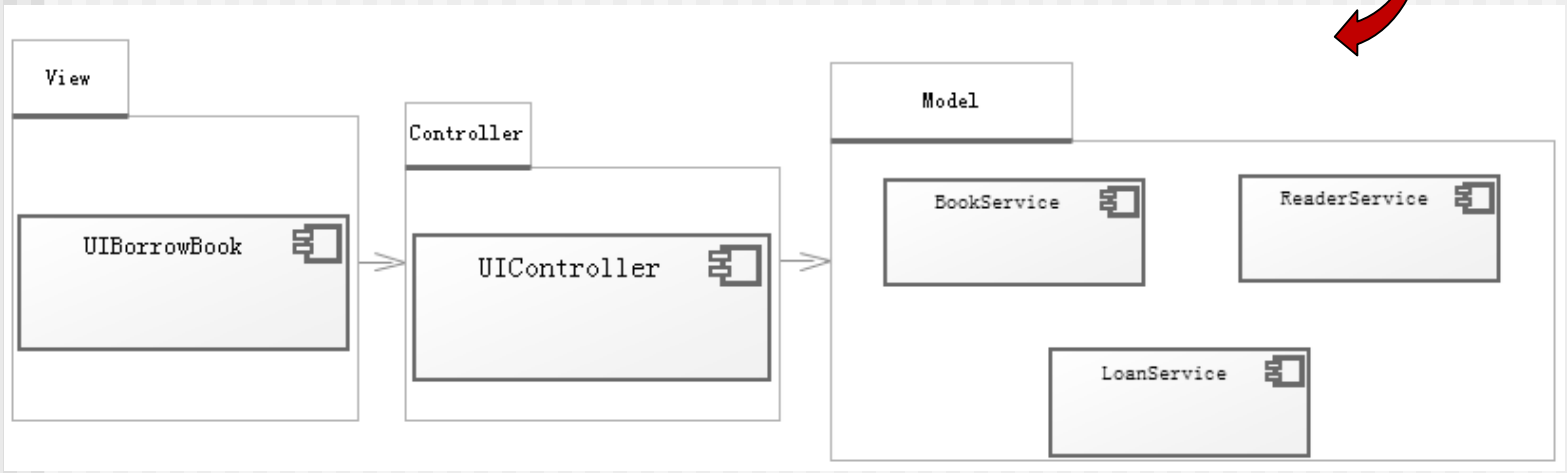


9.4.2 面向对象设计案例

- (2) 针对“借书”需求，识别结构原型



- (3) 对“借书”构件，采取MVC风格进行细化



9.4.2 面向对象设计案例

- 案例3

对大规模系统，可先按照层次结构风格（如数据采集层、数据处理层、服务层、应用层等）分解，得到多个子系统。

