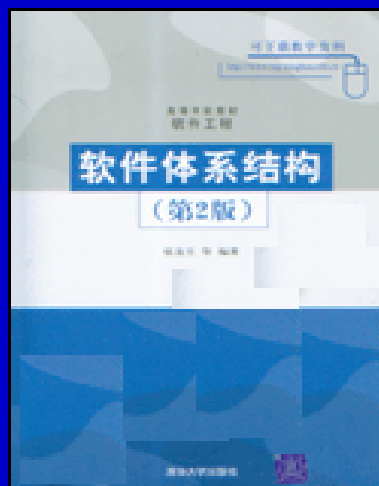


# 软件体系结构

张友生  
希赛IT教育研发中心

# 关于教材



- ◇ 出版社：清华大学出版社
- ◇ 作者：张友生

# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

### ◇ 软件危机的表现

- ◎ 软件成本日益增长
- ◎ 开发进度难以控制
- ◎ 软件质量差
- ◎ 软件维护困难

### ◇ 软件危机的表现

#### ◎ 软件成本日益增长

20世纪50年代，软件成本在整个计算机系统成本中所占的比例为10%-20%。到20世纪60年代中期，软件成本在计算机系统所占的比例已经增长到50%左右。

而且，该数字还在不断地递增，下面是一组来自美国空军计算机系统的数据：1955年，软件费用约占总费用的18%，1970年达到60%，1975年达到72%，1980年达到80%，1985年达到85%左右。

### ◇ 软件危机的表现

#### ◎ 开发进度难以控制

由于软件是逻辑、智力产品，软件的开发需建立庞大的逻辑体系，这是与其他产品的生产不一样的。

在软件开发过程中，用户需求变化等各种意想不到的情况层出不穷，令软件开发过程很难保证按预定的计划实现，给项目计划和论证工作带来了很大的困难。

盲目增加软件开发人员并不能成比例地提高软件开发能力。相反，随着人员数量的增加，人员的组织、协调、通信、培训和管理等方面的问题将更为严重。

### ◇ 软件危机的表现

#### ◎ 软件质量差

软件项目即使能按预定日期完成，结果却不尽人意。1965年至1970年，美国范登堡基地发射火箭多次失败，绝大部分故障是由应用程序错误造成的。

在“软件作坊”里，由于缺乏工程化思想的指导，程序员几乎总是习惯性地以自己的想法去代替用户对软件的需求，软件设计带有随意性，很多功能只是程序员的“一厢情愿”而已，这是造成软件不能令人满意的重要因素。

### ◇ 软件危机的表现

#### ◎ 软件维护困难

由于在软件设计和开发过程中，没有严格遵循软件开发标准，各种随意性很大，没有完整的真实反映系统状况的记录文档，给软件维护造成了巨大的困难。

特别是在软件使用过程中，原来的开发人员可能因各种原因已经离开原来的开发组织，使得软件几乎不可维护。

有资料表明，工业界为维护软件支付的费用占全部硬件和软件费用的40%-75%。



### ◇ 软件危机的原因

- ◎ 用户需求不明确
- ◎ 缺乏正确的理论指导
- ◎ 软件规模越来越大
- ◎ 软件复杂度越来越高

### ◇ 软件危机的原因

#### ◎ 用户需求不明确

在软件开发完成之前，用户不清楚软件的具体需求；

用户对软件需求的描述不精确，可能有遗漏、有二义性、甚至有错误；

在软件开发过程中，用户还提出修改软件功能、界面、支撑环境等方面的要求；

开发人员对用户需求的理解与用户本来愿望有差异。

### ◇ 软件危机的原因

#### ◎ 缺乏正确的理论指导

缺乏有力的方法学和工具方面的支持。由于软件不同于大多数其他工业产品，其开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员高度的智力投入。由于过分地依靠程序设计人员在软件开发过程中的技巧和创造性，加剧软件产品的个性化，也是发生软件危机的一个重要原因。

### ◇ 软件危机的原因

#### ◎ 软件规模越来越大

随着软件应用范围的增广，软件规模愈来愈大。大型软件项目需要组织一定的人力共同完成，而多数管理人员缺乏开发大型软件系统的经验，而多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确、有时还会产生误解。

软件项目开发人员不能有效地、独立自主地处理大型软件的全部关系和各个分支，因此容易产生疏漏和错误。

### ◇ 软件危机的原因

#### ◎ 软件复杂度越来越高

软件不仅仅是在规模上快速地发展扩大，而且其复杂性也急剧地增加。软件产品的特殊性和人类智力的局限性，导致人们无力处理“复杂问题”。

所谓“复杂问题”的概念是相对的，一旦人们采用先进的组织形式、开发方法和工具提高了软件开发效率和能力，新的、更大的、更复杂的问题又摆在人们的面前。

### ◇ 如何克服软件危机

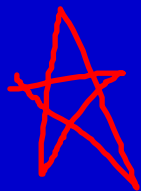
人们面临的不光是技术问题，更重要的是管理问题。管理不善必然导致失败。

要提高软件开发效率，提高软件产品质量，必须采用工程化的开发方法与工业化的生产技术。

在技术上，应该采用基于重用的软件生产技术；  
在管理上，应该采用多维的工程管理模式。

### ◇ 构件模型及实现

#### ◎ 构件的定义



构件是指语义完整、语法正确和有可重用价值的单位软件，是软件重用过程中可以明确辨识的系统；结构上，它是**语义描述**、**通讯接口**和**实现代码**的复合体。

### ◇ 构件模型及实现

#### ◎ 构件模型的三个主要流派

OMG (Object Management Group, 对象管理集团) 的 CORBA (Common Object Request Broker Architecture, 通用对象请求代理结构)

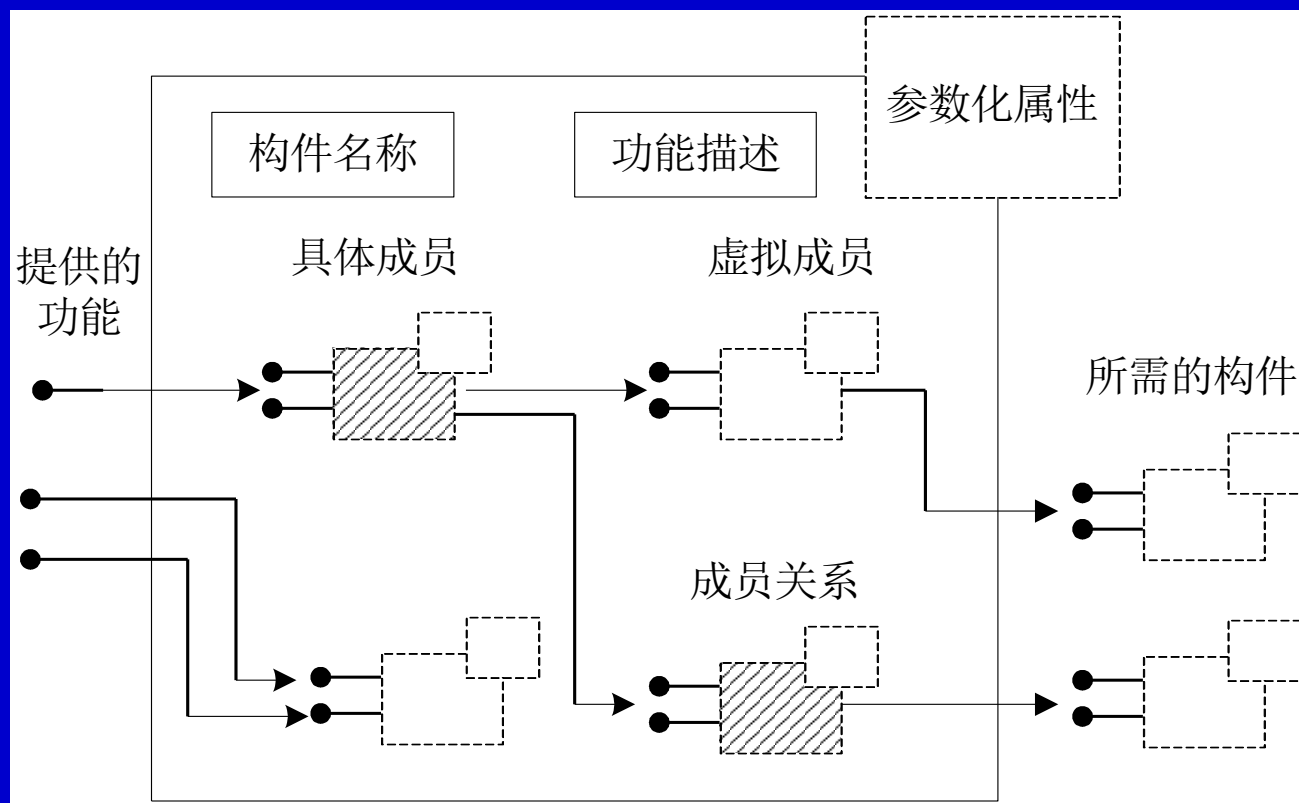
Sun的EJB (Enterprise Java Bean)

Microsoft的DCOM (Distributed Component Object Model, 分布式构件对象模型)。



### ◇ 构件模型及实现

#### ◎ 青鸟构件模型



### ◇ 构件获取

从现有构件中获得符合要求的构件，直接使用或作适应性修改，得到可重用的构件；

通过遗留工程，将具有潜在重用价值的构件提取出来，得到可重用的构件；

从市场上购买现成的商业构件，即COTS  
(Commercial Off-The-Shelf) 构件；

开发新的符合要求的构件。

### ◇ 构件管理

- ◎ 构件描述
- ◎ 构件分类与组织
- ◎ 人员及权限管理

### ◇ 构件管理

#### ◎ 构件描述

构件模型是对构件本质的抽象描述，主要是为构件的制作与构件的重用提供依据；

从管理角度出发，也需要对构件进行描述，例如：实现方式、实现体、注释、生产者、生产日期、大小、价格、版本和关联构件等信息，它们与构件模型共同组成了对构件的完整描述。

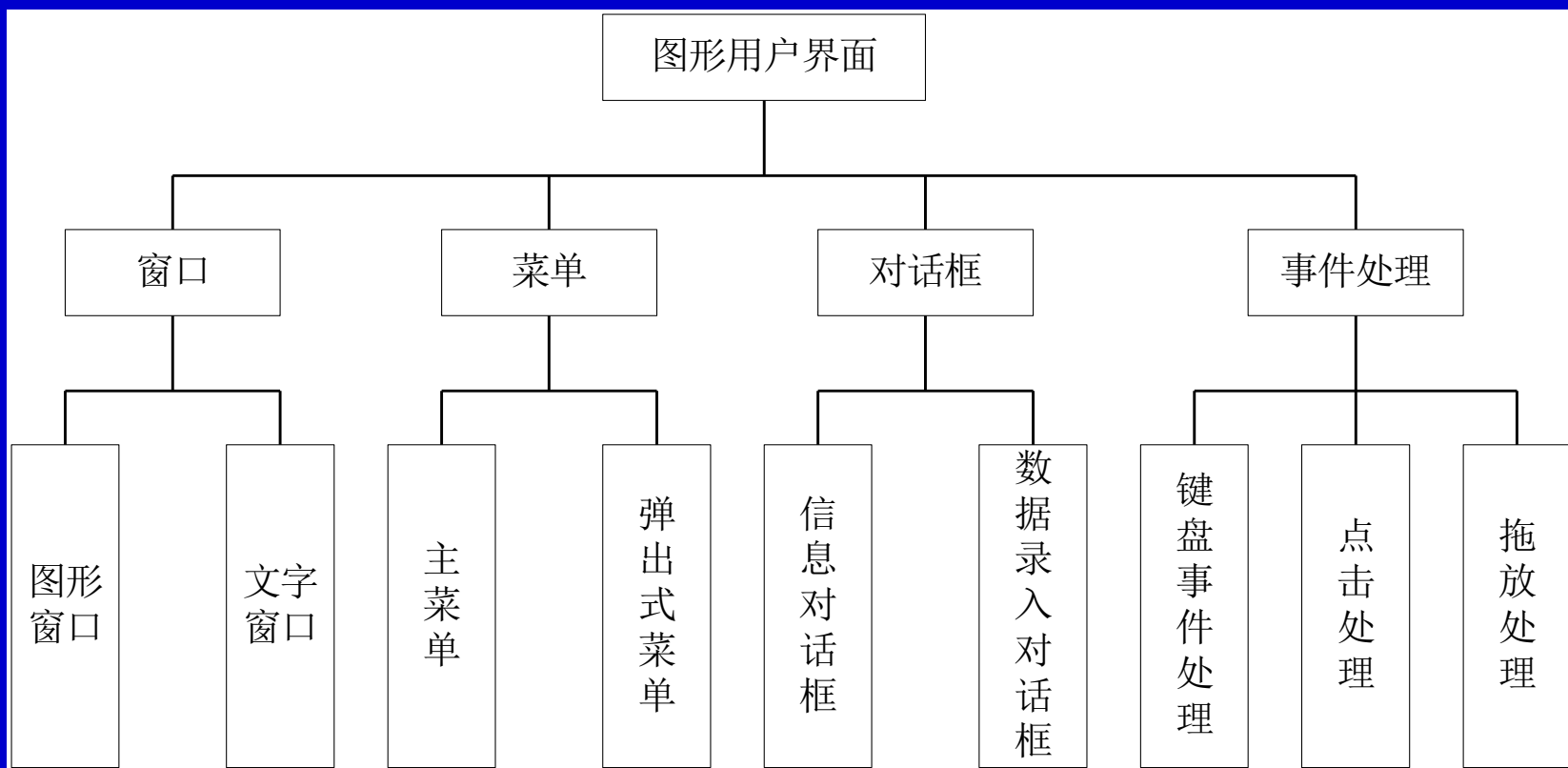
### ◇ 构件管理

#### ◎ 构件分类与组织

- ◇ 关键字分类法
- ◇ 刻面分类法
- ◇ 超文本组织方法

### ◇ 构件管理

#### ◎ 关键字分类法



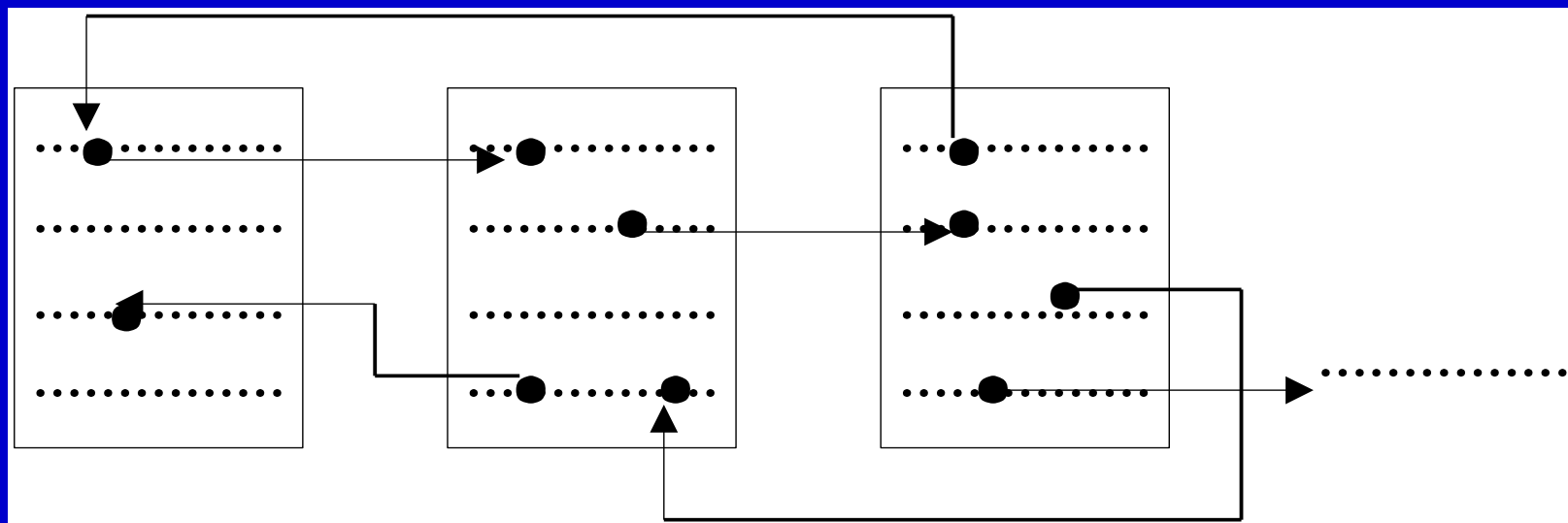
### ◇ 构件管理

#### ◎ 刻面分类法

- ◇ 使用环境
- ◇ 应用领域
- ◇ 功能
- ◇ 层次
- ◇ 表示方法

### ◇ 构件管理

#### ◎ 超文本组织法





### ◇ 构件管理

#### ◎ 人员及权限管理

一般来讲，构件库系统可包括五类用户，即注册用户、公共用户、构件提交者、一般系统管理员和超级系统管理员。

### ◇ 构件重用

- ◎ 检索与提取构件
- ◎ 理解与评价构件
- ◎ 修改构件
- ◎ 构件组装

### ◇ 构件重用

#### ◎ 检索与提取构件

- ◇ 基于关键字的检索
- ◇ 刻面检索法
- ◇ 超文本检索法
- ◇ 其他检索方法

### ◇ 构件重用

#### ◎ 理解与评价构件

- ◇ 构件的功能与行为
- ◇ 相关的领域知识
- ◇ 可适应性约束条件与例外情形
- ◇ 可以预见的修改部分及修改方法

### ◇ 构件重用

#### ◎ 修改构件

理想的情形是对库中的构件不作修改而直接用于新的软件项目。

但是，在大多数情况下，必须对构件进行或多或少的修改，以适应新的需求。

为了减少构件修改的工作量，要求开发人员尽量使构件的功能、行为和接口设计更为抽象化、通用化和参数化。

### ◇ 构件重用

#### ◎ 构件组装

- ◇ 基于功能的组装技术
- ◇ 基于数据的组装技术
- ◇ 面向对象的组装技术

### ◇ 构件重用

#### ◎ 构件组装

##### ◇ 基于功能的组装技术

基于功能的组装技术采用子程序调用和参数传递的方式将构件组装起来。它要求库中的构件以子程序/过程/函数的形式出现，并且接口说明必须清晰。当使用这种组装技术进行软件开发时，开发人员首先应对目标软件系统进行功能分解，将系统分解为强内聚、松耦合的功能模块。然后根据各模块的功能需求提取构件，对它进行适应性修改后再挂接在上述功能分解框架中。

### ◇ 构件重用

#### ◎ 构件组装

##### ◇ 基于数据的组装技术

首先根据当前软件问题的核心数据结构设计出一个**框架**，然后根据框架中各结点的需求**提取构件**并进行适应性修改，再将构件逐个**分配**至框架中的适当位置。此后，构件的组装方式仍然是传统的**子程序调用与参数传递**。这种组装技术也要求库中构件以子程序形式出现，但它所依赖的软件设计方法不再是功能分解，而是**面向数据**的设计方法，例如Jackson系统开发方法。



### ◇ 构件重用

#### ◎ 构件组装

##### ◇ 面向对象的组装技术

###### · 构造法

在子类中引进基类的对象作为子类的成员变量，然后在子类中通过成员变量重用基类的属性和方法。

###### · 子类法

将新子类直接说明为库中基类的子类，通过继承和修改基类的属性与行为完成新子类的定义。

### ◇ 软件重用实例

自学

### ◇ 背景资料

- ◎ 随着软件系统规模越来越大、越来越复杂，整个系统的结构和规格说明显得越来越重要。
- ◎ 对于大规模的复杂软件系统来说，对总体的系统结构设计和规格说明比起对计算的算法和数据结构的选择已经变得明显重要得多。
- ◎ 对软件体系结构的系统、深入的研究将会成为提高软件生产率和解决软件维护问题的新的最有希望的途径。

### ◇ 背景资料

◎ 事实上，软件总是有体系结构的，不存在没有体系结构的软件。

◎ 软件体系结构虽脱胎于软件工程，但其形成同时借鉴了计算机体系结构和网络体系结构中很多宝贵的思想和方法，最近几年软件体系结构研究已完全独立于软件工程的研究，成为计算机科学的一个最新的研究方向和独立学科分支。

### ◇ 软件体系结构的定义

◎ Dewayne Perry和Alexander Wolf

软件体系结构是具有一定形式的**结构化元素**，即构件的集合，包括处理构件、数据构件和连接构件。

处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。

这一定义注重区分处理构件、数据构件和连接构件，这一方法在其他的定义和方法中基本上得到保持。

### ◇ 软件体系结构的定义

◎ Mary Shaw和David Garlan

软件体系结构是软件设计过程中的一个层次，这一层次超越计算过程中的算法设计和数据结构设计。

体系结构问题包括总体组织和全局控制、通讯协议、同步、数据存取，给设计元素分配特定功能，设计元素的组织，规模和性能，在各设计方案间进行选择等。

软件体系结构处理算法与数据结构之上关于整体系统结构设计和描述方面的一些问题，如全局组织和全局控制结构、关于通讯、同步与数据存取的协议，设计构件功能定义，物理分布与合成，设计方案的选择、评估与实现等。

### ◇ 软件体系结构的定义

#### ◎ Kruchten

软件体系结构有四个角度，它们从不同方面对系统进行描述：**概念**角度描述系统的主要构件及它们之间的关系；**模块**角度包含功能分解与层次结构；**运行**角度描述了一个系统的动态结构；**代码**角度描述了各种代码和库函数在开发环境中的组织。

### ◇ 软件体系结构的定义

#### ◎ Hayes Roth

软件体系结构是一个抽象的**系统规范**，主要包括用其行为来描述的功能构件和构件之间的相互连接、接口和关系。



### ◇ 软件体系结构的定义

◎ David Garlan 和 Dewne Perry

软件体系结构是一个程序 / 系统各构件的结构、它们之间的相互关系以及进行设计的原则和随时间演化的指导方针。

### ◇ 软件体系结构的定义

#### ◎ Barry Boehm

软件体系结构包括一个软件和系统构件，互联及约束的集合；一个**系统需求**说明的集合；一个基本原理用以说明这一构件，互联和约束能够满足系统需求。

### ◇ 软件体系结构的定义

#### ◎ Bass, Clements 和 Kazman

软件体系结构包括一个或一组软件构件、软件构件的**外部的可见特性**及其相互关系。其中，“软件外部的可见特性”是指软件构件提供的服务、性能、特性、错误处理、共享资源使用等。

### ◇ 软件体系结构的定义

#### ◎ 我们的定义

软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。软件体系结构不仅指定了系统的**组织结构**和**拓扑结构**，并且显示了系统需求和构成系统的元素之间的对应关系，提供了一些设计决策的基本原理。

### ◇ 软件体系结构的意义

#### ◎ 体系结构是风险承担者进行交流的手段

软件体系结构代表了系统的公共的高层次的抽象。这样，系统的大部分有关人员（即使不是全部）能把它作为建立一个互相理解的基础，形成统一认识，互相交流。

体系结构提供了一种共同语言来表达各种关注和协商，进而对大型复杂系统能进行理智的管理。这对项目最终的质量和使用有极大的影响。

### ◇ 软件体系结构的意义

#### ◎ 体系结构是早期设计决策的体现

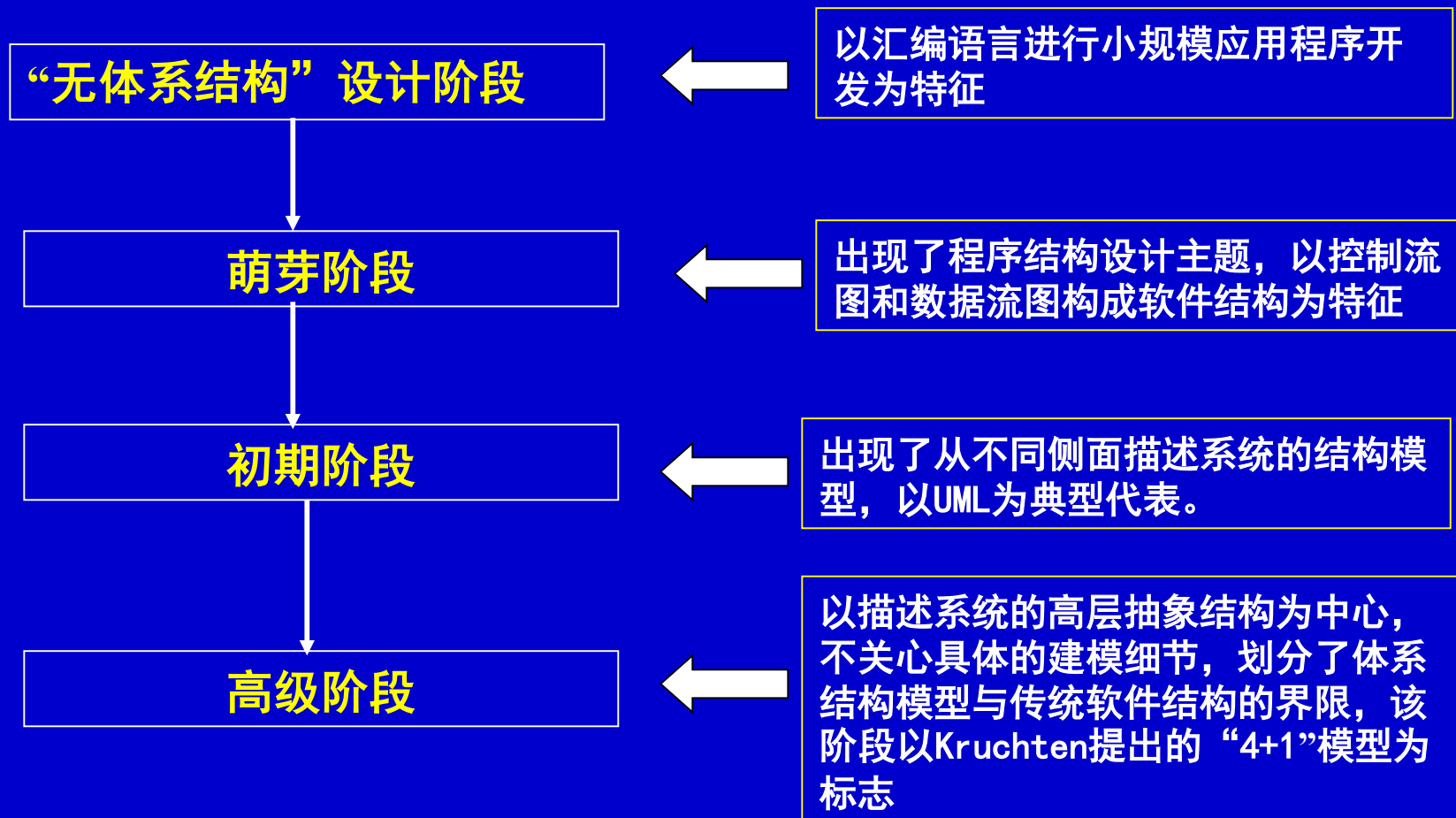
- (1) 软件体系结构明确了对系统实现的约束条件
- (2) 软件体系结构决定了开发和维护组织的组织结构
- (3) 软件体系结构制约着系统的质量属性
- (4) 通过研究软件体系结构可能预测软件的质量
- (5) 软件体系结构使推理和控制更改更简单
- (6) 软件体系结构有助于循序渐进的原型设计
- (7) 软件体系结构可以作为培训的基础

### ◇ 软件体系结构的意义

#### ◎ 软件体系结构是可传递和可重用的模型

软件体系结构级的重用意味着体系结构的决策能在具有相似需求的多个系统中发生影响，这比代码级的重用要有更大的好处。

### ◇ 软件体系结构的发展史





### ◇ 软件体系结构的发展史

Perry和Wolf认为

未来的年代是研究软件体系结构的时代

### ◇ 软件体系结构的应用现状

- ◎ 软件体系结构描述语言
- ◎ 体系结构描述构造与表示
- ◎ 体系结构分析、设计与验证
- ◎ 体系结构发现、演化与重用
- ◎ 基于体系结构的软件开发方法
- ◎ 特定领域的体系结构框架
- ◎ 软件体系结构支持工具
- ◎ 软件产品线体系结构
- ◎ 建立评价软件体系结构的方法

### ◇ 软件体系结构的应用现状

#### ◎ 软件体系结构描述语言

ADL提供了具体的语法与刻画体系结构的概念框架。ADL使得系统开发者能够很好地描述他们设计的体系结构，以便与他人交流，能够用提供的工具对许多实例进行分析。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构描述构造与表示（1）

按照一定的描述方法，用体系结构描述语言对体系结构进行说明的结果则称为体系结构的表示，而将描述体系结构的过程称为体系结构构造

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构描述构造与表示（2）

（1）Kruchten提出的“4+1”模型。

（2）Booch从UML的角度给出了一种由设计视图、过程视图、实现视图和部署视图，再加上一个用例视图构成的体系结构描述模型。

（3）IEEE于1995年成立了体系结构工作组，起草了体系结构描述框架标准IEEE P1471。

（4）Rational从资产重用的角度提出了体系结构描述的规格说明框架。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构分析、设计与验证（1）

体系结构分析的内容可分为**结构分析**、**功能分析**和**非功能分析**。

非功能分析：定量分析方法、推断分析方法。

Kazman等人提出了一种非功能分析的体系结构分析方法SAAM，并运用场景技术，提出了基于场景的体系结构分析方法，而Barbacci等人提出了多质量属性情况下的体系结构质量模型、分析与权衡方法ATAM。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构分析、设计与验证（2）

生成一个满足软件需求的体系结构的过程即为体系结构设计。体系结构设计过程的本质在于：将系统分解成相应的组成成分（如构件、连接件），并将这些成分重新组装成一个系统。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构分析、设计与验证（3）

体系结构设计有两大类方法：过程驱动方法和问题列表驱动方法。

基于过程驱动的体系结构设计方法适用范围广，易于裁减，具备动态特点，通用性与实践性强。

问题列表驱动法的基本思想是**枚举**设计空间，并考虑设计维的相关性，以此来选择体系结构的风格。该方法适用于特定领域，是静态的，并可以实现量化体系结构设计空间。



### ◇ 软件体系结构的应用现状

#### ◎ 体系结构分析、设计与验证（4）

体系结构设计研究的重点内容之一就是体系结构风格或模式，体系结构模式在本质上反映了一些特定的元素、按照特定的方式组成一个特定的结构，该结构应有利于上下文环境下的特定问题的解决。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构分析、设计与验证（5）

体系结构模式分为两个大类：固定术语和参考模型。

已知的固定术语类的体系结构模型包括管道过滤器、客户/服务器、面向对象、黑板、分层、对等模式、状态转换、一些派生的固定术语类的体系结构模式，包括Gen Voca，C2和REST等；而参考模型则相对较多，常常与特定领域相关。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构分析、设计与验证（6）

体系结构测试着重于仿真系统模型，解决体系结构层的主要问题。由于测试的抽象层次不同，体系结构测试策略可以分为单元/子系统/集成/验收测试等阶段的测试策略。

在体系结构集成测试阶段，Debra等人提出了一组针对体系结构的测试覆盖标准，Paola Inveradi提出了一种基于CHAM的体系结构语义验证技术。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构发现、演化与重用（1）

体系结构发现解决如何从已经存在的系统中提取软件的体系结构，属于逆向工程范畴。

Waters等人提出了一种迭代式体系结构发现过程，即由不同的人员对系统进行描述，然后对这些描述进行分类并融合，发现并解除冲突，将体系结构新属性加入到已有的体系结构模型中，并重复该过程直至体系结构描述充分。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构发现、演化与重用（2）

由于系统需求、技术、环境、分布等因素的变化而最终导致软件体系结构的变动，称之为软件体系结构演化。

软件系统在运行时刻的体系结构变化称为体系结构的动态性，而将体系结构的静态修改称为体系结构扩展。体系结构扩展与体系结构动态性都是体系结构适应性和演化性的研究范畴。

### ◇ 软件体系结构的应用现状

#### ◎ 体系结构发现、演化与重用（3）

体系结构重用属于设计重用，比代码重用更抽象。由于软件体系结构是系统的高层抽象，反映了系统的主要组成元素及其交互关系，因而较算法更稳定，更适合于重用。

体系结构模式就是体系结构重用研究的一个成果，而体系结构参考模型则是特定域软件体系结构的重用的成熟的象征。

### ◇ 软件体系结构的应用现状

#### ◎ 基于体系结构的软件开发方法（1）

在引入了体系结构的软件开发之后，应用系统的构造过程变为“问题定义—>软件需求—>软件体系结构—>软件设计—>软件实现”，可以认为软件体系结构架起了软件需求与软件设计之间的一座桥梁。

### ◇ 软件体系结构的应用现状

#### ◎ 基于体系结构的软件开发方法（2）

软件开发模型是跨越整个软件生存周期的系统开发、运行、维护所实施的全部工作和任务的结构框架，给出了软件开发活动各阶段之间的关系。

目前，常见的软件开发模型大致可分为三种类型：

- （1）以软件需求完全确定为前提的瀑布模型。
- （2）在软件开发初始阶段只能提供基本需求时采用的渐进式开发模型，如螺旋模型等。
- （3）以形式化开发方法为基础的变换模型。



### ◇ 软件体系结构的应用现状

#### ◎ 基于体系结构的软件开发方法（3）

所有开发方法都是要解决需求与实现之间的差距。但是，这三种类型的软件开发模型都存在这样或那样的缺陷，不能很好地支持基于软件体系结构的开发过程。

在基于构件和基于体系结构的软件开发逐渐成为主流情况下，已经出现了**基于构件的软件工程**。

但是，对体系结构的描述、表示、设计和分析以及验证等内容的研究还相对不足，随着需求复杂化及其演化，切实可行的体系结构设计规则与方法将更为重要。

### ◇ 软件体系结构的应用现状

#### ◎ 特定领域的体系结构框架

特定领域的体系结构是将体系结构理论应用到具体领域的过程，常见的DSSA有：CASE体系结构、CAD软件的参考模型、信息系统的参考体系结构、网络体系结构DSSA、机场信息系统的体系结构和信息处理DSSA等。国内学者提出的DSSA有：北京邮电大学周莹新博士提出的电信软件的体系结构，北京航空航天大学金茂忠教授等人提出的测试环境的体系结构等。

### ◇ 软件体系结构的应用现状

#### ◎ 软件体系结构支持工具

几乎每种体系结构都有相应的支持工具，如Unicon, Aesop等体系结构支持环境，C2的支持环境ArchStudio, 支持主动连接件的Tracer工具等。

支持体系结构分析的工具，如支持静态分析的工具、支持类型检查的工具、支持体系结构层次依赖分析的工具、支持体系结构动态特性仿真工具、体系结构性能仿真工具等。

### ◇ 软件体系结构的应用现状

#### ◎ 软件产品线体系结构（1）

产品线代表着一组具有公共的系统需求集的软件系统，它们都是根据基本的用户需求对标准的产品线构架进行定制，将可重用构件与系统独有的部分集成而得到的。

软件产品线是一个十分适合专业的软件开发组织的软件开发方法，能有效地提高软件生产率和质量、缩短开发时间、降低总开发成本。

### ◇ 软件体系结构的应用现状

#### ◎ 软件产品线体系结构（2）

软件体系结构有利于形成完整的软件产品线。

体系结构在软件产品线的开发中具有至关重要的作用，在这种开发生产中，基于同一个软件体系结构，可以创建具有不同功能的多个系统。

### ◇ 软件体系结构的应用现状

#### ◎ 建立评价软件体系结构的方法

目前，常用的三个软件体系结构评估方法是：

- (1) 体系结构权衡分析方法（ATAM方法）
- (2) 软件体系结构分析方法（SAAM方法）
- (3) 中间设计的积极评审（ARID方法）

### ◇ 软件体系结构的应用现状

目前，软件体系结构尚处在迅速发展之中，越来越多的研究人员正在把注意力投向软件体系结构的研究。关于软件体系结构的研究工作主要在国外展开的，国内到目前为止对于软件体系结构的研究尚处在起步阶段。软件体系结构在国内未引起人们广泛注意的原因主要有两点：

(1) 软件体系结构从表面上看起来是一个老话题，似乎没有新东西。

(2) 与国外相比，国内对大型和超大型复杂软件系统开发的经历相对较少，对软件危机的灾难性体会没有国外深刻，因而对软件体系结构研究的重要性的认识还不很充分。

- 1、根据自己的经验，谈谈对软件危机的看法。
- 2、就项目管理方面而言，软件重用项目与非重用项目有哪些不同之处。
- 3、实际参与/组织一个软件重用项目的开发，然后总结你是如何组织该项目的开发的。
- 4、为什么要研究软件体系结构？
- 5、根据软件体系结构的定义，你认为软件体系结构的模型应该由哪些部分组成？
- 6、在软件体系结构的研究和应用中，你认为还有哪些不足之处？



# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

#### ◇ 软件体系结构建模的种类

◎ 结构模型

◎ 框架模型

◎ 动态模型

◎ 过程模型

◎ 功能模型

#### ◇ 软件体系结构建模的种类

##### ◎ 结构模型

这是一个最直观、最普遍的建模方法。这种方法以体系结构的构件、连接件和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质等。

