第八章　面向对象的系统设计 II

曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计
http://c.pku.edu.cn/

内容提要

什么是控制驱动部分

控制驱动部分

是 OOD 模型的外围组成部分之一，由系统中全体主动类构成。这些主动类描述了整个系统中所有的主动对象，每个主动对象是系统中一个控制流的驱动者

控制流（control flow）：进程（process）和线程（thread）的总称
有多个控制流并发执行的系统称作并发系统（多任务系统）

为什么需要控制驱动部分

- 并发行为是现实中固有的
  - 外围设备与主机并发工作的系统
  - 有多个窗口进行人机交互的系统
  - 多用户系统
  - 多个子系统并发工作的系统
  - 单处理机上的多任务系统
  - 多处理机系统
- 多任务的设置
  - 描述问题域固有的并发行为
  - 表达实现所需的设计决策
- 隔离硬件、操作系统、网络的变化对整个系统的影响

由系统总体方案决定的实现条件

- 计算机硬件：性能、容量和 CPU 数目
- 操作系统：对并发和通讯的支持
- 网络方案：网络软硬件设施、网络拓扑结构、通讯速率、网络协议等
- 软件体系结构
- 编程语言：对进程和线程的描述能力
- 其它软件：如数据管理系统、界面支持系统、构件库等——对共享和并发访问的支持

软件体系结构

软件体系结构

描述了构成系统的元素、这些元素之间的相互作用、指导其组合的模式以及对这些模式的约束

几种典型的软件体系结构风格

- 管道与过滤器风格（pipe and filter style）
- 面向对象风格（object-oriented style）
- 层次风格（layered style）
- 黑板风格（blackboard style）
- 进程控制风格（process control style）
- 客户-服务器风格（client-server style）

分布式系统的体系结构风格

- 主机 + 仿真终端体系结构
- 文件共享体系结构
- 客户-服务器体系结构
  - 二层客户-服务器体系结构
  - 三层客户-服务器体系结构
  - 对等式客户-服务器体系结构
  - 瘦客户-服务器体系结构
  - 胖客户-服务器体系结构
- 浏览器-服务器体系结构

系统的并发性

进程（process）概念出现之前，并发程序设计困难重重，主要原因：

- 并发行为彼此交织，理不出头绪
- 与时间有关的错误不可重现

进程概念的提出使这个问题得到根本解决

系统的并发性

进程的全称是顺序进程（sequential process），其基本思想是把并发程序分解成一些顺序执行的进程，使得：

- 每个进程内部不再包含并发行为
  所以叫做顺序进程，其设计避免了并发问题
- 多个进程之间是并发（异步）执行的
  所以能够构成并发程序

进程与线程

由于并行计算的需要，要求人为地在顺序程序内部定义和识别可并发执行的单位，线程的概念就诞生了

线程与进程的区别：

- 进程既是处理机分配单位，也是存储空间、设备等资源的分配单位（重量级的控制流）
- 线程只是处理机分配单位（轻量级的控制流）
- 一个进程可以包含多个线程，也可以是单线程的

应用系统的并发性

从网络、硬件平台的角度看：

- 分布在不同计算机上的进程之间的并发
- 在多 CPU 的计算机上运行的进程或线程之间的并发
- 在一个 CPU 上运行的多个进程或线程之间的并发

应用系统的并发性

从应用系统的需求看：

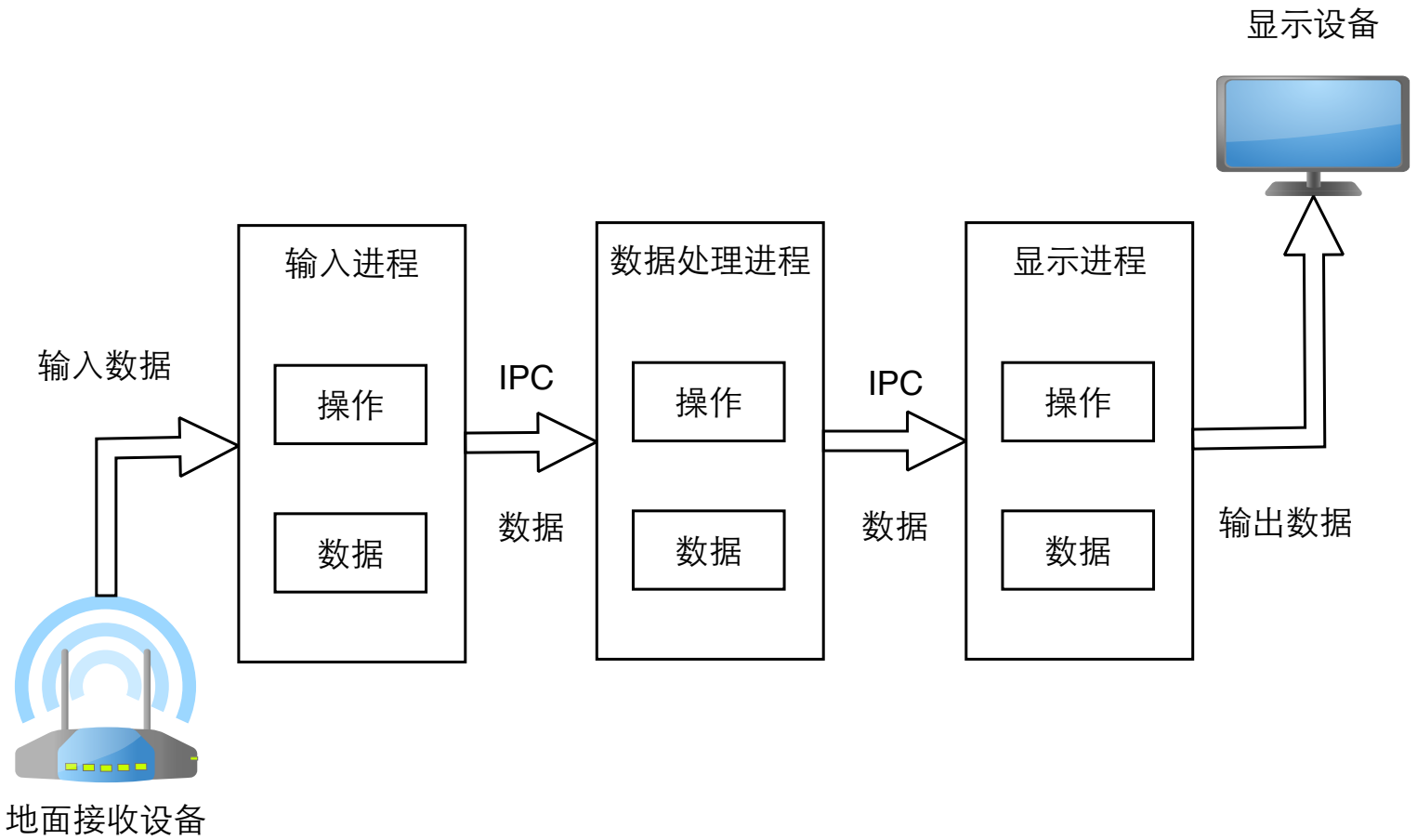
- 需要跨地域进行业务处理的系统
- 需要同时使用多台计算机或多个 CPU 进行处理的系统
- 需要同时供多个用户或操作者使用的系统
- 需要在同一时间执行多项功能的系统
- 需要与系统外部多个参与者同时进行交互的系统

