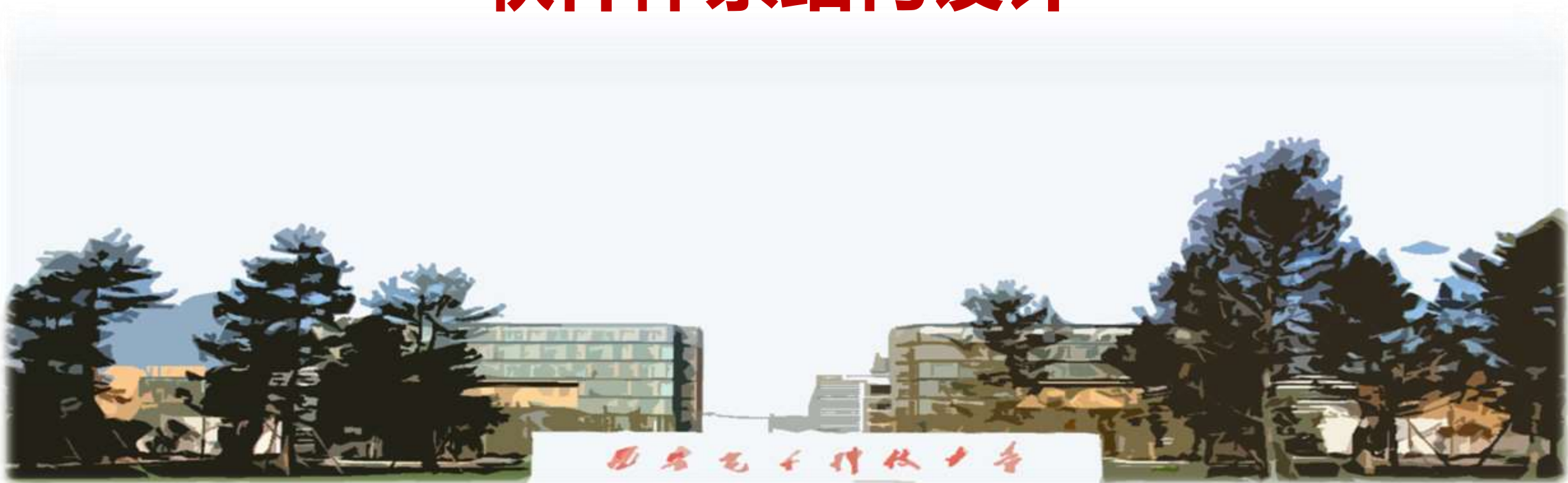




西安电子科技大学
XIDIAN UNIVERSITY

软件体系结构设计





内容

1. 何为软件体系结构

- ✓ 概念、视图、模型及UML表示
- ✓ 软件体系结构风格

2. 软件体系结构设计

- ✓ 任务、目标、要求和原则
- ✓ 体系结构设计过程

3. 文档化和评审软件体系结构设计

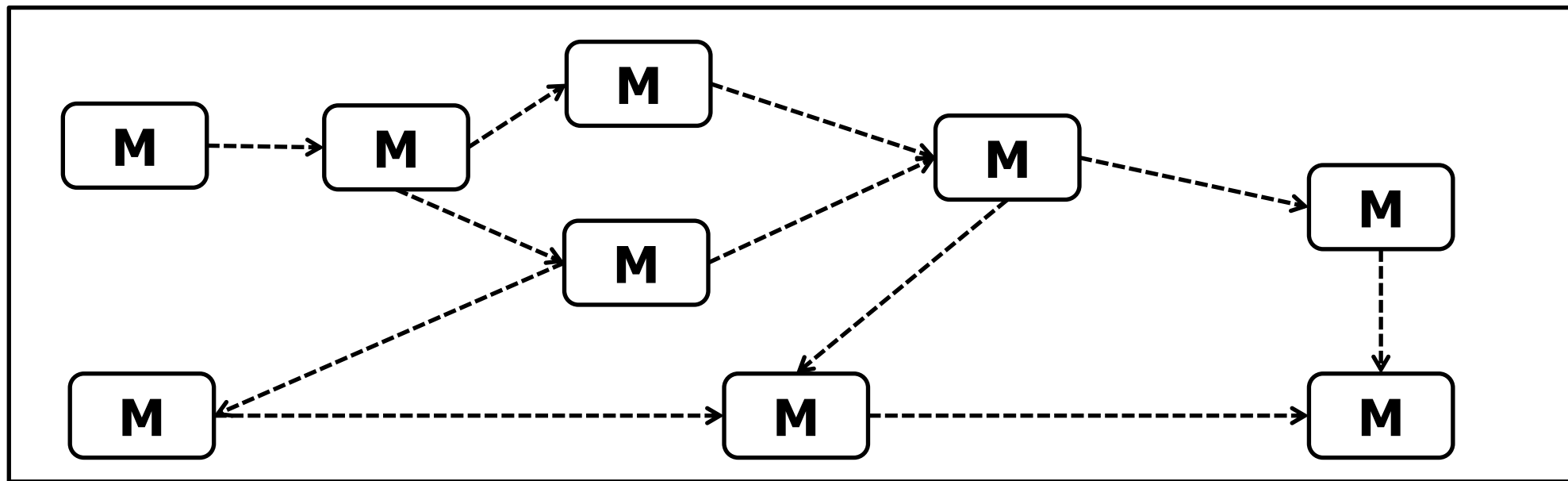
- ✓ 文档模板、验证原则



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**高层抽象**角度刻画组成软件系统的**设计元素**及它们之间的**逻辑关联**



软件体系结构如何体现了高层和抽象的特征？这里的设计元素是什么？



1.2 软件体系结构的设计元素

□ 构件(Component)

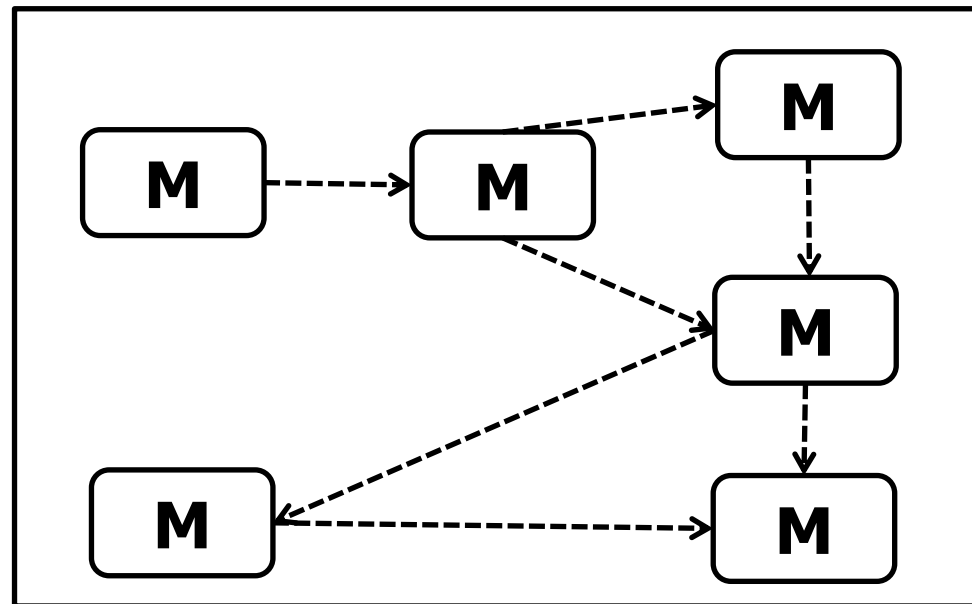
✓ 构成体系结构的基本功能部件

□ 连接件(Connector)

✓ 组件之间的连接和交互关系

□ 约束(Constraint)

✓ 组件中的元素应满足的条件以及组件经由连接件组装成更大模块时应满足的条件



1.2.1 构件

□何为构件

- ✓ 软件系统中的**物理模块**，具有特定的功能和精确定义的对对外接口，外界可通过接口来访问它

□特点

- ✓ **可分离**：一个或数个可独立部署执行码文件
- ✓ **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

□.dll文件

 ado	2019/8/4 17:43	文件夹	
 en-GB	2019/8/4 17:38	文件夹	
 en-US	2019/3/19 19:41	文件夹	
 msadc	2019/3/19 19:41	文件夹	
 Ole DB	2019/8/4 17:50	文件夹	
 zh-CN	2019/3/19 19:41	文件夹	
 wab32.dll	2019/3/19 12:45	应用程序扩展	853 KB
 wab32res.dll	2019/3/19 12:45	应用程序扩展	942 KB

□.jar文件

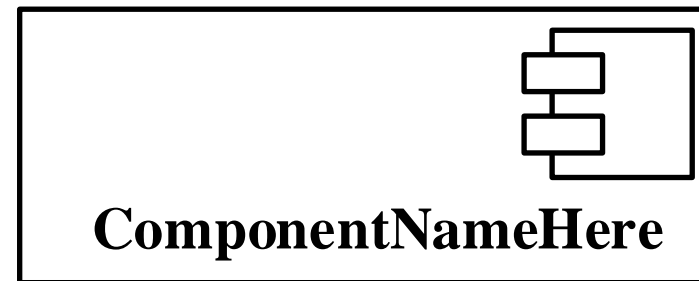
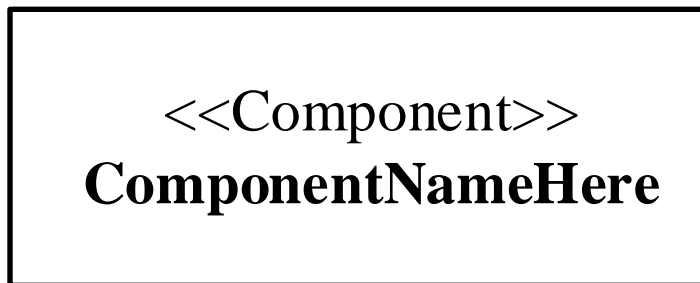
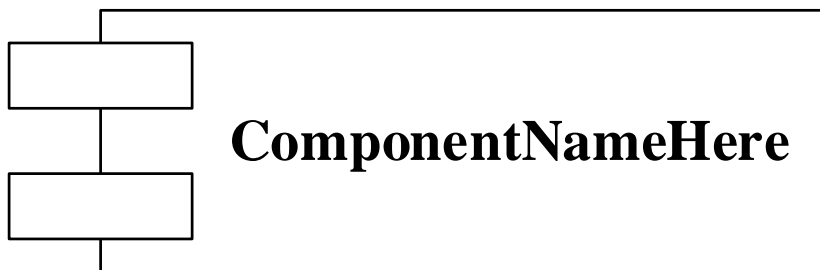
- 构件不是源程序，而是可运行的二进制代码
- 构件是客观物理存在的（即有实际的文件），而非是逻辑存在的
- 构件是可访问的，以获取其功能和服务
- 构件应该是粗粒度的，而非细粒度的

示例：软构件

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
AXEParser.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
中的软构件**

构件的UML表示



可以用不同的UML图符来表示软构件



1.2.2 连接子

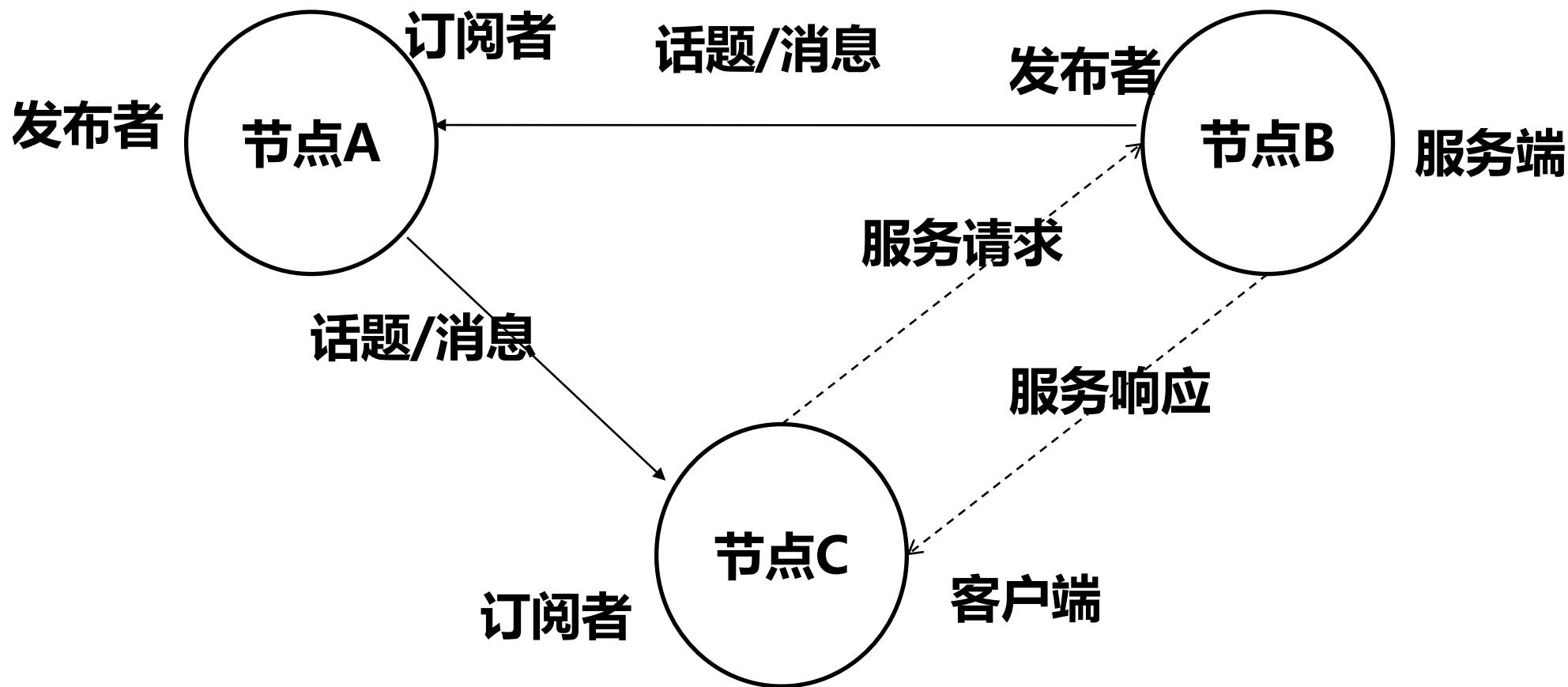
□连接子表示软构件之间的**连接和交互**关系

- ✓每个软构件并非孤立，它们之间通过连接进行交互
- ✓交互的目的是为了交换数据、获得服务

□软构件之间的典型交互方式

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

示例：构件之间的连接和交互



□二种接口(Interface)

- ✓ **供给接口**：对外提供的接口
- ✓ **需求接口**：请求其他构件帮助所需的接口

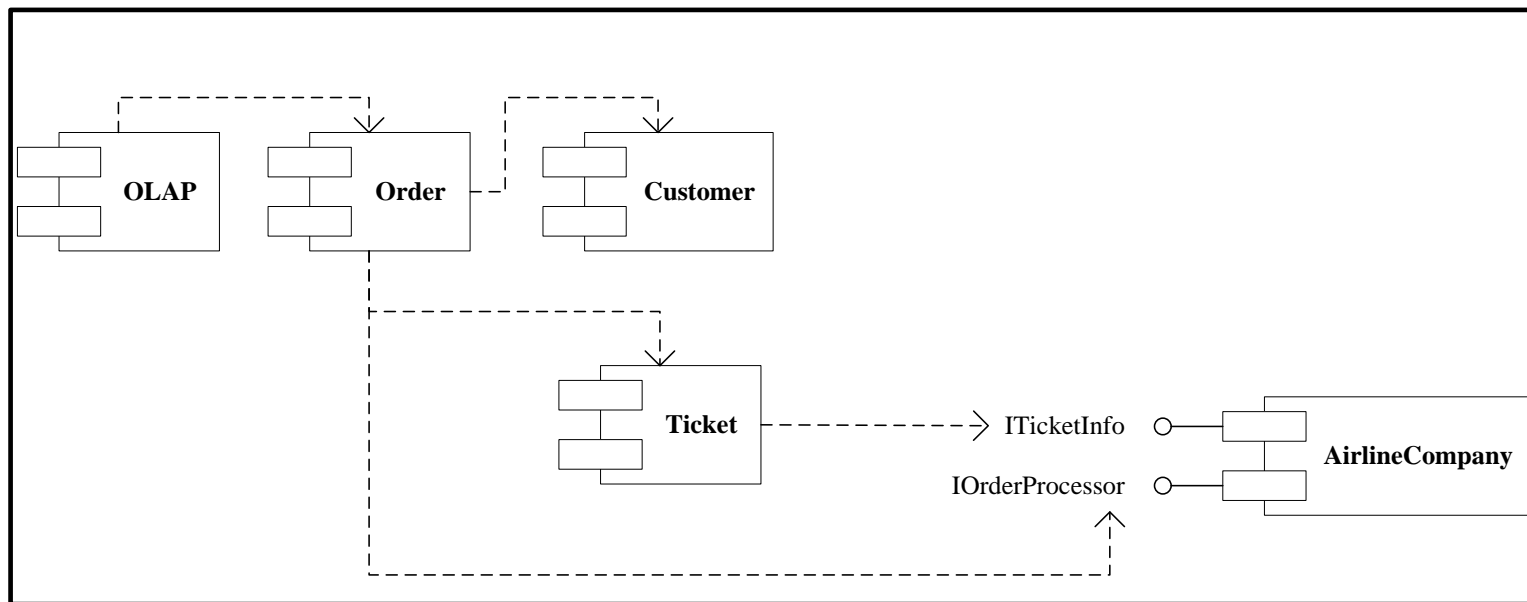
- 构件通过接口对外提供服务
- 构件通过接口与其他构件进行交互

□连接端口(Port)