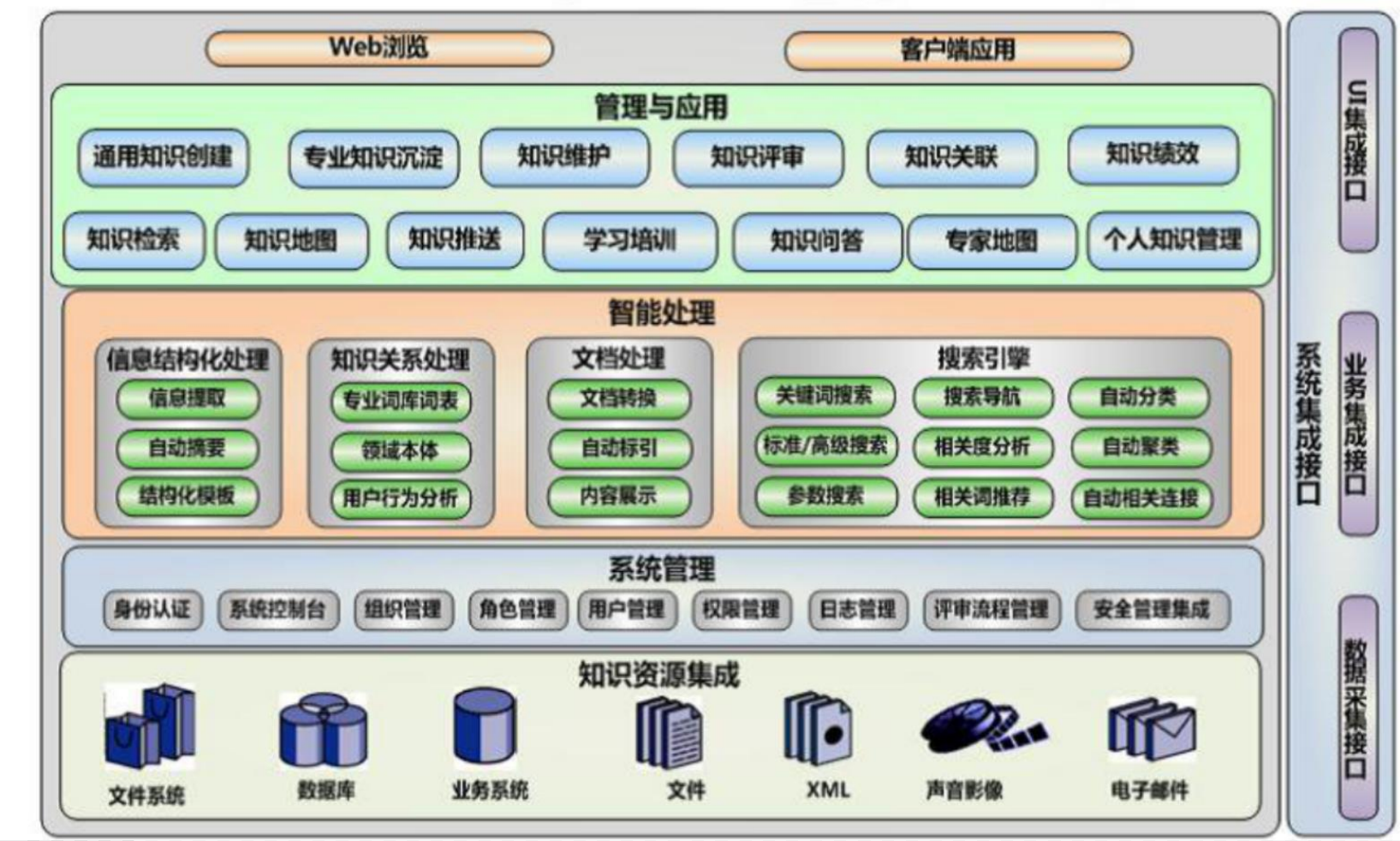
再对每个子系统中的构件，选择合适的风格，进一步细化为子构件。

记住：

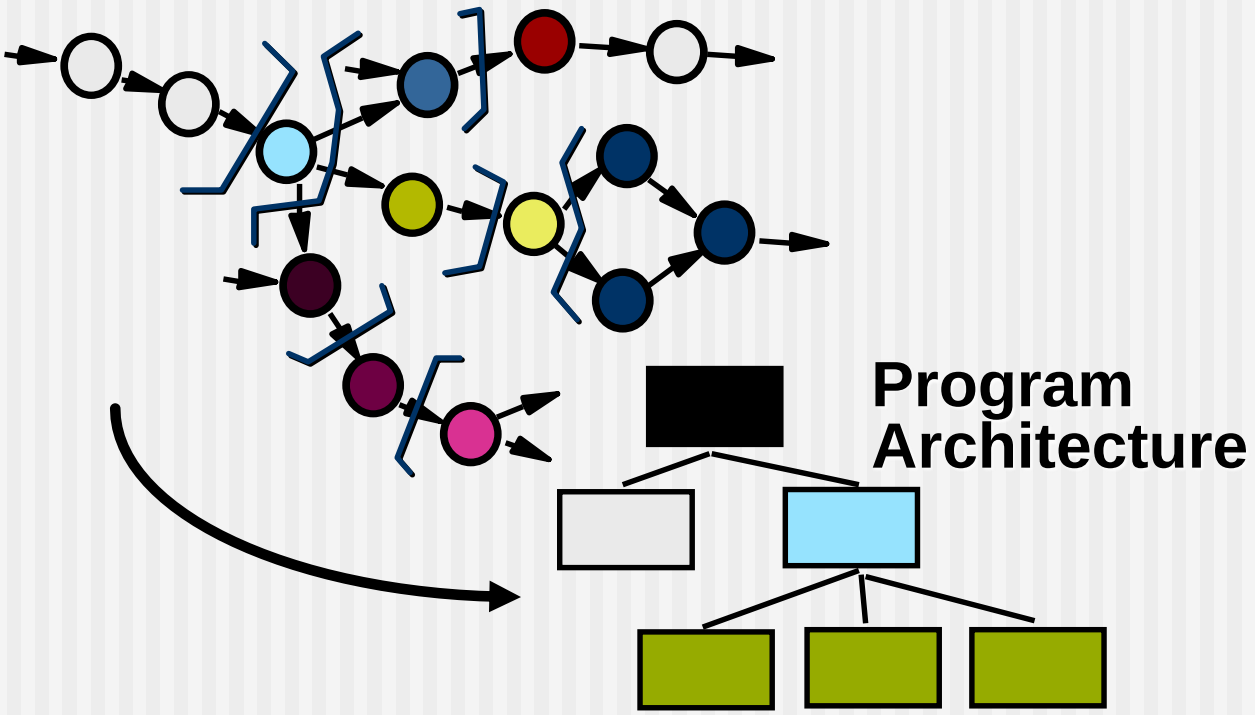
(1) 行为可以变，但行为的提供者往往是不变的，利用这一点发现系统结构原型（关键构件）。

