

软件体系结构设计

内容

□何为软件体系结构

- ✓概念、组成元素、视图与模型
- ✓软件体系结构风格

□如何开展软件体系结构设计

- ✓价值、目标
- ✓体系结构设计过程

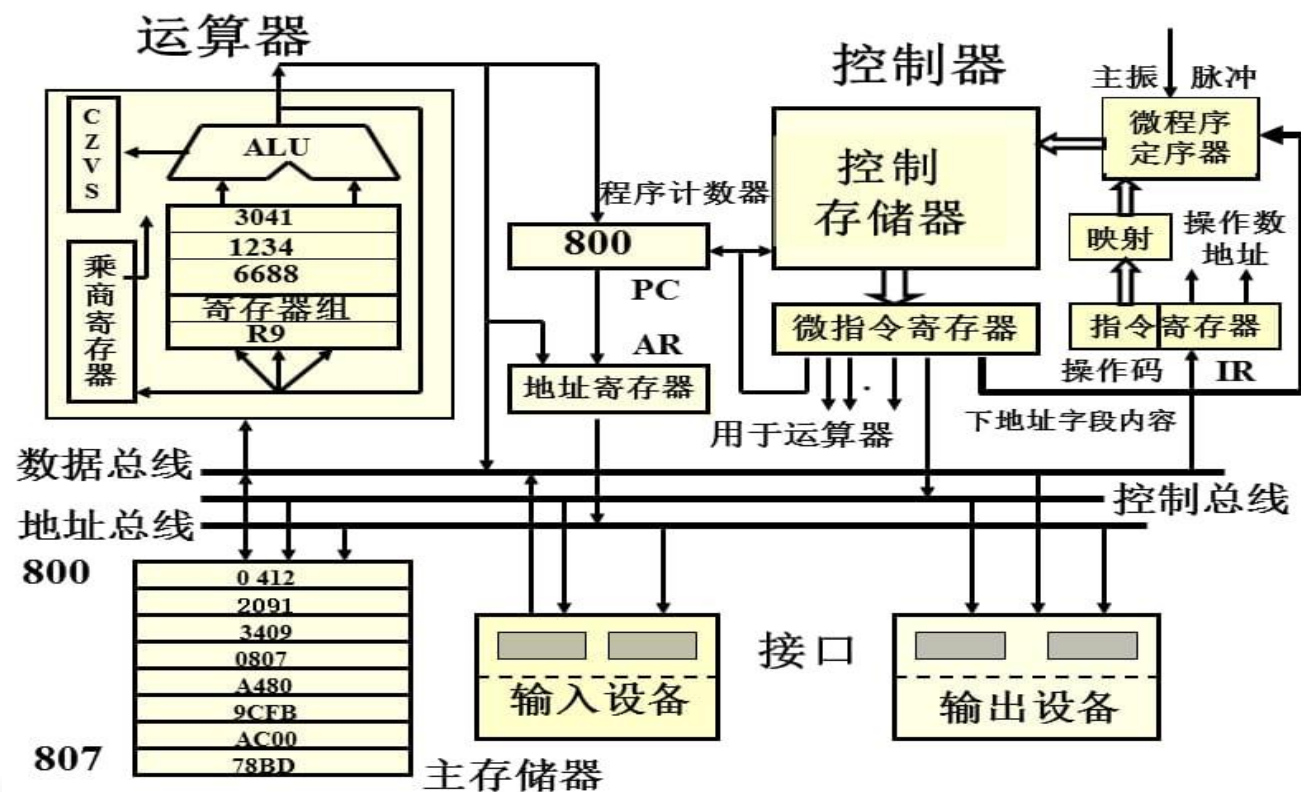
□软件体系结构设计结果及评审

- ✓文档模板、验证原则



从计算机组成理解“体系结构”

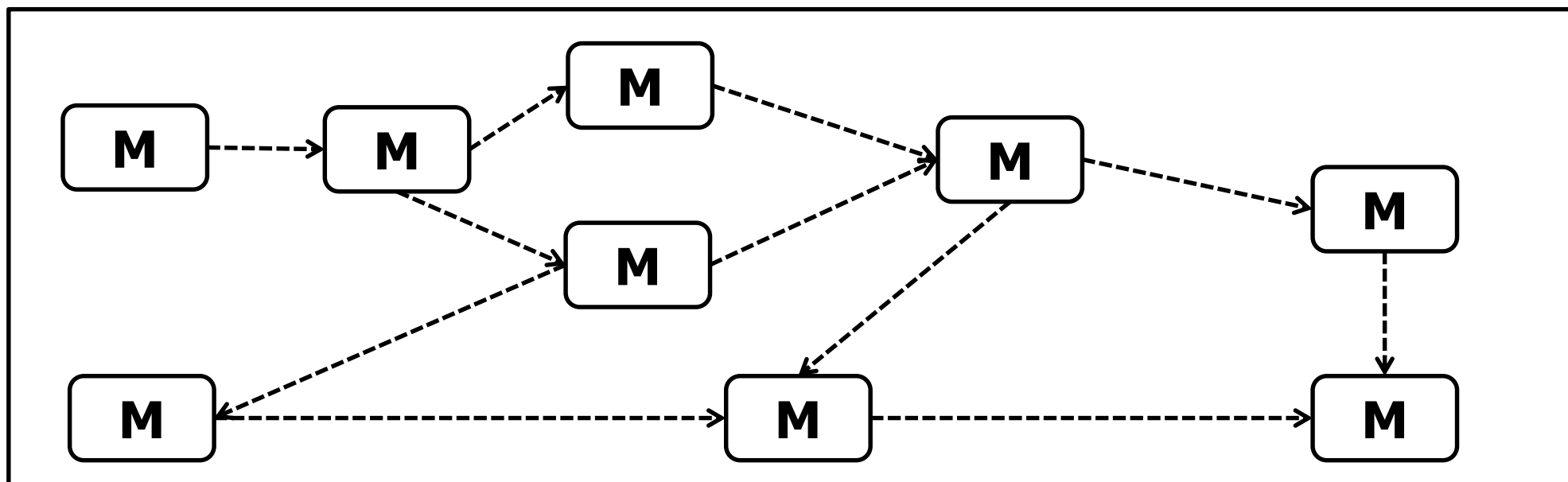
- 体系结构存在于很多事物中，如建筑、计算机、汽车等
- **构件**：控制器、运算器、内存储器、外存储器、输入设备.....
- **连接件**：总线（控制总线、地址总线、数据总线）



1.1 软件体系结构的概念

□软件体系结构(Software Architecture, SA)

✓也称软件架构，从**宏观、抽象**角度刻画组成软件系统的**构成元素**及它们之间的**逻辑关联**



1.2 软件体系结构的组成元素

□ 构件(Component)

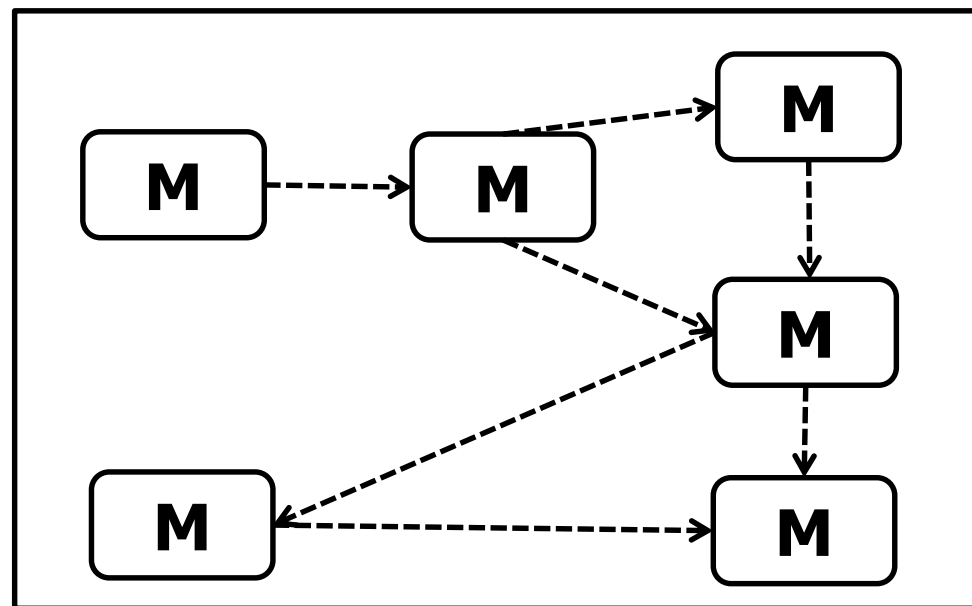
✓ 构成软件系统的模块或单元

□ 连接件(Connector)

✓ 构件之间的连接和交互关系

□ 约束(Constraint)

✓ 构件中的元素应满足的条件，以及构件组装成更大模块时应满足的条件



1. 构件






















□ 何为构件

- ✓ 软件系统中的**可复用软件单元**，具有特定的**功能**和**对外接口**
- ✓ 构件可以是代码库、类、函数、服务或任意可独立实现和测试的软件单元

□ 特点

- ✓ **可分离**：一个或数个可独立部署执行代码文件
- ✓ **可替换**：构件实例可被其它任何实现了相同接口的另一构件实例所替换
- ✓ **可配置**：可通过配置机制修改构件配置数据，影响构件对外服务的功能或行为
- ✓ **可复用**：构件可不经源代码修改，无需重新编译，即可应用于多个软件项目或软件产品

示例：构件

 FileInfo	2020/4/4 10:28	文件夹
 HowTo	2020/4/4 10:27	文件夹
 Javascripts	2020/4/4 10:28	文件夹
 PalmPilot	2020/4/4 10:28	文件夹
 plug_ins	2020/4/4 10:28	文件夹
 Sequences	2020/4/4 10:28	文件夹
 SPPlugins	2020/4/4 10:28	文件夹
 Updater	2020/4/4 10:28	文件夹
 WebAccess	2020/4/4 10:28	文件夹
 Ace.dll	2003/5/14 23:45	应用程序扩展
 <u>Acrobat.exe</u>	2003/5/19 12:17	应用程序
 acrobat.tlb	2003/3/27 17:47	TLB 文件
 <u>Acrofx32.dll</u>	2003/5/15 0:47	应用程序扩展
 <u>AcroIEFavClient.dll</u>	2003/5/15 1:03	应用程序扩展
 <u>AdobeUpdateManager.exe</u>	2003/5/15 0:56	应用程序
 <u>Agm.dll</u>	2003/5/14 23:45	应用程序扩展
 <u>atl.dll</u>	2000/7/14 18:18	应用程序扩展
 AXEParse.dll	2003/5/14 23:45	应用程序扩展
 Bib.dll	2003/5/14 23:45	应用程序扩展
 CoolType.dll	2003/5/14 23:45	应用程序扩展
 cos2xpdf.xml	2003/5/15 0:01	XML 文档

Acrobat软件中的构件

- 物理存在
- 粗粒度
- 可访问
- 可运行

2. 连接件

□连接件表示构件之间的**连接和交互**关系

- ✓每个构件并非孤立，它们之间通过连接进行交互
- ✓交互的目的是为了交换数据、获取功能和服务

□构件之间的典型交互方式

- ✓消息传递 (Message Sending)
- ✓事件通知和广播
- ✓过程调用
- ✓远程过程调用 (Remote Procedure Call, RPC)
- ✓主题订阅等等



**硬件
接口**

3. 约束

□约束

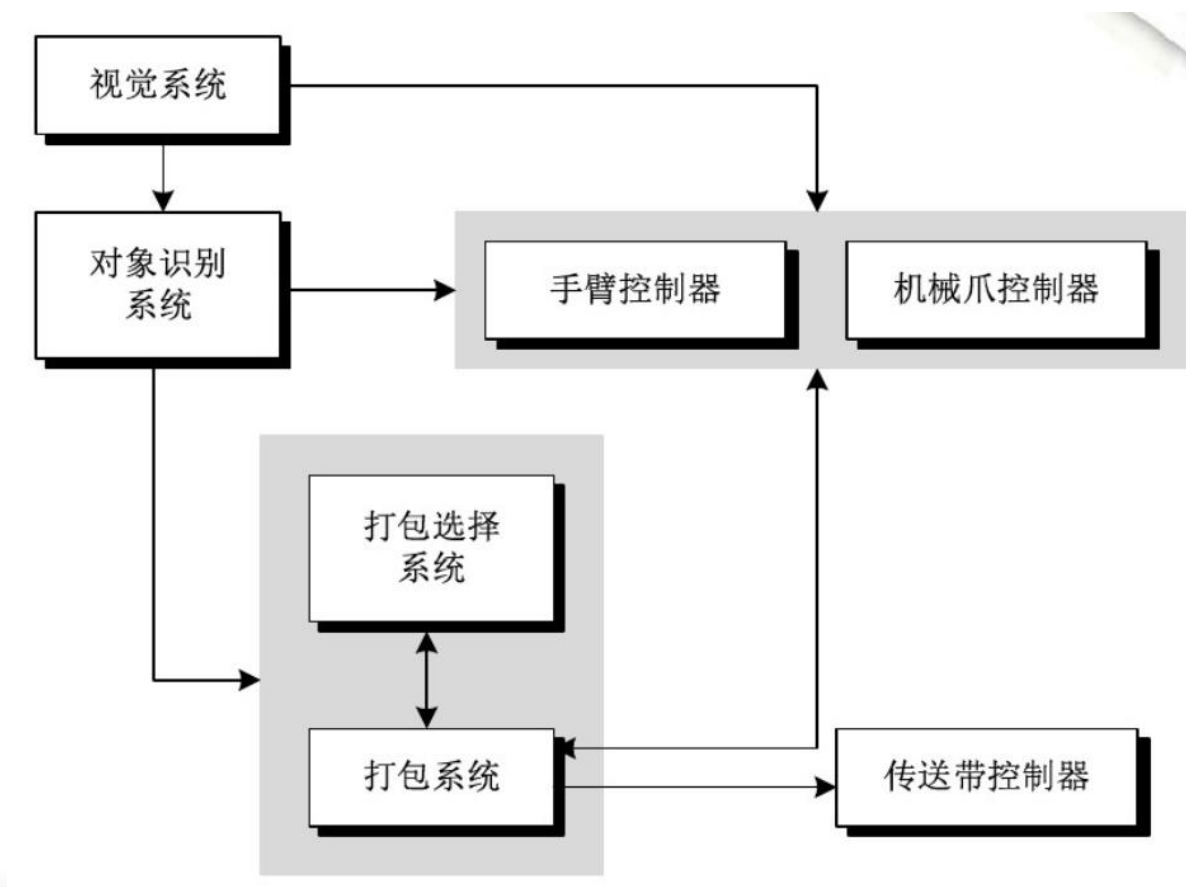
- ✓对构件的布局及相互之间的交互进行必要的**限制**

□示例

- ✓**高层次**软件元素可向**低层次**软件元素**发请求**，反之不行
- ✓每个软件元素根据其职责位于适当的层次，不可错置。如核心层不能包含界面输入输出职责
- ✓每个层次都是**可替换的**，一个层次可以被实现**同样服务接口**的层次所替代

实例：打包机器人控制系统体系结构

- 机器人使用视觉子系统获取传送带上的对象，识别对象类型并选择正确的打包方式，然后从传送带取下对象，打包，再送往另一个传送带。



职责：
感知-决策-控制

1.3 软件体系结构模型表示

基于不同的**关注点**，我们可以从多个**不同视角**对软件体系结构进行建模

- **逻辑视图**：描述系统的逻辑架构，主要通过包图来组织类、接口和其他包等元素
- **运行视图**：在特定时刻构件的**运行**情况，如同步或并行关系、交互与协作等，以**活动图**、**顺序图**表示
- **开发视图**：从**程序员**的角度透视系统，描述开发环境中**软件**的静态**组织结构**，以**构件图**表示
- **物理视图**：构件的物理**部署**及其连接和交互，以**部署图**表示

1.3.1 包图

视点	图 (diagram)	说明
结构	包图 (package diagram)	从包层面描述系统的静态结构
	类图 (class diagram)	从类层面描述系统的静态结构
	对象图 (object diagram)	从对象层面描述系统的静态结构
	构件图(component diagram)	描述系统中构件及其依赖关系
行为	状态图(statechart diagram)	描述状态的变迁
	活动图(activity diagram)	描述系统活动的实施
	通信图(communication diagram)	描述对象间的消息传递与协作
	顺序图(sequence diagram)	描述对象间的消息传递与协作
部署	部署图 (deployment diagram)	描述系统中工件在物理运行环境中的部署情况
用例	用例图 (use case diagram)	从外部用户角度描述系统功能