- ✓ 每个**端口**绑定了一组供给接口和/或需求接口
- ✓ 构件通过端口与外部世界交互
- ✓ 当外部请求到达端口时，构件的端口知道如何将外部请求路由至合适的接口的**实现体**
- ✓ 当构件通过端口请求外部服务时，端口也知道如何分辨该请求所对应的需求接口

构件与接口

- 构件的实现与构件的接口相分离
- 构件开发者可自由选择实现方法，只要它实现供给接口中的操作及属性
- 两个构件的实现遵循相同接口定义，它们就可自由替换



连接和接口的表示

对外提供的接口

ProvidedInterfaceHere



ComponentNameHere

RequiredInterfaceHere

访问其他构件的接口

1.2.3 约束

- 约束表示构件中的元素应满足的条件，以及构件经由连接件组装成更大模块是应满足条件。
- **高层次**软件元素可向**低层次**软件元素**发请求**，低层次软件元素完成计算后向高层次发送应答，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接收职责
- 每个层次都是**可替换的**，一个层次可以被实现了同样的对外服务接口的层次所替代

1.3 软件体系结构的不同视图

软件体系结构是从结构的角度来刻画软件
可从不同角度、层次来表示和分析软件体系结构

□逻辑视图：

- ✓要素及关系，站在结构视点和抽象层次，包图、类图等，是用户和软件架构师关注点

□运行视图：

- ✓在特定时刻软构件的具体情况，活动图，系统集成、管理和维护工程师

□开发视图：

- ✓构件的代码组织及其形式，包图、构件图，架构师和程序员关注点

□物理视图：

- ✓软构件的部署及其连接，系统工程师关注点，部署图

软件体系结构的不同视图

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

1.3.1 软件体系结构的逻辑视图：包图

包图：系统（子系统）、构件及关系

□包模型

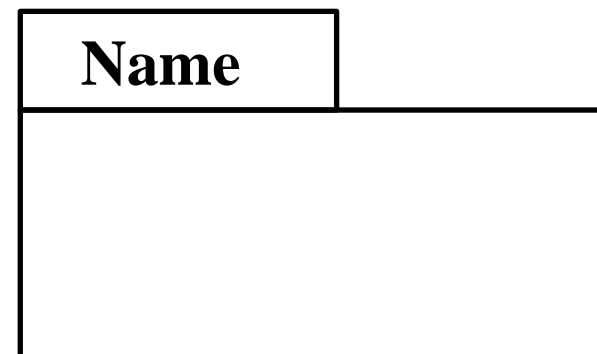
- ✓刻画**包间**的**构成和依赖**关系
- ✓包可以包含一组具有逻辑关联的UML模型元素
- ✓包间关系：**构成关系、依赖关系**

□包图可描述软件系统的高层结构

- ✓包图以结构化、层次化方式组织、管理大型的软件模型，使得分别处理不同包的开发团队之间的相互干扰程度降至最低

何为包？

- 一组具有逻辑关联的**UML模型元素**（例如用例、类等）、模型图（用例图、类图、交互图、状态图、活动图等），以及其他的包
- **包在模型管理过程中是配置管理的基本单元**，同时也为访问控制提供基本手段



□功效

✓刻画包间的**构成和依赖**关系

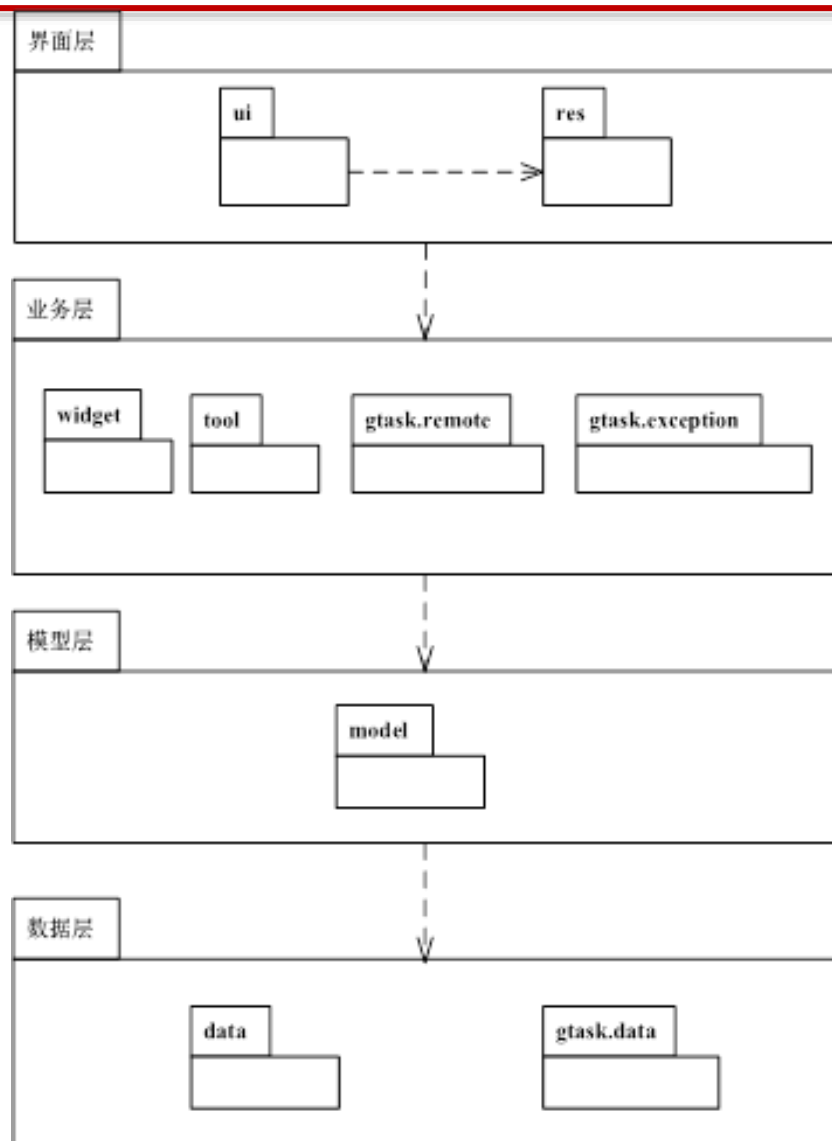
□图的构成

✓**节点**：包

✓**边**：包间的关系

□包间的关系

✓组成和依赖



□作为**软件模型**的组织单元

- ✓将大型软件系统划分成不同包，以构建各类模型

□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

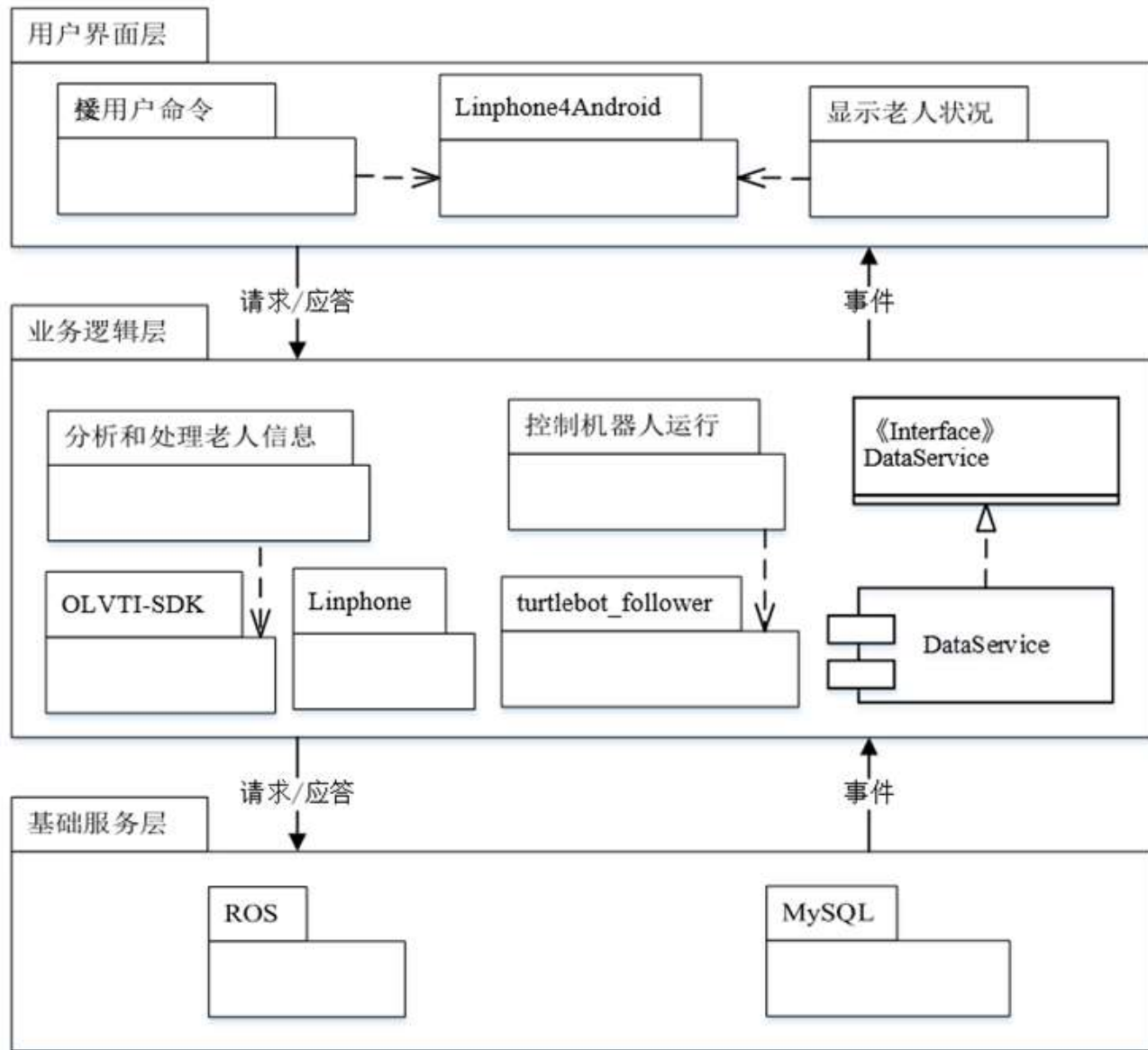
□作为系统**高层结构**中的组成元素。

- ✓逐级细分的包才是软件高层结构中恰当的组成元素

□作为**访问控制**的基本手段

- ✓将包视为名字空间，每个模型元素可通过在其名称之前拼接包名，构成全系统范围内的唯一限定名来指称它，正如在文件系统中以文件的全路径名来唯一地指称一个文件

示例：软件体系结构的包图表示



包图表示软件体系结构

1.3.2 软件体系结构的逻辑视图：构件图

□描述软件系统中构件及构件间的关系

- ✓ **结点**：构件、类和包
- ✓ **关系**：构成关系、依赖关系

□对软件体系结构的描述

- ✓ 描述软件系统中**构件的接口定义及构件间的依赖关系**，以便评估软件变更的影响范围
- ✓ 描述软件系统或其中某个局部的**构件化设计**，为在后续开发阶段实现构件化的软件模块订立设计规约

构件及其接口

□构件

- ✓ 一个或者多个可独立部署的**执行码文件**
- ✓ 具有精确定义的**供给接口和需求接口**
- ✓ **可分离**，接口和实现分离
- ✓ **可替换**，构件实例可被任何实现相同接口的同一构件实例所替换

□接口

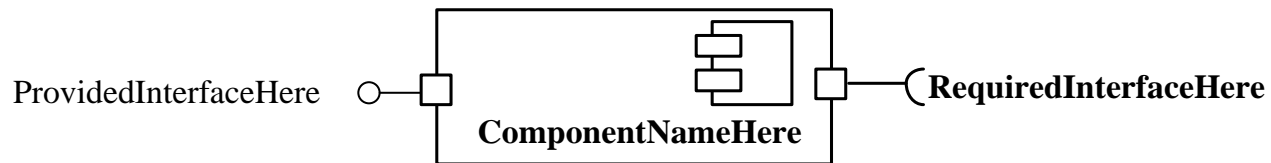
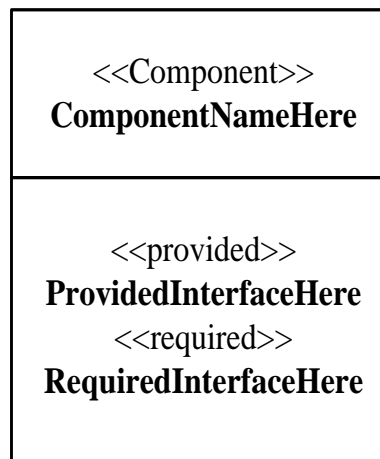
- ✓ 一组操作 和/或 属性的说明（不含操作的实现），它用作服务提供方和使用方之间的协议
- ✓ 每个构件还可以定义一些端口（port），每个端口绑定了一组**供给接口和/或需求接口**
- ✓ 接口由类或构件实现