处理应用系统并发的例子

问题描述:

某单位想开发一个卫星遥感信息处理系统，要求是：实时把通过地面接收设备传来的卫星遥感图片信息输入系统，经过必要的数据处理，及时将图片显示在屏幕上。

处理应用系统并发的例子



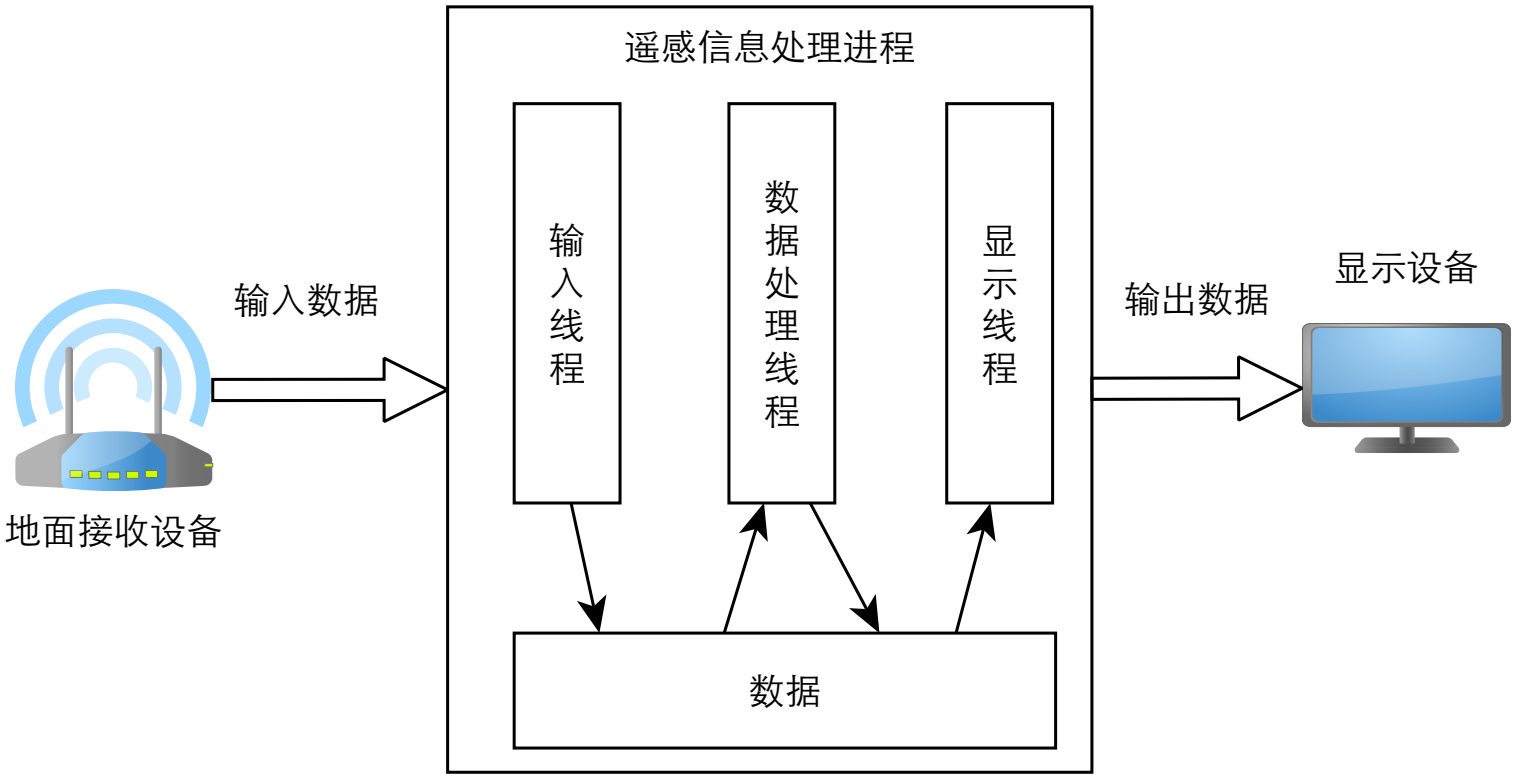
用多进程实现的遥感信息处理系统

处理应用系统并发的例子

新的需求:

针对前页10例子中多进程共享数据速度慢的问题，希望改变设计，采用多线程技术实现并发，避免控制流之间传送大量数据。

处理应用系统并发的例子



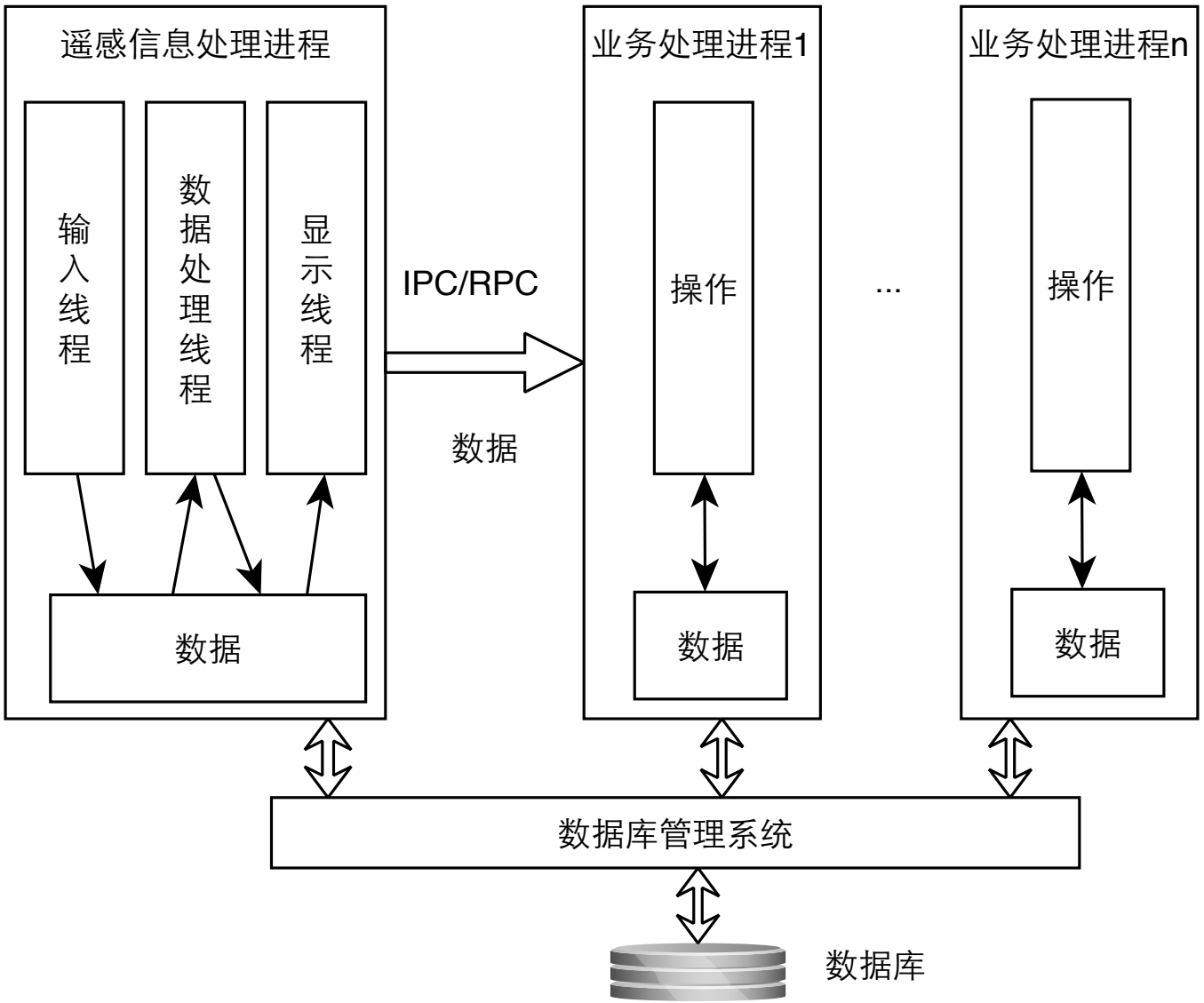
用多线程实现的遥感信息处理系统

处理应用系统并发的例子

拓展业务:

考虑面向不同应用的遥感信息处理系统，不仅需要把图片信息实时显示出来，而且需进行更多处理，如面向地理信息系统的特征信息提取等。

处理应用系统并发的例子



同时采用多线程和多进程的多应用遥感信息处理系统

讨论：进程 vs 线程

进程：重量，分布内存
线程：轻量，共享内存

- 数据访问的成本和效率？
- 创建、销毁、切换的代价？
- 健壮性？
- 易于程序员编写并发程序？

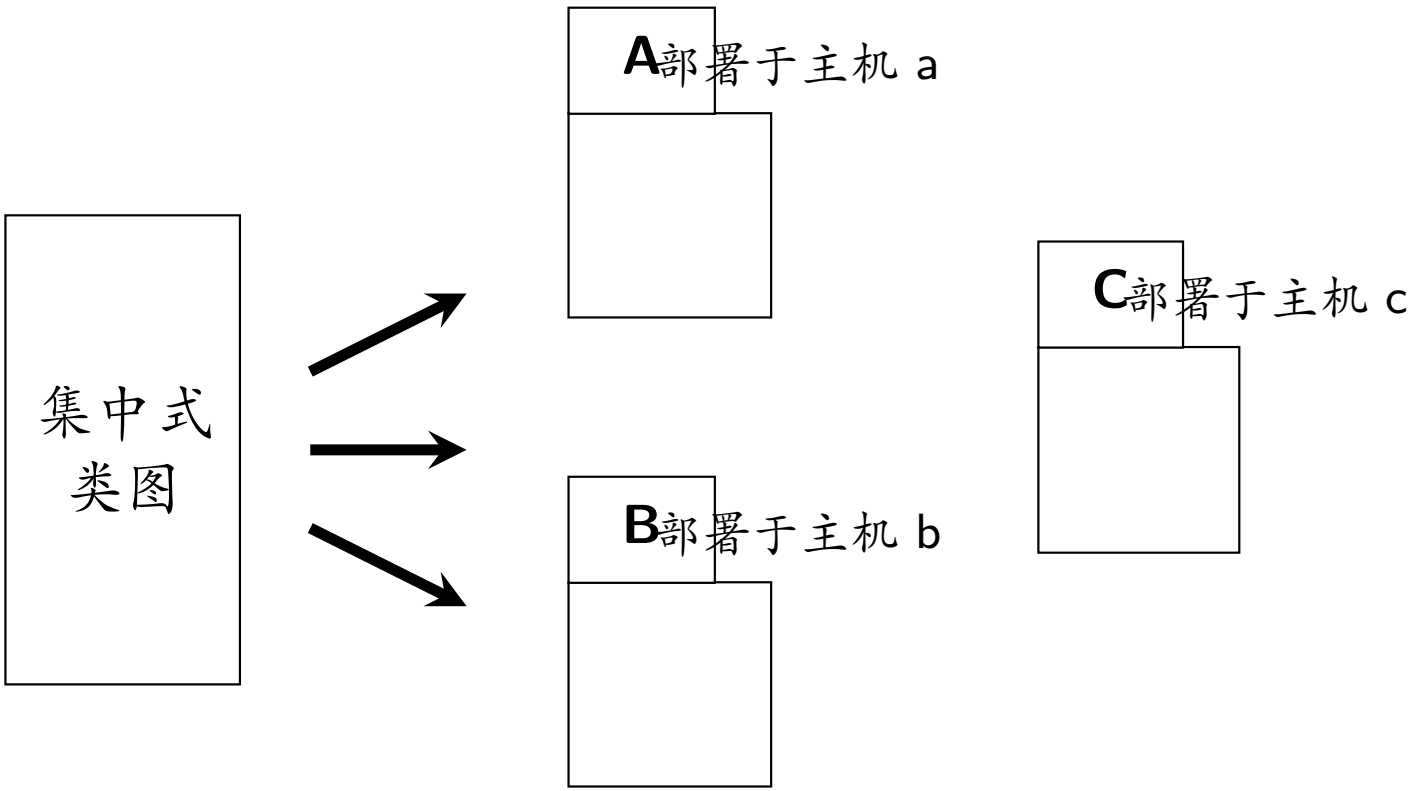
内容提要

选择软件体系结构风格

- 二层客户–服务器体系结构
  （数据）服务器–客户机
- 三层客户–服务器体系结构
  数据服务器–应用服务器–客户机

确定系统分布方案

考虑分布方案之前：暂时将系统看作集中式的
确定分布方案之后：将对象分布到各个处理机上，
以每台处理机上的类作为一个包



确定系统分布方案

系统分布包括功能分布和数据分布，在面向对象的系统中都体现于对象分布

原则：减少远程传输，便于管理

决定对象分布：

- 软件体系结构
- 系统功能在哪些结点提供
- 数据在哪些结点长期存储管理，在哪些结点临时使用
- 参照用况，把合作紧密的对象尽可能分布在同一结点
- 追踪消息，把一个控制流经历的对象分布在同一结点