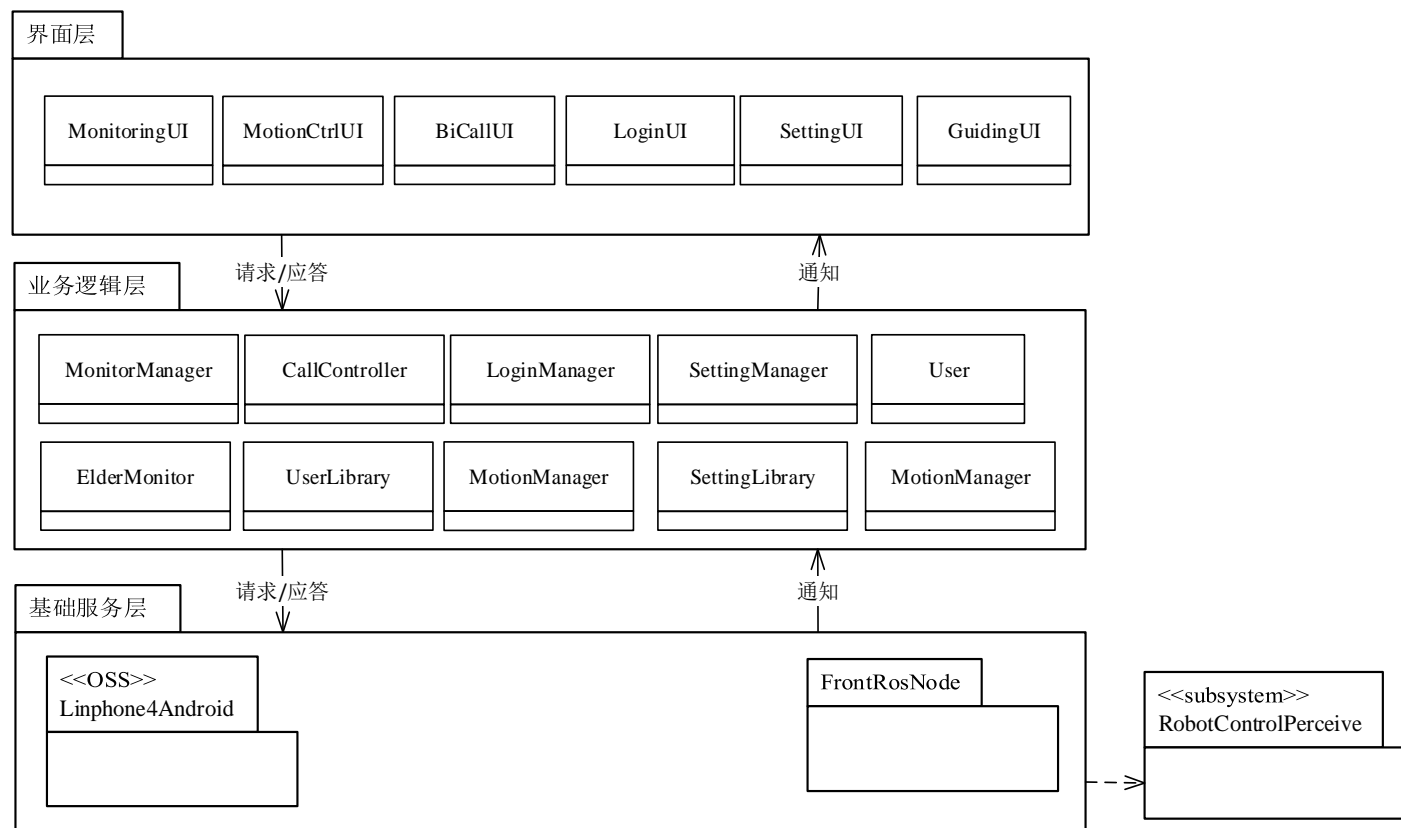
包图

□ 功效

- ✓ 对设计类进行**分类管理**的一种形式
- ✓ 将复杂系统抽象为一个包，下层继续分解为子包、类等
- ✓ 以**职责**为依据划分包并**分层**

□ 包间的关系

- ✓ **组成、依赖**



老人状况监控APP子系统 架构图

包图的作用

□侧重于描述系统的**逻辑结构**和模型元素的组织方式

✓将大型软件系统划分成不同包，可以更好规划和**组织代码结构**

□作为**模型管理**的基本单元

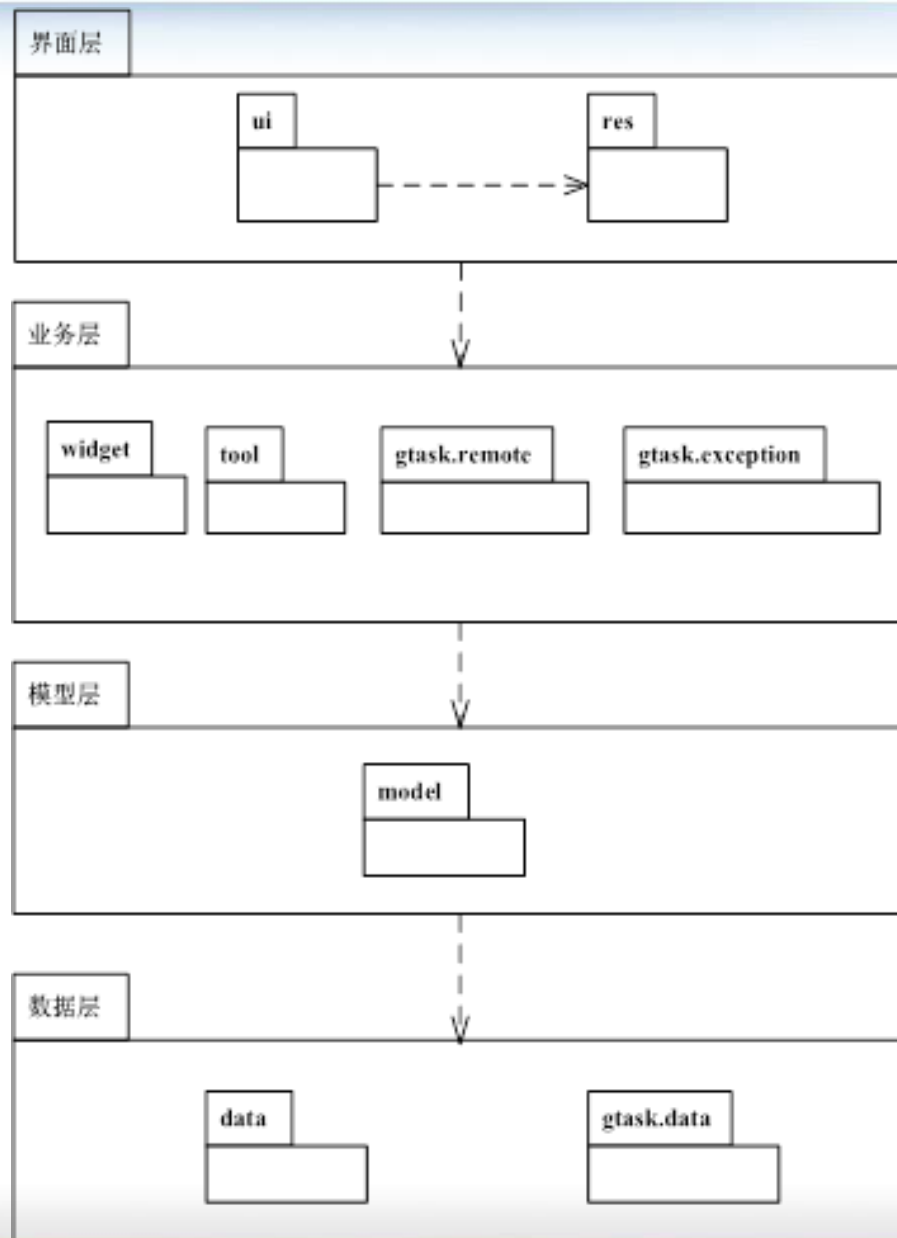
✓以包为单位**分派开发任务**安排计划，包是天然的基本处理单元

□作为**访问控制**的基本手段

✓将包视为**名字空间**，通过“包名+模型元素名”构成唯一限定名

示例：基于包图的逻辑视图表示

基于包图的小米便签系统体系结构



1.3.2 构件图

视点	图 (diagram)	说明
结构	包图 (package diagram)	从包层面描述系统的静态结构
	类图 (class diagram)	从类层面描述系统的静态结构
	对象图 (object diagram)	从对象层面描述系统的静态结构
	构件图(component diagram)	描述系统中构件及其依赖关系
行为	状态图(statechart diagram)	描述状态的变迁
	活动图(activity diagram)	描述系统活动的实施
	通信图(communication diagram)	描述对象间的消息传递与协作
	顺序图(sequence diagram)	描述对象间的消息传递与协作
部署	部署图 (deployment diagram)	描述系统中工件在物理运行环境中的部署情况
用例	用例图 (use case diagram)	从外部用户角度描述系统功能

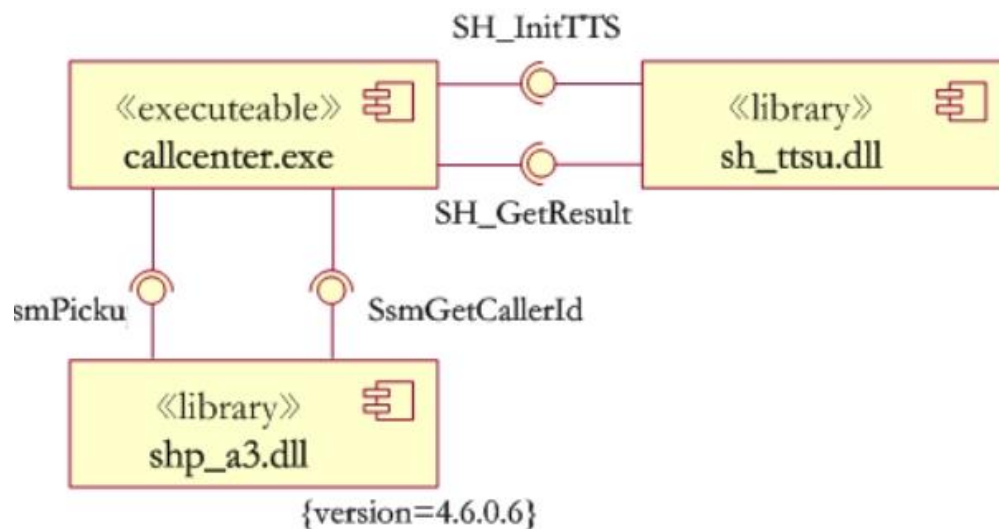
构件图

□功效

- ✓描述软件系统中**构件**及构件间的关系
- ✓侧重于描述系统的**物理**组织结构，即**代码**层面。
- ✓构件是可部署的,而类不能。

□图的构成

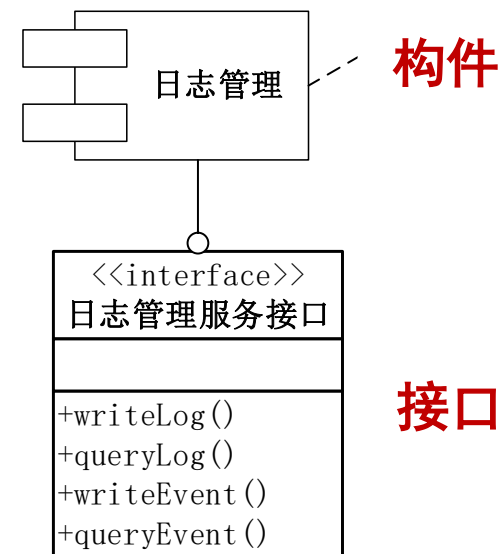
- ✓**节点**：构件
- ✓**边**：构件间的依赖关系



构件及其接口

□ 构件

- ✓ 构件内部实现部分进行了封装和隐藏，外部只能通过**接口**访问其服务
- ✓ 构件**接口与内部实现应严格分离**

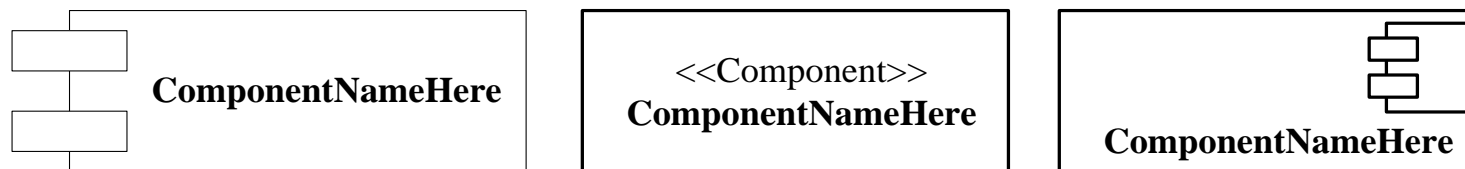


□ 接口

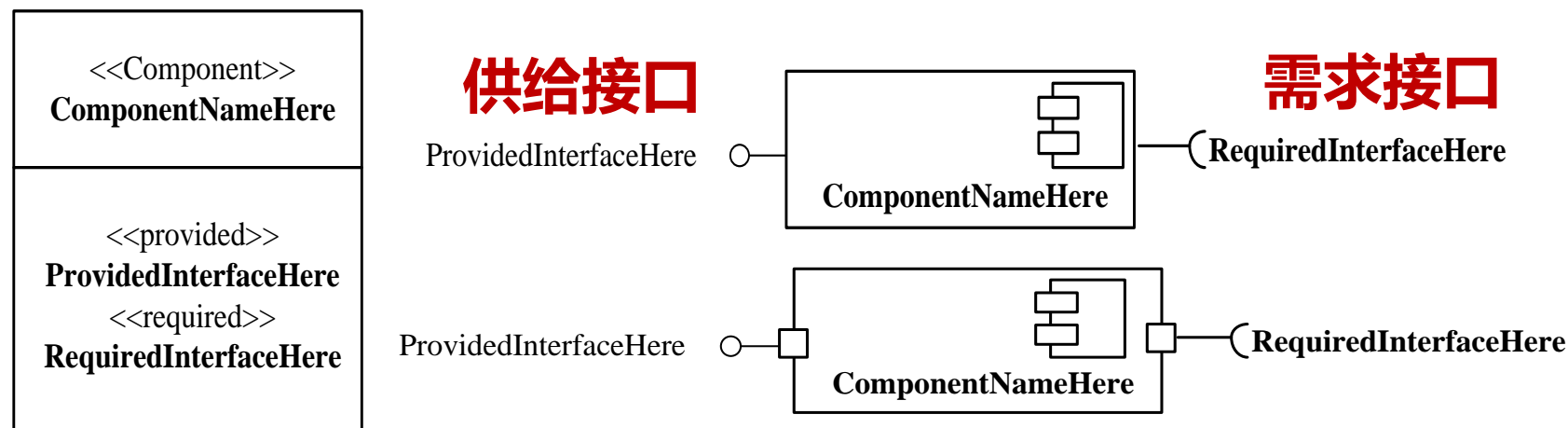
- ✓ 一组操作 和/或 属性的说明(不含实现)，用作服务提供方和使用方之间的**协议**
- ✓ 接口由类或构件具体**实现**
- ✓ 每个构件包含两种接口：**供给接口(提供服务方) 或 需求接口**

构件和接口的表示

□ 构件的三种图元



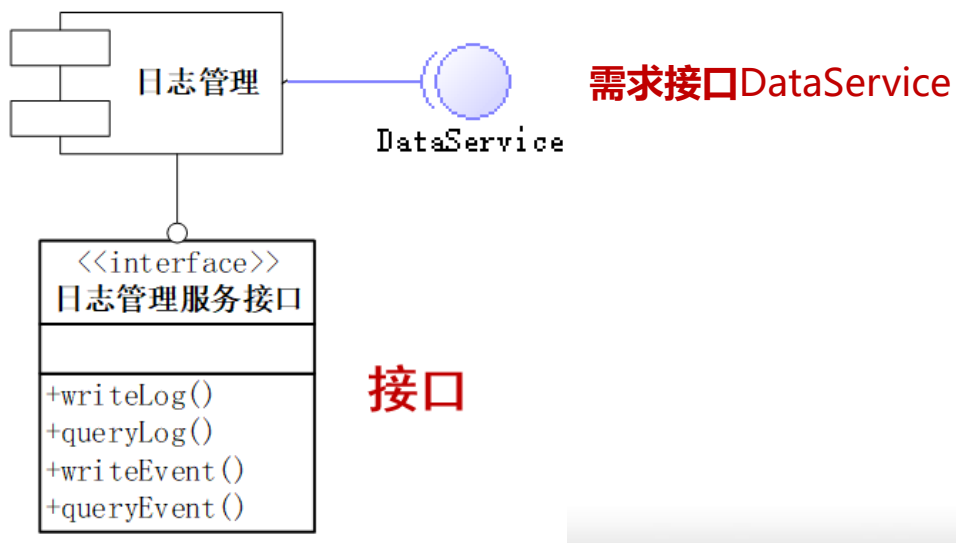
□ 构件及其接口的两种表示方法



供给接口实现

供给接口Ilog

```
public interface ILog {  
    void writeLog();  
    void queryLog();  
    void writeEvent();  
    void queryEvent();  
}
```