构件和接口的表示

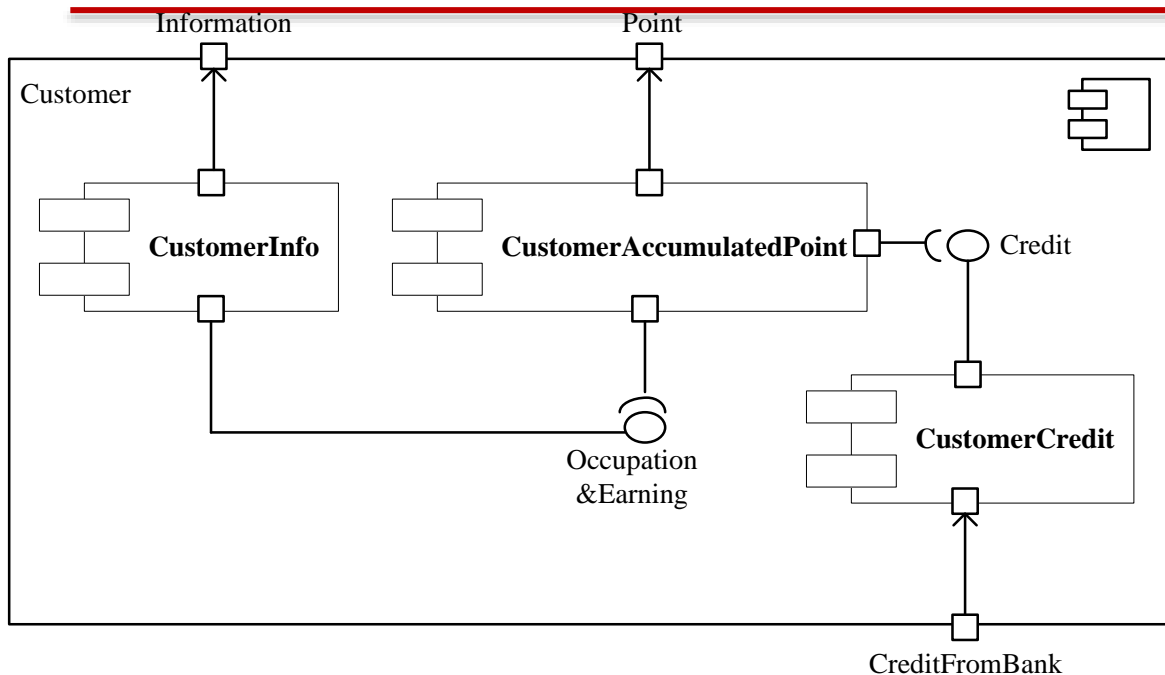
□ 构件的三种图元



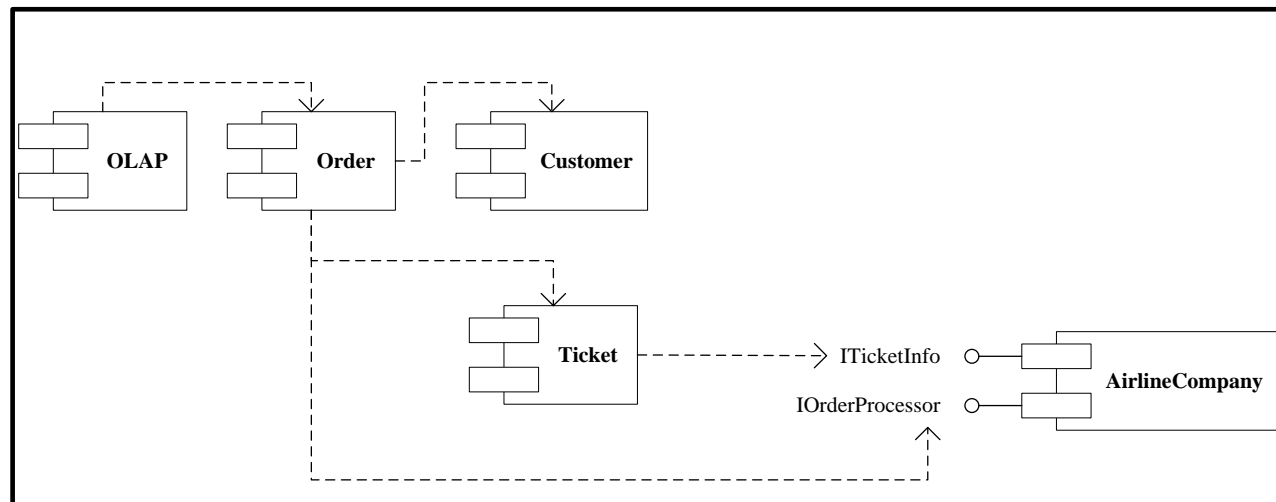
□ 构件及其接口的两种表示方法



示例：软件体系结构的构件图表示



- 构件
- 接口和端口
- 关系(依赖)



1.3.5 软件体系结构的运行视图

- 软件运行时进程、线程的划分，它们之间的并发和同步，它们与逻辑视图和开发视图之间的映射关系
- 可用UML的活动图、对象图来表示

□功效

- ✓ 表示软件系统可执行工件(artifact)在运行环境中的部署和分布情况
- ✓ **工件**是指软件中相对独立的物理实现单元，如动态链接库（DLL）文件、Java类库（jar）文件

□图的构成

- ✓ 节点：计算节点、工件、构件
- ✓ 边：通信和依赖

□两种部署图

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

描述性部署图的节点

□节点

- ✓表示软件运行环境中的一组**计算资源**（如Web服务器、应用服务器、数据库服务器等），软件工件驻留于其中，并依赖其提供的基础服务

□工件

- ✓以构造型<<artifact>>标识，工件位于节点图元之内

□构件

- ✓位于节点之内的构件表示该构件部署于相应节点，位于节点之外的构件主要用来指明工件与构件之间的依赖关系

□与用例有关的执行者

- ✓在部署图中引入执行者意在强调执行者与部署于节点中的工件之间的使用关系及信息交互

描述性部署图的边

□节点之间的通信关联

- ✓表示两个节点间的通信连接

□工件之间的依赖关系

- ✓如果工件定义了对外接口，那么应确保工件间的依赖关系表现为工件接口上的依赖关系，如此可降低工件之间的耦合度

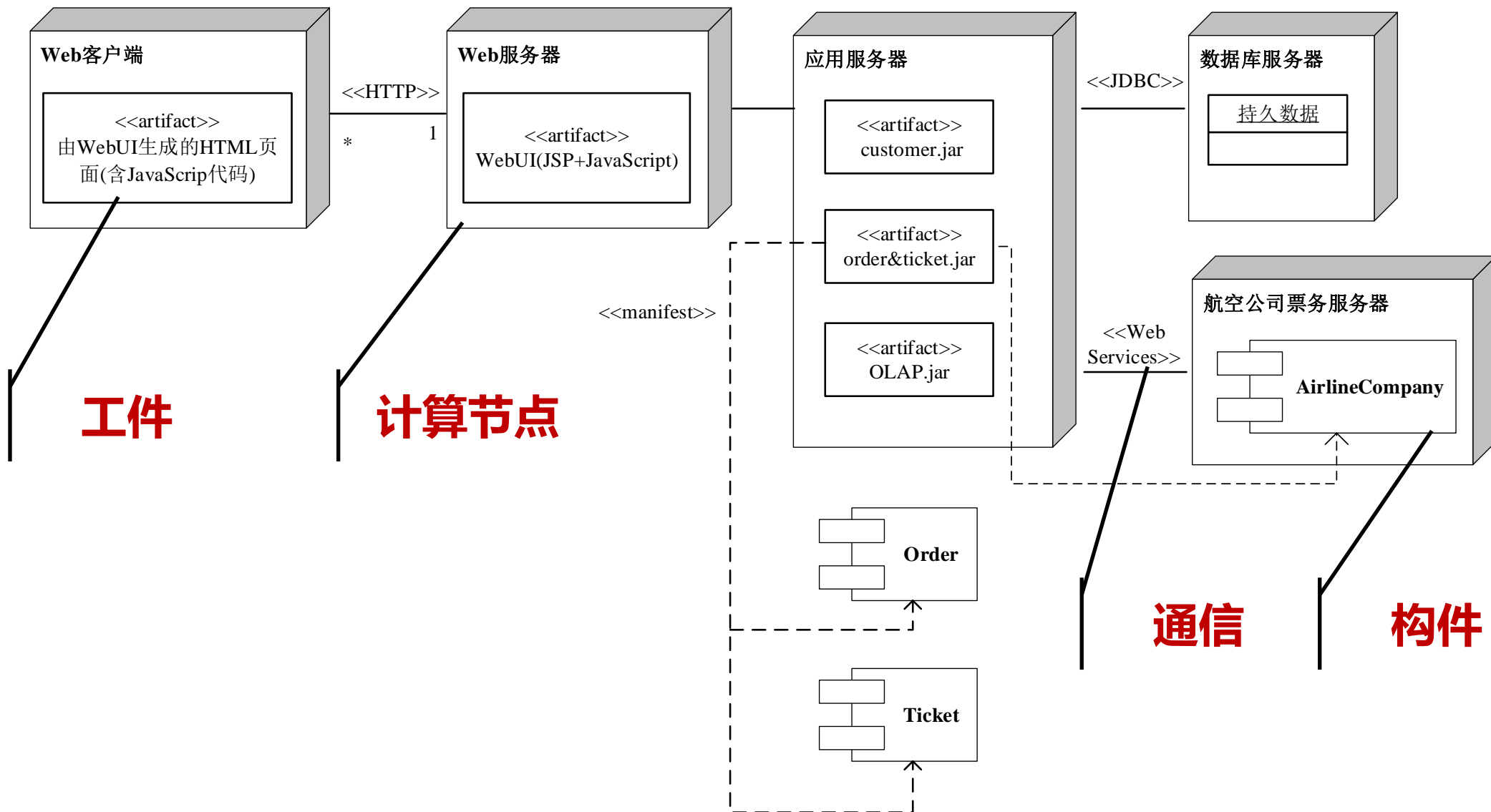
□工件与构件之间的依赖关系

- ✓表示工件具体实现了构件

□用例执行者与工件之间的依赖关系

- ✓表示执行者使用工件提供的服务或工件使用执行者提供的服务

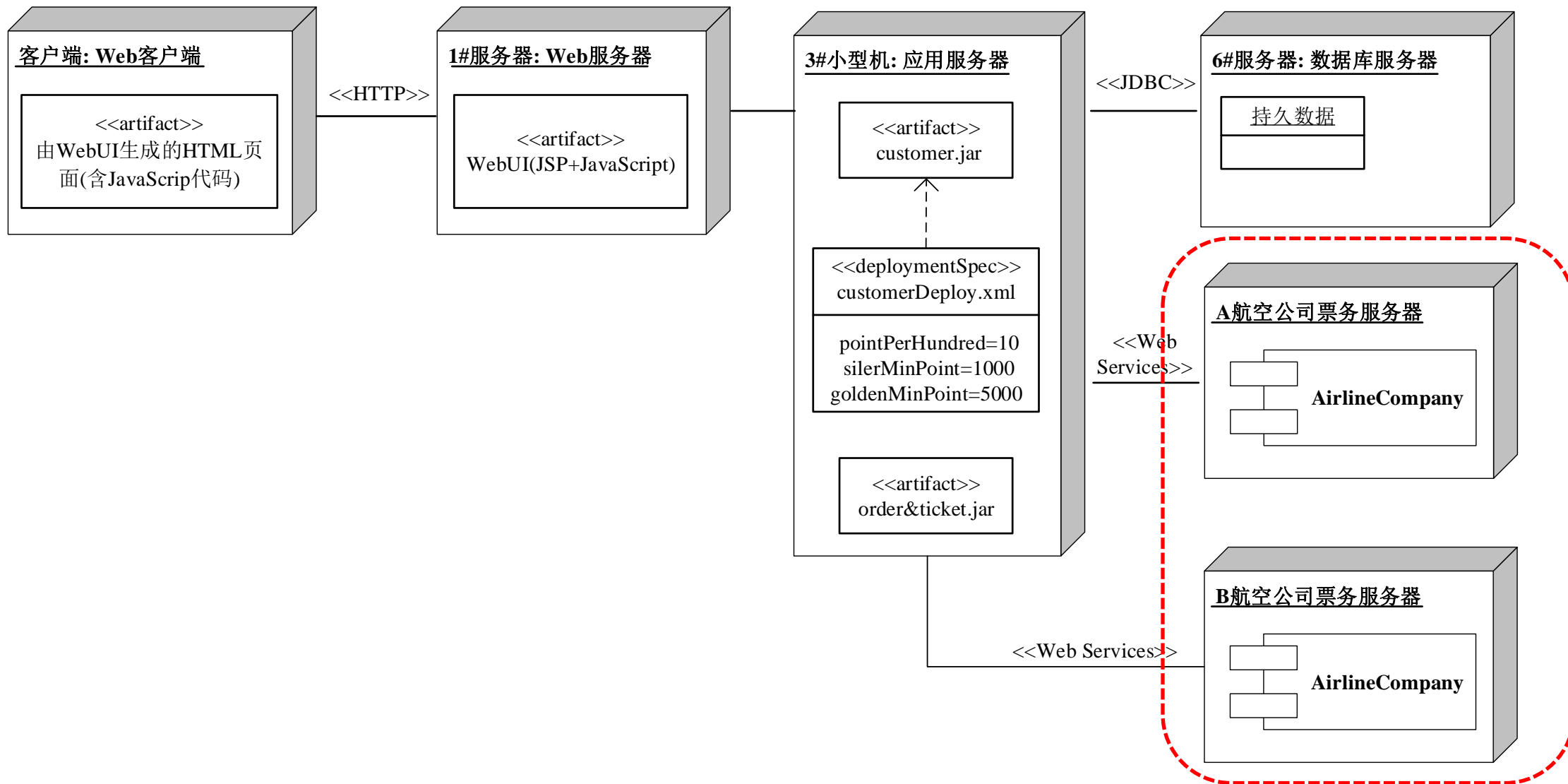
示例：描述性部署图



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

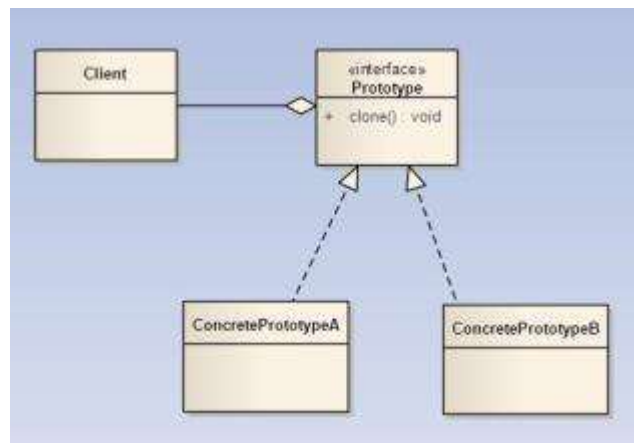


部署图的建模原则

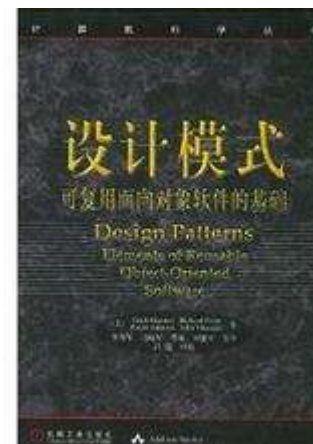
- 节点按行上下对齐，按列左右对齐，边绘制成水平线段、垂直线段或者由水平、垂直两种线段组合而成的折线，避免斜线
- 工件和构件的放置应尽量使依赖关系的方向从左至右
- 供给接口位于构件的左侧，需求接口位于构件的右侧

1.5 软件设计模式

- 借用以往的**经验**来解决问题
- 以**设计重用**为目的，采用一种良定义的、正规的、一致的方式记录的软件设计经验
- 关注在一般或特定设计环境中可能重复出现的设计问题，并给出经过**充分实践考验**的软件解决方案



- 名称
- 问题
- 施用和约束条件
- 解决方案
- 效果



不同抽象层次的设计模式

□ 体系结构风格

✓ 面向整个软件系统

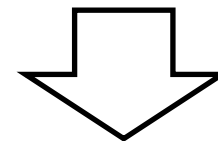
□ 构件设计模式

✓ 面向子系统或者构件

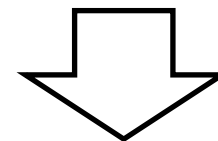
□ 实现设计模式

✓ 针对子系统或构件中的某个特定问题

整体、全局



局部



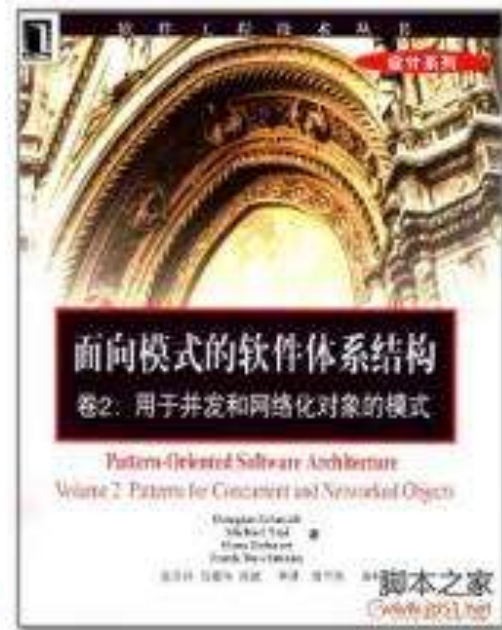
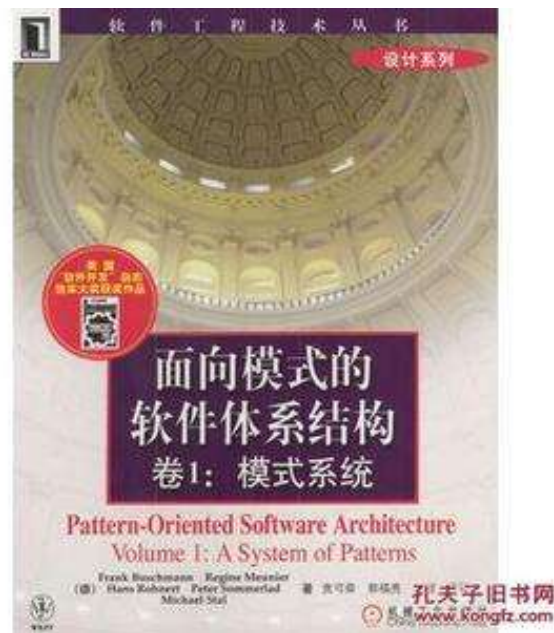
细节

