



基于体系结构的软件开发

lliao@seu.edu.cn

目录

- 设计模式
- 中间件技术
- 基于体系结构的设计方法
- 体系结构的设计和演化
- 基于体系结构的软件开发模型
- 基于体系结构的软件过程
- 软件体系结构演化模型

设计模式概述

◎ 模式是指从某个具体的形式中得到的一种抽象，在特殊的非任意性环境中，该形式不断地重复出现。

◎ 模式是前人经验的总结，它使人们可以方便地复用成功的设计和体系结构。

◎ 当人们在特定的环境下遇到特定类型的问题，采用他人已使用过的一些成功解决方案，一方面可以降低分析、设计和实现的难度，另一方面可以使系统具有更好的可复用性和灵活性。

设计模式概述

- 模式是给定上下文中普遍问题的普遍解决方案，在软件开发方面分为：
 - 体系结构模式（风格）
 - 设计模式
 - 惯用法
- 前两者与程序开发语言无关。

设计模式概述

- 一个软件体系结构的模式描述了一个出现在特定设计语境中的特殊的再现设计问题，并为它的解决方案提供了一个经过充分验证的通用图示。
- 解决方案图示通过描述其组成构件及其责任和相互关系以及它们的协作方式来具体指定。

设计模式概述

- ◎ 软件系统的基本组成单元是模块、类等，但单个的模块或类并没有什么意义，不能提供应用的解决方案。
- ◎ 往往需要多个模块、类、对象、服务、进程、线程、构件等共同协作，提供应用问题的解决方案。
- ◎ 设计模式可以方便地重用成功的设计和结构，还提供了类和对象接口的明确的说明书和这些接口的潜在意义。
- ◎ 模式的实现要比解决方案复杂得多。可以使用各种语言来实现。

设计模式概述

◎ 以MVC为例

- 软件系统的重要功能之一是从数据存储（文件/数据库）中检索数据，并将其显示给用户。在用户更改数据之后，系统再将更新内容存储到数据存储中。如果将数据存储和界面显示代码放在一起，则可以减少编码量并提高应用程序性能。
- 但这样做也存在很多问题。

设计模式概述

◎ MVC的背景

➤用户界面逻辑的更改往往比业务逻辑频繁，尤其是在基于 Web 的应用程序中。基于 Web 的瘦客户端应用程序的优点之一是可以随时更改用户界面，而不必重新分发应用程序。如果将显示代码和业务逻辑组合在一起并放在单个对象中，则每次更改用户界面时，都必须修改包含业务逻辑的对象。

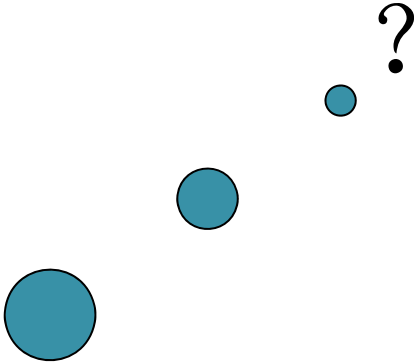
设计模式概述

◎ MVC的背景

- 在很多情况下，应用程序需要以不同的方式显示同一数据。例如，表格显示，趋势图显示，柱状图显示，饼图显示。
- 界面开发与业务逻辑开发所需要的技能不同，在这两方面都精通比较困难。
- 与业务逻辑相比，用户界面代码对设备的依赖性往往更大。例如，要将应用程序从基于浏览器的应用程序迁移到个人数字助理（PDA）或支持 Web 的手机上，则必须替换很多用户界面代码，而业务逻辑可能不受影响。

设计模式概述

◎ MVC的背景



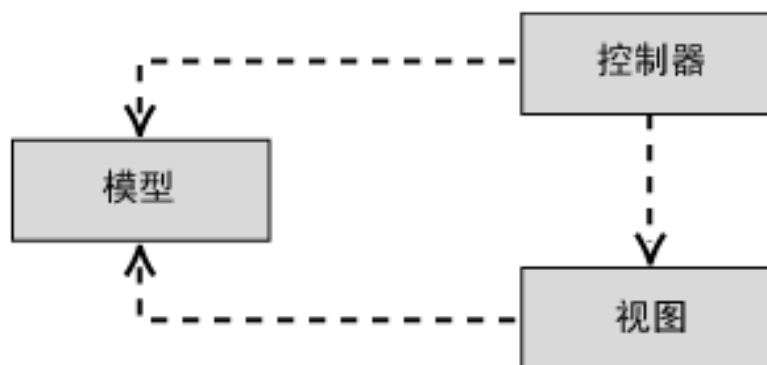
如何让 Web 应用程序的用户界面和功能实现模块化，以便可以轻松地单独修改各个部分？

设计模式概述

◎ MVC解决方案

MVC模式将建模、显示和操作分为三种独立的类：

- **模型**。模型用于管理应用程序域的行为和数据，并响应为获取其状态信息（通常来自视图）而发出的请求，还会响应更改状态的指令（通常来自控制器）。
- **视图**。视图用于管理信息的显示。
- **控制器**。控制器用于解释用户的鼠标和键盘输入，以通知模型和/或视图进行相应的更改。



设计模式概述

◎ MVC的处理过程

- (1) 控制器接收用户的请求，并决定应该调用哪个模型来进行处理；
- (2) 模型用业务逻辑来处理用户的请求并返回数据；
- (3) 控制器用相应的视图格式化模型返回的数据，并通过表示层呈现给用户。

设计模式概述

◎ MVC模式的优点

➤支持多个视图。因为视图与模型分离，而且模型与视图之间没有直接依赖性，所以用户界面可以同时显示同一数据的多个视图。

➤适应更改。用户界面要求的更改往往比业务规则快。用户可能更喜欢新设备（如手机或 PDA）采用另一颜色、字体、屏幕布局和支持级别。因为模型不依赖于视图，所以将新类型的视图添加到系统中通常不会影响模型。因此，更改的作用范围仅限于视图。

设计模式概述

◎ MVC模式的缺点

- **复杂性**。MVC 模式引入了新的间接级别，因此增加了解决方案的复杂性。还增加了用户界面代码的事件驱动特性，调试用户界面代码会变得更加困难。
- **频繁更新的成本**。将模型与视图分离并不意味着模型的开发人员可以忽略视图的特性。例如，如果模型发生频繁更改，则它可能向视图发出大量更新请求。一些视图（如图形显示）的显示可能需要一定时间。

设计模式的基本成分

◎ 模式名称：通常用来描述一个设计问题、它的解法和后果，由一到两个词组成。

◎ 问题：告诉我们什么时候要使用设计模式、解释问题及其背景。

例如：MVC模式关心用户界面经常变化的问题。它必须易于修改用户界面，但软件的功能核心不能被这种修改影响。

设计模式的基本成分

◎ 解决方案：描述设计的基本要素，它们的关系、各自的任务以及相互之间的合作。

例如：MVC模式中控制器接收输入，输入往往是鼠标移动、单击鼠标按键或键盘输入等事件。事件转换成服务请求，这些请求再发送给模型或视图。

◎ 效果：描述应用设计模式后的结果和权衡。比较与其他设计方法的异同，得到应用设计模式的代价和优点。

例如：不同方案的时间和空间权衡；方案的可重用性、灵活性、可扩展性、可移植性等。

设计模式的描述

◎ 如果要理解和讨论模式，就必须以适当形式描述模式。

◎ 好的描述有助于我们立即抓住模式的本质，即模式关心的问题是什么，以及提出的解决方案是什么？

◎ 模式也应该以统一方式来描述

设计模式的描述

Erich Gamma博士等人采用下面的固定模式来描述：

- (1) **模式名称和分类**：模式名称和一个简短的摘要。
- (2) **目的**：即设计模式的用处、基本原理和目的、它针对的是什么样的设计问题。
- (3) **别名**：同一个模式可能会有不同的命名。
- (4) **动机**：描述一个设计问题的方案，以及模式中类和对象的结构如何解决这个问题。

设计模式的描述

- (5) **应用**：在什么情况下可以应用本设计模式。
- (6) **结构**：用对象模型技术对本模式进行表示。
- (7) **成分**：组成本设计模式的类和对象及它们的职责。
- (8) **合作**：成分间如何合作实现他们的任务。
- (9) **效果**：该模式如何支持它的对象；如何在使用的本模式时进行权衡。

设计模式的描述

(10) **实现**：在实现本模式的过程中，要注意哪些缺陷、线索或者技术；是否与编程语言有关。

(11) **例程代码**：说明如何用C++或其他语言来实现该模块的代码段。

(12) **已知的应用**：现实系统中使用该模式的实例。

(13) **相关模式**：与本模式相关的其他模式，它们之间的区别，以及本模式是否要和其他模式共同使用。

模式和软件体系结构

◎ 模式作为体系结构构造块

- 对软件体系结构而言，模式的一个重要目标就是用已定义属性进行特定的软件体系结构的构造。
- 软件体系结构的一般技术并没有针对特定问题的解决方案。
- 模式使用特定的**面向问题的技术**来有效补充这些通用的与问题无关的体系结构技术。

模式和软件体系结构

◎ 构造异构体系结构

- 单个模式不能完成一个完整的软件体系结构的详细构造，它仅仅帮助设计师设计应用程序的某一方面。
- 为了有效使用模式，需要将它们组织成模式系统。模式系统统一描述模式，对它们分类，更重要的是，说明它们之间如何交互。

模式和软件体系结构

◎ 模式和方法

- 好的模式描述应包含实现指南，可将其看成是一种微方法，用来创建解决一个问题的方案。
- 通过提供方法的步骤来解决软件开发中的具体再现问题，这些微方法补充了通用的但与问题无关的分析和设计方法。

模式和软件体系结构

◎ 实现模式

➤ 目前的许多软件模式具有独特的面向对象风格。因此，人们往往认为，能够有效实现模式的唯一方式是使用面向对象编程语言，其实不然。

➤ 在设计层次，大多数模式只需要适当的编程语言的抽象机制，如模块或数据抽象。因此，可以用几乎所有的编程范例，并在所有的编程语言中来实现模式。

设计模式方法分类

- ◎ Coad的面向对象模式
- ◎ 代码模式
- ◎ 框架应用模式
- ◎ 形式合约

设计模式方法分类

1、Coad的面向对象模式

1992年，美国面向对象技术大师Peter Coad从MVC的角度对面向对象系统进行了讨论，设计模式由最底层的构成部分（类和对象）及其关系来区分。他使用了一种通用的方式来描述一种设计模式：

- （1） 模式所能解决问题的简要介绍与讨论；
- （2） 模式的非形式文本描述以及图形表示；
- （3） 模式的使用方针：在何时使用以及能够与哪些模式结合使用。

设计模式方法分类

1、Coad的面向对象模式

将Coad的模式划分为以下三类：

（1）**基本的继承和交互模式**：主要包括OOPL所提供的基本建模功能，继承模式声明了一个类能够在其子类中被修改或被补充，交互模式描述了在有多个类的情况下消息的传递。