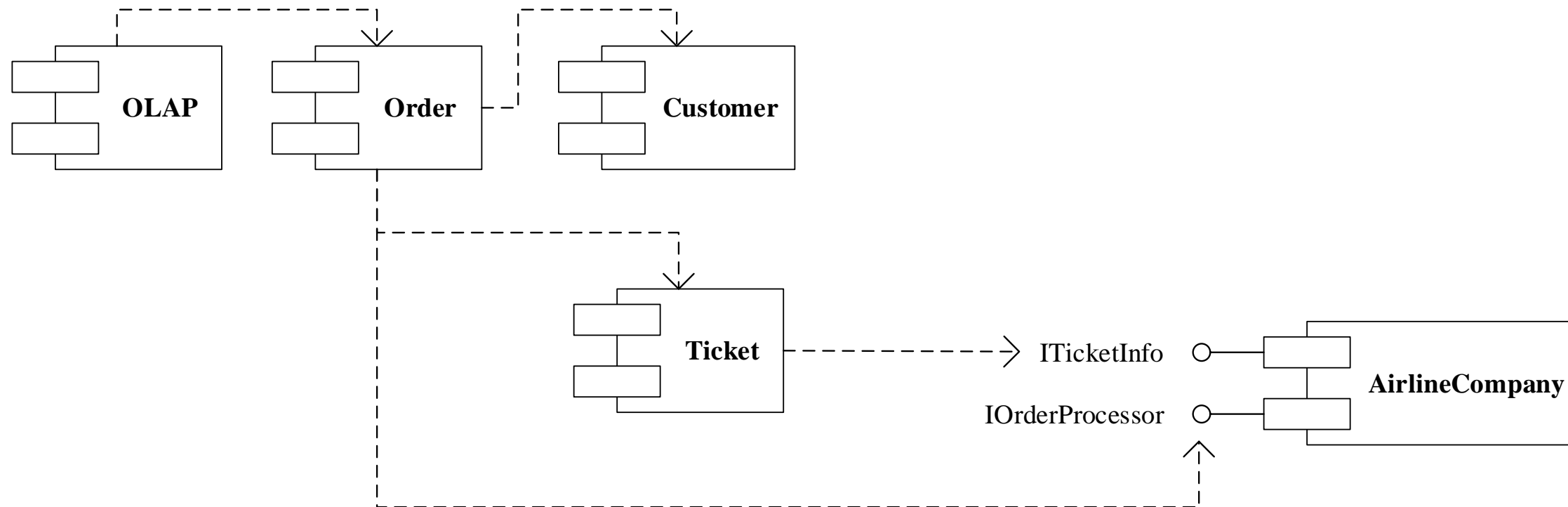


//构件实现LogManagement

```
public class LogManagement implements Ilog{  
    public void writeLog() {  
        .....  
    }  
    public void queryLog(){  
        .....  
    }  
  
    public void consumeMethod(DataService ds) {  
        ds.Service(); //通过需求接口取得其它构件的服务  
    }  
}
```

示例：基于构件图的开发视图表示

依赖关系



1.3.3 部署图

视点	图 (diagram)	说明
结构	包图 (package diagram)	从包层面描述系统的静态结构
	类图 (class diagram)	从类层面描述系统的静态结构
	对象图 (object diagram)	从对象层面描述系统的静态结构
	构件图(component diagram)	描述系统中构件及其依赖关系
行为	状态图(statechart diagram)	描述状态的变迁
	活动图(activity diagram)	描述系统活动的实施
	通信图(communication diagram)	描述对象间的消息传递与协作
	顺序图(sequence diagram)	描述对象间的消息传递与协作
部署	部署图 (deployment diagram)	描述系统中制品在运行环境中的部署情况
用例	用例图 (use case diagram)	从外部用户角度描述系统功能

□功效

- ✓表示系统中各个**可执行制品**(artifact)在运行环境中的部署情况
- ✓**可执行制品**是可独立运行的**实现单元**，如DLL文件、Java类库 (jar) 等

□图的构成

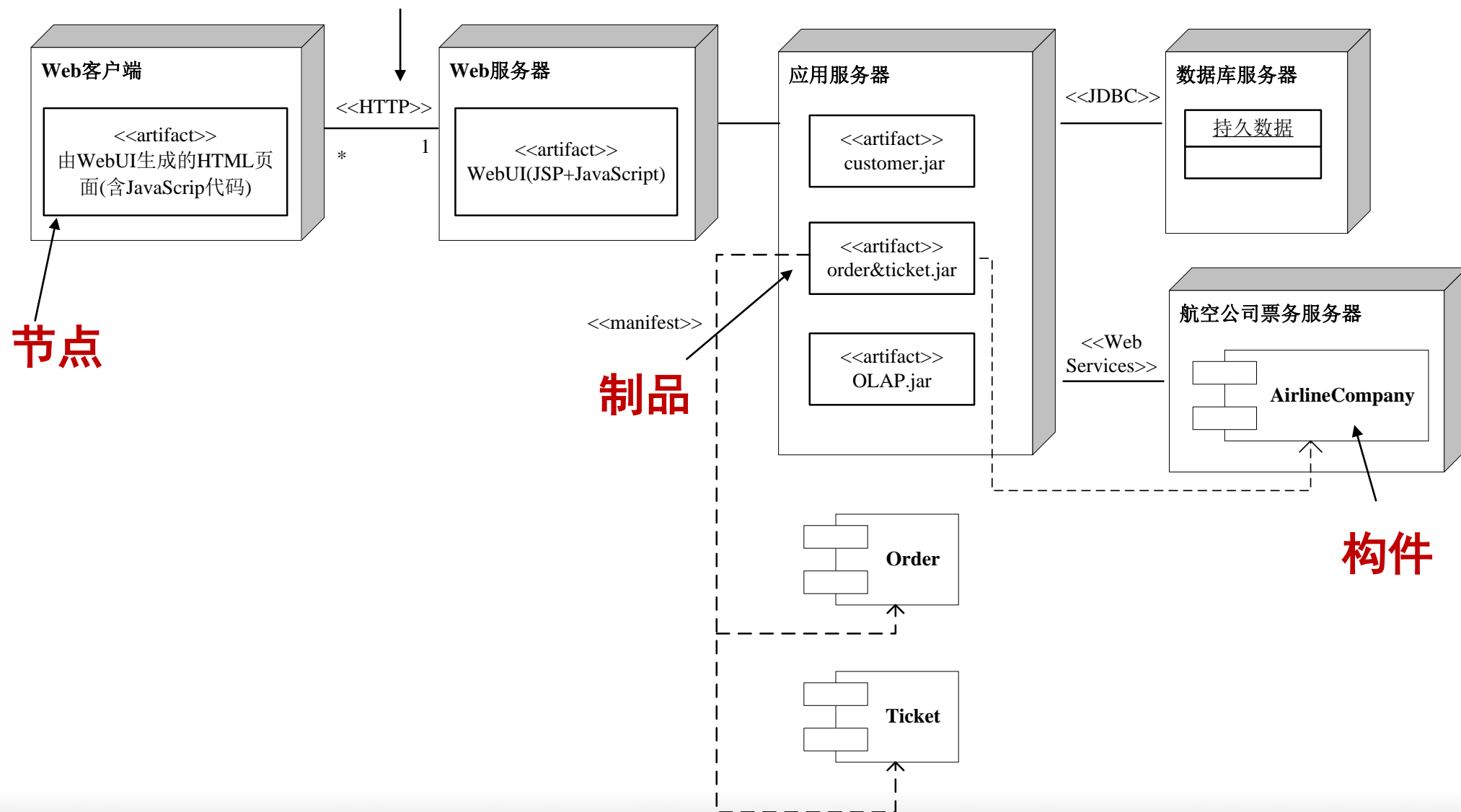
- ✓节点：**计算节点、制品、构件**
- ✓边：通信和依赖

□两种部署图

- ✓**描述性部署图**：描述软件的逻辑布局
- ✓**实例性部署图**：针对具体运行环境和特定的系统配置，描述系统的物理部署情况

示例：描述性部署图

节点间的通信连接

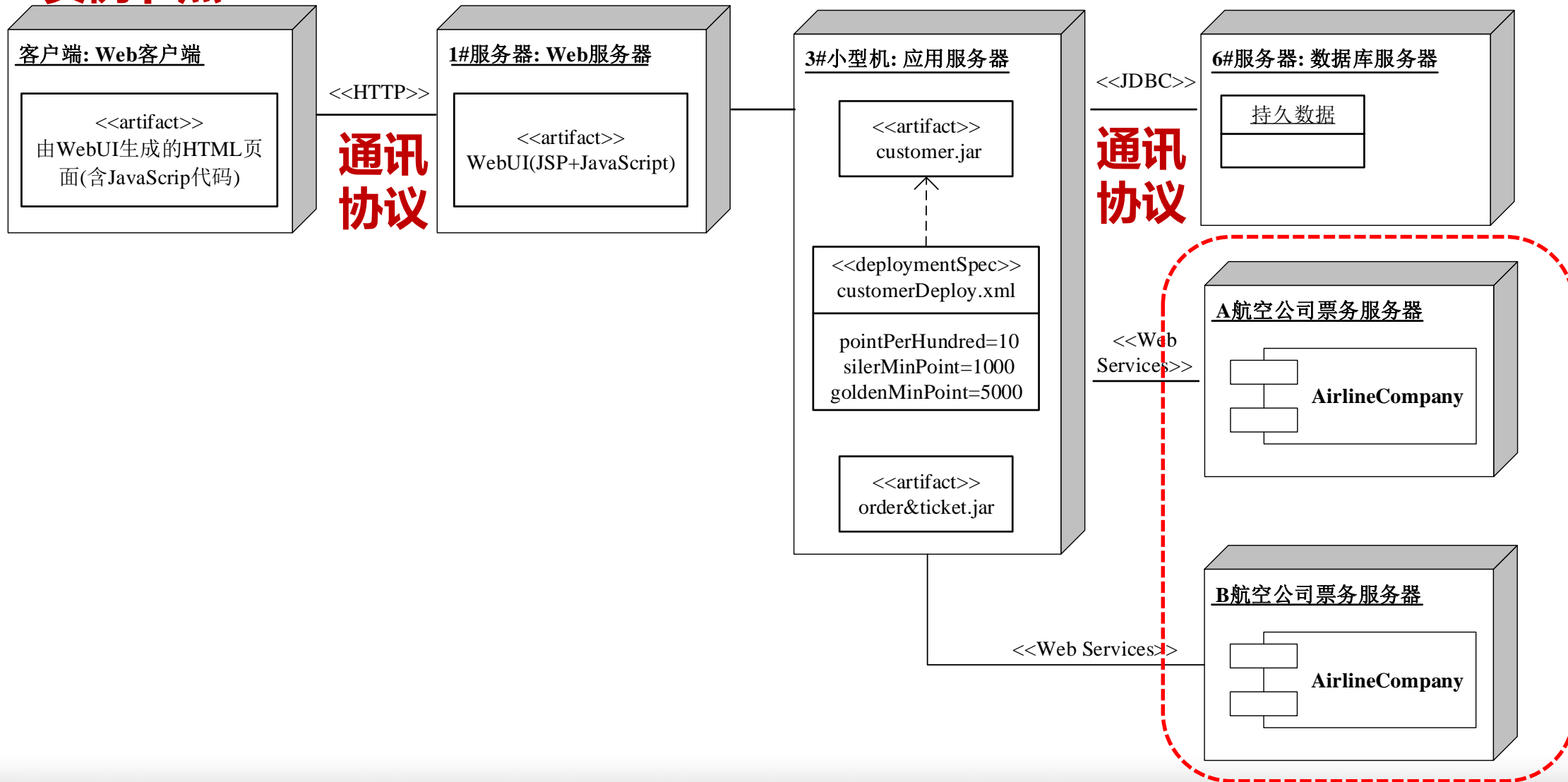


实例性部署图

- 实例性部署图与描述性部署图之间的关系可类比为对象图与类图之间的关系
- 实例性部署图中节点的命名方式为“节点名：类型名”，其中类型名为描述性部署图中的节点名

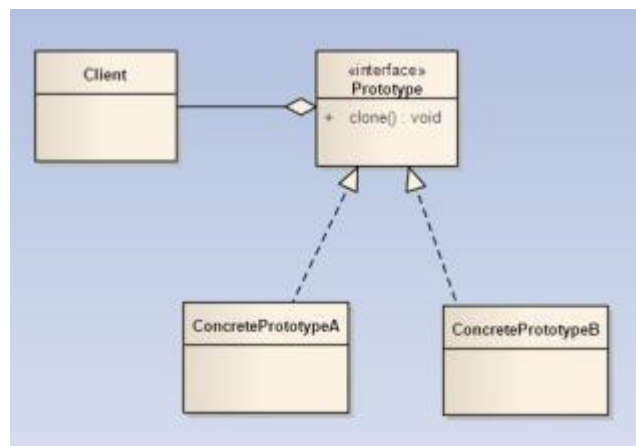
示例：实例性部署图

实例节点



1.4 软件设计模式

- 设计模式：针对可能**重复出现**的一些设计问题，积累的经过**充分实践考验**的解决方案。
- 要提高设计的质量，必须站在“巨人”的肩上，即：借用以往的**经验**来解决当前问题。



- 名称
- 问题
- 施用和约束条件
- 解决方案
- 效果



不同层次的设计模式

□体系结构风格

✓面向整个软件系统

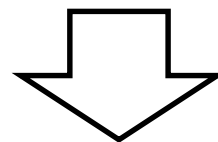
□构件设计模式

✓面向子系统或者构件

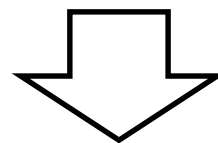
□实现设计模式

✓针对子系统或构件中的某个特定问题

整体、全局



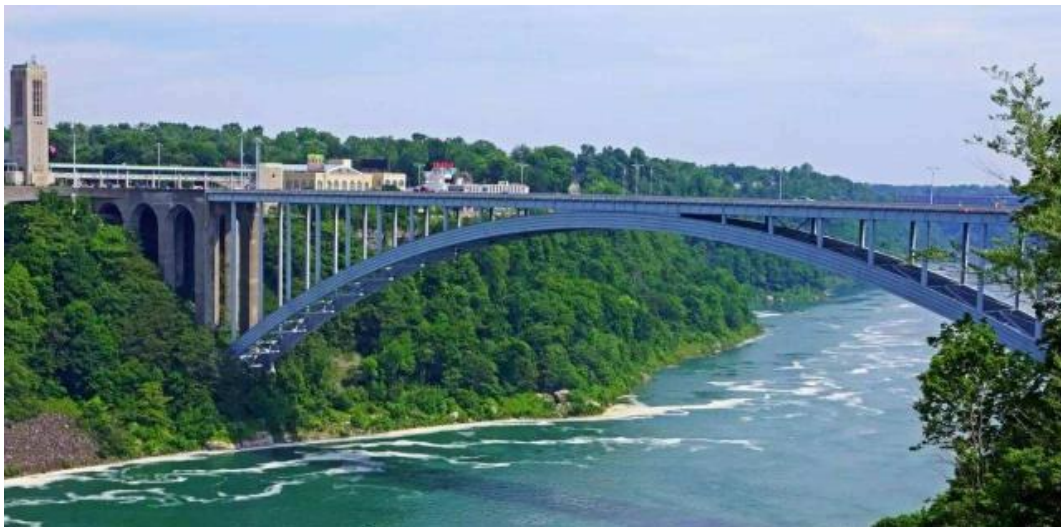
局部



细节

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。



体系结构风格

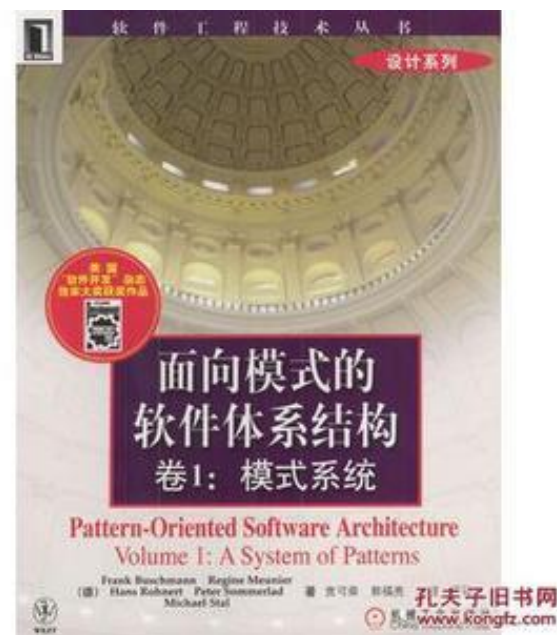
- ❑ 软件体系结构也有特定的“风格”：描述某一应用领域中众多软件系统所**共有**的结构和特性。
- ❑ 不同风格体现在**构件类型、连接方式、拓扑、约束**等差异性上
- ❑ 对同一软件，如果选用**不同设计风格**，得到的软件体系结构也将完全不同