(2) 复用已有体系结构风格，适配构件之间的联系。

9.4.2 面向对象设计案例



9.4.3 结构化设计方法



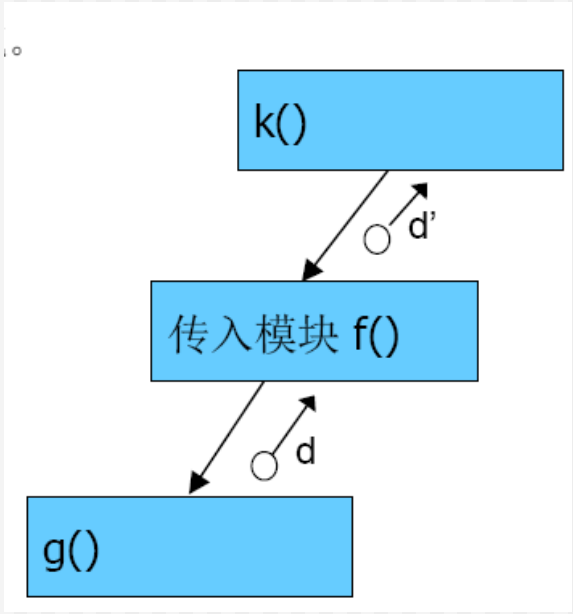
目标:根据DFD的不同类型，通过对系统中模块的合理划分，得到软件体系结构图

9.4.3 结构化设计方法

- 体系结构风格：主程序/子程序风格
- 将系统分割为多个功能模块
- 确定各个模块之间的调用关系
- 用系统结构图表示设计模型

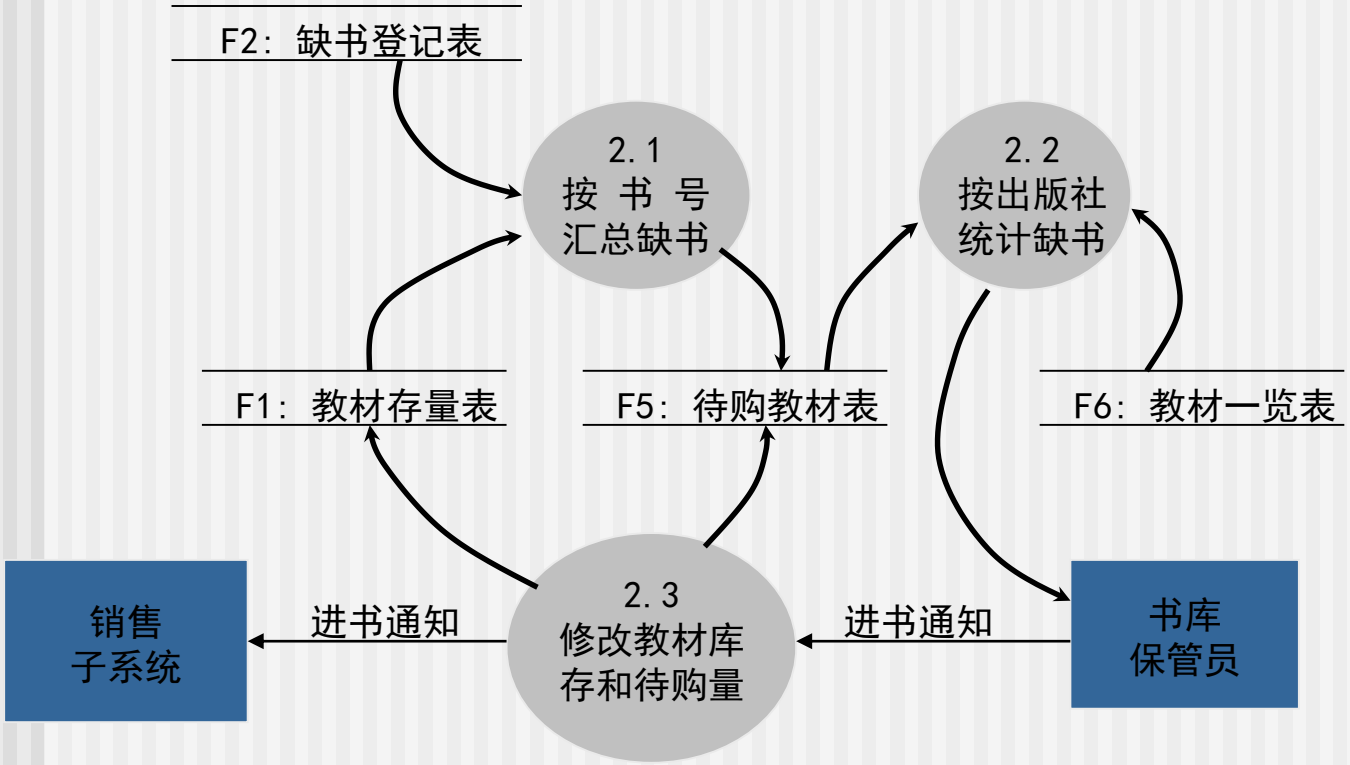
```
void k () {
   ...
   int d' = f();
   ...
}

int f() {
   ...
   int d = g();
   int d' = HelloWorld (d);
   return d'
}
```