（2）**面向对象软件系统的结构化模式**：描述了在适当情况下，一组类如何支持面向对象软件系统结构的建模。主要包括条目（item）描述模式、为角色变动服务的设计模式和处理对象集合的模式。

设计模式方法分类

1、Coad的面向对象模式

(3) 与MVC框架相关的模式。几乎所有Coad提出的模式都指明如何构造面向对象软件系统，有助于设计单个的或者一小组构件，描述了MVC框架的各个方面。但是，他没有重视抽象类和框架，没有说明如何改造框架。

设计模式方法分类

2、代码模式

代码模式的抽象方式与OOPPL中的代码规范很相似，该类模式有助于解决某种面向对象程序设计语言中的特定问题。

主要目标在于：

- (1) 指明结合基本语言概念的可用方式；
- (2) 构成源码结构与命名规范的基础；
- (3) 避免面向对象程序设计语言（尤其是C++语言）的缺陷。

设计模式方法分类

2、代码模式

代码模式与具体的程序设计语言或者类库有关，它们主要从语法的角度对软件系统的结构方面提供一些基本的规范。

这些模式对于类的设计不适用，同时也不支持程序员开发和应用框架，命名规范是类库中的名字标准化的基本方法，以免在使用类库时产生混淆。

设计模式方法分类

3、框架应用模式

- 框架是从特定域中提取出来的一组组件及其相互关系的可重用的体系结构。它定义了整体结构、类和对象的分割，各部分的主要责任，类和对象如何协作，以及控制流程。
- 框架记录了其应用领域共同的设计决策，因而框架更强调设计复用。
- 程序员将框架作为应用程序开发的基础，特定的框架适用于特定的需求。

设计模式方法分类

4、形式合约

- 形式合约(formal contracts)也是一种描述框架设计的方法，强调组成框架的对象间的交互关系。
- 有人认为它是面向交互的设计，对其他方法的发展有启迪作用。
- 形式化方法由于其过于抽象，而有很大的局限性，仅仅在小规模程序中使用。

设计模式的分类

Gamma和他的同事已发布了可用于OO系统的一系列设计模式，用一种类似分类目录的形式将设计模式记载下来。我们称这些设计模式为设计模式目录。

根据模式的目标，可以将它们分成创建型模式、结构型模式和行为型模式。创建型模式处理的是对象的创建过程，结构型模式处理的是对象/类的组合，行为型模式处理类和对象间的交互方式和任务分布。

根据它们主要的应用对象，又可以分为主要应用于类的和主要应用于对象的。

设计模式的分类

1. 创建型模式

- 创建型模式对类的实例化过程进行了抽象，能够使软件模块做到与对象的创建和组织无关。
- 创建型模式隐藏了对象是如何被创建和组合在一起的，已达到使整个系统独立的目的。
- 创建型模式包括：工厂方法（**factory method**）模式、抽象工厂（**abstract factory**）模式、原型（**prototype**）模式、单例（**singleton**）模式和建造者（**builder**）模式

设计模式的分类

2. 结构型模式

- 结构型模式描述如何将类或对象结合在一起构成更大的结构。
- 根据描述事物的不同，可以分为类结构型模式和对象结构型模式。
- 结构型模式包括：适配器（**adapter**）模式、桥接（**bridge**）模式、组合（**composite**）模式、装饰（**decorator**）模式、外观（**facade**）模式、享元（**flyweight**）模式和代理（**proxy**）模式

设计模式的分类

3. 行为型模式

- 行为型模式是对在不同的对象之间划分责任和算法的抽象化，它不仅关于类和对象的，而且是关于它们之间的相互作用的。
- 根据描述事物的不同，可以分为类行为模式和对象行为模式。
- 行为型模式包括：职责链模式、命令模式、解释器模式、迭代器模式、中介者模式、备忘录模式、观察者、状态模式、策略模式、模板方法模式和访问者模式等。

中间件技术

- 随着信息化建设的逐步进行，一个企业可能同时运行着多个不同的业务系统，分别基于不同的操作系统、数据库和异构的网络环境。
- 对每个应用系统而言，在设计和开发时不仅要关心业务逻辑，还必须要处理分布式环境中复杂的通信和异构问题，为此出现了中间件。

中间件技术

- 中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。
- 中间件位于操作系统之上，管理计算资源和网络通信，实现应用之间的互操作。
- 例如：CORBA ， J2EE等

中间件技术

➤ 中间件的基本功能：

- 负责客户机和服务器之间的连接和通信，以及客户机和应用层之间的高效率通信机制；
- 提供应用层不同服务之间的互操作机制，以及应用层和数据库之间的连接和控制机制；
- 提供一个多层体系结构的应用开发和运行的平台，支持模块化应用开发；
- 屏蔽硬件、操作系统、网络和数据库差异；
- 提供应用的负载均衡和高可用性、安全机制和管理功能；
- 提供一组通用的服务，避免重复的工作和使应用之间可以协作。

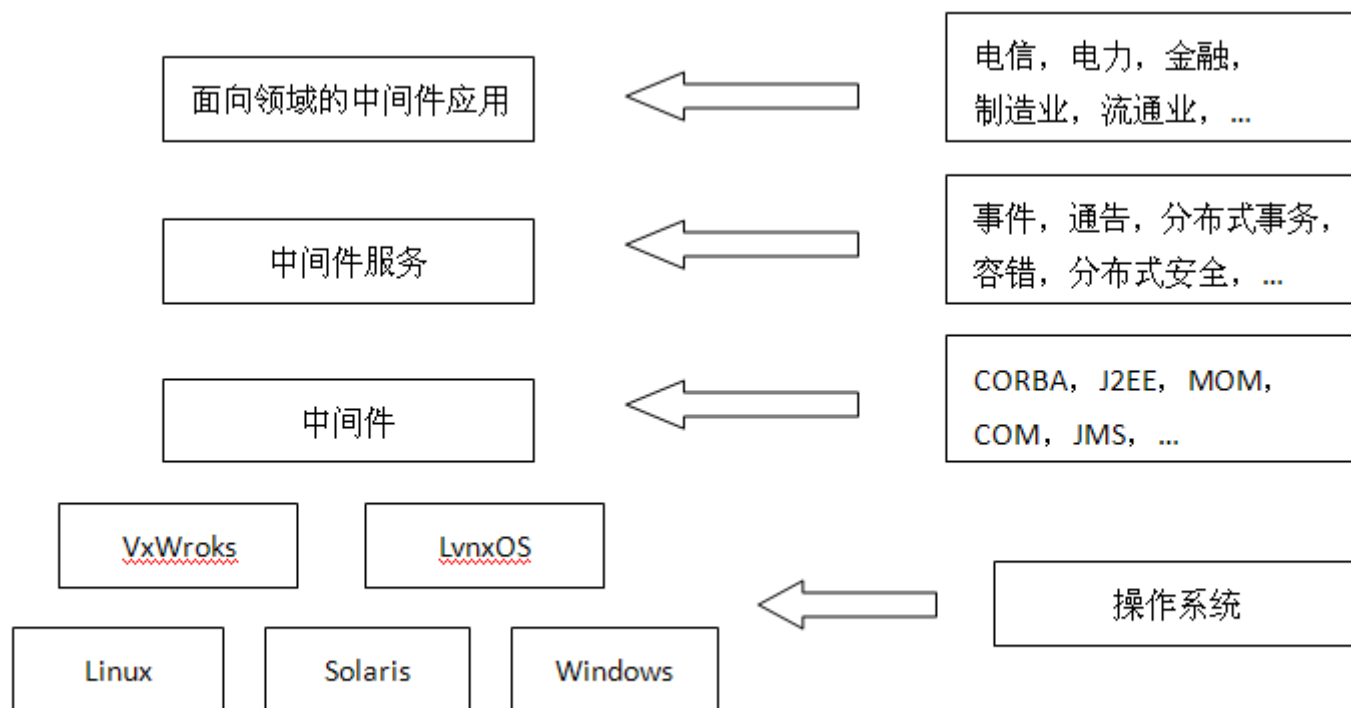
中间件技术

➤ 中间件的分类

- ✓ 采用自底向上的方式来划分，可分为底层中间件、通用型中间件和集成型中间件三个层次。
- ✓ 底层中间件主流技术：JVM（Java虚拟机）、CLR（公共语言运行库）、ACE（自适应通信环境）
- ✓ 通用型中间件主流技术：RPC（远程过程调用）、ORB（对象请求代理）
- ✓ 集成型中间件主流技术：Workflow，EAI（企业应用集成）

中间件技术

➤ 中间件的应用：中间件提供了应用系统基本的运行环境，也为应用系统提供了更多的高级服务功能。



中间件技术应用层次图

中间件技术

➤ 中间件的应用



不同层次的集成示意图

中间件技术

➤ 中间件的发展趋势

- ✓ 规范化。规范化的建立极大地促进了中间件技术的发展，同时保证了系统的扩展性、开放性和互操作性。
- ✓ 构件化和松耦合。除了已经得到广泛应用的CORBA、DCOM等适应Intranet的构件技术外，随着企业业务流程整合和电子商务应用的发展，中间件技术朝着面向Web和松散耦合的方向发展。
- ✓ 平台化。一些大的中间件厂商在已有的中间件产品基础上，提出了完整的面向互联网的软件平台战略计划和应用解决方案。