体系结构设计风格

- 人们对大量软件体系结构设计总结和分析之后发现，相似的应用常采用相同的软件体系结构形式，提炼形成一系列体系结构风格
- 面向**整个软件系统**，在**抽象层次**给出软件体系结构的结构化组织方式
- 提供一些**预定义**的子系统或者构件，规定其**职责**，明确它们之间的**相互关系**、协作方式的规则或指南
- 针对不同的问题采用不同的体系结构模式

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
-



1.5.1 分层体系结构模式

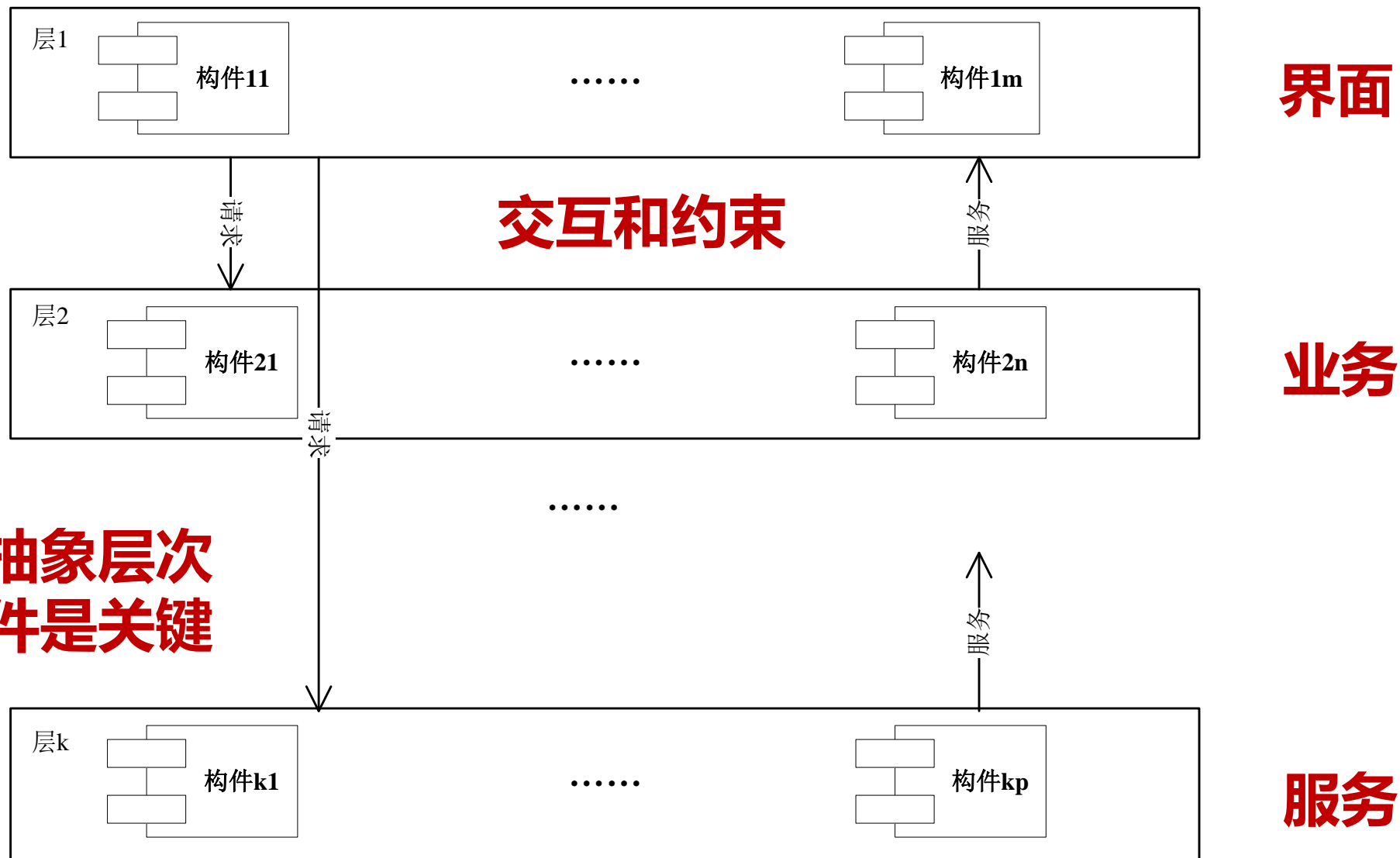
□模式思想

- ✓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

□典型层次示例

- ✓**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：分层体系结构风格



合理地设计抽象层次
和组织软构件是关键

分层体系结构模式的约束

□ 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
- ✓ 上层向下层发出**服务请求**，下层为上层反馈服务结果
- ✓ 下层主动探测或被动获知计算环境的变化后，向上层提供**事件信息**，上层对下层通知做出处理

□ 服务接口的组织方式

- ✓ 层次中的每个构件**分别**公开其服务接口
- ✓ 将服务接口封装与层次的内部，每个层次提供统一的服务接口

分层体系结构模式的特点

□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

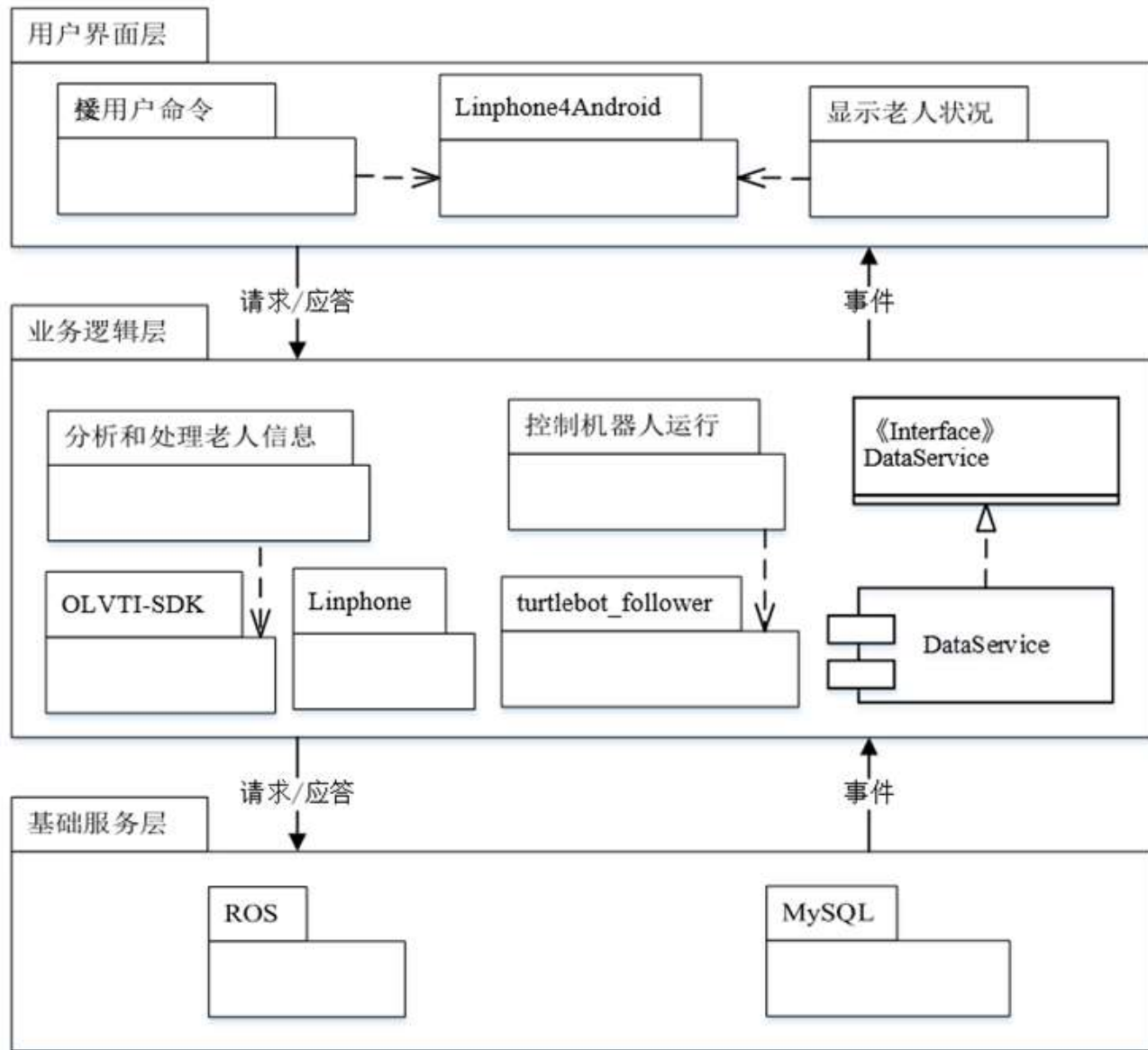
□可复用

- ✓层次和整个系统可重用

□标准化

- ✓支持体系结构及其层次、接口的标准化

示例：分层的软件体系结构



1.5.2 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

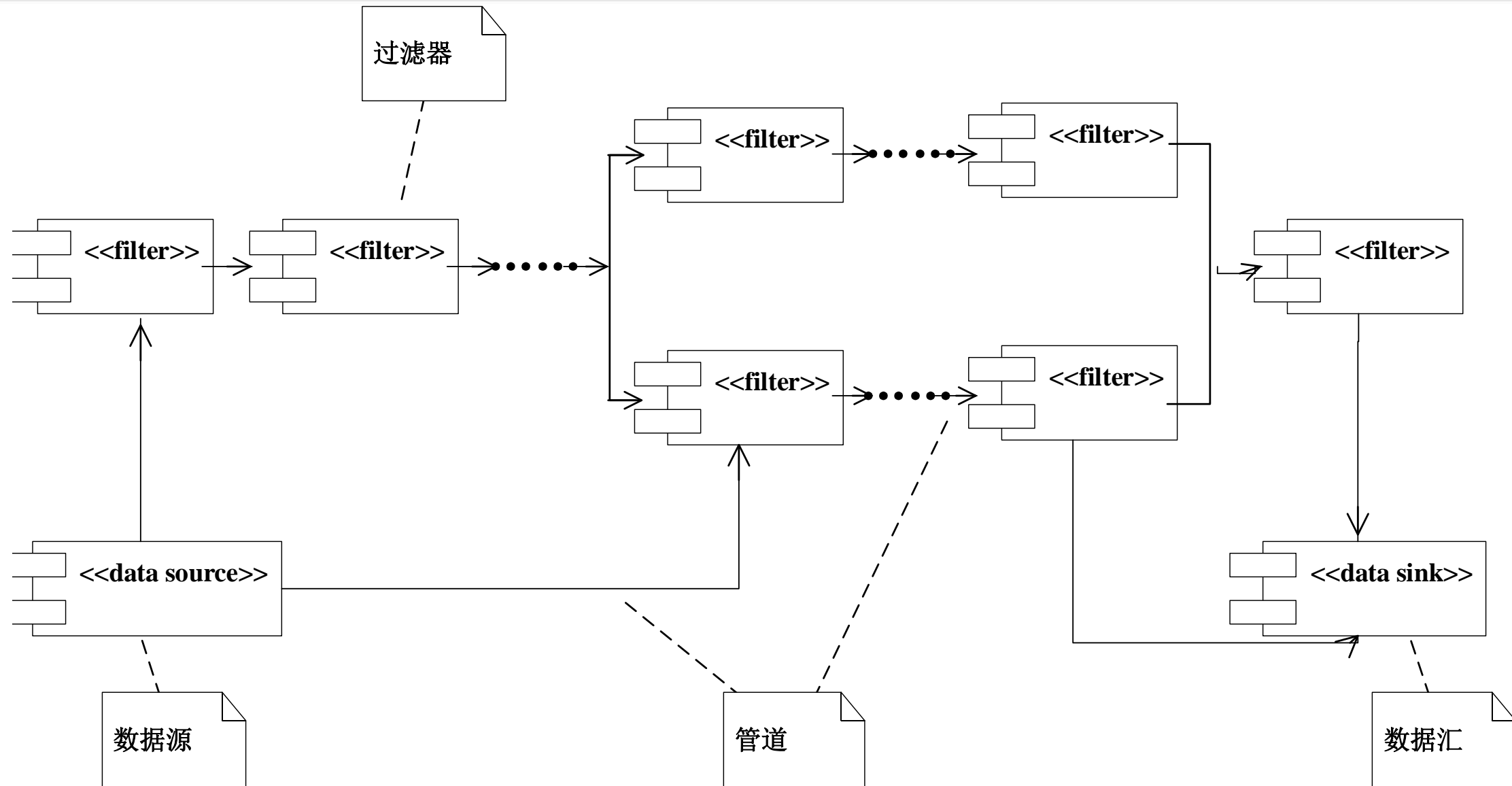
□ 连接子

- ✓ 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

管道与过滤器模式的示例



管道与过滤器风格的约束

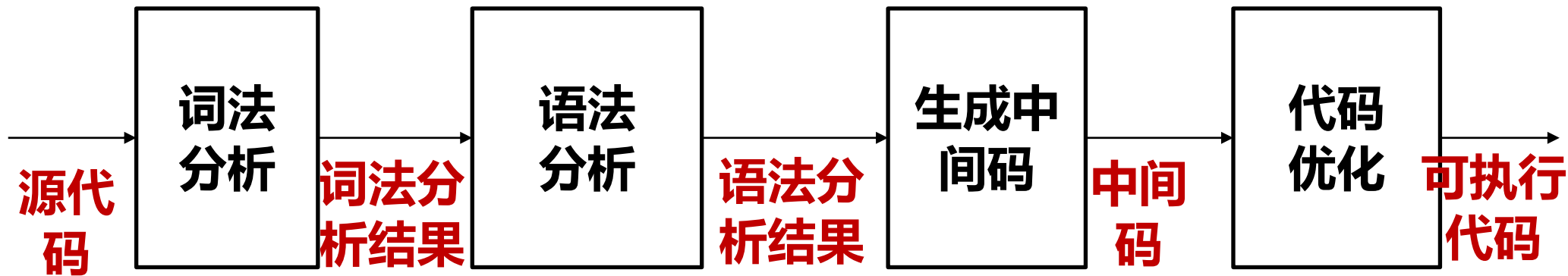
□过滤器与管道之间的协作方式

- ✓ 过滤器以循环的方式不断地从管道提取输入数据，并将其输出数据压入管道，称为**主动过滤器**
- ✓ 管道将输入数据压入到位于其目标端过滤器，过滤器被动地等待数据，称为**被动过滤器**
- ✓ 管道负责提取位于其源端过滤器的输出数据

□设计考虑

- ✓ 如果管道连接的两端均为主动过滤器，那么管道必须负责它们之间的同步，典型的同步方法是**先进先出缓冲器**
- ✓ 如果管道的一端为主动过滤器，另一端为被动过滤器，那么管道的数据流转功能可通过前者直接调用后者来实现