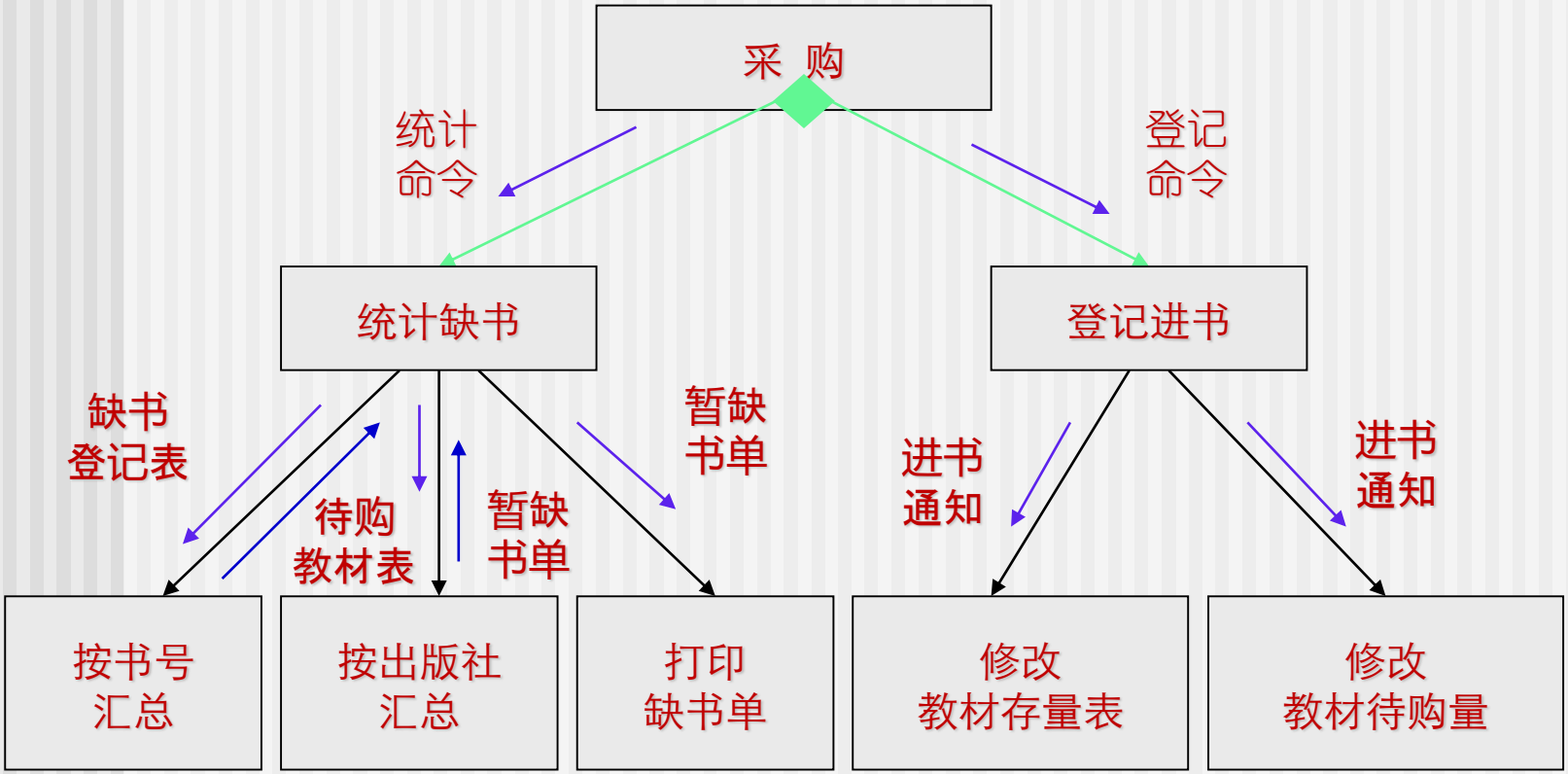
9.4.3 结构化设计方法

实例：采购子系统DFD图



9.4.3 结构化设计方法

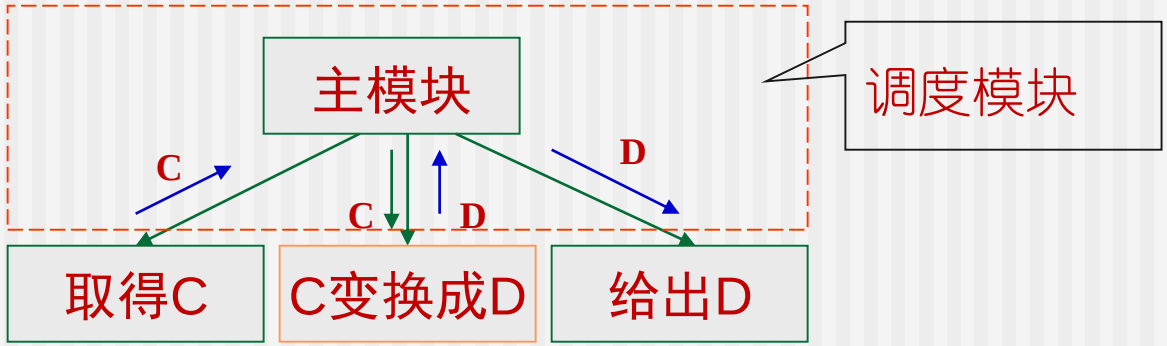
实例：采购子系统结构图



9.4.3 结构化设计方法

变换分析

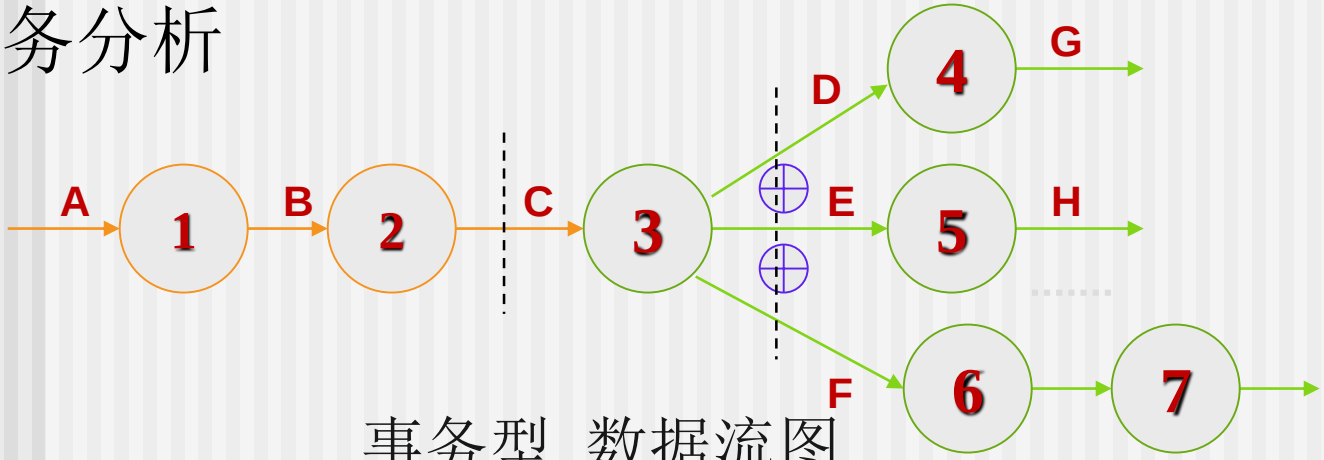
变换型 数据流图



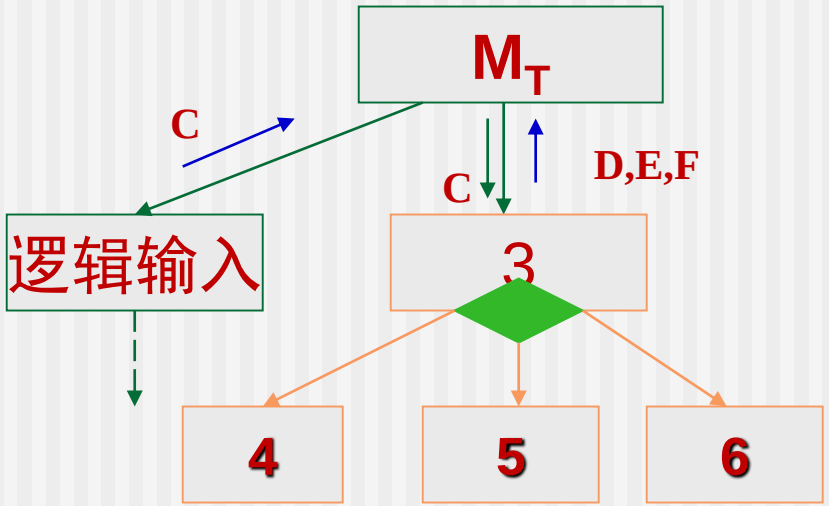