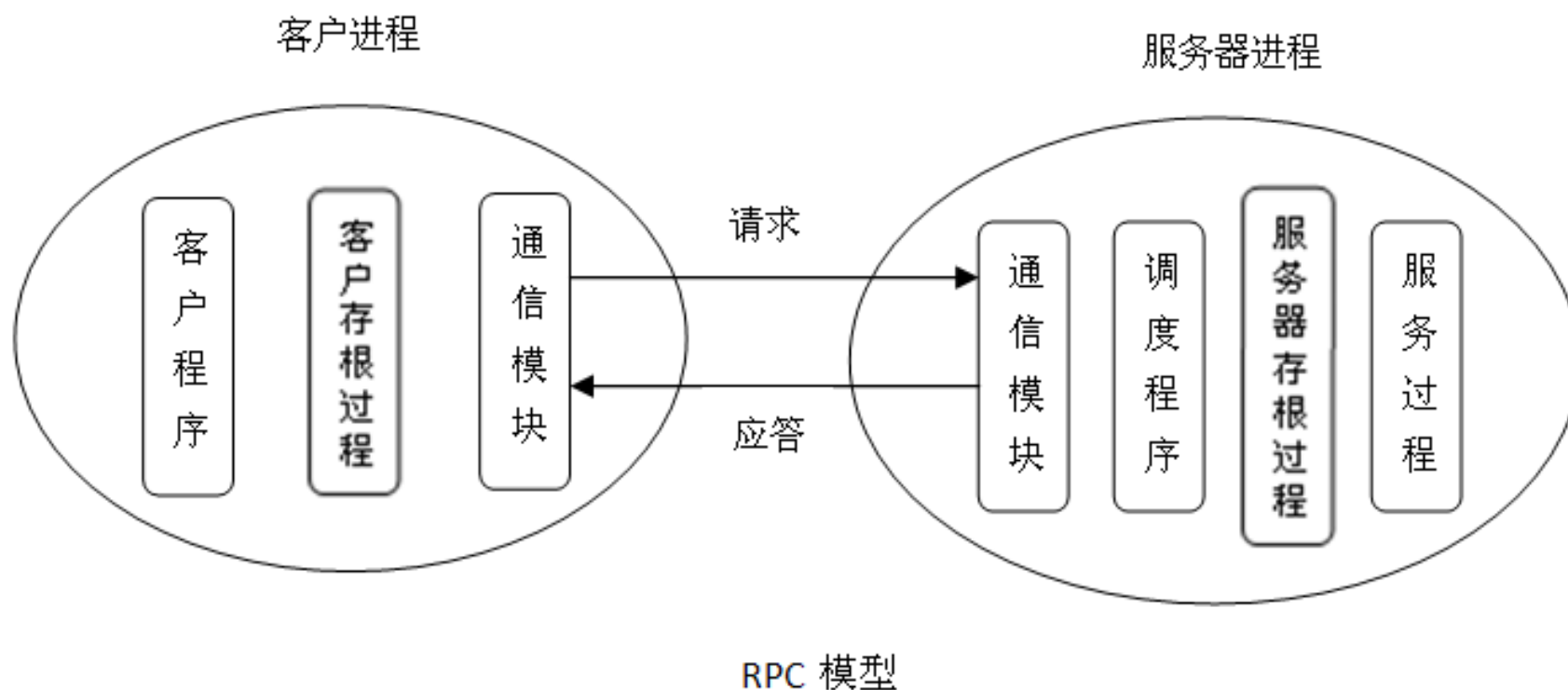
中间件技术

➤主要的中间件

✓远程过程调用：

- ✓RPC是一种广泛使用的分布式应用程序处理方法。
- ✓应用程序使用RPC来远程执行一个位于不同地址空间里的过程，并且从效果上看和执行本地调用相同。
- ✓一个RPC应用可以分为两个部分，分别是服务器和客户（进程）。
- ✓服务器和客户可以位于同一台计算机，也可以在不同计算机上，它们之间通过网络进行通信。

- ✓通信模块实现请求应答协议
- ✓调度程序根据请求消息中的过程标识选择服务器存根过程
- ✓存根过程支持数据转换和通信服务，屏蔽操作系统和网络协议
- ✓服务过程实现服务接口中的过程



中间件技术

➤ 主要的中间件

✓ 对象请求代理:

✓ 随着对象技术与分布式计算技术的发展，两者相结合形成了分布对象计算。1990年底，OMG首次推出对象管理体系结构（Object Management Architecture, OMA）模型，ORB是这个模型的核心组件。

✓ ORB的作用：提供一个通信框架，透明地在异构的分布式环境中传递对象请求。

中间件技术

➤ 主要的中间件

✓ 远程方法调用：

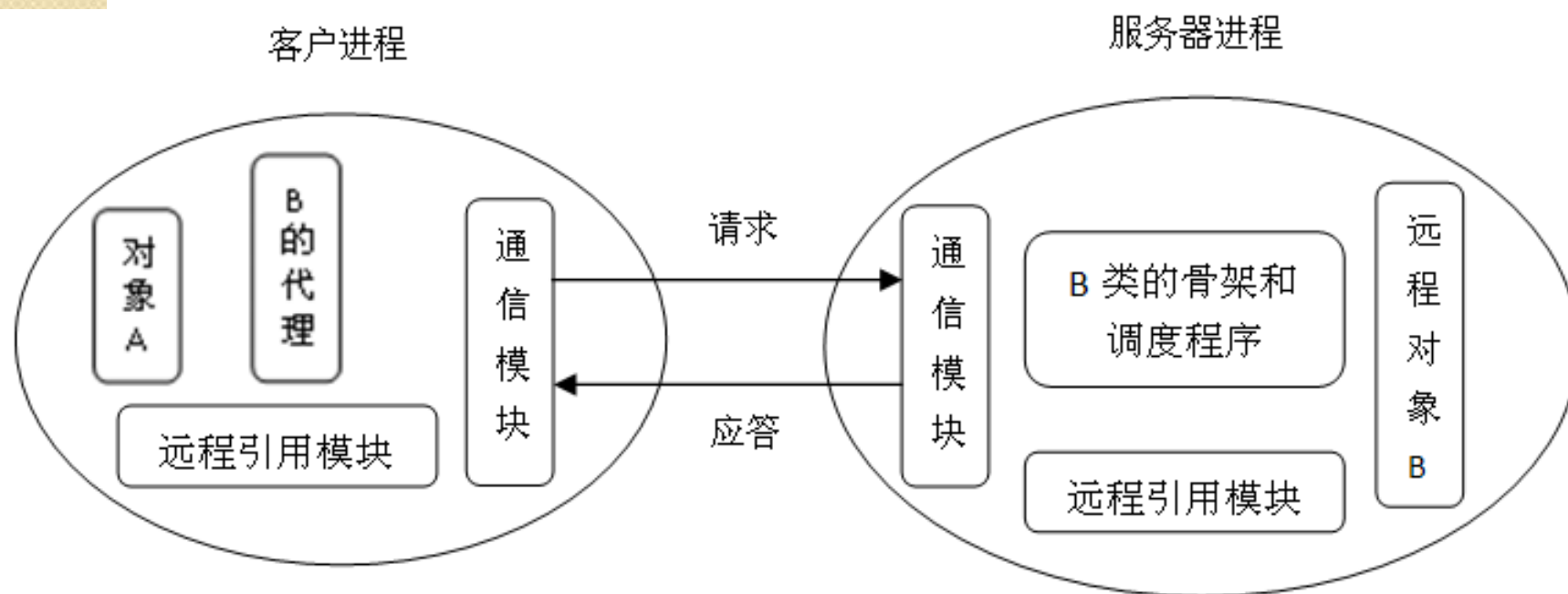
✓ RMI是Java的一组用于开发分布式应用程序的API。RMI使用Java语言接口定义远程对象，它集合了序列化和Java远程方法协议。

✓ RMI通常也包括两个独立的部分：服务器和客户

✓ 服务器：创建多个远程对象，使之能够被引用。

✓ 客户：从服务器中得到一个或多个远程对象的引用，然后调用远程对象的方法。

- ✓通信模块实现请求应答协议
- ✓远程引用模块负责翻译本地和远程对象引用，以及创建远程对象引用。
- ✓代理使得远程方法调用对客户透明。
- ✓调度收集来自通信模块的请求，传递消息，选择骨架中的方法。骨架用于实现调用，传递信息给代理。



RMI 模型

中间件技术

➤ 主要的中间件

- ✓ 面向消息的中间件。MOM指的是利用高效可靠的消息传递机制进行平台无关的数据交换，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，可在分布式环境下扩展进程间的通信，并支持多种通信协议、语言、应用程序、硬件和软件平台。
- ✓ IBM的MQService、BEA的MessageQ等都属于MOM产品。

中间件技术

➤ 中间件与构件的关系

- ✓ 中间件是对分布式应用的抽象。应用在中间件提供的环境中能够更好地集中于业务逻辑，并以构件的形式存在；
- ✓ 中间件与体系结构在本质上是一致的。都是构件存在的基础。

基于体系结构的软件设计方法

- 基于体系结构的软件设计（**architecture-based software design, ABSD**）方法为软件系统的概念体系结构提供构造方法，概念体系结构描述了系统的主要设计元素及其关系。
- 概念体系结构代表了在开发过程中作出的第一个选择，它是达到系统质量和业务目标的关键，为达到预定功能提供了基础。

基于体系结构的软件设计方法

➤ ABSD方法有三个基础：

- (1) 功能分解：在功能分解中，ABSD方法使用已有的基于模块的内聚和耦合技术；
- (2) 通过选择体系结构风格来实现质量和业务需求。
- (3) 软件模板的使用：利用一些软件系统的结构。

基于体系结构的软件设计方法

- 软件模板是一个特殊类型的软件元素，包括描述所有这种类型的元素在共享服务和底层构造的基础上如何进行交互。
- 软件模板还包括属于这种类型的所有元素的功能，这些功能的例子有：每个元素必须记录某些重大事件，每个元素必须为运行期间的外部诊断提供测试点等。

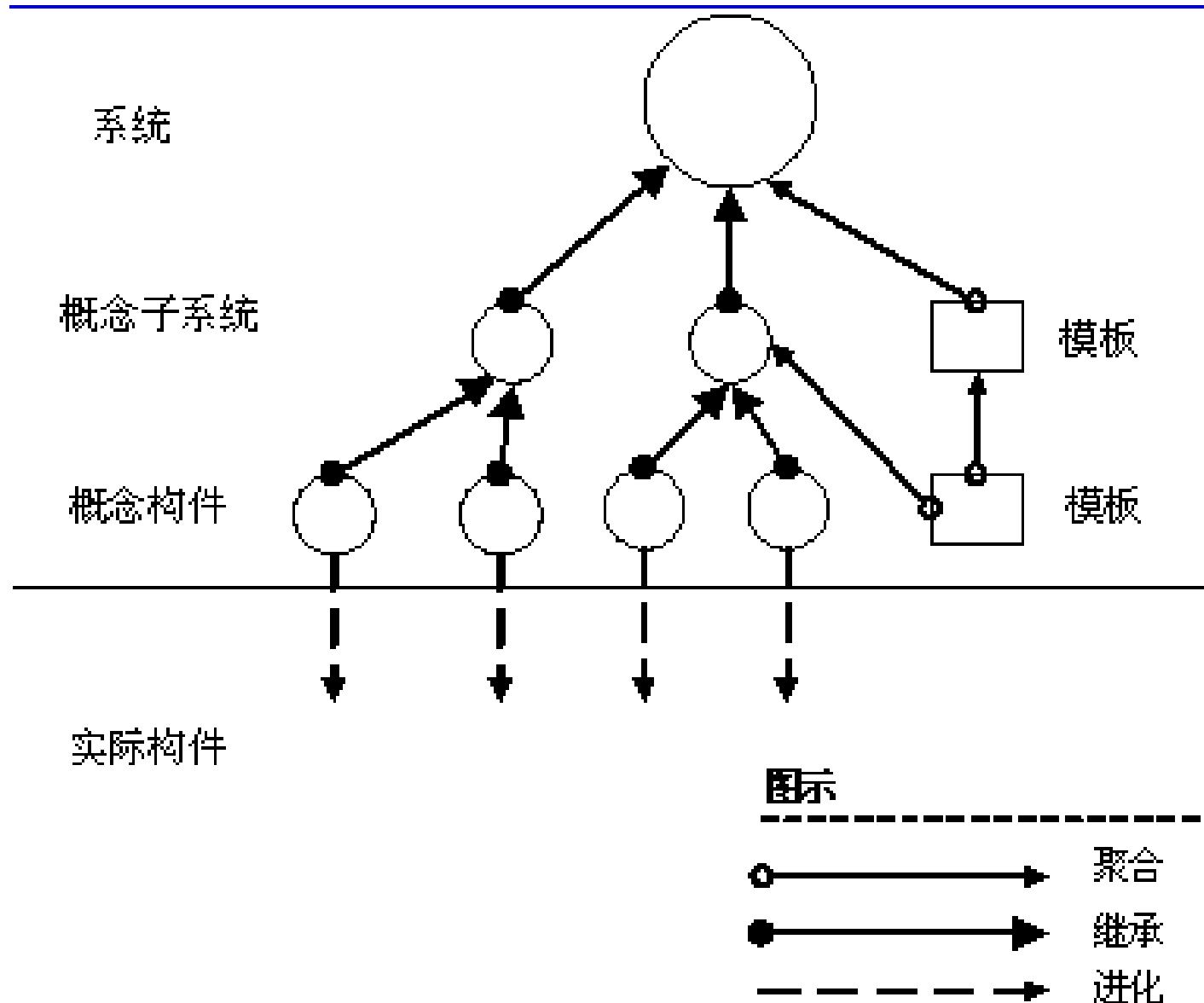
有关术语

1. 设计元素

- ABSD方法的目的是组织最早的设计策略，不包括形成实际的软件构件和类。但要作出有关功能划分和达到不同质量属性机制的决策。
- ABSD方法是一个递归细化的方法，软件系统的体系结构通过该方法得到细化，直到能产生软件构件和类。