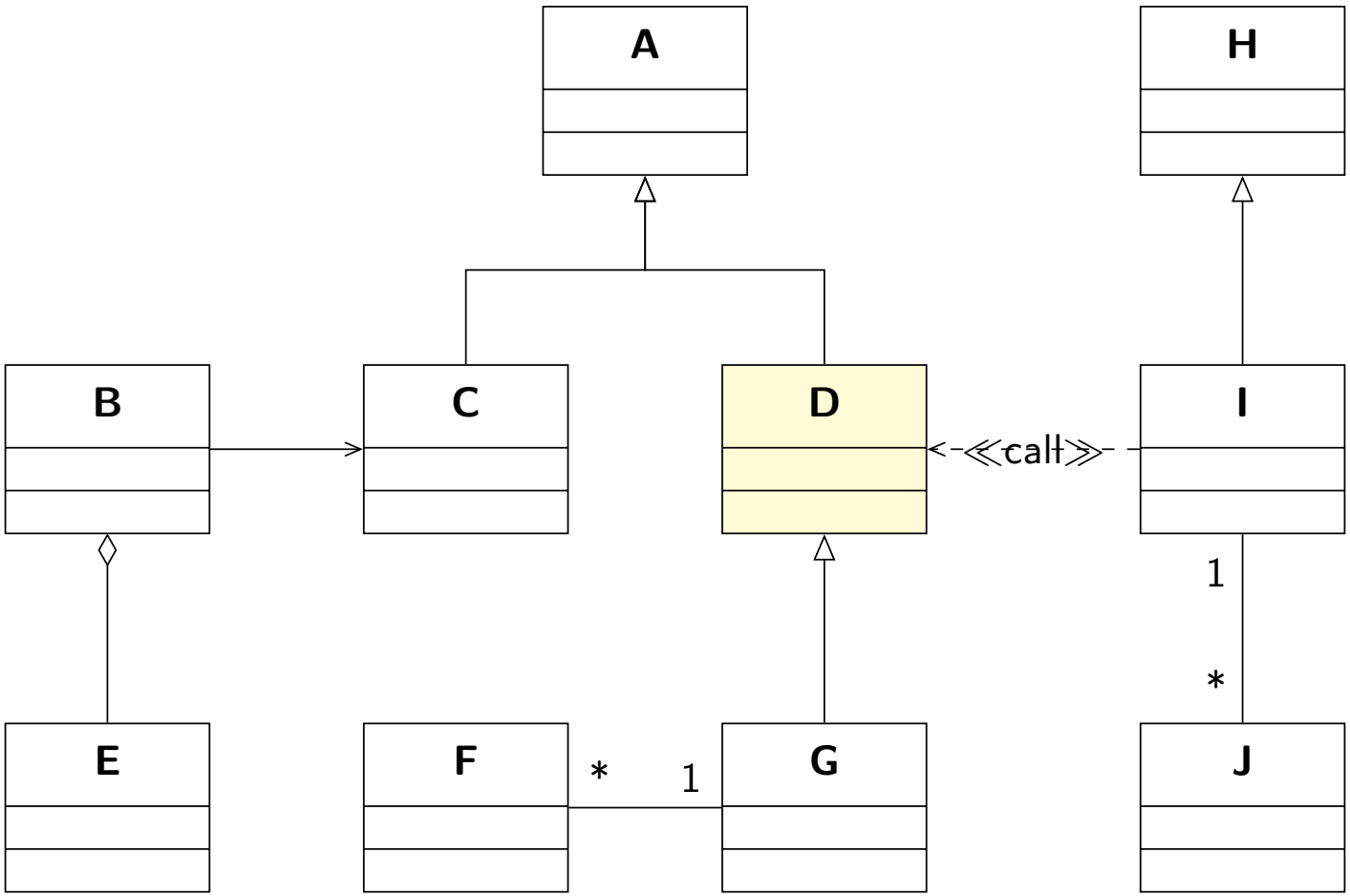
确定系统分布方案

类的分布：根据对象分布的需要
分布在每个结点上的对象，都需要相应的类来创建

策略 1　如果一个类只需要在一个结点上创建对象实例：
把这个类分布在该结点上

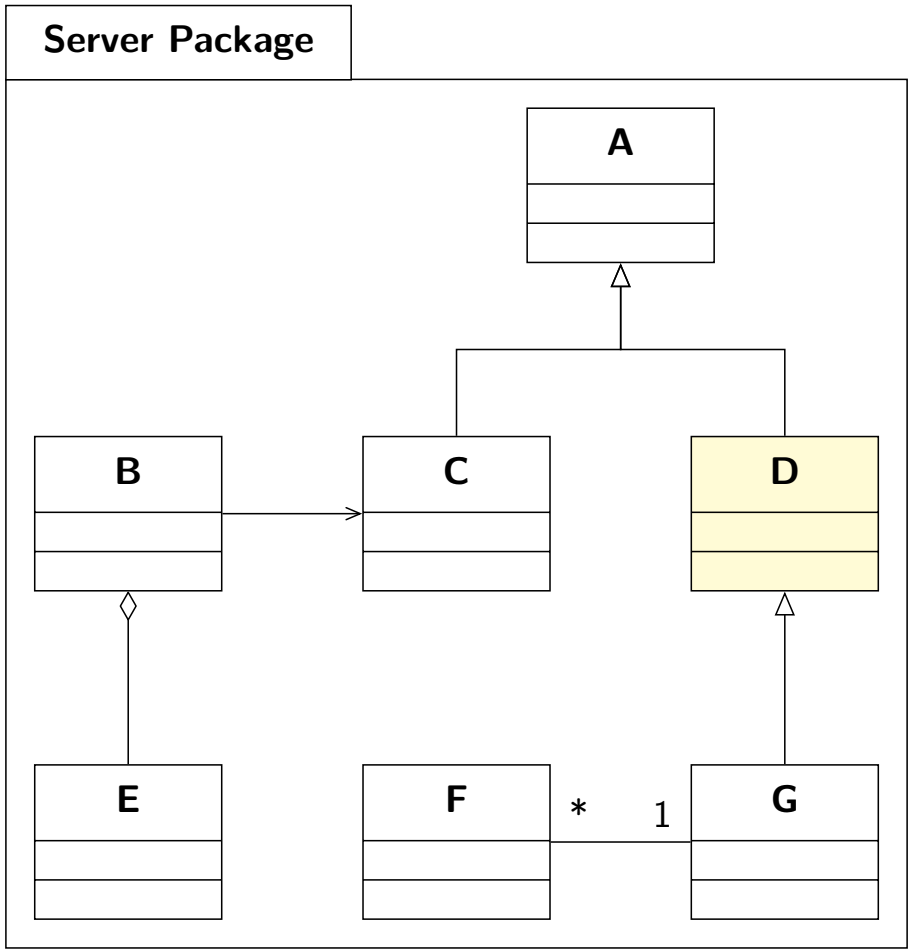
策略 2　如果一个类需要在多个结点上创建对象实例：
把这个类分布到每个需要创建其实例的结点上，
其中一个作为正本，其他作为副本

确定系统分布方案



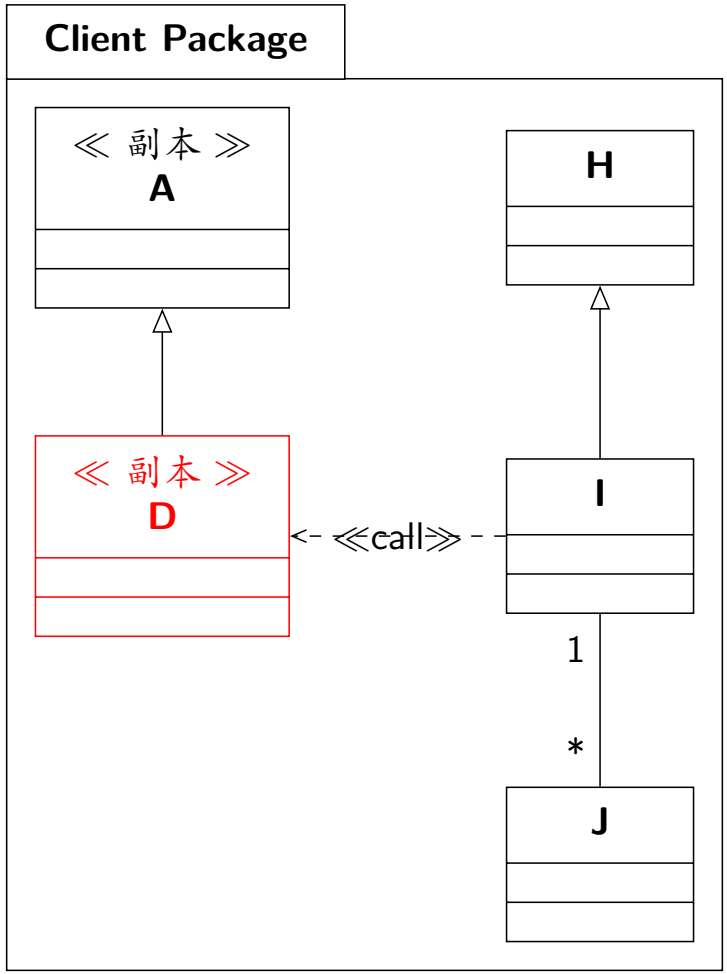
例：一个集中式类图

确定系统分布方案



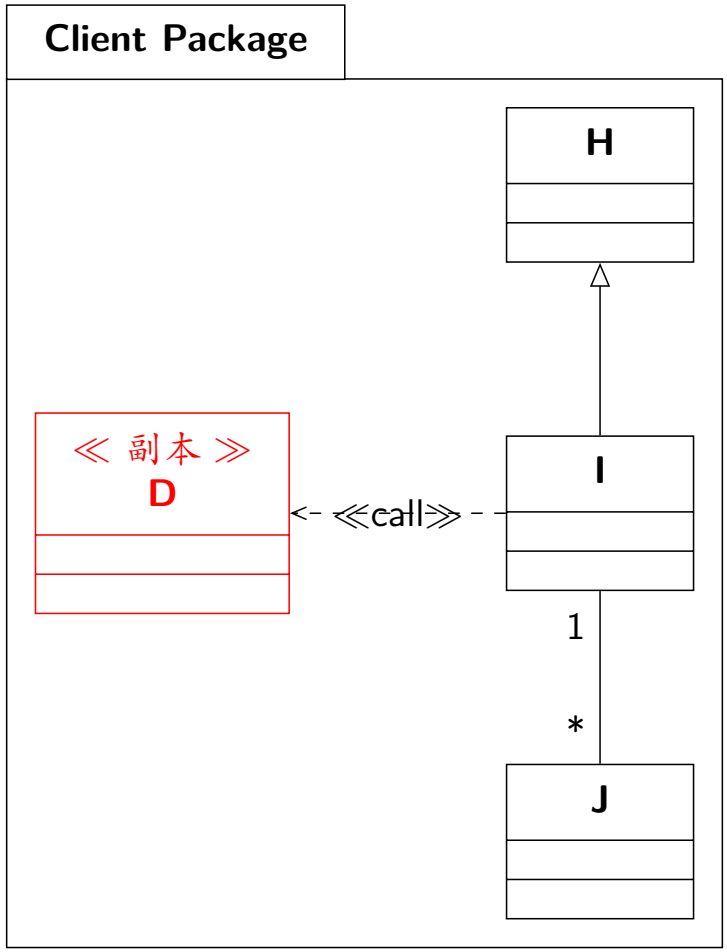
例：服务器包

确定系统分布方案



例：客户机包（第一种策略）

确定系统分布方案



例：客户机包（第二种策略）

识别控制流

1. 以结点为单位识别控制流
   - 不同结点上程序的并发问题已经解决
   - 考虑在每个结点上运行的程序还需要如何并发
2. 从用户需求出发认识控制流
   - 有哪些任务必须在同一台计算机上并发执行
3. 从用况认识控制流关注描述如下三类功能的用况
   - 要求与其他功能同时执行的功能
   - 用户随时要求执行的功能
   - 处理系统异常事件功能