有哪些**可选风格**？针对不同应用，**如何选择**适当的体系结构风格？



常用软件体系结构风格

- 分层风格
- 管道与过滤器风格
- 黑板风格
- MVC风格
- SOA风格
- 总线风格
-



1.4.1 分层体系结构

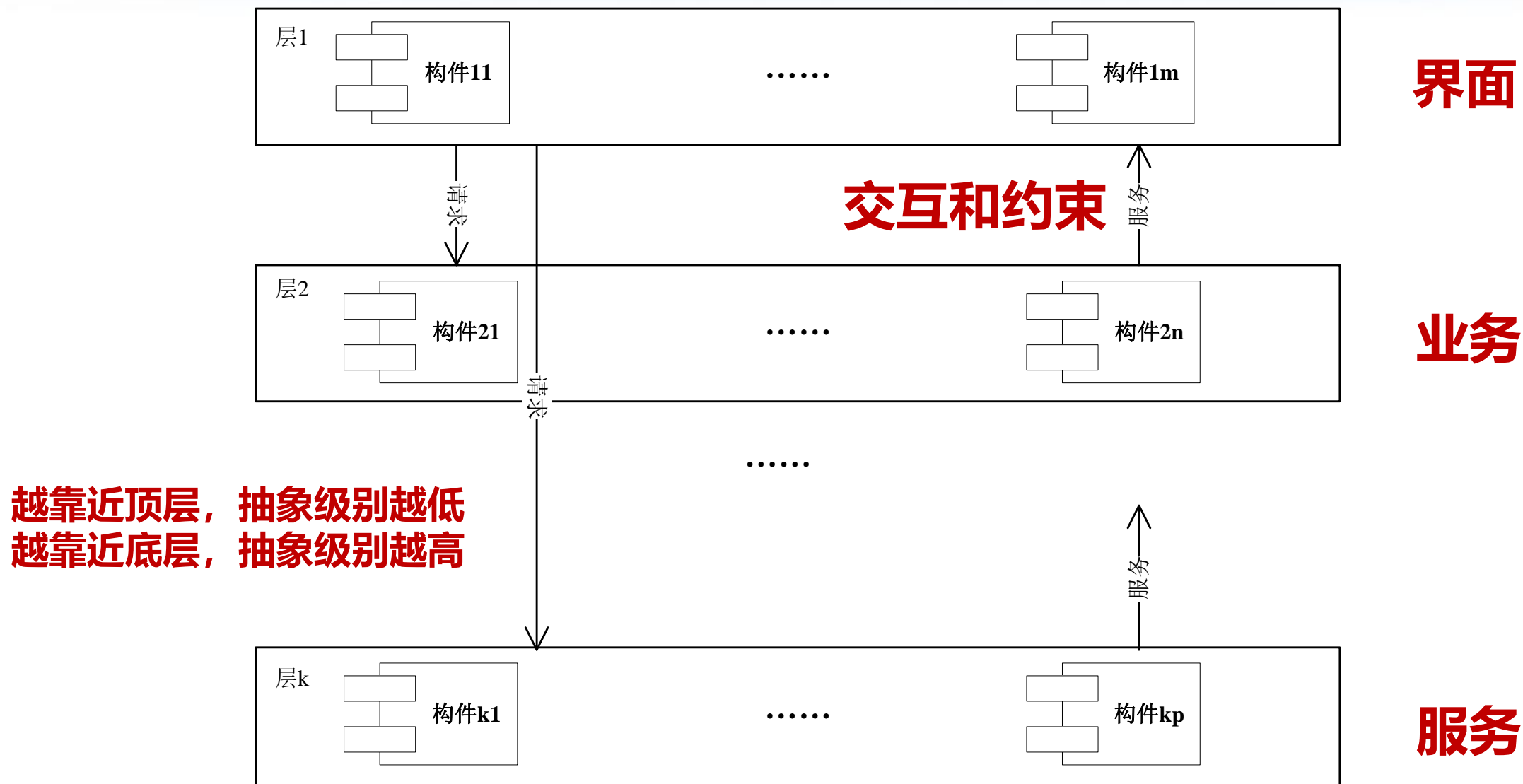
□模式思想

- ✓将软件系统按照**抽象级别**逐次递增或递减的顺序，组织为**若干层次**，每层由一些抽象级别相同构件组成

□层次结构示例

- ✓**顶层**：直接面向**用户**提供软件系统的交互界面
- ✓**底层**：则负责提供**基础性、公共性**的技术服务，它比较接近于硬件计算环境、操作系统或数据库管理系统
- ✓**中间层**：介乎二者之间，负责具体的**业务处理**

示例：分层体系结构



分层体系结构的约束

□ 层次间的关系

- ✓ 每层为其**紧邻上层**提供服务，使用**紧邻下层**所提供的服务（**约束**）
- ✓ 上层向下层发出服务**请求**，下层向上层提供事件**信息**（**连接**）

□ 服务接口的组织方式（连接）

- ✓ 层次中的每个构件**分别**公开其服务接口
- ✓ 每个层次**统一**对外提供整合的服务接口

分层体系结构的特点

□松耦合

- ✓减低整个软件系统的耦合度

□可替换

- ✓一个层次可以有多个实现实例

□可复用

- ✓层次和整个系统可重用

1.4.2 管道与过滤器风格

□ 构件

- ✓ 将软件功能实现为一系列**处理步骤**，每个步骤封装在一个**过滤器构件**中

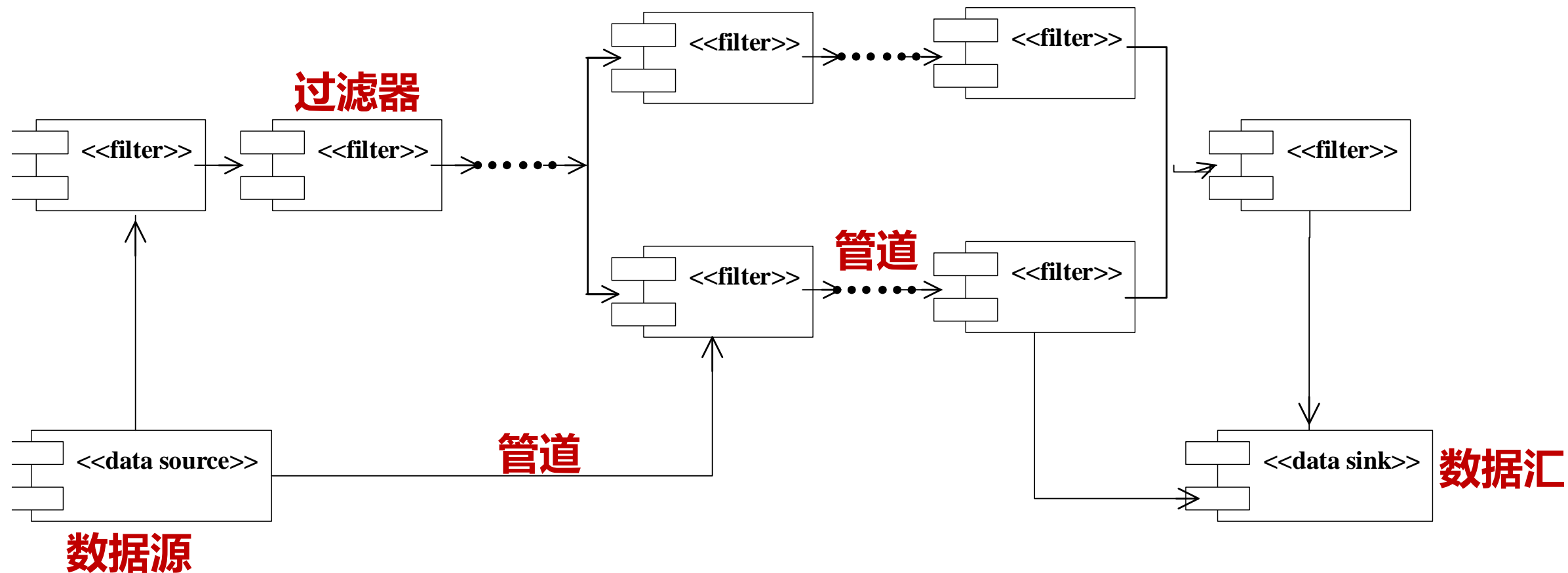
□ 连接

- ✓ 相邻过滤器间以**管道**连接，一个过滤器的输出数据借助管道流向后续过滤器，作为其输入数据

□ 数据

- ✓ 软件系统的**输入**由**数据源** (data source) 提供
- ✓ 软件最终**输出**由源自某个过滤器的管道流向**数据汇** (data sink)
- ✓ 典型数据源和数据汇包括**数据库、文件、其他软件系统、物理设备等**

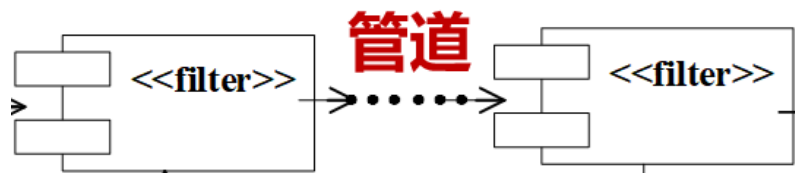
管道与过滤器风格的示例



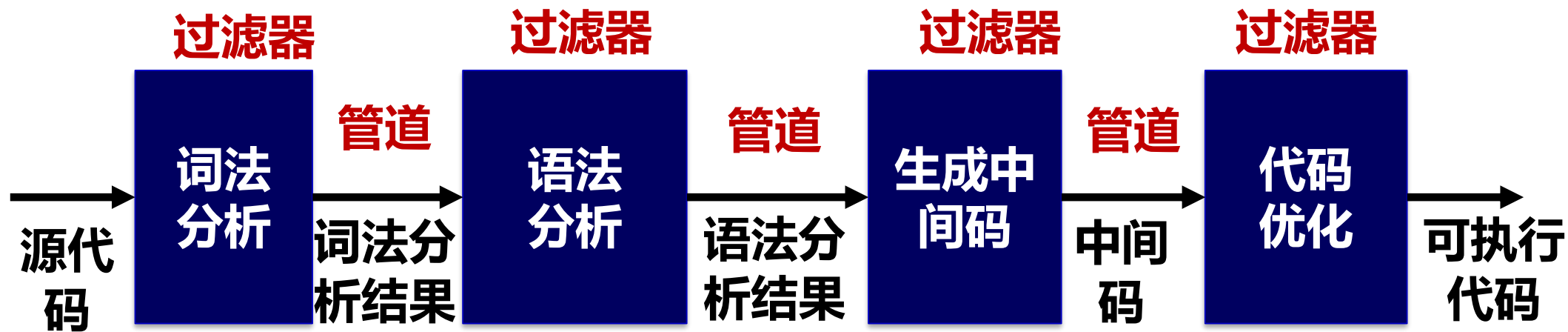
管道与过滤器风格的约束

□过滤器与管道之间的协作方式（连接）

- ✓ **过滤器主动方式**：过滤器通过循环，不断从管道提取输入数据，并将输出数据压入管道
- ✓ **过滤器被动方式**：过滤器被动地等待管道压入的数据
- ✓ **管道主动方式**：管道负责提取其源端过滤器的输出数据



示例：管道与过滤器风格



编译器采用的就是一个典型的管道/过滤器风格

管道与过滤器风格的特点

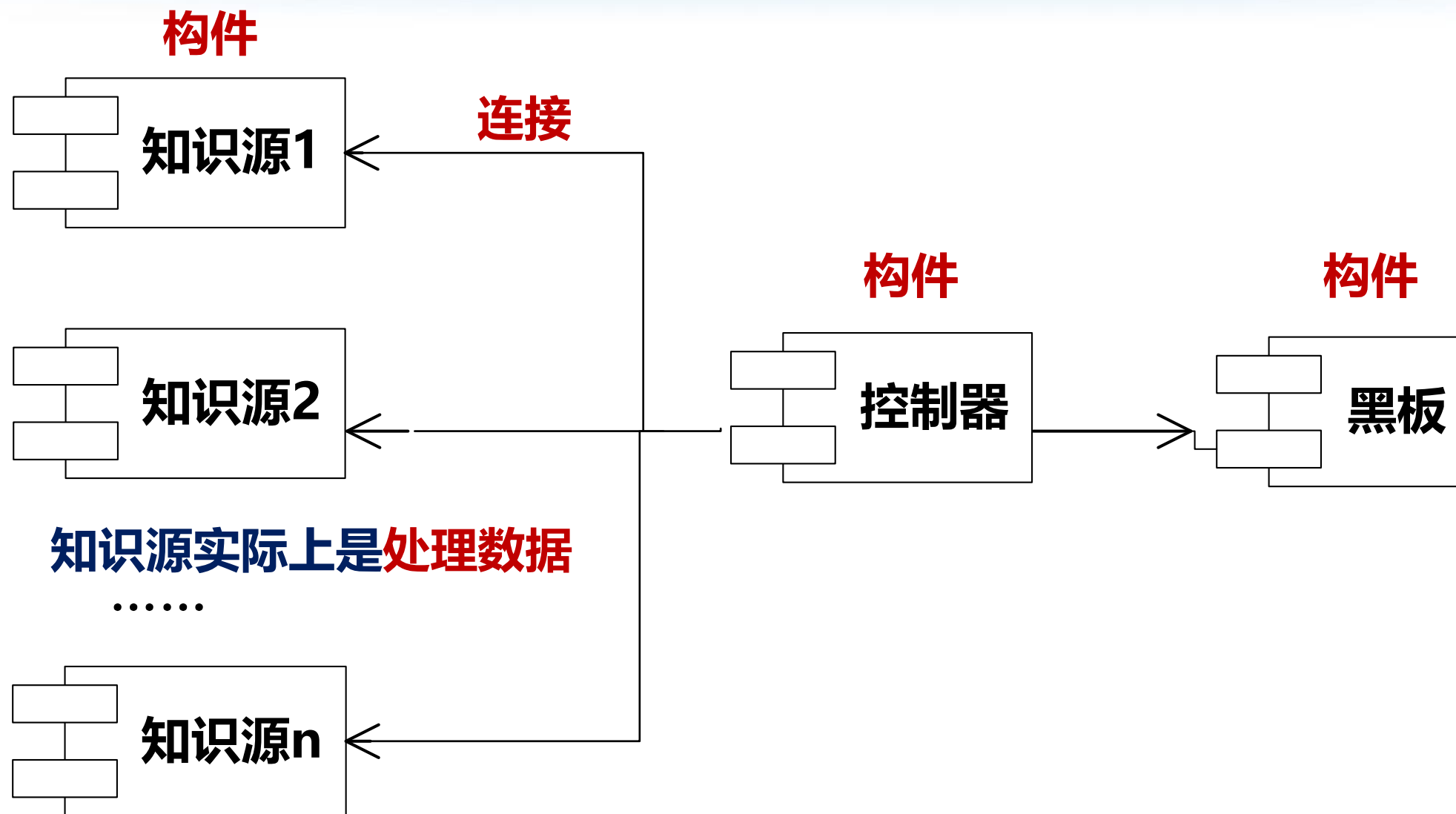
- 自然地解决具有**数据流特征**的软件需求
- 适合于**批处理**方式的软件系统，不适合交互式、事件驱动型应用
- 可独立地**更新、升级过滤器**来实现软件系统的扩展和进化

1.4.3 黑板风格

□将软件系统划分解为黑板、知识源和控制器三类构件

- ✓ **黑板**：负责**保存**问题求解过程中的**状态数据**，并提供这些数据的读写服务
- ✓ **知识源**：负责**部分问题求解**，并将此工作的**结果数据**写入黑板
- ✓ **控制器**：负责**监视**黑板中不断更新的**状态数据**，**安排**（多个）知识源的活动。

黑板风格示意图



黑板风格的约束

- **控制构件**通过观察**黑板**中的状态，决定调用哪些知识源参与下一步求解活动
- 被选中的**知识源**基于黑板中的状态数据将**问题求解**工作向前推进，并根据结果**更新黑板中的状态数据**
- 控制构件不断重复上述控制过程，及至问题求解获得满意的结果

黑板风格的特点

- 可灵活升级和更换知识源和控制构件

- 知识源的独立性和可重用性好

 - ✓知识源之间没有交互

- 软件系统具有较好的容错性和健壮性

 - ✓知识源的问题求解动作是探索性的，允许失败和纠错

例如：语音识别系统中，多个“知识源”可以并行地对语音信号分别进行预处理、特征提取、模式匹配等操作。操作的结果都会存储在黑板中。

1.4.4 MVC风格

□模型构件

- ✓负责存储业务数据并提供**业务逻辑处理功能**

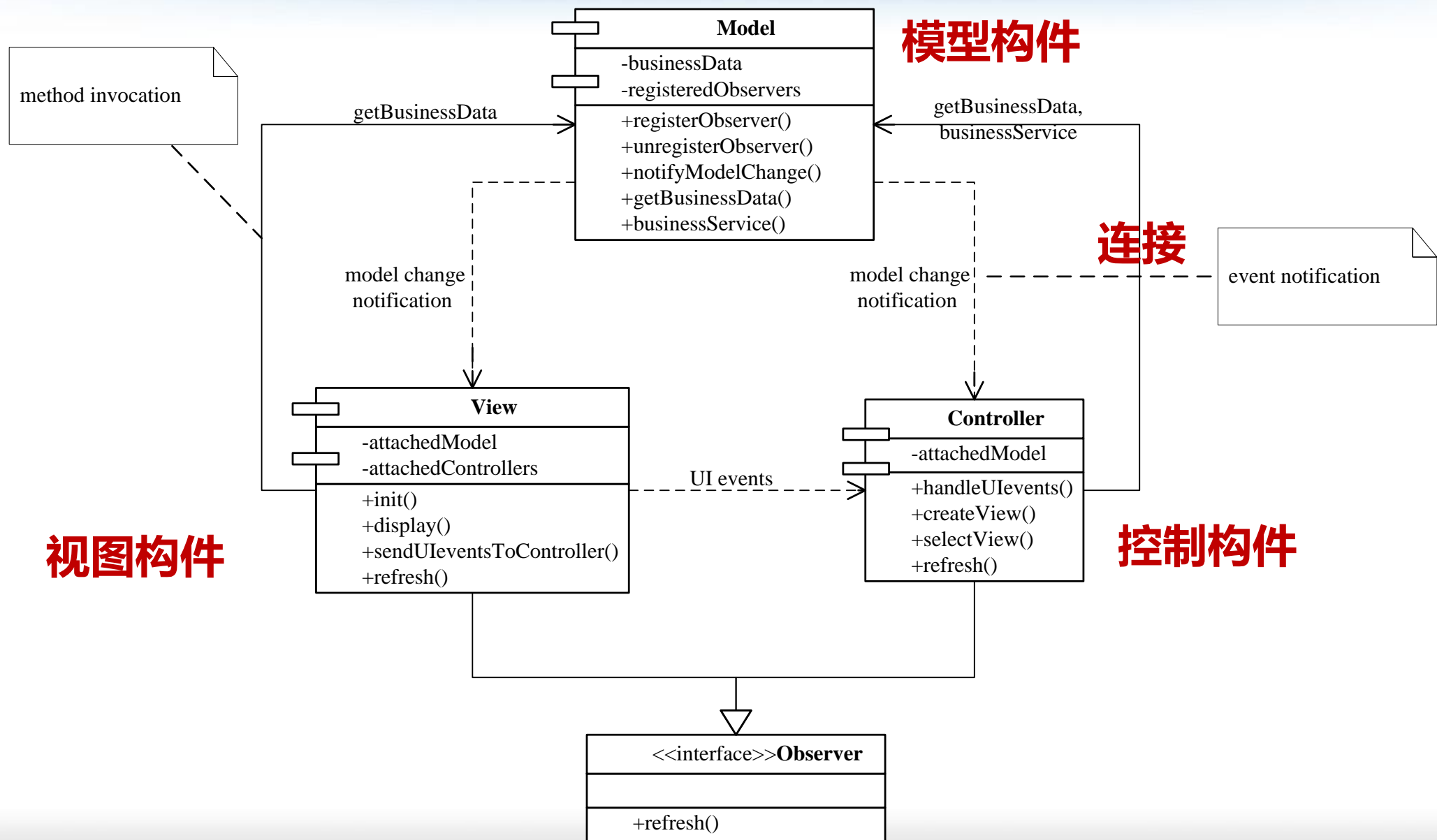
□视图构件

- ✓负责向用户**呈现**模型中的**数据**

□控制器

- ✓在接获模型的业务逻辑处理结果后，负责**选择适当的视图**作为软件系统对用户的界面动作的响应

MVC体系结构示意图



MVC风格的约束

□ 创建视图，视图对象从模型中**获取数据并呈现用户界面**

- ✓ 视图**接受界面动作**，将其转换为内部事件**传递给控制器**
- ✓ 所有视图在接获来自模型的业务数据变化通知后向模型**查询**新的数据，并据此**更新**视图