变换映射

9.4.3 结构化设计方法

事务分析



事务型 数据流图



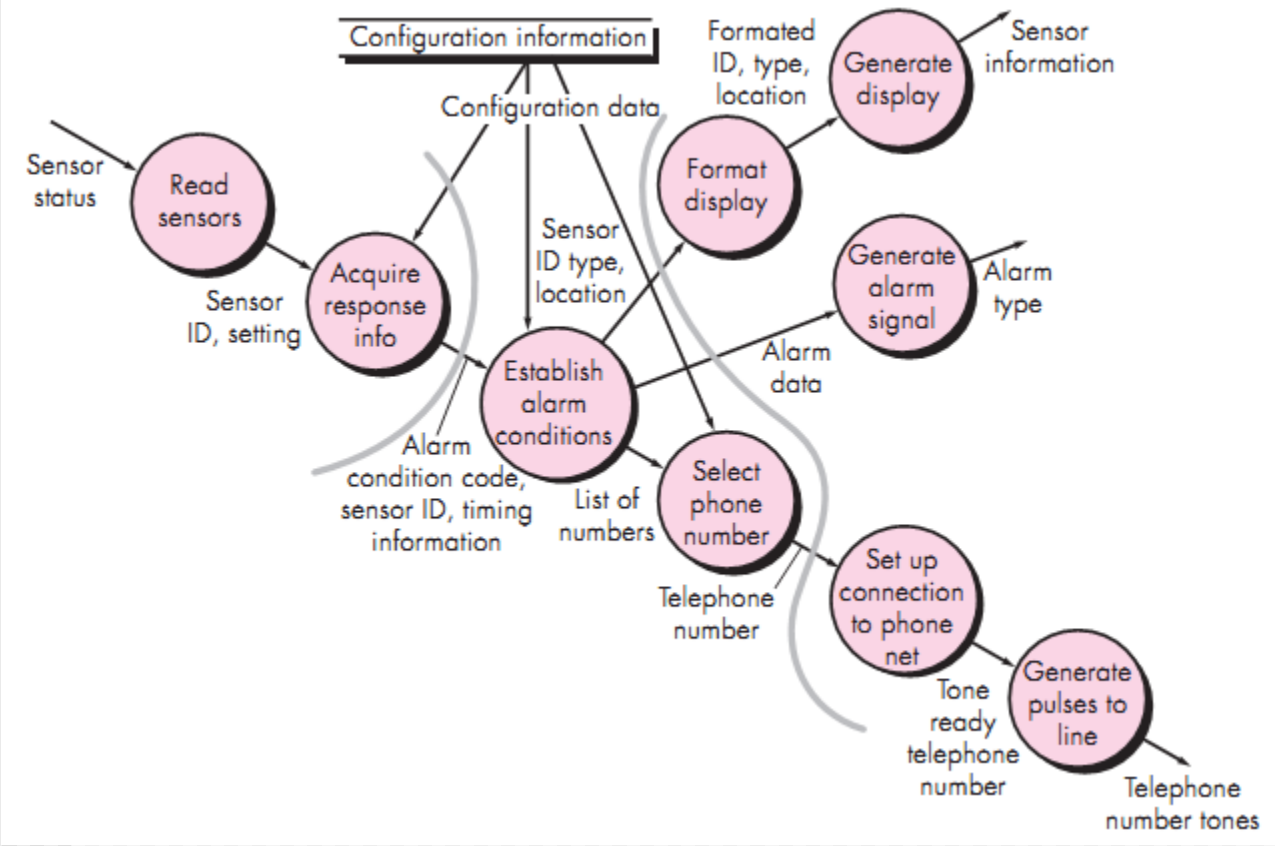
事务映射

9.4.3 结构化设计方法

- 结构化设计基本步骤
  (1) 分析确定数据流图类型。
  (2) 标识输入、输出流边界。
  (3) 依据变化或事务分析规则，将DFD图映射为顶层和第一层模块结构。
  (4) 对第一层中的模块继续细化和分解。
  (5) 利用启发式规则优化初始系统结构图。