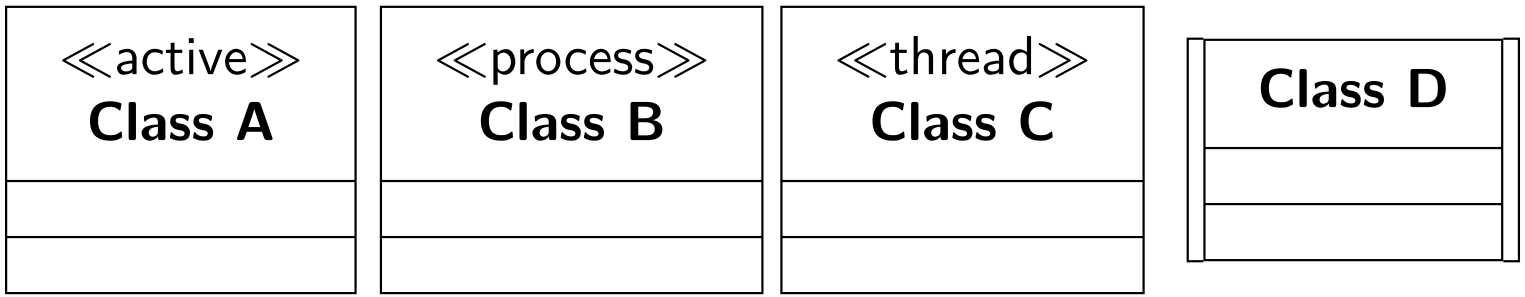
识别控制流

4. 参照 OOA 模型中的主动对象
5. 为改善性能而增设的控制流
   - 高优先级任务
   - 低优先级任务
   - 紧急任务
6. 实现并行计算的控制流（线程/进程）
7. 实现结点之间通讯的控制流（进程）
8. 对其它控制流进行协调的控制流