□ 控制器将用户界面事件转换为**业务逻辑处理功能的调用**

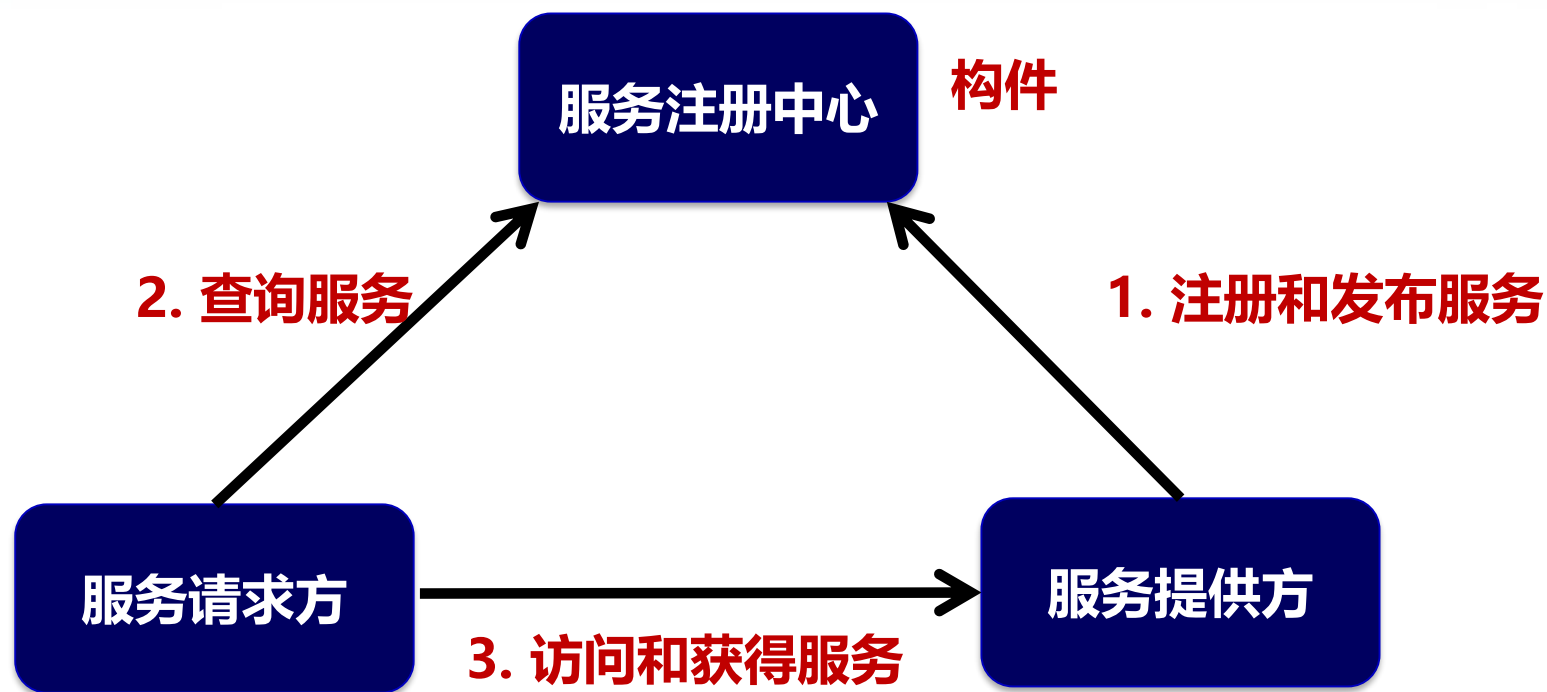
- ✓ 控制器根据模型的处理结果**创建新的视图**、选择其他视图或维持原有视图

□ 模型进行业务逻辑处理，将处理结果**回送给控制器**，必要时还需将业务数据变化事件**通知给所有视图**

1.4.5 SOA风格

- 将软件系统的软构件抽象为一个个的**服务**（Service），每个服务封装了特定的功能并提供了对外可访问的**接口**
- 任何一个服务既可以充当服务的**提供方**，接受其他服务的访问请求；也可充当服务的**请求方**，请求其他服务为其提供功能
- 任何服务需要向服务注册中心进行**注册登记**，描述其可提供的服务以及访问方式，才可对外提供服务

SOA风格示意图



在线零售商自动化订单处理系统：可以使用SOA来自动化其订单处理流程，包括订单接收、库存检查、支付处理和配送安排。每个步骤都由独立的服务完成。

SOA风格的特点

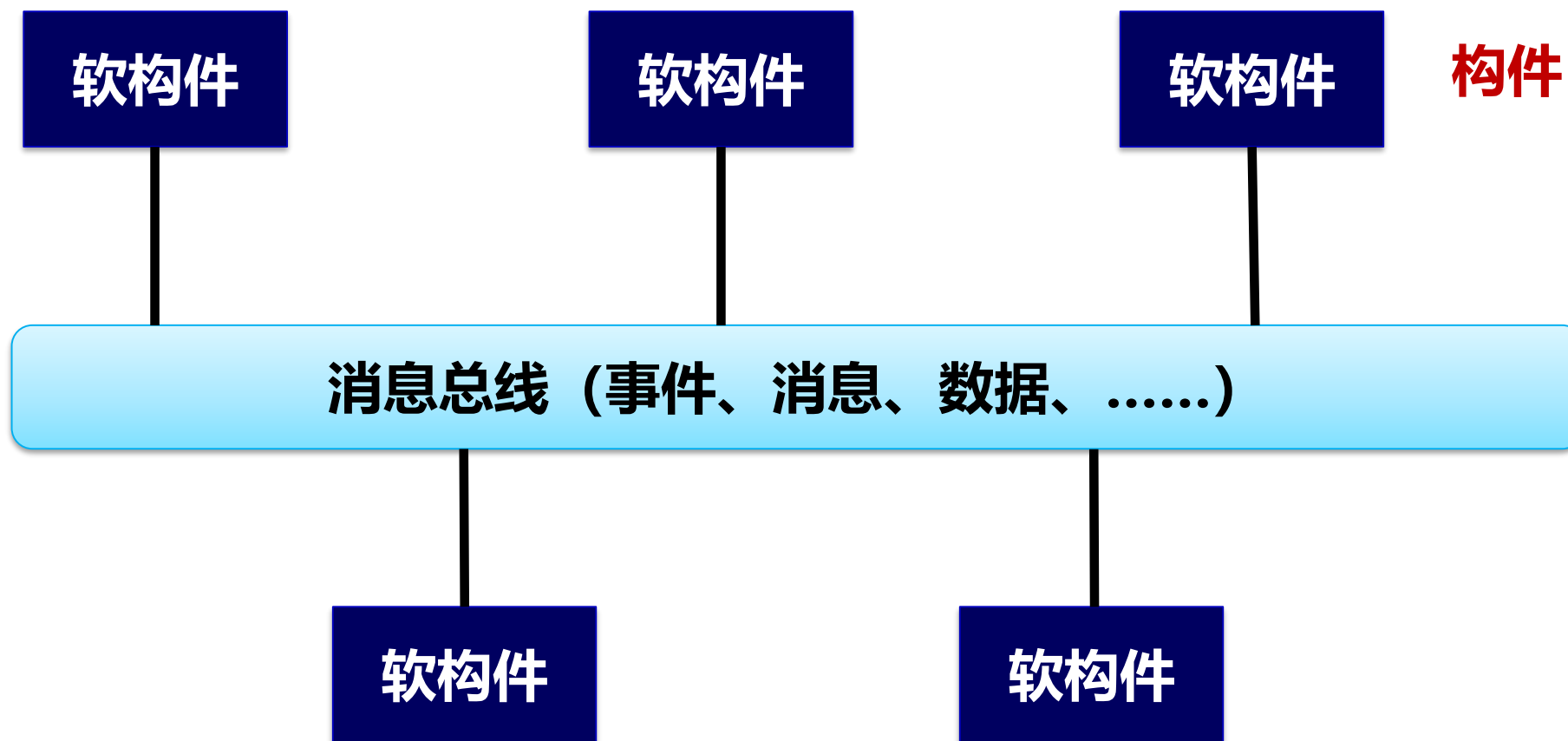
- 将服务提供方和服务请求方独立开来，因而支持服务间的松耦合定义
- 允许任何一个服务在运行过程中所扮演角色的动态调整，因而具有非常强的灵活性
- 提供了诸如UDDI、SOAP、WSDL等协议来支持服务的注册、描述和绑定等，因而可有效支持异构服务间的交互（服务可以基于不同编程语言、技术和框架）

1.4.6 消息总线风格

□包含了一组**软构件**和一条称为“**消息总线**”的**连接件**来连接各个软构件

- ✓消息总线成为软构件之间的**通信桥梁**，实现各个软构件之间的消息**发送、接收、转发、处理**等功能
- ✓每一个软构件通过接入总线，实现消息的发送和接收功能

消息总线风格示意图

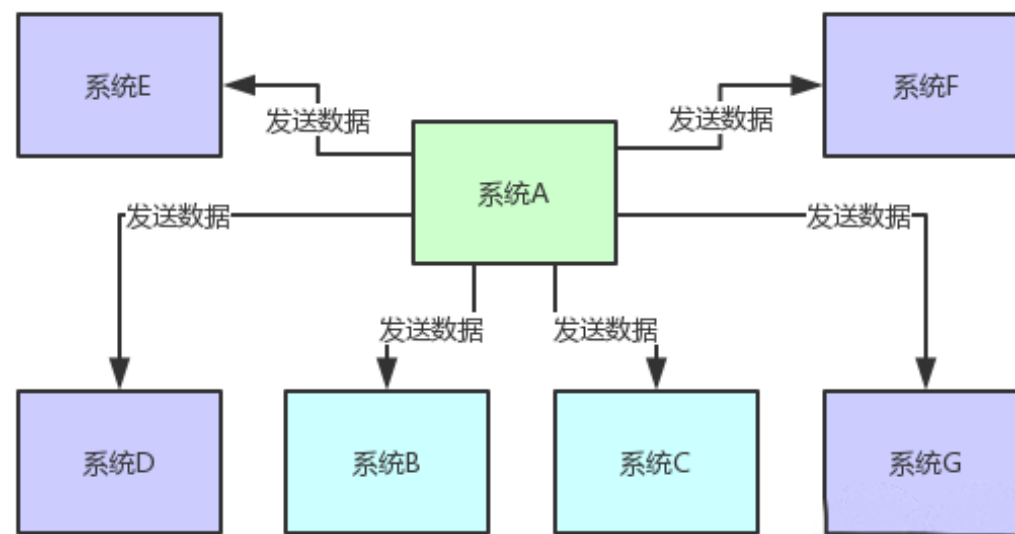
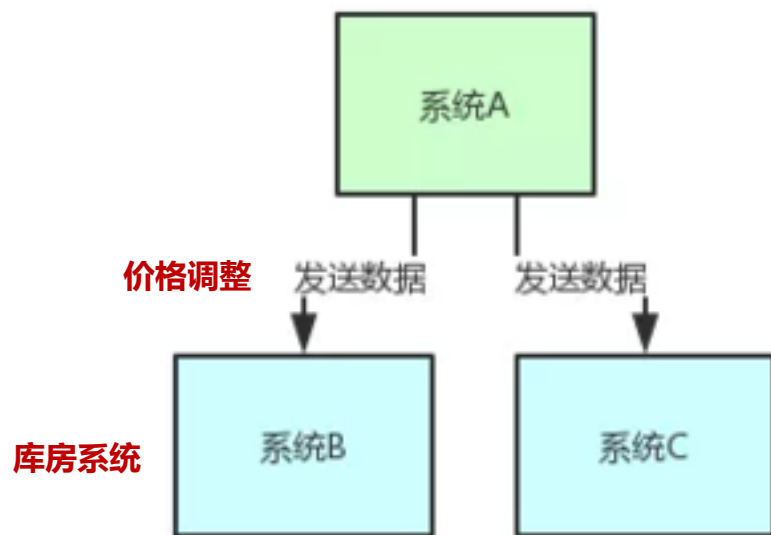


消息中间件

□消息总线风格已逐渐演变为**消息中间件**。

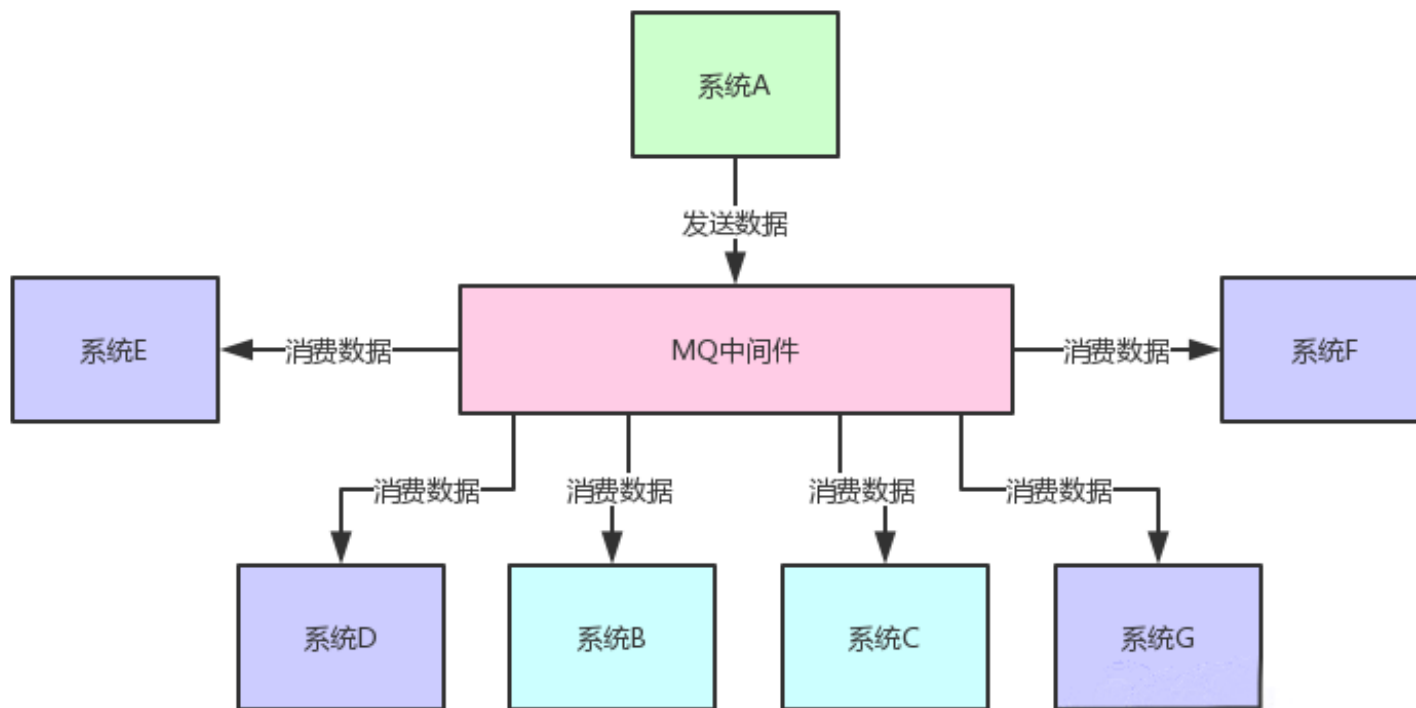
□消息中间件应用场景：**系统解耦、异步通讯、削峰**。

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



消息中间件

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



讨论：实践项目的软件体系结构

基于上述软件体系结构模式，思考你的实践项目应该采用什么样的软件体系结构模式？为什么？



内容

□何为软件体系结构

- ✓概念、组成元素、视图与模型
- ✓软件体系结构风格

□如何开展软件体系结构设计

- ✓价值、目标
- ✓体系结构设计过程

□软件体系结构设计结果及评审

- ✓文档模板、验证原则

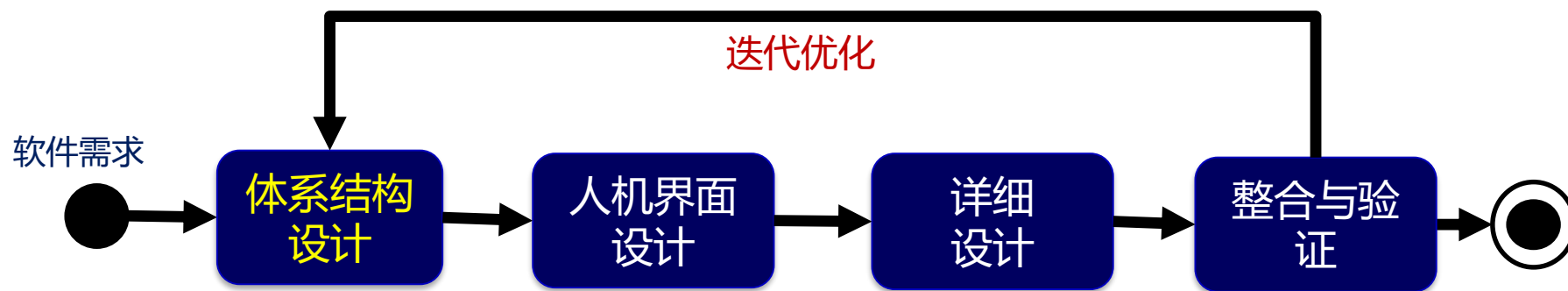


2.1 体系结构设计的重要性(1/2)