研究结构模型的核心是体系结构描述语言。

#### ◇ 软件体系结构建模的种类

##### ◎ 框架模型

框架模型与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。

框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。

#### ◇ 软件体系结构建模的种类

##### ◎ 动态模型

动态模型是对结构或框架模型的补充，研究系统的“大颗粒”的行为性质。例如，描述系统的重新配置或演化。动态可以指系统总体结构的配置、建立或拆除通信通道或计算的过程。

#### ◇ 软件体系结构建模的种类

##### ◎ 过程模型

过程模型研究构造系统的步骤和过程。

结构是遵循某些过程脚本的结果。

#### ◇ 软件体系结构建模的种类

##### ◎ 功能模型

功能模型认为体系结构是由一组功能构件按层次组成，下层向上层提供服务。

功能模型可以看作是一种特殊的框架模型。

#### ◇ “4+1”模型概述

视图



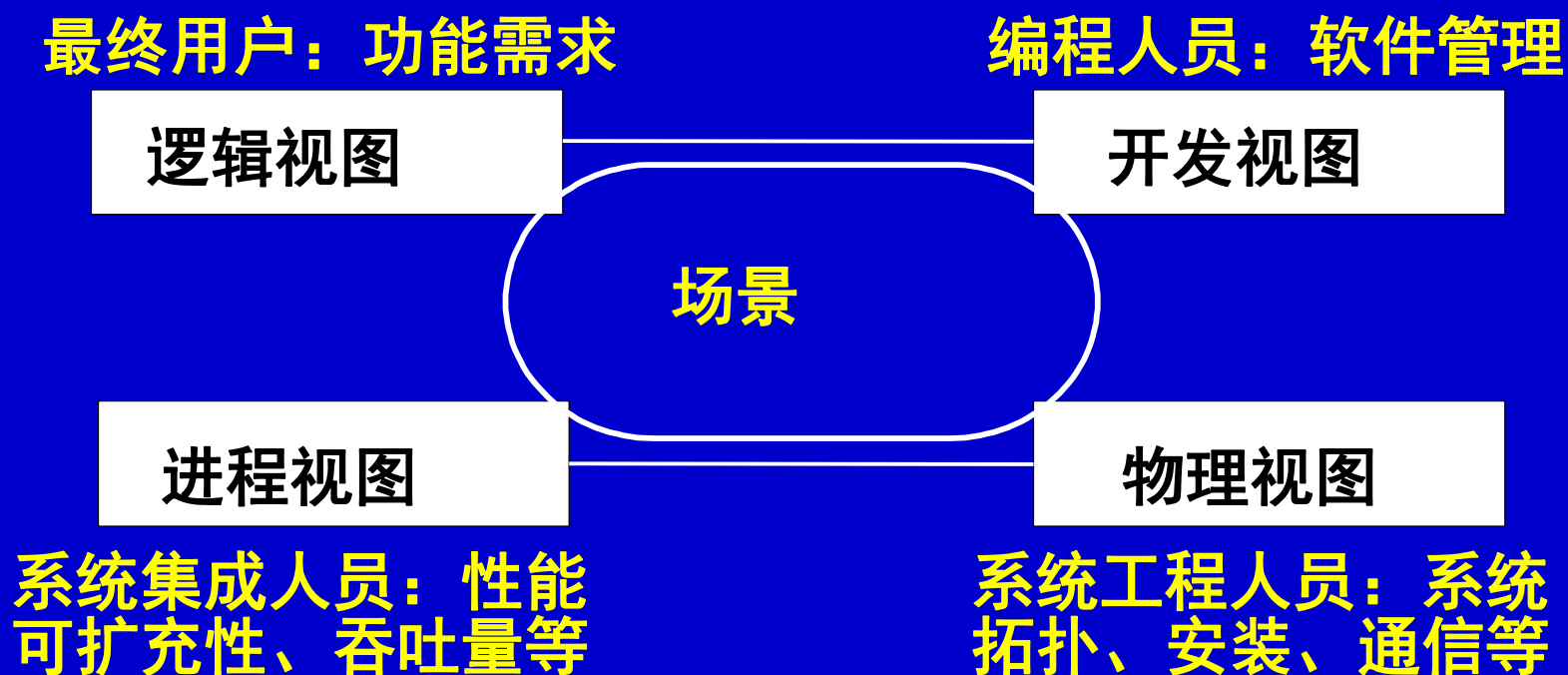
Kruchten在1995年提出了“4+1”的视图模型。

“4+1”视图模型从5个不同的视角包括逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件体系结构。

每一个视图只关心系统的一个侧面，5个视图结合在一起才能反映系统的软件体系结构的全部内容。



#### ◇ “4+1”模型概述



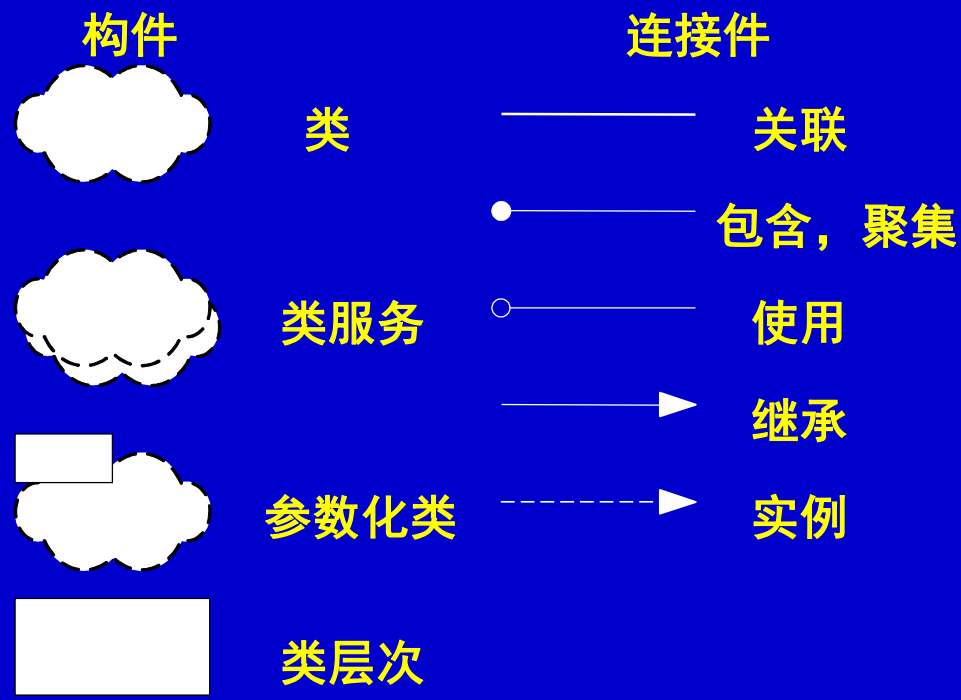
#### ◇ 逻辑视图

逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。

在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表逻辑视图，用类图来描述逻辑视图。

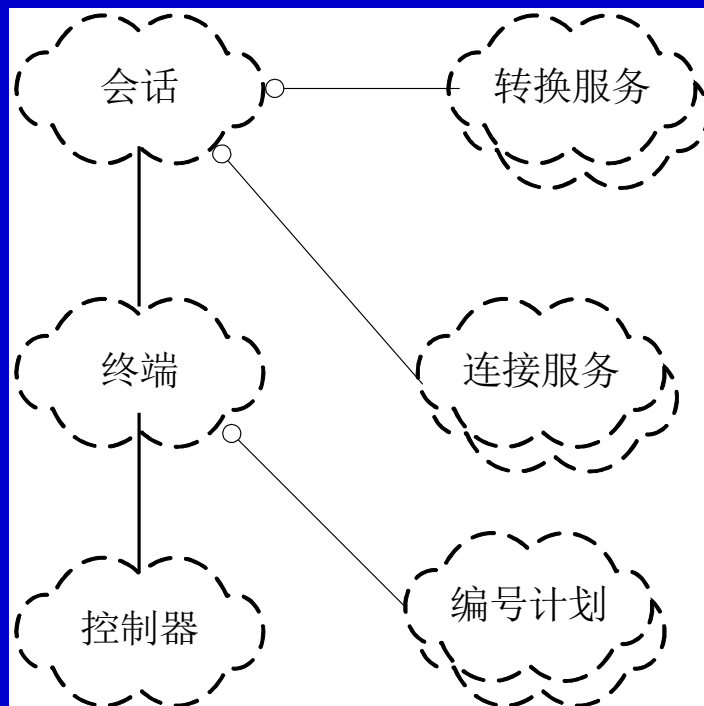
### ◇ 逻辑视图

可以从Booch标记法中导出逻辑视图的标记法，只是从体系结构级的范畴来考虑这些符号，用Rational Rose进行体系结构设计。



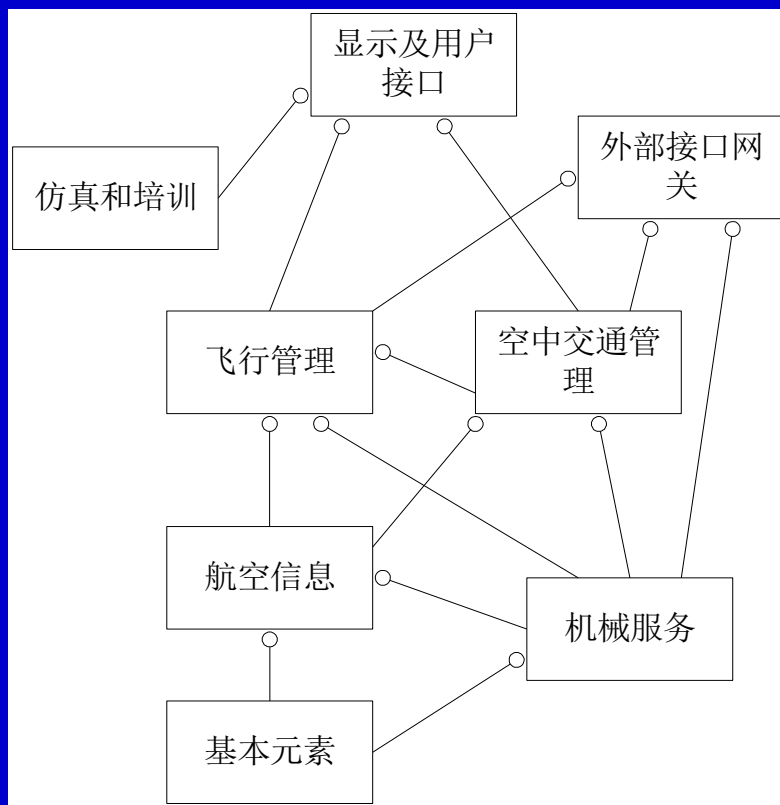
#### ◇ 逻辑视图

逻辑视图中使用的风格为面向对象的风格，逻辑视图设计中要注意的主要问题是要保持一个单一的、内聚的对象模型贯穿整个系统。



### ◇ 逻辑视图

对于规模更大的系统来说，体系结构级中包含数十甚至数百个类。



#### ◇ 开发视图

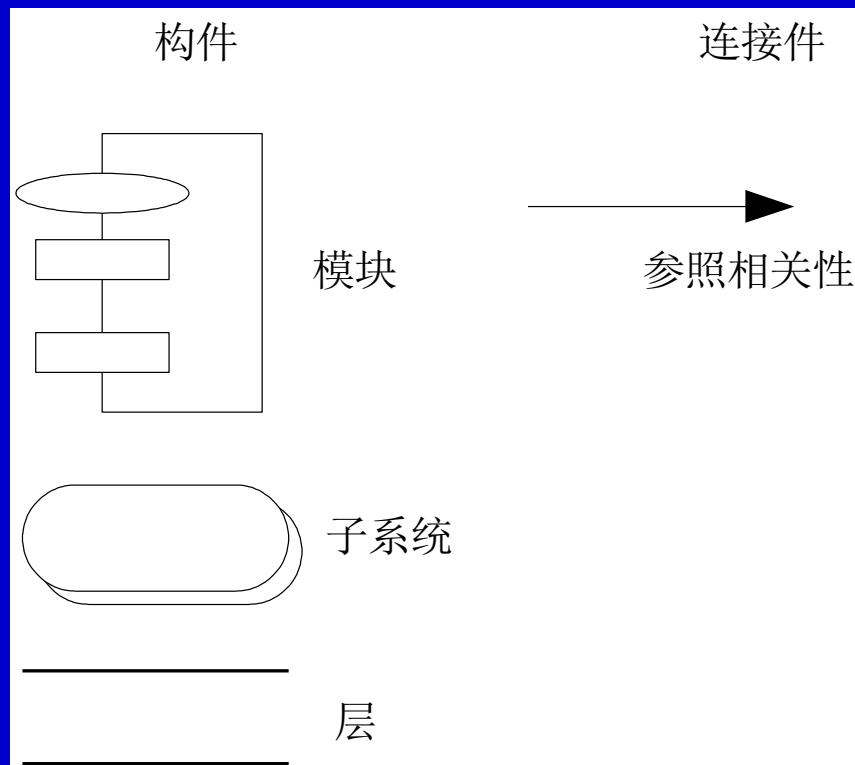
开发视图也称模块视图，主要侧重于软件模块的组织和管理。

开发视图要考虑软件内部的需求，如软件开发的容易性、软件的重用和软件的通用性，要充分考虑由于具体开发工具的不同而带来的局限性。

开发视图通过系统输入输出关系的模型图和子系统图来描述。

### ◇ 开发视图

与逻辑视图一样，可以使用Booch标记法中某些符号来表示开发视图。



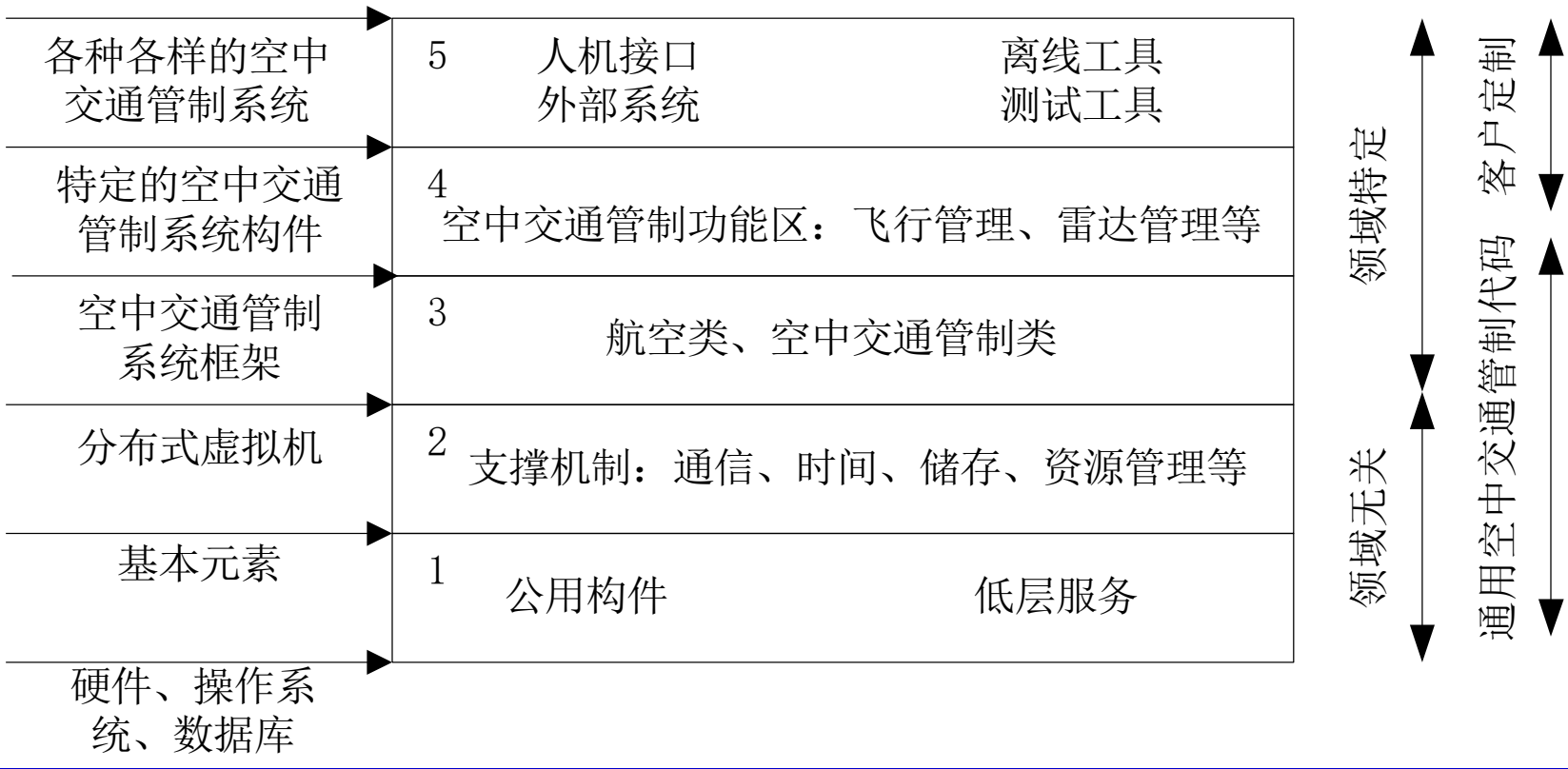
#### ◇ 开发视图

在开发视图中，最好采用4-6层子系统，而且每个子系统仅仅能与同层或更低层的子系统通讯，这样可以使每个层次的接口既完备又精练，避免了各个模块之间很复杂的依赖关系。

设计时要充分考虑，对于各个层次，层次越低，通用性越强，这样，可以保证应用程序的需求发生改变时，所做的改动最小。开发视图所用的风格通常是层次结构风格。



### ◇ 开发视图



#### ◇ 进程视图

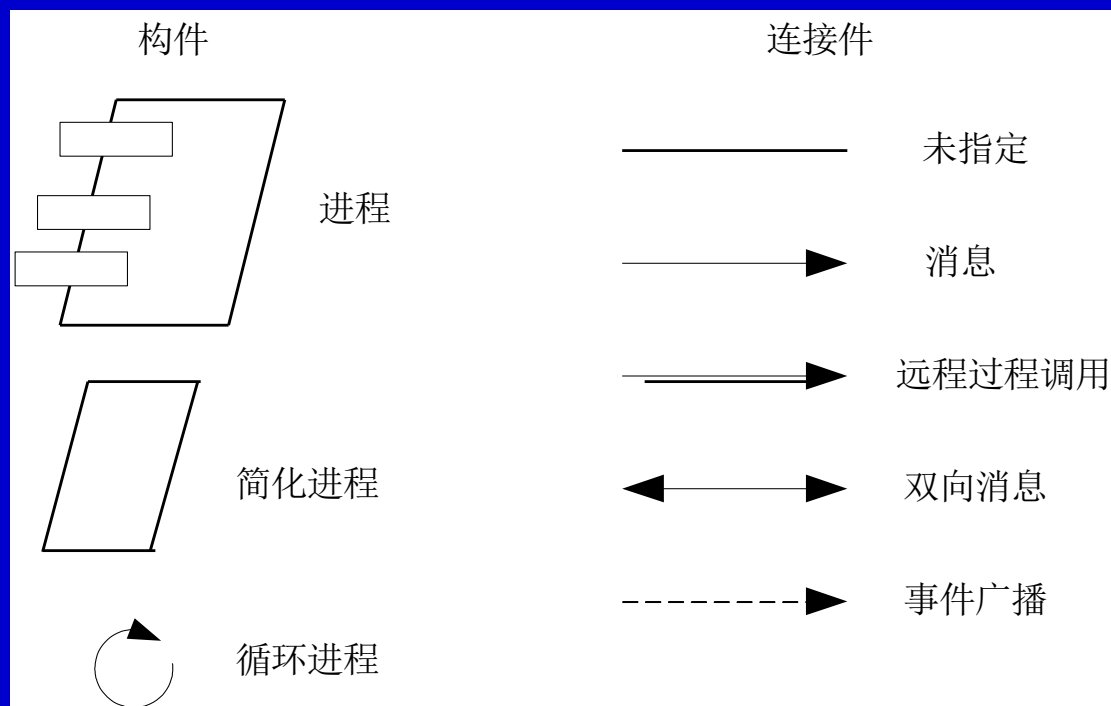
进程视图侧重于系统的运行特性，主要关注一些非功能性的需求。

进程视图强调并发性、分布性、系统集成性和容错能力，以及从逻辑视图中的主要抽象如何适合进程结构。它也定义逻辑视图中的各个类的操作具体是在哪一个线程中被执行的。

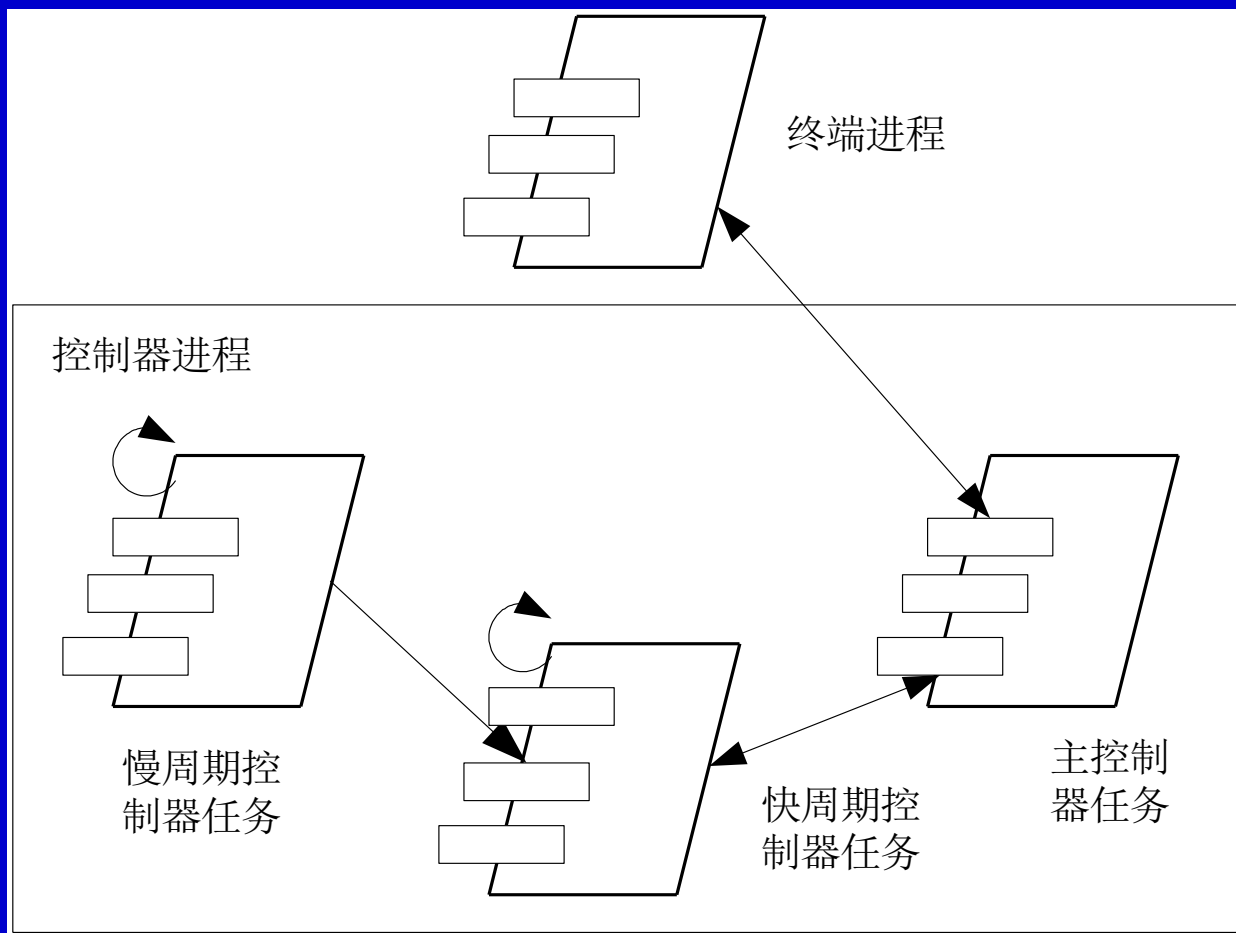
进程视图可以描述成多层抽象，每个级别分别关注不同的方面。在最高层抽象中，进程结构可以看作是构成一个执行单元的一组任务。它可看成一系列独立的，通过逻辑网络相互通信的程序。它们是分布的，通过总线或局域网、广域网等硬件资源连接起来。

### ◇ 进程视图

通过扩展Booch对Ada任务的表示法，来表示进程视图。



### ◇ 进程视图



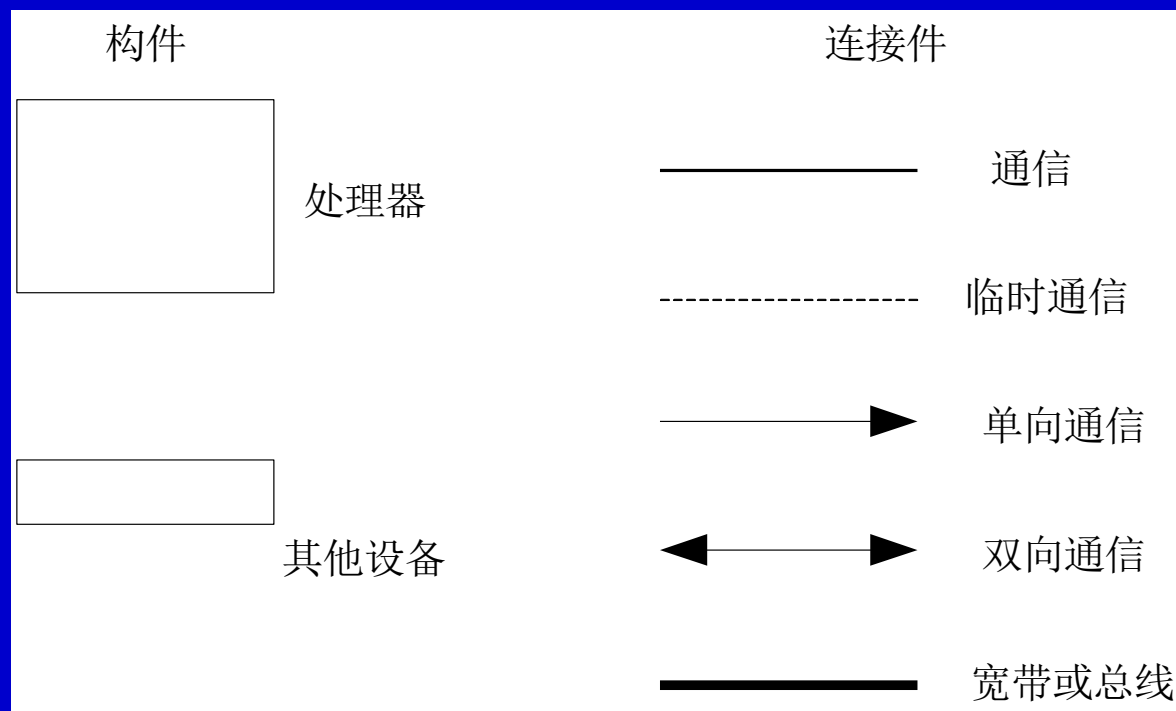
#### ◇ 物理视图

物理视图主要考虑如何把软件映射到硬件上，它通常要考虑到系统性能、规模、可靠性等。解决系统拓扑结构、系统安装、通讯等问题。

当软件运行于不同的节点上时，各视图中的构件都直接或间接地对应于系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其他视图的影响最小。

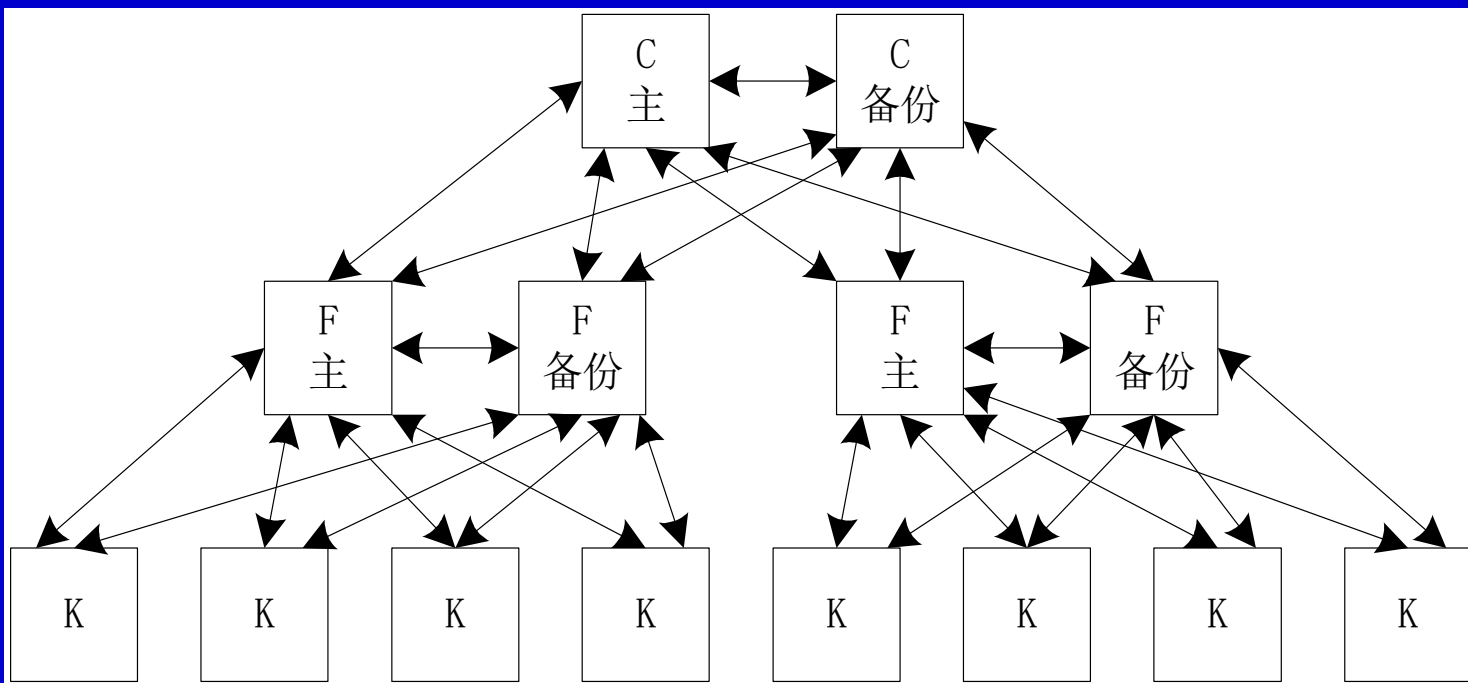
### ◇ 物理视图

大型系统的物理视图可能会变得十分混乱，因此可以与进程视图的映射一道，以多种形式出现，也可单独出现。



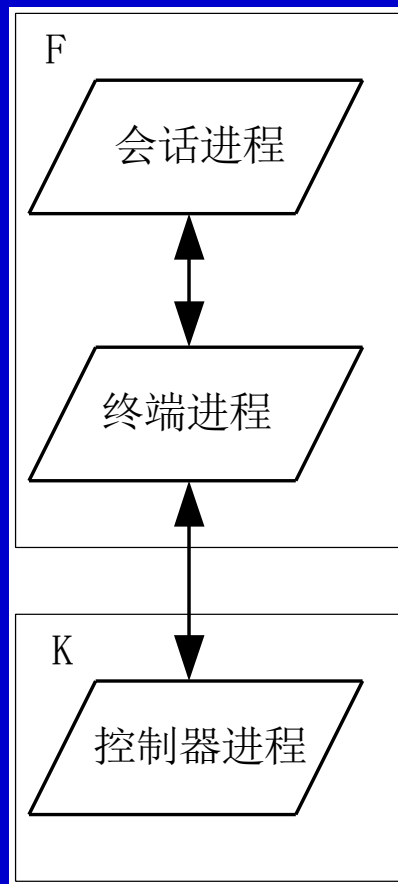
### ◇ 物理视图

ACS系统的物理视图



### ◇ 物理视图

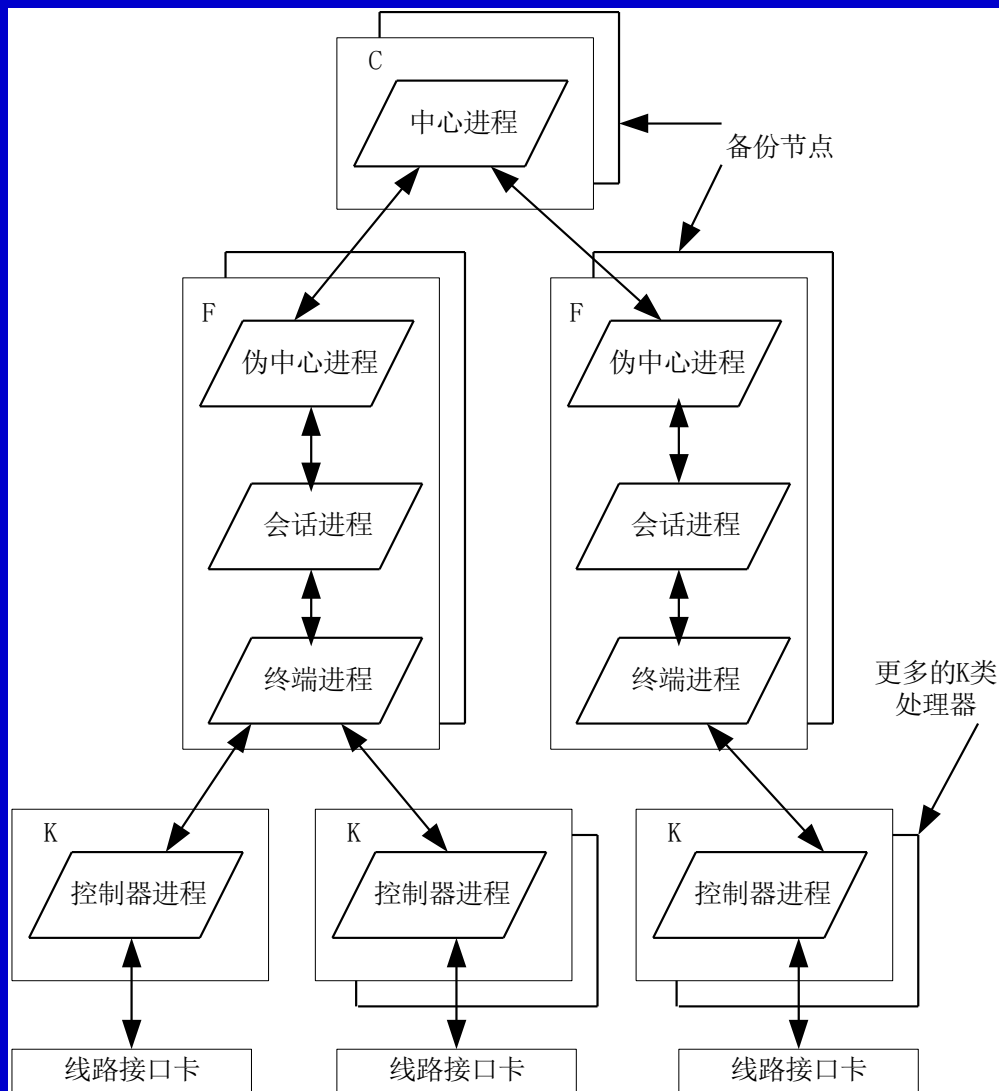
具有进程分配  
的小型ACS系统  
的物理视图





### ◇ 物理视图

具有进程分  
配的大型  
ACS系统的  
物理视图



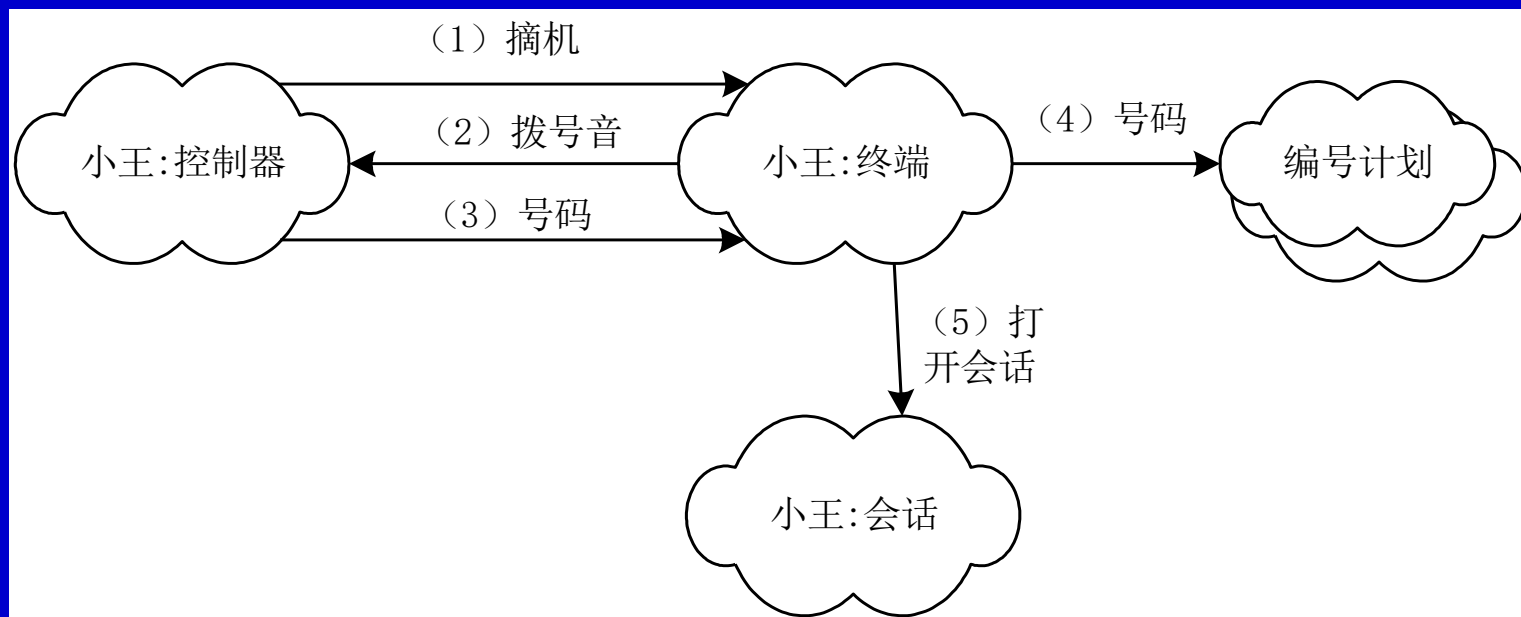
#### ◇ 场景

场景可以看作是那些重要系统活动的抽象，它使四个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。在开发体系结构时，它可以帮助设计者找到体系结构的构件和它们之间的作用关系。同时，也可以用场景来分析一个特定的视图，或描述不同视图构件间是如何相互作用的。

场景可以用文本表示，也可以用图形表示。

#### ◇ 场景

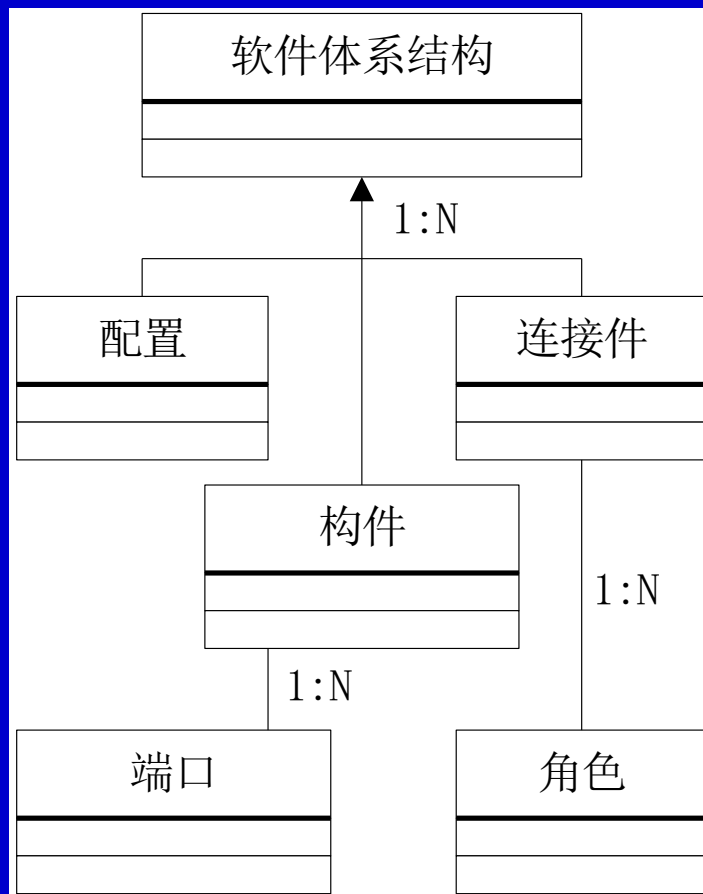
##### 本地呼叫场景的一个原型



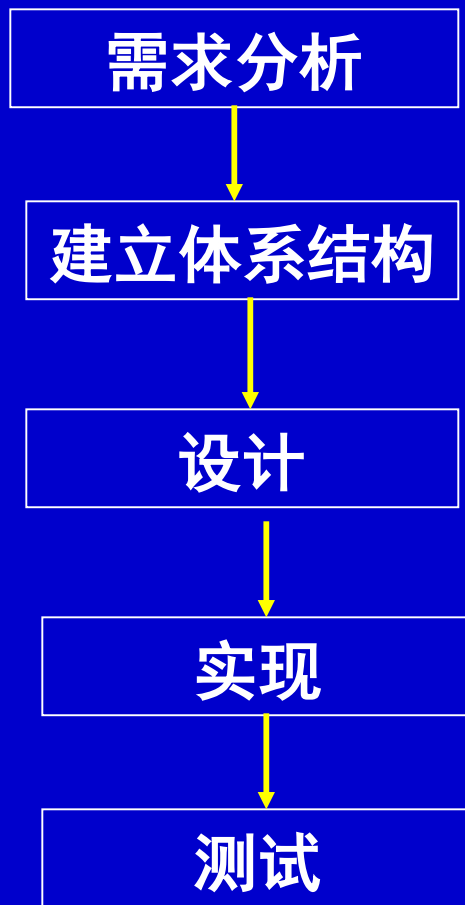
#### ◇ 小结

逻辑视图和开发视图描述系统的静态结构，而进程视图和物理视图描述系统的动态结构。

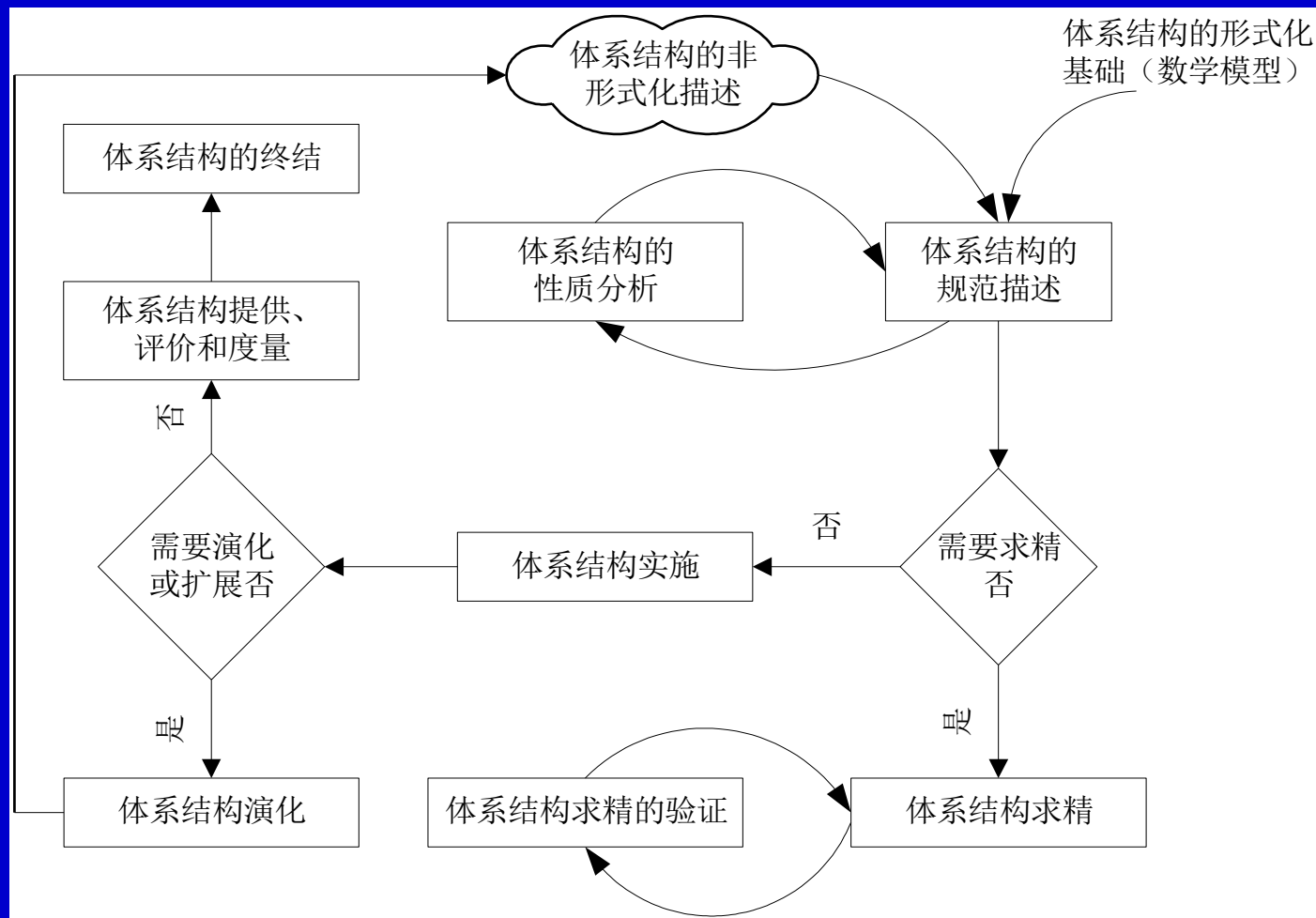
对于不同的软件系统来说，侧重的角度也有所不同。例如，对于管理信息系统来说，比较侧重于从逻辑视图和开发视图来描述系统，而对于实时控制系统来说，则比较注重于从进程视图和物理视图来描述系统。



#### ◇ 软件过程



### ◇ 生命周期模型



选读

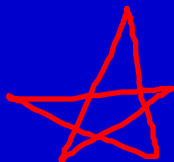


- 1、选择一个规模合适的系统，为其建立“4+1”模型。
- 2、引入了软件体系结构以后，传统软件过程发生了哪些变化？这种变化有什么好处？
- 3、软件体系结构的生命周期模型与软件生命周期模型有什么关系？

# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

### ◇ 定义



软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式。

体系结构风格定义了一个系统家族，即一个体系结构定义一个词汇表和一组约束。词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。

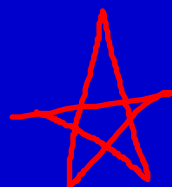
体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

#### ◇ 讨论体系结构风格时要回答的问题

- ◎ 构件和连接件的类型是什么？
- ◎ 可容许的结构模式是什么？
- ◎ 基本的计算模型是什么？
- ◎ 风格的基本不变性是什么？
- ◎ 其使用的常见例子是什么？
- ◎ 使用此风格的优缺点是什么？
- ◎ 其常见的特例是什么？

### ◇ 经典的体系结构风格

- ◎ 数据流风格：批处理序列；管道/过滤器。
- ◎ 调用/返回风格：主程序/子程序；面向对象风格；层次结构。
- ◎ 独立构件风格：进程通讯；事件系统。
- ◎ 虚拟机风格：解释器；基于规则的系统。
- ◎ 仓库风格：数据库系统；超文本系统；黑板系统。



#### ◇ 管道和过滤器

每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。

这里的构件被称为过滤器，这种风格的连接件就象是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。



#### ◇ 管道和过滤器风格的优点

- ◎ 使得软构件具有良好的隐蔽性和高内聚、低耦合的特点；
- ◎ 允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；
- ◎ 支持软件重用。只要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来；
- ◎ 系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉；
- ◎ 允许对一些如吞吐量、死锁等属性的分析；
- ◎ 支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。



#### ◇ 管道和过滤器的缺点

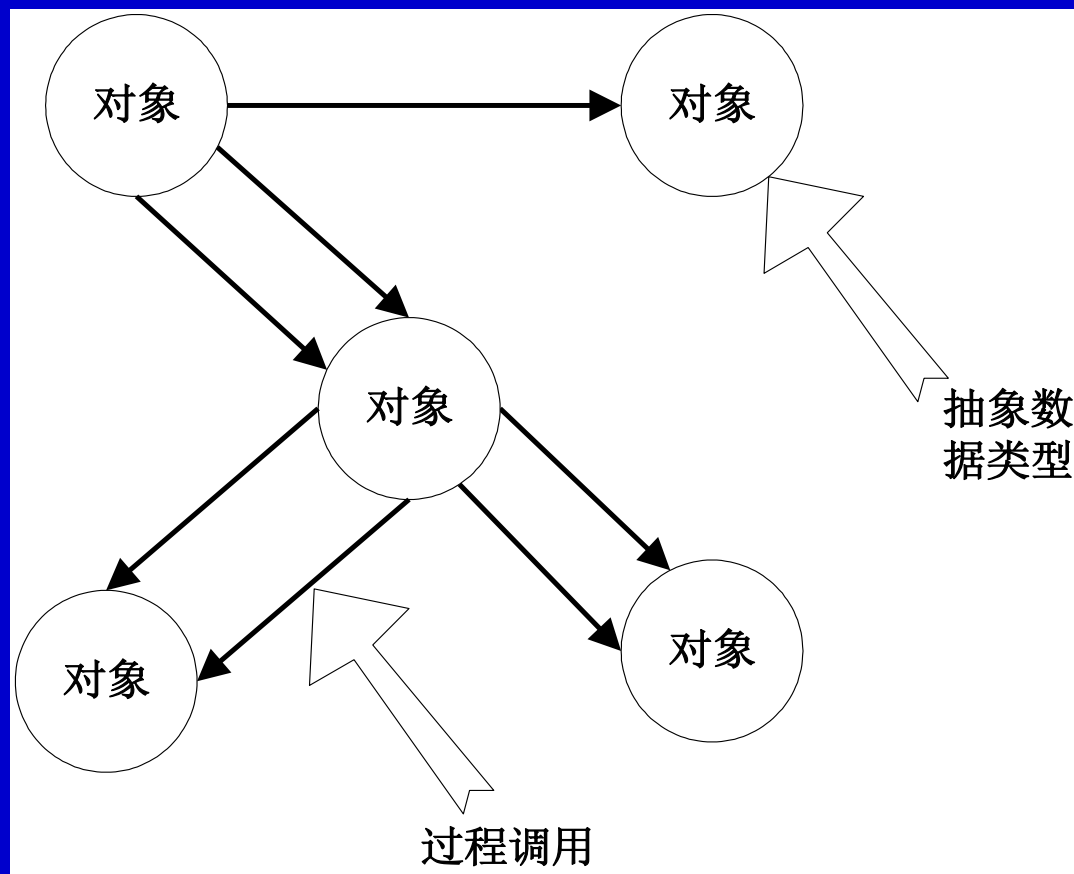
- ◎ 通常导致进程成为批处理的结构。这是因为虽然过滤器可增量式地处理数据，但它们是独立的，所以设计者必须将每个过滤器看成一个完整的从输入到输出的转换；
- ◎ 不适合处理交互的应用。当需要增量地显示改变时，这个问题尤为严重；
- ◎ 因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性。

#### ◇ 数据抽象和面向对象组织

这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。

这种风格的构件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的构件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。

### ◇ 数据抽象和面向对象组织



#### ◇ 面向对象系统的优点

- ◎ 因为对象对其它对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其它的对象；
- ◎ 设计者可将一些数据存取操作的问题分解成一些交互的代理程序的集合。

#### ◇ 面向对象系统的缺点

- ◎ 为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象；
- ◎ 必须修改所有显式调用它的其它对象，并消除由此带来的一些副作用。例如，如果A使用了对象B，C也使用了对象B，那么，C对B的使用所造成的对A的影响可能是料想不到的。

#### ◇ 基于事件的隐式调用

构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

这种风格的构件是一些模块，模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。

这种风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

#### ◇ 基于事件隐式调用的优点

- ◎ 为软件重用提供了强大的支持。当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中。
- ◎ 为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其它构件的接口。

#### ◇ 基于事件的隐式调用的缺点

- ◎ 构件放弃了对系统计算的控制。一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的构成，它也不能保证这些过程被调用的顺序。
- ◎ 数据交换的问题。有时数据可被一个事件传递，但另一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。在这些情况下，全局性能和资源管理便成了问题。
- ◎ 既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理存在问题。

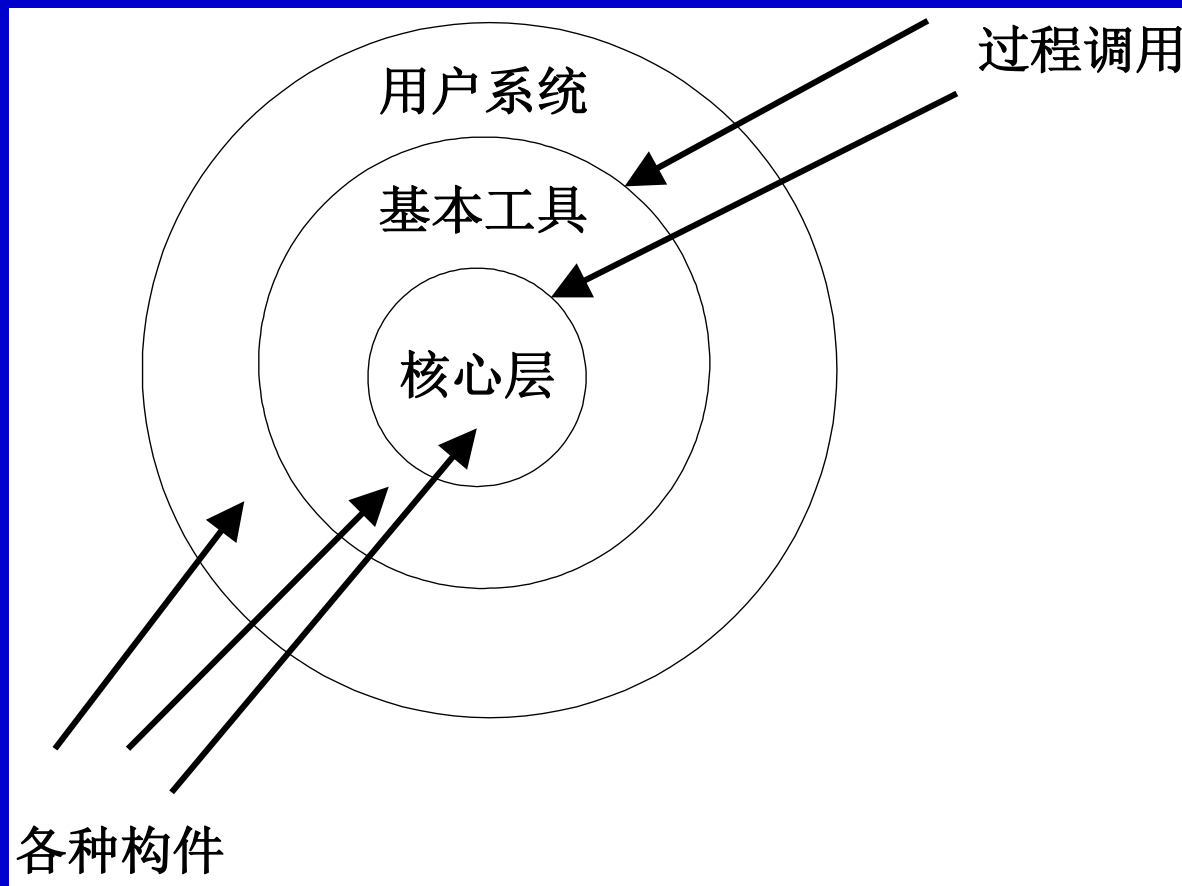


#### ◇ 分层系统

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机（在另一些层次系统中层是部分不透明的）。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

这种风格支持基于可增加抽象层的设计。允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

### ◇ 分层系统



#### ◇ 分层系统的优点

- ◎ 支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解；
- ◎ 支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层；
- ◎ 支持重用。只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，而允许各种不同的实现方法。

#### ◇ 分层系统的缺点

- ◎ 并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来；
- ◎ 很难找到一个合适的、正确的层次抽象方法。

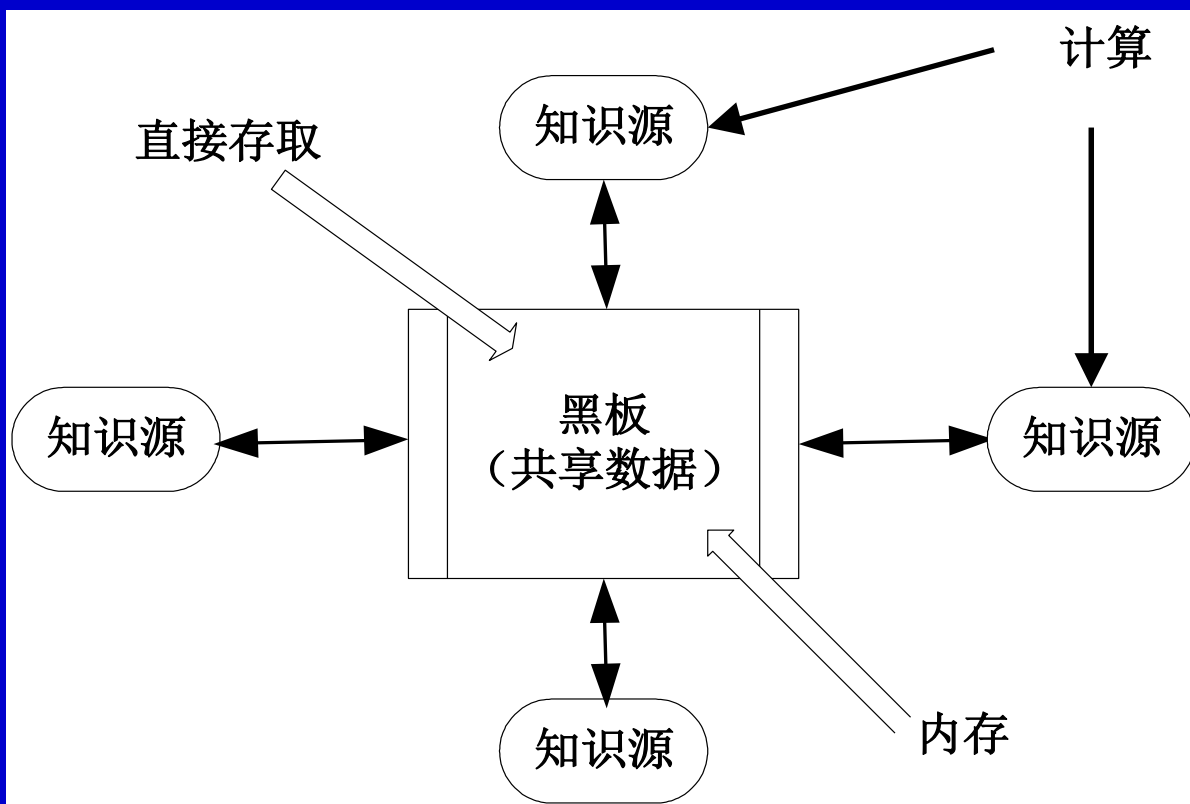
#### ◇ 仓库系统及知识库

在仓库风格中，有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存贮上执行，仓库与外构件间的相互作用在系统中会有大的变化。

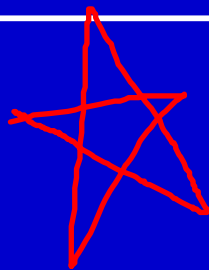
控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统。

黑板系统的  
概念

### ◇ 仓库系统及知识库



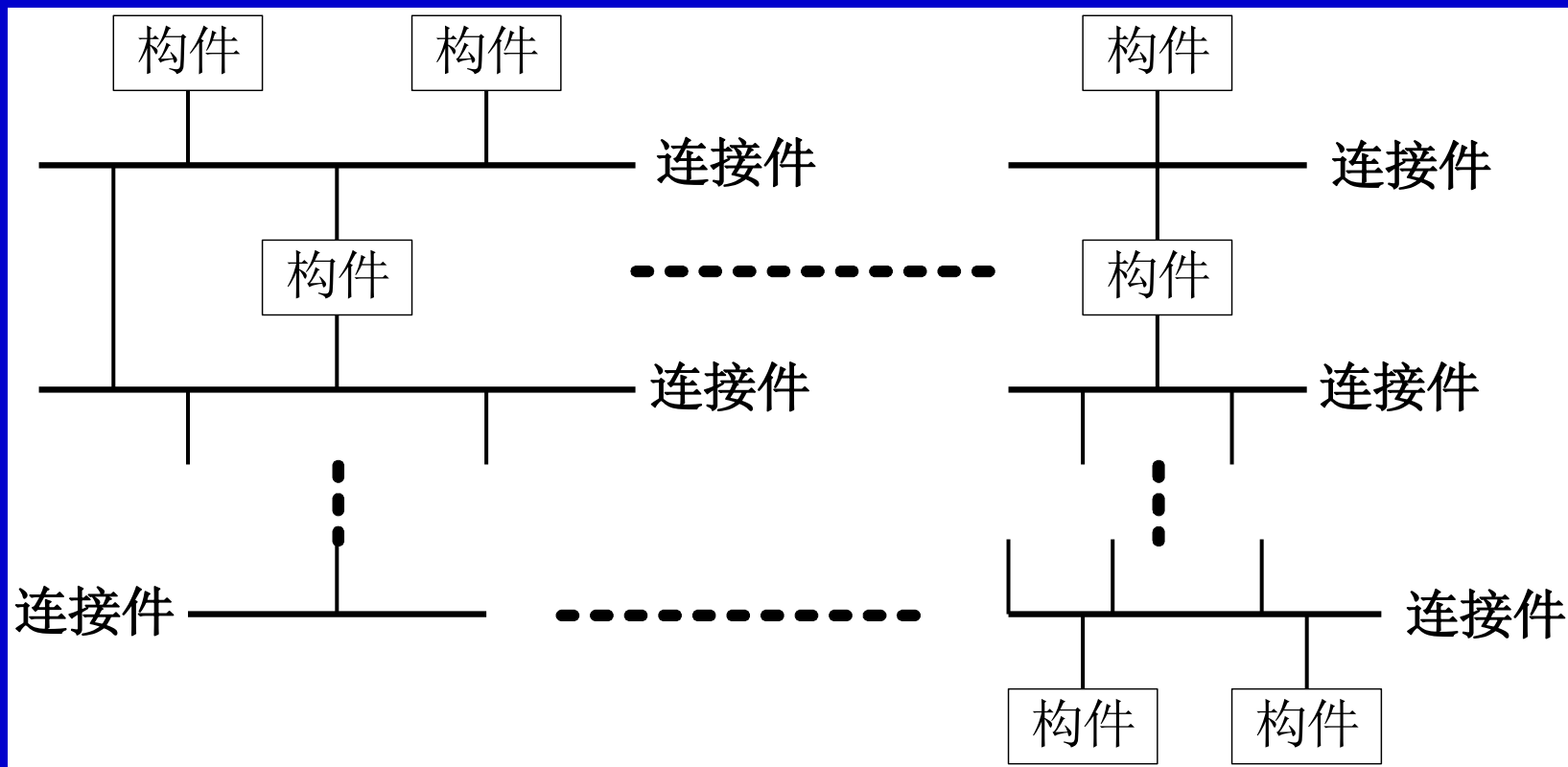
### ◇ C2风格



通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2风格中的系统组织规则如下：

- ◎ 系统中的构件和连接件都有一个顶部和一个底部；
- ◎ 构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；
- ◎ 一个连接件可以和任意数目的其它构件和连接件连接；
- ◎ 当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

### ◇ C2风格





#### ◇ C2风格的特点

- ◎ 系统中的构件可实现应用需求，并能将任意复杂度的功能封装在一起；
- ◎ 所有构件之间的通讯是通过以连接件为中介的异步消息交换机制来实现的；
- ◎ 构件相对独立，构件之间依赖性较少。系统中不存在某些构件将在同一地址空间内执行，或某些构件共享特定控制线程之类的相关性假设。

#### ◇ 产生背景

◎ 在集中式计算技术时代广泛使用的是大型机/小型机计算模型。它是通过一台物理上与宿主机相连接的非智能终端来实现宿主机上的应用程序。

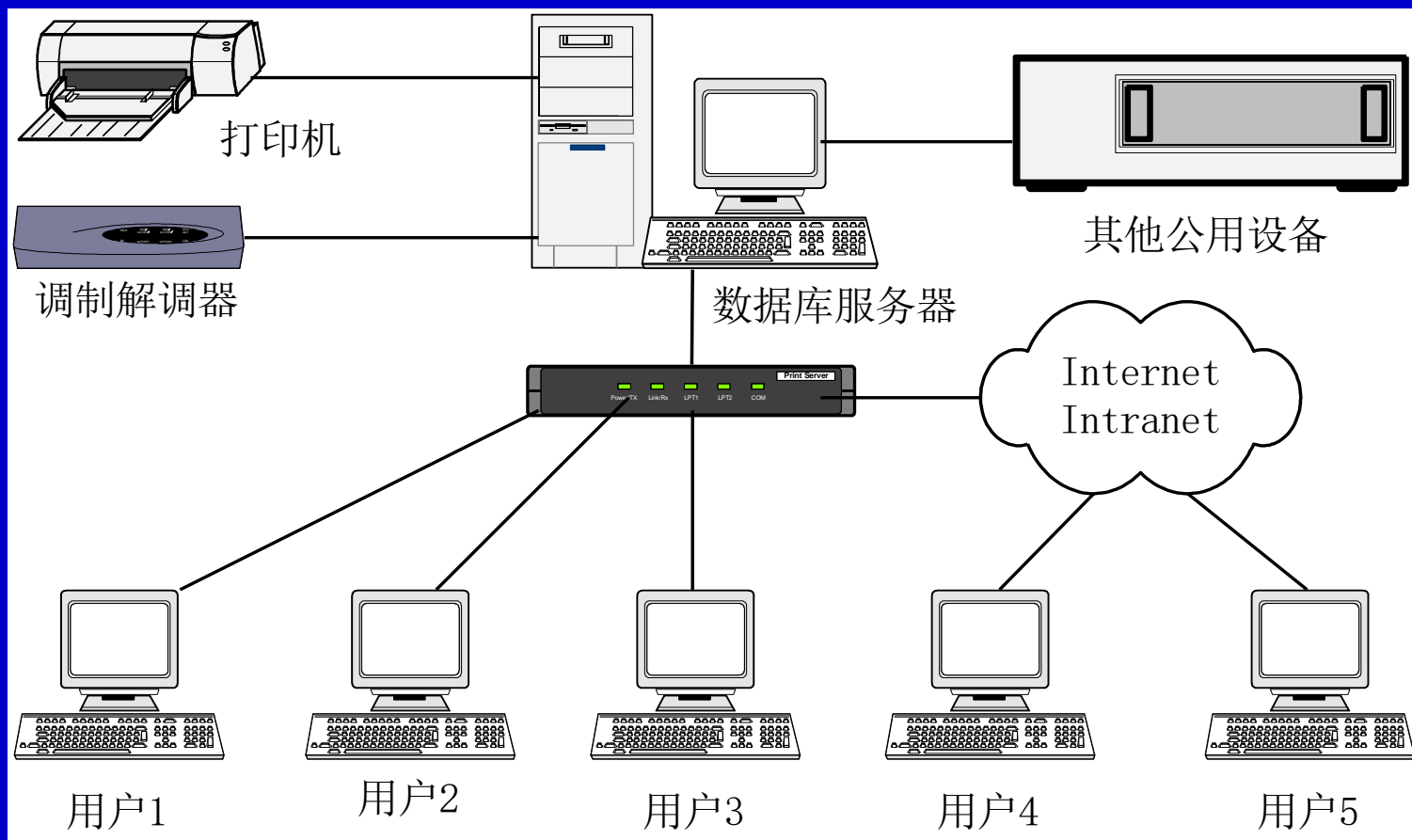
◎ 20世纪80年代以后，集中式结构逐渐被以PC机为主的微机网络所取代。个人计算机和工作站的采用，永远改变了协作计算模型，从而导致了分散的个人计算模型的产生。