用主动对象表示控制流

控制流

是主动对象中一个主动操作的一次执行。其间可能要调用其他对象的操作，后者又可能调用另外一些对象的操作，这就是一个控制流的运行轨迹。



UML1 和 UML2 中的主动类表示法

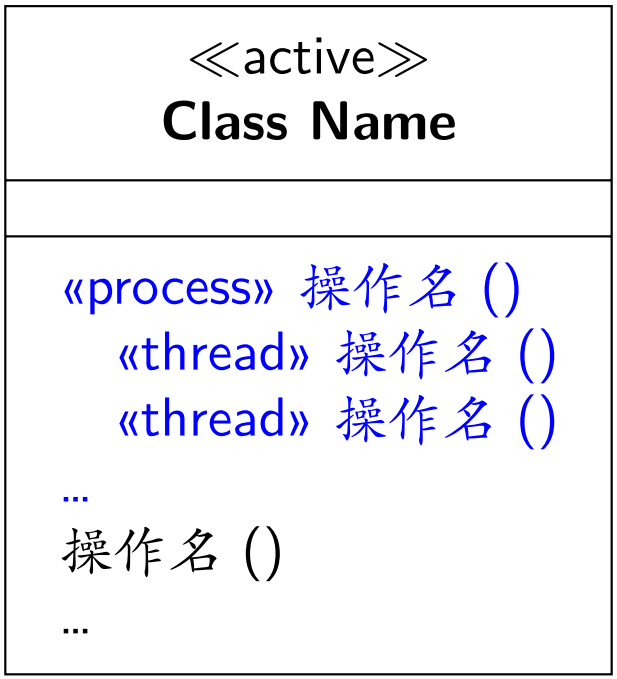
用主动对象表示控制流

问题：

一个主动类可以有多个主动操作和若干被动操作，UML 的表示法如何显式地表示哪个（哪些）操作是主动操作？

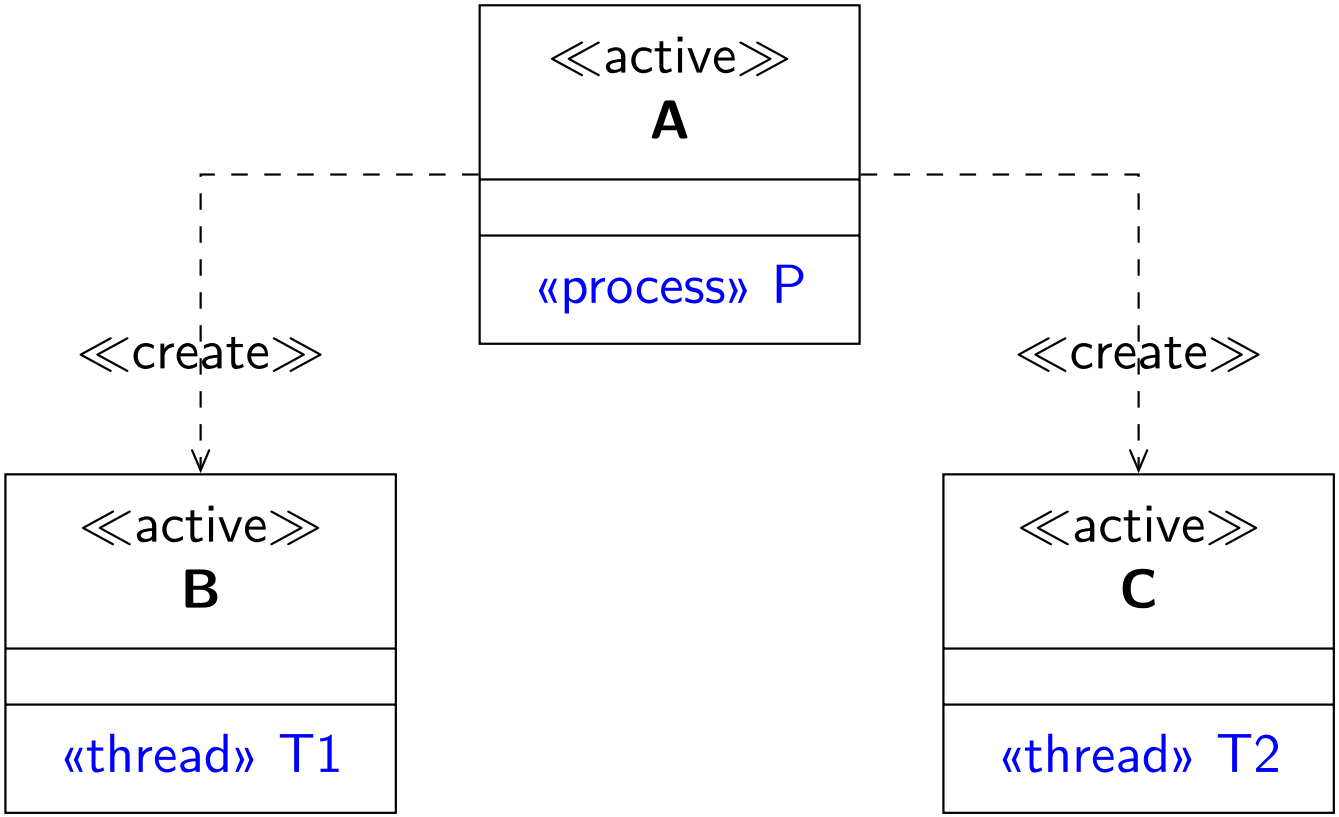
用主动对象表示控制流

用关键词表示主动操作

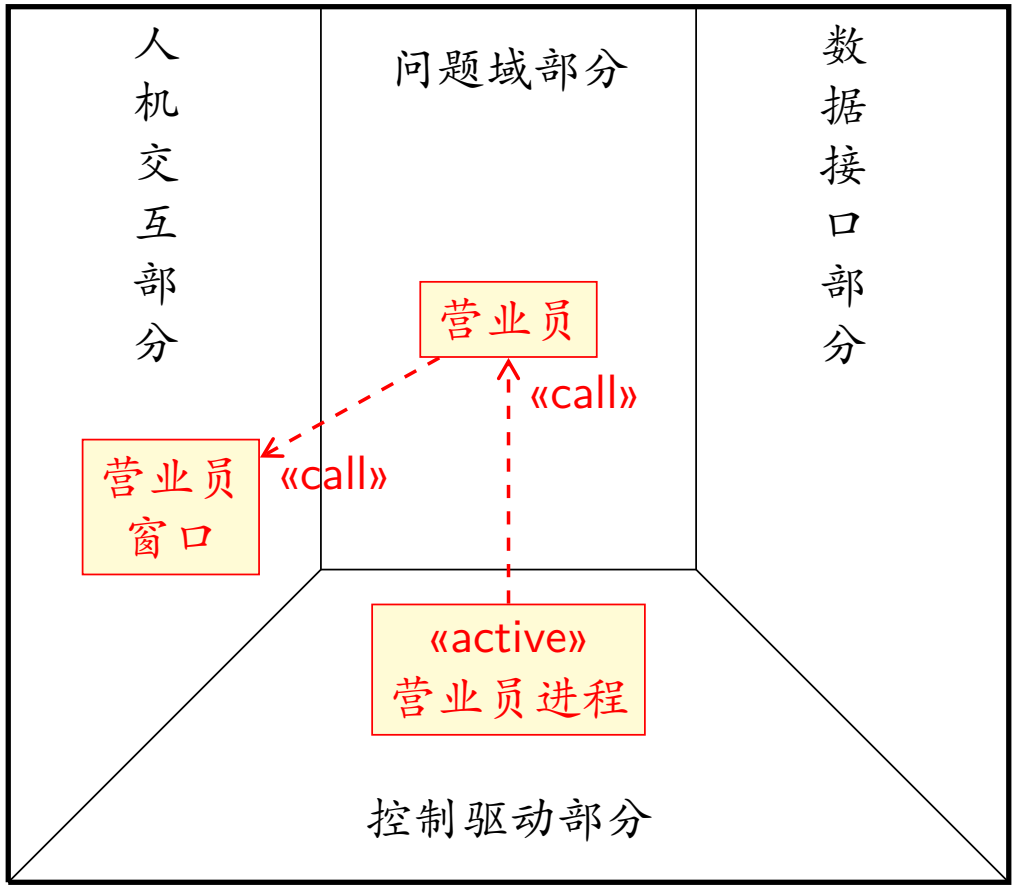