□ 体系结构设计对后续设计具有重要影响

✓ 体系结构设计为详细设计提供可操作指导

✓ 详细设计须遵循体系结构设计：如详细设计只能实现、不能更改体系结构中模块的接口和行为。



2.1 体系结构设计的重要性(2/2)

□对软件质量具有决定性影响

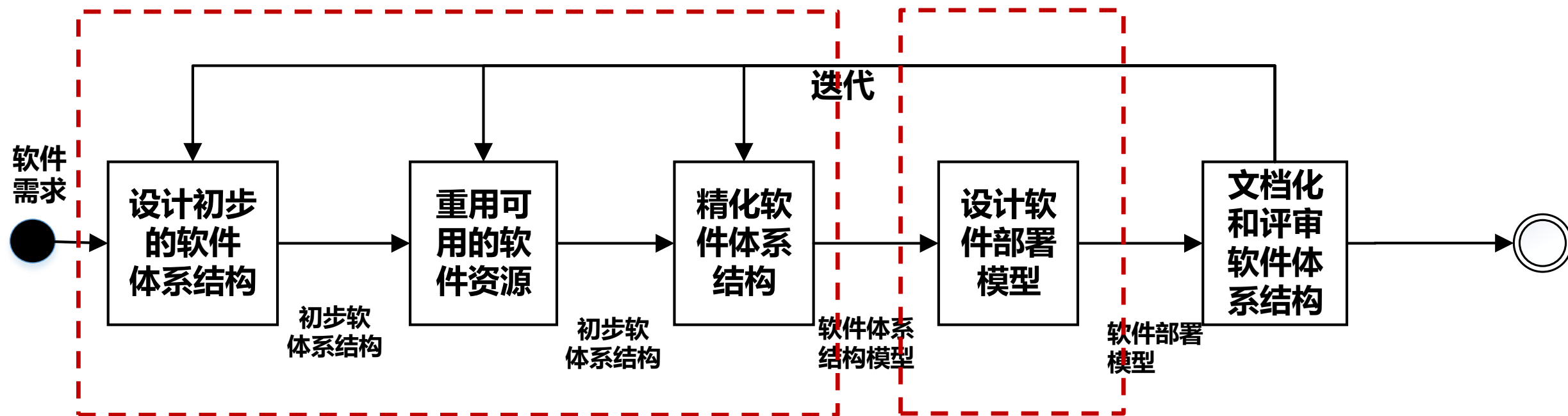
体系结构反映了软件开发的**早期关键设计决策**。这些决策对系统的性能、可靠性、安全性和可维护性有着深远的影响。

体系结构如同“骨架”，猴子的“骨架”决定了它永远无法飞起来。

例如：紧耦合的架构设计，使得各模块之间高度依赖，当某个模块发生故障或需要升级时，整个系统的运行都会受到影响。

2.2 软件体系结构设计过程

(1) 确定软件体系结构



(2) 设计部署模型

2.2.1 设计初步的软件体系结构

□任务

- ✓基于**关键软件需求**，参考已有软件**体系结构风格**，设计目标系统的**初始体系结构**，明确每个构件的职责以及协作关系。

□输出：

- ✓**初步的顶层架构**
- ✓用**包图**对顶层架构进行表示

如何辨识关键软件需求

□最能体现软件特色的**核心功能需求**

□对系统影响最大的**质量需求**

✓例如可靠性，可扩展性

□软件开发的**约束需求**

✓例如分布式计算或者集中式计算

□实现**难度较大、风险较高**的软件需求

根据关键需求选择体系结构风格

- 积累体系结构风格**清单**，熟悉每种风格的**特点和应用场所**
- 根据**关键需求**，选择当前项目**适用的风格**

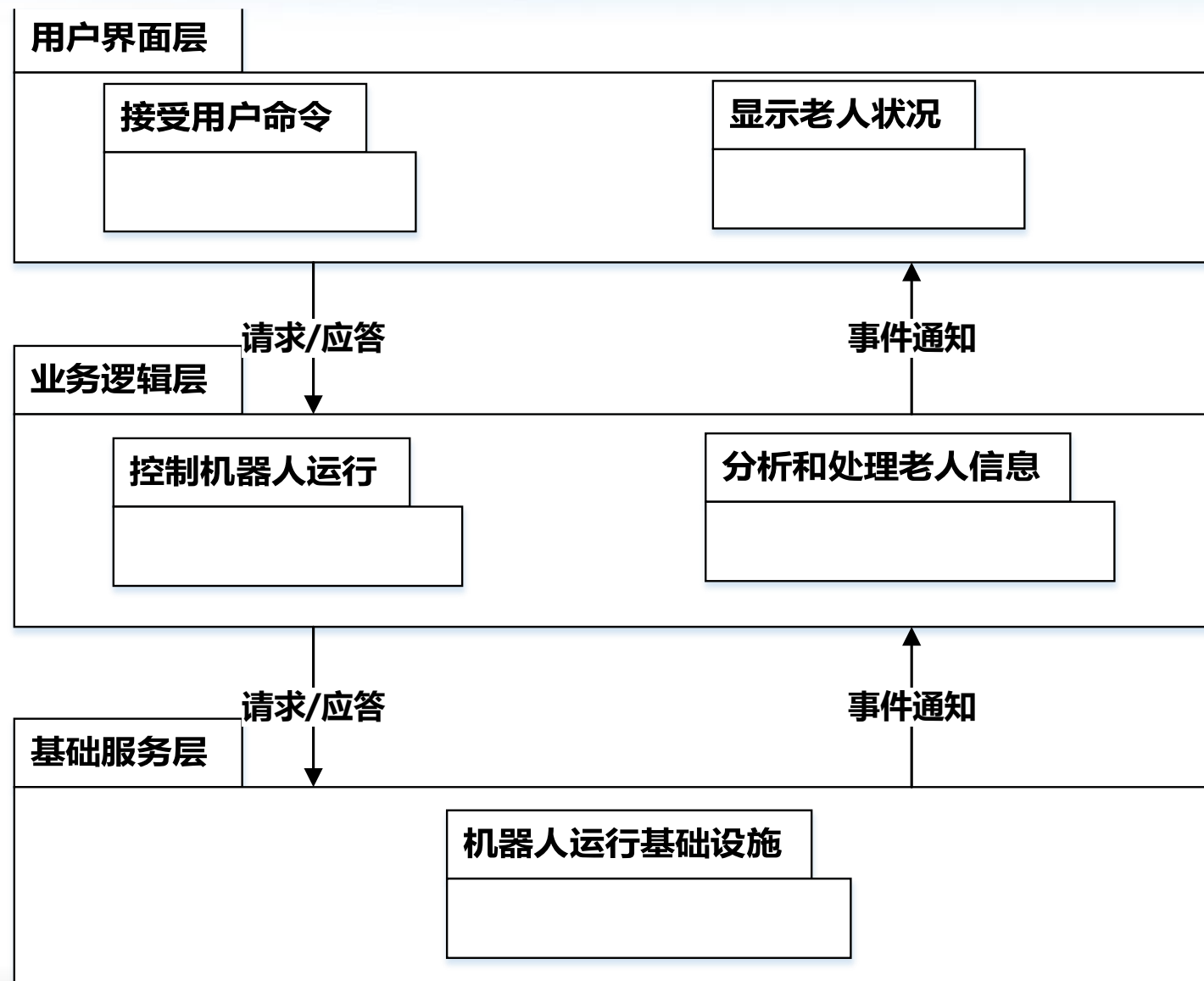
类别	特点	典型应用
管道/过滤器风格	数据驱动的分级处理，处理流程可灵活重组，过滤器可重用	数据驱动 的事务处理软件，如编译器、Web服务请求等
层次风格	分层抽象、层次间耦合度低、层次的功能可重用和可替换	绝大部分的应用软件
MVC风格	模型、处理和显示的职责明确，构件间的关系局部化，各个软构件可重用	单机软件系统，Web应用软件系统
SOA风格	以 服务 作为基本的构件，支持异构构件之间的互操作，服务的灵活重用和组装	部署和运行在云平台上的软件系统
消息总线风格	提供统一的消息总线，支持异构构件之间的消息传递和处理	异构构件之间 消息通信 密集型的软件系统

示例：初步体系结构设计(1)

□ “空巢老人看护软件” 的关键软件需求

- ✓ 监视老人状况、自主跟随老人、获取老人信息、检测异常状况、通知异常状况、远程控制机器人、视频/语音交互、提醒服务等八项功能性需求为核心软件需求。
- ✓ 可将性能、易用性、安全性、私密性、可靠性、可扩展性等质量需求作为关键需求
- ✓ 需要采用分布式运行和部署形式
 - 前端软件部署在Android手机上，后端软件要支持多种机器人的运行，以方便老人家属和医生的灵活和便捷使用。该开发约束将作为关键软件需求来指导软件体系结构的设计。

示例：初步体系结构设计(2)



建立逻辑视图步骤:

(1) 根据结构层次风格, 首先划分出层次;

(2) 建立不同层级间的协作关系;

(3) 将分析模型中的分析类依据其职责将其注入到不同层次中;

“空巢老人看护软件” 初步软件
体系结构设计

2.2.2 重用软件资源到体系结构

□搜索已有的粗粒度软件资源

- ✓ **可重用的软件开发包**，如函数库、类库、构件库等
- ✓ **互联网上的云服务**，为特定问题提供独立功能，如身份验证、图像识别、语音分析等等
- ✓ **遗留软件系统**，已存在的软件系统
- ✓ **开源软件**，提供针对特定功能的完整程序代码

□寻找可用于支持目标系统的软件制品

重用软件资源到体系结构

□可直接使用的软件资源

- ✓定义它们与当前系统间的交互接口
- ✓包括数据交换的格式、互操作协议等

□不可直接使用但具复用潜力的设计资源

- ✓采用接口重构、适配器等方法将其引入到当前体系结构中
- ✓接口重构是指，调整当前结构的调用接口，使之与复用资源提供的服务接口相匹配

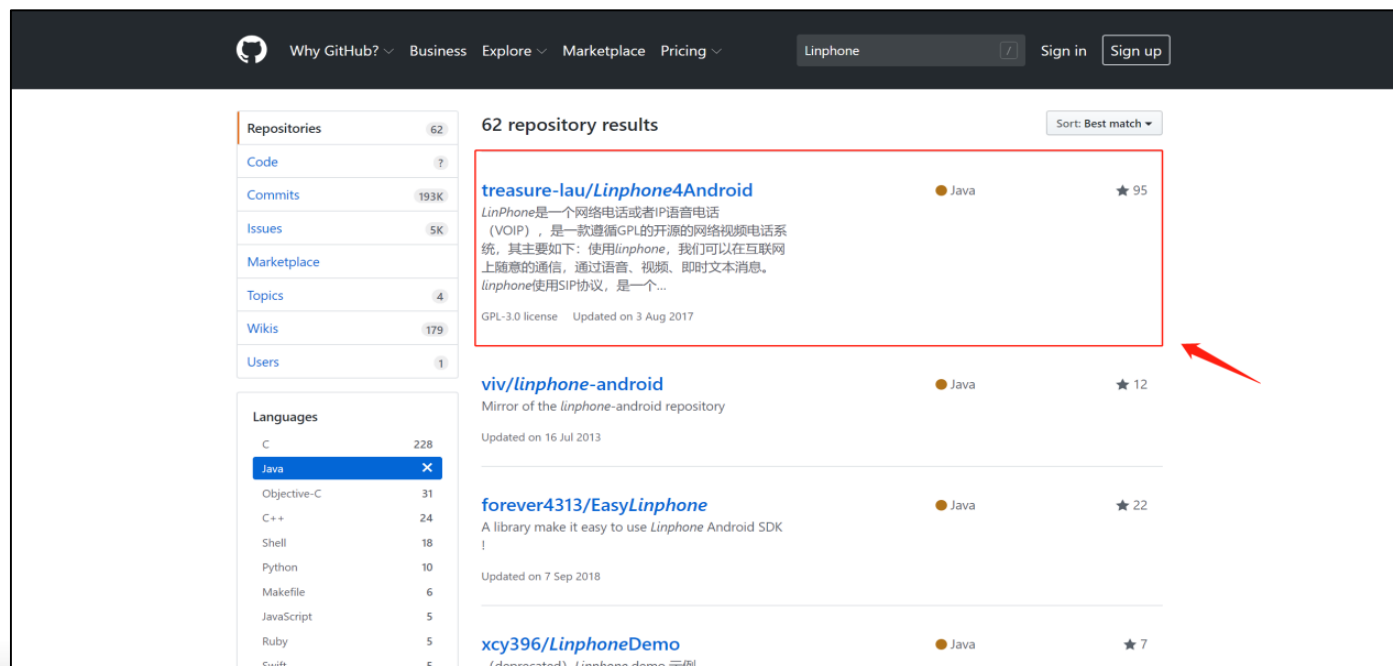
示例：搜寻和重用软件资产(1)

□软件系统需要实现“视频/语音双向交互”的功能

✓访问GitHub，在搜索框输入“**Linphone**”

✓点击进入，**查看开源软件项目说明，了解功能及使用方法**

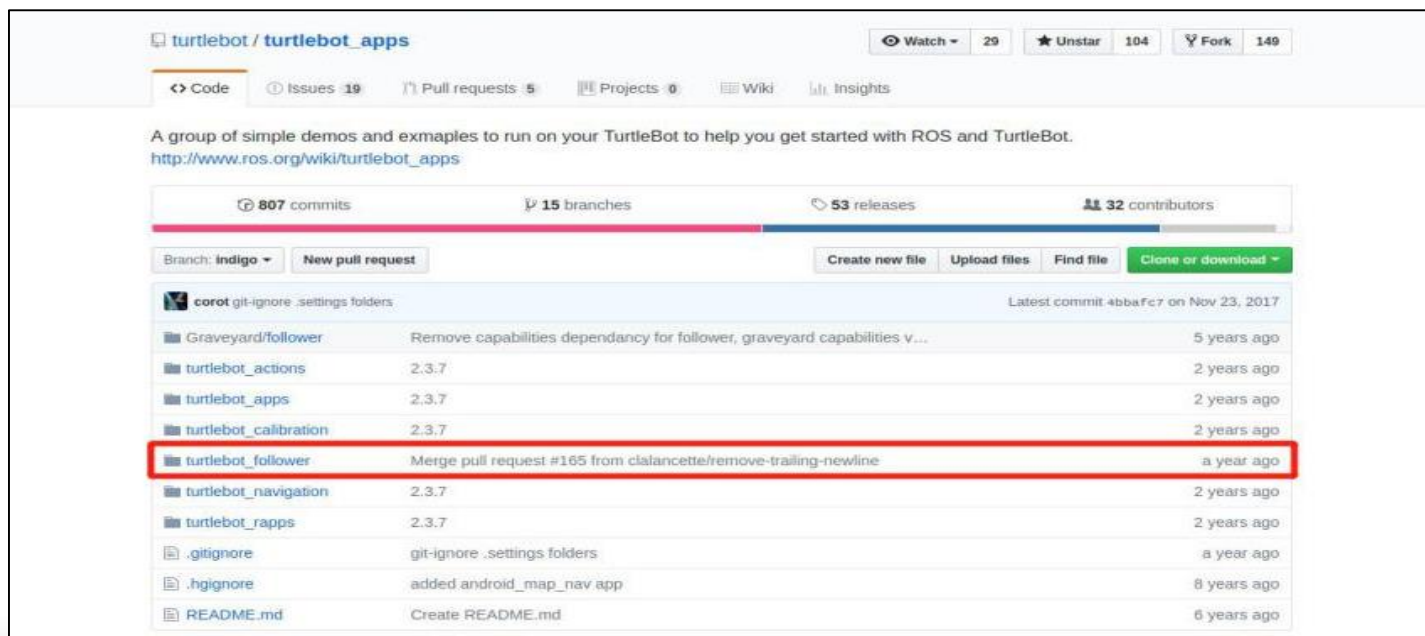
✓**下载和获取**“Linphone” 开源代码



示例：搜寻和重用软件资产(2)