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

管道与过滤器模式的特点

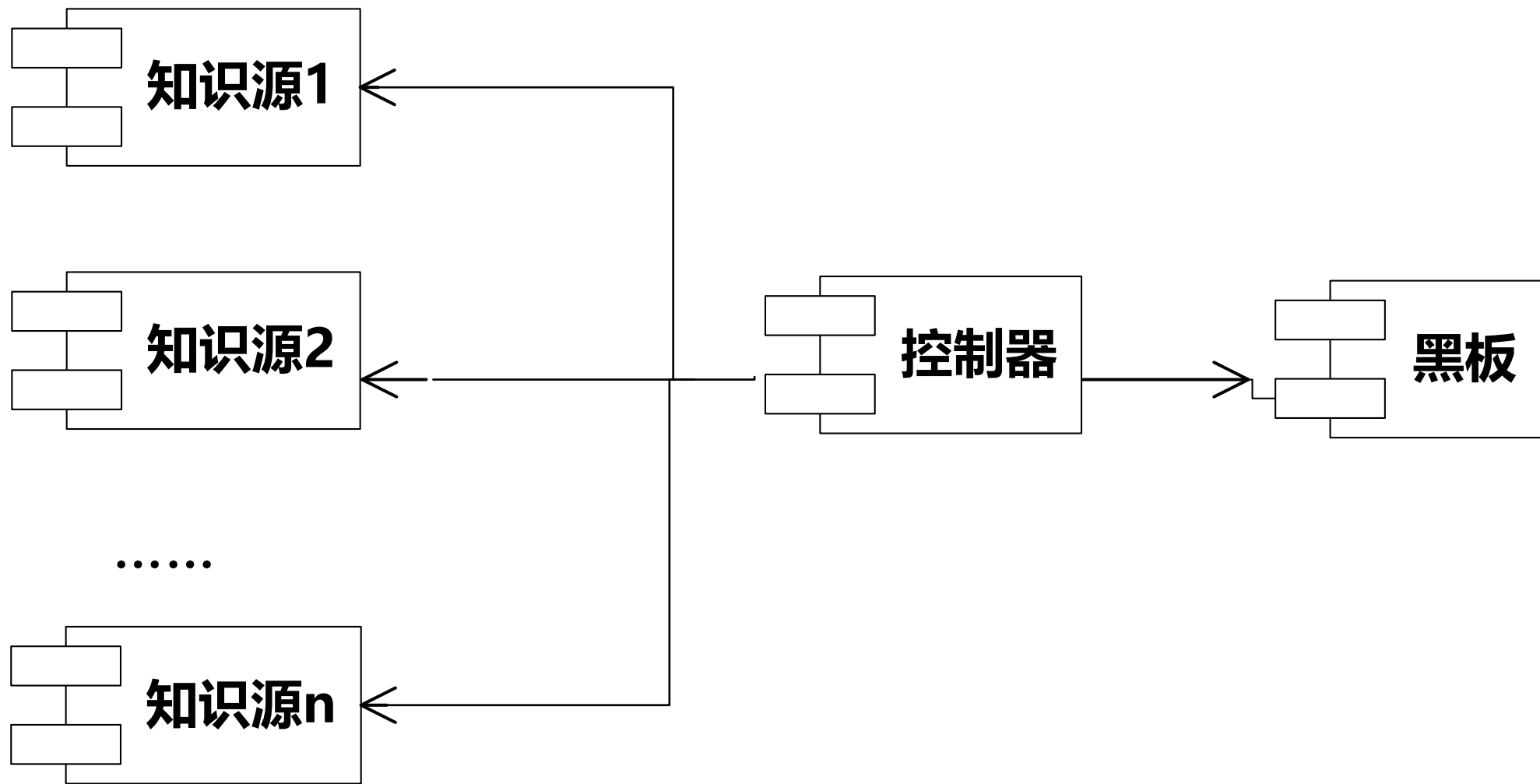
- 自然地解决具有数据流特征的软件需求
- 可独立地更新、升级过滤器来实现软件系统的扩展和进化

1.5.3 黑板风格

□将软件系统划分为黑板、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**：负责根据黑板中存储的问题求解状态评价其自身的可应用性，进行部分问题求解工作，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**中的状态数据来决定哪些知识源对后续的问题求解可能有所贡献，然后调用这些知识源的评价功能以选取参与下一步求解活动的知识源
- 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

1.5.4 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**

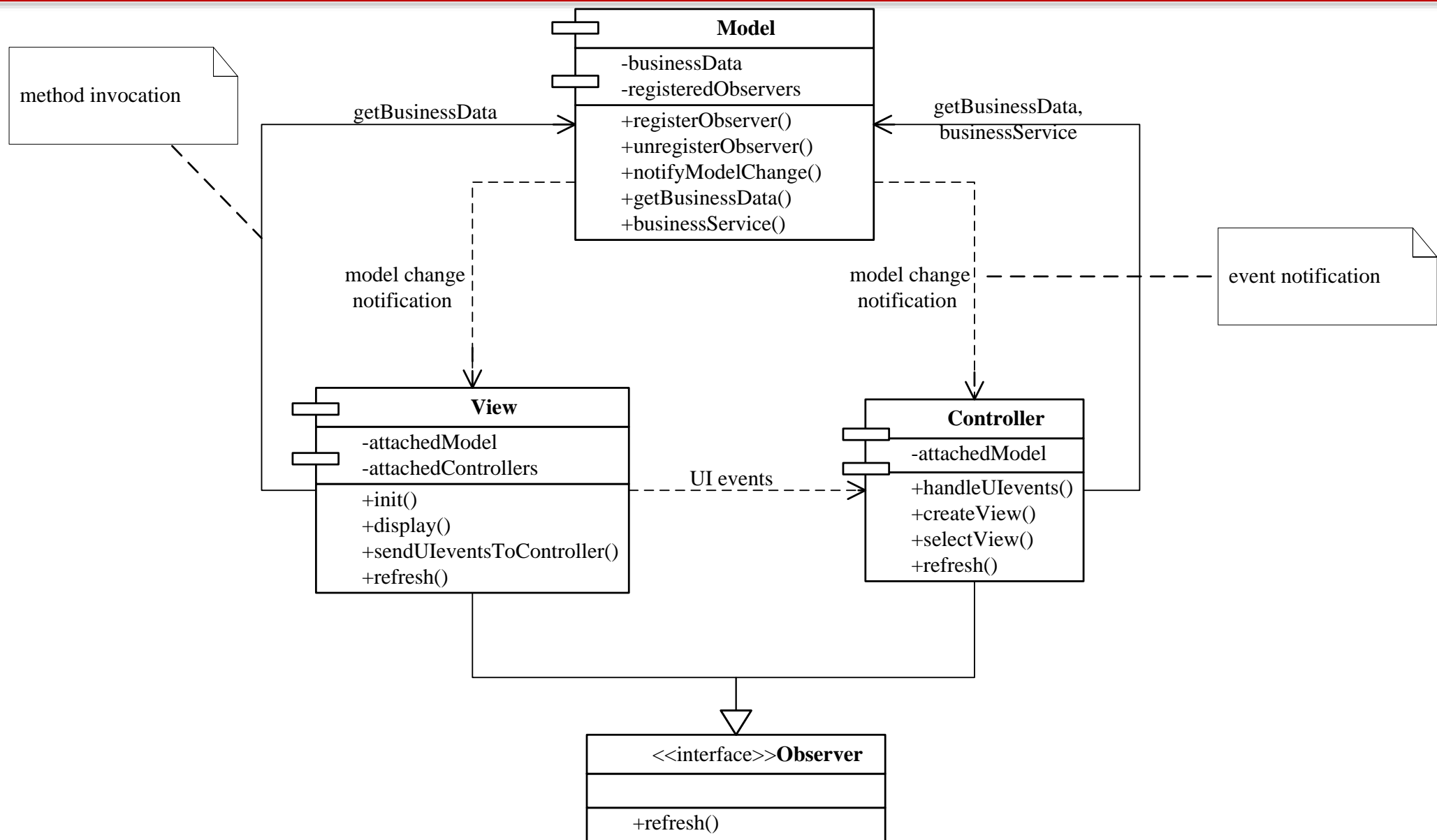
□视图构件

- ✓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**

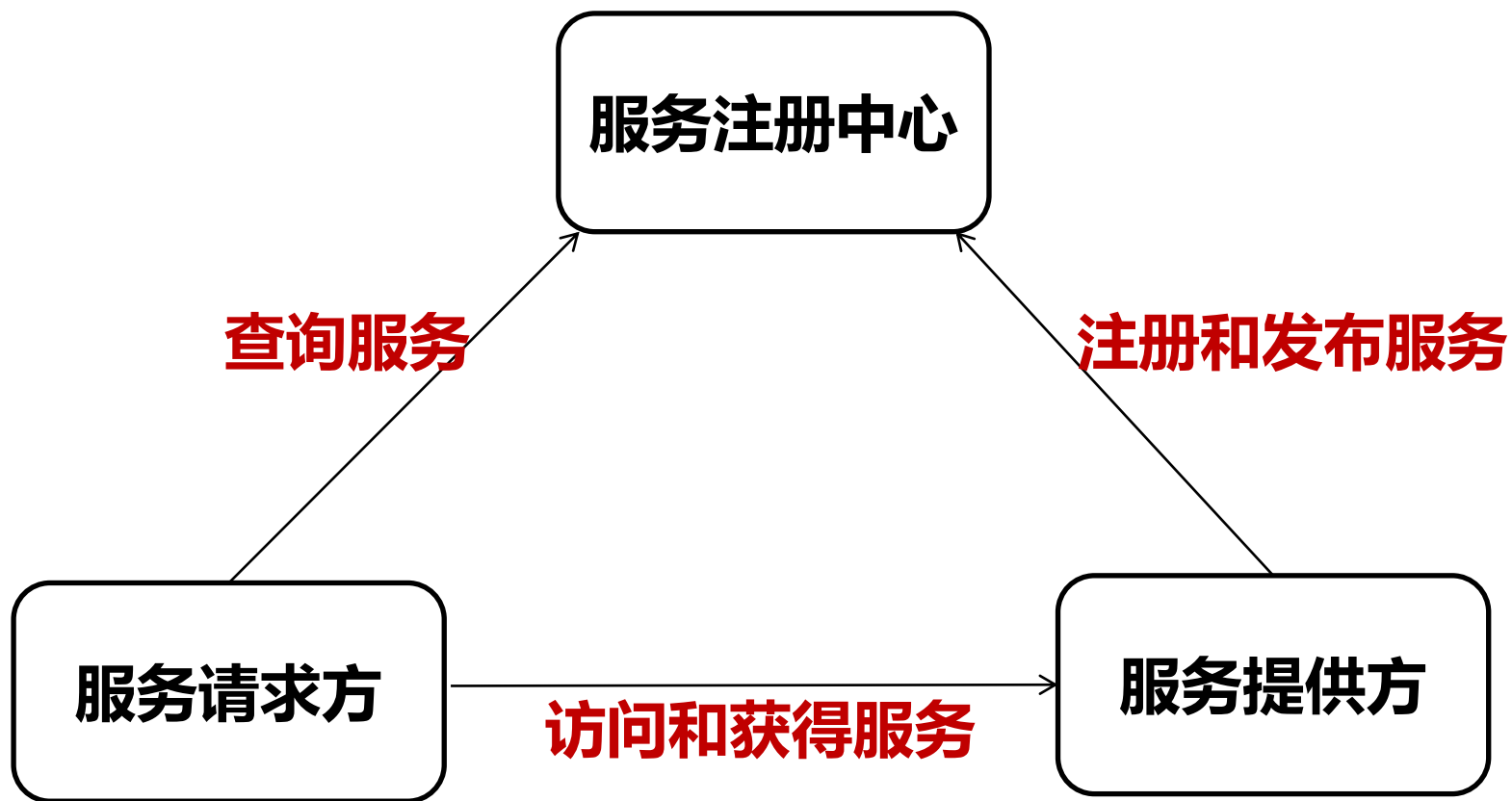


1.5.5 SOA风格（面向服务的体系架构）

- 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软件体系结构风格示意图





SOA风格的特点

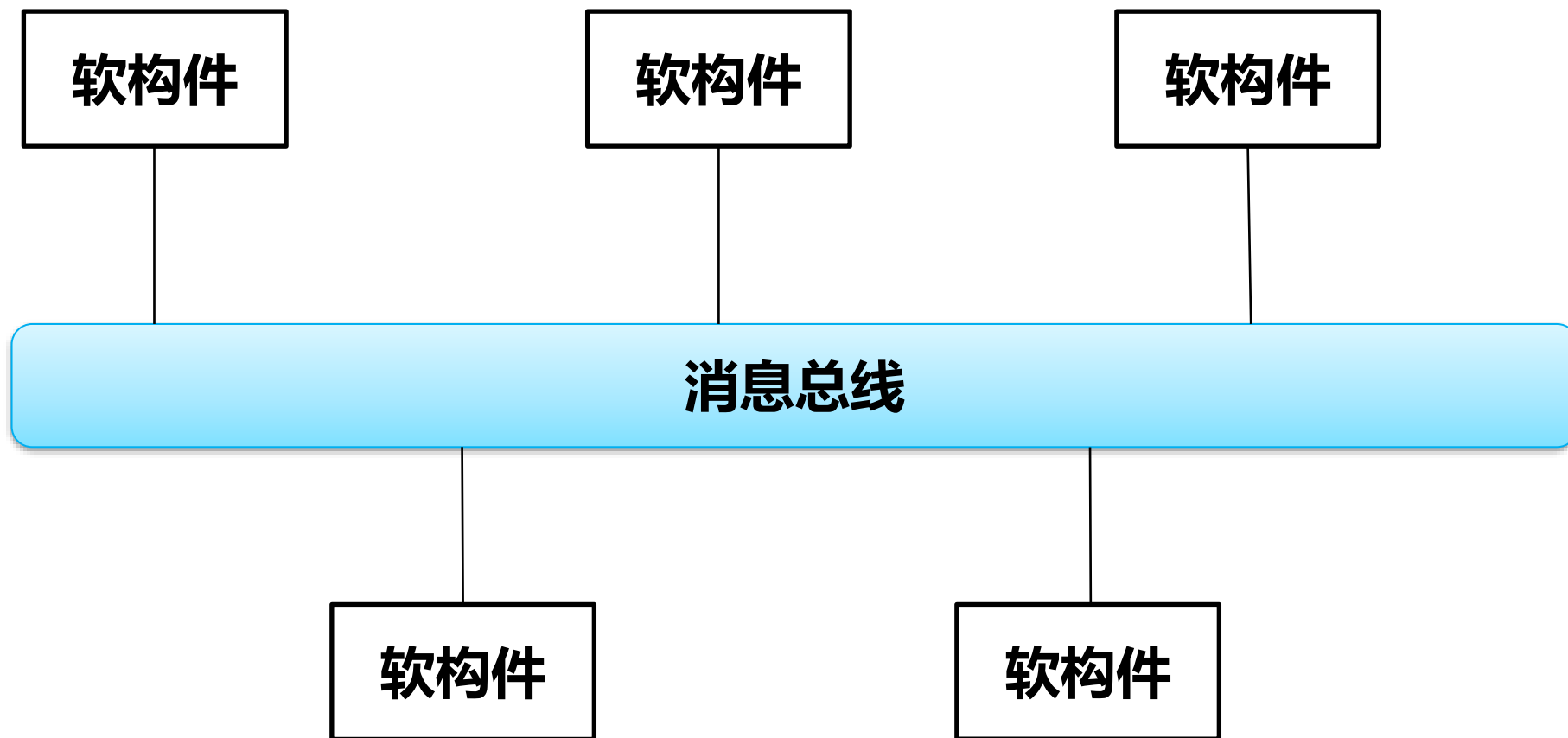
- 将服务提供方和服务请求方独立开来，因而支持服务间的**松耦合定义**
- 允许任何一个服务在运行过程中所**扮演角色**的动态调整，支持**服务集合**在运行过程中的**动态变化**，因而具有非常强的灵活性
- 提供了诸如UDDI、SOAP、WSDL等协议来支持服务的注册、描述和绑定等，因而可有效支持**异构服务间的交互**