9.4.3.1 变换分析实例

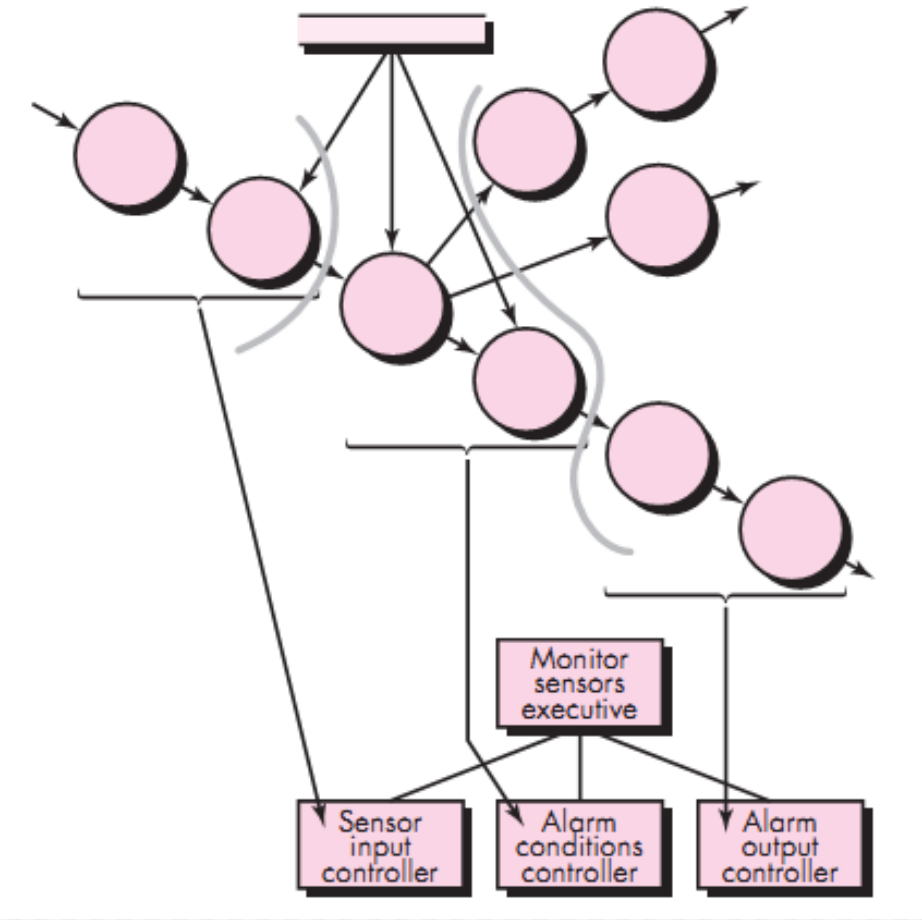
Step 1: 标识流边界



监控传感器DFD

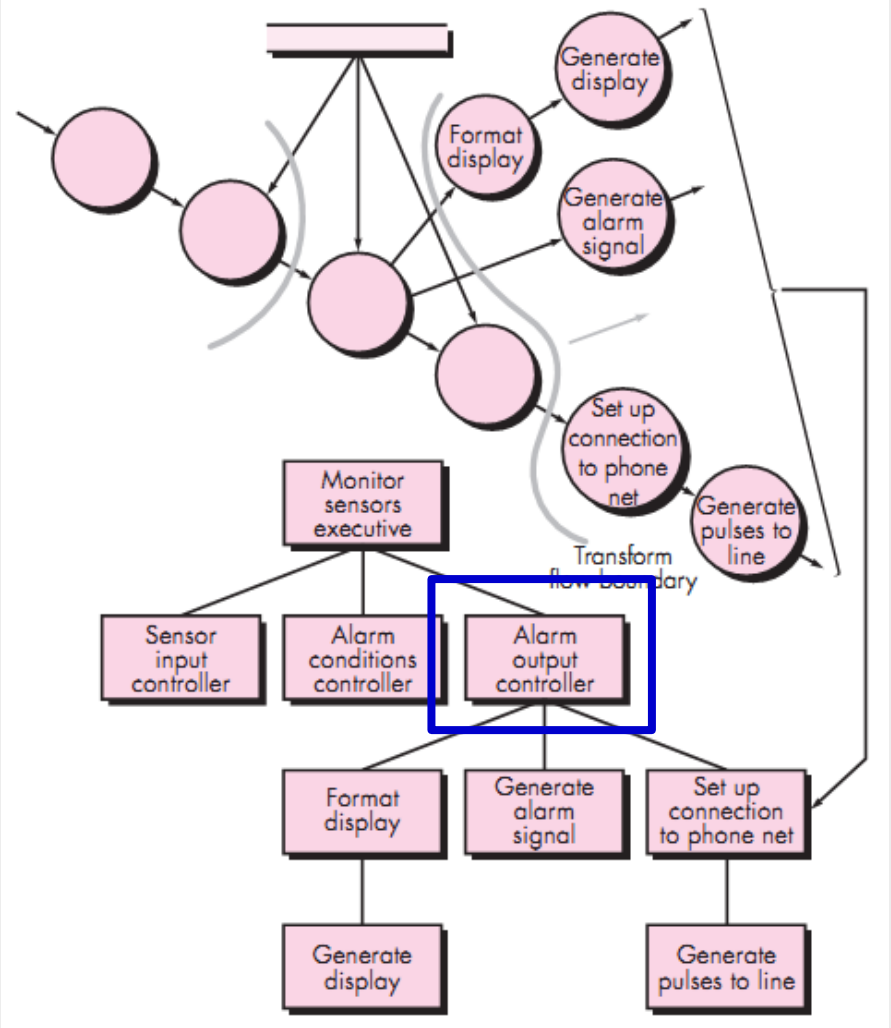
9.4.3.1 变换分析实例

Step 2: 第一级分解