有关术语

ABSD方法中所使用的元素。



有关术语

2. 视角和视图

- 考虑体系结构时，需要从不同的视角 (perspective) 来描述系统。如逻辑视图、实现视图（开发视图）、进程视图（并发视图）、部署视图（物理视图）。

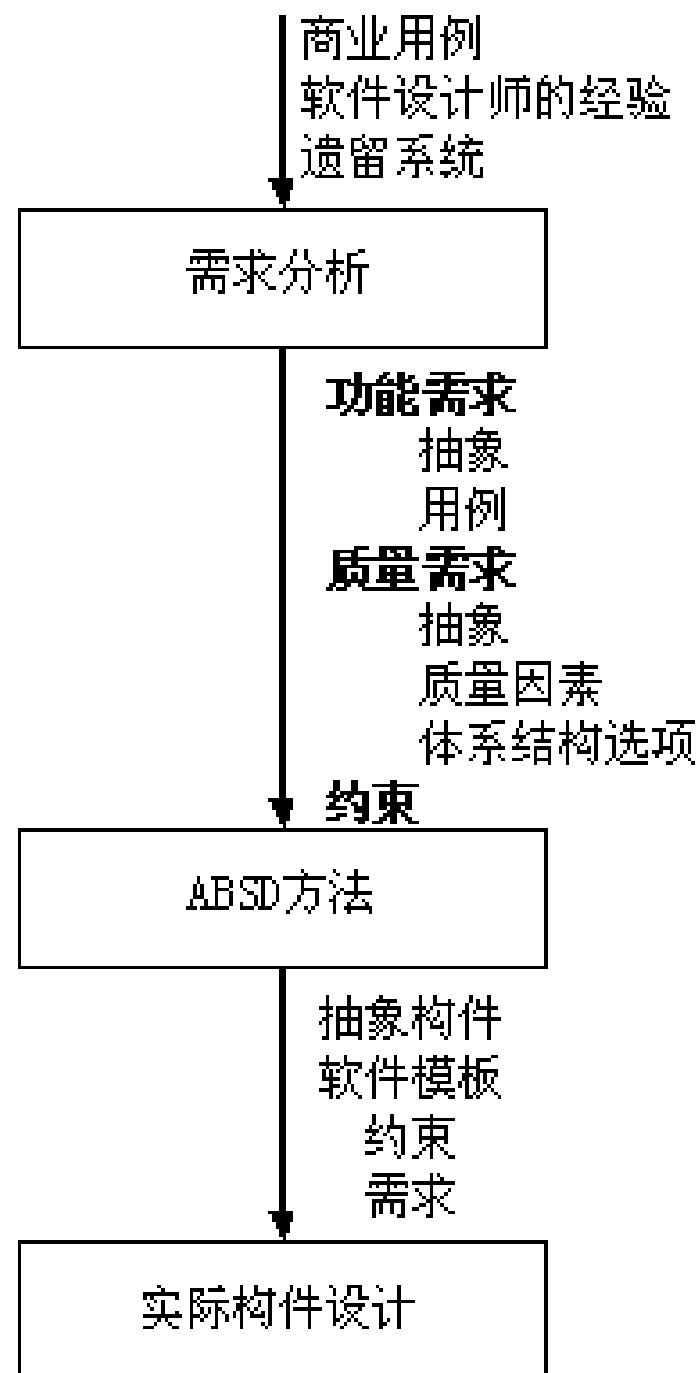
有关术语

3. 用例和质量场景

- 用例除了使功能需求具体化，还必须使质量需求具体化。
- 在使用用例捕获功能需求的同时，我们通过定义特定场景来捕获质量需求，并称这些场景为质量场景。
- 在一般的软件开发过程中，我们使用质量场景捕获变更、性能、可靠性和交互性，分别称为变更场景、性能场景、可靠性场景和交互性场景。

ABSD方法与生命周期

ABSD方法在生命周期中的位置



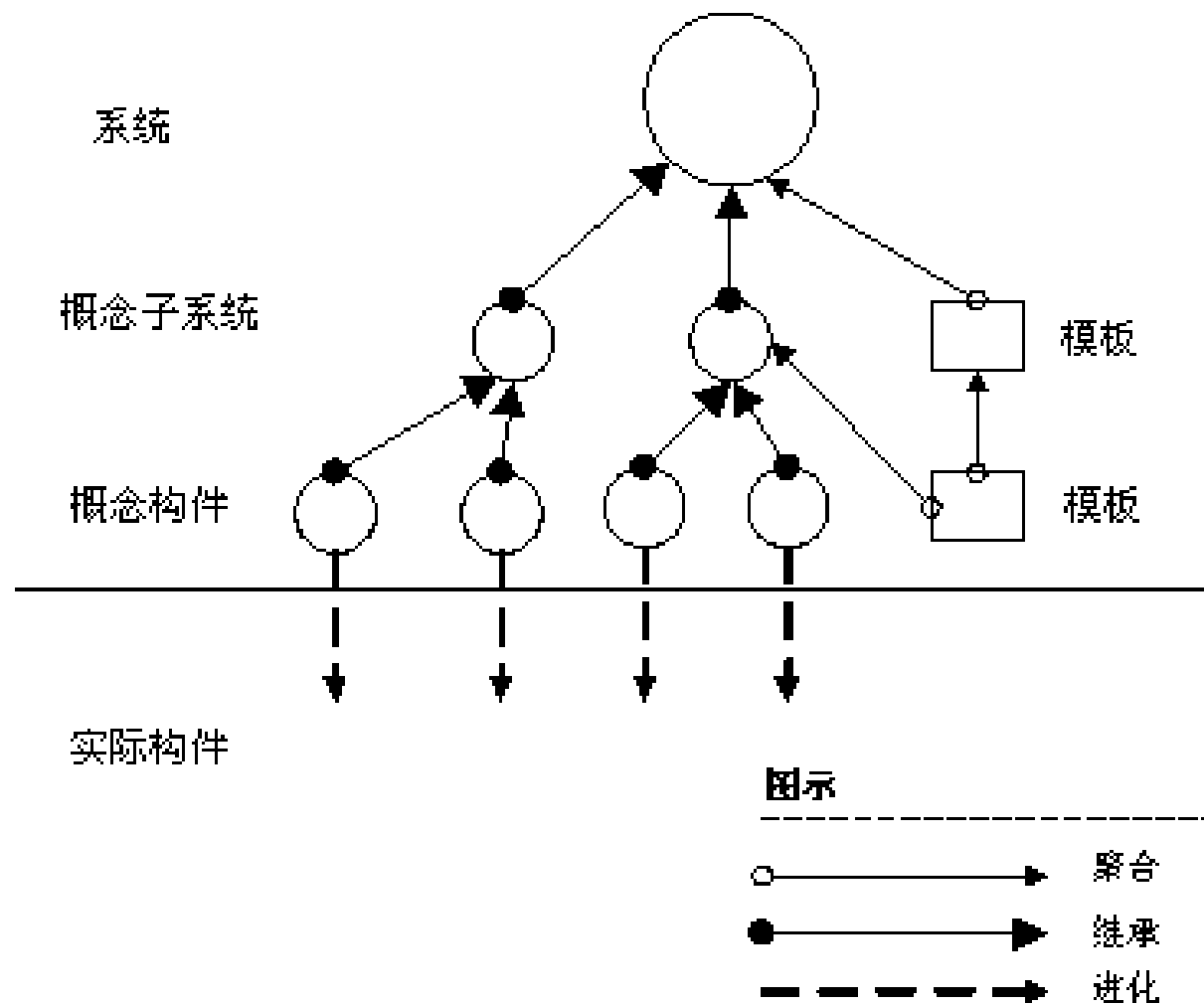
ABSD方法与生命周期

ABSD方法的输入由下列部分组成：

- **抽象功能需求**：包括变化的需求和通用的需求。
- **用例**：功能需求具体化, 在体系结构设计阶段，重要的用例才有用。
- **抽象的质量和商业需求**：质量需求尽量具体化。
- **质量因素**：实际质量和商业需求。采用质量场景可以对质量需求进行特定扩充，使质量因素具体化。
- **体系结构选项**：对于每个质量和业务需求，都要例举出能满足该需求的所有可能的体系结构。
- **约束**：前置的设计决策。例如：必须考虑某遗留系统特征。