1.5.6 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





内容

1. 何为软件体系结构

- ✓ 概念、视图、模型及UML表示
- ✓ 软件体系结构风格

2. 软件体系结构设计

- ✓ 任务、目标、要求和原则
- ✓ 体系结构设计过程

3. 文档化和评审软件体系结构设计

- ✓ 文档模板、验证原则



2.1 软件体系结构设计任务

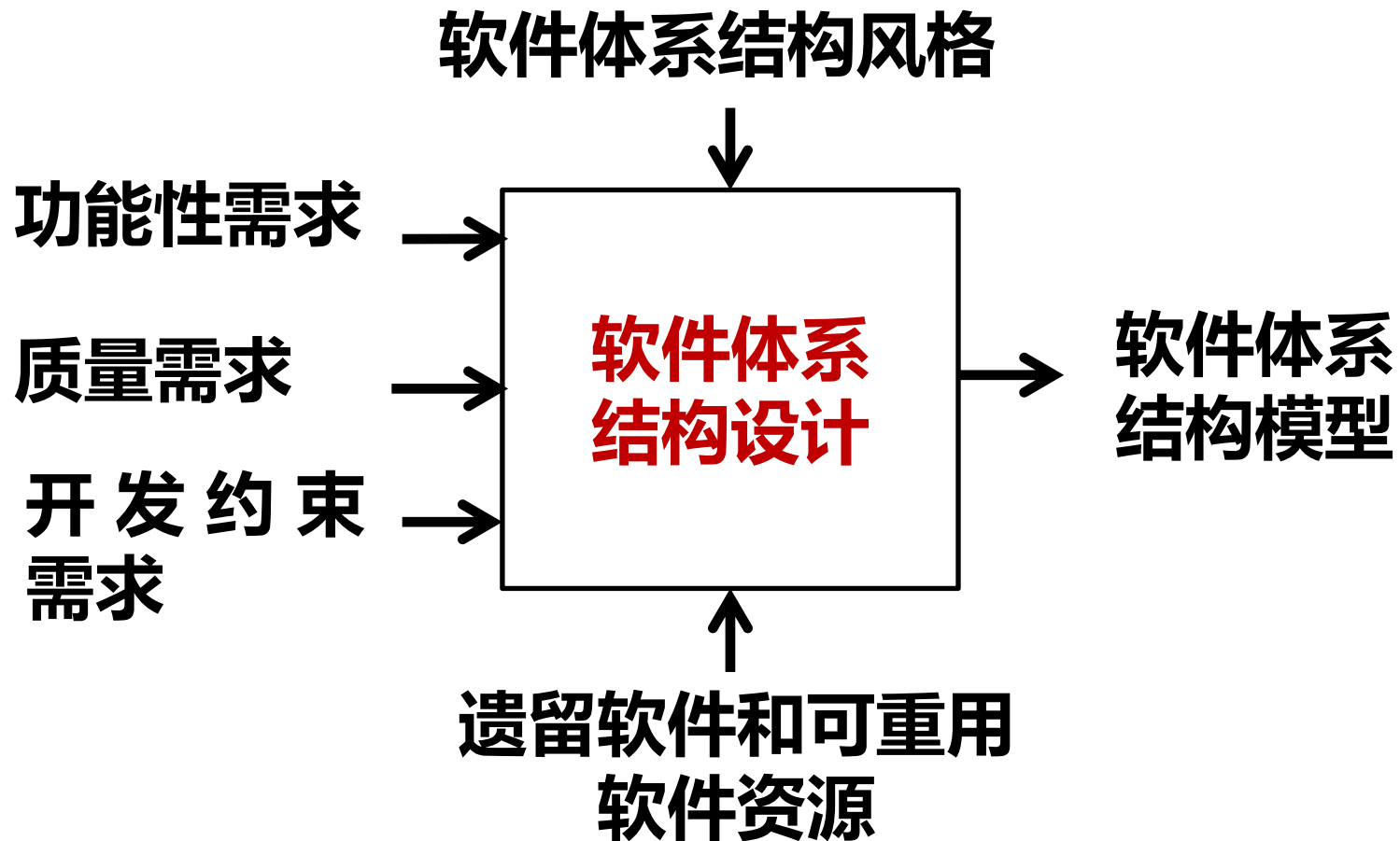
□建立满足软件需求的软件体系结构

- ✓明确定义各子系统、构件、关键类的**职责划分及协作关系**
- ✓明确它们在物理运行环境下的部署

□特点

- ✓针对软件系统**全局性、基础性技术问题**给出技术解决方案
- ✓宏观、全局、层次、战略、多视点、关键性

软件体系结构设计示意图



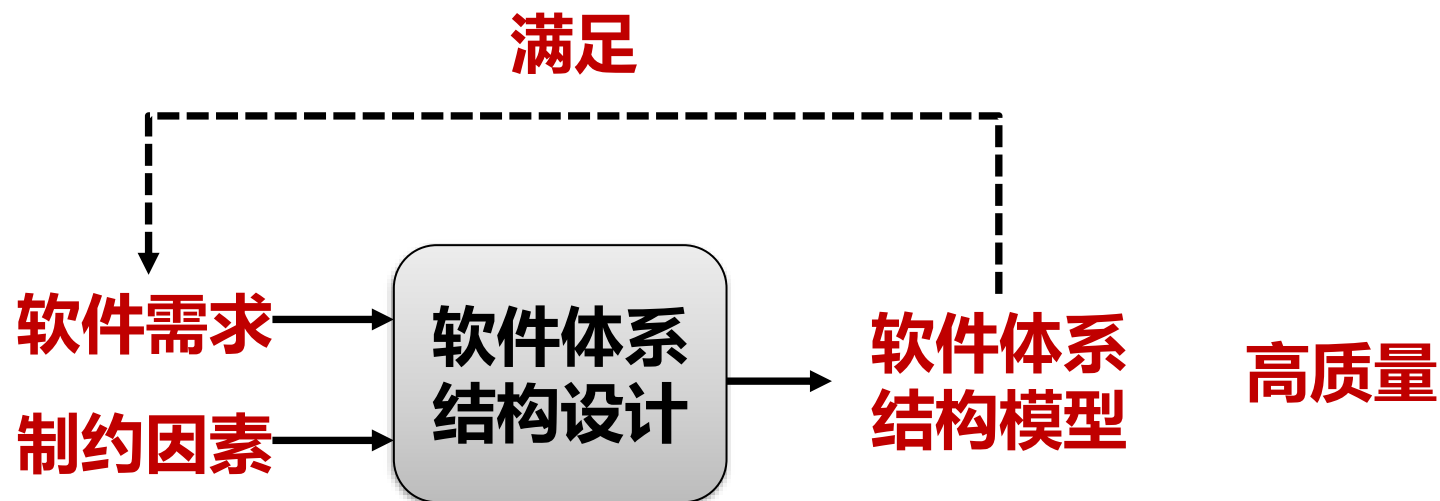
2.2 软件体系结构设计的目标

□ 满足软件需求

- ✓ 实现软件需求
- ✓

□ 追求高质量

- ✓ 可扩展
- ✓ 可伸缩
- ✓ 易维护
- ✓



非功能需求在体系结构设计中起着非常重要的作用

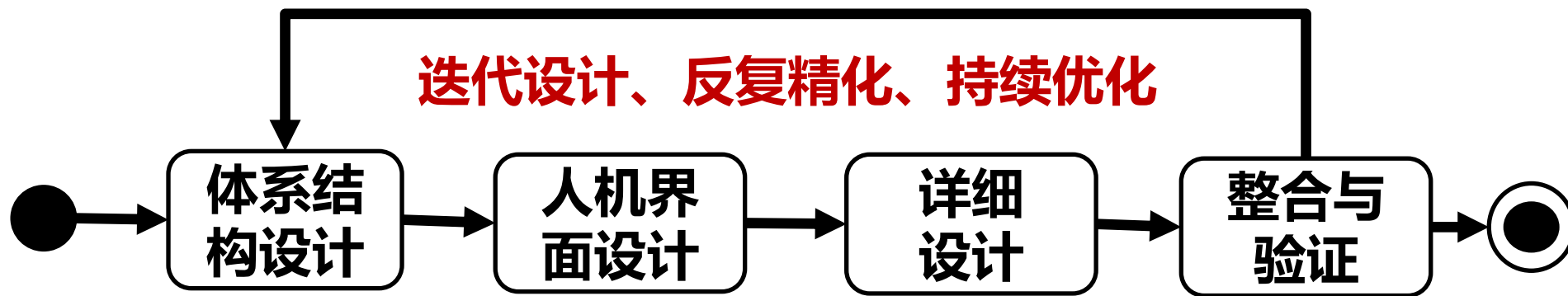
软件体系结构设计 vs 软件需求

- 体系结构是以软件需求实现为目标的**软件设计蓝图**
- **软件需求**是体系结构设计的基础和驱动因素
- 软件需求，尤其是**非功能需求**，对软件体系结构具有关键性的塑形作用



软件体系结构设计 vs 详细设计

- 体系结构设计为详细设计提供**可操作指导**
- 详细设计是对体系结构设计中设计要素的**局部设计**
- 详细设计须**遵循体系结构设计**：如接口和约束
- 详细设计**只能实现、不能更改**体系结构设计中规定模块的接口和行为



基于体系结构设计的结果来指导详细设计



软件体系结构设计的重要性和关键性

□承上启下

- ✓ 软件需求 — 详细设计，后续的详细设计和软件实现的主要工作基础

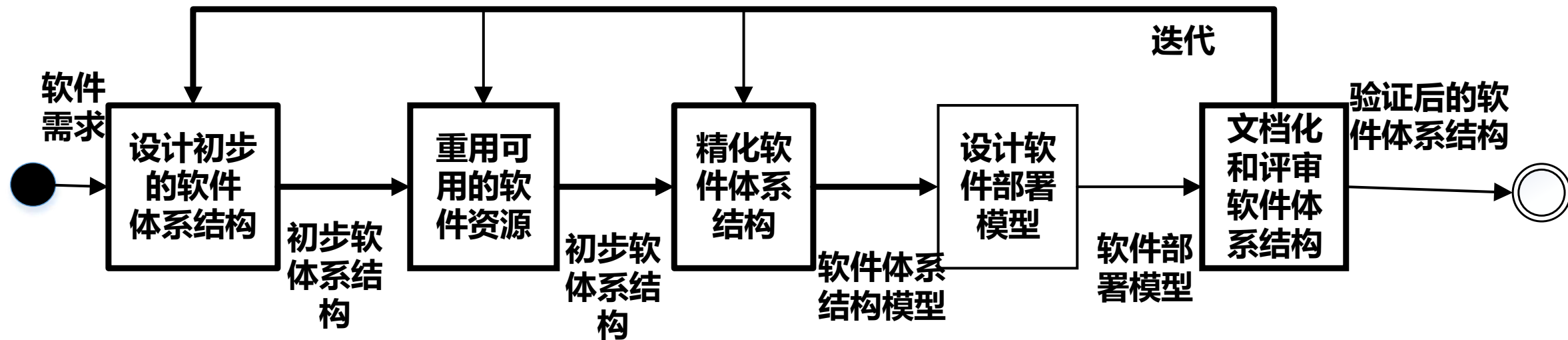
□影响深远

- ✓ 对性能、灵活性、可修改性、可扩充性等质量需求的影响是决定性的，软件质量的瓶颈

□定型质量

- ✓ 对软件质量的全局性、决定性影响

2.3 软件体系结构设计的过程



软件体系结构设计的原则

- 高层抽象和组织
- 模块化
- 信息隐藏
- 软件重用
- 多视点分离

遵循这些设计原则有助于得到高质量的软件体系结构

2.3.1 设计初步的软件体系结构

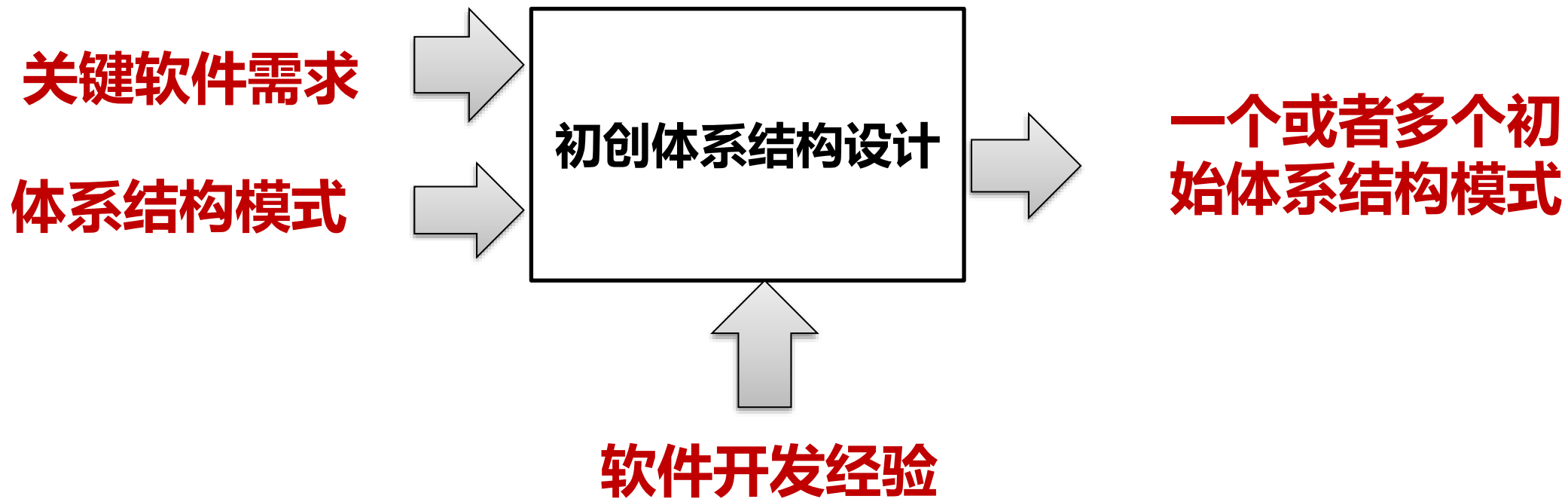
□任务

- ✓ 基于功能性和非功能性**软件需求**，参考业界已有的**软件体系结构设计风格**，设计出目标软件系统的初始体系结构，明确每个构件的职责以及构件间的通信和协作关系

□输出：初步的软件体系结构

- ✓ 针对软件需求，寻求体系结构风格，给出**初步和粗糙**的顶层架构，以供后续设计阶段的精化和细化
- ✓ 无需关注架构中各个子系统或构件内部的实现细节
- ✓ 用**UML中的包图**对所设计的顶层架构进行直观表示

初始体系结构设计的方法



辨识关键软件需求

□确定最能体现待开发软件特色、价值的**核心功能需求**

- ✓通常是杀手功能，独有的功能、基础性功能

□对软件系统影响大的质量需求；

- ✓质量属性需要在体系结构层面加以考虑和实现

□软件开发的设计约束

- ✓不同的计算形式会对软件体系结构选择产生重大影响

□标识实现**难度较大**、实现**风险较高**的功能

- ✓高难度、高风险功能的设计，及其可操作的风险化解策略

根据关键需求选择合适体系结构风格

□积累体系结构风格清单

- ✓人们总结出了几十种软件体系结构风格

□对风格进行分门别类

- ✓便于快速找到适合于关键需求的风格

□要熟悉每种风格的特点和应用场所

- ✓各种风格适合解决的问题和需求

□针对关键需求对比多种风格

- ✓正面效果是否适合、负面效果是否可容忍、是否与关键需求相冲突

不同体系结构风格适合的应用

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，各个软件构件可重用	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

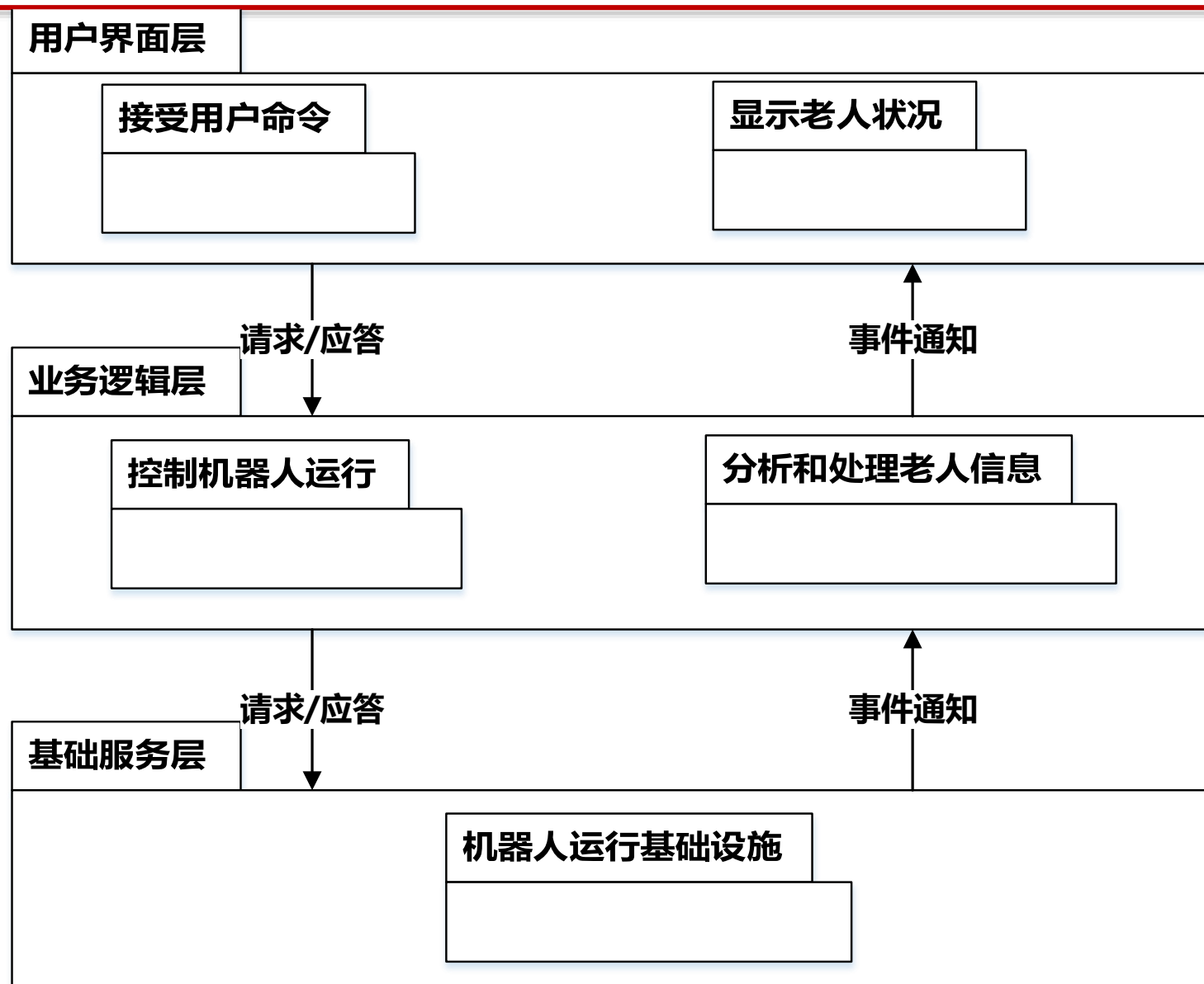


示例：初步软件体系结构的设计(1)