□机器人的自主跟随功能

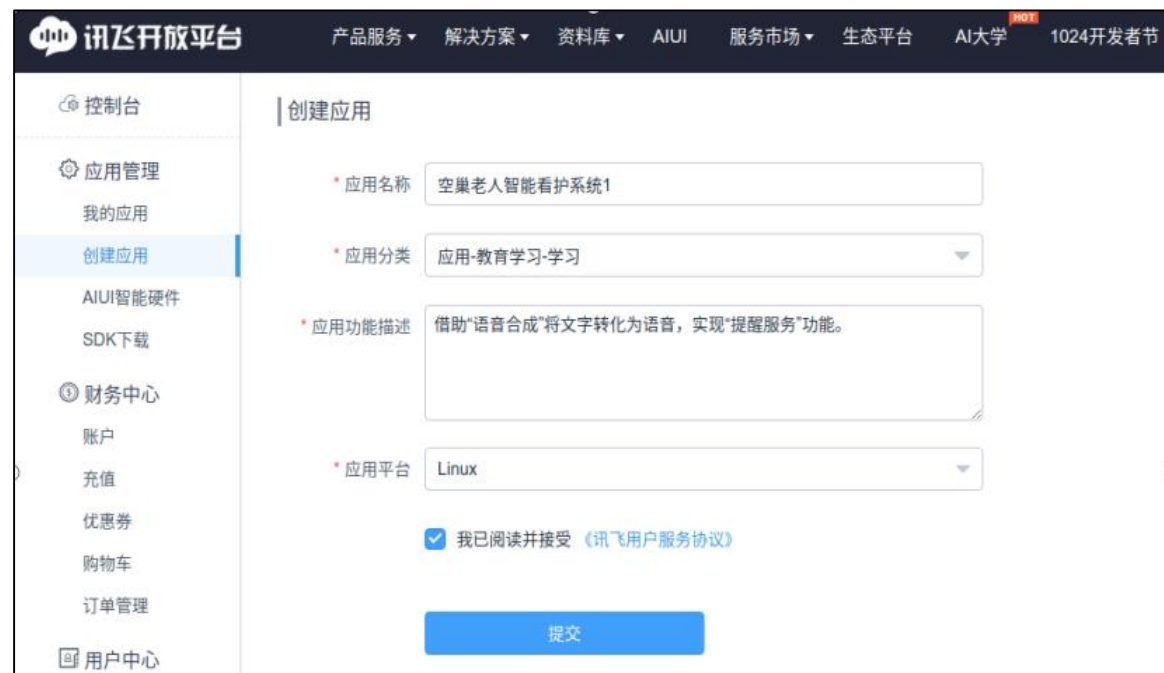
- ✓ 搜寻和重用ROS代码 “turtlebot_follower” 来实现
- ✓ 在Github搜索 “turtlebot app” , **理解其功能和使用**
- ✓ 点击 “turtlebot_follower” , 阅读和下载其开源代码



示例：搜寻和重用软件资产(3)

□搜寻和重用“**离线语音合成**”软件包OLVTI-SDK实现文字到语音转换

- ✓讯飞开放平台 (www.xfyun.cn/) 提供“离线语音转换和合成”功能
- ✓平台提供软件开发包SDK
- ✓**理解其功能和使用**



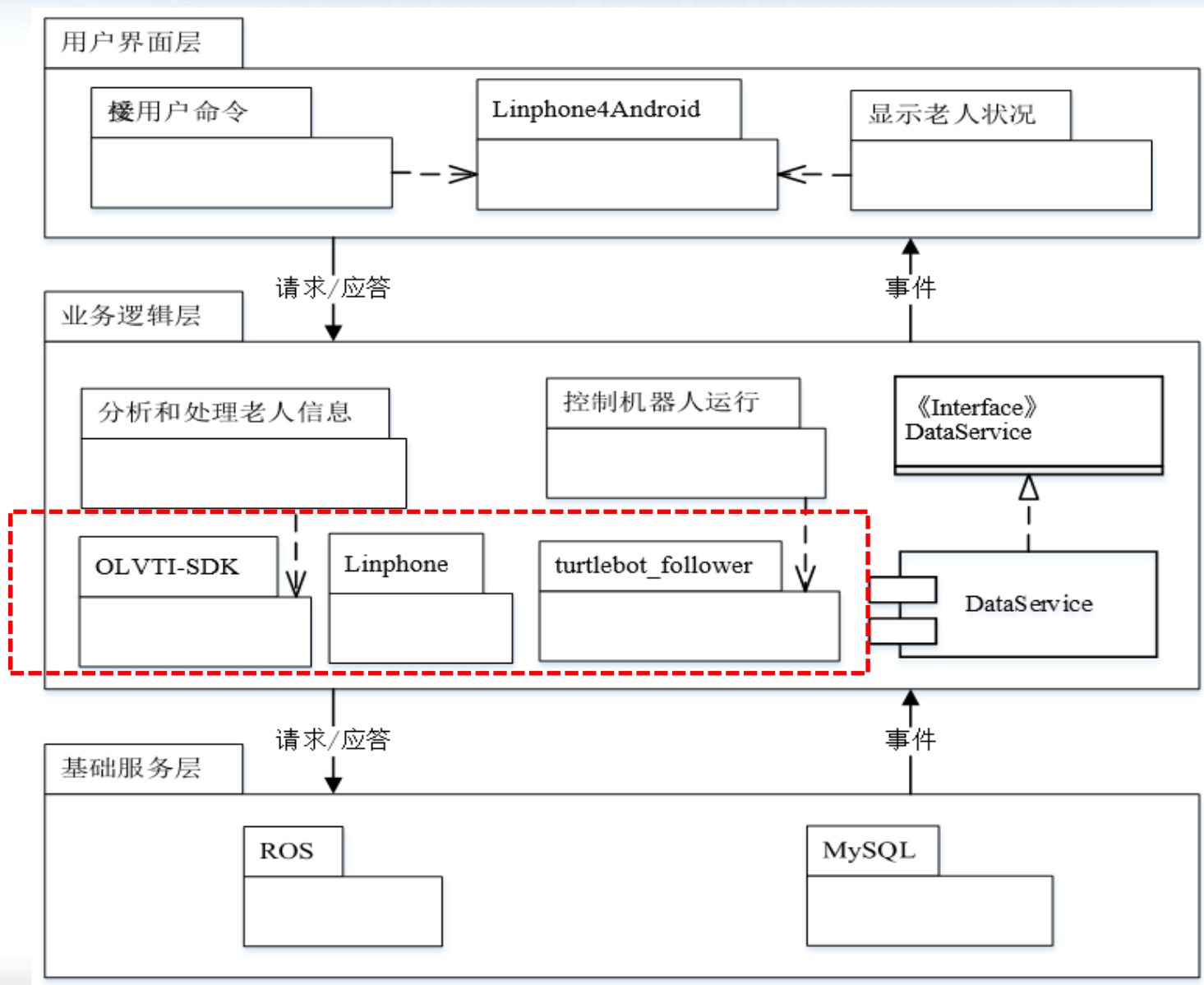
The screenshot shows the '讯飞开放平台' (Xfyun Open Platform) interface. The top navigation bar includes links for '产品服务', '解决方案', '资料库', 'AIUI', '服务市场', '生态平台', 'AI大学', and '1024开发者节'. The left sidebar contains a menu with '控制台', '应用管理', '我的应用', '创建应用' (highlighted), 'AIUI智能硬件', 'SDK下载', '财务中心', '账户', '充值', '优惠券', '购物车', '订单管理', and '用户中心'. The main content area is titled '创建应用' and contains the following fields:

- * 应用名称: 空巢老人智能看护系统1
- * 应用分类: 应用-教育学习-学习
- * 应用功能描述: 借助“语音合成”将文字转化为语音, 实现“提醒服务”功能。
- * 应用平台: Linux

At the bottom, there is a checkbox labeled '我已阅读并接受《讯飞用户服务协议》' which is checked, and a blue '提交' (Submit) button.

示例：“空巢老人看护软件”的开源软件重用

可重用软件资产



2.2.3 精化软件体系结构

- 目标：将初步软件体系结构**精化**为粒度适中的设计元素。
主要包括下项子任务：
 - (1) 确定**公共基础设施及服务**
 - (2) 确定**设计元素**

(1) 确定公共基础设施及服务

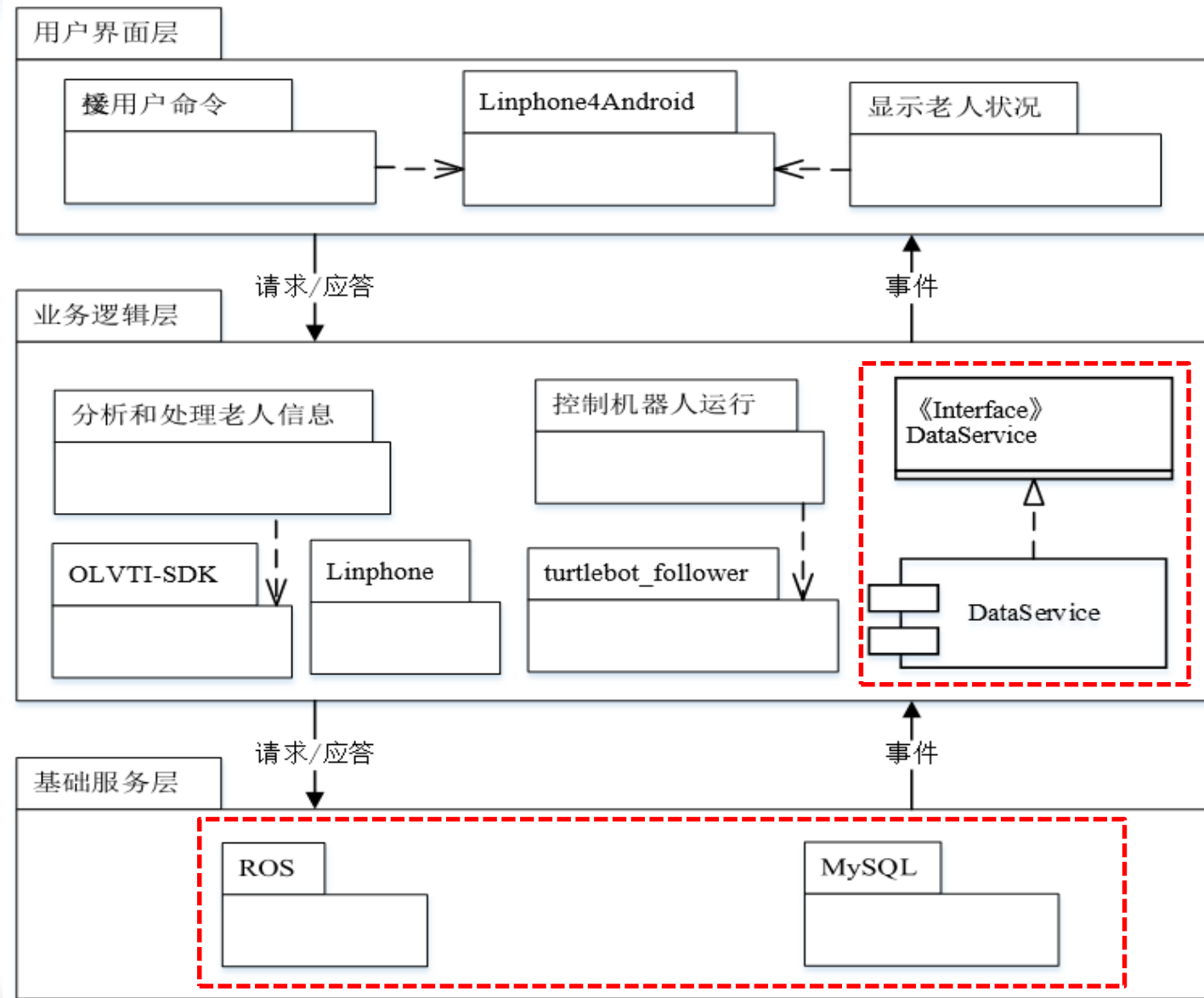
□确定公共基础设施

- ✓包括最底层的**操作系统**，到稍高层次的**软件中间件、软件开发框架**，或者**数据库管理系统**等

□设计基础服务

- ✓基础服务不针对用户，却是上层多应用所必须的，如**数据持久服务、隐私保护服务、安全控制服务、消息通讯服务**等
- ✓基础服务应注重**稳定性**，即使软件需求发生了变化，基础服务仍可为其提供服务

示例：“空巢老人看护软件”的基础设施



数据持久服务, 提供应用数据增删改查

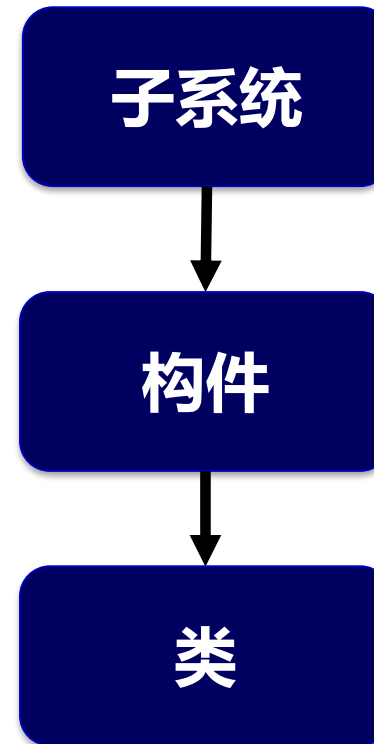
基础服务

(2) 确立设计元素

□ 以实现**所有软件需求**为目标，依据需求分析模型（类图，交互图等），将**分析类**转为**设计元素**，确定其**职责及相互关系**

□ 设计元素

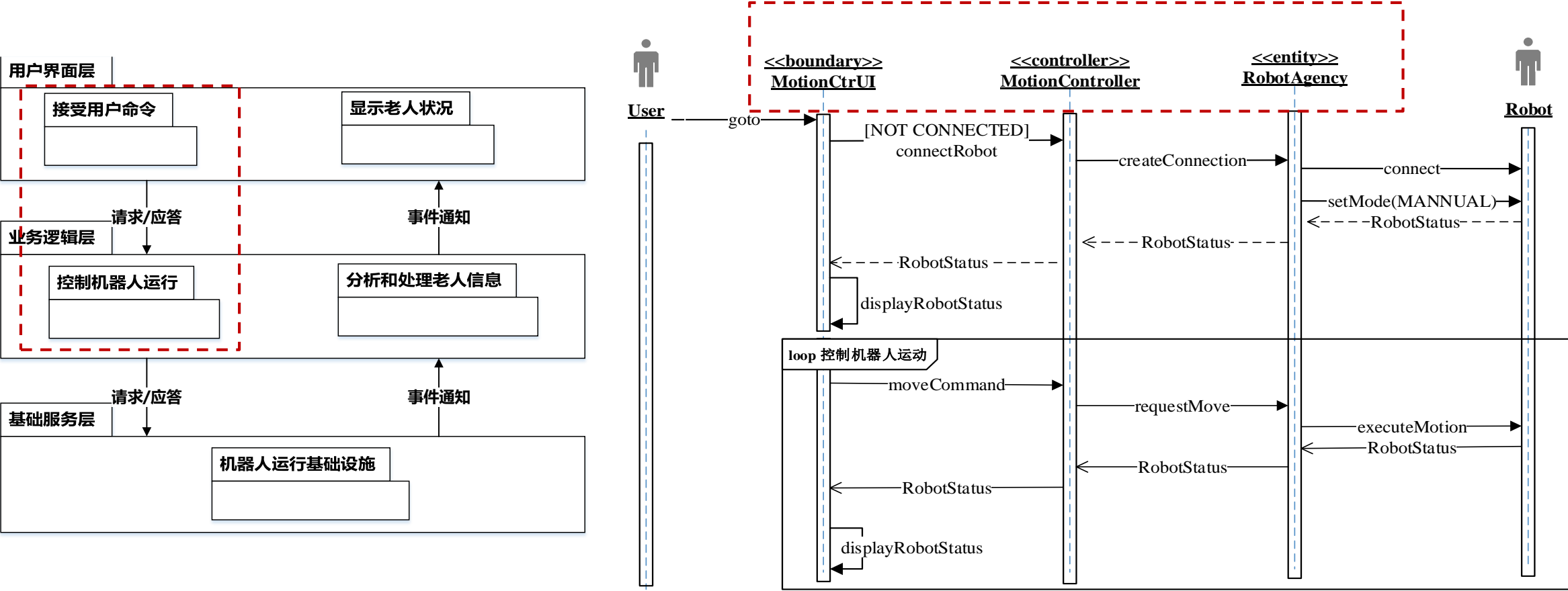
- ✓ 三类：**子系统** **构件** **设计类**
- ✓ 只确定设计元素的接口及职责，无须细化