ABSD方法的步骤

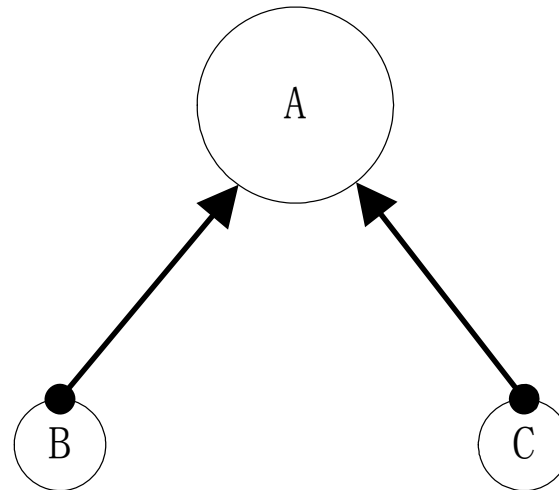
1、ABSD方法定义的设计元素



ABSD方法的步骤

2、设计元素的产生顺序

- 对ABSD方法的设计元素树可以进行广度遍历，也可以进行深度遍历。
- 下图描述了设计元素A分解为两个小的设计元素B和C。



ABSD方法的步骤

2、设计元素的产生顺序

对一个特定开发来说，决定遍历设计元素树的考虑如下：

- 领域知识
- 新技术的融合：如果新技术当作中间件或者操作系统，就必须构造原型。
- 体系结构设计团队的个人经验：有不同经验的人可以研究设计元素书的不同部分。

ABSD方法的步骤

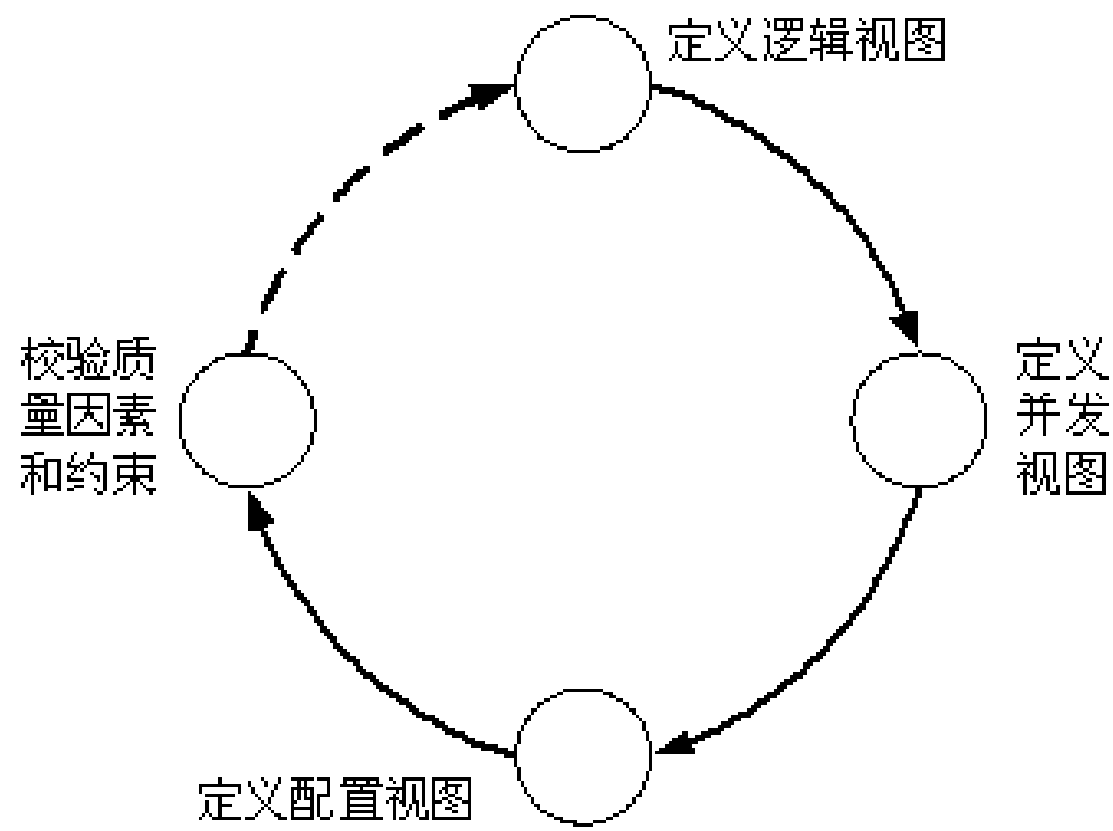
3、设计元素的活动

分解设计元素的方法：根据ABSD方法，可以利用**一组需求**（包括功能需求和性能需求）、**适合设计元素的一个模板和一组约束**来开始分解每个设计元素。

分解设计元素的输出：一个子设计元素列表（需求、自身模板和约束）。

ABSD方法的步骤

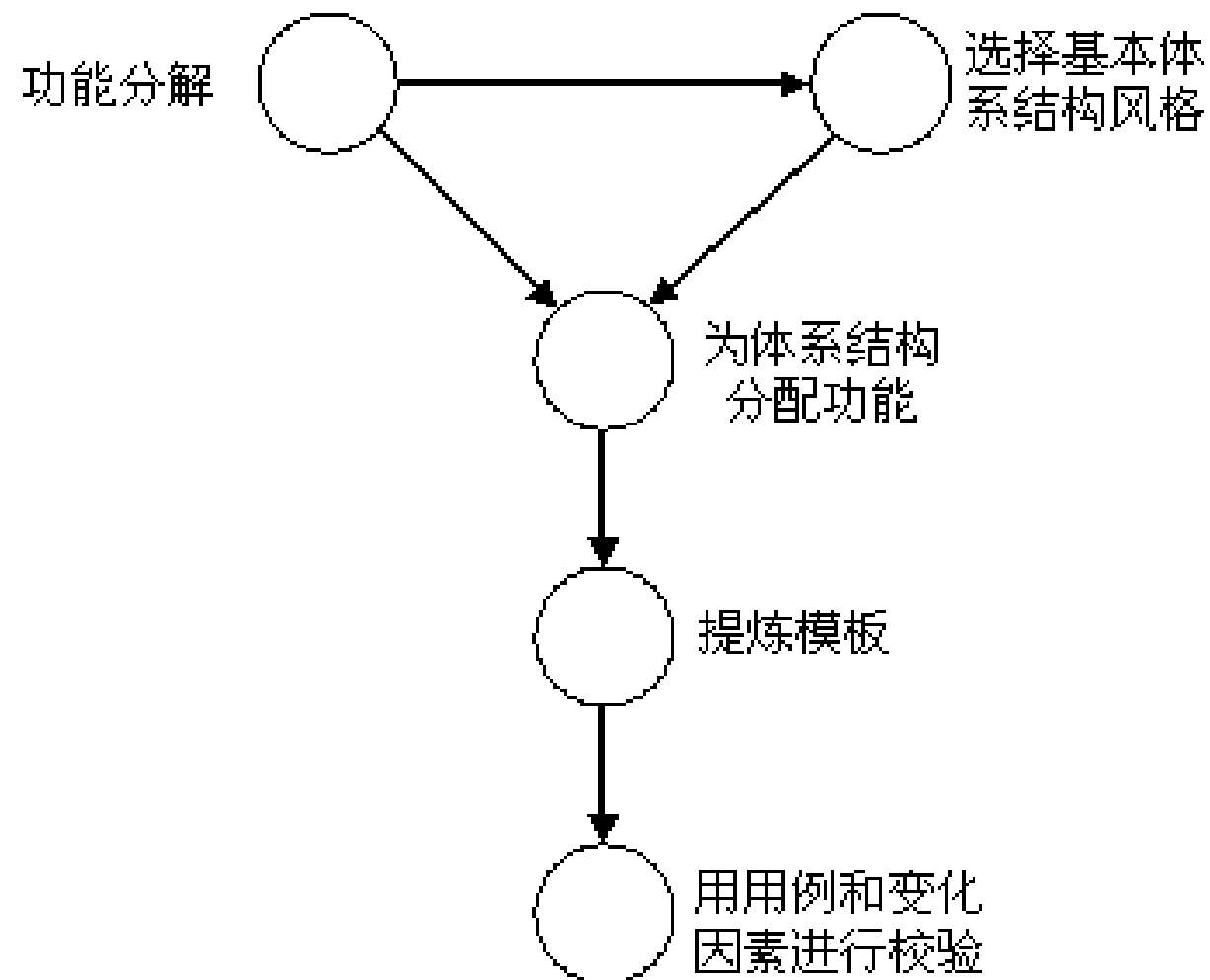
3、设计元素的活动



分解一个设计元素的步骤

ABSD方法的步骤

定义逻辑视图



ABSD方法的步骤

(1) 功能分解

一个设计元素有一组功能，这些功能必须分组。分解的目的是使每个组在体系结构内代表独立的元素。分解可以进一步细化。

功能的分组可选择几个标准：

- 1) 功能聚合。
- 2) 数据或计算行为的类似模式。
- 3) 类似的抽象级别。
- 4) 功能的局部性。公共功能区分出来。

ABSD方法的步骤

(2) 选择体系结构风格

- 每个设计元素有一个主要的体系结构风格或模式，这是设计元素如何完成它的功能的基础。主要风格并不是唯一风格，为了达到特定目的，可以进行修改。
- 体系结构风格的选择建立在设计元素的体系结构驱动基础上。
- 在软件设计过程中，并不总是有现成的体系结构风格可供选择为主要的体系结构风格。

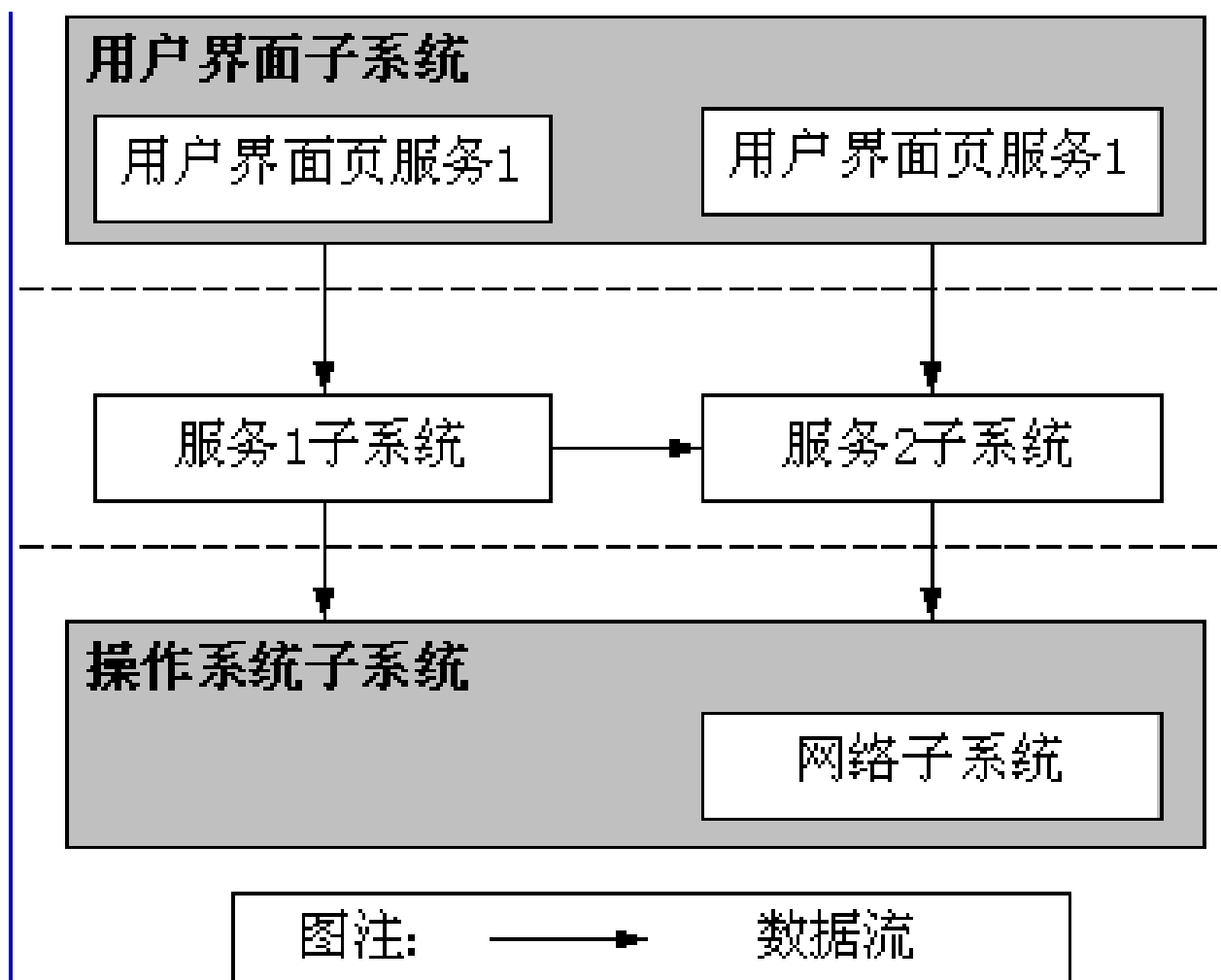
ABSD方法的步骤

(2) 选择体系结构风格

- 一旦选定了一个主要的体系结构风格，该风格必须适应这个设计元素的质量需求，体系结构选择必须满足质量需求。
- 为设计元素选择体系结构风格是一个重要的选择，这种选择在很大程度上依赖于软件设计师的个人设计经验。