#### ◇ 基本概念

◎ C/S软件体系结构是基于资源不对等，且为实现共享而提出来的，是20世纪90年代成熟起来的技术，C/S体系结构定义了工作站如何与服务器相连，以实现数据和应用分布到多个处理机上。

◎ C/S体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络。

### ◇ 体系结构



#### ◇ 任务分配

#### ◎ 服务器

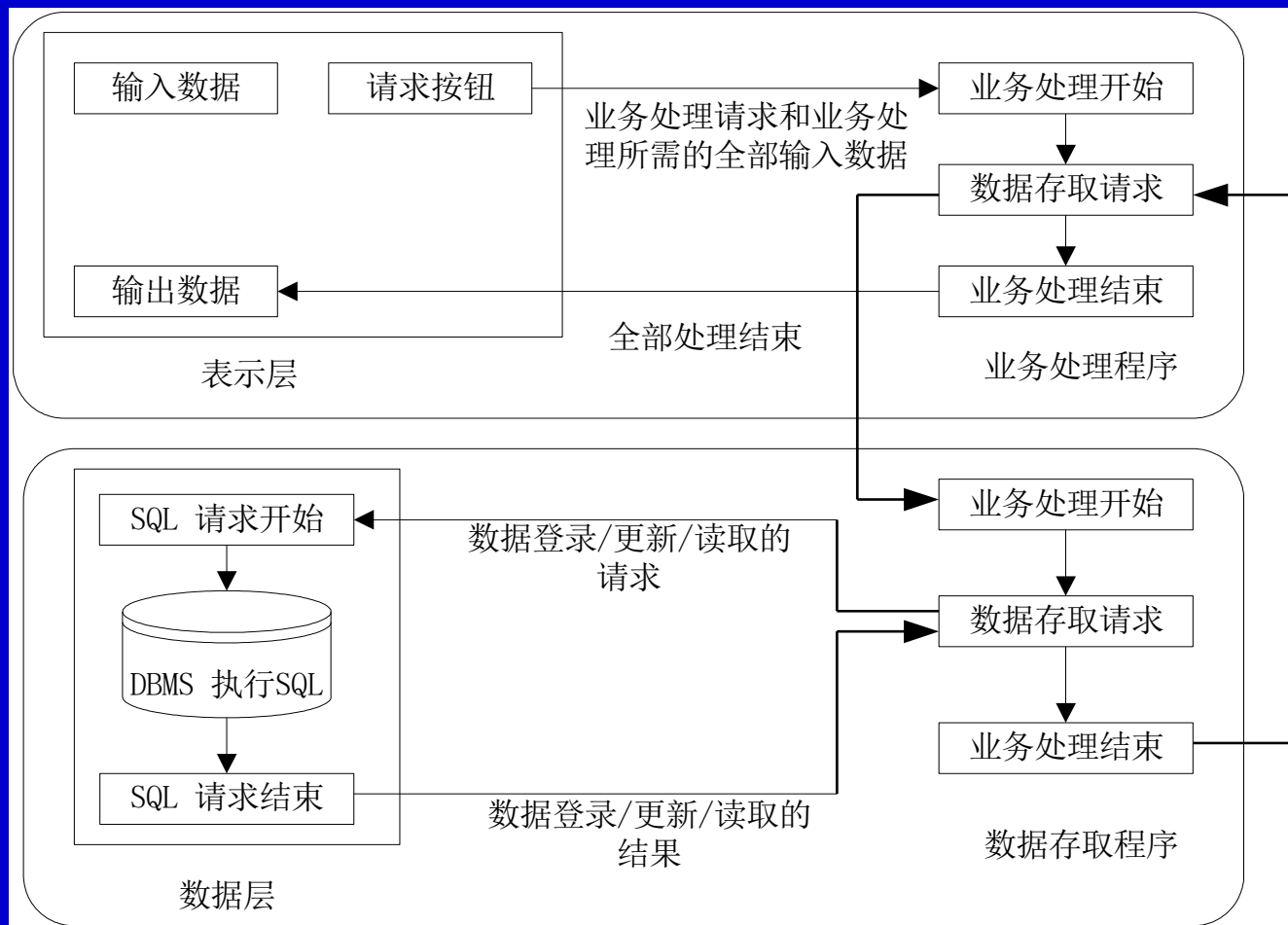
- (1) 数据库安全性的要求；
- (2) 数据库访问并发性的控制；
- (3) 数据库前端的客户应用程序的全局数据完整性规则；
- (4) 数据库的备份与恢复。

#### ◇ 任务分配

#### ◎ 客户应用程序

- (1) 提供用户与数据库交互的界面；
- (2) 向数据库服务器提交用户请求并接收来自数据库服务器的信息；
- (3) 利用客户应用程序对存在于客户端的数据执行应用逻辑要求。

### ◇ 处理流程



### ◇ 优点

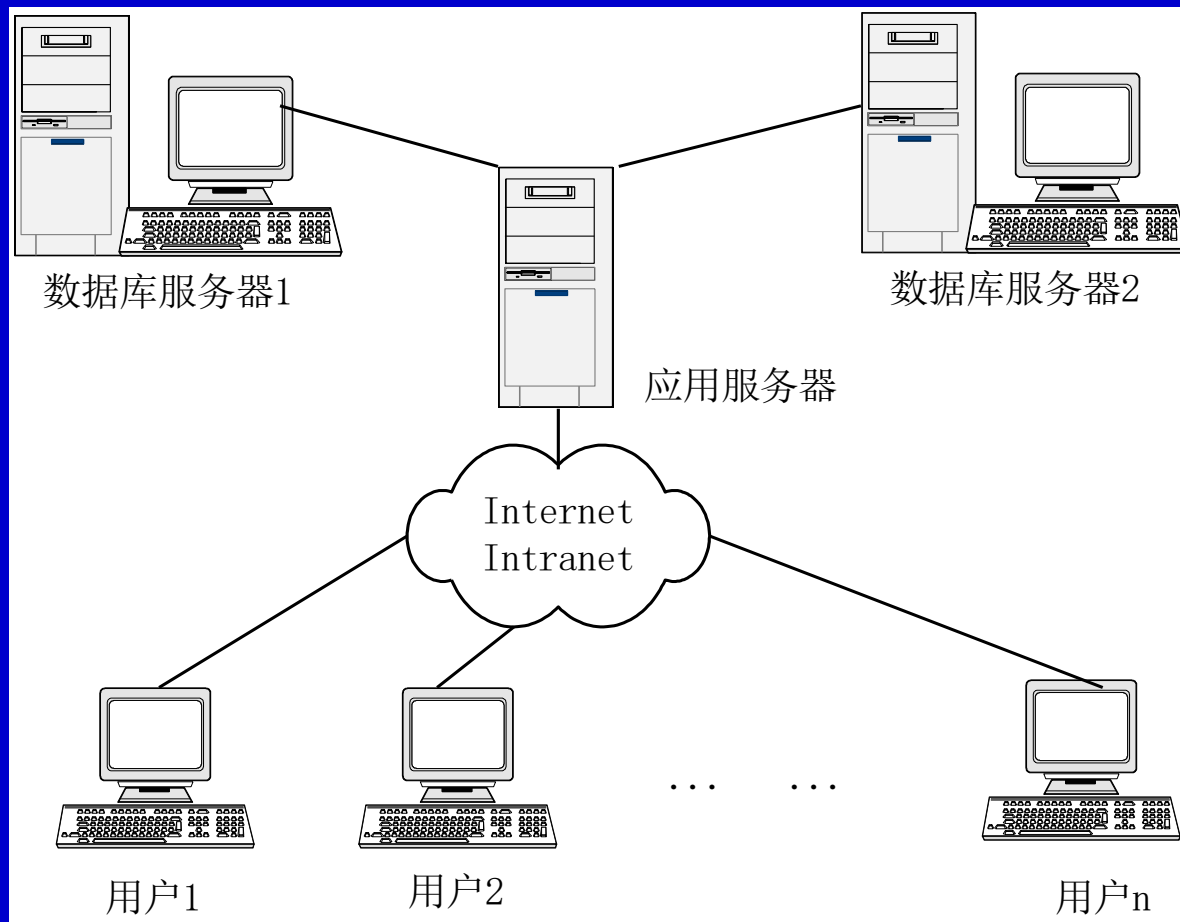
- ◎ C/S 体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。
- ◎ 系统的客户应用程序和服务构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。
- ◎ 在C/S体系结构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个DBMS进行编码。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。



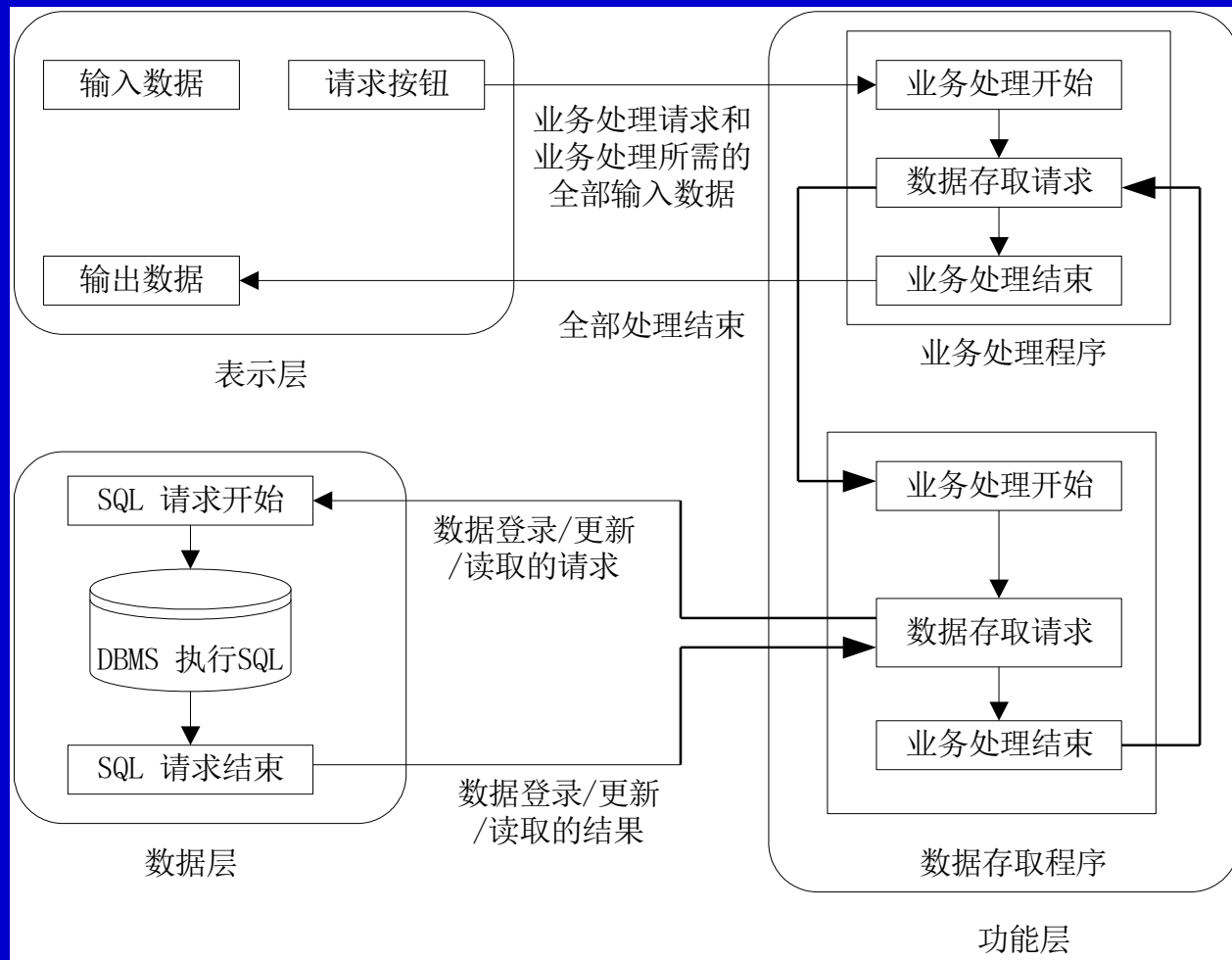
### ◇ 缺点

- ◎ 开发成本较高
- ◎ 客户端程序设计复杂
- ◎ 信息内容和形式单一
- ◎ 用户界面风格不一，使用繁杂，不利于推广使用
- ◎ 软件移植困难
- ◎ 软件维护和升级困难
- ◎ 新技术不能轻易应用

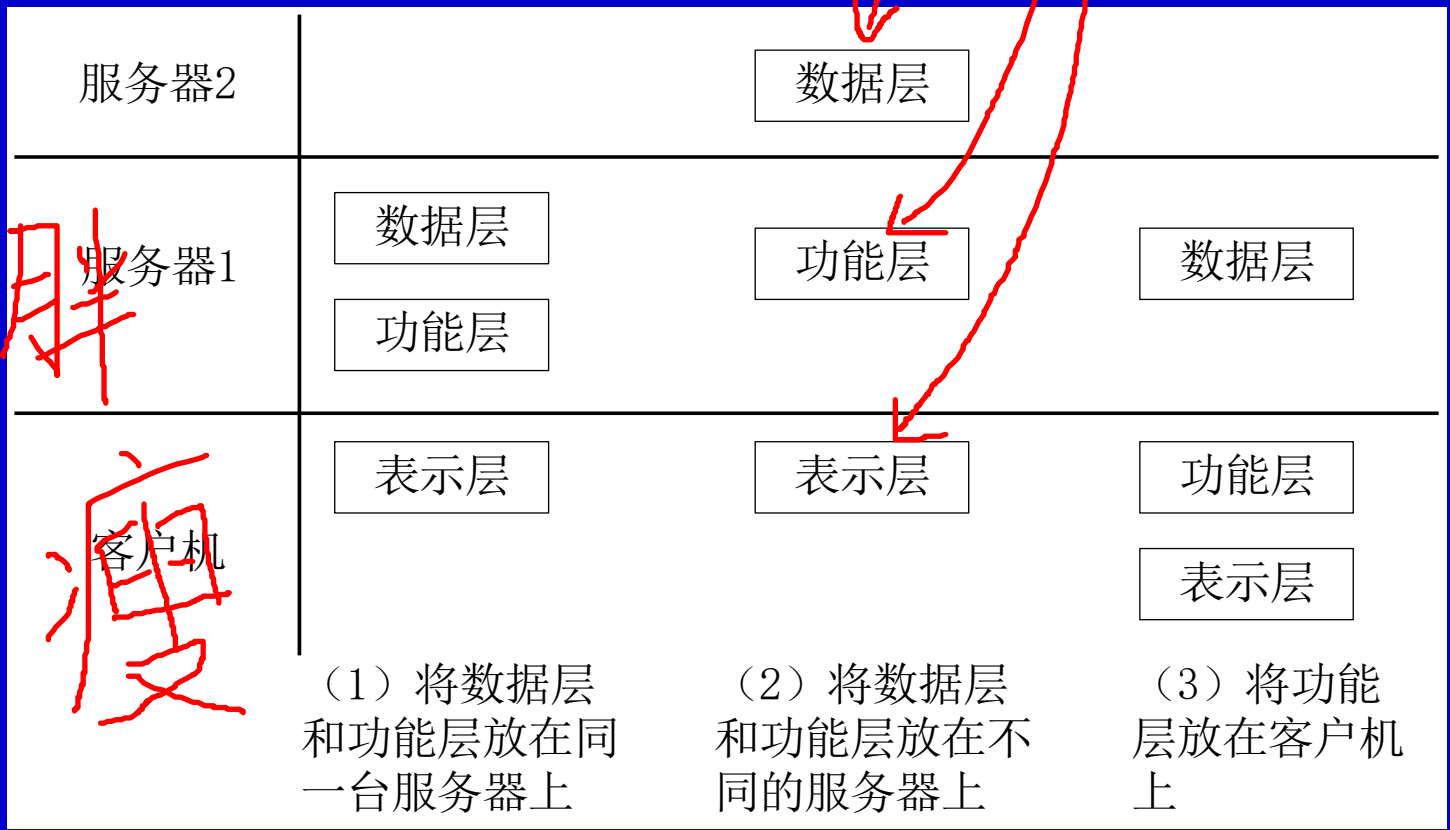
### ◇ 体系结构



### ◇ 处理流程



### ◇ 物理结构



### ◇ 应用实例

自学

#### ◇ 优点

- ◎ 允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，能提高系统和软件的可维护性和可扩展性。
- ◎ 允许更灵活有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层；并且这些平台和各个组成部分可以具有良好的可升级性和开放性。
- ◎ 应用的各层可以并行开发，可以选择各自最适合的开发语言。
- ◎ 利用功能层有效地隔离开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，为严格的安全管理奠定了坚实的基础。

#### ◇ 要注意的问题

- ◎ 三层C/S结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。
- ◎ 设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层C/S结构的关键问题。

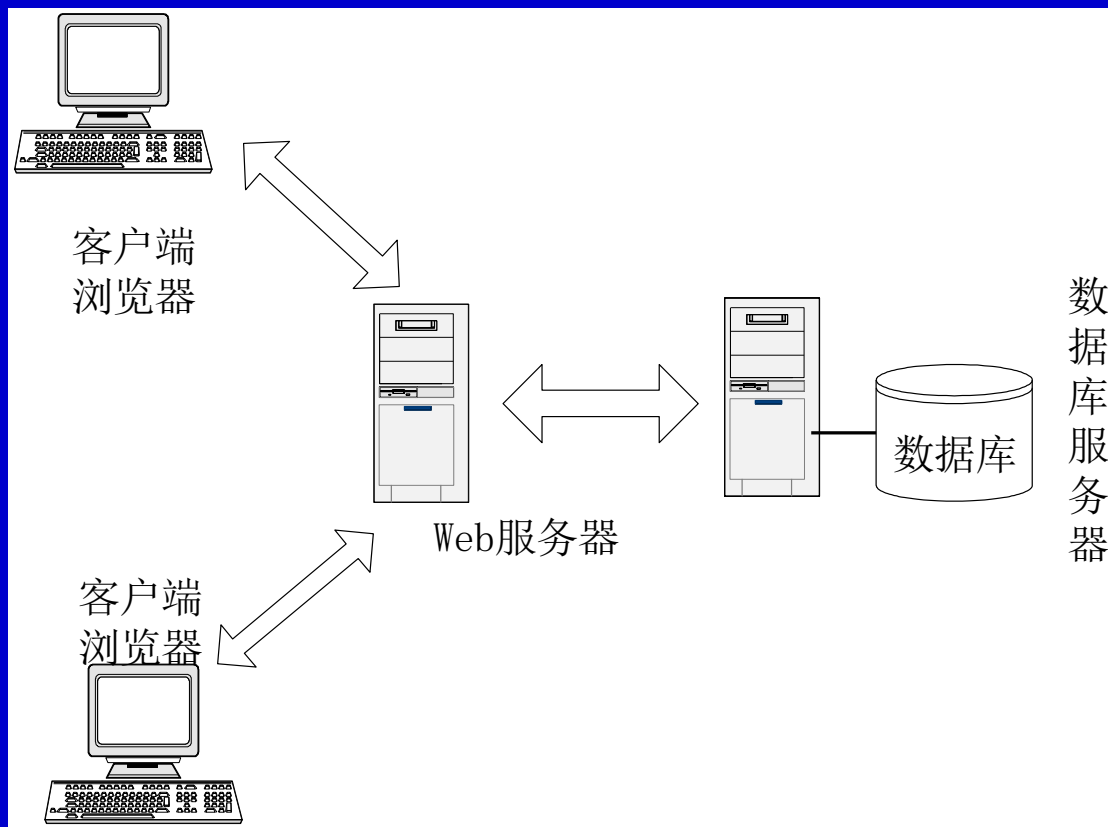
### ◇ 基本概念

◎ 浏览器/服务器（B/S）风格就是上述三层应用结构的一种实现方式，其具体结构为：浏览器/Web服务器/数据库服务器。

◎ B/S体系结构主要是利用不断成熟的WWW浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。从某种程度上来说，B/S结构是一种全新的软件体系结构。



### ◇ 体系结构



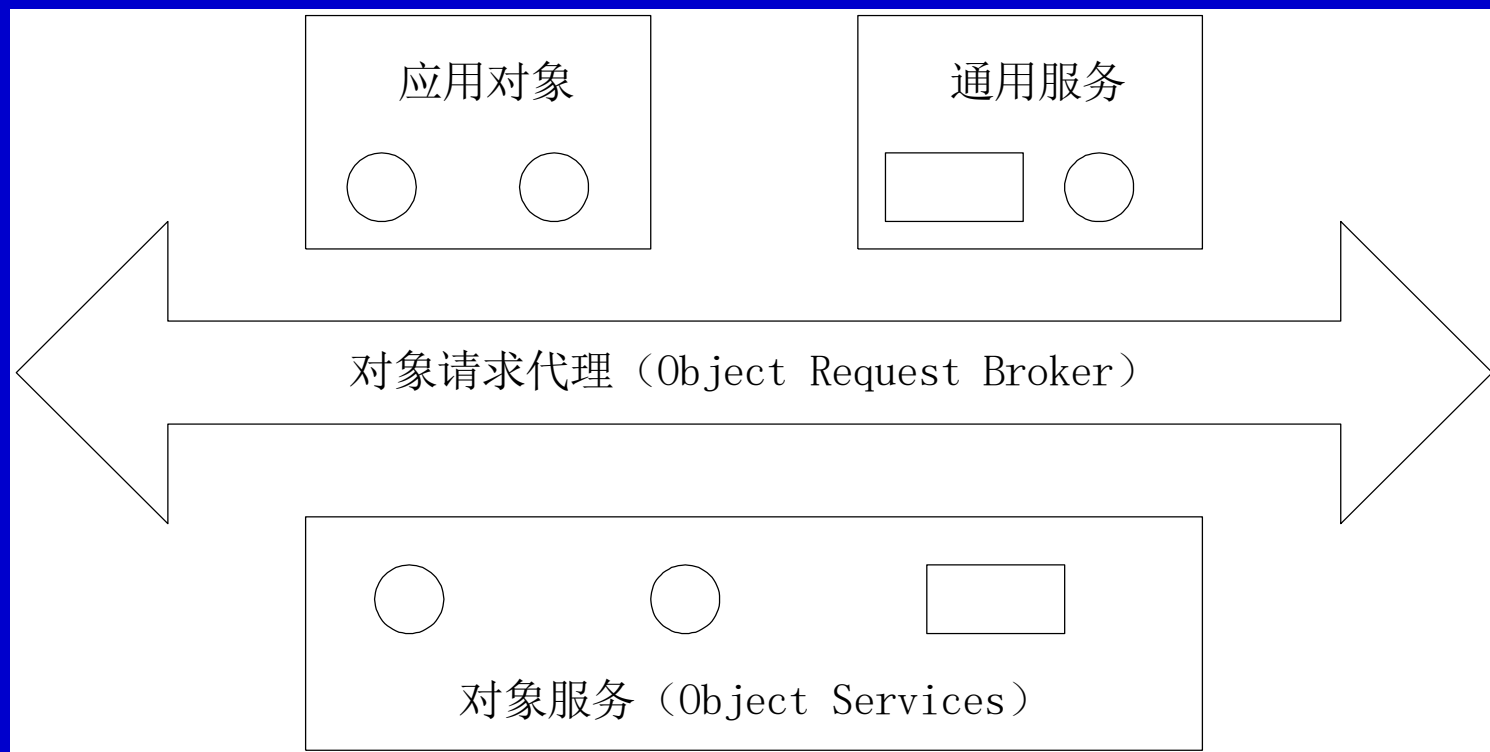
#### ◇ 优点

- ◎ 基于B/S体系结构的软件，系统安装、修改和维护全在服务器端解决。用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。
- ◎ B/S体系结构还提供了异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础。

#### ◇ 缺点

- ◎ B/S体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能。
- ◎ B/S体系结构的系统扩展能力差，安全性难以控制。
- ◎ 采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远地低于C/S体系结构。
- ◎ B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理(OLTP)应用。

### ◇ 对象管理结构



### ◇ CORBA技术规范

- ◎ 接口定义语言（IDL）
- ◎ 接口池（IR）
- ◎ 动态调用接口（DII）
- ◎ 对象适配器（OA）

## ◇ CORBA技术规范

### ◎ 接口定义语言

CORBA利用IDL统一地描述服务器对象（向调用者提供服务的对象）的接口。IDL本身也是面向对象的。它虽然不是编程语言，但它为客户对象（发出服务请求的对象）提供了语言的独立性，因为客户对象只需了解服务器对象的IDL接口，不必知道其编程语言。

IDL语言是CORBA规范中定义的一种中性语言，它用来描述对象的接口，而不涉及对象的具体实现。

在CORBA中定义了IDL语言到C、C++、SmallTalk和Java语言的映射。

#### ◇ CORBA技术规范

##### ◎ 接口池

CORBA的接口池包括了分布计算环境中所有可用的服务器对象的接口表示。它使动态搜索可用服务器的接口、动态构造请求及参数成为可能。

#### ◇ CORBA技术规范

##### ◎ 动态调用接口

CORBA的动态调用接口提供了一些标准函数以供客户对象动态创建请求、动态构造请求参数。客户对象将动态调用接口与接口池配合使用可实现服务器对象接口的动态搜索、请求及参数的动态构造与动态发送。当然，只要客户对象在编译之前能够确定服务器对象的IDL接口，CORBA也允许客户对象使用静态调用机制。显然，静态机制的灵活性虽不及动态机制，但执行效率却胜过动态机制。

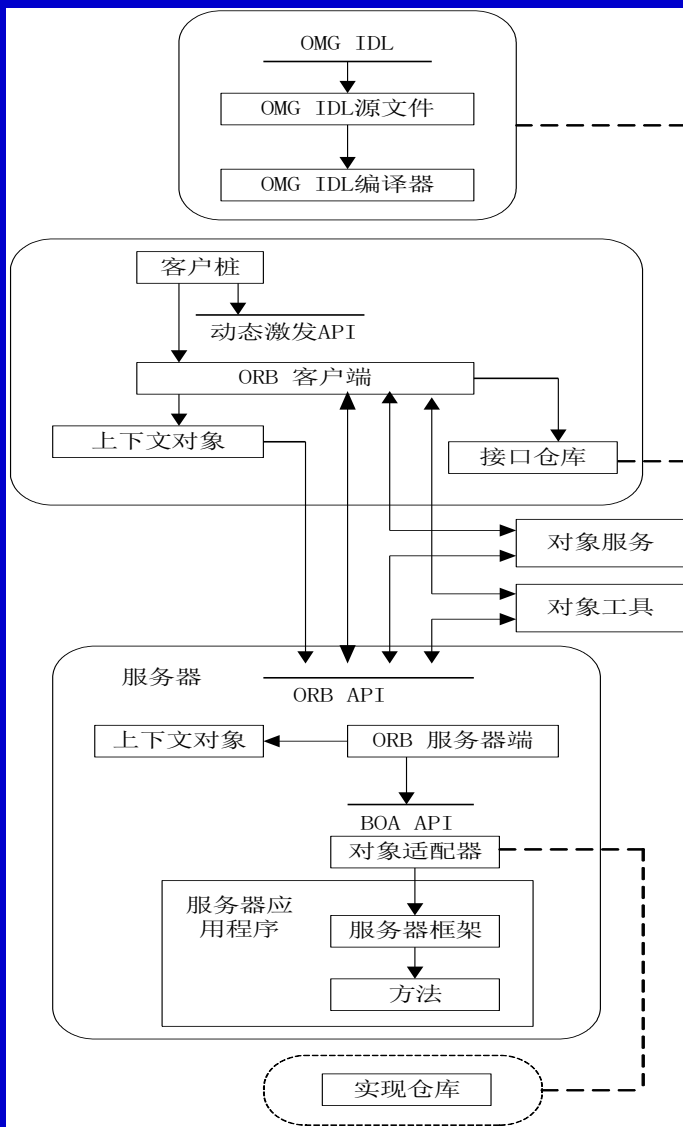


#### ◇ CORBA技术规范

##### ◎ 动态调用接口

在CORBA中，对象适配器用于屏蔽ORB内核的实现细节，为服务器对象的实现者提供抽象接口，以便他们使用ORB内部的某些功能。这些功能包括服务器对象的登录与激活、客户请求的认证等。

### ◇ 体系结构



### ◇ 特点

- ◎ 引入中间件作为事务代理，完成客户机向服务对象方（Server）提出的业务请求。
- ◎ 实现客户与服务对象的完全分开，客户不需要了解服务对象的实现过程以及具体位置。
- ◎ 提供软总线机制，使得在任何环境下、采用任何语言开发的软件只要符合接口规范的定义，均能够集成到分布式系统中。
- ◎ CORBA规范软件系统采用面向对象的软件实现方法开发应用系统，实现对象内部细节的完整封装，保留对象方法的对外接口定义。



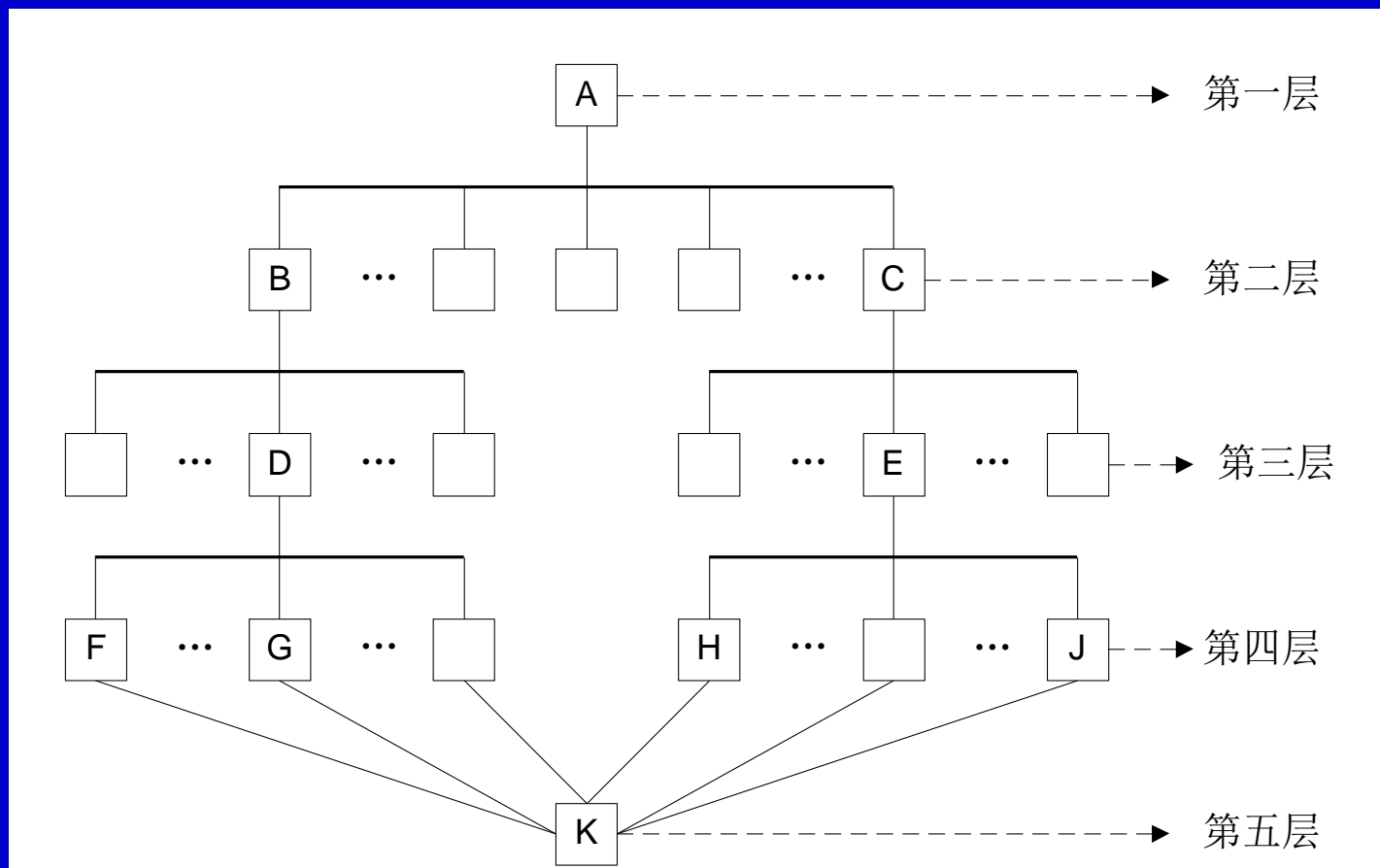
### ◇ 概念

线索：纵向；组织层：横向

正交软件体系结构由组织层和线索的构件构成。层是由一组具有相同抽象级别的构件构成。线索是子系统的特例，它是由完成不同层次功能的构件组成（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，在同一层中的构件之间是不存在相互调用的。

如果线索是相互独立的，即不同线索中的构件之间没有相互调用，那么这个结构就是完全正交的。

### ◇ 框架



### ◇ 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）。

### ◇ 实例

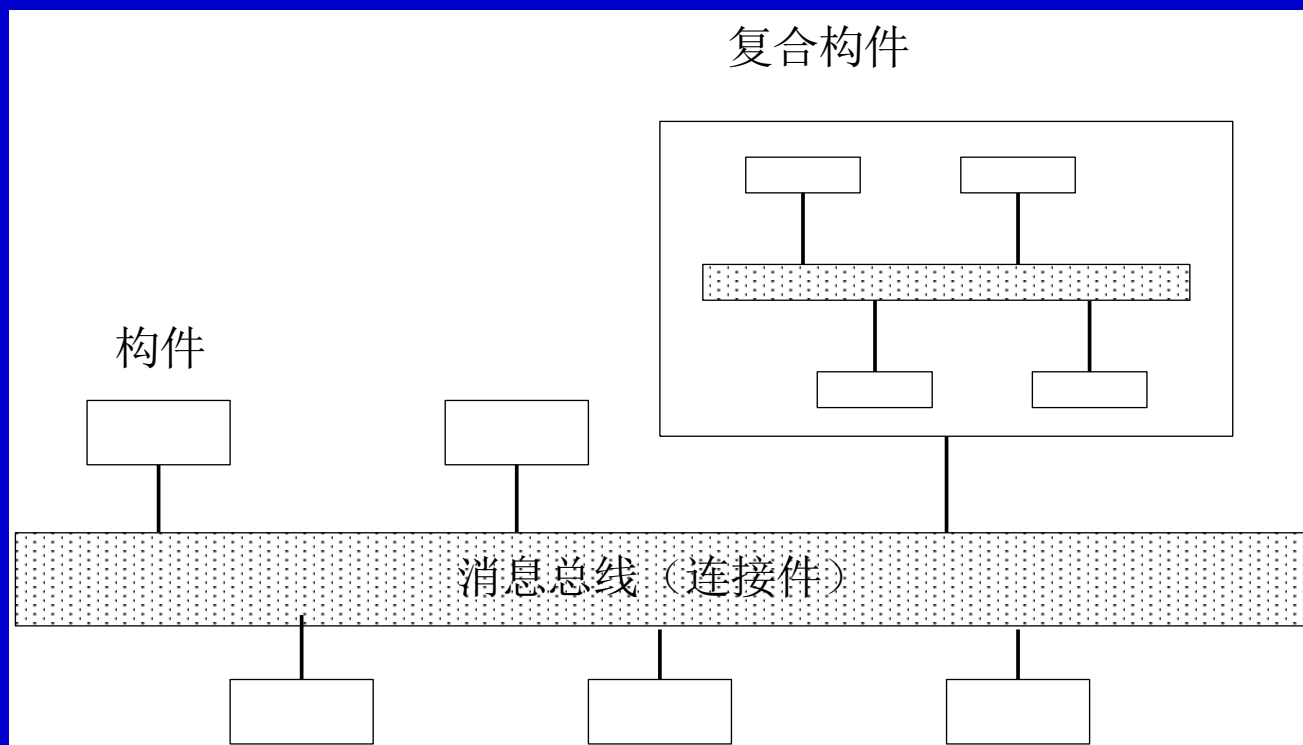
自学

### ◇ 优点

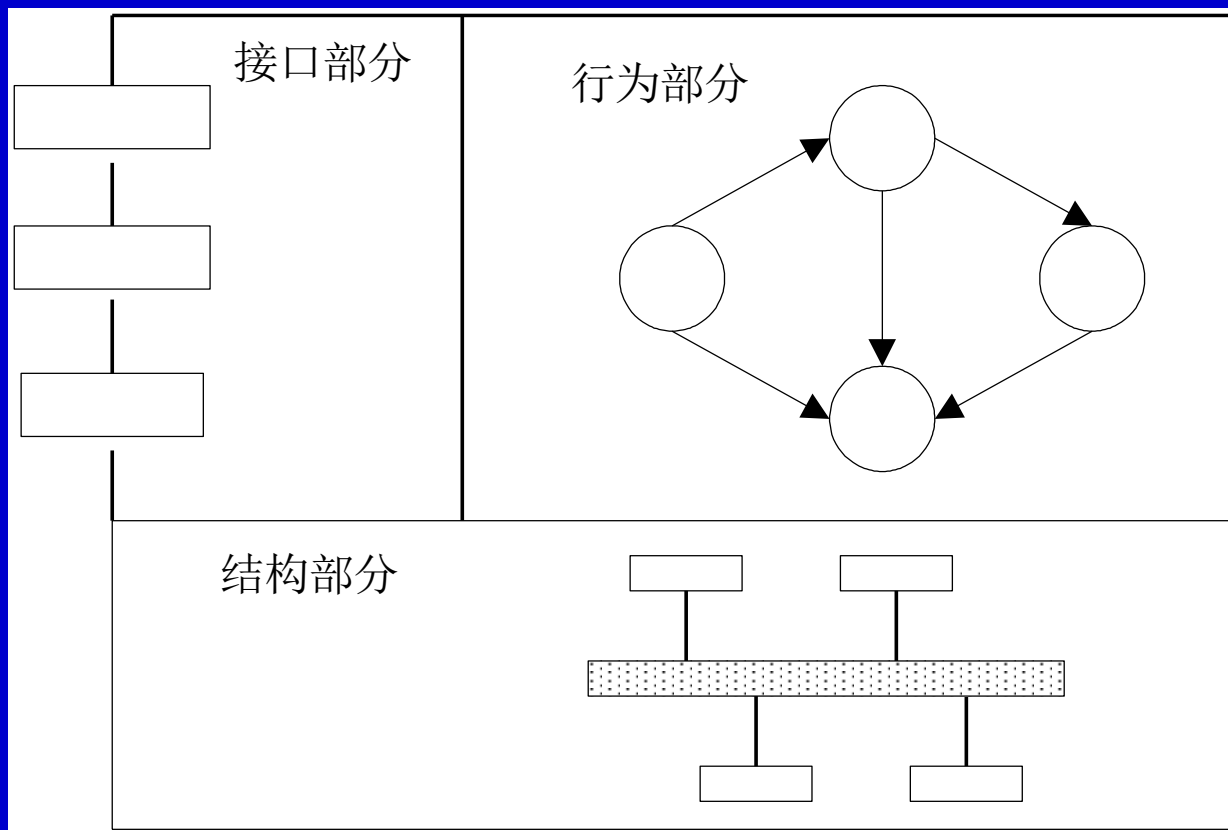
- ◎ 结构清晰，易于理解。由于线索功能相互独立，不进行互相调用，结构简单、清晰，构件在结构图中的位置已经说明它所实现的是哪一级抽象，担负的是什么功能。
- ◎ 易修改，可维护性强。由于线索之间是相互独立的，所以对一个线索的修改不会影响到其他线索。系统功能的增加或减少，只需相应的增删线索构件族，而不影响整个正交体系结构，因此能方便地实现结构调整。
- ◎ 可移植性强，重用粒度大。因为正交结构可以为一个领域内的所有应用程序所共享，这些软件有着相同或类似的层次和线索，可以实现体系结构级的重用。



### ◇ 概述



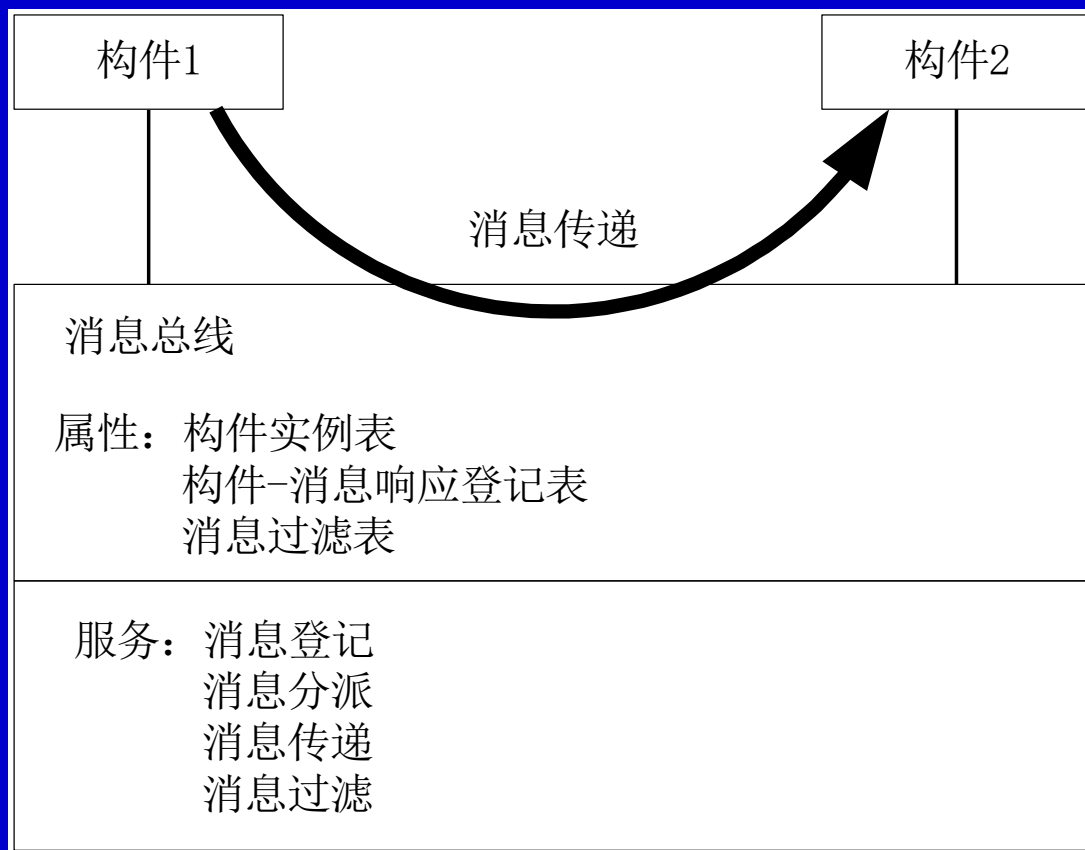
### ◇ HMB风格的构件模型



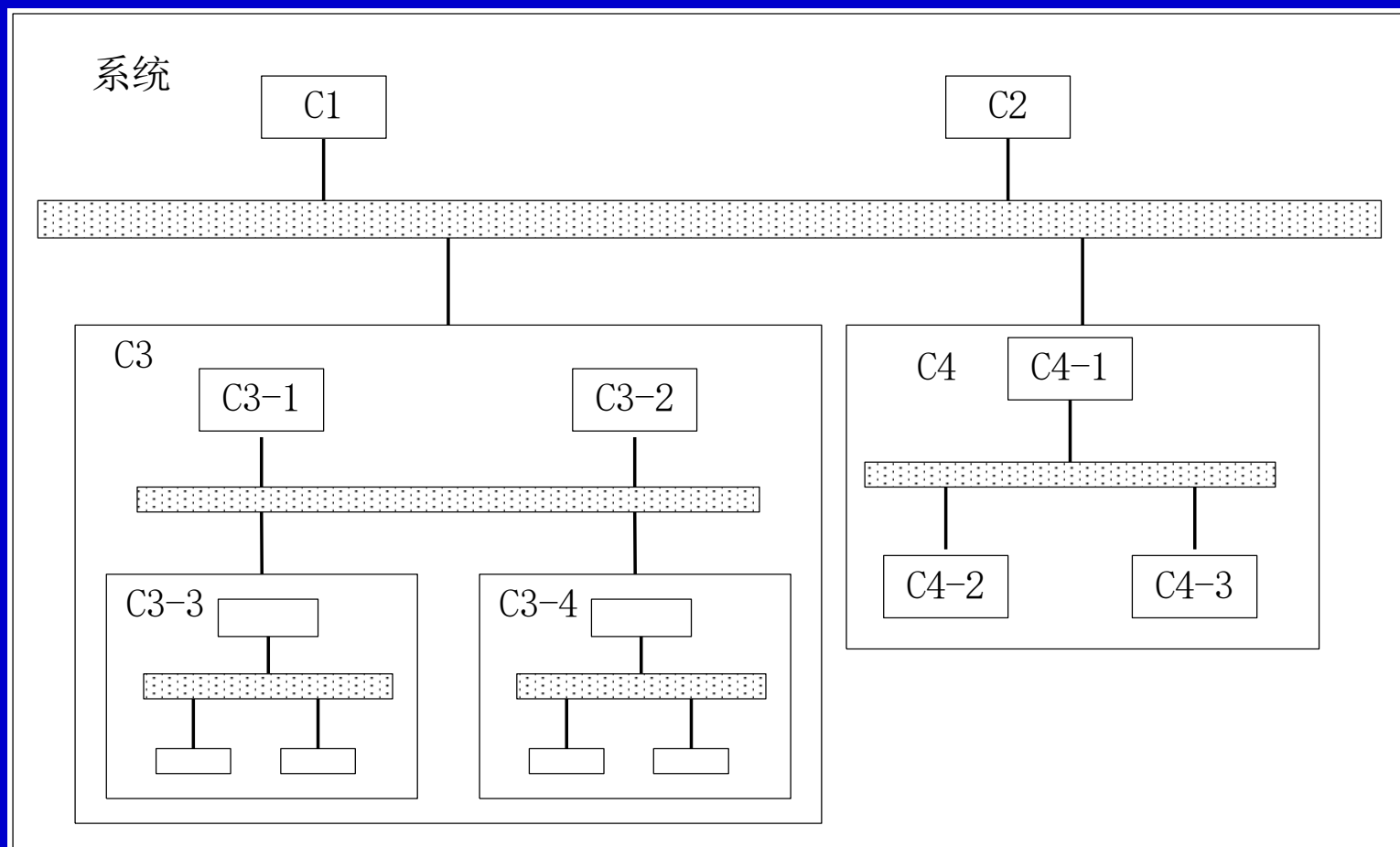
#### ◇ 构件接口

- ◎ HMB风格的构件接口是一种基于消息的互联接口，可以较好地支持体系结构设计。构件之间通过消息进行通讯，接口定义了构件发出和接收的消息集合。
- ◎ 当某个事件发生后，系统或构件发出相应的消息，消息总线负责把该消息传递到此消息感兴趣的构件。
- ◎ 按照响应方式的不同，消息可分为同步消息和异步消息。

### ◇ 消息总线



### ◇ 构件静态结构



#### ◇ 构件动态行为

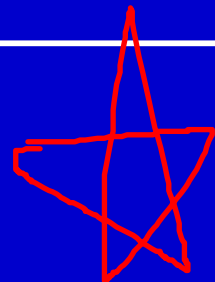
◎ 构件的行为就由外来消息的类型唯一确定，即一个消息和构件的某个操作之间存在着固定的对应关系。对于这类构件，可以认为构件只有一个状态，或者在每次对消息响应之前，构件处于初始状态。

◎ 更通常的情况是，构件的行为同时受外来消息类型和自身当前所处状态的影响。

#### ◇ 运行时刻的系统演化

- ◎ 动态增加或删除构件
- ◎ 动态改变构件响应的消息类型
- ◎ 消息过滤

### ◇ 为什么要使用异构结构



可能考大  
题

◎ 不同的结构有不同的处理能力的强项和弱点，一个系统的体系结构应该根据实际需要进行选择，以解决实际问题。

◎ 关于软件包、框架、通信以及其他一些体系结构上的问题，目前存在多种标准。即使在某段时间内某一种标准占统治地位，但变动最终是绝对的。

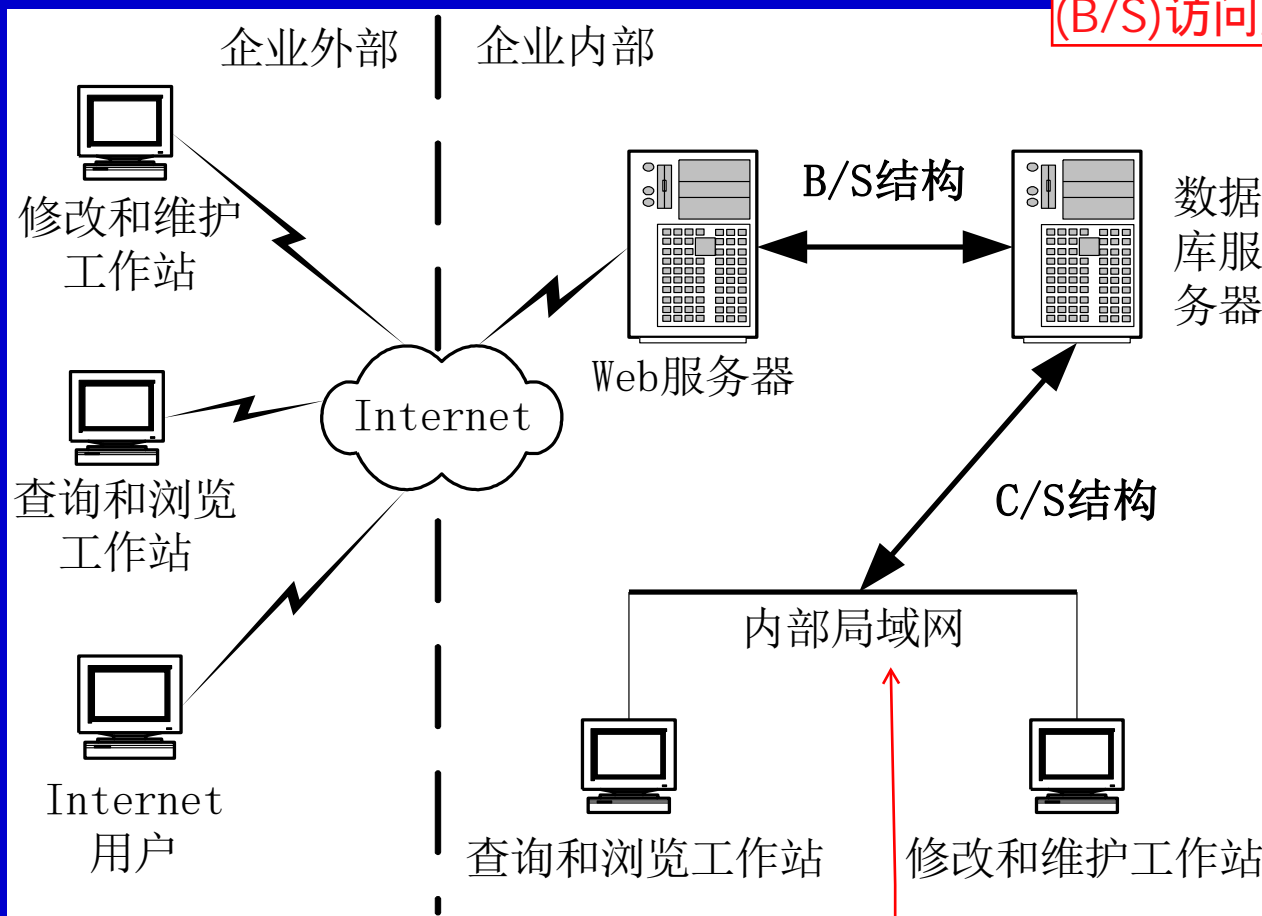
◎ 实际工作中，我们总会遇到一些遗留下来的代码，它们仍有效用，但是却与新系统有某种程度上的不协调。然而在许多场合，将技术与经济综合进行考虑时，总是决定不再重写它们。