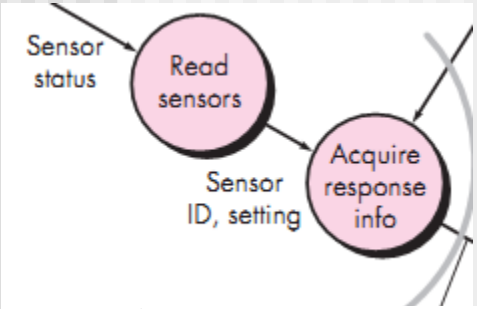


9.4.3.1 变换分析实例

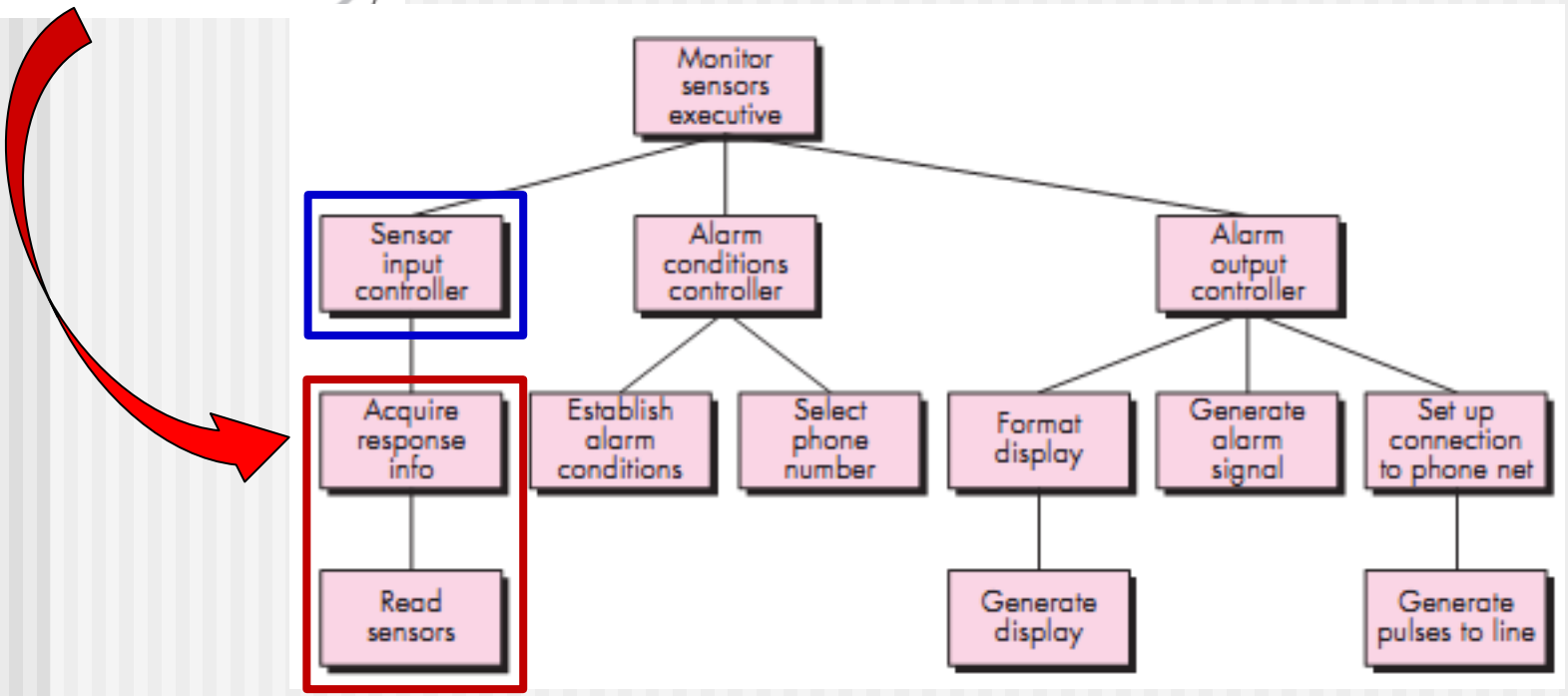
Step 3: 对输出模块第二级分解



9.4.3.1 变换分析实例

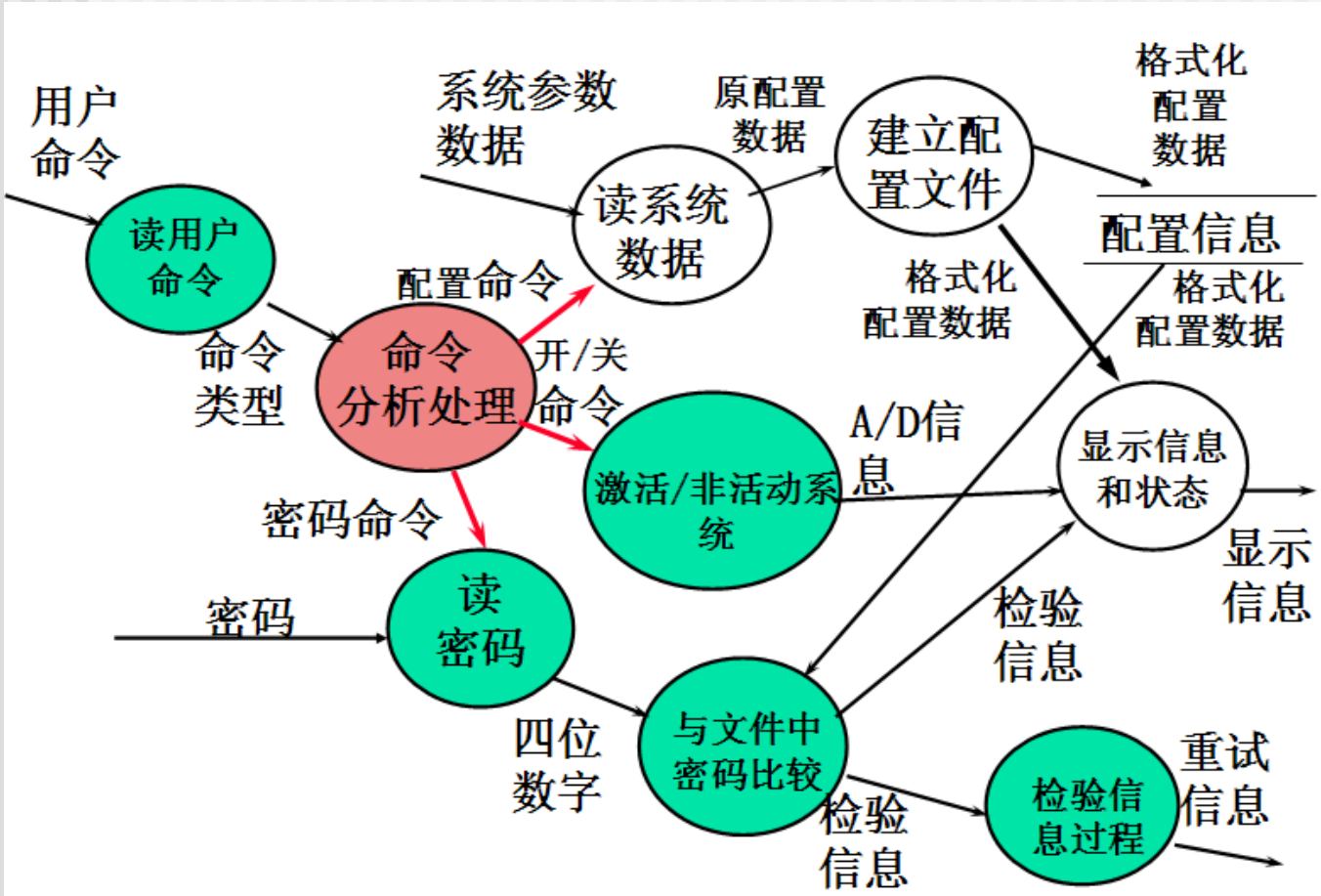