ABSD方法的步骤

某系统的逻辑视图



ABSD方法的步骤

(3) 为风格分配功能

- 选择体系结构风格时产生了一组构件类型，我们必须决定这些类型的数量和每个类型的功能，这就是分配的目的。在功能分解时产生的功能组，应该分配给选择体系结构风格时产生的构件类型，这包括决定将存在多少个每个构件类型的实例，每个实例将完成什么功能。这样分配后产生的构件将作为设计元素分解的子设计元素。
- 每个设计元素的概念接口也必须得到标识，这个接口包含了设计元素所需的信息和在已经定义了的体系结构风格内的每个构件类型所需要的数据和控制流。

ABSD方法的步骤

(4) 细化模板

- 被分解的设计元素有一组属于它的模板。在ABSD方法的初期，系统没有模板。当模板细化了以后，就要把功能增加上去。这些功能必须由实际构件在设计过程中加以实现。
- 最后，需要检查模板的功能，以判断是否需要增加附加功能到系统任何地方的设计元素中。也就是说，要识别在该级别上已经存在的任何横向服务。模板包括好的设计元素和那些应该共享的功能。每种类型的功能可以根据需要附加支持功能，这种附加功能一旦得到识别，就要进行分配。

ABSD方法的步骤

(5) 功能校验

- 用例用来验证设计元素，验证设计元素是否能够通过一定的结构达到目的。子设计元素的附加功能可以通过用例的使用得到判断。
- 也可以使用变化因素，因为执行一个变化的难点取决于功能的分解。
- 从这种类型的校验出发，设计就是显示需求（通过用例）和支持修改（通过变化因素）。

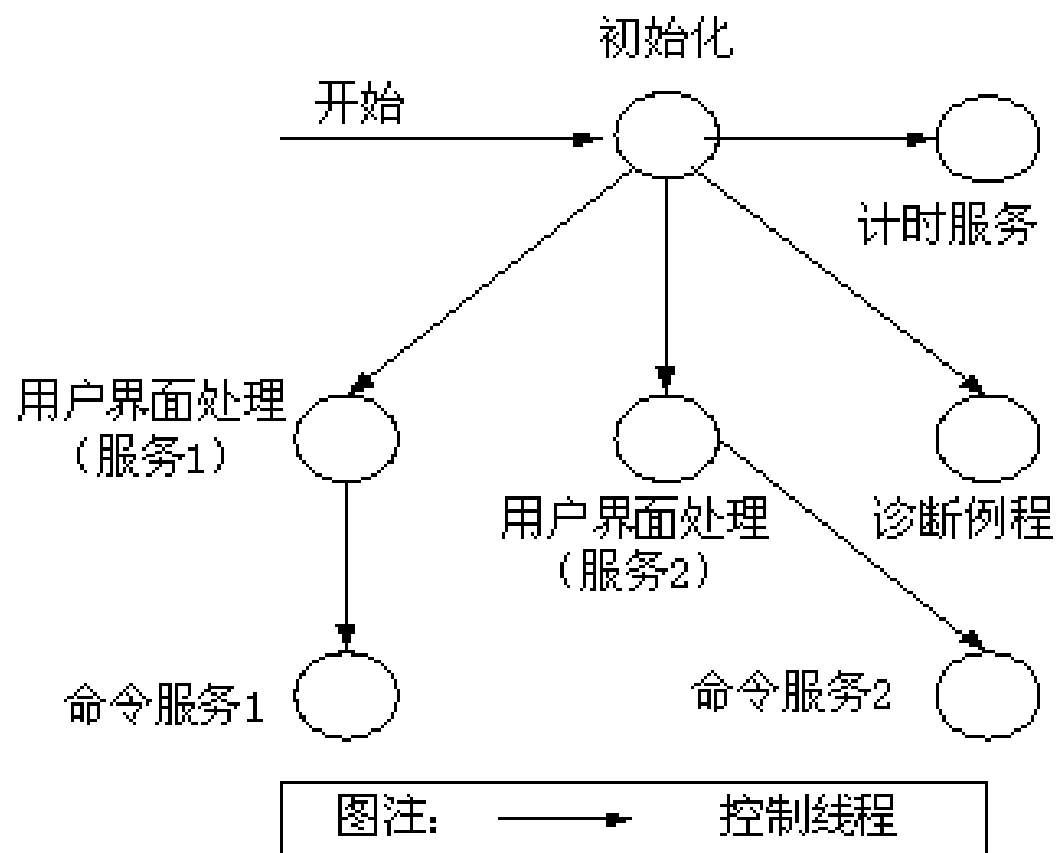
ABSD方法的步骤

(6) 创建并发视图

- 检查并发视图的目的是判断哪些活动是可以并发执行的。这些活动必须得到识别，产生进程同步和资源竞争。
- 对并发视图的检查是通过虚拟进程来实现的。虚拟进程是通过程序、动态模块或一些其他的控制流执行的一条单独路径。

ABSD方法的步骤

(6) 创建并发视图



ABSD方法的步骤

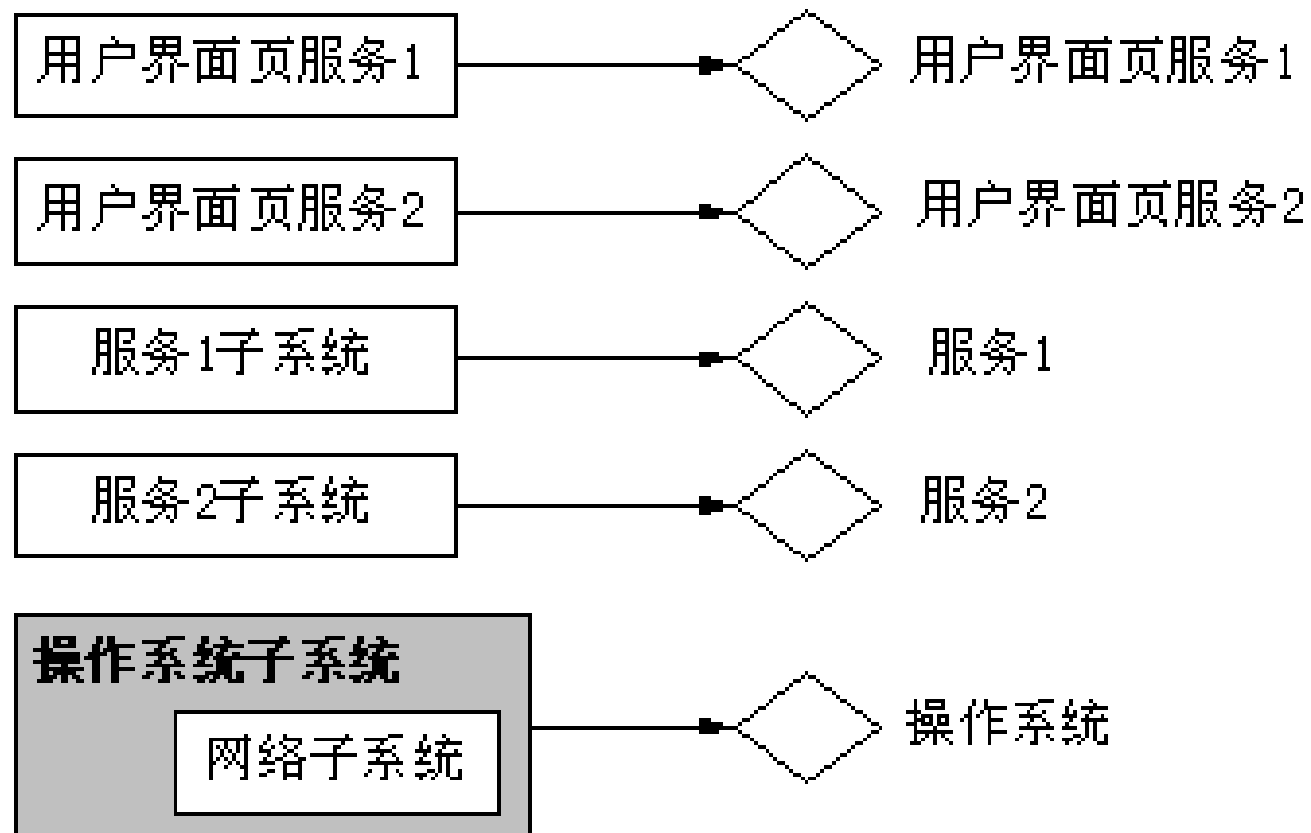
(7) 创建配置视图

➤ 如果在一个系统中，使用了多个处理器，则需要对不同的处理器配置设计元素，这种配置通过配置视图来进行检查。

➤ 例如，我们检查网络对虚拟线程的影响，一个虚拟线程可以通过网络从一个处理器传递到另一个处理器。我们使用物理线程来描述在某个特定处理器中的线程。也就是说，一个虚拟线程是由若干个物理线程串联而成的。通过这种视图，我们可以发现一个单一的处理器上的同步的物理线程和把一个虚拟线程从一个处理器传递到其他处理器上的需求。

ABSD方法的步骤

(7) 创建配置视图



图注：——➡ 控制线程 概念子系统 配置单元

ABSD方法的步骤

(8) 验证质量场景

一旦创建了三个视图，就要把质量场景应用到所创建的子设计元素上。对每个质量场景，都要考虑是否仍然满足需求，每个质量场景包括了一个质量属性刺激和所期望的响应。考虑到目前为止所作出的设计决策，看其是否能够达到质量属性的要求。

如果不能达到，则需重新考虑设计决策，或者设计师必须接受创建质量场景失败的现实。

ABSD方法的步骤

(9) 验证约束

最后一步就是要验证所有的约束没有互相矛盾的地方，对每一个约束，都需提问“该约束是否有可能实现？”。一个否定的回答就意味着对应的质量场景也不能满足。这时，需要把问题记录进文档，对导致约束的决策进行重新验证。