◎ 即使在某一单位中，规定了共享共同的软件包或相互关系的一些标准，仍会存在解释或表示习惯上的不同。



### ◇ C/S与B/S混合之**内外有别模型**

内外有别：内部的局域网通过客户端(C/S)访问，外部通过浏览器(B/S)访问服务器



外部用户无法直接访问内部数据库服务器

内部用户交互性强

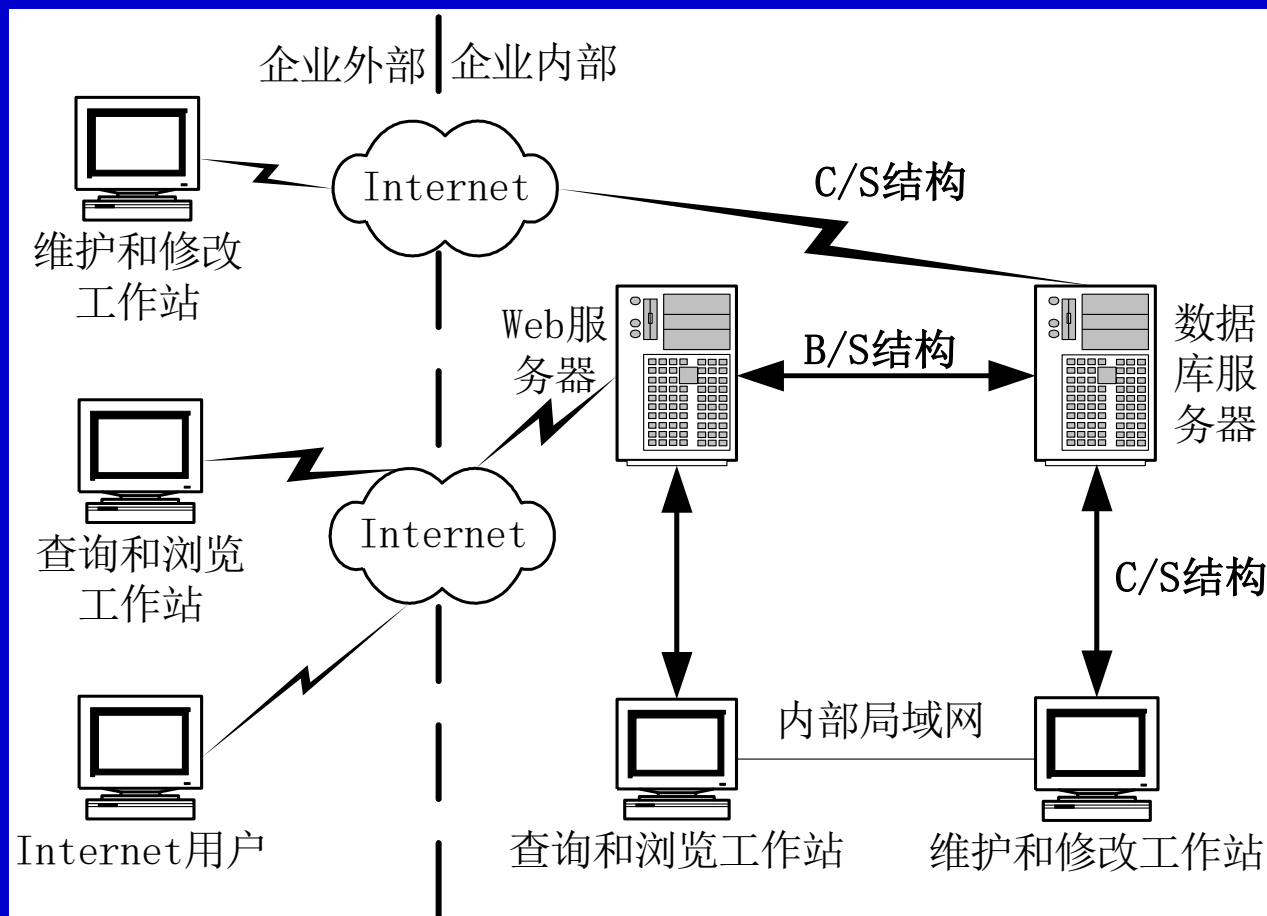
画图  
explain

# 第3章 软件体系结构风格

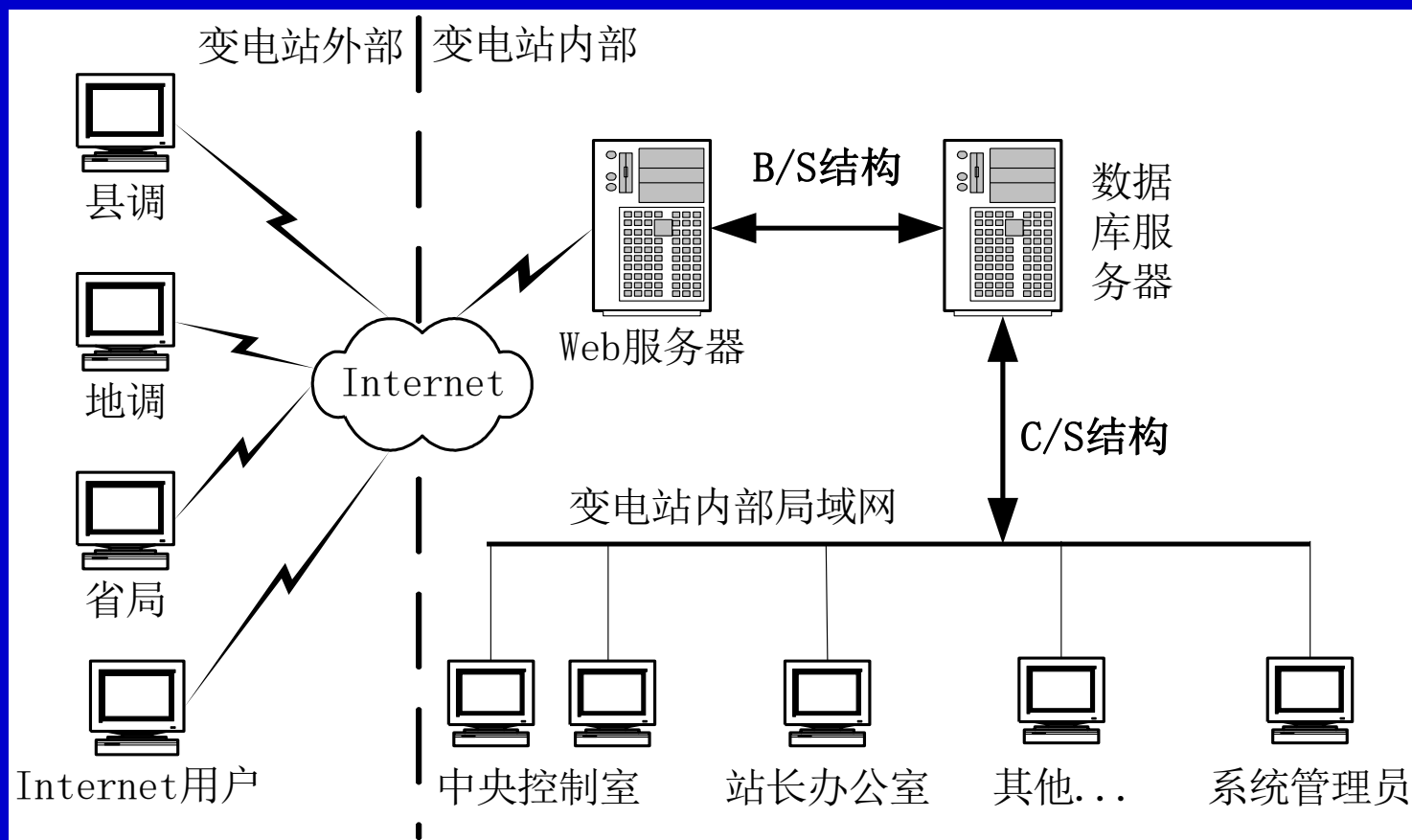
## 3.9 异构结构风格

### ◇ C/S与B/S混合之查改有别模型

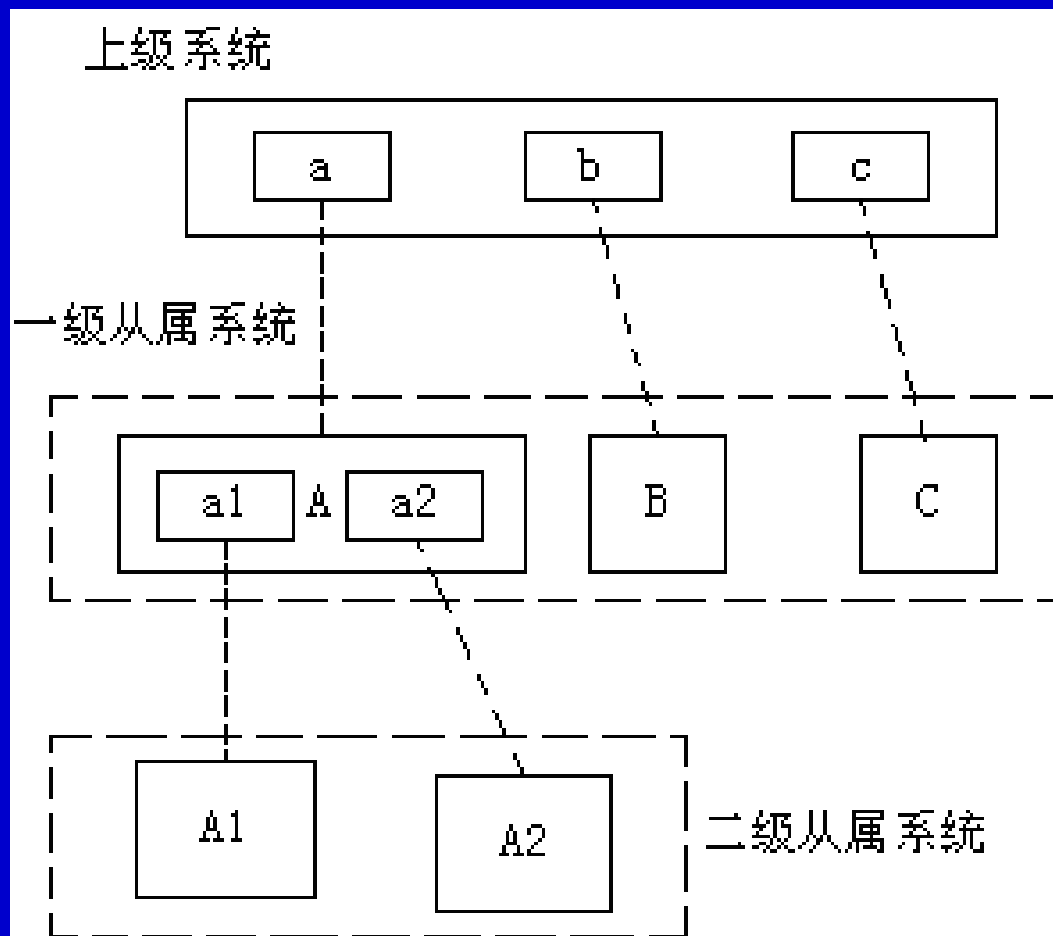
内外要维护和修改都是通过C/S，要查询和浏览都是通过B/S



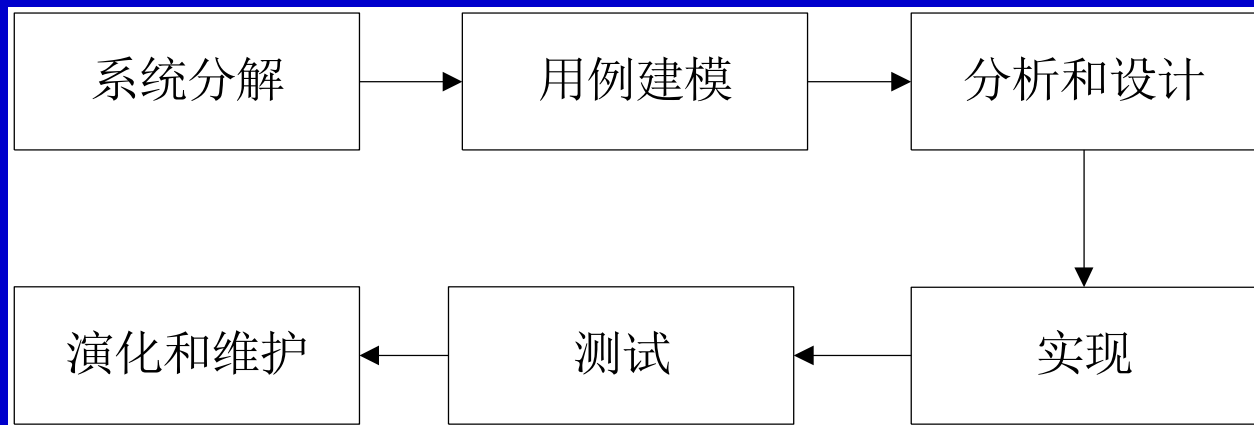
### ◇ 异构实例



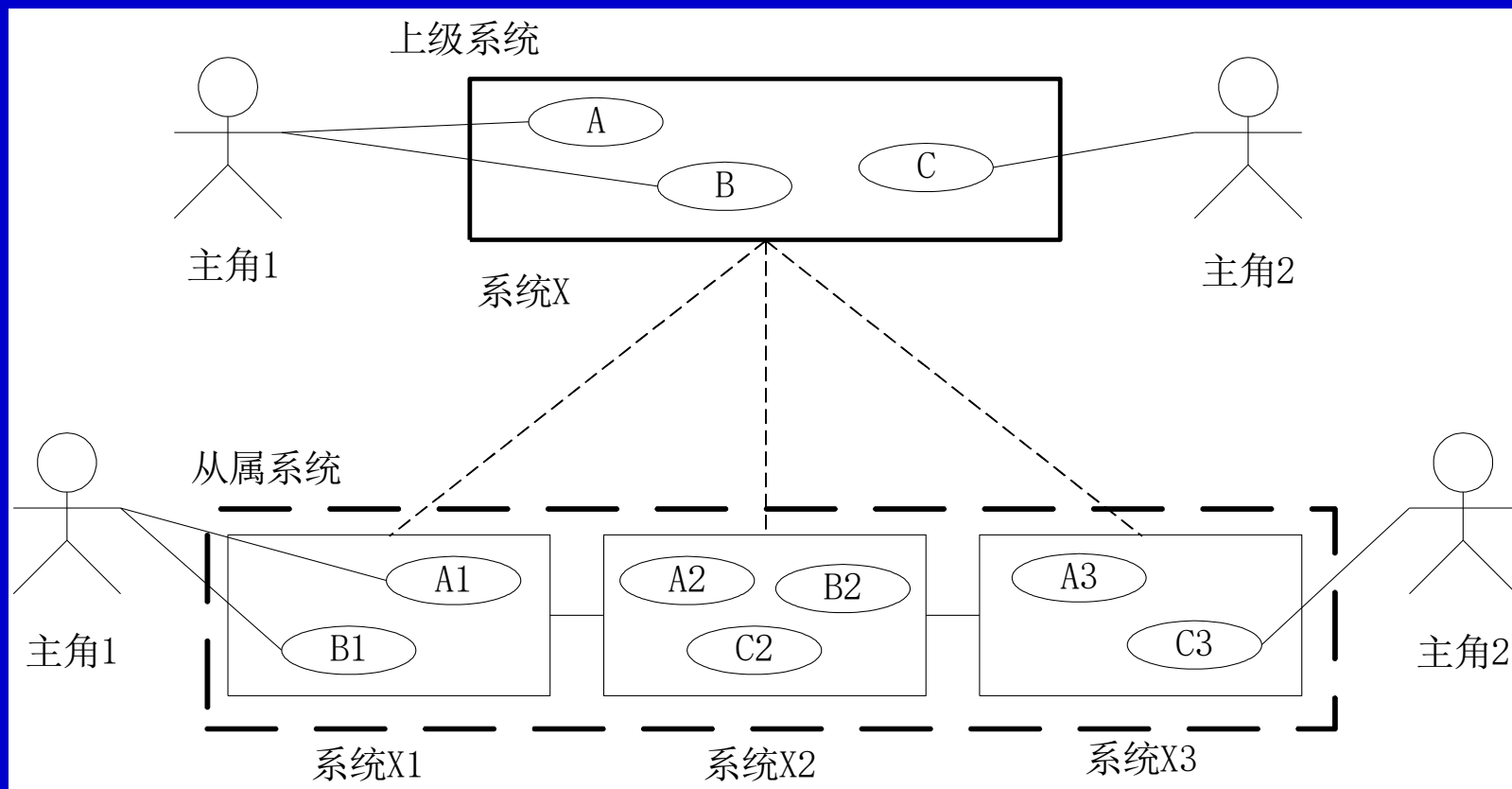
### ◇ 互连系统构成的系统



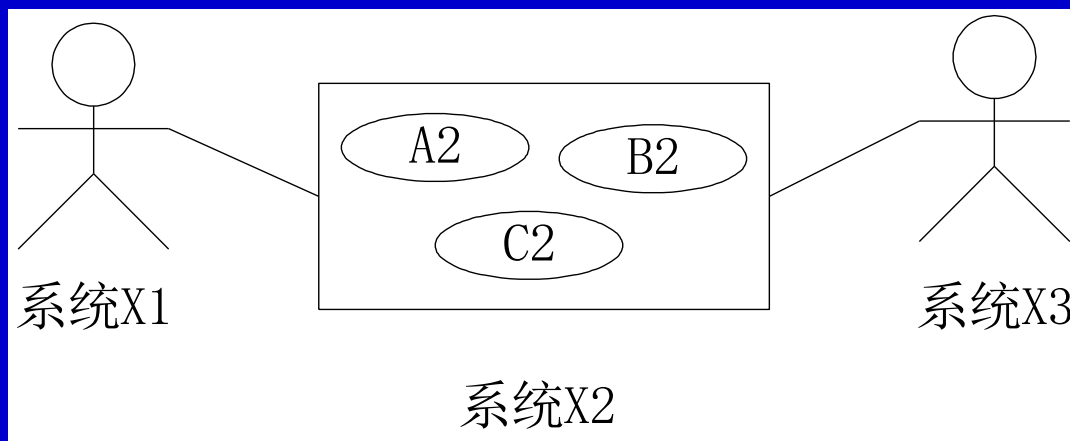
### ◇ 基于SASIS的软件过程



### ◇ 基于SASIS的软件过程



### ◇ 基于SASIS的软件过程



#### ◇ 定义

◎ Hayes-Roth对DSSA的定义如下：“DSSA就是专用于一类特定类型的任务（领域）的、在整个领域中能有效地使用的、为成功构造应用系统限定了标准的组合结构的软件构件的集合”。

◎ Tracz的定义为：“DSSA就是一个特定的问题领域中支持一组应用的领域模型、参考需求、参考体系结构等组成的开发基础，其目标就是支持在一个特定领域中多个应用的生成”。



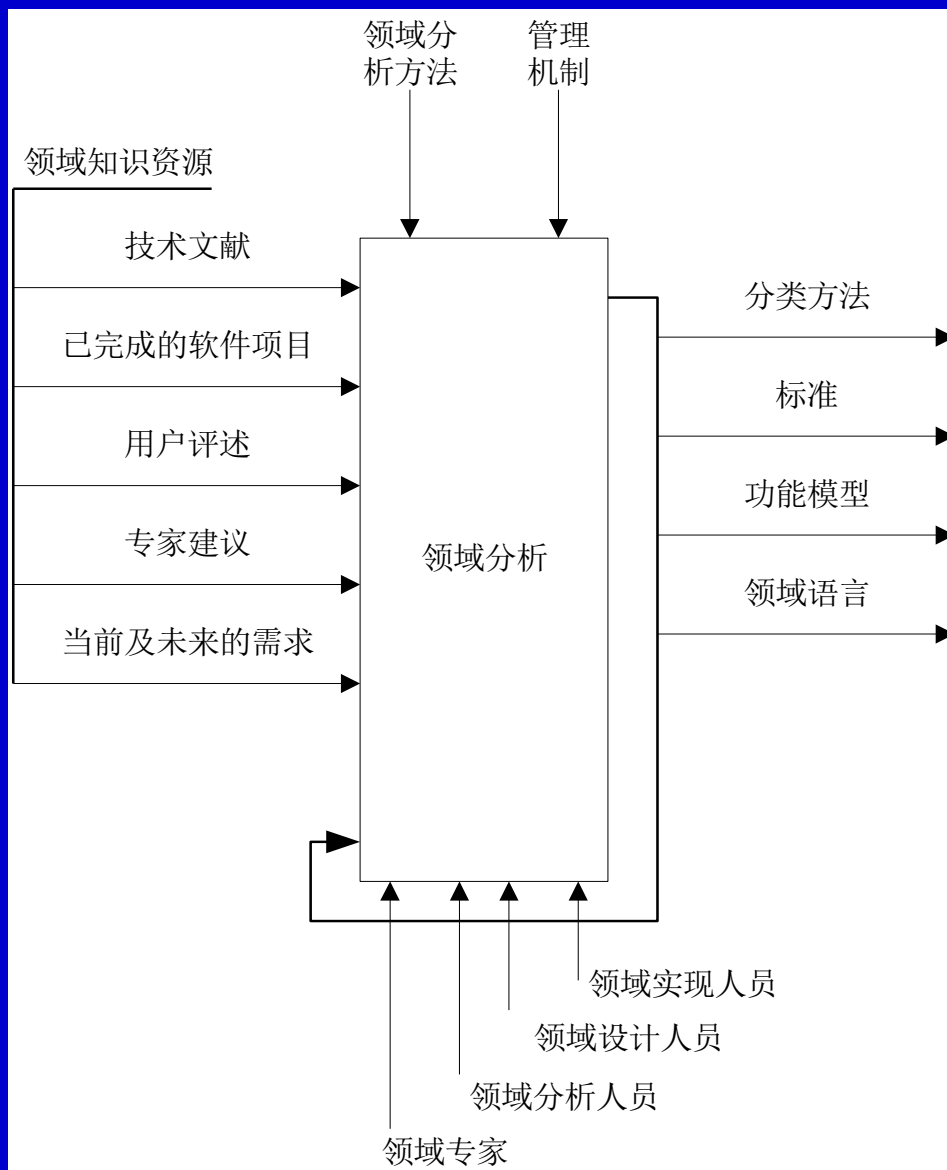
### ◇ 定义

- ◎ **垂直域**：定义了一个特定的系统族，包含整个系统族内的多个系统，结果是在该领域中可作为系统的可行解决方案的一个通用软件体系结构。
- ◎ **水平域**：定义了在多个系统和多个系统族中功能区域的共有部分，在子系统级上涵盖多个系统族的特定部分功能，无法为系统提供完整的通用体系结构。

### ◇ 基本活动



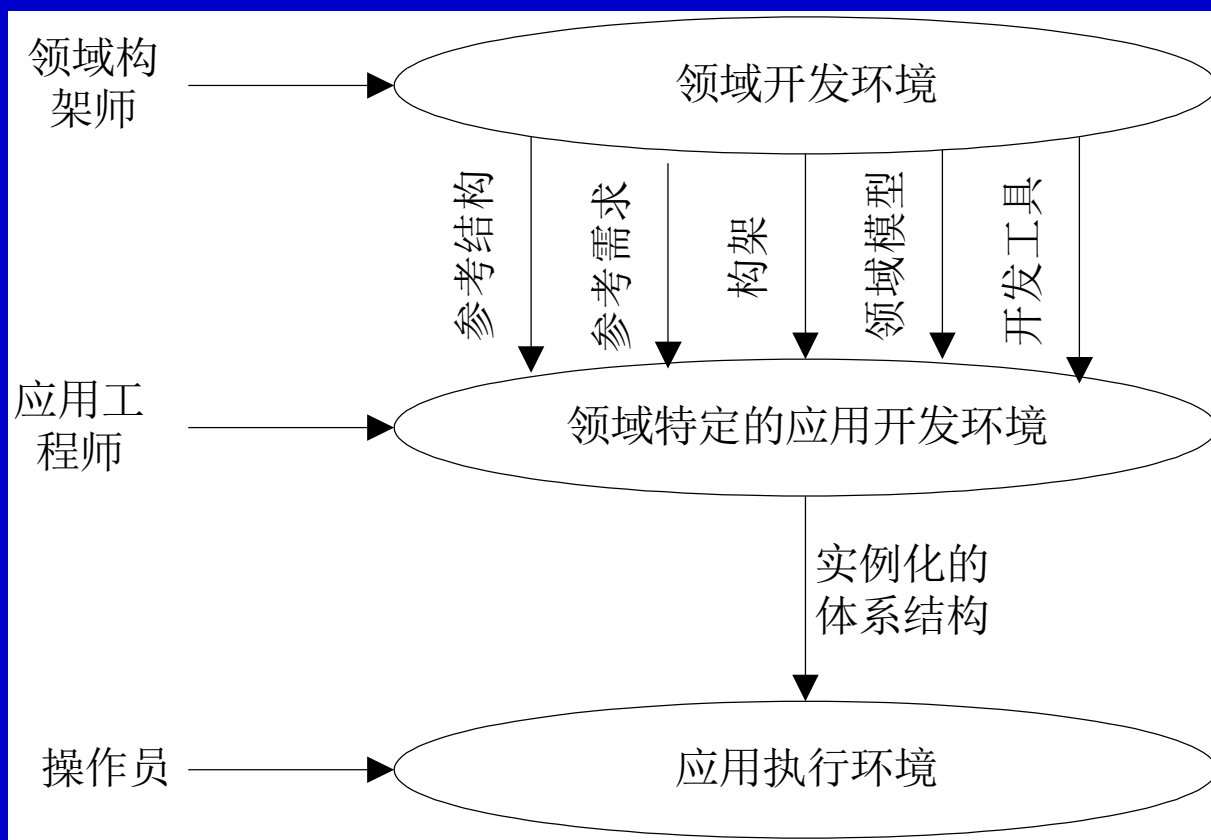
### ◇ 领域分析



### ◇ 建立过程

- ◎ 定义领域范围：确定什么在感兴趣的领域中以及本过程到何时结束。
- ◎ 定义领域特定的元素：编译领域字典和领域术语的同义词词典。识别领域中应用间的共同性和差异性；
- ◎ 定义领域特定的设计和实现需求约束：描述解空间中有差别的特性。不仅要识别出约束，并且要记录约束对设计和实现决定造成的后果，还要记录对处理这些问题时产生的所有问题的讨论；
- ◎ 定义领域模型和体系结构：产生一般的体系结构，并说明构成它们的模块或构件的语法和语义；
- ◎ 产生、搜集可重用的产品单元：为DSSA增加构件使得它可以被用来产生问题域中的新应用。

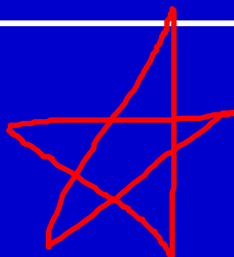
### ◇ 三层次系统模型



### ◇ 实例

自学

### ◇ DSSA和体系结构风格的比较



- ◎ DSSA以问题域为出发点，体系结构风格以解决域为出发点。
- ◎ DSSA只对某一个领域进行设计专家知识的提取、存储和组织，但可以同时使用多种体系结构风格；而在某个体系结构风格中进行体系结构设计专家知识的组织时，可以将提取的公共结构和设计方法扩展到多个应用领域。
- ◎ DSSA通常选用一个或多个适合所研究领域的体系结构风格，并设计一个该领域专用的体系结构分析设计工具。
- ◎ 体系结构风格的定义和该风格应用的领域是直交的，提取的设计知识比用DSSA提取的设计专家知识的应用范围要广。
- ◎ DSSA和体系结构风格是互为补充的两种技术。

- 1、层次系统结构和基于消息的层次系统结构有什么区别？
- 2、试分析和比较B/S，二层C/S和三层C/S，指出各自的优点和缺点。
- 3、组织或参与一个采用B/S和C/S混合体系结构的软件项目的开发，总结开发经验。
- 4、组织或参与一个采用三层体系结构的软件项目的开发，总结开发经验。
- 5、SIS和DSSA分别用在哪些场合？
- 6、在软件开发中，采用异构结构有什么好处，其负面影响有哪些？



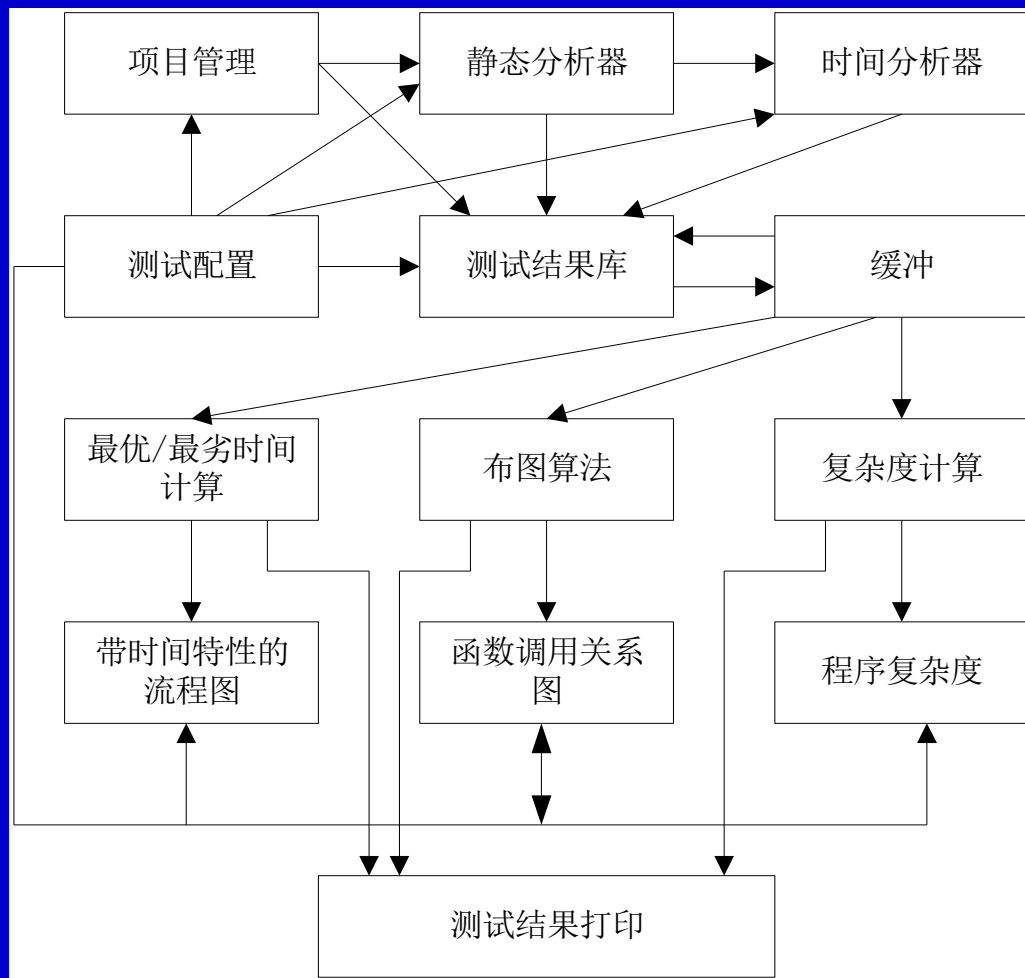
# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

### ◇ 描述方法的种类

- ◎ 图形表达工具
- ◎ 模块内连接语言
- ◎ 基于软构件的系统描述语言
- ◎ 软件体系结构描述语言

### ◇ 图形表达工具



### ◇ 模块内连接语言

◎ 采用将一种或几种传统程序设计语言的模块连接起来的模块内连接语言。由于程序设计语言和模块内连接语言具有严格的语义基础，因此它们能支持对较大的软件单元进行描述，诸如定义/使用和扇入/扇出等操作。例如，Ada语言采用use实现包的重用，Pascal语言采用过程（函数）模块的交互等。

◎ MIL方式对模块化的程序设计和分段编译等程序设计与开发技术确实发挥了很大的作用。但是由于这些语言处理和描述的软件设计开发层次过于依赖程序设计语言，因此限制了它们处理和描述比程序设计语言元素更为抽象的高层次软件体系结构元素的能力。

### ◇ 基于软构件的系统描述语言

◎ 基于软构件的系统描述语言将软件系统描述成一种是由许多以特定形式相互作用的特殊软件实体构造组成的组织或系统。

◎ 例如，一种多变配置语言就可以用来在一个较高的抽象层次上对系统的体系结构建模，Darwin最初用作设计和构造复杂分布式系统的配置说明语言，因具有动态特性，也可用来描述动态体系结构。

◎ 这种表达和描述方式虽然也是较好的一种以构件为单位的软件系统描述方法，但是他们所面向和针对的系统元素仍然是一些层次较低的以程序设计为基础的通信协作软件实体单元，而且这些语言所描述和表达的系统一般而言都是面向特定应用的特殊系统，这些特性使得基于软构件的系统描述仍然不是十分适合软件体系结构的描述和表达。

### ◇ 软件体系结构描述语言

◎ 软件体系结构的第四种描述和表达方法是参照传统程序设计语言的设计和开发经验，重新设计、开发和使用针对软件体系结构特点的专门的软件体系结构描述语言。

◎ 由于ADL是在吸收了传统程序设计中的语义严格精确的特点基础上，针对软件体系结构的整体性和抽象性特点，定义和确定适合于软件体系结构表达与描述的有关抽象元素，因此，ADL是当前软件开发和设计方法学中一种发展很快的软件体系结构描述方法，目前，已经有几十种常见的ADL。


### ◇ IEEE P1471

- ◎ IEEE P1471于2000年9月21日通过IEEE-SA标准委员会评审。
- ◎ IEEE P1471适用于软件密集的系统，其目标在于：便于体系结构的表达与交流，并通过体系结构要素及其实践标准化，奠定质量与成本的基础。
- ◎ IEEE P1471详细介绍了一套体系结构描述的概念框架，并给出建立框架的思路。但如何描述以及具体的描述技术等方面缺乏更进一步的指导。

### ◇ Rational

- ◎ Rational起草了可重用的软件资产规格说明，专门讨论了体系结构描述的规格说明，提出了一套易于重用的体系结构描述规范。该建议草案已经提交OMG。
- ◎ 基于RUP（Rational United Process）、采用UML模型描述软件的体系结构，认为体系结构描述的关键是定义视点、视图以及建模元素之间的映射关系。
- ◎ 与IEEE P1471相比，该建议标准的体系结构描述方案涉及面比较窄，所注重的层次比较低，因而更具体。由于将体系结构的描述限于UML和RUP，具有一定的局限性，但该建议标准结合了业界已经广泛采用的建模语言和开发过程，因而易于推广，可以有效实现在跨组织之间重用体系结构描述结果。





ADL是在底层语义模型的支持下，为软件系统的概念体系结构建模提供了具体语法和概念框架。基于底层语义的工具为体系结构的表示、分析、演化、细化、设计过程等提供支持。其三个基本元素是：构件、连接件、体系结构配置。

主要的体系结构描述语言有Aesop、MetaH、C2、Rapide、SADL、Unicon和Wright等，尽管它们都描述软件体系结构，却有不同的特点。

这些ADL强调了体系结构不同的侧面，对体系结构的研究和应用起到了重要的作用，但也有负面的影响。每一种ADL都以独立的形式存在，描述语法不同且互不兼容，同时又有许多共同的特征，这使设计人员很难选择一种合适的ADL，若设计特定领域的软件体系结构又需要从头开始描述。

### ◇ ADL与其他语言的比较(1)

- ◎ 构造能力：ADL能够使用较小的独立体系结构元素来建造大型软件系统；
- ◎ 抽象能力：ADL使得软件体系结构中的构件和连接件描述可以只关注它们的抽象特性，而不管其具体的实现细节；
- ◎ 重用能力：ADL使得组成软件系统的构件、连接件甚至是软件体系结构都成为软件系统开发和设计的可重用部件；

#### ◇ ADL与其他语言的比较(2)

- ◎ 组合能力：ADL使得其描述的每一系统元素都有其自己的局部结构，这种描述局部结构的特点使得ADL支持软件系统的动态变化组合；
- ◎ 异构能力：ADL允许多个不同的体系结构描述关联存在；
- ◎ 分析和推理能力：ADL允许对其描述的体系结构进行多种不同的性能和功能上的多种推理分析。

### ◇ 典型元素含义比较

程序设计语言		软件体系结构	
程序构件	组成程序的基本元素及其取值或值域范围	系统构件	模块化级别的系统组成成分实体，这些实体可以被施以抽象的特性化处理，并以多种方式得到使用
操作符	连接构件的各种功能符号	连接件	对组成系统的有关抽象实体进行各种连接的连接机制
抽象规则	有关构件和操作符的命名表达规则	组合模式	系统中的构件和连接件进行连接组合的特殊方式，也就是软件体系结构的风格
限制规则	一组选择并决定具体使用何种抽象规则来作用于有关的基本构件及其操作符的规则和原理	限制规则	决定有关模式能够作为子系统进行大型软件系统构造和开发的合法子系统的有关条件
规范说明	有关句法的语义关联说明	规范说明	有关系统组织结构方面的语义关联说明

### ◇ 常见的软件体系结构元素

系统构件元素		连接件元素	
纯计算单元	这类构件只有简单的输入/输出处理关联,对它们的处理一般也不保留处理状态,如数学函数、过滤器和转换器等	过程调用	在构件实体之间实现单线程控制的连接机制,如普通过程调用和远程过程调用等
数据存储单元	具有永久存储特性的结构化数据,如数据库、文件系统、符号表和超文本等	数据流	系统中通过数据流进行交互的独立处理流程连接机制,其最显著的特点是根据得到的数据来进行构件实体的交互控制,如 Unix 操作系统中的管道机制等
管理器	对系统中的有关状态和紧密相关操作进行规定与限制的实体,如抽象数据类型和系统服务器等	隐含触发器	由并发出现的事件来实现构件实体之间交互的连接机制,在这种连接机制中,构件实体之间不存在明显确定的交互规定,如时间调度协议和自动垃圾回收处理等
控制器	控制和管理系统中有关事件发生的时间序列,如调度程序和同步处理协调程序等	消息传递	独立构件实体之间通过离散和非在线的数据(可以是同步或非同步的)进行交互的连接机制,如 TCP/IP 等
连接器	充当有关实体间信息转换角色的实体,如通讯连接器和用户界面等	数据共享协议	构件之间通过相同的数据空间进行并发协调操作的机制,如黑板系统中的黑板和多用户数据库系统中的共享数据区等

### ◇ C2概述(1)

- ◎ C2和其提供的设计环境（Argo）支持采用基于时间的风格来描述用户界面系统，并支持使用可替换、可重用的构件开发GUI的体系结构。
- ◎ 在C2中，连接件负责构件之间消息的传递，而构件维持状态、执行操作并通过两个名字分别为“*top*”和“*bottom*”的端口和其它的构件交换信息。
- ◎ 每个接口包含一种可发送的消息和一组可接收的消息。构件之间的消息要么是请求其它构件执行某个操作的请求消息，要么是通知其他构件自身执行了某个操作或状态发生改变的通知消息。

#### ◇ C2概述(2)

- ◎ 构件之间的消息交换不能直接进行，而只能通过连接件来完成。每个构件接口最多只能和一个连接件相连，而连接件可以和任意数目的构件或连接件相连。
- ◎ 请求消息只能向上层传送而通知消息只能向下层传送。
- ◎ 通知消息的传递只对应于构件内部的操作，而和接收消息的构件的需求无关。
- ◎ C2对构件和连接件的实现语言、实现构件的线程控制、构件的部署以及连接件使用的通讯协议等都不加限制。

#### ◇ C2对构件接口的描述

```
Component ::=  
  component component_name is  
    interface component_message_interface  
    parameters component_parameters  
    methods component_methods  
    [behavior component_behavior]  
    [context component_context]  
  end component_name;
```



### ◇ C2对构件的描述

```
component_message_interface ::=
    top_domain_interface
    bottom_domain_interface
```

```
top_domain_interface ::=
    top_domain is
        out interface_requests
        in interface_notifications
```

```
bottom_domain_interface ::=
    bottom_domain is
        out interface_notifications
        in interface_requests
```

```
interface_requests ::=
    {request; } | null;
```

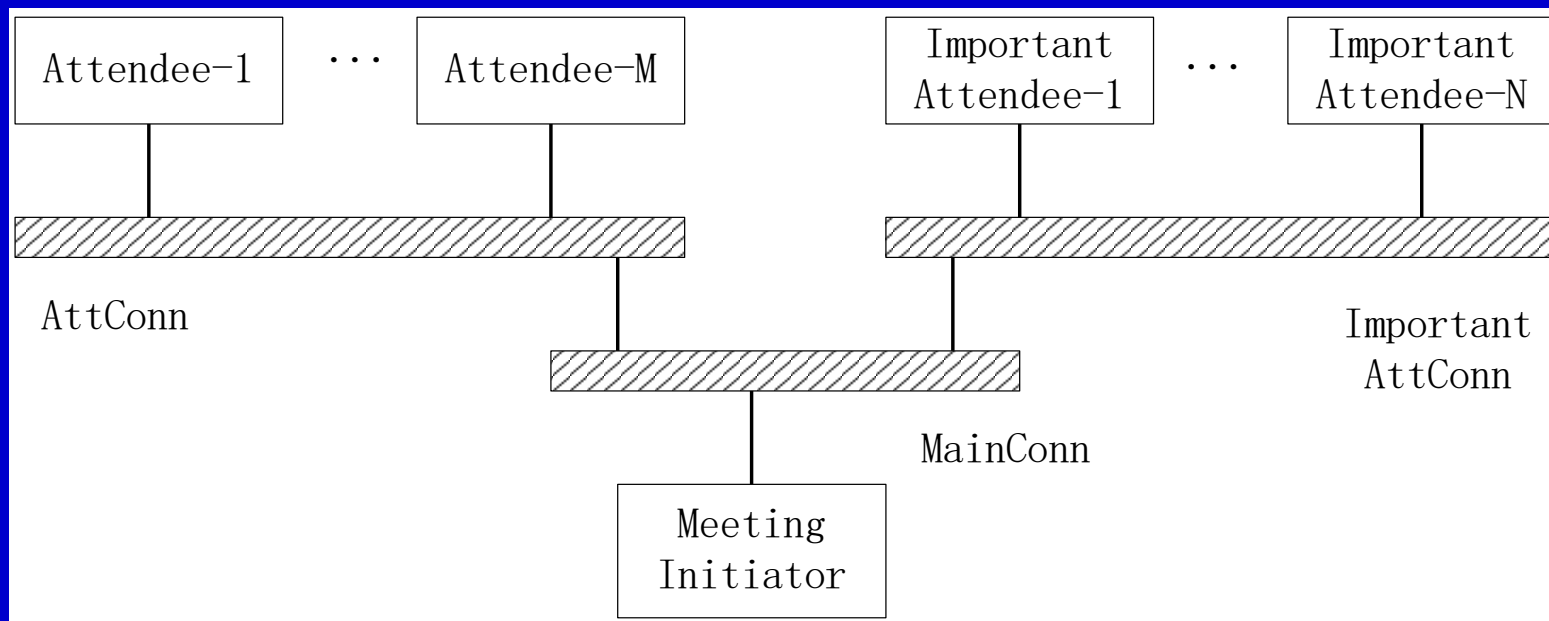
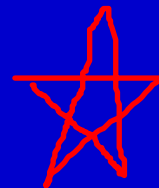
```
interface_notifications ::=
    {notification; } | null;
```

```
request ::=
    message_name(request_parameters)
```

```
request_parameters ::=
    [to component_name][parameter_list]
```

```
notification ::=
    message_name[parameter_list]
```

### ◇ 会议安排系统的C2风格



#### ◇ C2对*MeetingInitiator*构件的描述(1)

```
component MeetingInitiator is
  interface
    top_domain is
      out
        GetPrefSet();
        GetExclSet();
        GetEquipReqs();
        GetLocPrefs();
        RemoveExclSet();
        RequestWithdrawal(to Attendee);
        RequestWithdrawal(to ImportantAttendee);
        AddPrefDates();
        MarkMtg(d:date; l:lov_type);
```

#### ◇ C2对*MeetingInitiator*构件的描述(2)

```
in
    PrefSet(p:date_mg);
    ExclSet(e:data_mg);
    EquipReqs(eq:equip_type);
    LocPref(l:loc_type);
behavior
    startup always_generate GetPrefSet,  GetExclSet,  GetEquipReqs,
    GetLocPrefs;
    received_messages PrefSet may_generate RemoveExclSet xor
RequestWithdrawal xor MarkMtg;
    received_messages    ExclSet    may_generate    AddPrefDates    xor
RemoveExclSet
xor RequestWithdrawal xor MarkMtg;
    received_messages EquipReqs may_generate AddPrefDates xor
RemoveExclSet xor RequestWithdrawal xor MarkMtg;
    received_messages LocPref always_generate null;
end MeetingInitiator;
```

### ◇ C2对*Attendee*构件的描述(1)

```
component Attendee is
  interface
    bottom_domain is
      out
        PrefSet(p:date_mg);
        ExclSet(e:date_mg);
        EquipReqs(eq:equip_type);
      in
        GetPrefSet();
        GetExclSet();
        GetEquipReqs();
        RemoveExclSet();
        RequestWithdrawal();
        AddPrefDates();
        MarkMtg(d:date; l:loc_type);
```

#### ◇ C2对*Attendee*构件的描述(2)

```
behavior
received_messages GetPrefSet always_generate PrefSet;
received_messages AddPrefDates always_generate PrefSet;
received_messages GetExclSet always_generate ExclSet;
received_messages GetEquipReqs always_generate EquipReqs;
received_messages RemoveExclSet always_generate ExclSet;
received_messages ReuestWithdrawal always_generate null;
received_messages MarkMtg always_generate null;
end Attendee;
```

#### ◇ C2对 *ImportantAttendee*构件的描述

```
component ImportantAttendee is subtype Attendee(in and beh)
interface
  bottom_domain is
    out
      LocPrefs(l:loc_type);
      ExclSet(e:date_mg);
      EquipReqs(eq:equip_type);
    in
      GetLocPrefs();
  behavior
received_messages GetLocPrefs always_generate LocPrefs;
end ImportantAttendee;
```

### ◇ C2对体系结构的描述

```
architecture MeetingScheduler is  
  conceptual_components
```

```
    Attendee; ImportantAttendee; MeetingInitiator;
```

```
  connectors
```

```
    connector MainConn is message_filter no_filtering;
```

```
      connector AttConn is message_filter no_filtering;
```

```
      connector ImportantAttConn is message_filter no_filtering;
```

```
  architectural_topology
```

```
    connector AttConn connections
```

```
      top_ports Attendee;
```

```
      bottom_ports MainConn;
```

```
    connector ImportantAttConn connections
```

```
      top_ports ImportantAttendee;
```

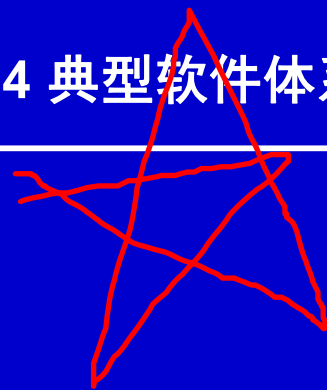
```
      bottom_ports MainConn;
```

```
    connector MainConn connections
```

```
      top_ports AttConn; ImportantAttConn;
```

```
      bottom_ports MeetingInitiator;
```

```
end MeetingScheduler;
```





#### ◇ C2对会议安排系统的描述

```
system MeetingScheduler_1 is
architecture MeetingScheduler with
    Attendee instance Att_1, Att_2, Att_3;
    ImportantAttendee instance ImpAtt_1, ImpAtt_2;
    MeetingInitiator instance MtgInit_1;
end MeetingScheduler_1;
```

### ◇ UML简介

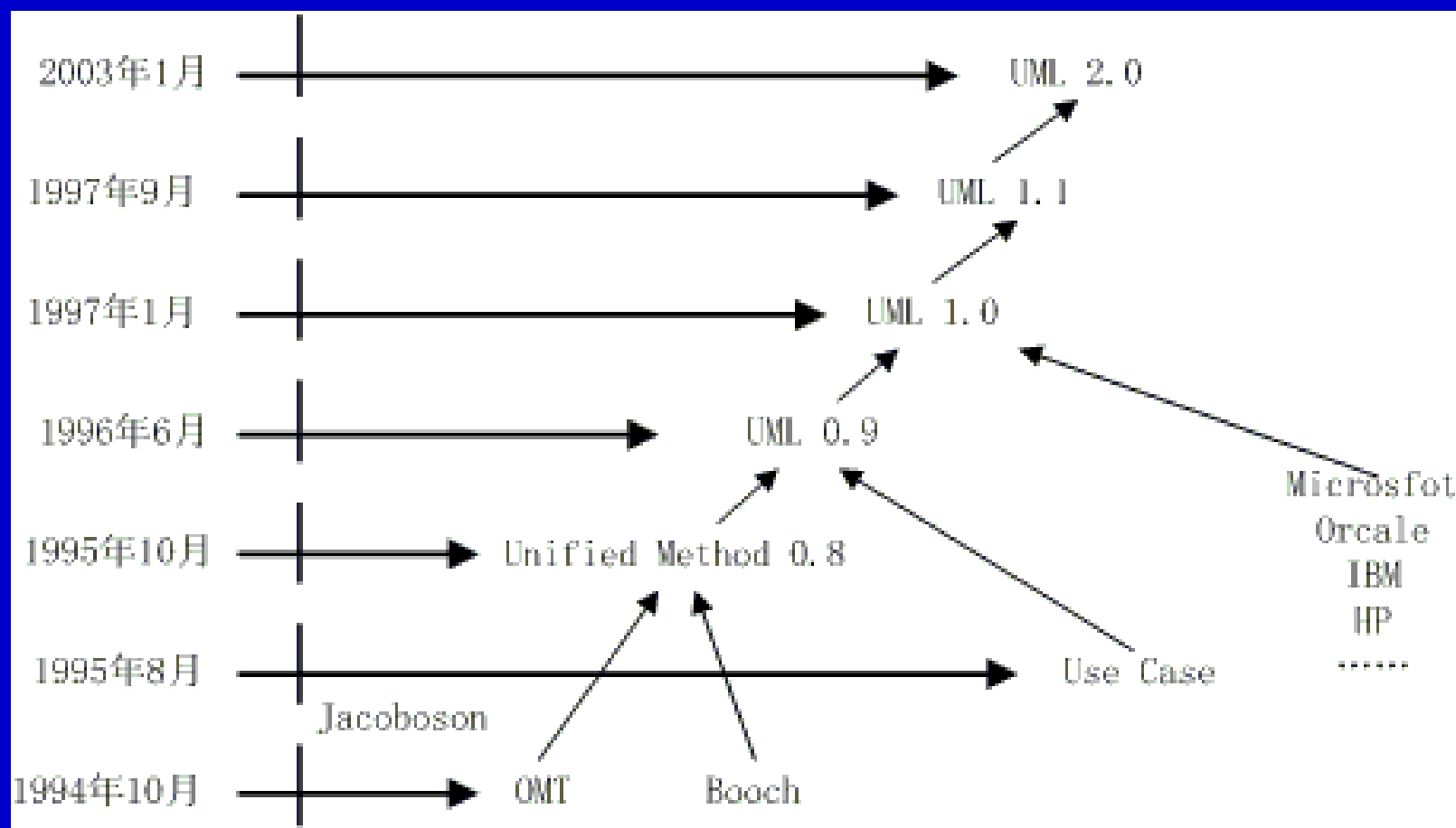
◎ UML (Unified Modeling Language) 是下面这些最好的建模方法中最好部分的集成：