□ “空巢老人看护软件” 的关键软件需求

- ✓ 监视老人、自主跟随老人、获取老人信息、检测异常状况、通知异常状况、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。其他二项软件需求为非关键软件需求。
- ✓ 分析不同质量需求对软件系统的竞争力带来的影响和挑战，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
- ✓ “空巢老人看护软件” 需要采用分布式的运行和部署形式，前端软件制品部署在Android手机上，后端软件制品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：初步软件体系结构的设计(2)



**“空巢老人看护软件”
初步软件体系结构设计**

2.3.2 重用开源软件及已有软件资产

□ **搜寻已有的软件资源**，找到可用于实现软件功能的可重用软件制品，为初步软件体系结构**注入新的设计元素**，**充实和完善初步软件体系结构**

□ **搜索已有的粗粒度软件资产**

- ✓ **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

将软件资产集成到体系结构设计中

□可直接使用的软件资产

- ✓ 清晰地定义它们与当前软件系统间的交互接口
- ✓ 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产

- ✓ 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，调整当前体系结构中面向可复用设计资产的调用接口，使之与其提供的服务接口相匹配

到哪里找开源软件？

□ Github(github.com/)

✓ 开源软件托管平台和开源社区，180万商业组织，2700万开发者，8000万个版本库

□ SourceForge(sourceforge.net/)

✓ 开源软件仓库和开发平台，370万用户，43万项目

□ Gitee(www.gitee.com/)

✓ 中国人的开源社区

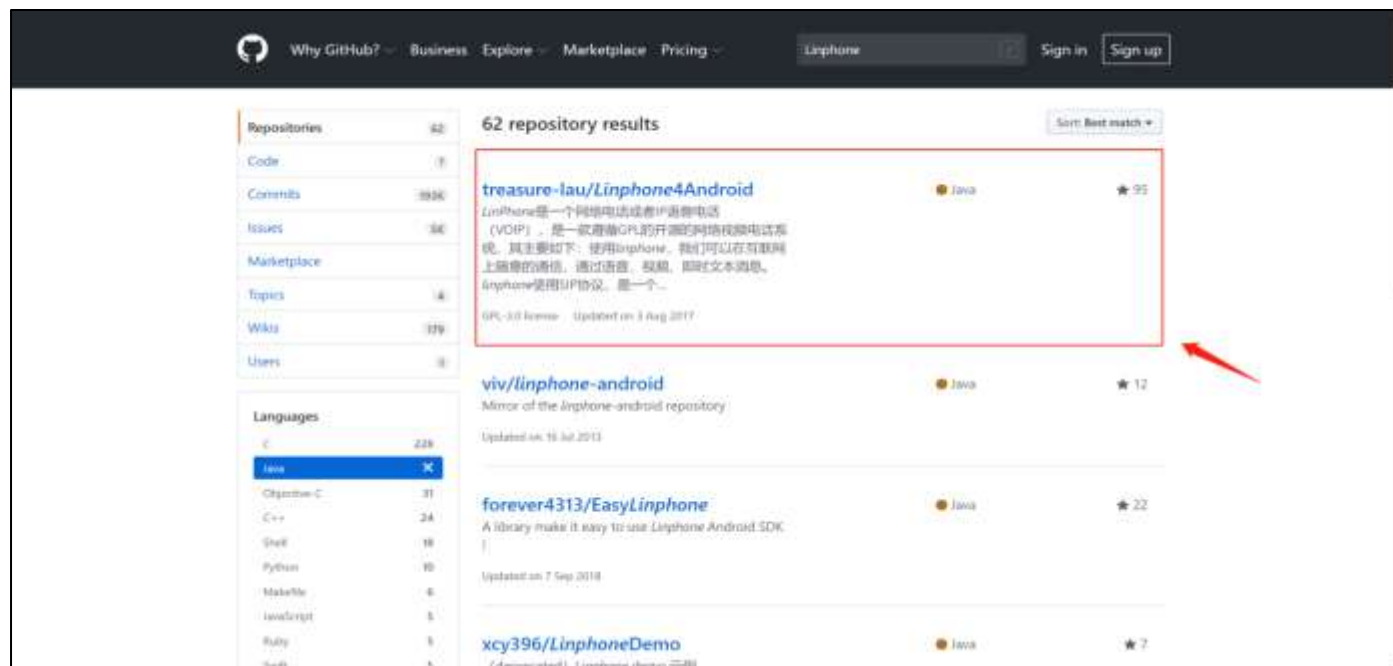
□



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

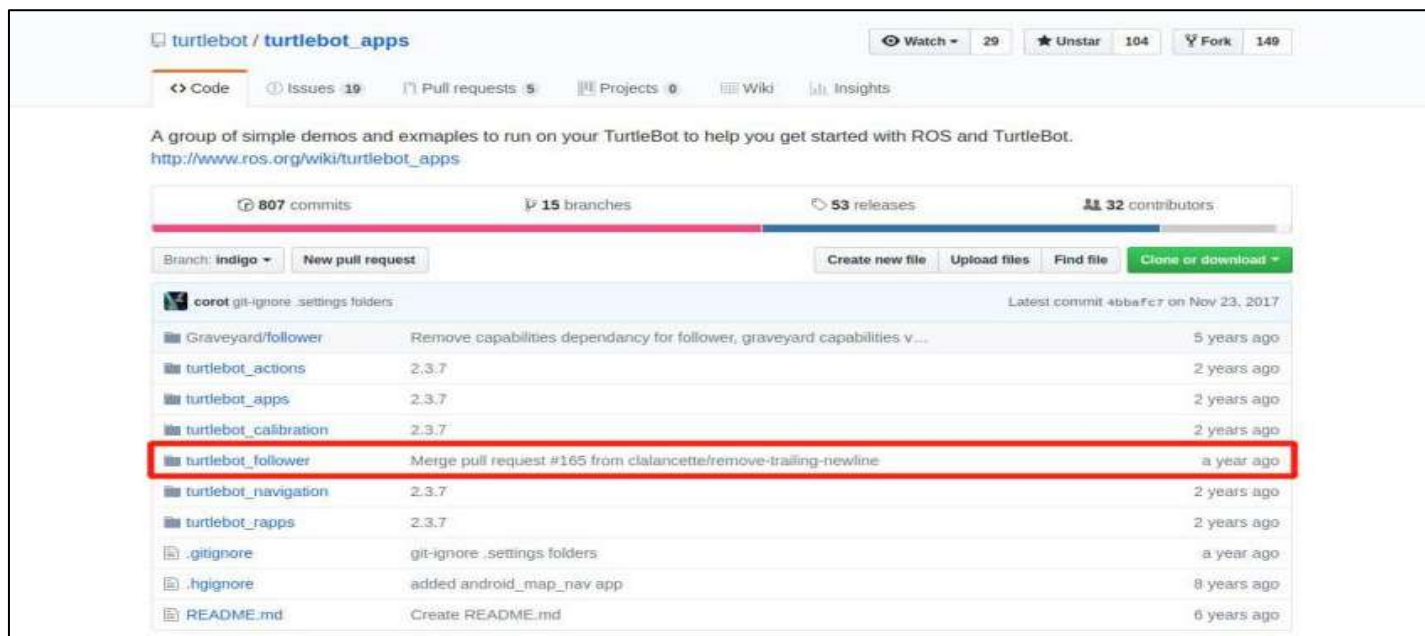
- ✓访问GitHub，在搜索框输入“**Linphone**”
- ✓点击进入“Linphone”查看开源软件项目
- ✓下载和获取“Linphone”开源代码



示例：搜寻和重用软件资产(2)

□机器人的自主跟随功能

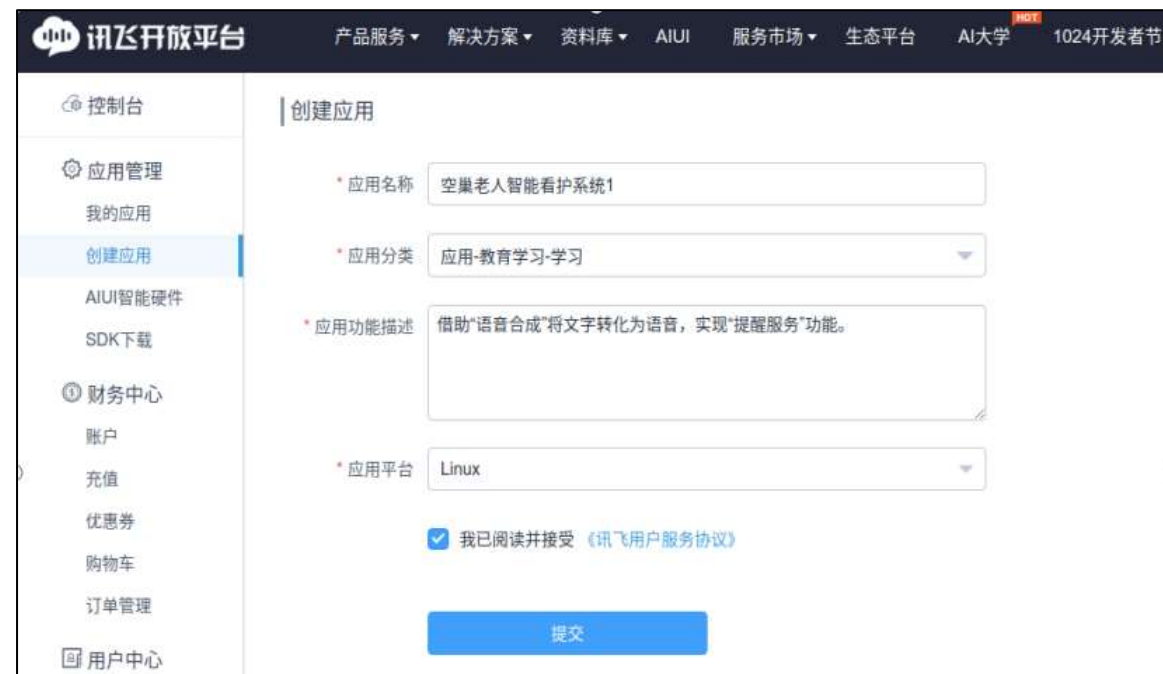
- ✓ 搜寻和重用ROS代码 “turtlebot_follower” 来实现
- ✓ 在Github搜索 “turtlebot app”
- ✓ 点击 “turtlebot_follower” ，阅读和下载其开源代码



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



讯飞开放平台

产品服务 ▾ 解决方案 ▾ 资料库 ▾ AIUI 服务市场 ▾ 生态平台 AI大学 1024开发者节

控制台

应用管理

我的应用

创建应用

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

用户中心

创建应用

* 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

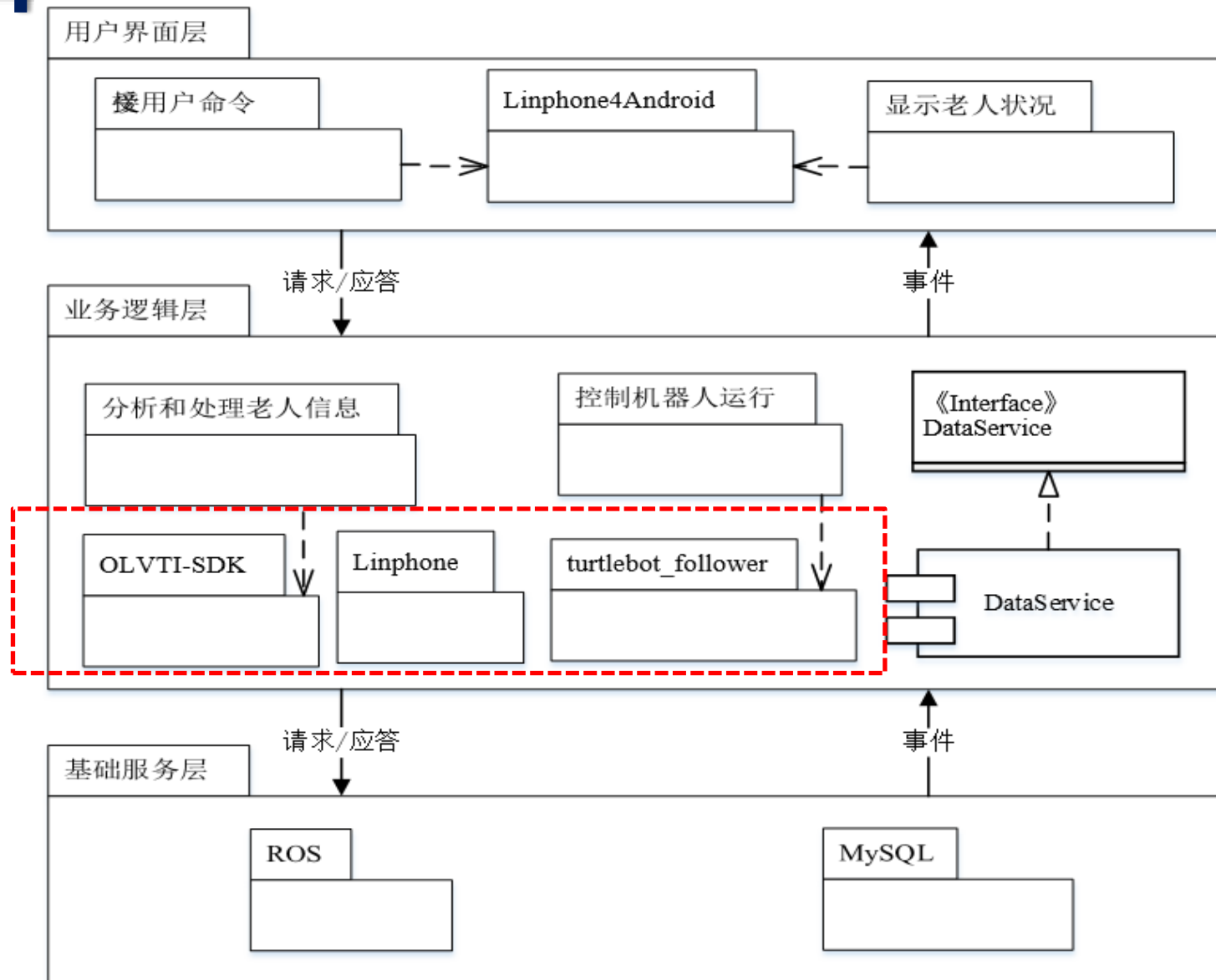
* 应用功能描述 借助“语音合成”将文字转化为语音，实现“提醒服务”功能。

* 应用平台 Linux

☒ 我已阅读并接受 [《讯飞用户服务协议》](#)