用主动对象表示控制流

显式地表示由进程创建线程

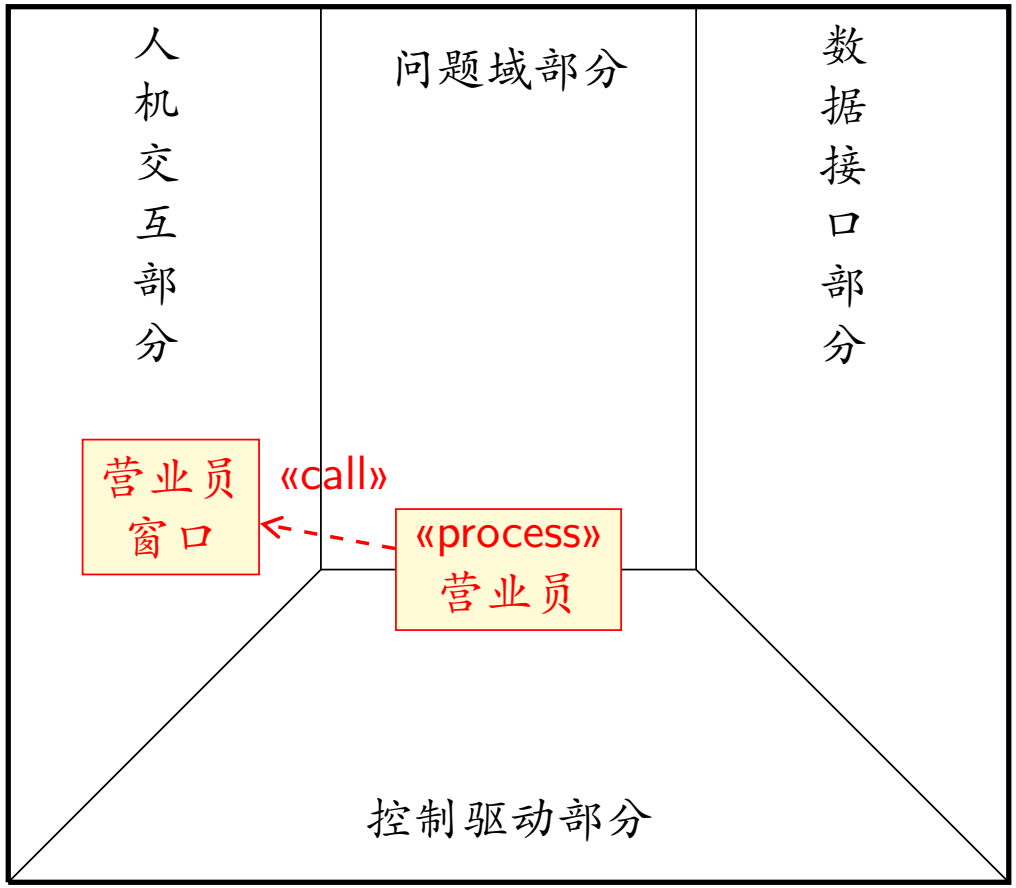


主动对象在 OOD 模型中的位置



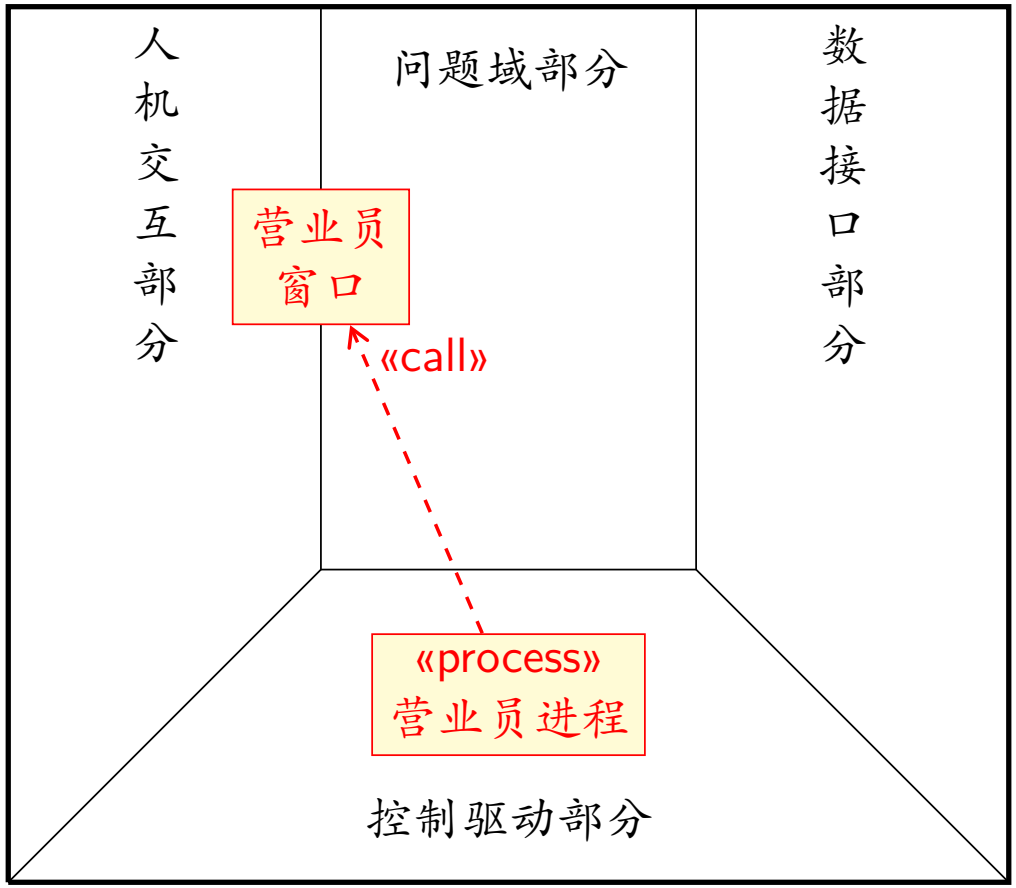
订单系统中营业员对象：无交叉方案

主动对象在 OOD 模型中的位置



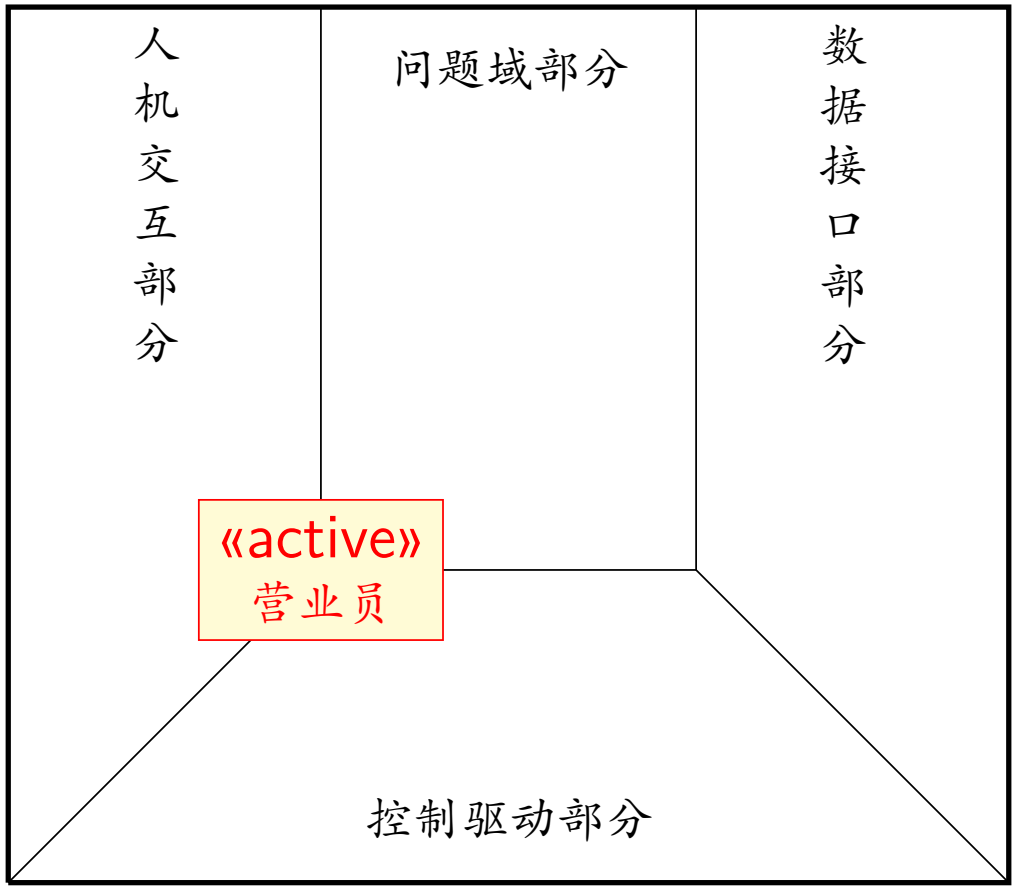
订单系统中营业员对象：问题域和控制驱动部分交叉

主动对象在 OOD 模型中的位置



订单系统中营业员对象：问题域和人机交互部分交叉

主动对象在 OOD 模型中的位置



问题域、人机交互、控制驱动部分都交叉