- ◇ 商务流程模型 (Work Flow)
- ◇ 对象建模方法
- ◇ 软构件建模思想

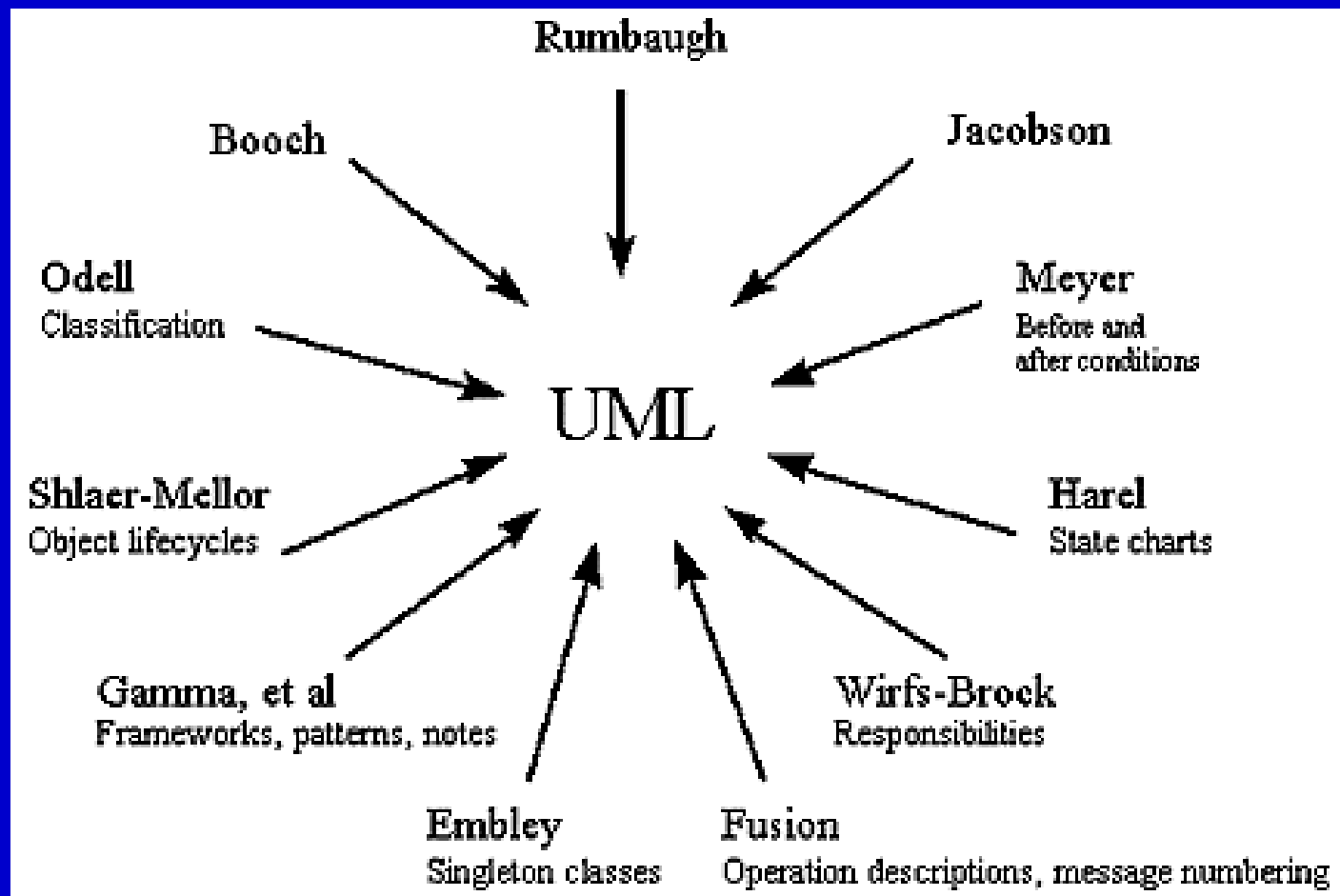
◎ UML是一种用可视化方法对软件系统进行描述、实施和说明的标准语言。

◎ 支持用不同实现技术进行的软件开发全过程。

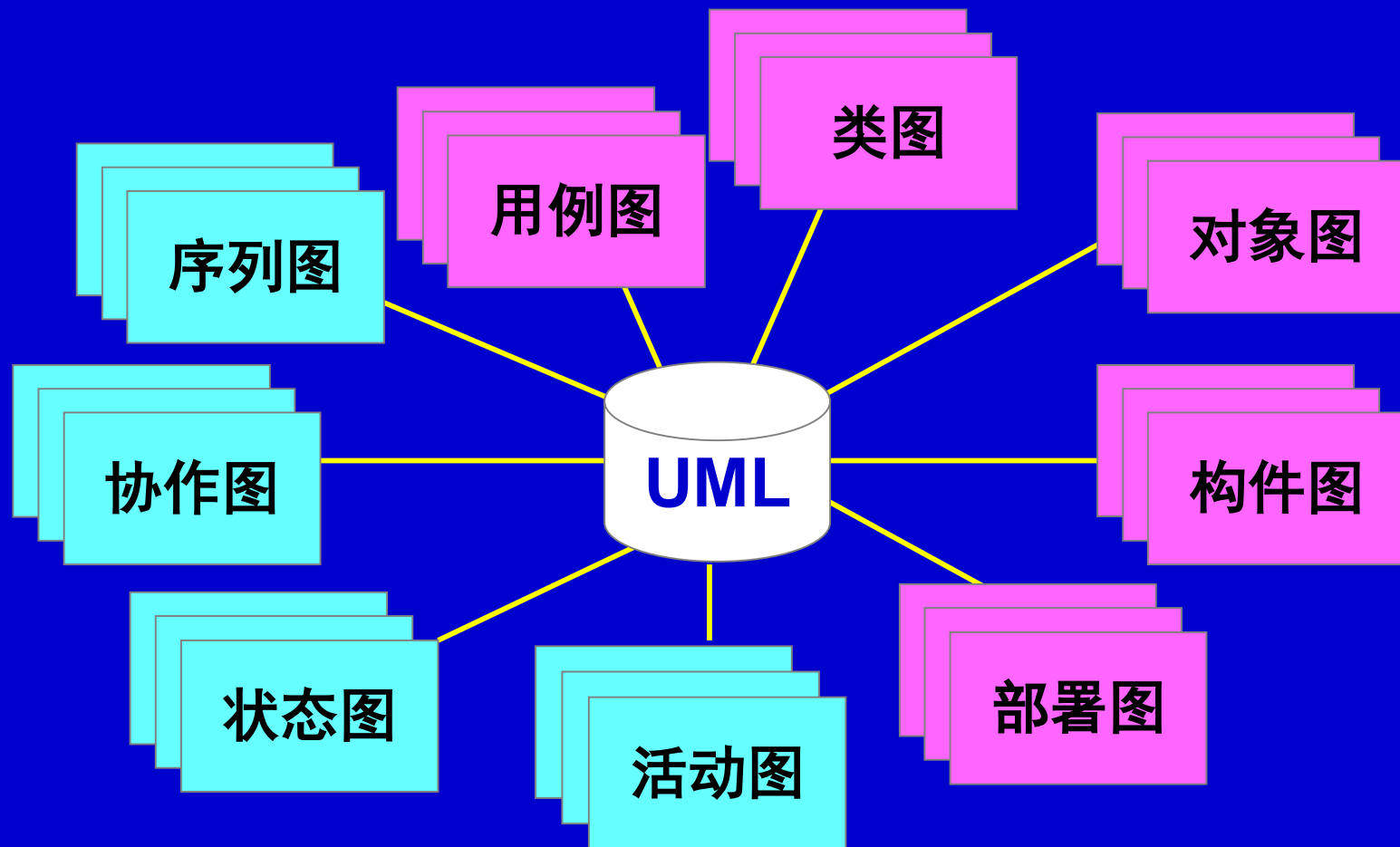
### ◇ UML简介



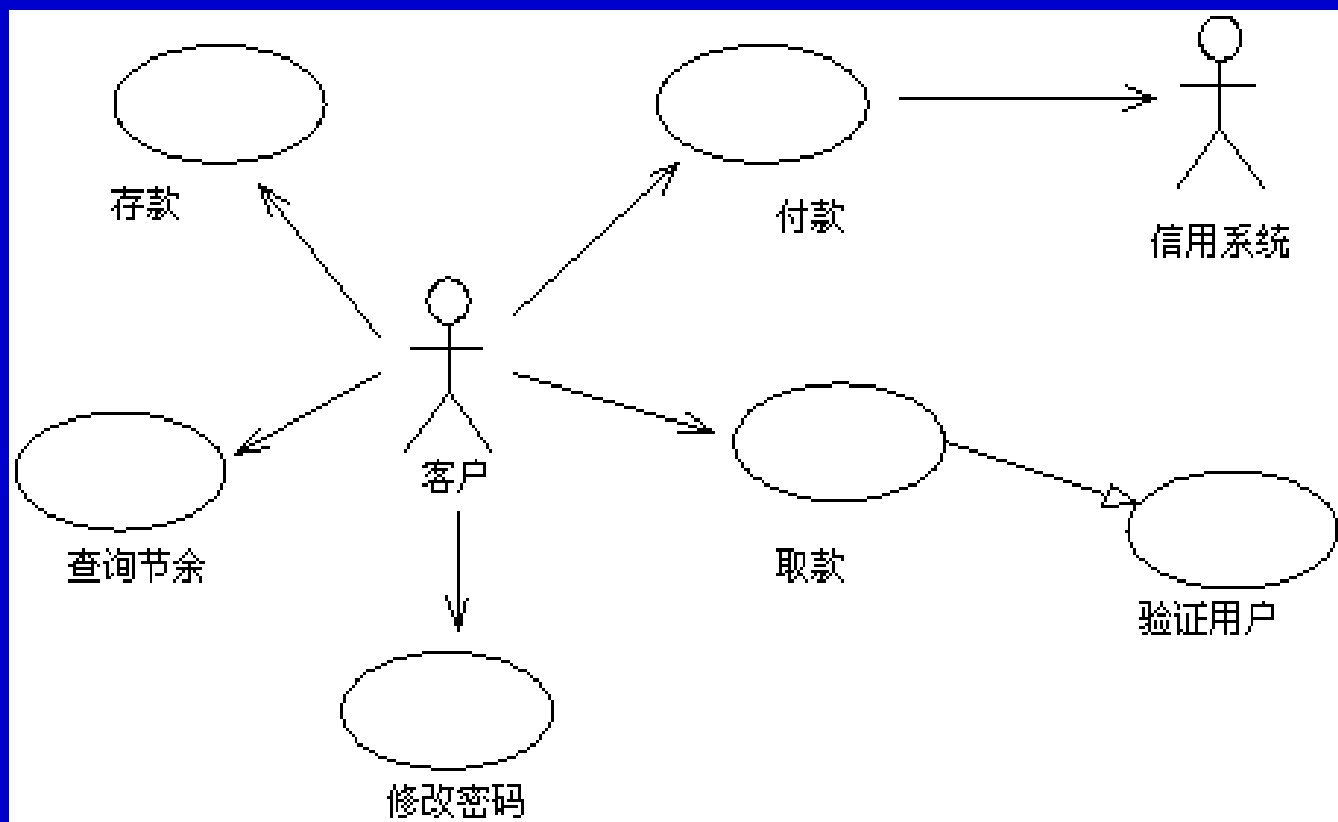
### ◇ UML简介



### ◇ UML简介

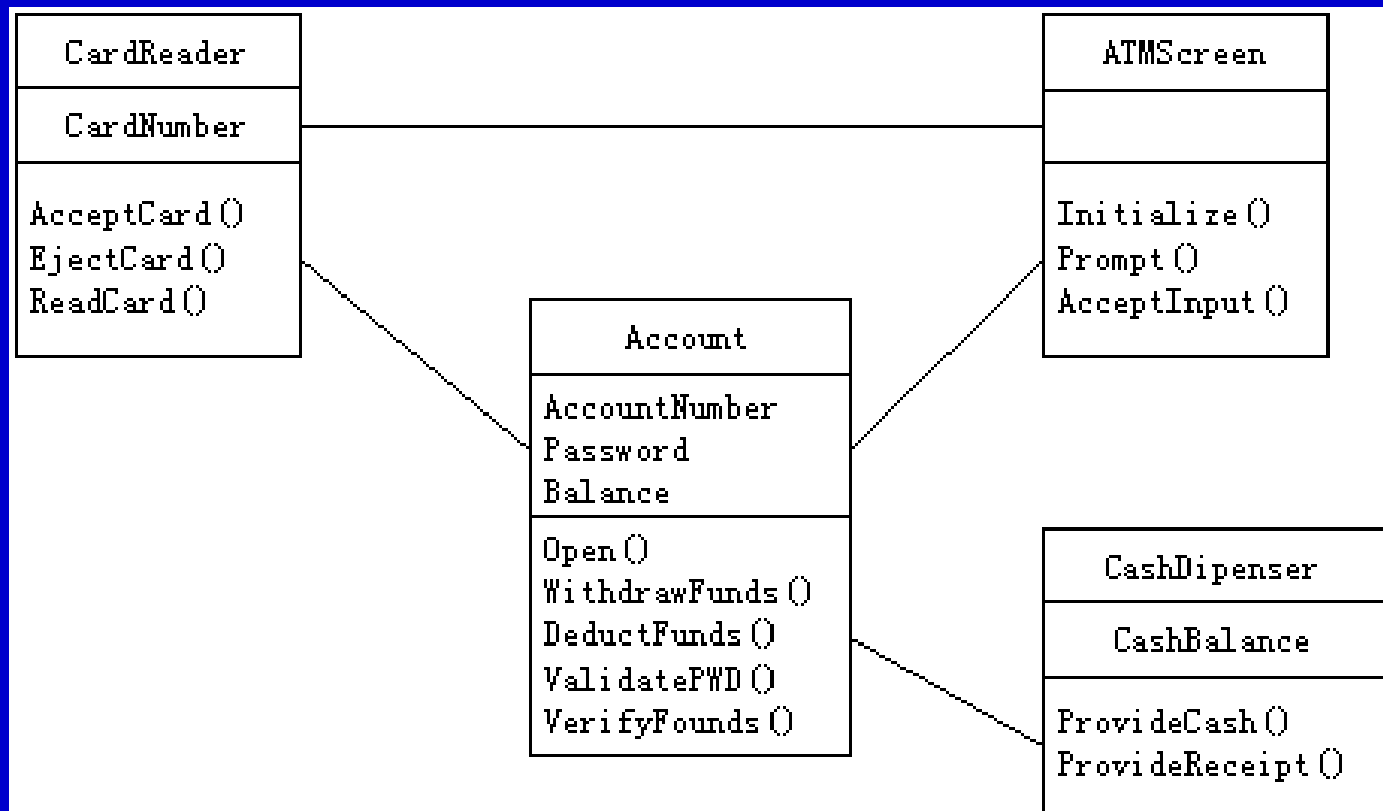


### ◇ 用例图



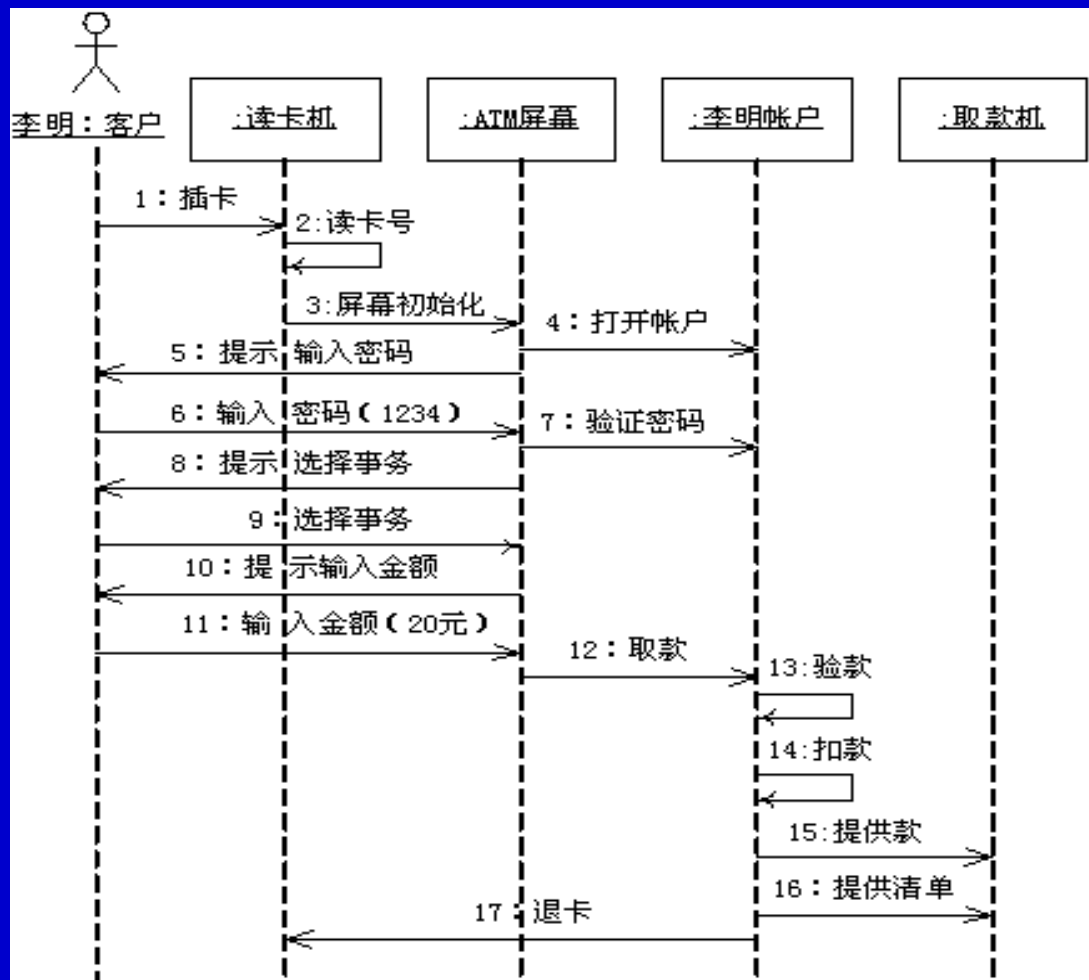
用于显示若干角色以及这些角色与系统提供的用例之间的连接关系。用例是系统提供的功能的描述

### ◇ 类图



表示系统中的类和类之间的关系，它是对系统静态结构的描述

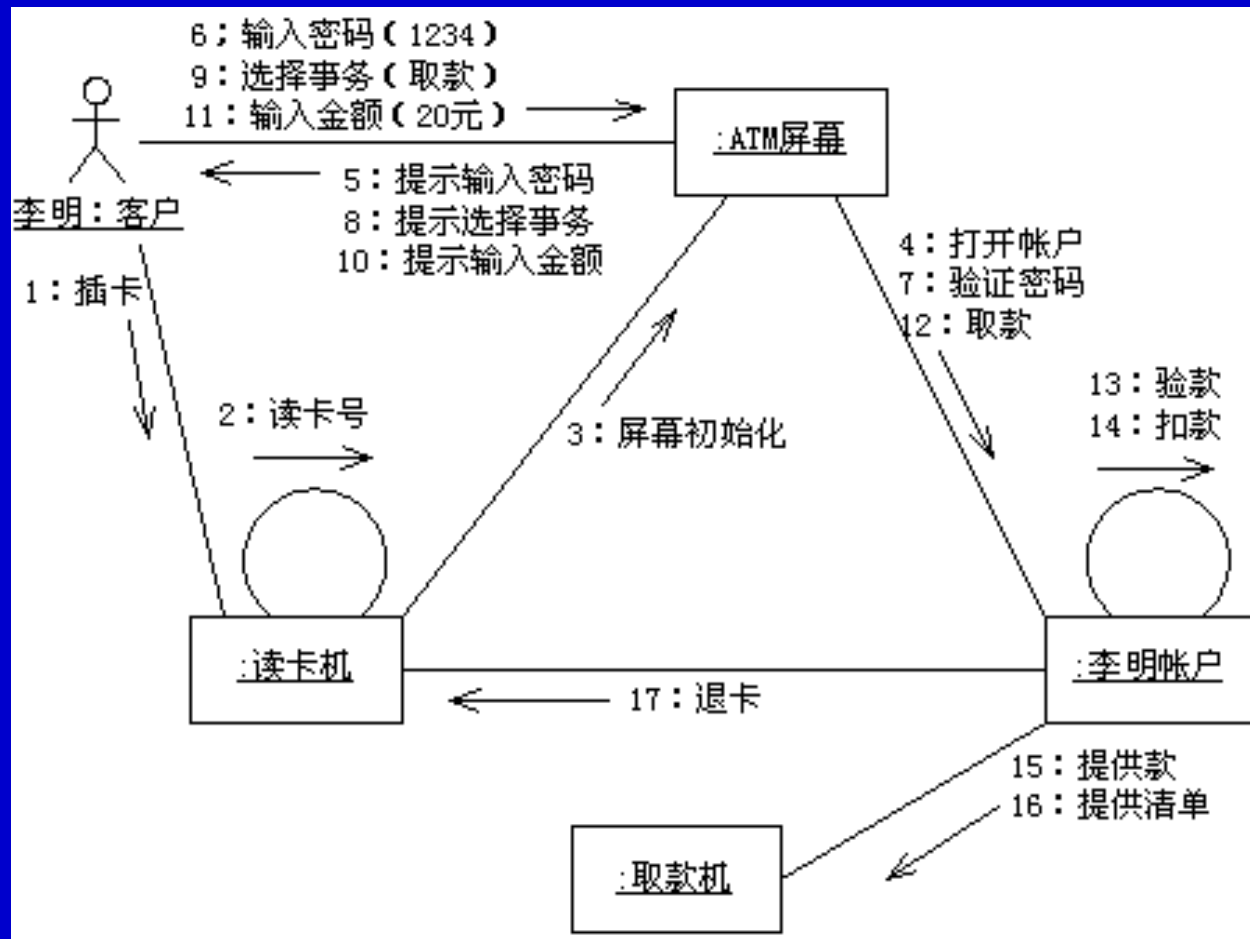
### ◇ 序列图



用来反映若干个对象之间的动态协作关系，也就是随着时间的推移，对象之间是如何交互的



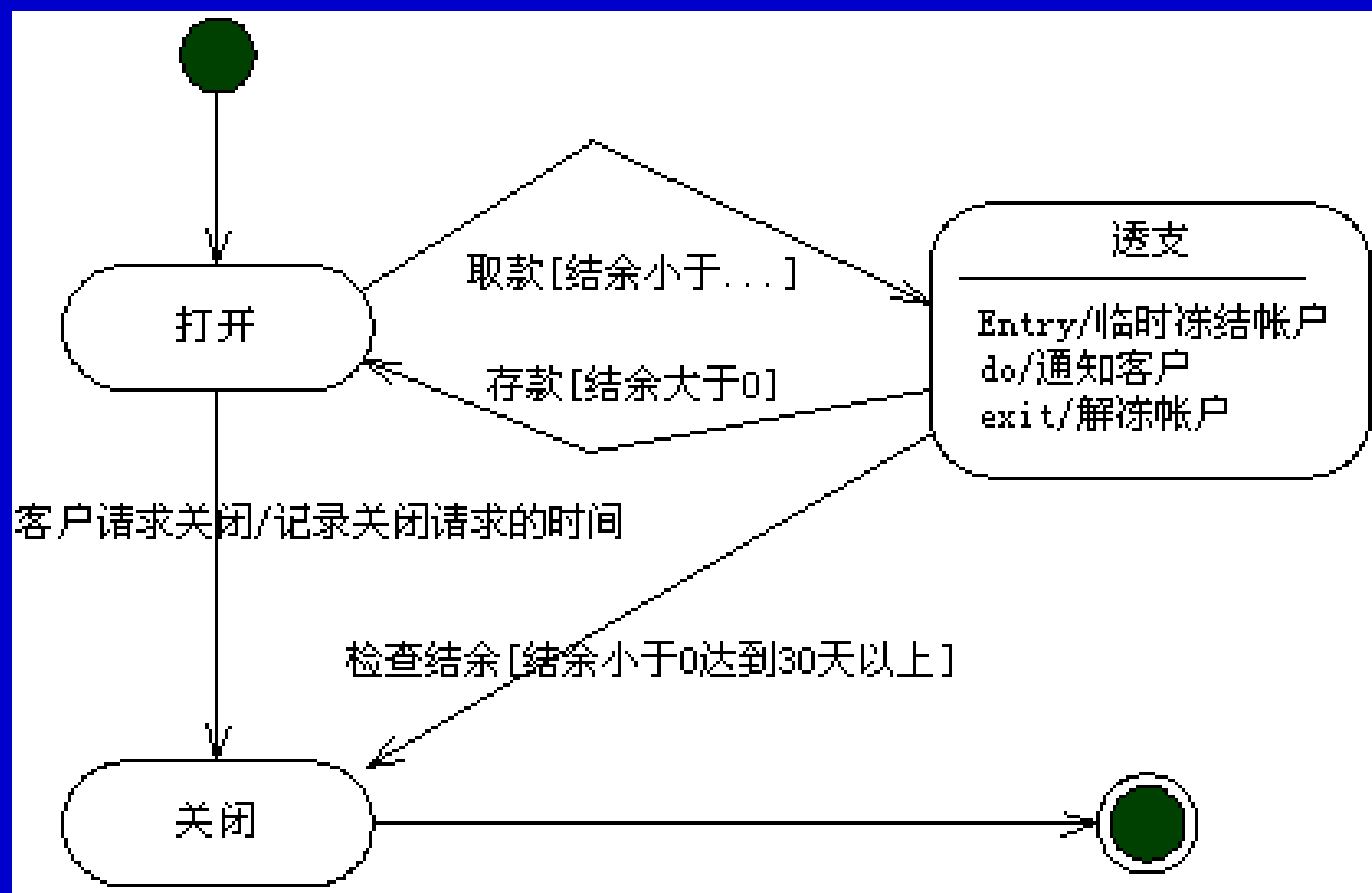
### ◇ 协作图



描述对象间的协作关系，协作图跟序列图相似，显示对象间的动态合作关系。

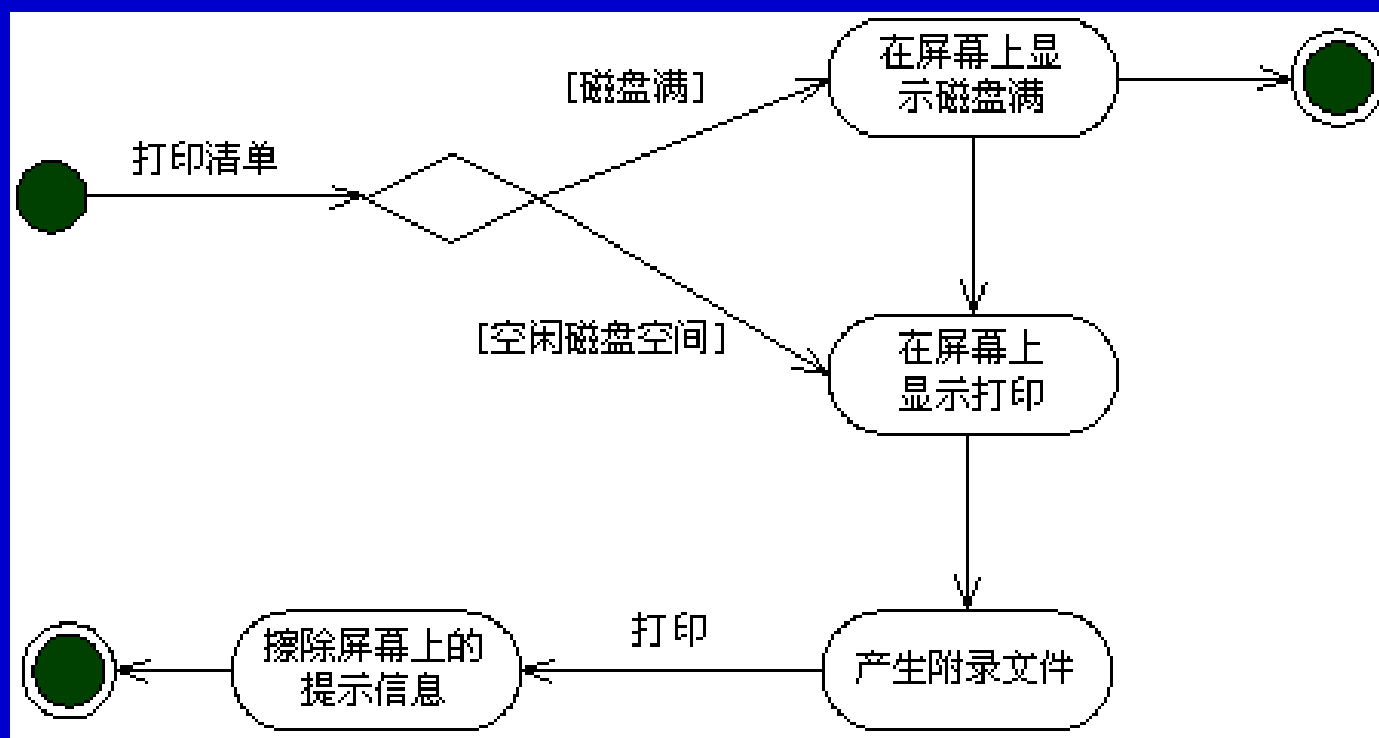
如果强调时间和顺序，则使用序列图；如果强调上下级关系，则选择协作图。这两种图合称为交互图。

### ◇ 状态图



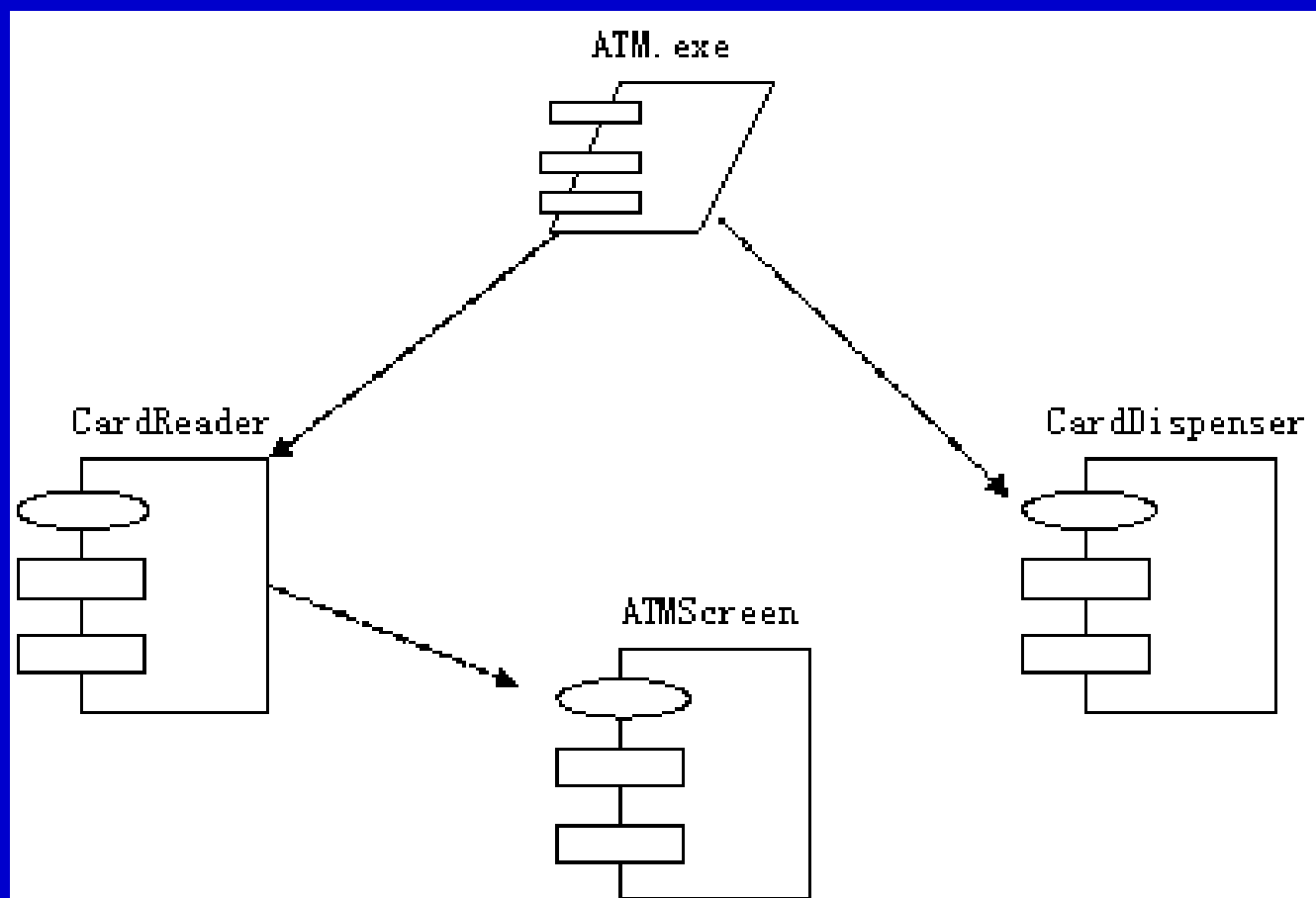
描述类的对象所有可能的状态以及事件发生时状态的转移条件。通常，状态图是对类图的补充

### ◇ 活动图



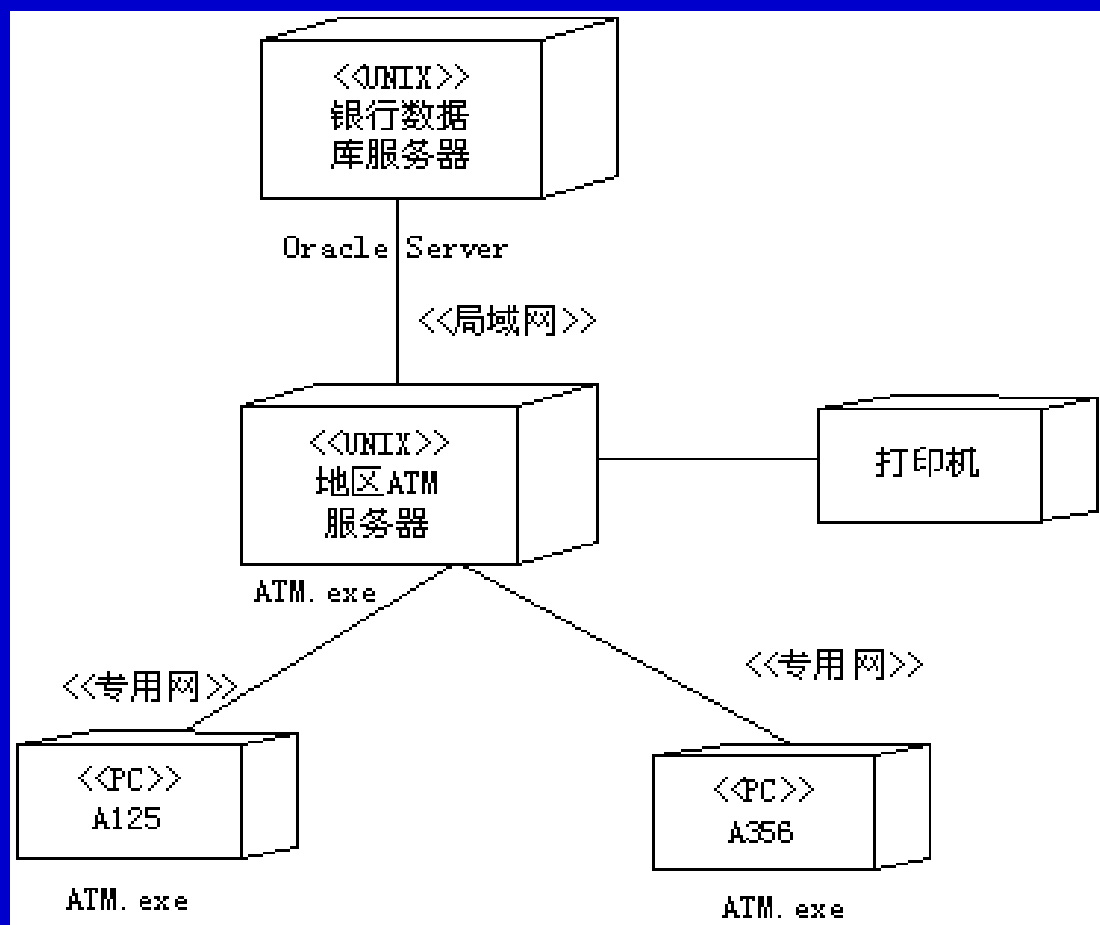
描述满足用例要求所要进行的活动以及活动间的约束关系，有利于识别并行活动

### ◇ 构件图



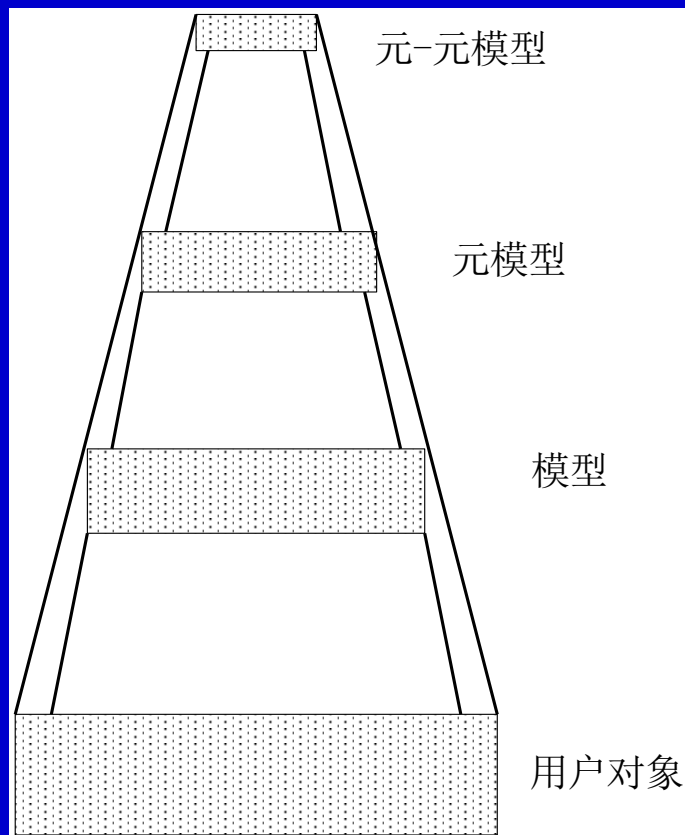
描述代码构件的物理结构及各构件之间的依赖关系

### ◇ 部署图



部署图  
定义系  
统中软  
硬件的  
物理体  
系结构

### ◇ 直接使用UML建模



元-元模型层定义了元模型层的规格说明语言，元模型层为给定的建模语言定义规格说明，模型层用来定义特定软件系统的模型，用户对象用来构建给定模型的特定实例。

#### ◇ 直接使用UML建模

#### ◎ 语义约束

由对象约束语言OCL表示，OCL基于一阶谓词逻辑，每一个OCL表达式都处于一些UML模型元素的背景下（由“self”引用），可使用该元素的属性和关系作为其项（term），同时OCL定义了集合（sets）、袋（bags）等上的公共操作集和遍历建模元素间关系的构造，因此，其它建模元素的属性也可以作为它的项。

### ◇ 直接使用UML建模

#### ◎ UML中的通用表示

- (1) 字符串：表示有关模型的信息；
- (2) 名字：表示模型元素；
- (3) 标号：不同于编程语言中的标号，是用于表示或说明图形符号的字符串；
- (4) 特殊字符串：表示某一模型元素的特性；
- (5) 类型表达式：声明属性、变量及参数，含义同编程语言中的类型表达式；
- (6) 实体类型：它是UML的扩充机制，运用实体类型可定义新类型的模型元素；



### ◇ 直接使用UML建模

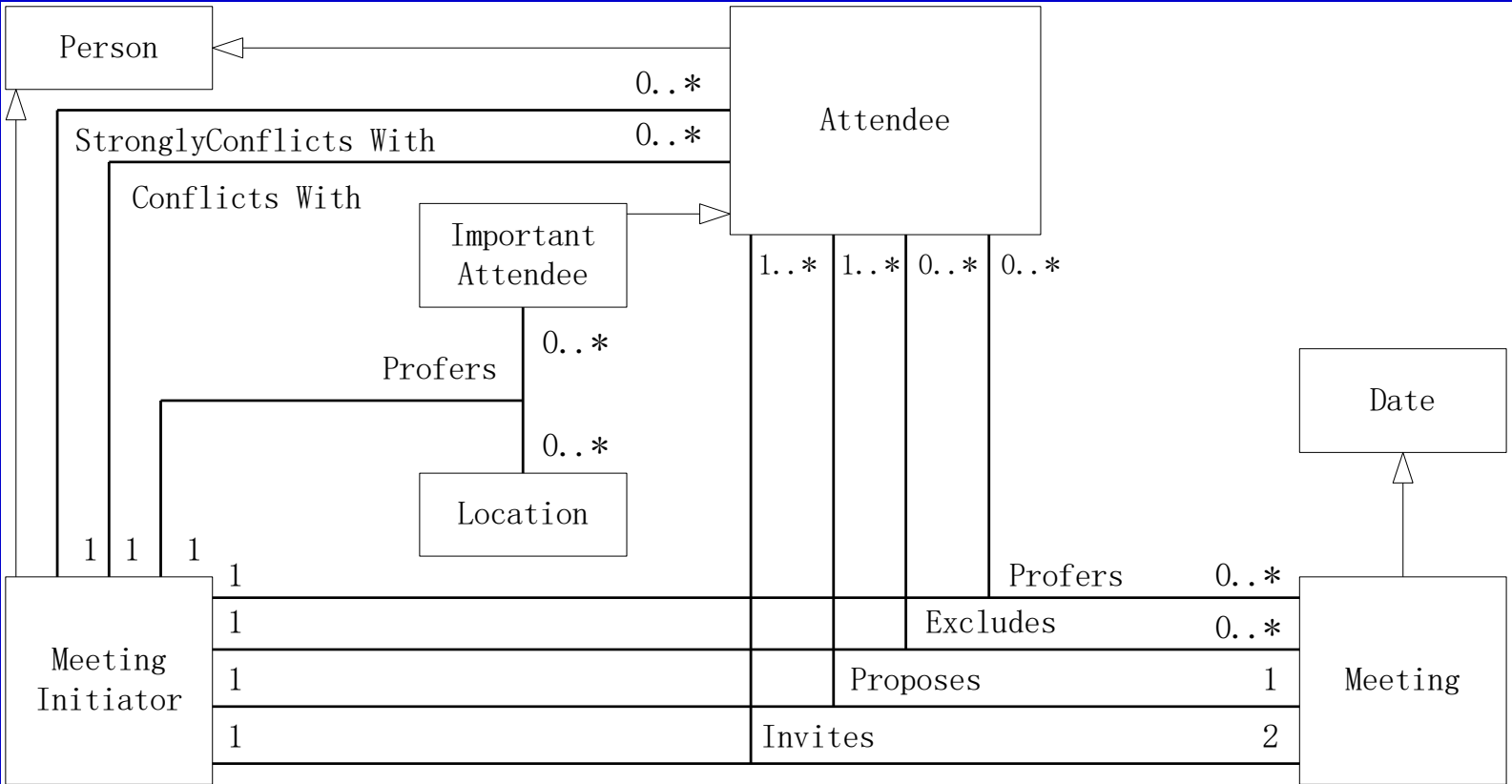
#### ◎ UML语义部分

- ◇ 通用元素：主要描述UML中各元素的语义。通用元素是UML中的基本构造单位，包括模型元素和视图元素，模型元素用来构造系统，视图元素用来构成系统的表示成分；
- ◇ 通用机制：主要描述使UML保持简单和概念上一致的机制的语义。包括定制、标记值、注记、约束、依赖关系、类型-实例、类型-类的对应关系等机制；
- ◇ 通用类型：主要描述UML中各种类型的语义。这些类型包括布尔类型、表达式类型、列表类型、多重性类型、名字类型、坐标类型、字符串类型、时间类型、用户自定义类型等。