确定子系统/构件 (1/2)

步骤1：依据分析类的职责或类别，将其注入到初步体系结构对应包中

例如：针对“控制机器人运行”进行细化，这部分对应系统用例“远程控制机器人”



远程控制机器人用例顺序图

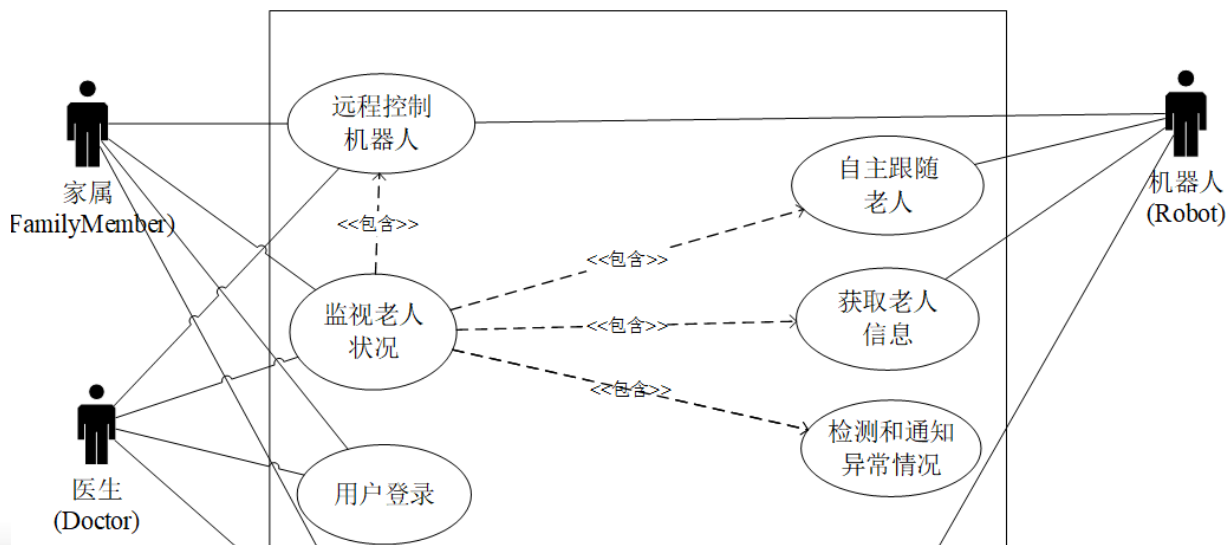
设计实例：“空巢老人智能看护系统”

根据分析类，我们可以抽取出3个设计元素：MotionCtrUI, MotionCtroller, RobotAgency，并分别将其注入到用户界面层和业务逻辑层的

包“接收用户命令”： MotionCtrUI

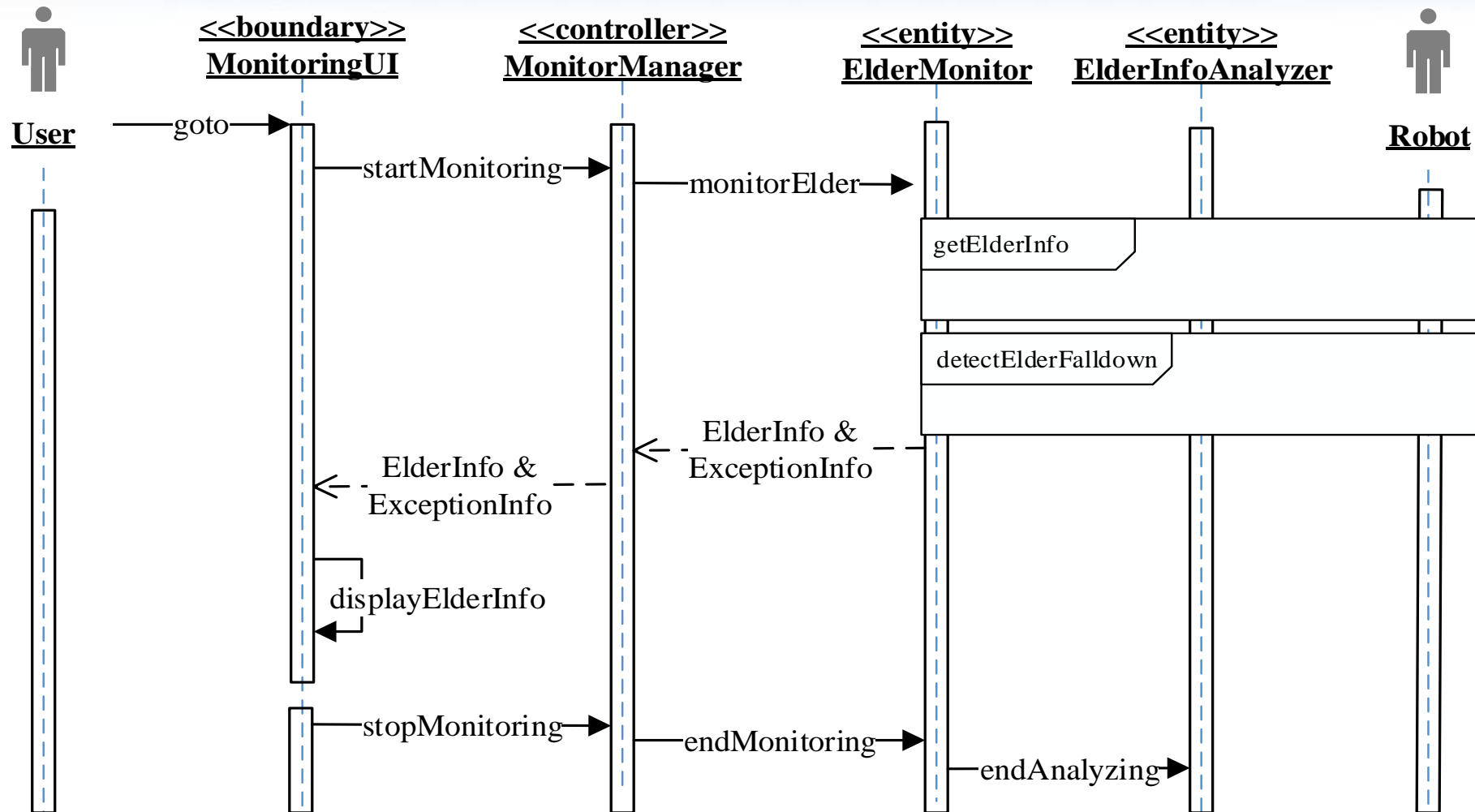
包“控制机器人运行”： MotionCtroller, RobotAgency

对分析处理老人信息，其涉及的用例包括：监视老人状况、自主跟随老人、获取老人信息、检测异常状况等。对每个用例，如法炮制，将得到的所有构件分别注入到每层包中。



注意：这几个用例之间是include关系

设计实例：“空巢老人智能看护系统”



监视老人状况用例顺序图

确定子系统/构件(2/2)

步骤2：依据以下原则，对设计元素进行合并或分组，建立子系统

- ✓ **用例相关性**：根据用例在**业务或功能**的相关/相似性分组，**同一类用例**设计为一个子系统/构件；

例如：浏览图书和搜索图书可以归为一类

- ✓ **控制类相似性**：按**职责**将相关或相似的控制类分组或合并，每组对应一个子系统/构件；

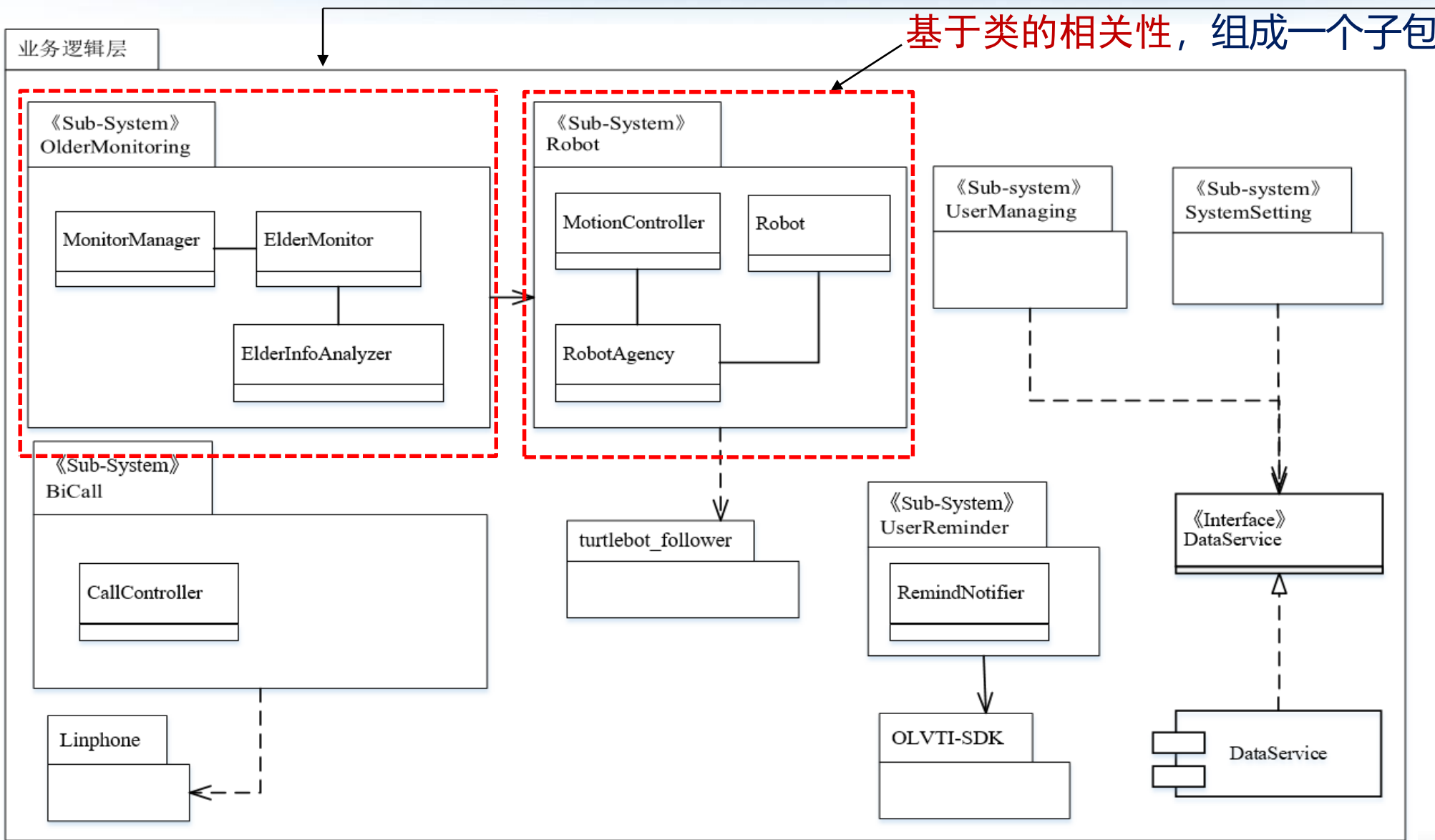
例如：运动控制子系统包括

速度控制构件：负责生成和管理机器人的线速度和角速度指令。

底盘驱动构件：接收速度指令，并将其转换为电机驱动信号，驱动机器人运动。

- ✓ **实体类相关性**：按**职责**将相关或相似了的实体类分组或合并，每组对应于一个子系统/构件；

设计实例：“空巢老人智能看护系统”



监视老人状况、自主跟随老人、获取老人信息、检测异常状况，4个用例相似，因此合并为一个子包

这样建立的体系结构，内聚性才更强

思考和讨论

- **子系统和构件**都可对外提供服务和功能，二者有何区别和联系？
- 什么情况下应该**设计为子系统**，在什么情况下应该**设计为软构件**？

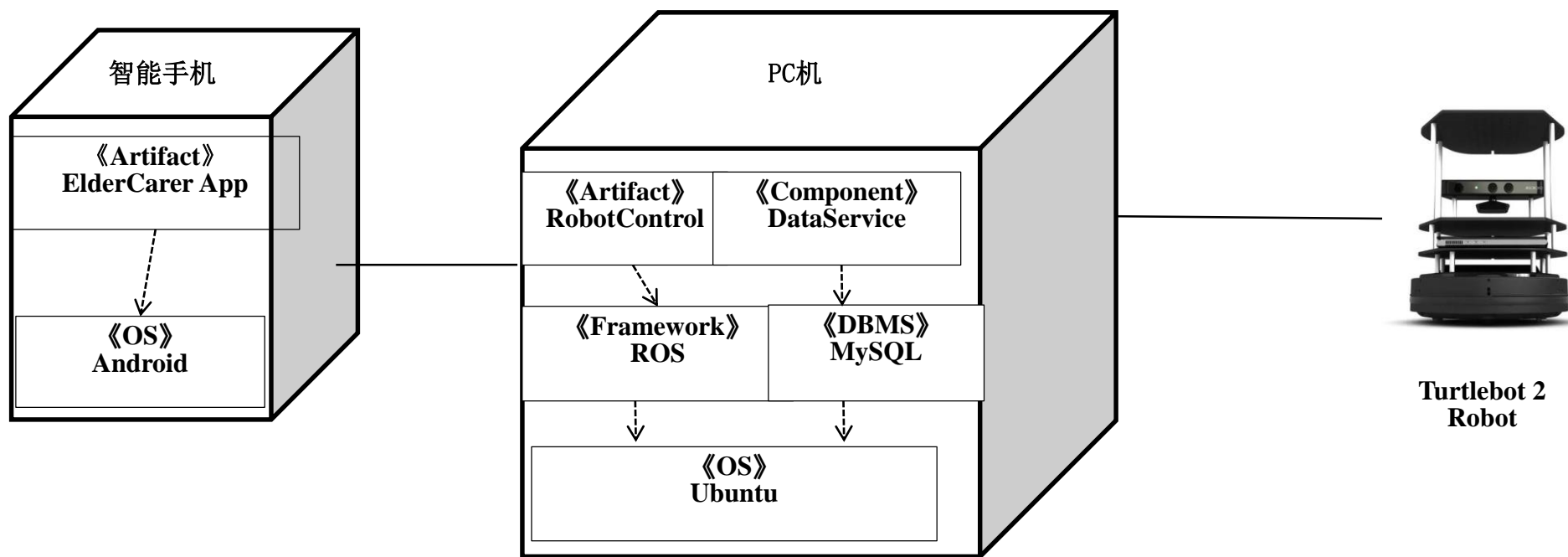
- ✓ 子系统通常包含多个构件，可以提供比单个构件更复杂、更完整的功能
- ✓ 构件的内聚度高于子系统
- ✓ 子系统的粒度更大，可直接运行，不考虑复用性；而构件则粒度更细，但可复用性更强，可以通过复用构建其它更复杂的构件



2.2.4 设计部署模型

□设计软件系统的物理部署模型

✓刻画软件系统的各个子系统、构件如何部署到计算节点上



内容

□何为软件体系结构

- ✓概念、组成元素、视图与模型
- ✓软件体系结构风格

□如何开展软件体系结构设计

- ✓价值、目标
- ✓体系结构设计过程

□软件体系结构设计结果及评审

- ✓文档模板、验证原则



软件体系结构设计的输出

□软件体系结构设计**模型**

- ✓用UML包图、部署图、构件图等描述

□软件体系结构设计**文档**

- ✓体系结构设计规格说明书

软件体系结构设计文档示例

1. 文档概述
2. 系统概述
3. 设计目标和原则
4. 设计约束和现实限制
5. 逻辑 viewpoints 的体系结构设计
6. 部署 viewpoints 的体系结构设计
7. 开发 viewpoints 的体系结构设计
8. 运行 viewpoints 的体系结构设计

小结

□软件体系结构设计的特殊性

- ✓具有宏观、全局、层次、战略、多视点、关键性等特点
- ✓逻辑视点、物理视点等，可用包图、部署图、构件图等来表示

□软件体系结构设计的重要性

- ✓起到承上启下的作用，详细设计的基础和前提

□软件体系结构的风格

- ✓管道、层次、MVC、黑板等等，针对不同的软件需求及特点

□软件体系结构设计的过程、策略和成果

- ✓考虑软件关键需求、利用已有软件资产、关注软件质量
- ✓产生软件体系结构设计模型，撰写设计文档

□任务：开源软件的体系结构设计

□方法

- ✓针对开源软件新增加的软件需求，考虑软件体系结构风格，搜寻可用的软件资源（包括开源软件），扩展和优化原有的软件体系结构，引入新的设计元素或者重新设计软件体系结构

□要求

- ✓针对开源软件及其新构思的软件需求，在原有软件体系结构的基础上，调整、优化或重新设计开源软件的体系结构

□结果：软件体系结构模型（至少包括逻辑视点和物理视点的体系结构模型），软件体系结构设计文档

□任务：软件体系结构设计

□方法

- ✓针对关键软件需求，考虑软件体系结构风格，搜寻可用的软件资源（包括开源软件），设计初步的软件体系结构；对软件体系结构进行精化设计，进一步确定其软构件、子系统和设计类等设计元素，以满足所有软件需求；给出软件体系结构的部署模型

□要求

- ✓针对构思的软件需求，开展软件体系结构设计，产生软件体系结构设计模型

□结果：软件体系结构模型（至少包括逻辑视图和物理视图的体系结构模型），软件体系结构设计文档