Step 3: 对输入、变化中心第二级分解



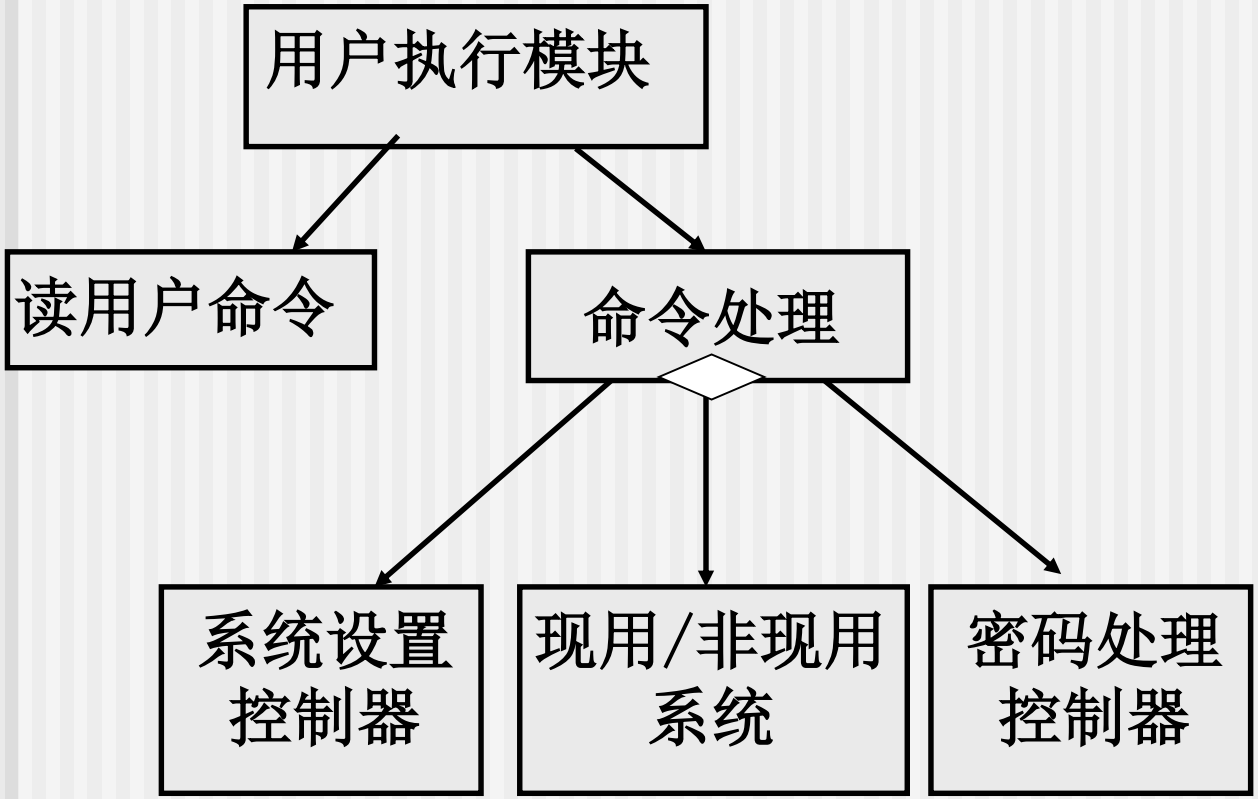
9.4.3.2 事务分析实例

假定某工控系统DFD图如下



9.4.3.2 事务分析实例

- Step1:创建顶层控制结构



9.4.3.2 事务分析实例

- Step 2:对输出分支进行细化