三部分不是相互独立的，而是相互交叉重叠、紧密相连，共同构成了UML的完整语义。

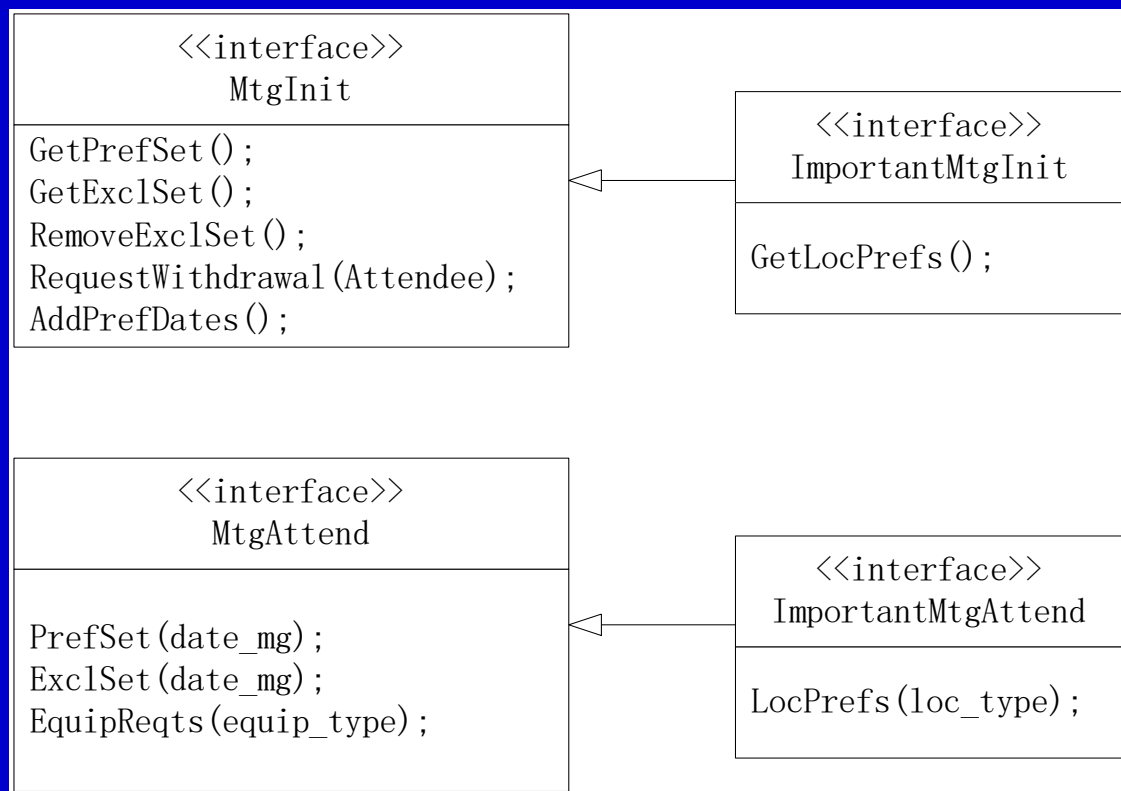
### ◇ 直接使用UML建模

### ◎ 会议安排系统的类图



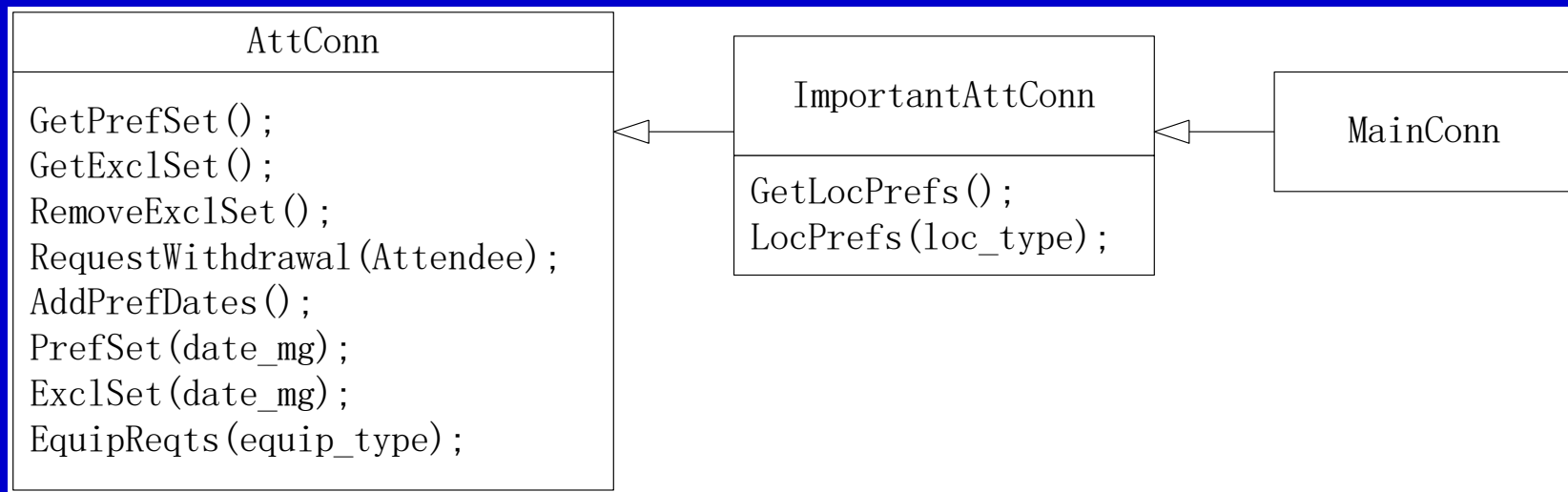
### ◇ 直接使用UML建模

#### ◎ 会议安排系统类接口



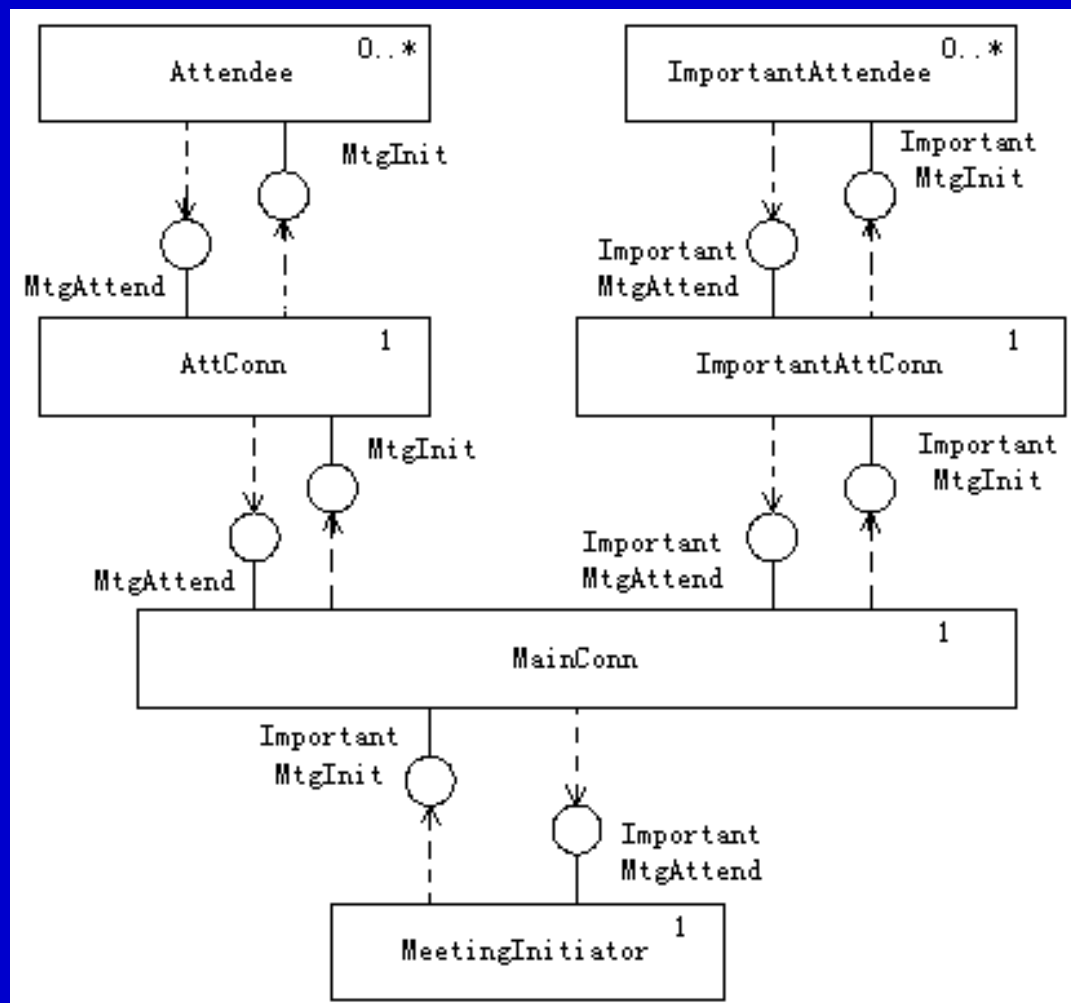
### ◇ 直接使用UML建模

#### ◎ C2连接件模型



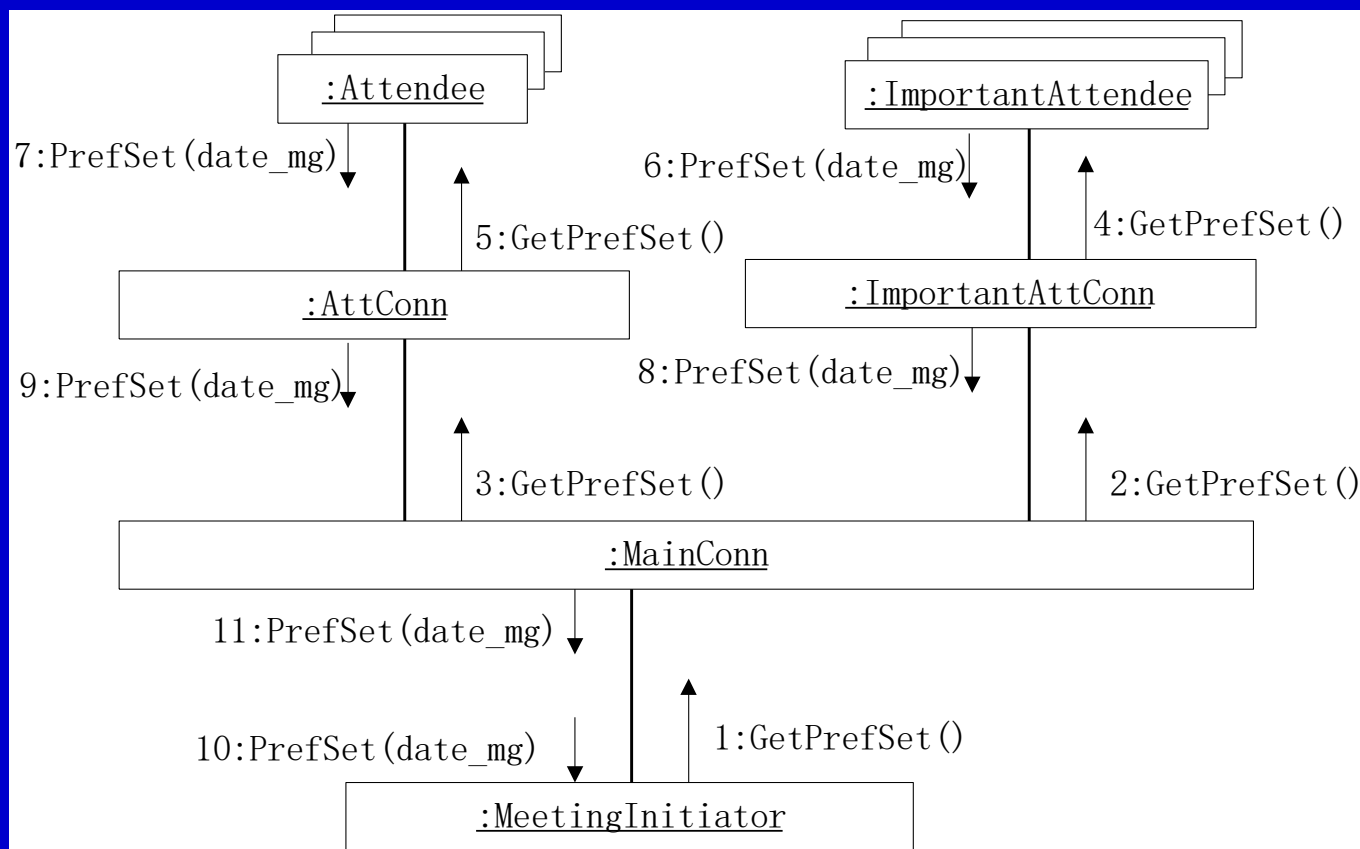
### ◇ 直接使用UML建模

#### ◎ 细化的类图



### ◇ 直接使用UML建模

#### ◎ 会议安排系统的协作图



### ◇ 使用UML扩展机制

自学

- 1、体系结构描述有哪些方法？有哪些标准和规范？
- 2、体系结构描述语言与程序设计语言有什么区别？
- 3、选择一个规模适中的系统，使用UML为其建模。



# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

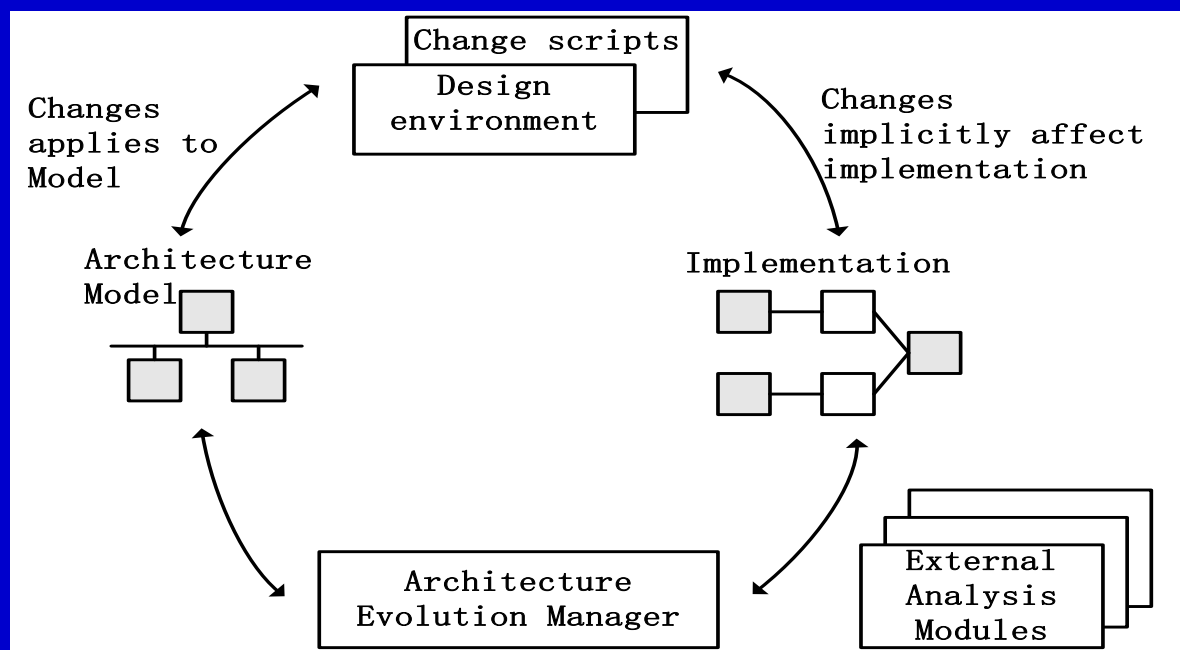
### ◇ 软件体系结构动态性

- ◎ 交互式动态性
- ◎ 结构化动态性
- ◎ 体系结构动态性

### ◇ 动态体系结构的研究

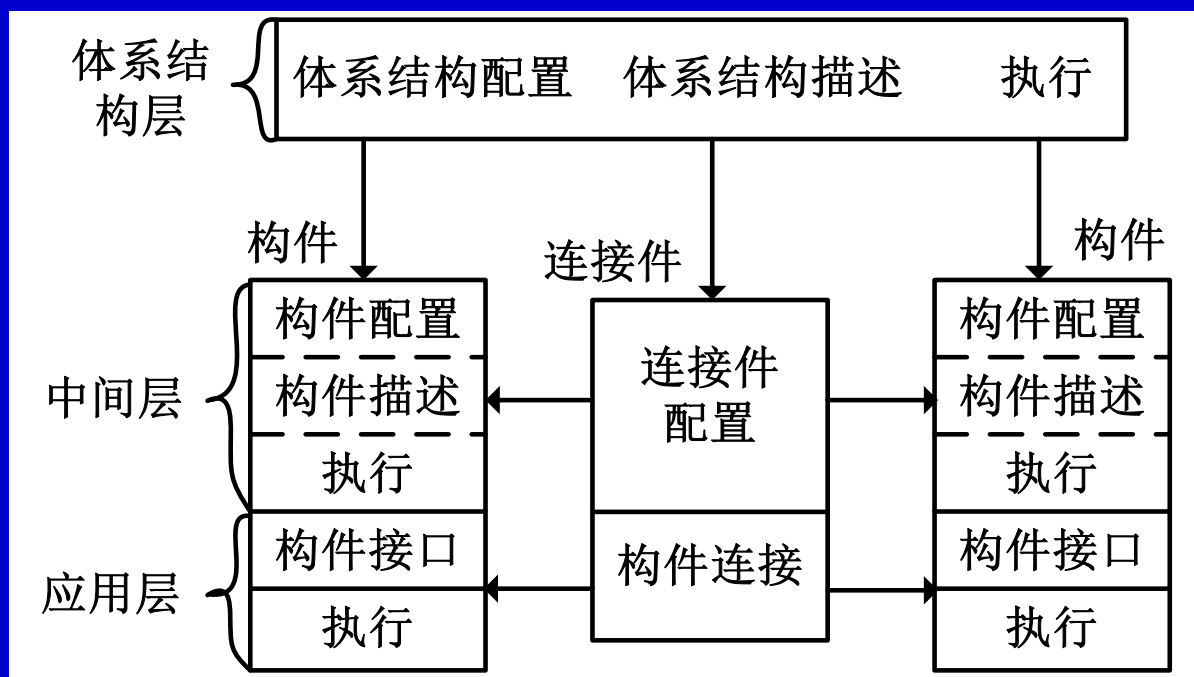
◎ 模拟和描述体系结构动态更新

◎ 体系结构动态更新的执行



### ◇ 基于构件的动态系统结构模型

#### ◎ 模型简介



### ◇ 基于构件的动态系统结构模型

#### ◎ 更新请求描述

- ◇ 更新类型
- ◇ 更新对象列表
- ◇ 对象的新版本说明
- ◇ 对象更新方法
- ◇ 更新函数
- ◇ 更新限制

### ◇ 基于构件的动态系统结构模型

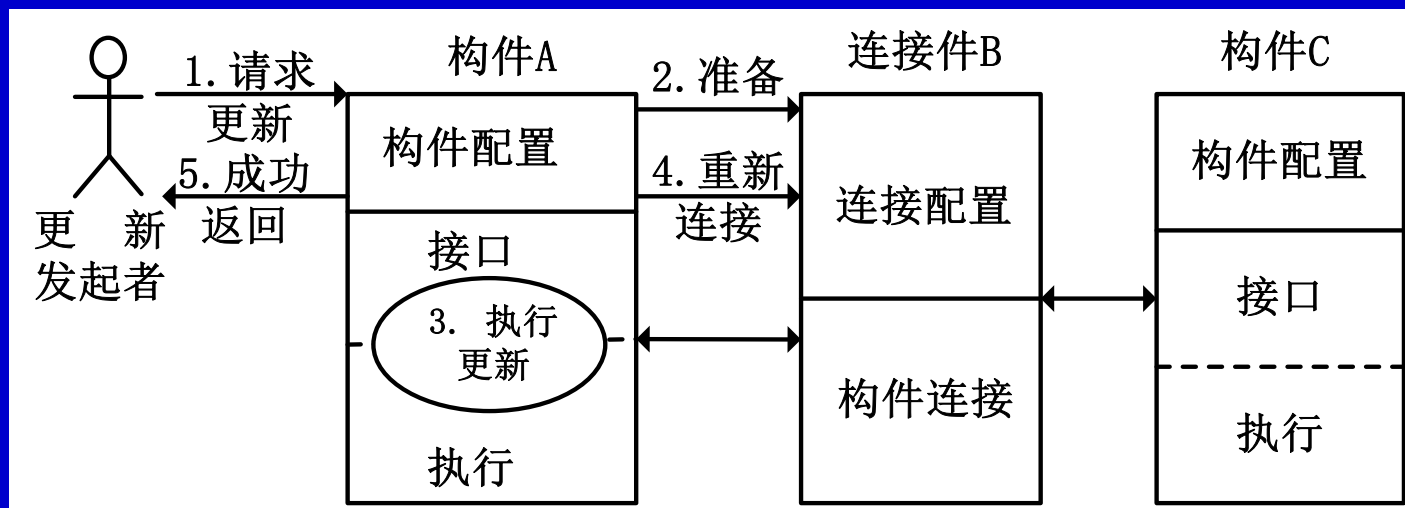
#### ◎ 更新执行步骤

- ◇ 检测更新的范围
- ◇ 更新准备工作
- ◇ 执行更新
- ◇ 存储更新

### ◇ 基于构件的动态系统结构模型

#### ◎ 实例分析

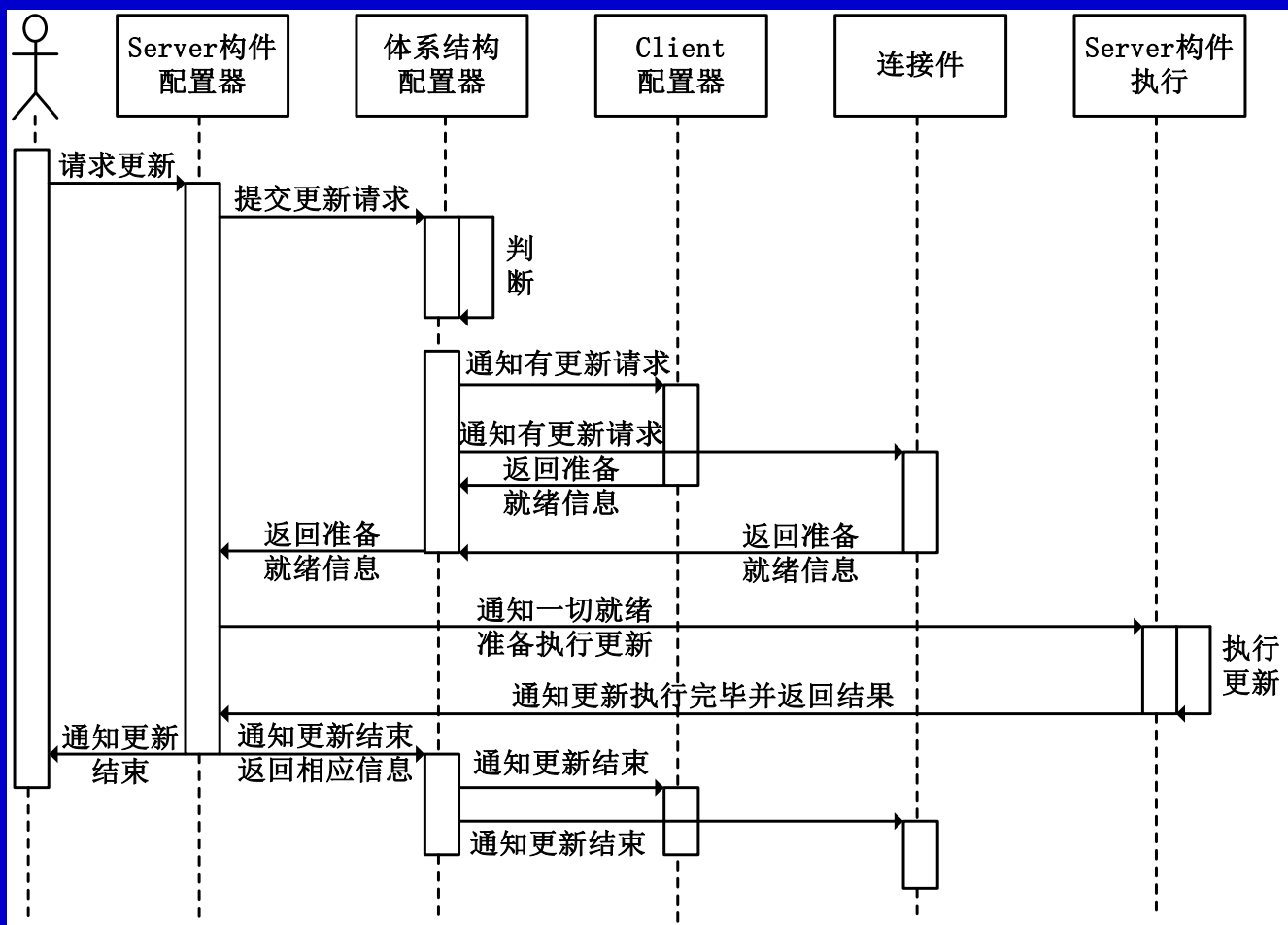
#### ◇ 局部更新



### ◇ 基于构件的动态系统结构模型

#### ◎ 实例分析

#### ◇ 全局更新



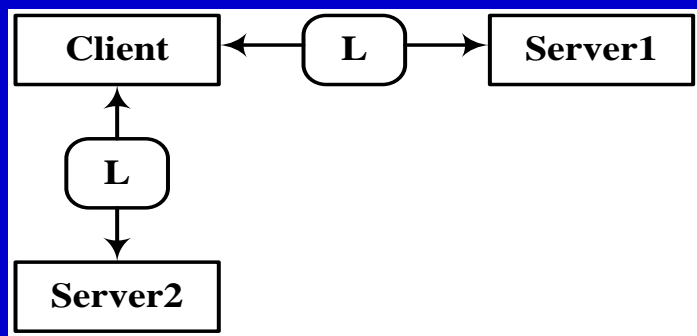


### ◇ $\pi$ ADL动态体系结构

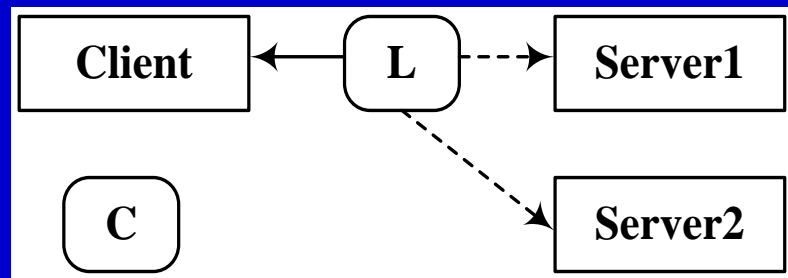
选读

### ◇ 动态体系结构描述语言

#### ◎ Dynamic Wright



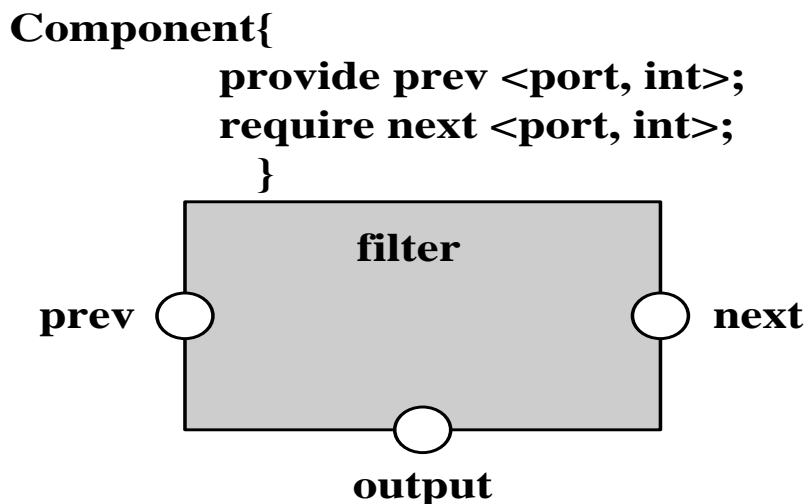
静态描述



动态描述

### ◇ 动态体系结构描述语言

#### ◎ Darwin



### ◇ 动态软件体系结构的形式化描述

#### ◎ 图形化方法

	Architectural Structure		Architectural Element Behavior		Architectural Reconfiguration
	Architectural Style	System Architecture	components	connectors	
Le Metayer approach	Context-free graph grammar	Graph(formally defined as a multiset)	nodes of a graph and a csp like behavior specification	edges of a graph	graph rewriting rules with side conditions to refer to the status of public variables
Hirsch et al approach	Context-free graph grammar	hypergraph	edges of a graph with CCS labels	nodes of a graph[point-point communication and broadcast communication]	graph rewriting rules
Taentzer et al approach	—	distributed graph (network graph)	local graph of each network graph node and local transformations between local graphs	edges of a graph	graph rewriting rules
CHAM	creation CHAM	—	molecule	links between two component molecules	evolution CHAM reaction rules

### ◇ 动态软件体系结构的形式化描述

#### ◎ 进程代数方法

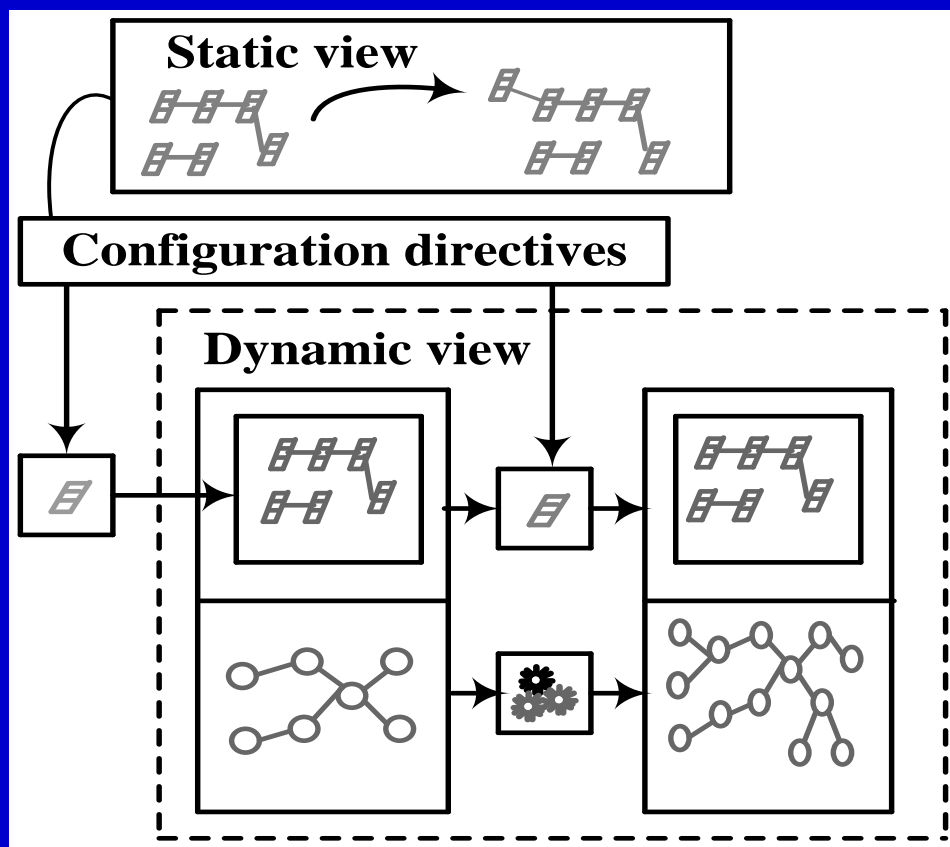
	Architectural Structure		Architectural Element Behavior		Architectural Reconfiguration
	Architectural Style	System Architecture	components	connectors	
Dynamic Wright	—	implicit graph representation(components and connectors are nodes)	port(interface)+computation (behavior)	roles(interface) + glue(behavior)	CSP
Darwin	—	implicit graph representation	programming language + component specification of objects	support for simple bindings	CSP
LEDA	—	implicit graph representation	interface specification, composition and attachment specification	attachments at top level components	$\pi$ -calculus
PiLar	—	implicit graph representation	components with ports, instances of other components and constraints	support for simple bindings	CCS

### ◇ 动态软件体系结构的形式化描述

#### ◎ 逻辑化描述方法

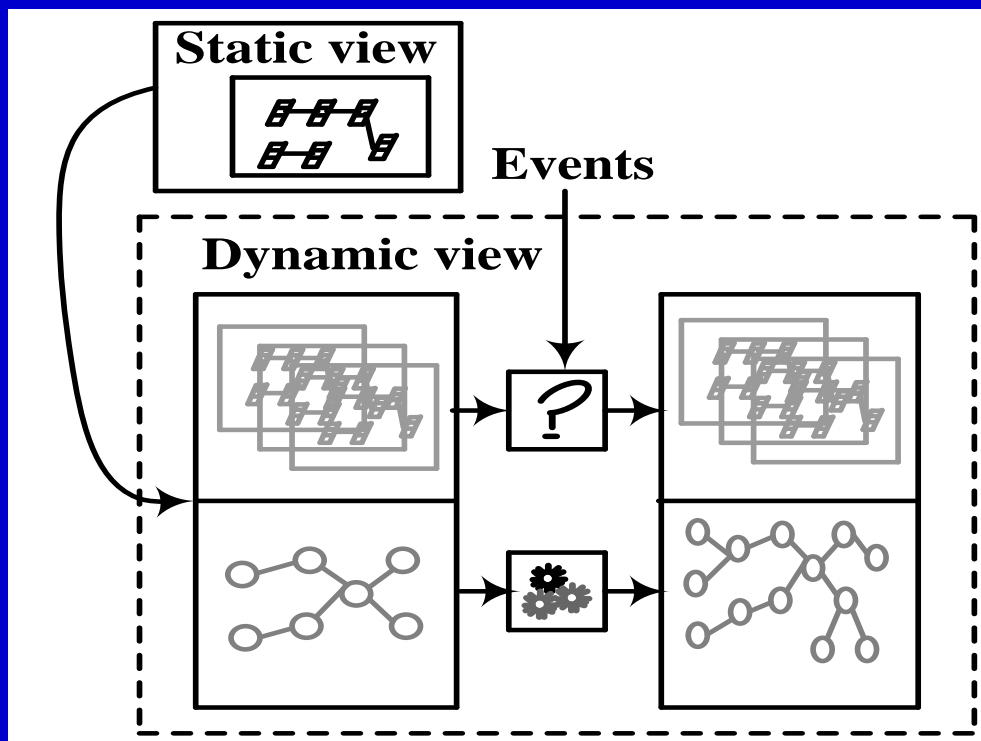
	Architectural Structure		Architectural Element Behavior		Architectural Reconfiguration
	Architectural Style	System Architecture	components	connectors	
Gerel	—	implicit graph representation	interface in Gerel language and behavior in a programming language	defined by bind operation in configuration component	first order logic
Aguirre-Maibaum	—	implicit graph representation	class with attributes, actions and read variables	association consisting of participants and synchronization connections	first order logic, temporal logic
ZCL	—	implicit graph representation (defined by set of state schema in Z)	state schema in Z	connection between ports of component	operation schema in Z(predicate logic and set theory)

### ◇ 可构造性动态特征



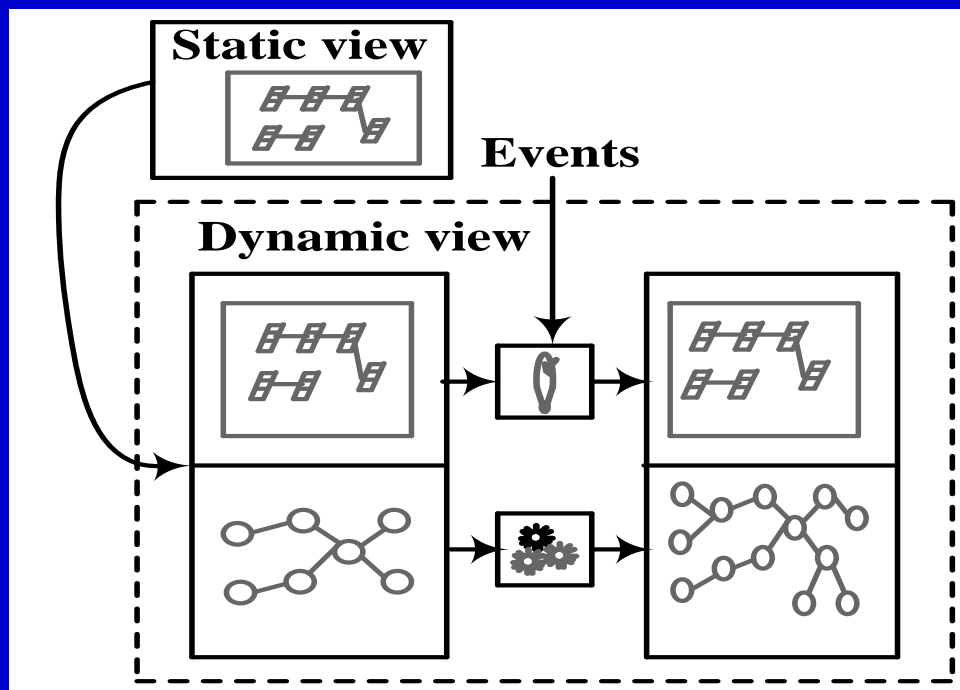
★  
特征  
区别

### ◇ 适应性动态特征





### ◇ 智能性动态特征



- 1、什么是动态软件体系结构？动态软件体系结构与静态软件体系结构有什么区别？
- 2、基于构件的动态软件体系结构模型的层次结构是什么？
- 3、如何使用  $\pi$  ADL进行动态体系结构建模？
- 4、试比较Dynamic Wright和Darwin的特点。
- 5、试用Dynamic Wright描述B/S结构。

# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ **Web服务体系结构**
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

### ◇ 什么是Web服务

◎ Web服务作为一种新兴的Web应用模式，是一种崭新的分布式计算模型，是Web上数据和信息集成的有效机制。

◎ Web服务就像Web上的构件编程，开发人员通过调用Web应用编程接口，将Web服务集成进他们的应用程序，就像调用本地服务一样。

### ◇ 什么是Web服务

- ◎ 数据层
- ◎ 数据访问层
- ◎ 业务层
- ◎ 业务面
- ◎ 监听者

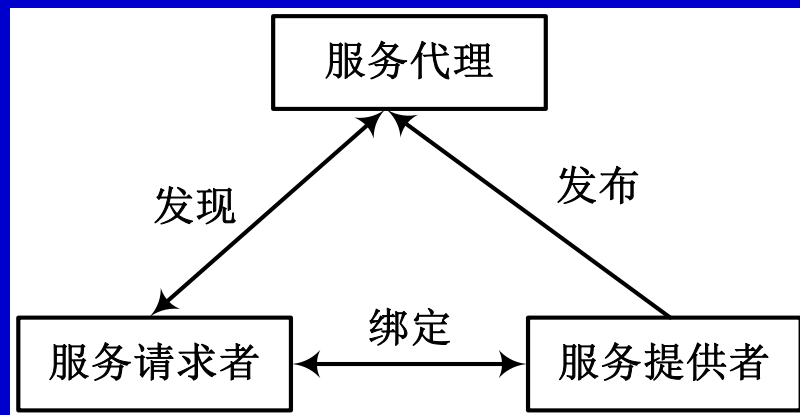
### ◇ Web服务的不同描述

- ◎ 应用的分布式
- ◎ 应用到应用的交互
- ◎ 平台无关性

### ◇ Web服务的特点

- ◎ 使用标准协议规范
- ◎ 使用协约的规范性
- ◎ 高度集成能力
- ◎ 完好的封装性
- ◎ 松散耦合

### ◇ Web服务模型





### ◇ Web服务开发生命周期

◎ 构建

◎ 部署

◎ 运行

◎ 管理

### ◇ Web服务栈

发现服务	UDDI、DISCO
描述服务	WSDL、XML Schema
消息格式层	SOAP
编码格式层	XML
传输协议层	HTTP, TCP/IP, SMTP等

### ◇ Web服务体系结构的优势

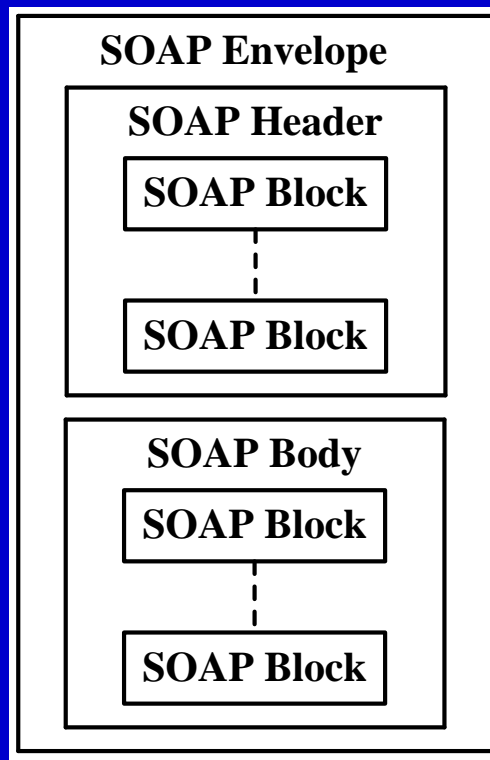
- ◎ 高度的通用性和易用性
- ◎ 完全的平台、语言独立性
- ◎ 高度的集成性
- ◎ 容易部署和发布

#### ◇ 作为Web服务基础的XML

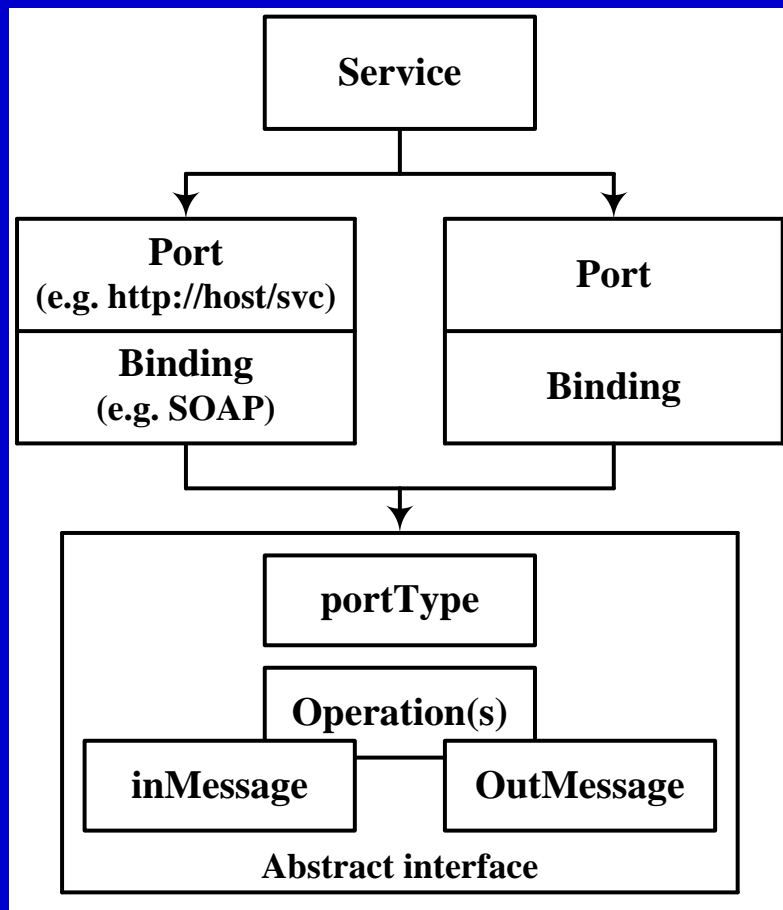
◎ XML是W3C制定的作为Internet上数据交换和表示的标准语言，是一种允许用户定义自己的标记的元语言。

### ◇ 简单对象访问协议

- ◎ SOAP信封
- ◎ SOAP编码规则
- ◎ SOAP RPC表示
- ◎ SOAP绑定



### ◇ Web服务描述语言



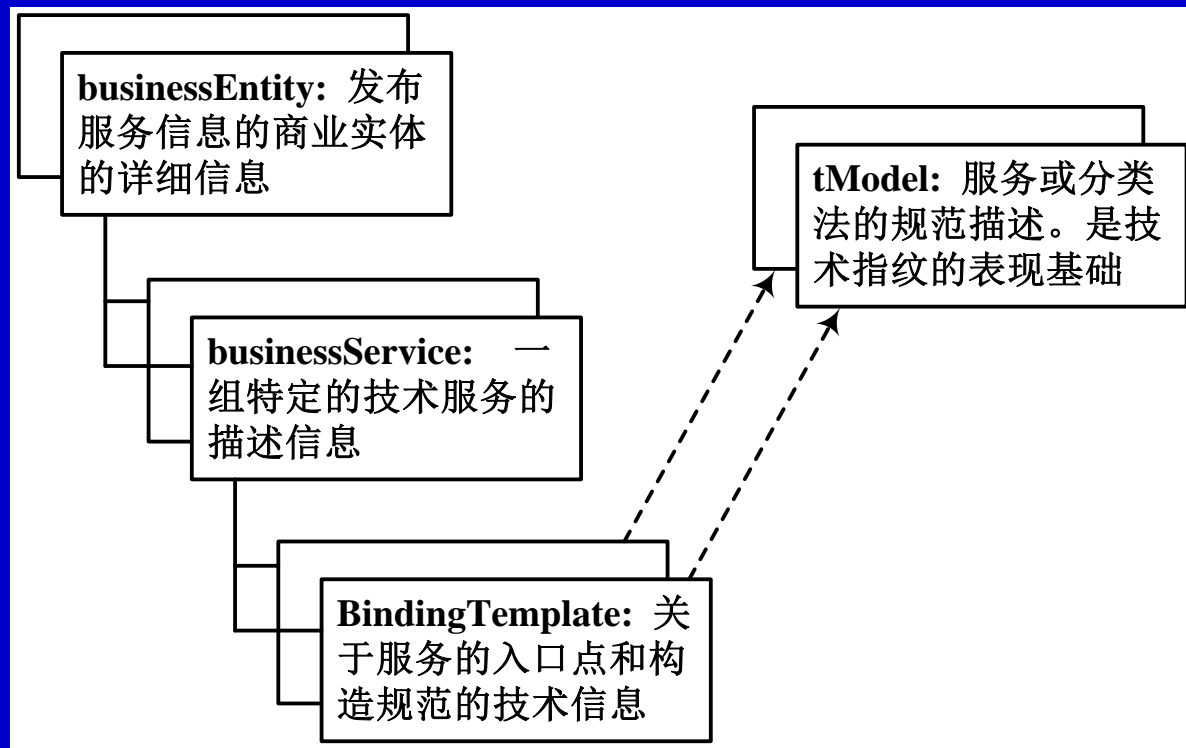
### ◇ 统一描述、发现和集成协议

◎ 商业实体结构

◎ 商业服务结构

◎ 绑定模板

◎ t模型结构



#### ◇ SOA的概念

##### ◎ W3C定义

SOA为一种应用程序体系结构，在这种体系结构中，所有功能都定义为独立的服务，这些服务带有定义明确的可调用接口，可以以定义好的顺序调用这些服务来形成业务流程。



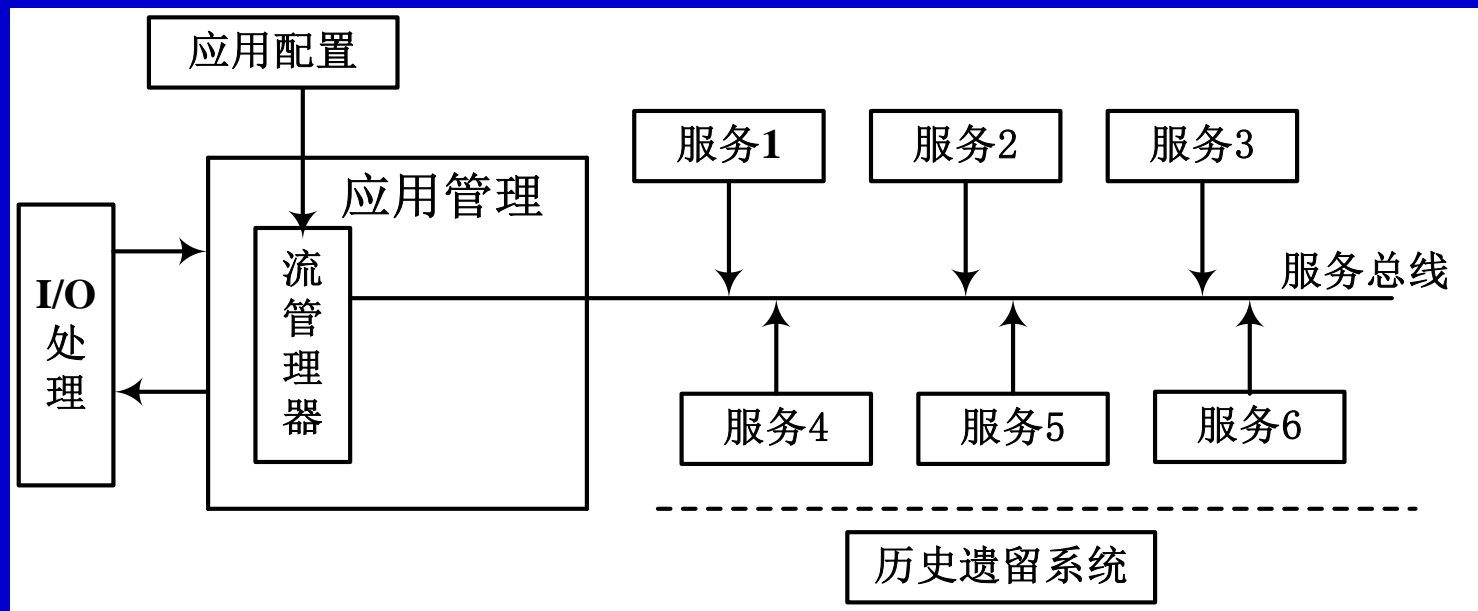
#### ◇ SOA的概念

##### ◎ Gartner定义

SOA为客户端/服务器的软件设计方法，一项应用由软件服务和软件服务使用者组成，SOA与大多数通用的客户端/服务器模型不同之处，在于它着重强调软件构件的松散耦合，并使用独立的标准接口。

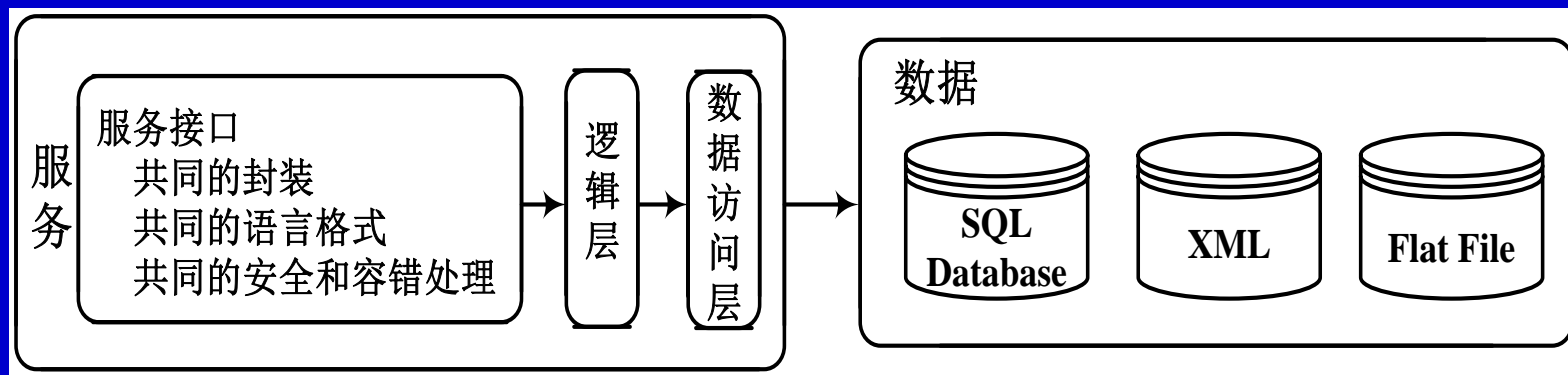
### ◇ SOA的概念

#### ◎ 一个完整的面向服务的体系结构模型



### ◇ SOA的概念

#### ◎ 单个服务内部结构



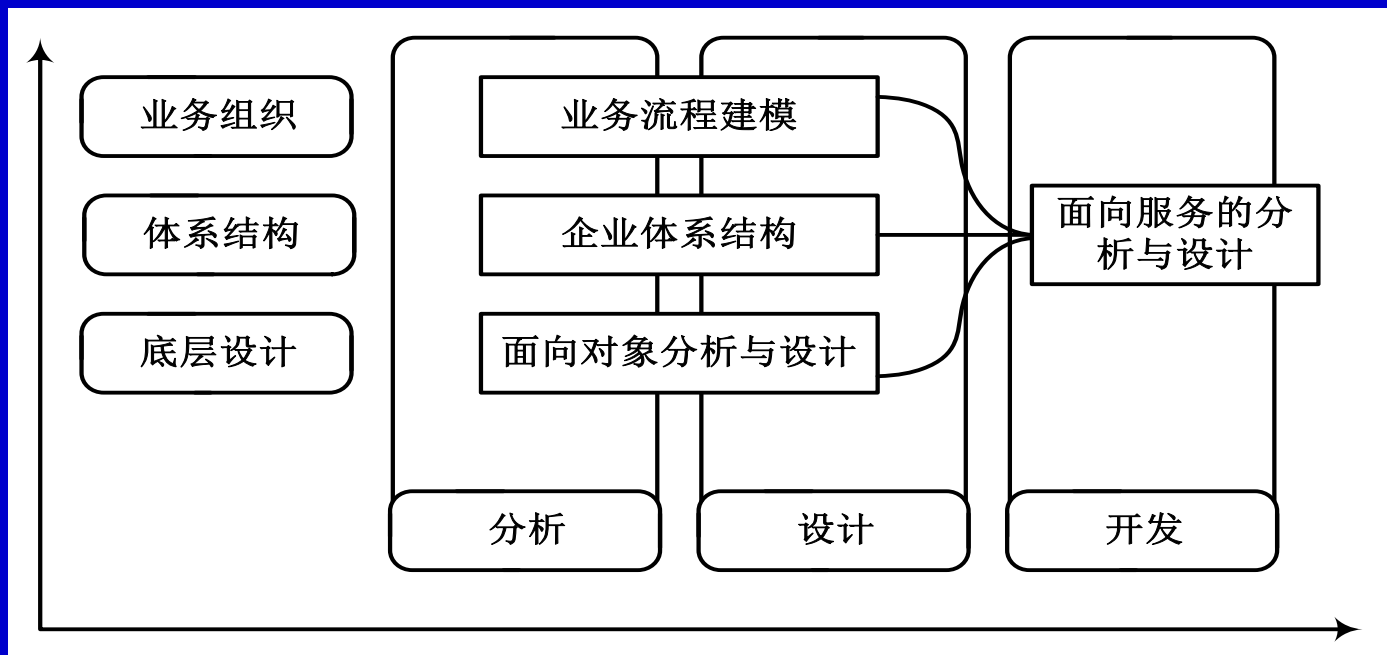
### ◇ SOA的概念

#### ◎ SOA模型的特征

- ◎ 松散耦合
- ◎ 粗粒度服务
- ◎ 标准化接口

### ◇ SOA的设计原则

#### ◎ 面向服务的分析与设计原理

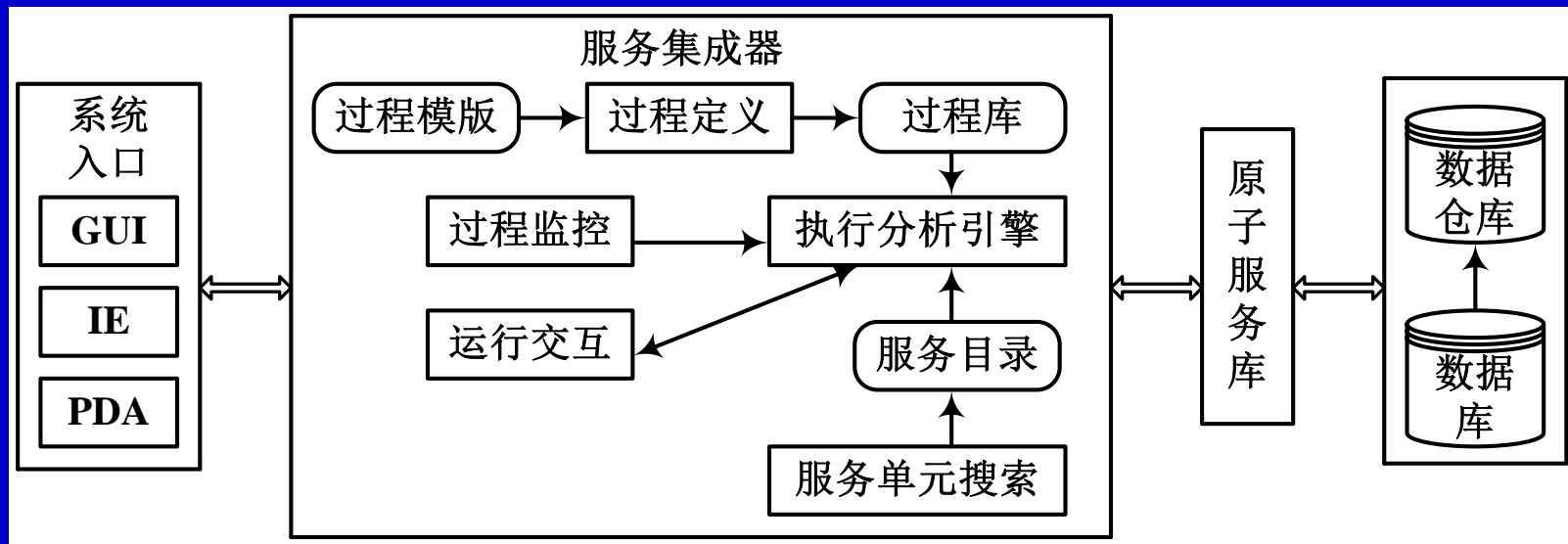


#### ◇ SOA的设计原则

##### ◎ SOA的实践原则

◎ 业务驱动服务，服务驱动技术

◎ 业务敏捷是基本的业务需求



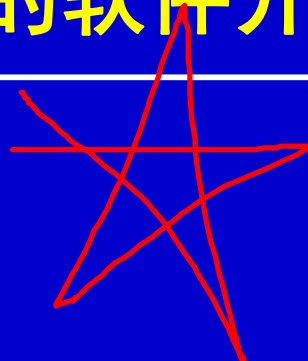
- 1、什么是Web服务体系结构？与传统的结构相比，使用Web服务有哪些好处？
- 2、在Web服务中，如何实现其松散耦合的特点？
- 3、试分析服务提供者、服务请求者和服务代理三者的作用，以及它们之间的工作流程。
- 4、试解释Web服务栈的层次结构。
- 5、Web服务有哪些核心技术，这些技术是如何在Web服务中发挥作用的。
- 6、从管理的角度看，SOA有什么优点？
- 7、在实际开发中，如何实现Web服务和SOA结构？



# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

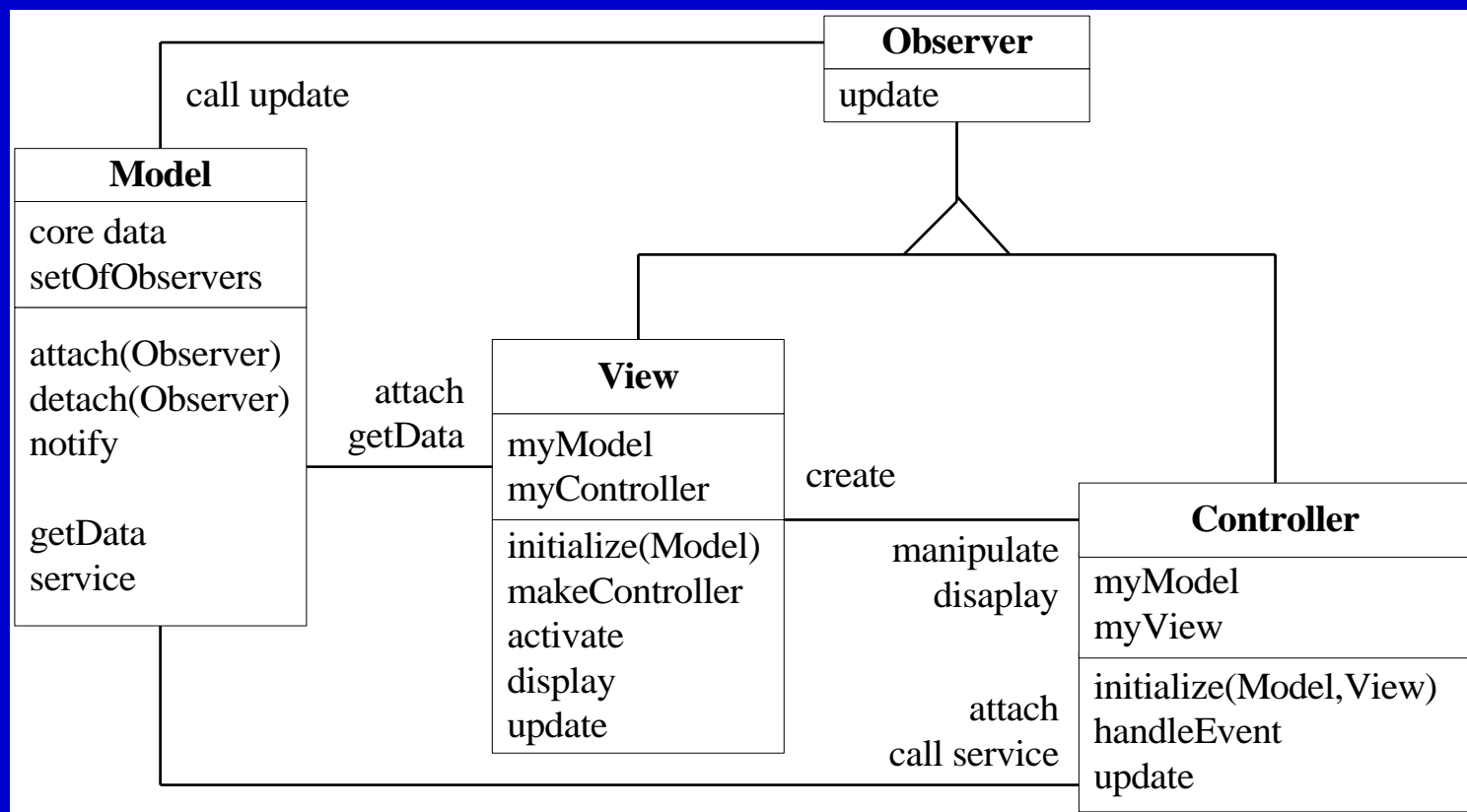
### ◇ 设计模式概述



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

#### ◎ MVC模式



### ◇ 设计模式概述

一个好的模式必须做到以下几点：

- ◎ 解决一个问题：从模式可以得到解，而不仅仅是抽象的原则或策略。
- ◎ 是一个被证明了的观念：模式通过一个记录得到解，而不是通过理论或推测。
- ◎ 解并不是显然的：许多解决问题的方法（例如软件设计范例或方法）是从最基本的原理得到解；而最好的方法是以非直接的方式得到解，对大多数比较困难的设计问题来说，这是必要的。
- ◎ 描述了一种关系：模式并不仅仅描述模块，它给出更深层的系统结构和机理。
- ◎ 模式有重要的人为因素：所有的软件服务于人类的舒适或生活质量，而最好的模式追求它的实用性和美学。

### ◇ 设计模式的基本成分

- ◎ 模式名称
- ◎ 问题
- ◎ 解决方案
- ◎ 后果

### ◇ 设计模式的描述（1）

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
- （2）目的
- （3）别名
- （4）动机
- （5）应用
- （6）结构

### ◇ 设计模式的描述（2）

Erich Gamma博士等人采用下面的固定模式来描述：

- （7）成分
- （8）合作
- （9）后果
- （10）实现
- （11）例程代码
- （12）已知的应用
- （13）相关模式

### ◇ 模式和软件体系结构

- ◎ 模式作为体系结构构造块
- ◎ 构造异构体系结构
- ◎ 模式和方法
- ◎ 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（1）

1992年，美国的面向对象技术的大师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（2）

（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。

（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OOP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。

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于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

在应用程序框架“菜谱”中有很多“菜谱条”，它们用一种不很规范的方式描述了如何应用框架来解决特定的问题。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。

“菜谱条”通常并不讲解框架的内部设计实现，只讲如何使用。

不同的框架有各自的“菜谱”。

### ◇ 设计模式方法分类

#### 4、形式合约(1)

形式合约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。但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。

##### 优点：

(1) 符号所包含的元素很少，并且其中引入的概念能够被映射成为面向对象程序设计语言中的概念。例如，参与者映射成为对象。

(2) 形式合约中考虑到了复杂行为是由简单行为组成的事实，合约的修订和扩充操作使得这种方法很灵活，易于应用。

### ◇ 设计模式方法分类

#### 4、形式合约(2)

缺点：

(1) 在某些情况下很难用，过于繁琐。若引入新的符号，则又使符号系统复杂化。

(2) 强制性地要求过分精密，从而在说明中可能发生隐患（例如冗余）。

(3) 形式合约的抽象程度过低，接近面向对象的程序设计语言，不易分清主次。

### ◇ 设计模式目录的内容

Gamma在他的博士论文中总结了一系列的设计模式，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性模式、结构性模式和行为性模式。创建性模式处理的是对象的创建过程，结构性模式处理的是对象/类的组合，行为性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。



#### ◇ 有关概念

ABSD方法为产生软件系统的概念体系结构提供构造，概念体系结构是由Hofmeister、Nord和Soni提出的四种不同的体系结构中的一种，它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，相应地，它是达到系统质量和商业目标的关键，为达到预定功能提供了一个基础。

体系结构驱动，是指构成体系结构的商业、质量和功能需求的组合。

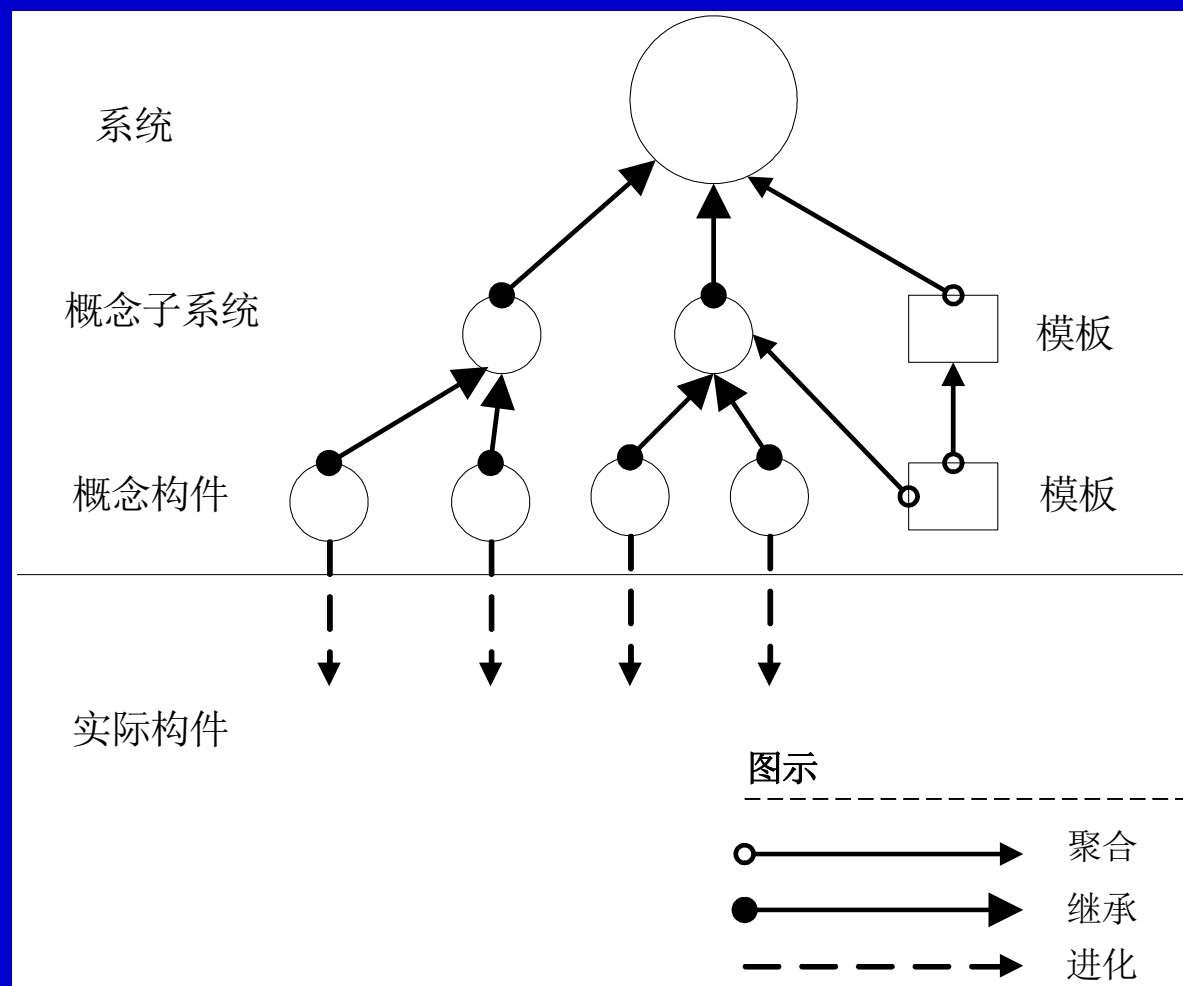
使用ABSD方法，设计活动可以在体系结构驱动一决定就开始，这意味着需求抽取和分析还没有完成，就开始了软件设计。设计活动的开始并不意味着需求抽取和分析活动就可以终止，而是应该与设计活动并行。特别是在不可能预先决定所有需求时，例如产品线系统或长期运行的系统，快速开始设计是至关重要的。

### ◇ 有关概念

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。

### ◇ 有关概念

#### 设计元素

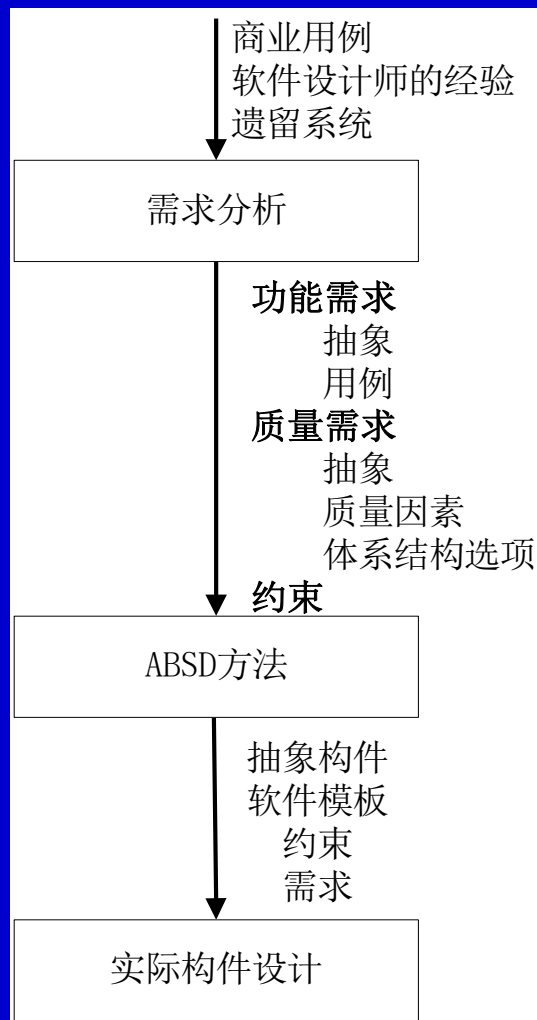


### ◇ 有关概念

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。这样一来，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，分别称之为变更场景、性能场景、可靠性场景和交互性场景。质量场景必须包括预期的和非预期的刺激。

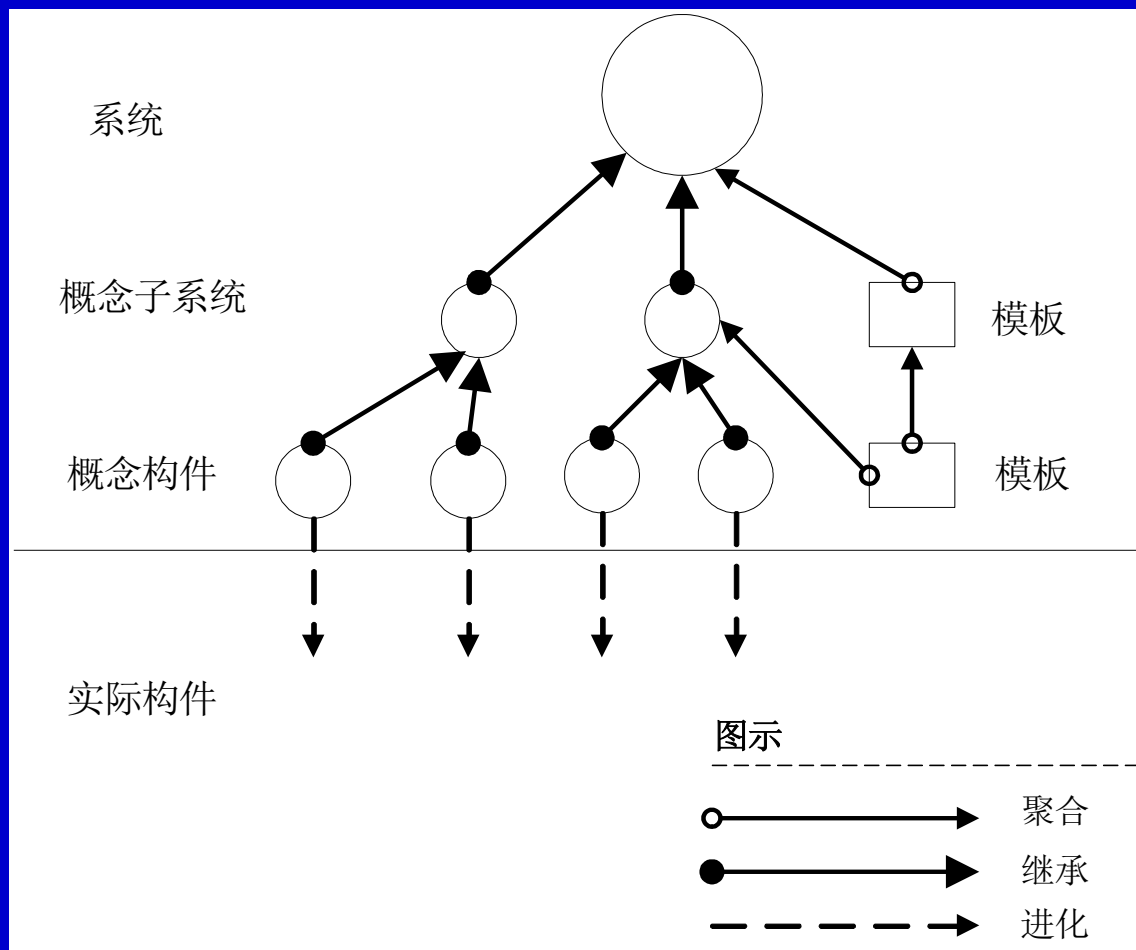
例如，一个预期的性能场景是估计每年用户数量增加10%的影响，一个非预期的场景是估计每年用户数量增加100%的影响。非预期场景可能不能真正实现，但它们在决定设计的边界条件时很有用。

### ◇ ABSD方法与生命周期



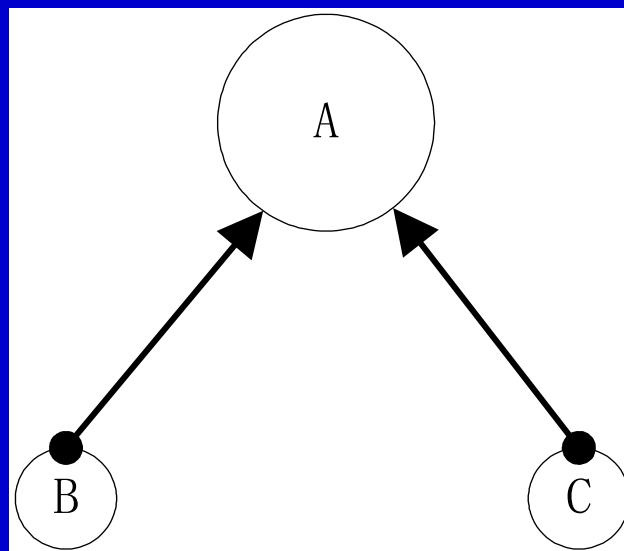
### ◇ ABSD方法的步骤

#### 1、ABSD方法定义的设计元素



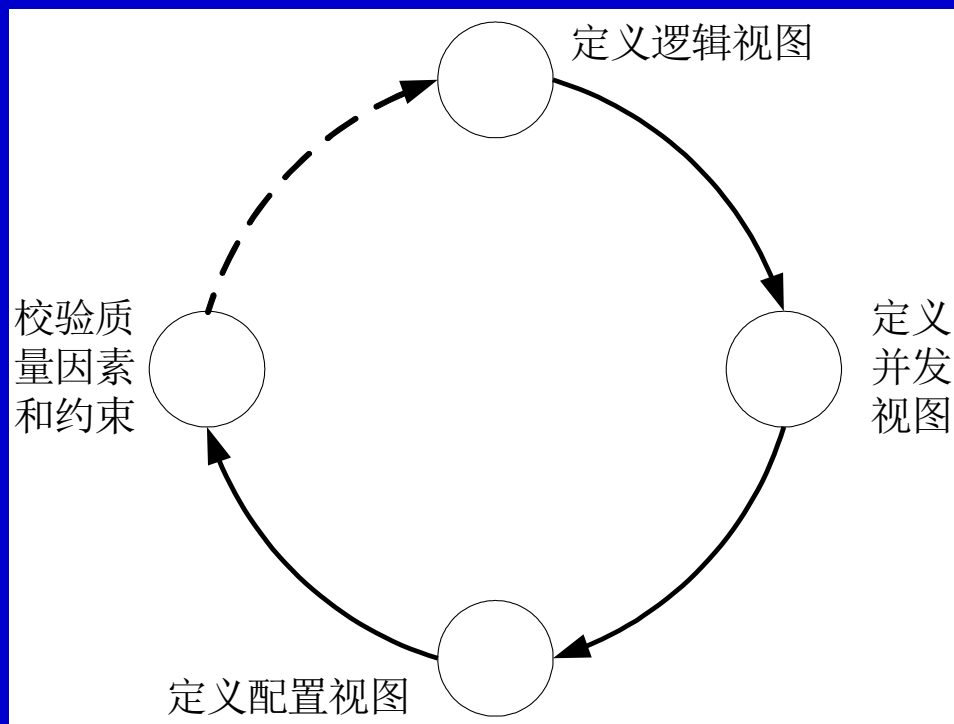
### ◇ ABSD方法的步骤

#### 2、设计元素的产生顺序



### ◇ ABSD方法的步骤

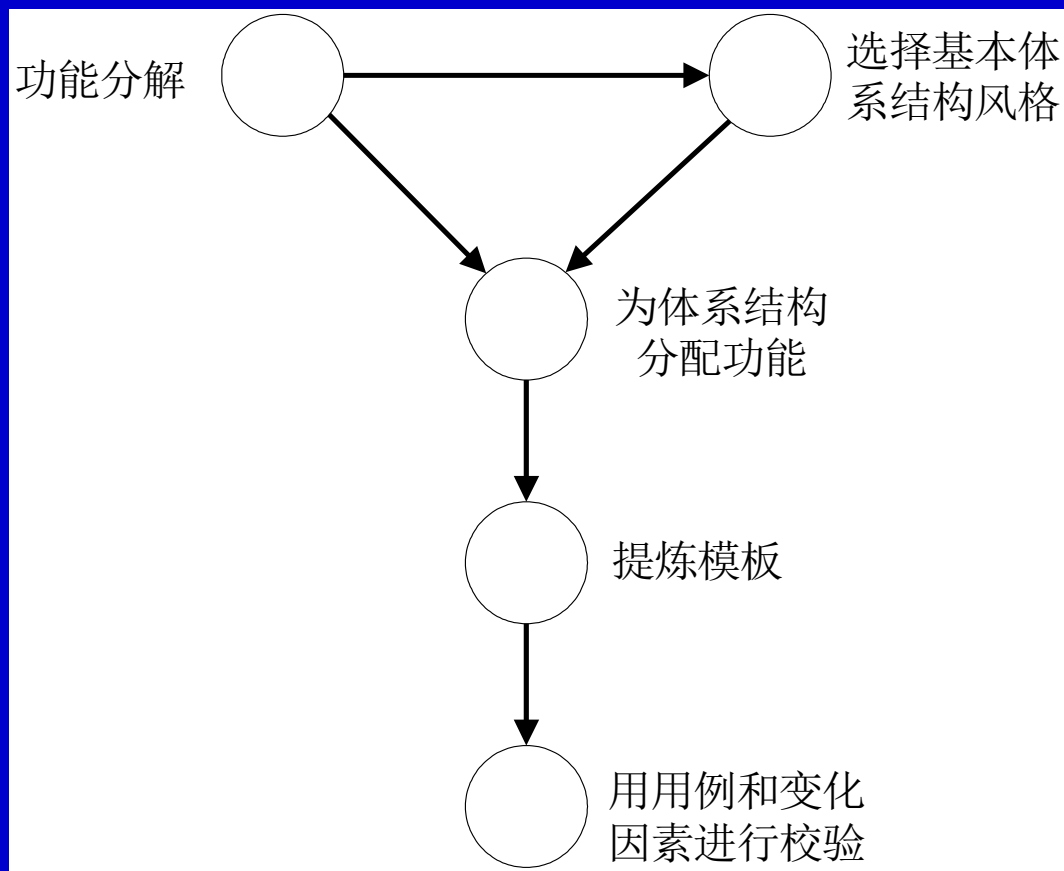
#### 3、设计元素的活动





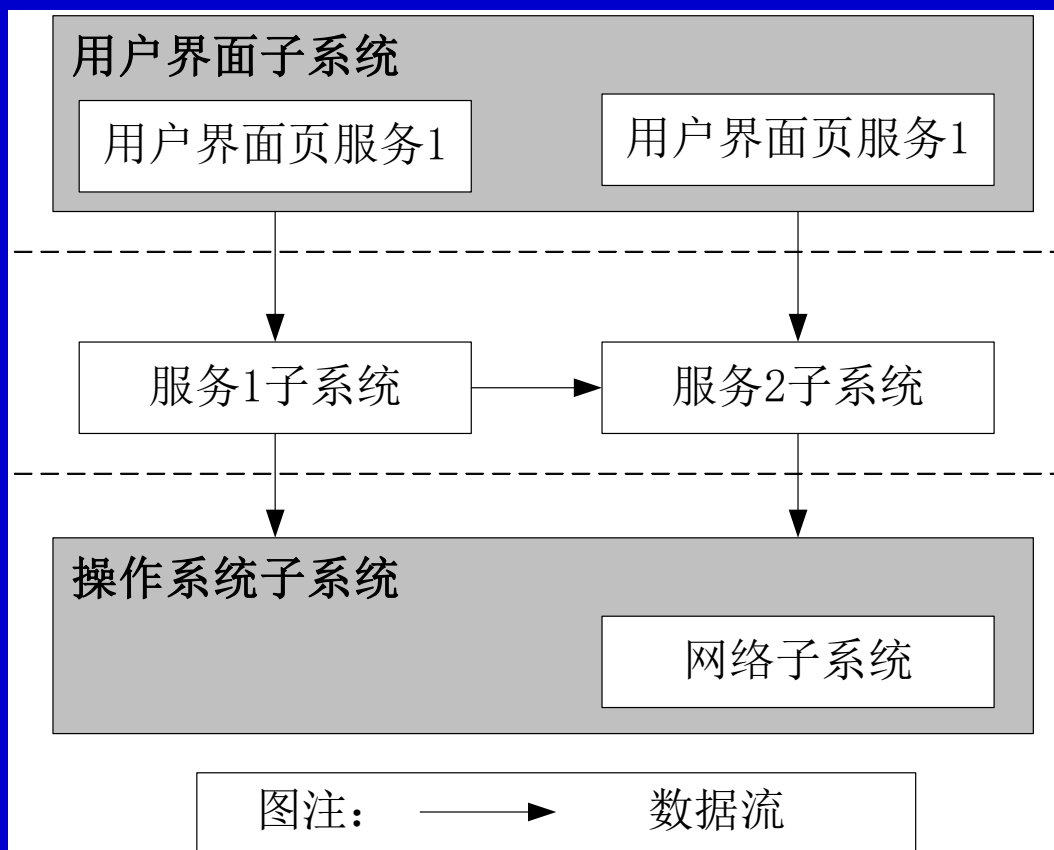
### ◇ ABSD方法的步骤

#### 定义逻辑视图



### ◇ ABSD方法的步骤

#### 某系统的逻辑视图



### ◇ 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。这种分解的标准取决于对一个特定的设计元素来说是很重要的性能。在不同的性能基础上，可以进行多重分解。

如果象通常的产品一样，在分解中起关键作用的性能要求是可修改的，则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。

### ◇ 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。

一旦选定了一个主要的体系结构风格，该风格必须适应基于属于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

#### (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。模板包括了什么是一个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

用例用来验证他们通过有目的的结构能够达到。子设计元素的附加功能将可能通过用例的使用得到判断。然而，如果用例被广泛地使用于功能分解的过程中，将几乎不能发现附加功能。

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

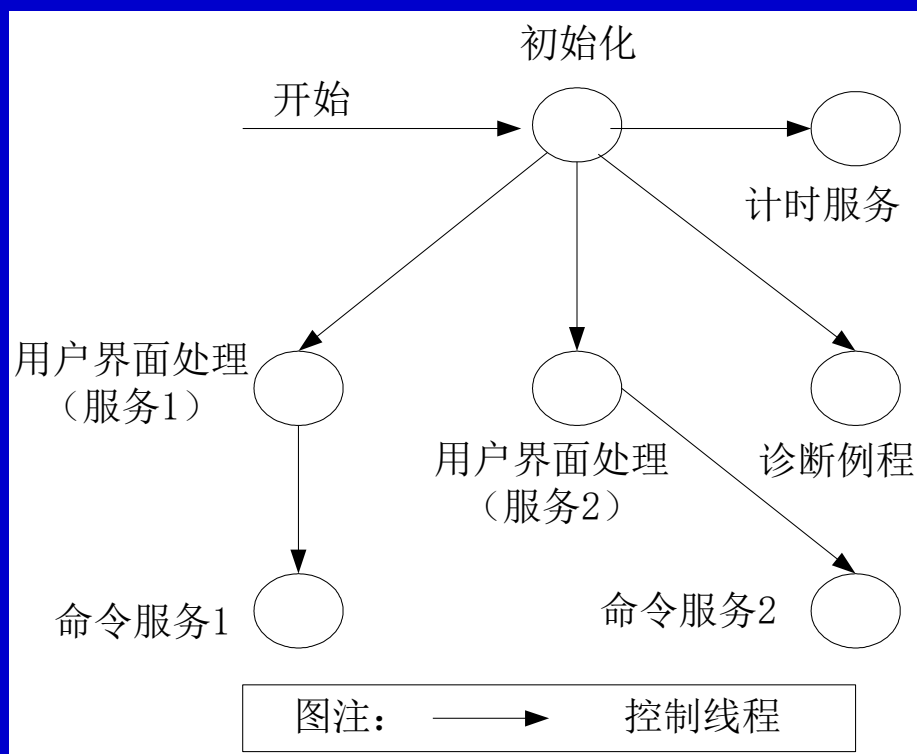
检查并发视图的目的是判断哪些活动是可以并发执行的。这些活动必须得到识别，产生进程同步和资源竞争。

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。虚拟进程与操作系统的进程概念不一样，操作系统的进程包括了额外的地址空间的分配和调度策略。一个操作系统进程是几个虚拟进程的连接点，但每个虚拟进程不一定是操作系统进程。虚拟进程用来描述活动序列，使同步或资源竞争可以在多个虚拟进程之间进行。



### ◇ ABSD方法的步骤

#### (6) 创建并发视图



### ◇ ABSD方法的步骤

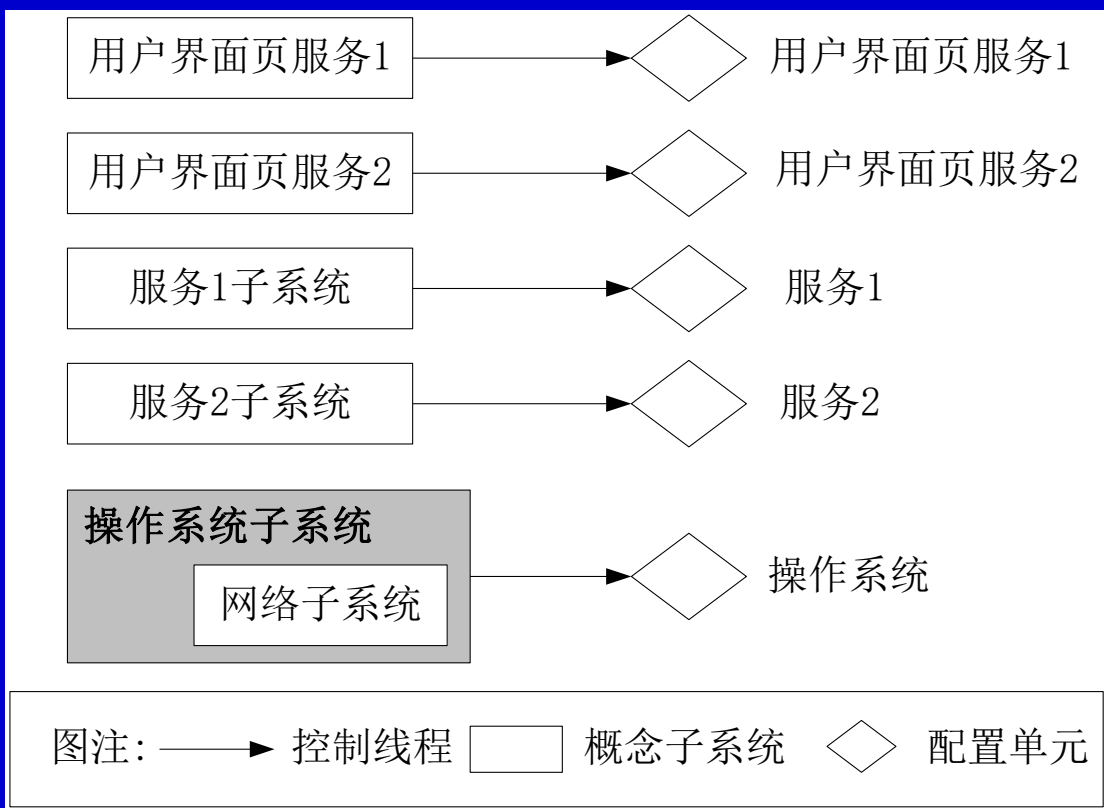
#### (7) 创建配置视图

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



### ◇ 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。

### ◇ 设计和演化过程

◎ 实验原型阶段

◎ 演化开发阶段

#### ◇ 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了两个小组，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**。两个小组要并行地工作，尽量不要发生相互牵制的现象。

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，该模型覆盖了问题域的子集。用户界面设计由**水平原型**表示，也就是说，运行的程序只是实现一些用户界面控制，没有实现真正的系统功能。

### ◇ 实验原型阶段

#### 第二个开发周期

- ◎ 标识构件
- ◎ 提出软件体系结构模型
- ◎ 把已标识的构件映射到软件体系结构中
- ◎ 分析构件之间的相互作用
- ◎ 产生软件体系结构
- ◎ 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。

此时，虽然这个模型是理想化的（其中的某些部分可能错误地表示了应用的特征），但是，该模型为将来的调整和演化过程建立了目标。

### ◇ 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。

### ◇ 演化开发阶段

- ◎ 需求变动归类
- ◎ 制订体系结构演化计划
- ◎ 修改、增加或删除构件
- ◎ 更新构件的相互作用
- ◎ 产生演化后的体系结构
- ◎ 迭代
- ◎ 对以上步骤进行确认，进行阶段性技术评审
- ◎ 对所做的标记进行处理

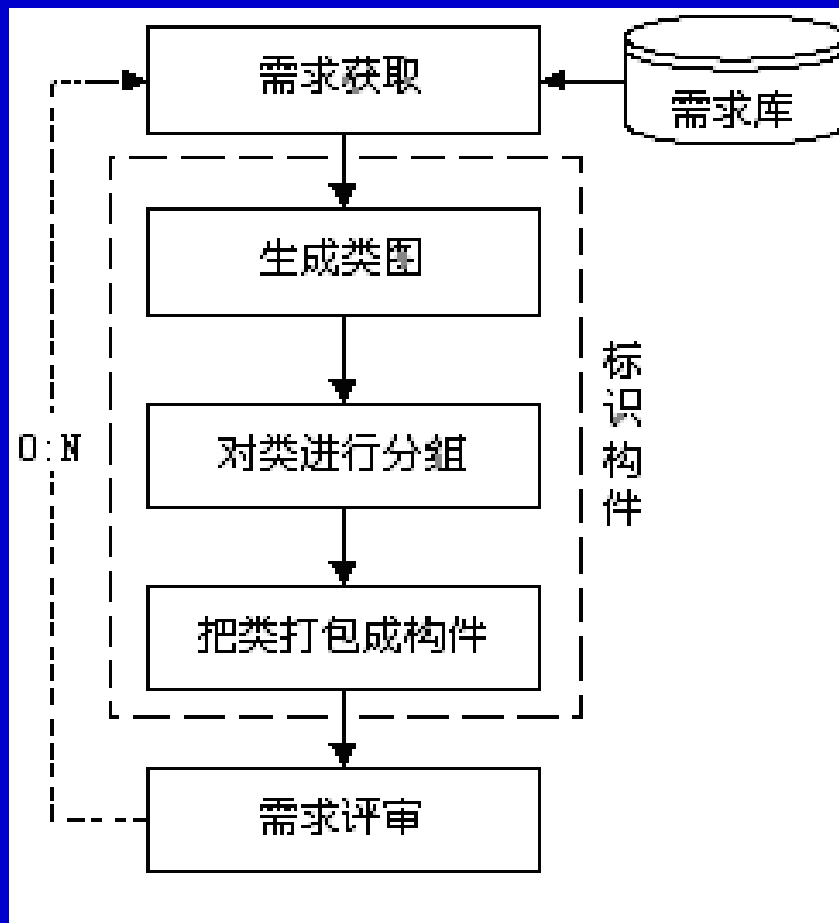
## 7.4 基于体系结构的软件开发模型

```
graph TD; A[体系结构需求] --> B[体系结构设计]; B --> C[体系结构文档化]; C --> D[体系结构复审]; D --> E[体系结构实现]; E --> F[体系结构演化]; F -. Feedback .-> A; D -. Feedback .-> B;
```

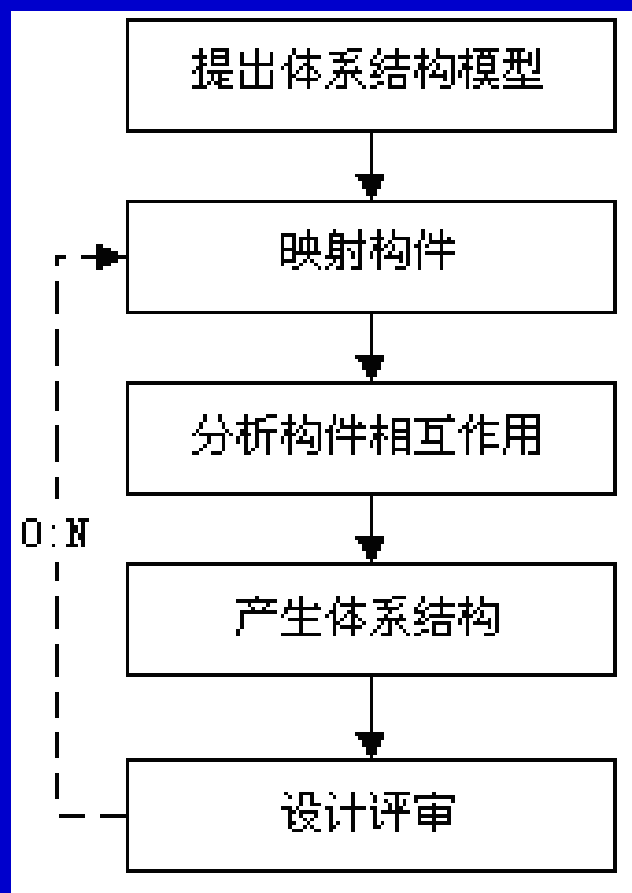
田 冬



### ◇ 体系结构需求



### ◇ 体系结构设计



#### ◇ 体系结构文档化

文档是在系统演化的每一个阶段，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。

软件体系结构的文档要求与软件开发项目中的其他文档是类似的。文档的完整性和质量是软件体系结构成功的关键因素。文档要从使用者的角度进行编写，必须分发给所有与系统有关的开发人员，且必须保证开发者手上的文档是最新的。

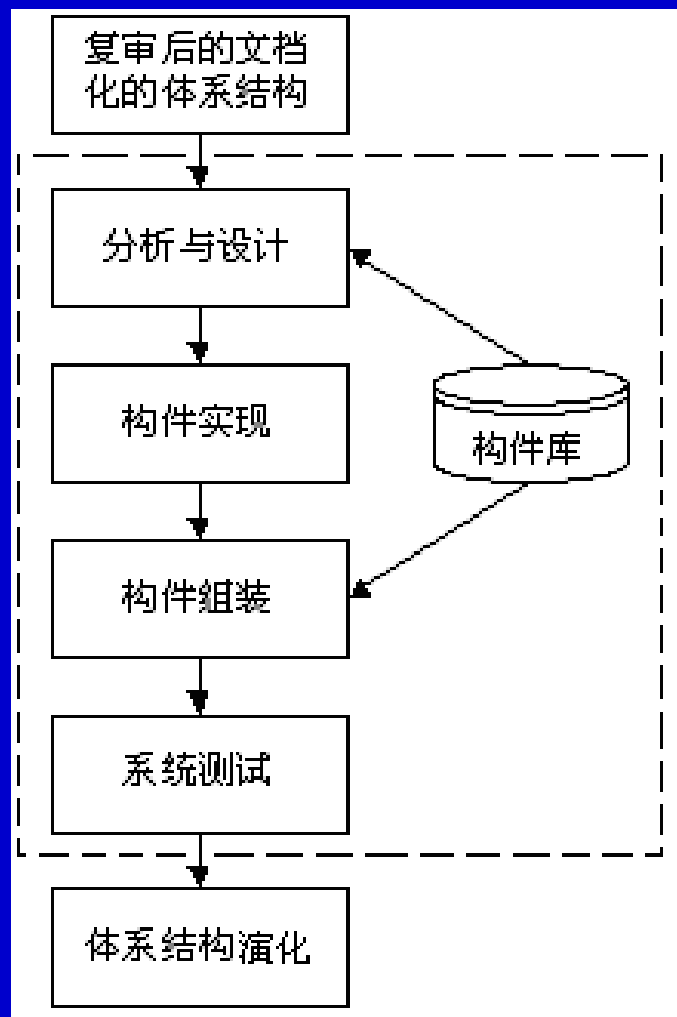
#### ◇ 体系结构复审

体系结构设计、文档化和复审是一个迭代过程。从这个方面来说，在一个主版本的软件体系结构分析之后，要安排一次由外部人员（用户代表和领域专家）参加的复审。

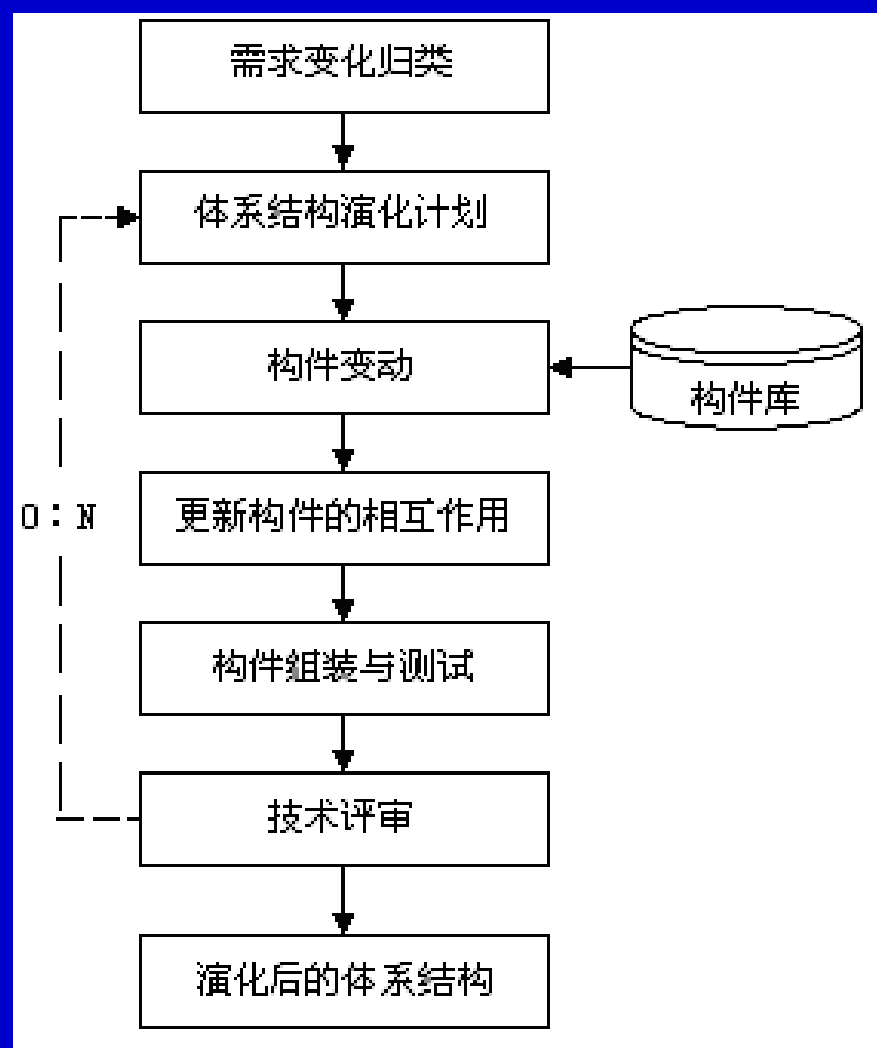
复审的目的是标识潜在的风险，及早发现体系结构设计中的缺陷和错误，包括体系结构能否满足需求、质量需求是否在设计中得到体现、层次是否清晰、构件的划分是否合理、文档表达是否明确、构件的设计是否满足功能与性能的要求等等。

由外部人员进行复审的目的是保证体系结构的设计能够公正地进行检验，使组织的管理者能够决定正式实现体系结构。

### ◇ 体系结构实现



### ◇ 体系结构演化



- 1、请把基于体系结构的软件开发模型与其他软件开发模型进行比较。
- 2、请把基于体系结构的软件设计方法与其他软件设计方法进行比较。
- 3、如何才能提高软件系统的可演化性。

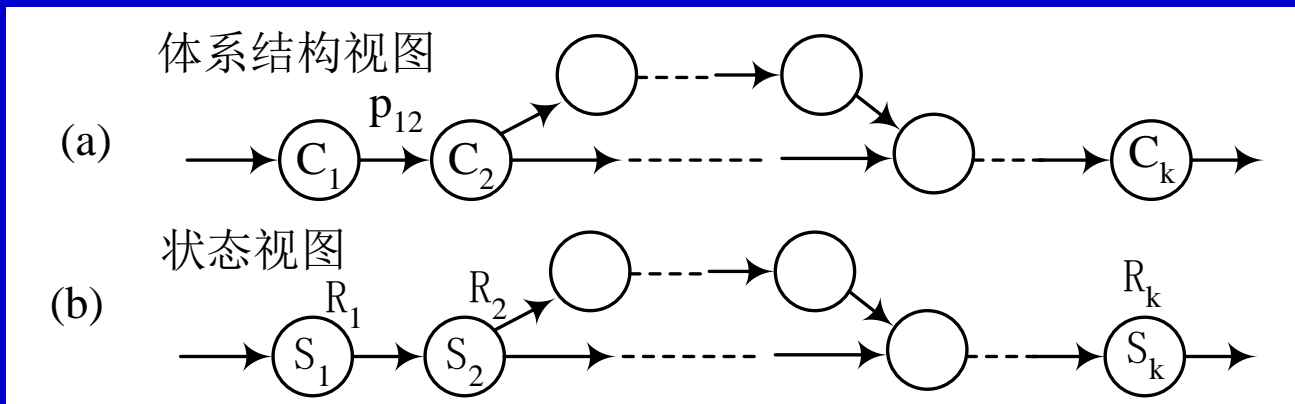
# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构



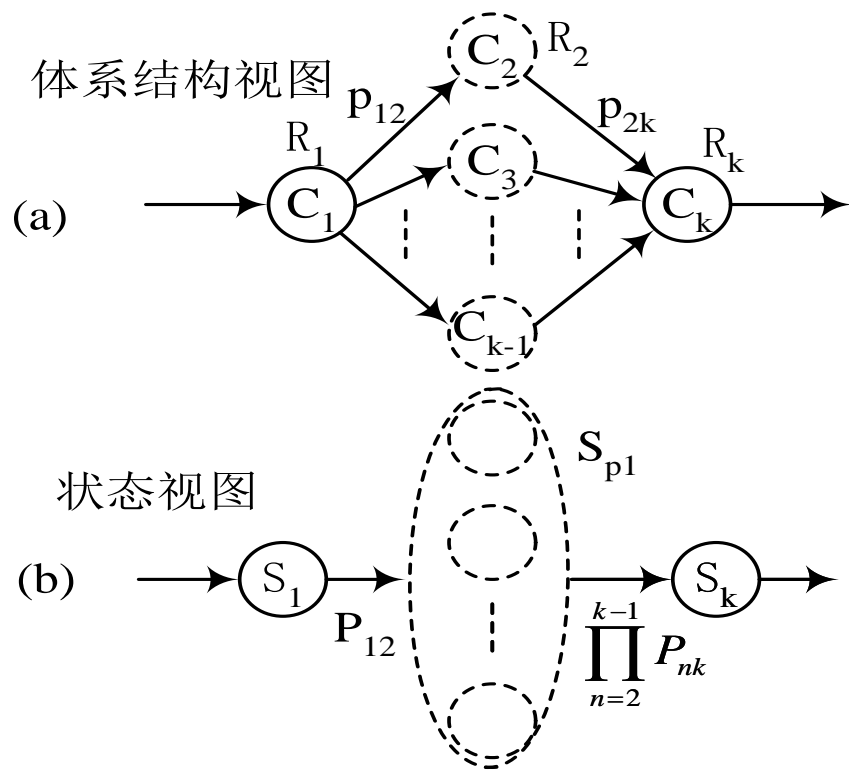
$$M = \begin{matrix} & \begin{matrix} S_1 & S_2 & \cdots & S_i & \cdots & S_{n-1} & S_n \end{matrix} \\ \begin{matrix} S_1 \\ S_2 \\ \vdots \\ S_i \\ \vdots \\ S_{n-1} \\ S_n \end{matrix} & \begin{bmatrix} 0 & R_1 P_{12} & \cdots & R_1 P_{1i} & \cdots & R_1 P_{1n-1} & R_1 P_{1n} \\ R_2 P_{21} & 0 & \cdots & R_2 P_{2i} & \cdots & R_2 P_{2n-1} & R_2 P_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ R_i P_{i1} & R_i P_{i2} & \cdots & 0 & \cdots & R_i P_{in-1} & R_i P_{in} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ R_{n-1} P_{(n-1)1} & R_{n-1} P_{(n-1)2} & \cdots & R_{n-1} P_{(n-1)i} & \cdots & 0 & R_{n-1} P_{(n-1)n} \\ R_n P_{n1} & R_n P_{n2} & \cdots & R_n P_{ni} & \cdots & R_n P_{n(n-1)} & 0 \end{bmatrix} \end{matrix}$$