讨论

体系结构的设计和演化过程

- ◎ 以演化和增量方法为基础的迭代开发过程已经成为面向对象开发过程的标准。
- ◎ 在软件开发的初始阶段，选择合理的软件体系结构是非常重要的，但设计好系统的最终结构又是不可能的，也是不现实的。因为需求还在不断地发生变更。
- ◎ 随着需求的不断变更，对问题的进一步理解，以及对实现系统的技术方式的进一步理解，体系结构本身也是可以演化的。

设计和演化过程

◎ 基于体系结构的软件开发过程可以分为独立的两个阶段：

- (1) **实验原型阶段**：这一阶段的重点是获得对问题域的理解。为此，需要构建一系列原型，与实际的最终用户一起进行讨论和评审，决定是否可以实现最终系统，如果可以，则进入第二个阶段。
- (2) **演化开发阶段**：重点是放在最终产品的开发上，也就是要将重点转移到构件的精确化上。这时，原型既被当作系统的规格说明，又可当作系统的演示版本。

设计和演化过程

- 虽然实验原型的结果可以决定是否开始实现最终系统，但在实验原型阶段之后，并不是所有的功能需求都已经足够准确。
- 在每个阶段，都必须以一系列的开发周期为单位安排和组织工作。每一个开发周期都要有不同的着重点，要有分析、设计和实现的过程，这个过程取决于当前对系统的理解和前一个开发周期的结果。

实验原型阶段

◎ 第一个开发周期

➤ 第一个开发周期没有具体的、明确的目标。此时，为了提高开发效率，缩短开发周期，所有开发人员可以分成两个小组，一个小组创建图形用户界面，另一个小组创建一个问题域模型。

➤ 在第一个周期结束时，形成了两个版本，一个是图形用户界面的初始设计，另一个是问题域模型。

实验原型阶段

◎ **第二个开发周期：**任务是设计一个正交软件体系结构。此周期又可细分为以下六个小的阶段。

- (1) 标识构件
- (2) 提出软件体系结构模型
- (3) 把已标识的构件映射到软件体系结构中
- (4) 分析构件之间的相互作用
- (5) 产生软件体系结构
- (6) 软件体系结构正交化

实验原型阶段

◎ 第二个开发周期

(1) 标识构件

第一步，生成类图

第二步，对类进行分组

第三步，把类打包成构件

实验原型阶段

◎ 第二个开发周期

(2) 提出软件体系结构模型

在建立体系结构的初期，选择一个合适的体系结构风格是首要的。在这个风格基础上，开发人员通过体系结构模型，可以获得关于体系结构属性的理解。

实验原型阶段

◎ 第二个开发周期

(3) 把已标识的构件映射到软件体系结构中

把在第（1）阶段已标识的构件映射到体系结构中，将产生一个中间结构，这个中间结构只包含那些能明确适合体系结构模型的构件。

实验原型阶段

◎ 第二个开发周期

(4) 分析构件之间的相互作用

为了把所有已标识的构件集成到体系结构中，必须认真分析这些构件的相互作用和关系。我们可以使用UML的顺序图来完成这个任务。

实验原型阶段

◎ 第二个开发周期

(5) 产生软件体系结构

- 一旦决定了关键的构件之间的关系和相互作用，就可以在第（3）阶段得到的中间结构的基础上进行精化。
- 可以利用顺序图标识中间结构中的构件和剩下的构件之间的依赖关系，分析第（2）阶段模型的不一致性(例如丢失连接等)。

实验原型阶段

◎ 第二个开发周期

(6) 软件体系结构正交化

- 在（1）-（5）阶段产生的软件体系结构不一定满足正交性（例如：同一层次的构件之间可能存在相互调用）。
- 整个正交化过程以原体系结构的线索和构件为单位，自顶向下、由左到右进行。
- 通过对构件的新增、修改或删除，调整构件之间的相互作用，把那些不满足正交性的线索进行正交化。

(6) 软件体系结构正交化

正交软件体系结构由组织层和线索的组件构成。层是由一组具有相同抽象级别的组件构成。线索是子系统的特例，它是由完成不同层次功能的组件组成（通过相互调用来关联），每一条线索完成整个系统中相对独立的一部分功能。每一条线索的实现与其他线索的实现无关或关联很少，在同一层中的组件之间是不存在相互调用的。

如果线索是相互独立的，即不同线索中的组件之间没有相互调用，那么这个结构就是完全正交的。从以上定义，我们可以看出，正交软件体系结构是一种以垂直线索组件族为基础的层次化结构，其基本思想是把应用系统的结构按功能的正交相关性，垂直分割为若干个线索（子系统），线索又分为几个层次，每个线索由多个具有不同层次功能和不同抽象级别的组件构成。各线索的相同层次的组件具有相同的抽象级别。因此，我们可以归纳正交软件体系结构的主要特征如下：

实验原型阶段

◎ 第二个开发周期

(6) 软件体系结构正交化

正交软件体系结构的特征：

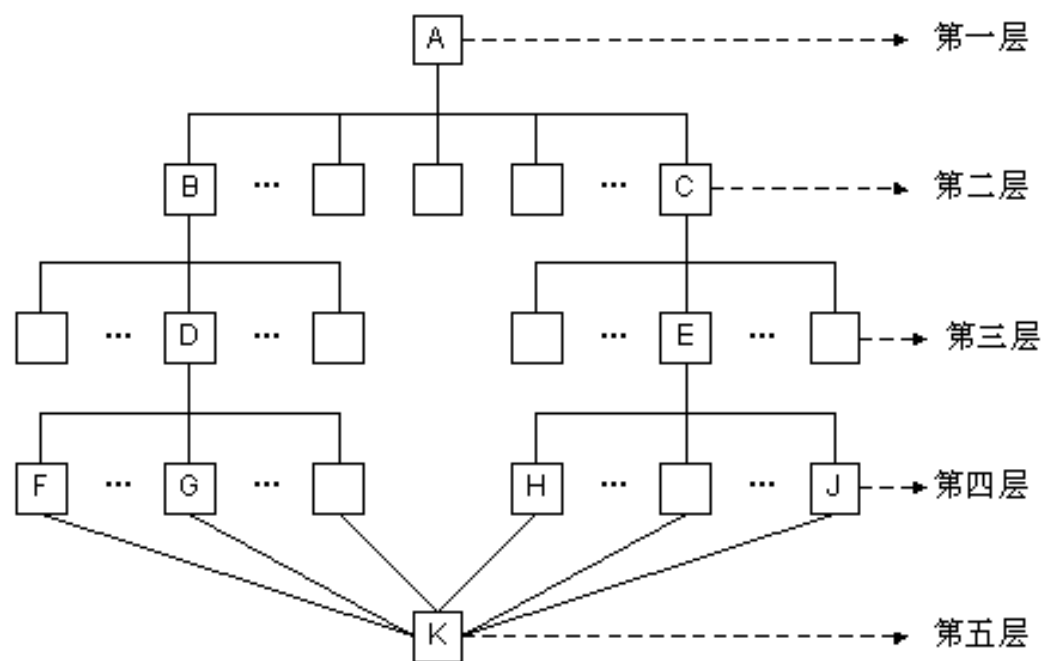
- ◎ 正交软件体系结构由完成不同功能的 n ($n > 1$) 个线索（子系统）组成；
- ◎ 系统具有 m ($m > 1$) 个不同抽象级别的层；
- ◎ 线索之间是相互独立的（正交的）；
- ◎ 系统有一个公共驱动层（一般为最高层）和公共数据结构（一般为最低层）。

实验原型阶段

◎ 第二个开发周期

(6) 软件体系结构正交化

正交软件体系结构的框架：



(6) 软件体系结构正交化

在软件进化过程中，系统需求会不断发生变化。在正交软件体系结构中，因线索的正交性，每一个需求变动仅影响某一条线索，而不会涉及到其他线索。这样，就把软件需求的变动局部化了，产生的影响也被限制在一定范围内，因此实现容易。

(6) 软件体系结构正交化

- 正交软件体系结构具有以下优点：
 - (1) **结构清晰，易于理解。**正交软件体系结构的形式有利于理解。由于线索功能相互独立，不进行互相调用，结构简单、清晰，组件在结构图中的位置已经说明它所实现的是哪一级抽象，担负的是什么功能。
 - (2) **易修改，可维护性强。**由于线索之间是相互独立的，所以对一个线索的修改不会影响到其他线索。因此，当软件需求发生变化时，可以将新需求分解为独立的子需求，然后以线索和其中的组件为主要对象分别对各个子需求进行处理，这样软件修改就很容易实现。系统功能的增加或减少，只需相应的增删线索组件族，而不影响整个正交体系结构，因此能方便地实现结构调整。
 - (3) **可移植性强，重用粒度大。**因为正交结构可以为一个领域内的所有应用程序所共享，这些软件有着相同或类似的层次和线索，可以实现体系结构级的重用。

演化开发阶段

- 一旦确定了软件的正交体系结构，就可以开始正式的构件开发工作。
- 由于体系结构的正交性，可以将开发人员分成若干小组进行并行开发。
- 在开发及运行阶段，需求都有可能发生变更。在这种情况下，就必须使用系统演化步骤去修改应用，以满足新的需求。主要包括以下八个步骤：

演化开发阶段

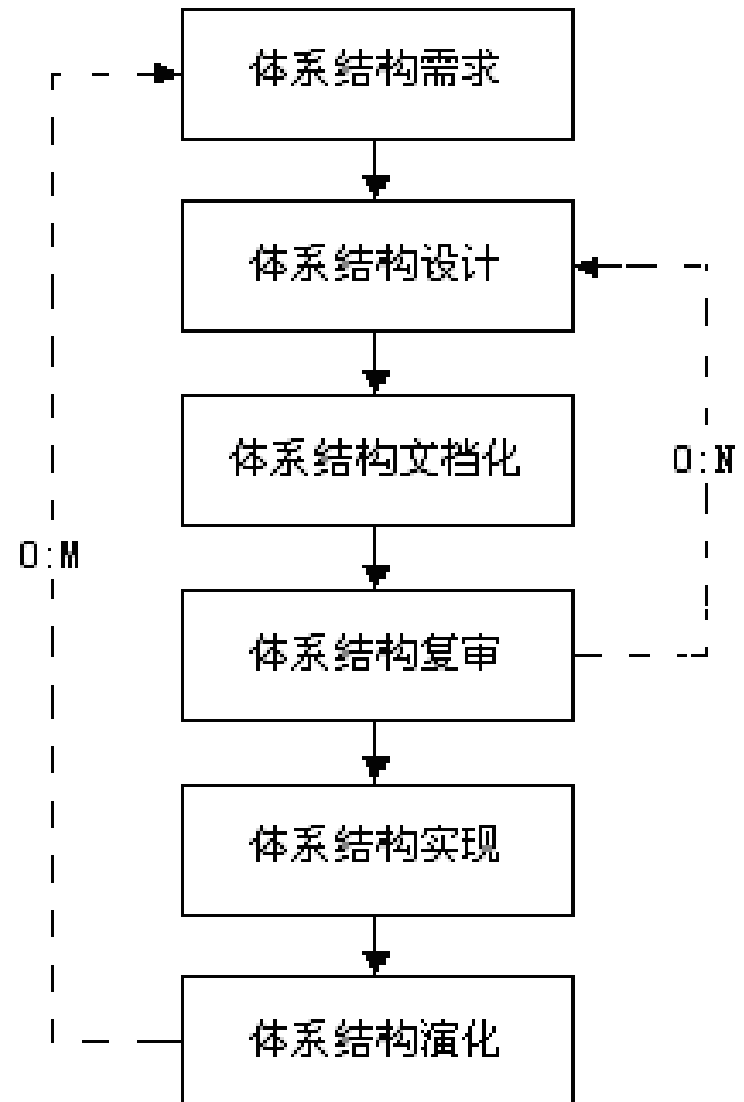
- (1) **需求变更归类**。首先必须对用户的需求变更进行归类，使变更的需求与已有构件和线索对应。对找不到对应构件和线索的变更，也要做好标记，在后续工作中，将创建新的构件或线索，以满足新的需求。
- (2) **制订体系结构演化计划**：在改变原有结构之前，开发组织必须制订一个周密的体系结构演化计划，作为后续演化开发工作的指南。
- (3) **修改、增加或删除构件**。
- (4) **更新构件的相互作用**。