提交

示例：空巢老人看护软件的开源软件重用



2.3.3 精化软件体系结构

□选择软件体系结构所依赖的公共基础设施

- ✓确定其中的基础性服务，从而为目标软件系统的运行提供基础性的技术支撑，如操作系统、软件中间件、数据库管理系统、软件开发框架、安全服务等

□确定软件体系结构中的设计元素

- ✓包括子系统、软构件和关键设计类等，明确其职责和接口，从而为开展详细设计奠定基础



确定公共基础设施及服务

□软件体系结构中的各个要素（如软构件、子系统等）都需要依赖于特定的基础设施来运行

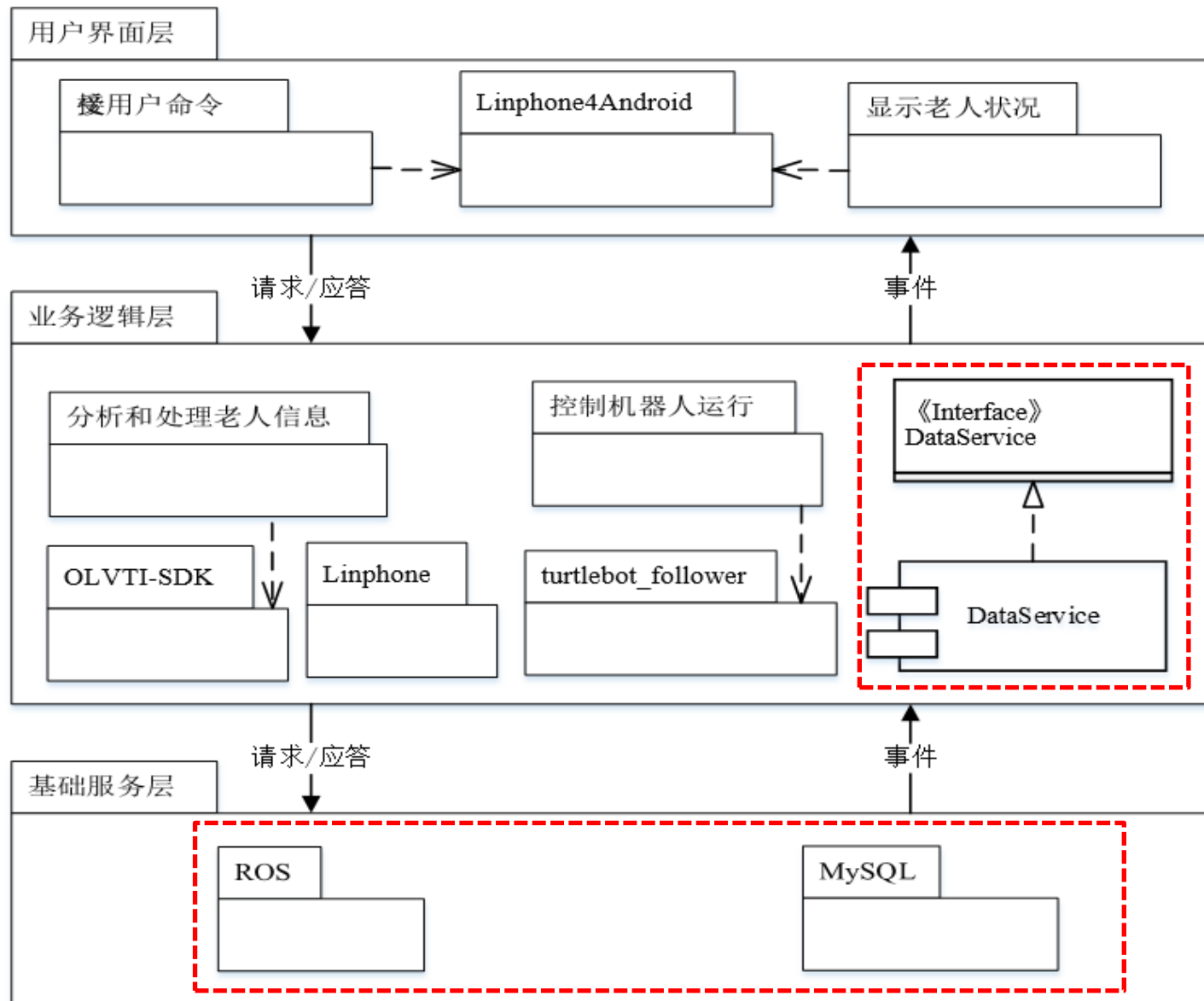
- ✓从最底层的操作系统，到稍高层次的软件中间件、软件开发框架等等，或者表现为诸如数据库管理系统、消息中间件等形式
- ✓为目标软件系统的运行提供基础性的技术支持，而且还为目标软件系统的构造提供可重用的基础服务

设计其所需的基础服务

□ 结合软件需求以及基础设施提供的功能及接口，开展基础服务的设计

- ✓ 如数据持久服务、隐私保护服务、安全控制服务、消息通讯服务等等
- ✓ 基础服务应具有良好的稳定性，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：全巢老人看护软件 的基础设施



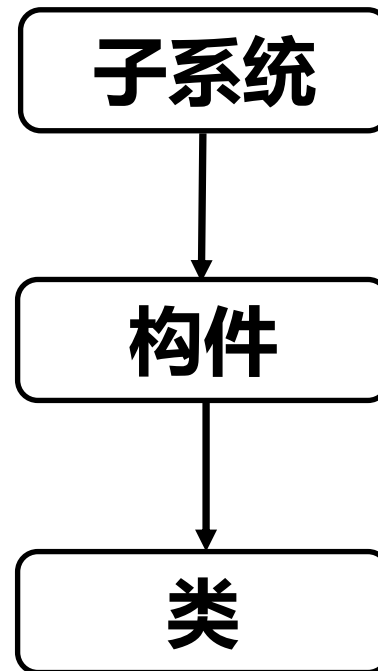
数据持久存储

确立设计元素

- 基于初步软件体系结构，以实现软件需求为目标，**精化和确立设计元素，明确设计元素的职责划分及相互间的协同**
- **将软件需求中的用例组织为一系列设计元素**

□ 设计元素

- ✓ 三类：子系统、构件和类
- ✓ 设计元素的接口及相互间的协作



(1) 确定子系统及其接口

□ 遵循问题分解和系统组织思想，将分析类按相关性原则加以组织和归类，形成软件系统的若干子系统

- ✓ **用例相关性原则**：将用例按业务相关性或相似性进行分组，每组用例组成一个子系统，参与这些用例的实现的分析类均成为此子系统的“设计类”
- ✓ **实现途径相关性原则**：分析用例的交互图，将具有相关或相似业务处理职责的控制类归为一个子系统；或将所有这些控制类的职责归并后按某种业务上的相关性或相似性进行分组，每组职责归为一个子系统
- ✓ **实体类相关性**：从便于管理和控制的角度，将分析模型中的实体类进行分组，每组对应于一个子系统，由子系统对所属的实体类加以管理

评估和改进所确立的子系统

- 每个子系统有明确、独立的**职责**
- 不同子系统间的职责是**正交**的，不应具有相同或相似职责
- 所有子系统职责**覆盖**软件系统所有职责
- 避免特别**庞大或特别细小**的子系统
- 子系统**接口极小化**，仅公开为外界使用该子系统所必需的接口函数，尽可能隐藏内部实现细节

确定子系统的接口

□ 每个子系统需要提供二类接口与外界进行交互

✓ **服务提供接口**：支持外部设计元素访问子系统

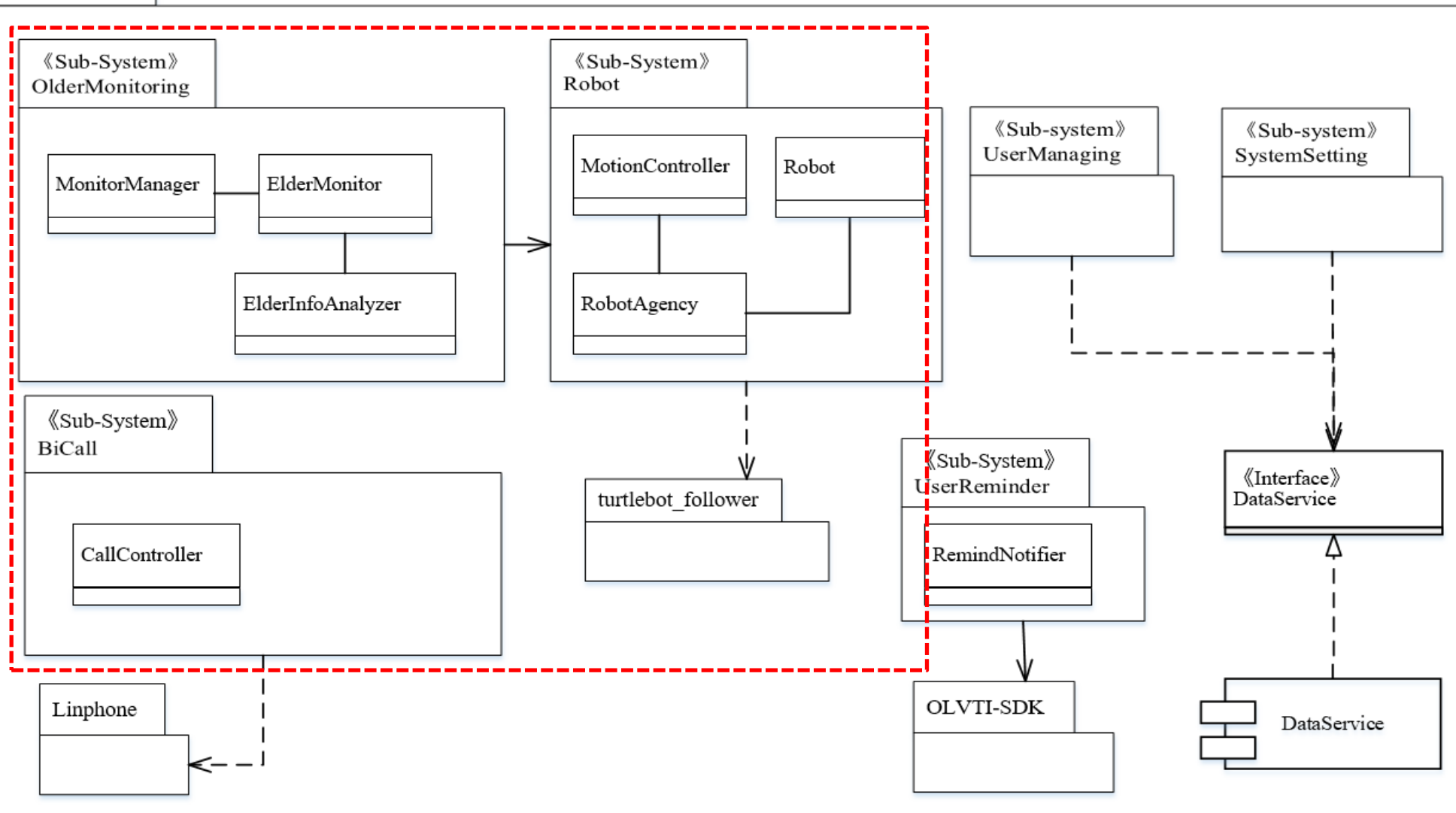
✓ **服务请求接口**：支持子系统访问外部设计元素以完成子系统职责

□ 一个子系统可根据业务处理的需要，定义一个或者多个服务提供接口和服务请求接口



示例：确定“空巢老人智能看护系统”子系统

业务逻辑层



子系统
OlderMonitoring
Robot
BiCall
UserReminder
.....

(2) 确定构件及其接口

□ 将一些分析类的相似或相关职责封装为一个**构件**，通过**接**

口向软件系统的其他部分提供功能和服务

- ✓ 构件的内聚度应该高于子系统
- ✓ 构件的设计应该追求可复用性目标

□ 构件的规模小于子系统

- ✓ 子系统中可以包含构件，但构件中不会包含子系统

(3) 确定关键设计类及其接口

□关键设计类

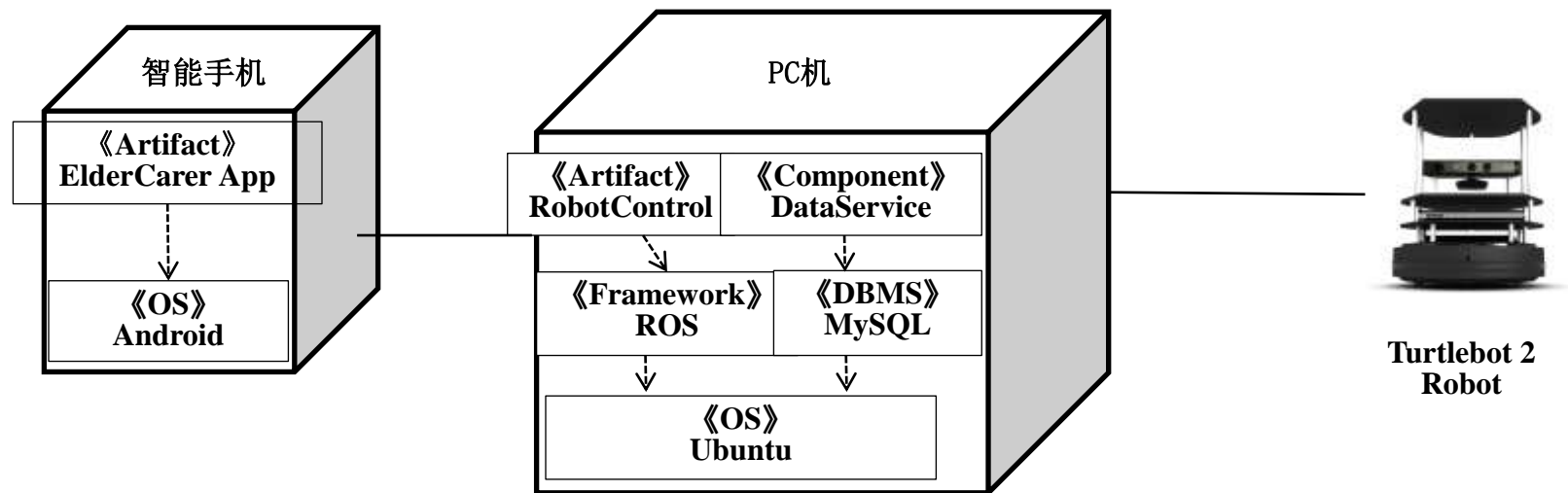
- ✓对于软件需求的实现具有比较重要的作用，但未归入子系统和构件的某项职责
- ✓与子系统或构件交互，或者作为软件体系结构中已有的设计元素之间的交互“桥梁”，缺少这种交互将导致体系结构无法完整地实现某项重要的软件需求

□所谓关键是指重要、不可或缺

2.3.4 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、软构件如何部署到计算节点上运行的，描述它们的部署和运行环境



2.4 整合体系结构设计

□整合和组织体系结构设计产生的设计结果

- ✓包括引入的开源软件 and 软件资产、顶层软件架构、确定的各类设计元素等等

□整合的目的

- ✓理清设计元素间的关系，明确它们之间的交互和协作
- ✓确保所产生的软件设计模型具有更好的模块性、封装性、可重用性等特征

□最终获得目标软件系统体系结构的完整逻辑视图

整合体系结构设计

□整合外部系统对应的子系统与分析模型中负责与外部系统交互的边界类

- ✓子系统是服务提供方，边界类是服务请求方

□整合引入的开源软件或者软件资产

- ✓必要时引入相关的接口以实现开源软件或者软件资产与设计元素之间的交互，从而它们与软件系统的设计融合

□整合外部系统与目标软件系统中的设计类

- ✓进一步清晰地定义相关的接口，支持它们之间的协作

优化体系结构设计

- 分解一些规模较大的子系统
- 合并若干职责相同或相似的设计元素
- 以软件重用为目的适当的调整设计元素所封装的功能、承担的职责和对外提供的接口
- 用简洁的软件元素，以尽可能高效、合理的方式实现所有软件需求
- 将不同设计元素中的公共职责提取成为服务构件或服务类

整合体系结构设计的结果

- 设计元素的职责划分更为明确
- 系统与外部接口、相关设计元素间的接口和协作更为清晰
- 整体体系结构设计方案更为优化



内容

1. 何为软件体系结构

- ✓ 概念、视图、模型及UML表示
- ✓ 软件体系结构风格

2. 软件体系结构设计

- ✓ 任务、目标、要求和原则
- ✓ 体系结构设计过程

3. 文档化和评审软件体系结构设计

- ✓ 文档模板、验证原则





撰写软件体系结构设计文档

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. 逻辑视图的体系结构设计
6. 部署视图的体系结构设计
7. 开发视图的体系结构设计
8. 运行视图的体系结构设计

评审软件体系结构设计

□ 满足性

- ✓ 体系结构是否能够满足软件需求，体系结构怎样满足软件需求

□ 优化性

- ✓ 体系结构是否以充分优化方式实现所有软件需求项

□ 可扩展性

- ✓ 是否易于扩展，以应对软件需求的变化

□ 可追踪性

- ✓ 软件体系结构中的所有设计元素是否有相对应的软件需求项

□ 详尽程度

- ✓ 体系结构的详略程度是否恰当

□软件模型

- ✓用UML包图、部署图等描述的软件体系结构模型

□软件文档

- ✓体系结构设计规格说明书文档

□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、部署图来表示

□软件体系结构设计的重要性

- ✓针对软件系统全局性、基础性技术问题给出技术解决方案

□软件体系结构的风格

- ✓管道、层次、MVC、黑板等等，针对不同软件需求及特点

□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
- ✓软件体系结构的设计模型和文档