演化开发阶段

- (5) 产生演化后的体系结构。在原来系统上所做的所有修改必须集成到原来的体系结构中。
- (6) 迭代：如果在第(5)步得到的体系结构还不够详细，不能实现变更的需求，可以将(3)–(5)步再迭代一次。
- (7) 对以上步骤进行确认，进行阶段性技术评审。
- (8) 对所做的标记进行处理。重新开发新线索中的所有构件，对已有构件按照标记的要求进行修改、删除或更换。

ABSDM

➤基于体系结构的软件开发模型(ABSDM)将软件开发过程划分为体系结构需求、设计、文档化、复审、实现、演化等六个子过程。



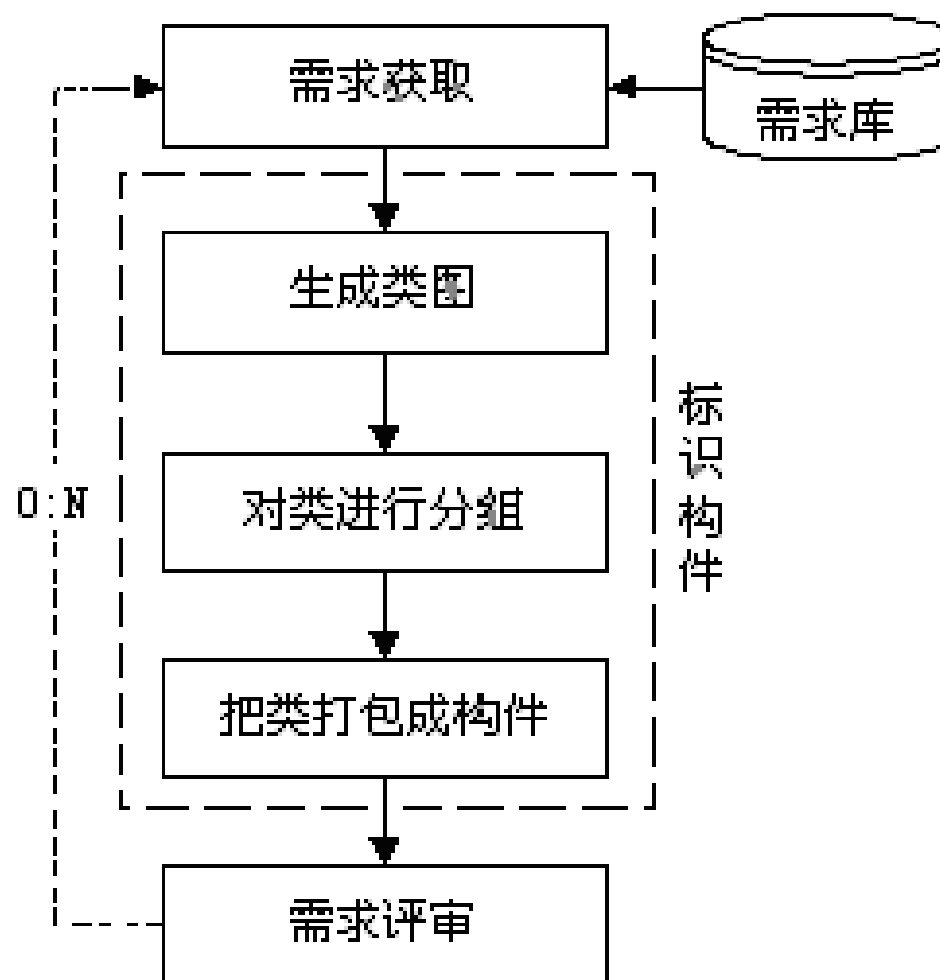
体系结构需求

➤ 体系结构的需求过程：

(1) 需求获取；

(2) 标识构件；

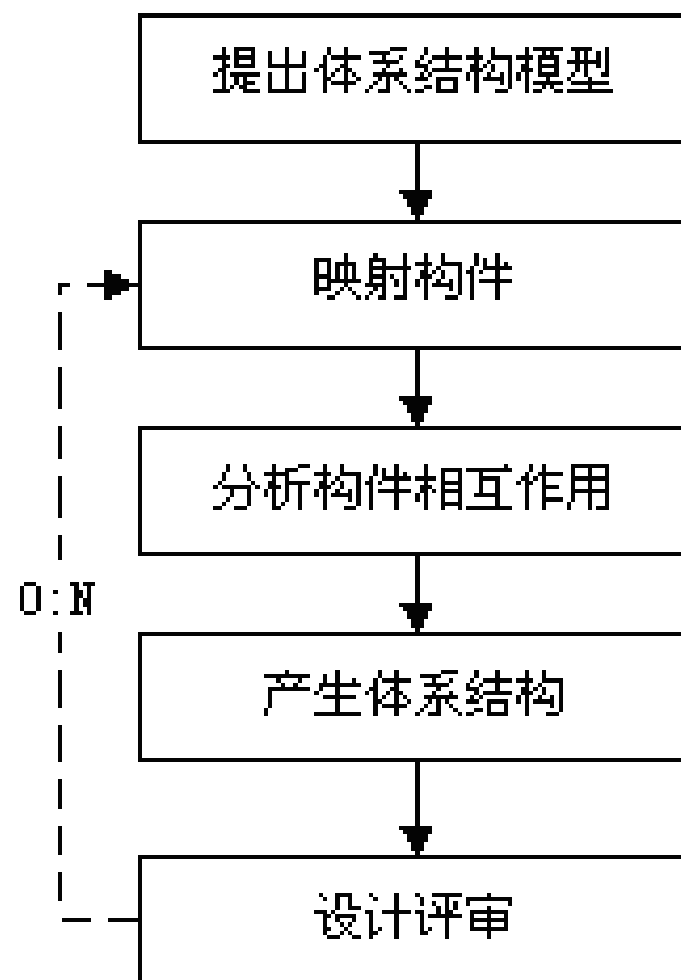
(3) 需求评审。



体系结构设计

➤ 体系结构的设计过程:

- (1) 提出软件体系结构模型;
- (2) 将已标识的构件映射到软件体系结构中;
- (3) 分析构件之间的相互作用;
- (4) 产生软件体系结构;
- (5) 设计评审。



体系结构文档化

- 在系统演化的每一个阶段，文档是系统设计与开发人员的通讯媒介，是为验证体系结构设计和提炼或修改这些设计（必要时）所执行预先分析的基础。
- 体系结构文档化过程的主要输出结果是体系结构需求规格说明和测试体系结构需求的质量设计说明书这两个文档。生成需求模型构件的精确的形式化的描述，作为用户和开发者之间的一个协约。

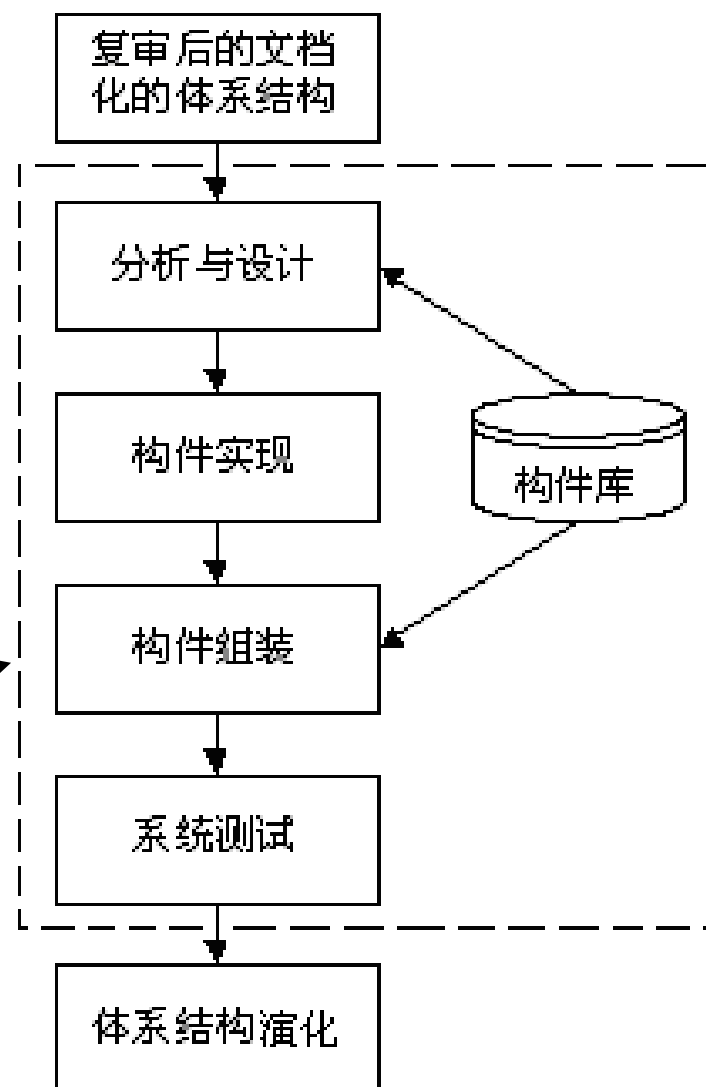
体系结构复审

- 体系结构设计、文档化和复审是一个迭代过程。
- 复审的目的是标识潜在的风险，及早发现体系结构设计中的缺陷和错误，包括体系结构能否满足需求、质量需求是否在设计中得到体现、层次是否清晰、构件的划分是否合理、文档表达是否明确、构件的设计是否满足功能与性能的要求等等。

体系结构实现

➤即按照体系结构所描述的结构化设计策略，分割成规定的构件，对构件进行分析设计、实现、组装（按规定方式互相交互），并进行测试。

体系结构实现过程



体系结构演化

➤ 体系结构演化过程