### ◇ 顺序结构风格



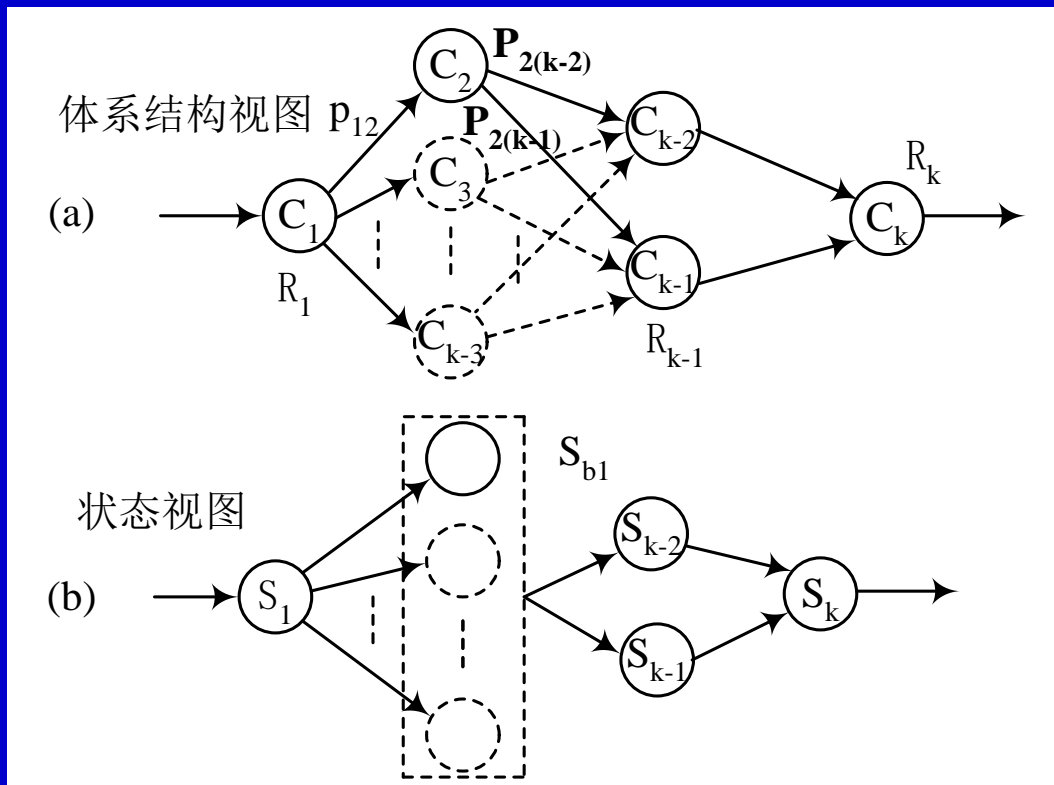
$$\begin{cases} M(i, j) = 0, S_i \text{ 不能够到达 } S_j \\ M(i, j) = r_i p_{ij}, S_i \text{ 能够到达 } S_j \end{cases}, \text{ for } 1 \leq i, j \leq k$$

### ◇ 并行/管道-过滤器结构风格



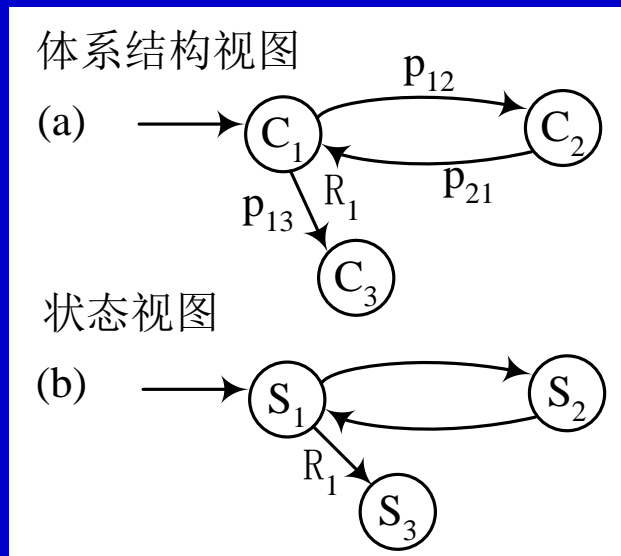
$$\begin{cases} M(i, j) = R_i P_{ij}, S_i \notin S_p \\ M(i, j) = \prod_{C_n \text{ in } S_i} R_n P_{nj}, S_i \in S_p, \text{ for } 1 \leq i, j \leq |S| \text{ and } 1 \leq n \leq k \\ M(i, j) = 0, S_i \text{ 不能到达 } S_j \end{cases}$$

### ◇ 容错结构风格



$$\begin{cases} M(i, j) = R_i P_{ij}, S_i \notin S_b \\ M(i, j) = R_{a1} + \sum_{q=a2}^{ar} \left( \left( \prod_{m=a1}^{q-1} (1 - R_m) \right) R_n \right), S_i \in S_b \text{ 并且 } S_i \text{ 包括 } C_{a1} \cdot C_{ar} \\ M(i, j) = 0, S_i \text{ 不能到达 } S_j, 1 \leq i, j \leq |S|, 1 \leq a_r \leq k \end{cases}$$

### ◇ 调用-返回结构风格



$$\begin{cases} M(i, j) = R_i P_{ij}, S_i \text{ 可以到达 } S_j \\ M(i, j) = P_{ij}, S_i \text{ 可以到达 } S_j, S_j \text{ 是被调用构件}, 1 \leq i, j \leq k \\ M(i, j) = 0, S_i \text{ 不能到达 } S_j \end{cases}$$

#### ◇ 系统的可靠性模型化步骤

- ◎ 通过系统的详细说明书，确定系统所采用的体系结构风格。
- ◎ 把每一种体系结构风格转换成状态视图，并计算状态视图中每一个状态的可靠性及其相应的迁移概率。
- ◎ 通过整个系统的体系结构视图，把所有的状态视图集成为一个整体状态视图。
- ◎ 通过整体状态视图构造系统的迁移矩阵，并计算系统的可靠性。

#### ◇ 软件体系结构风险分析背景

◎ 动态方法

◎ 构件依赖图

#### ◇ 软件体系结构风险分析方法

- ◎ 采用体系结构描述语言ADL对体系结构进行建模
- ◎ 通过模拟方法执行复杂性分析
- ◎ 通过FMEA和模拟运行执行严重性分析
- ◎ 为构件和连接件开发其启发式风险因子
- ◎ 建立用于风险评估的CDG
- ◎ 通过图论中的算法执行风险评估和分析

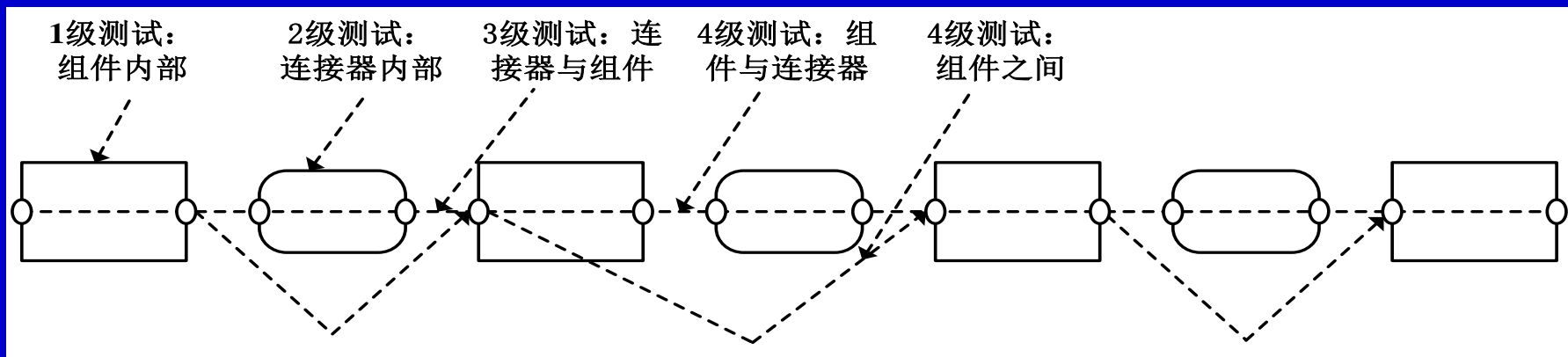


### ◇ 测试方法

◎ 测试内容

◎ 测试准则

◎ 测试需求和测试用例的生成



#### ◇ 实例与实现

自学

- 1、什么是软件体系结构的可靠性？为什么要研究软件体系结构的可靠性？
- 2、如何模型化系统的可靠性？
- 3、软件体系结构风险分析有哪些基本步骤？
- 4、软件体系结构测试与程序测试的主要区别是什么？

# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

#### ◇ 性能

性能是指系统的响应能力，即要经过多长时间才能对某个事件做出响应，或者在某段事件内系统所能处理的事件的个数。

经常用单位<sup>时间</sup>~~事件~~内所处理事务的数量或系统完成某个事务处理所需的时间来对性能进行定量的表示。

性能测试经常要使用基准测试程序（用以测量性能指标的特定事务集或工作量环境）。

#### ◇ 可靠性(1)

可靠性是软件系统在面对应用或系统错误时，在意外或错误使用的情况下维持软件系统的功能特性的基本能力。

可靠性通常用平均失效等待时间（MTTF）和平均失效间隔时间（MTBF）来衡量。在失效率为常数和修复时间很短的情况下，MTTF和MTBF几乎相等。

### ◇ 可靠性(2)

◎ 容错

◎ 健壮性

#### ◇ 可用性

可用性是系统能够正常运行的时间比例。经常用两次故障之间的时间长度或在出现故障时系统能够恢复正常的速度来表示。



#### ◇ 安全性

安全性是指系统在向合法用户提供服务的同时能够阻止非授权用户使用的企图或拒绝服务的能力。安全性是根据系统可能受到的安全威胁的类型来分类的。

安全性又可划分为机密性、完整性、不可否认性及可控性等特性。其中，机密性保证信息不泄露给未授权的用户、实体或过程；完整性保证信息的完整和准确，防止信息被非法修改；可控性保证对信息的传播及内容具有控制的能力，防止为非法者所用。

### ◇ 可修改性

- ◎ 可维护性
- ◎ 可扩展性
- ◎ 结构重组
- ◎ 可移植性

#### ◇ 功能性

功能性是系统所能完成所期望的工作的能力。一项任务的完成需要系统中许多或大多数构件的相互协作。

#### ◇ 可变性

可变性是指体系结构经扩充或变更而成为新体系结构的能力。这种新体系结构应该符合预先定义的规则，在某些具体方面不同于原有的体系结构。当要将某个体系结构作为一系列相关产品（例如，软件产品线）的基础时，可变性是很重要的。

#### ◇ 集成性

可集成性是指系统能与其他系统协作的程度。

#### ◇ 互操作性

作为系统组成部分的软件不是独立存在的，经常与其他系统或自身环境相互作用。为了支持互操作性，软件体系结构必须为外部可视的功能特性和数据结构提供精心设计的软件入口。程序和用其他编程语言编写的软件系统的交互作用就是互操作性的问题，这种互操作性也影响应用的软件体系结构。

### ◇ 基本概念

#### (1) 敏感点和权衡点

敏感点是一个或多个构件（和/或构件之间的关系）的特性。研究敏感点可使设计人员或分析员明确在搞清楚如何实现质量目标时应注意什么。

权衡点是影响多个质量属性的特性，是多个质量属性的敏感点。例如，改变加密级别可能会对安全性和性能产生非常重要的影响。提高加密级别可以提高安全性，但可能要耗费更多的处理时间，影响系统性能。如果某个机密消息的处理有严格的时间延迟要求，则加密级别可能就会成为一个权衡点。

### ◇ 基本概念

#### (2) 风险承担者

系统的体系结构涉及到很多人的利益，这些人都对体系结构施加各种影响，以保证自己的目标能够实现。



### ◇ 基本概念

#### (3) 场景(1)

在进行体系结构评估时，一般首先要精确地得出具体的质量目标，并以之作为判定该体系结构优劣的标准。我们把为得出这些目标而采用的机制叫做场景。场景是从风险承担者的角度对与系统的交互的简短描述。在体系结构评估中，一般采用刺激、环境和响应三方面来对场景进行描述。

### ◇ 基本概念

#### (3) 场景(2)

刺激是场景中解释或描述风险承担者怎样引发与系统的交互部分。例如，用户可能会激发某个功能，维护人员可能会做某个更改，测试人员可能会执行某种测试等，这些都属于对场景的刺激。

环境描述的是刺激发生时的情况。例如，当前系统处于什么状态？有什么特殊的约束条件？系统的负载是否很大？某个网络通道是否出现了阻塞等。

响应是指系统是如何通过体系结构对刺激作出反应的。例如，用户所要求的功能是否得到满足？维护人员的修改是否成功？测试人员的测试是否成功等。

#### ◇ 主要的评估方式

- ◇ 基于调查问卷或检查表的评估方式
- ◇ 基于场景的评估方式
- ◇ 基于度量的评估方式

#### ◇ 基于调查问卷或检查表的评估方式(1)

CMU/SEI的软件风险评估过程采用了这一方式。

调查问卷是一系列可以应用到各种体系结构评估的相关问题，其中有些问题可能涉及到体系结构的设计决策；有些问题涉及到体系结构的文档，有的问题针对体系结构描述本身的细节问题。

检查表中也包含一系列比调查问卷更细节和具体的问题，它们更趋向于考察某些关心的质量属性。

#### ◇ 基于调查问卷或检查表的评估方式(2)

这一评估方式比较自由灵活，可评估多种质量属性，也可以在软件体系结构设计的多个阶段进行。但是由于评估的结果很大程度上来自评估人员的主观推断，因此不同的评估人员可能会产生不同甚至截然相反的结果，而且评估人员对领域的熟悉程度、是否具有丰富的相关经验也成为评估结果是否正确的重要因素。

尽管基于调查问卷与检查表的评估方式相对比较主观，但由于系统相关的人员的经验和知识是评估软件体系结构的重要信息来源，因而它仍然是进行软件体系结构评估的重要途径之一。

#### ◇ 基于场景的评估方式(1)

基于场景的方式由SEI首先提出并应用在体系结构权衡分析方法(ATAM)和软件体系结构分析方法(SAAM)中。

这种软件体系结构评估方式分析软件体系结构对场景也就是对系统的使用或修改活动的支持程度,从而判断该体系结构对这一场景所代表的质量需求的满足程度。例如,用一系列对软件的修改来反映易修改性方面的需求,用一系列攻击性操作来代表安全性方面的需求等。

#### ◇ 基于场景的评估方式(2)

这一评估方式考虑到了包括系统的开发人员、维护人员、最终用户、管理人员、测试人员等在内的所有与系统相关的人员对质量的要求。基于场景的评估方式涉及到的基本活动包括确定应用领域的功能和软件体系结构的结构之间的映射，设计用于体现待评估质量属性的场景以及分析软件体系结构对场景的支持程度。

不同的应用系统对同一质量属性的理解可能不同，例如，对操作系统来说，可移植性被理解为系统可在不同的硬件平台上运行，而对于普通的应用系统而言，可移植性往往是指该系统可在不同的操作系统上运行。由于存在这种不一致性，对一个领域适合的场景设计在另一个领域内未必合适，因此基于场景的评估方式是特定于领域的。这一评估方式的实施者一方面需要有丰富的领域知识以对某以质量需求设计出合理的场景，另一方面，必须对待评估的软件体系结构有一定的了解以准确判断它是否支持场景描述的一系列活动。

#### ◇ 基于度量的评估方式(1)

度量是指为软件产品的某一属性所赋予的数值，如代码行数、方法调用层数、构件个数等。传统的度量研究主要针对代码，但近年来也出现了一些针对高层设计的度量，软件体系结构度量即是其中之一。代码度量和代码质量之间存在着重要的联系，类似地，软件体系结构度量应该也能够作为评判质量的重要依据。

赫尔辛基大学提出的基于模式挖掘的面向对象软件体系结构度量技术、Karlskrona和Ronneby提出的基于面向对象度量的软件体系结构可维护性评估、西弗吉尼亚大学提出的软件体系结构度量方法等都在这方面进行了探索，提出了一些可操作的具体方案。我们把这类评估方式称作基于度量的评估方式。



#### ◇ 基于度量的评估方式(2)

基于度量的评估技术都涉及三个基本活动：首先需要建立质量属性和度量之间的映射原则，即确定怎样从度量结果推出系统具有什么样的质量属性；然后从软件体系结构文档中获取度量信息；最后根据映射原则分析推导出系统的某些质量属性。因此，这些评估技术被认为都采用了基于度量的评估方式。

基于度量的评估方式提供更为客观和量化的质量评估。这一评估方式需要在软件体系结构的设计基本完成以后才能进行，而且需要评估人员对待评估的体系结构十分了解，否则不能获取准确的度量。自动的软件体系结构度量获取工具能在一定程度上简化评估的难度，例如MAISA可从文本格式的UML图中抽取面向对象体系结构的度量。

◇ 三种评估方式的比较

评估方式	调查问卷或检查表		场景	度量
	调查问卷	检查表		
通用性	通用	特定领域	特定系统	通用或特定领域
评估者对体系结构的了解程度	粗略了解	无限制	中等了解	精确了解
实施阶段	早	中	中	中
客观性	主观	主观	较主观	较客观

#### ◇ ATAM评估的步骤

整个ATAM评估过程包括九个步骤，按其编号顺序分别是描述ATAM方法、描述商业动机、描述体系结构、确定体系结构方法、生成质量属性效用树、分析体系结构方法、讨论和分级场景、分析体系结构方法（是第六步的重复）、描述评估结果。

#### ◇ 描述ATAM方法

ATAM评估的第一步要求评估小组负责人向参加会议的风险承担者介绍ATAM评估方法。在这一步，要解释每个人将要参与的过程，并预留出解答疑问的时间，设置好其他活动的环境和预期结果。关键是要使每个人都知道要收集哪些信息，如何描述这些信息，将要向谁报告等。特别是要描述以下事项：

- (1) ATAM方法步骤简介；
- (2) 获取和分析技术：效用树的生成，基于体系结构方法的获取/分析，场景的映射等；
- (3) 评估结果：所得出的场景及其优先级，用户理解/评估体系结构的问题，描述驱动体系结构的需求并对这些需求进行分类，所确定的一组体系结构方法和风格，一组所发现的风险点和无风险点、敏感点和权衡点。

#### ◇ 描述商业动机

参加评估的所有人员必须理解待评估的系统，在这一步，项目经理要从商业角度介绍系统的概况

商业环境/驱动描述（约12张幻灯片，45分钟）

（1）描述商业环境、历史、市场划分、驱动需求、风险承担者、当前需要以及系统如何满足这些需要（3-4张幻灯片）。

（2）描述商业方面的约束条件（例如：推向市场的时间、客户需求、标准和成本等）（1-3张幻灯片）。

（3）描述技术方面的约束条件（例如：COTS、与其他系统的互操作、所需要的软硬件平台、遗留代码的重用等）（1-3张幻灯片）。

（4）质量属性需求（例如：系统平台、可用性、安全性、可修改性、互操作性、集成性和这些需求来自的商业需要）（2-3张幻灯片）。

（5）术语表（1张幻灯片）。

#### ◇ 描述体系结构(1)

在这一步中，首席设计师或设计小组要对体系结构进行详略适当的介绍，这里的“详略适当”取决于多个因素，例如有多少信息已经决定了下来，并形成了文档；可用时间是多少；系统面临的风险有哪些等。这一步很重要，将直接影响到可能要做的分析及分析的质量。在进行更详细的分析之前，评估小组通常需要收集和记录一些额外的体系结构信息。

#### ◇ 描述体系结构(2)

体系结构描述（约20张幻灯片，60分钟）

（1）驱动体系结构的需求（例如：性能、可用性、安全性、可修改性、互操作性、集成性等），以及与这些需求相关的可度量的量和满足这些需求的任何存在的标准、模型或方法（2-3张幻灯片）。

（2）高层体系结构视图（4-8张幻灯片）。

- ① 功能：函数、关键的系统抽象、领域元素及其依赖关系、数据流；
- ② 模块/层/子系统：描述系统功能组成的子系统、层、模块，以及对象、过程、函数及它们之间的关系（例如：过程调用、方法使用、回调和包含等）；
- ③ 进程/线程：进程、线程及其同步，数据流和与之相连的事件；
- ④ 硬件：CPU、存储器、外设/传感器，以及连接这些硬件的网络和通信设备。

#### ◇ 描述体系结构(3)

(3) 所采用的体系结构方法或风格，包括它们所强调的质量属性和如何实现的描述（3-6张幻灯片）。

(4) COTS的使用，以及如何选择和集成（1-2张幻灯片）。

(5) 介绍1-3个最重要的用例场景，如果可能，应包括对每个场景的运行资源的介绍（1-3张幻灯片）。

(6) 介绍1-3个最重要的变更场景，如果可能，应描述通过变更构件、连接件或接口所带来的影响（1-3张幻灯片）。

(7) 与满足驱动体系结构需求相关的体系结构问题或风险（2-3张幻灯片）。

(8) 术语表（1张幻灯片）。

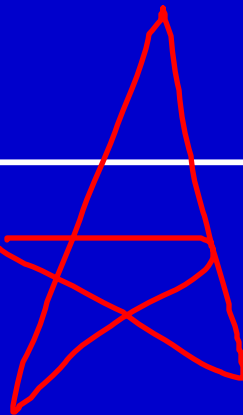


#### ◇ 确定体系结构方法

ATAM评估方法主要通过理解体系结构方法来分析体系结构，在这一步，由设计师确定体系结构方法，由分析小组捕获，但不进行分析。

ATAM评估方法之所以强调体系结构方法和体系结构风格的确定，是因为这些内容代表了实现最高优先级的质量属性的体系结构手段。也就是说，它们是保证关键需求按计划得以实现的手段。这些体系结构方法定义了系统的重要结构，描述了系统得以扩展的途径，对变更的响应，对攻击的防范以及与其他系统的集成等。

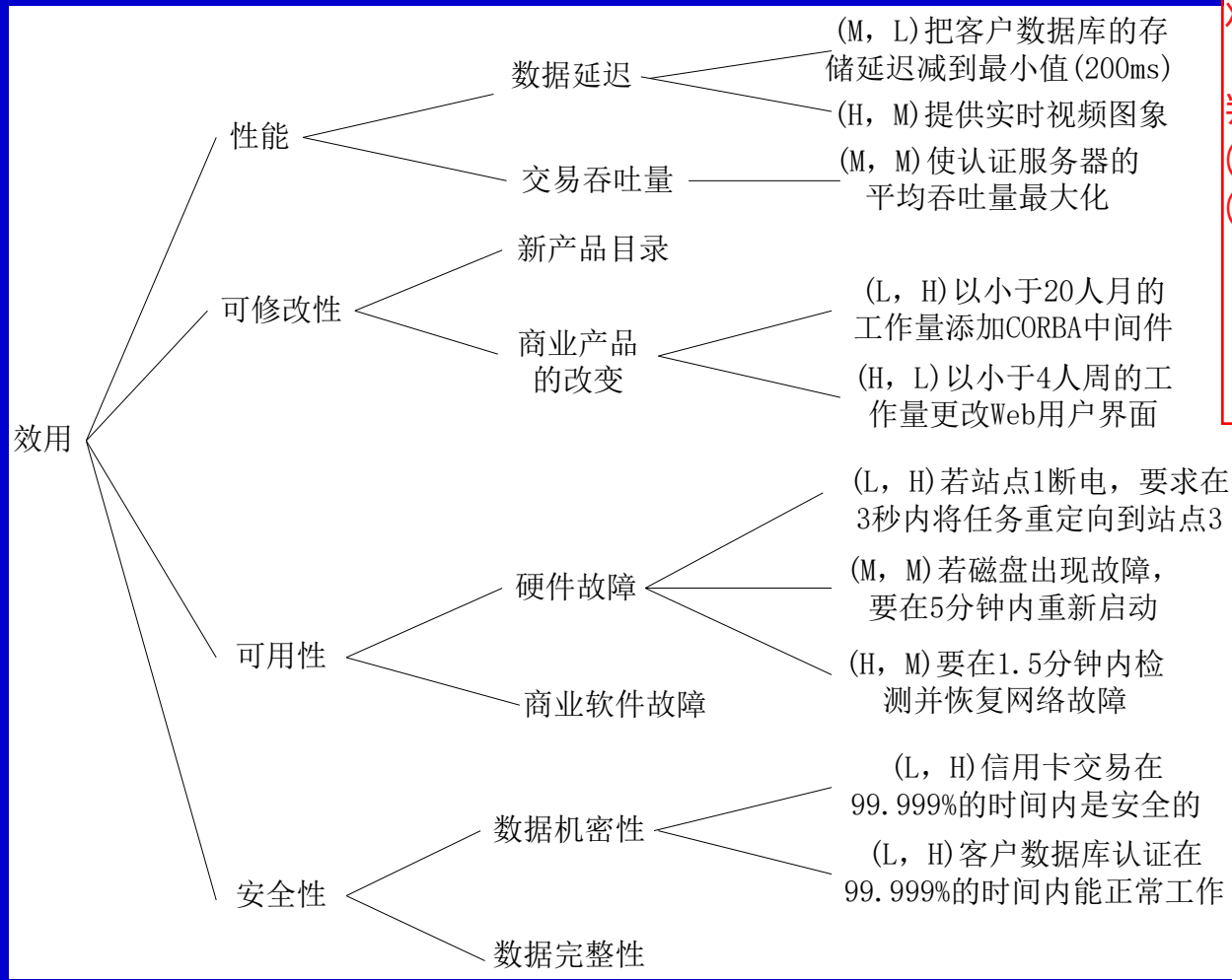
#### ◇ 生成质量属性效用树(1)



评估小组、设计小组、管理人员和客户代表一起确定系统最重要的质量属性目标，并对这些质量目标设置优先级和细化。这一步很关键，它对以后的分析工作起指导作用。即使是体系结构级的分析，也并不一定是全局的，所以，评估人员需要集中所有相关人员的精力，注意体系结构的各个方面，这对系统的成败起关键作用。这通常是通过构建效用树的方式来实现的。

效用树的输出结果是对具体质量属性需求（以场景形式出现）的优先级的确定，这种优先级列表为ATAM评估方法的后面几步提供了指导，它告诉了评估小组该把有限的时间花在哪里，特别是该在哪里去考察体系结构方法与相应的风险、敏感点和权衡

### ◇ 生成质量属性效用树 (2)



M：每个场景对系统成功与否的重要性，L：体系结构设计人员估计的实现难易程度

判定方法：  
(H,H)>(H,M)>(M,H)>(M,M)

#### ◇ 分析体系结构方法(1)

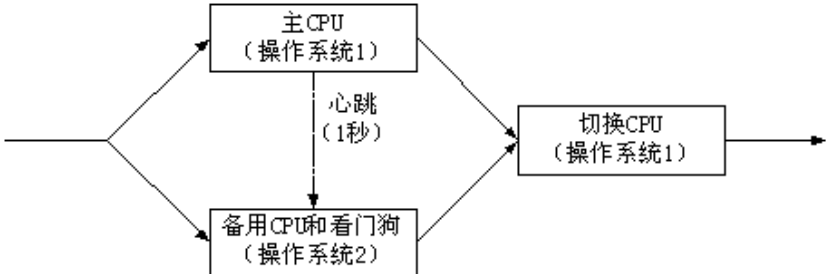
一旦有了效用树的结果，评估小组可以对实现重要质量属性的体系结构方法进行考察。这是通过注意文档化这些体系结构决策和确定它们的风险、敏感点和权衡点等来实现的。

在这一步中，评估小组要对每一种体系结构方法都考察足够的信息，完成与该方法有关的质量属性的初步分析。这一步的主要结果是一个体系结构方法或风格的列表，与之相关的一些问题，以及设计师对这些问题的回答。通常产生一个风险列表、敏感点和权衡点列表。

◇ 分析体系结构方法(2)

场景：〈来自效用树的一个场景〉			
属性：〈性能、安全性、可用性〉			
环境：〈对系统所依赖的环境的相关假设〉			
刺激：〈对该场景体现的质量属性刺激（例如：故障、安全威胁、修改等）的精确描述〉			
响应：〈对质量属性响应的精确叙述（例如：响应时间、修改难度等）〉			
体系结构决策	风险	敏感度	权衡
影响质量属性响应的 体系结构决策列表	风险列表	敏感点编号	权衡点编号
.....	.....	.....	.....
.....	.....	.....	.....
推理： 〈关于为什么这组体系结构决策能够满足质量属性响应需求的定性或定量的推理〉			
体系结构图： 〈一个或多个体系结构图，在图中标注出上述推理的体系结构信息，可带解释性文字描述〉			

### ◇ 分析体系结构方法(3)

场景: S12 (检测主 CPU 故障并恢复系统)			
属性: 可用性			
环境: 正常操作			
刺激: CPU 失效			
响应: 可用切换概率为 0.999999			
体系结构决策	风险	敏感度	权衡
备用 CPU	R8	S2	T3
无备用数据通道	R9	S3	
看门狗 (watchdog)		S4	
心跳 (heartbeat)		S5	
故障切换路由		S6	
<b>推理:</b> <ul style="list-style-type: none"><li>(1) 通过使用不同的硬件和操作系统, 保证没有通用模式故障;</li><li>(2) 完成恢复时间最多不超过 4 秒 (也就是一个运算状态时间);</li><li>(3) 基于心跳和看门狗的速度, 保证在 2 秒钟内能检测到故障;</li><li>(4) 看门狗简单的和可靠的 (已被证实过);</li><li>(5) 由于没有备用数据通道, 可用性需求可能存在风险</li></ul>			
<b>体系结构图:</b> 			

#### ◇ 讨论和分级场景(1)

风险承担者需进行两项相关的活动：集体讨论用例场景（描述风险承担者期望使用系统的方式）和改变场景（描述风险承担者所期望的系统在将来变更的方式）。用例场景是场景的一种，在用例场景中，风险承担者是一个终端用户，使用系统执行一些功能。改变场景代表系统的变更，可分为成长场景和考察场景两类。

成长场景描述的是体系结构在中短期的改变，包括期望的修改、性能或可用性的变更、移植性、与其他软件系统的集成等。考察场景描述的是系统成长的一个极端情形，即体系结构由下列情况所引起的改变：根本性的性能或可用性需求（例如数量级的改变）、系统基础结构或任务的重大变更等。成长场景能够使评估人员看清在预期因素影响系统时，体系结构所表现出来的优缺点，而考察场景则试图找出敏感点和权衡点，这些点的确定有助于评估者评估系统质量属性的限制。

#### ◇ 讨论和分级场景(2)

一旦收集了若干个场景后，必须要设置优先级。评估人员可通过投票表决的方式来完成，每个风险承担者分配相当于总场景数的30%的选票，且此数值只入不舍。例如，如果共有17个场景，则每个风险承担者将拿到6张选票，这6张选票的具体使用则取决于风险承担者，他可以把这6张票全部投给某一个场景，或者每个场景投2-3张票，还可以一个场景一张票等。

一旦投票结果确定，所有场景就可设置优先级。设置优先级和投票的过程既可公开也可保密。



◇ 讨论和分级场景 (3)

场景编号	场景描述	得票数量
4	在 10 分钟内动态地对某次任务得重新安排	28
27	把对一组车辆得管理分配给多个控制站点	26
10	在不重新启动系统的情况下，改变已开始任务的分析工具	23
12	在发出指令后 10 秒内，完成对不同车辆的重新分配，以处理紧急情况	13
14	在 6 人月内将数据分配机制从 CORBA 改变为新兴的标准	12

◇ 讨论和分级场景 (4)

场景编号	得票数量	质量属性
4	28	性能
27	26	性能、可修改性、可用性
10	23	可修改性
12	13	性能
14	12	可修改性

#### ◇ 分析体系结构方法

在收集并分析了场景之后，设计师就可把最高级别的场景映射到所描述的体系结构中，并对相关的体系结构如何有助于该场景的实现做出解释。

在这一步中，评估小组要重复第6步中的工作，把新得到的最高优先级场景与尚未得到的体系结构工作产品对应起来。在第7步中，如果未产生任何在以前的分析步骤中都没有发现的高优先级场景，则在第8步就是测试步骤。

#### ◇ 描述评估结果

最后，要把ATAM分析中所得到的各种信息进行归纳，并反馈给风险承担者。这种描述一般要采用辅以幻灯片的形式，但也可以在ATAM评估结束之后，提交更完整的书面报告。

在描述过程中，评估负责人要介绍ATAM评估的各个步骤，以及各步骤中得到的各种信息，包括商业环境、驱动需求、约束条件和体系结构等。最重要的是要介绍ATAM评估的结果：

- (1) 已文档化了的体系结构方法/风格；
- (2) 场景及优先级；
- (3) 基于属性的问题；
- (4) 效用树；
- (5) 所发现的风险决策；
- (6) 已文档化了的无风险决策；
- (7) 所发现的敏感点和权衡点。

#### ◇ ATAM评估的阶段

第一个阶段以体系结构为中心，重点是获取体系结构信息并进行分析。第二个阶段以风险承担者为中心，重点是获取风险承担者的观点，验证第一个阶段的结果。

之所以要分为两个阶段，是因为评估人员要在第一个阶段收集信息。在整个ATAM评估过程中，评估小组中的部分人（通常是1-3人）要与体系结构设计师和1-2个其他关键的风险承担者（例如，项目经理，客户经理，市场代表）一起工作，收集信息。对支持分析而言，在大多数情况下，这种信息是不完整的或不适当的，所以，评估小组必须与体系结构设计师一起协作引导出必须的信息，这种协作通常要花几周的时间。当评估人员觉得已经收集了足够的信息，并已把这些信息记录成文档，则就可进入第二个阶段了。

#### ◇ 第一阶段(1)

ATAM评估小组要与提交待评估的体系结构的小组见面（或许这是双方第一次会见），这一会议有两方面的目的，一是组织和安排以后的工作，二是收集相关信息。从组织角度来看，体系结构小组负责人要保证让合适的人选参加后续会议，还要保证这些人为参加相关会议做了充分的准备，抱着正确的态度。

第一天通常作为整个ATAM过程的一个缩影，主要关注1-6步的工作。第一次会议所收集的信息意味着要保证体系结构能得到正确的评估。同时，在第一次会议也会收集和分析一些初步的场景，作为理解体系结构、需要收集和提交的信息、所产生的场景的含义的一种途径。

#### ◇ 第一阶段(2)

例如，在第一天，体系结构设计师可能提交部分体系结构，确定部分体系结构风格或方法，创建初步的效用树，就选定的一组场景进行工作，展示每个场景是如何影响体系结构的（例如可修改性），体系结构又是如何作出响应的（例如：对质量属性而言，可以是性能、安全性和可用性）。其他的风险承担者（例如：关键开发人员、客户、项目经理等）可以描述商业环境、效用树的构建，以及产生场景的过程。

第一个阶段是一个小型会议，评估小组需要尽可能多地收集有关信息，这些信息用来决定：

- （1） 后续评估工作是否可行，能否顺利进行；
- （2） 是否需要更多的体系结构文档。如果需要，则应明确需要哪些类型的文档，如何提交这些文档；
- （3） 哪些风险承担者应参与第二个阶段的工作。

#### ◇ 第一阶段(3)

在这一天的最后，评估人员将对项目的状态和环境、驱动体系结构需求，以及体系结构文档都有较清晰的认识。

在第一次会议和第二次会议之间有一段中断时间，其长短取决于第一个阶段完成的情况。在这段时间内，体系结构设计小组和要评估小组协作，做一些探索和分析工作。前面已经提到过，在第一个阶段中评估小组并不构建详细的分析模型，而是构建一些初步模型，以使评估人员和设计人员能对体系结构有更充分的认识，从而保证第二个阶段的工作更有效率。另外，在这段时间内，还要根据评估工作的需要、可用人员的状况和计划来决定评估小组的最终人选。例如，如果待评估的系统对安全性的要求很高，则需要让安全专家参与评估工作；如果待评估的系统是以数据为中心的，则需要让数据库设计方面的专家参与评估。



#### ◇ 第二阶段

这时，体系结构已经被文档化，且有足够的信息来支持验证已经进行的分析和将要进行的分析。已经确定了参与评估工作的合适的风险承担者，并且给他们提供了一些书面阅读材料，如对ATAM方法的介绍，某些初步的场景，包括体系结构、商业案例和关键需求的系统文档等。这些阅读材料有助于保证风险承担者建立对ATAM评估方法的正确期望。

因为将有更多的风险承担者参与第二次会议，且因为在第一次会议和第二次会议之间，可能还要间隔几天或几个星期，所以第二个阶段首先有必要重新简单介绍ATAM方法，以使所有与会者达成共同的理解。另外，在每一步进行之前，简单扼要地介绍该步的工作，也是很有好处的。

### ◇ ATAM各步骤中相关的风险承担者

步骤编号	所做的工作	风险承担者群体
1	描述 ATAM 方法	评估小组/客户代表/体系结构设计小组
2	描述商业动机	评估小组/客户代表/体系结构设计小组
3	描述体系结构	评估小组/客户代表/体系结构设计小组
4	确定体系结构方法	评估小组/客户代表/体系结构设计小组
5	生成质量属性效用树	评估小组/客户代表/体系结构设计小组
6	分析体系结构方法	评估小组/客户代表/体系结构设计小组
7	讨论和对场景进行分级	所有风险承担者
8	分析体系结构方法	评估小组/客户代表/体系结构设计小组
9	描述评估结果	所有风险承担者

◇ ATAM评估日程安排(1)

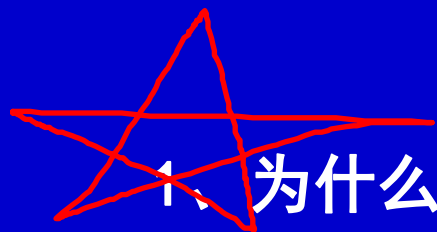
开始时间	所做的工作	
第一天		第一个阶段
8: 30	介绍/描述 ATAM 方法 (第 1 步)	
10: 00	客户描述商业动机 (第 2 步)	
10: 45	休息	
11: 00	客户描述体系结构 (第 3 步)	
12: 00	确定体系结构方法 (第 4 步)	
12: 30	中餐	
13: 45	生成质量属性效用树 (第 5 步)	
14: 45	分析体系结构方法 (第 6 步)	
15: 45	休息	
16: 00	分析体系结构方法 (第 6 步)	
17: 00	休会	
中断几个星期		

◇ ATAM评估日程安排(2)

中断几个星期		
第二天		第二个阶段
8: 30	介绍/描述 ATAM 方法 (第 1 步)	
9: 15	客户描述商业环境/动机 (第 2 步)	
10: 00	休息	
10: 15	客户描述体系结构 (第 3 步)	
11: 15	确定体系结构方法 (第 4 步)	
12: 00	中餐	
13: 00	生成质量属性效用树 (第 5 步)	
14: 00	分析体系结构方法 (第 6 步)	
15: 30	休息	
15: 45	分析体系结构方法 (第 6 步)	
17: 00	休会	
第三天		
8: 30	介绍/扼要重述 ATAM 方法	
8: 45	分析体系结构方法 (第 6 步)	
9: 30	讨论场景 (第 7 步)	
10: 30	休息	
10: 45	设置场景的优先级 (第 7 步)	
11: 15	分析体系结构方法 (第 8 步)	
12: 30	中餐	
13: 30	分析体系结构方法 (第 8 步)	
14: 45	准备汇报结果/休息	
15: 30	描述结果 (第 9 步)	
16: 00	进一步的分析/角色的分配	
17: 00	休会	

第二个阶段

自学



- 1、为什么要评估软件体系结构？
- 2、从哪些方面评估软件体系结构？
- 3、ATAM评估方法的基本步骤是什么？

# 课 程 内 容

- ◇ 软件体系结构概论
- ◇ 软件体系结构建模
- ◇ 软件体系结构风格
- ◇ 软件体系结构描述
- ◇ 动态软件体系结构
- ◇ Web服务体系结构
- ◇ 基于体系结构的软件开发
- ◇ 软件体系结构的分析与测试
- ◇ 软件体系结构评估
- ◇ 软件产品线体系结构

#### ◇ 出现和发展

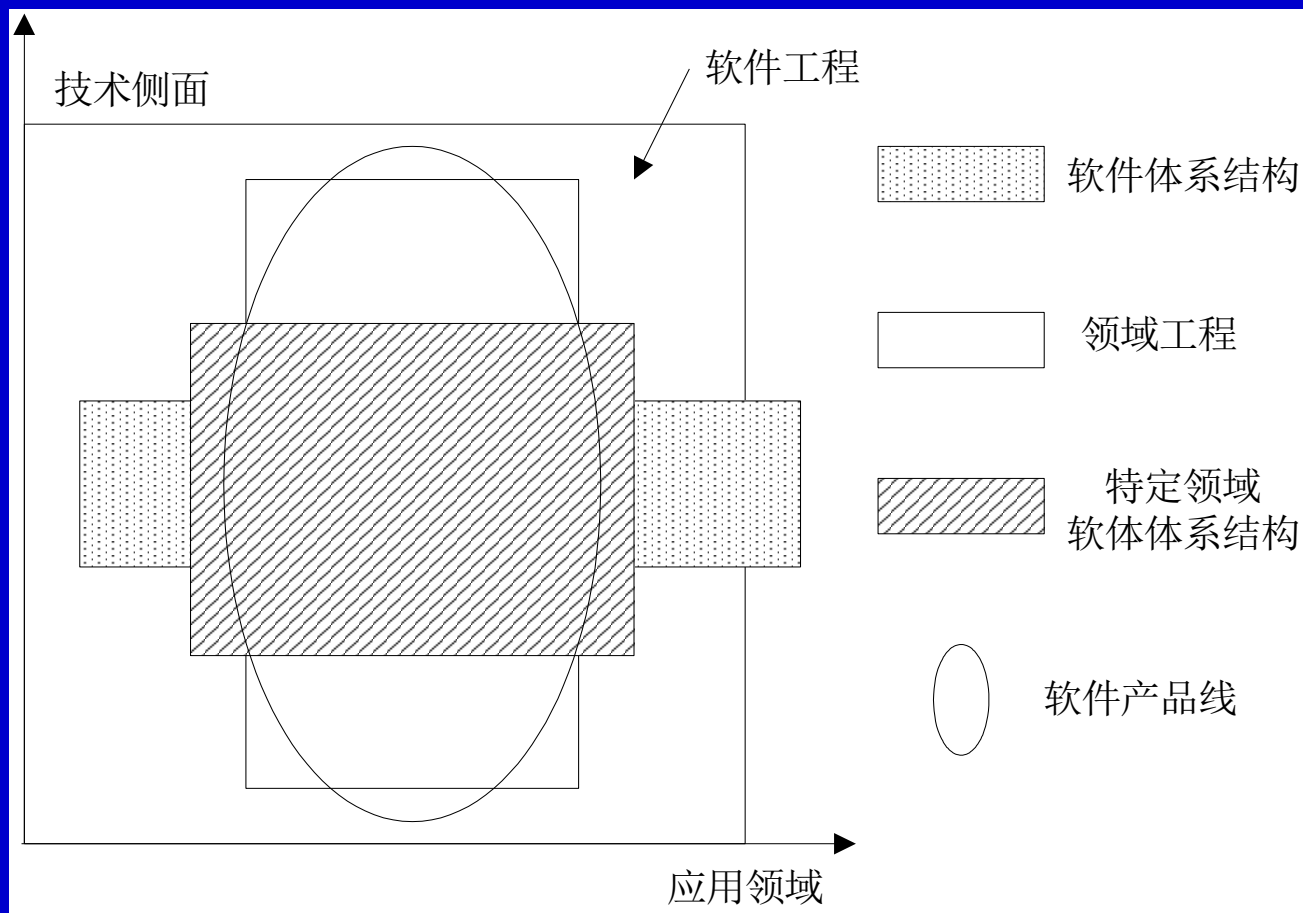
产品线的起源可以追溯到1976年Parnas对程序族的研究。软件产品线的实践早在20世纪80年代中期就出现。

据HP公司1996年对HP、IBM、NEC、AT & T等几个大型公司分析研究，他们在采用了软件产品线开发方法后，使产品的开发时间减少1.5-2倍，维护成本降低2—5倍，软件质量提升5—10倍，软件重用达50%—80%，开发成本降低12%—15%。

软件产品线的发展得益于软件体系结构的发展和软件重用技术的发展。



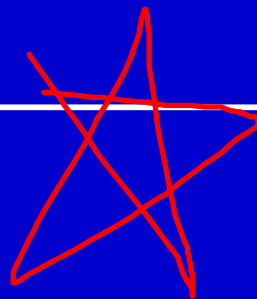
### ◇ 在软件工程中的地位



### ◇ 基本概念

- ◎ 将利用了产品间公共方面、预期考虑了可变性等设计的产品族称为产品线(Weiss和Lai)。
- ◎ 产品线就是由在系统的组成元素和功能方面具有共性和个性的相似的多个系统组成的一个系统族。
- ◎ 软件产品线就是在一个公共的软件资源集合基础上建立起来的，共享同一个特性集合的系统集合(Bass, Clements和Kazman)。
- ◎ 一个软件产品线由一个产品线体系结构、一个可重用构件集合和一个源自共享资源的产品集合组成，是组织一组相关软件产品开发的方式(Jan Bosch)。

### ◇ 基本概念

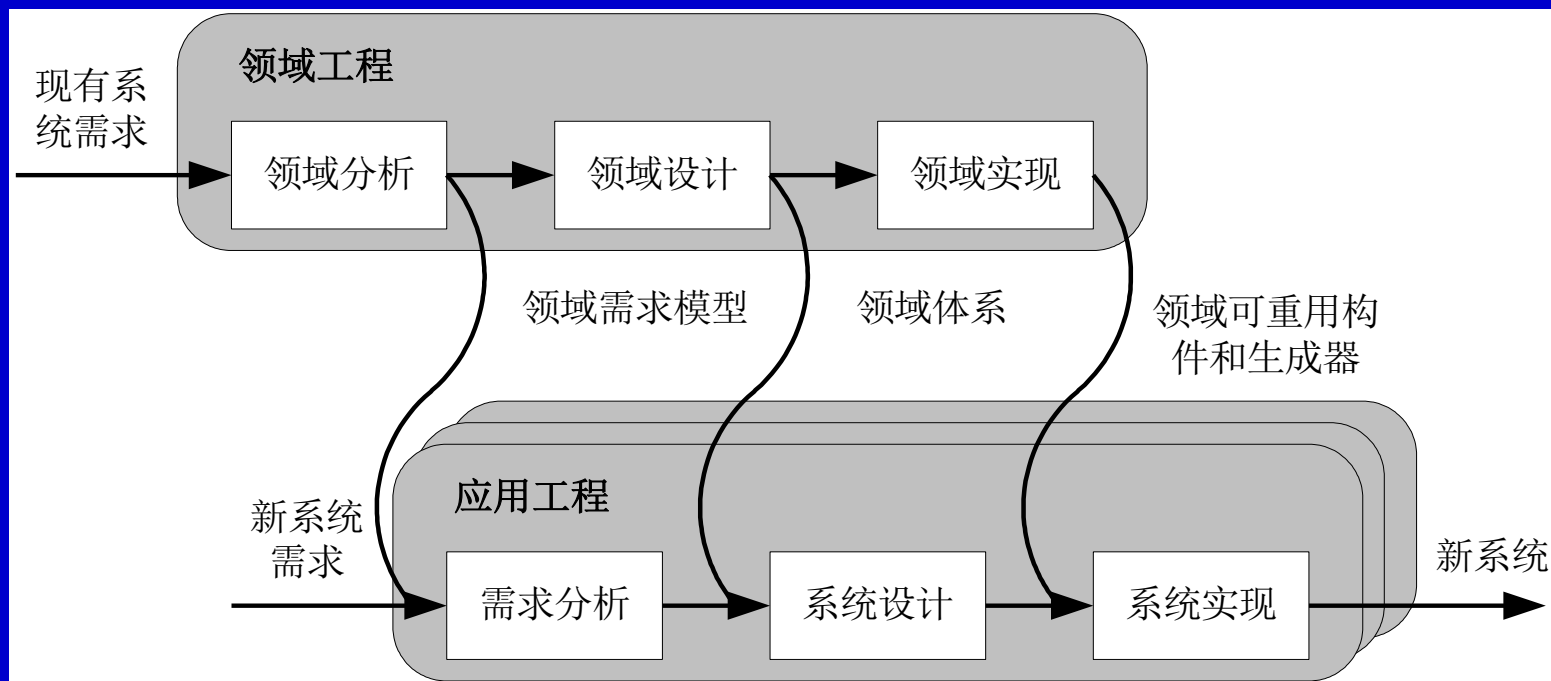


CMU/SEI的定义：“产品线是一个产品集合，这些产品共享一个公共的、可管理的特征集，这个特征集能满足选定的市场或任务领域的特定需求。这些系统遵循一个预描述的方式，在公共的核心资源(core assets)基础上开发的。”

根据SEI的定义，软件产品线主要由两部分组成：**核心资源、产品集合**。核心资源是领域工程的所有结果的集合，是产品线中产品构造的基础。也有组织将核心资源库称为“平台”。核心资源必定包含产品线中所有产品共享的产品线体系结构，新设计开发的或者通过对现有系统的再工程得到的、需要在整个产品线中系统化重用的软件构件。与软件构件相关的测试计划、测试实例以及所有设计文档，需求说明书和领域模型还有领域范围的定义也是核心资源，采用COTS的构件也属于核心资源。产品线体系结构和构件是用于软件产品线中的产品的构建和的核心资源最重要的部分。

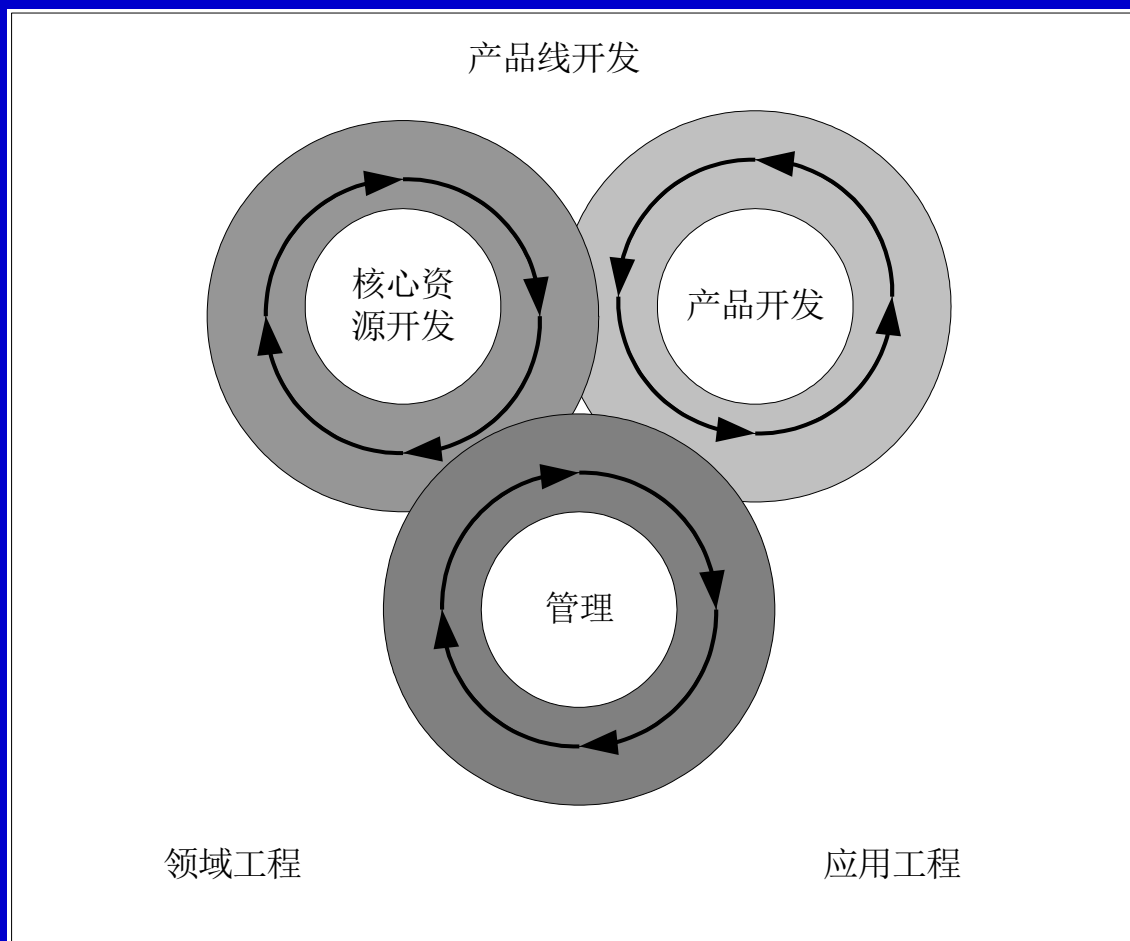
### ◇ 软件产品线的过程模型

#### 1、双生命周期模型



### ◇ 软件产品线的过程模型

#### 2、SEI模型

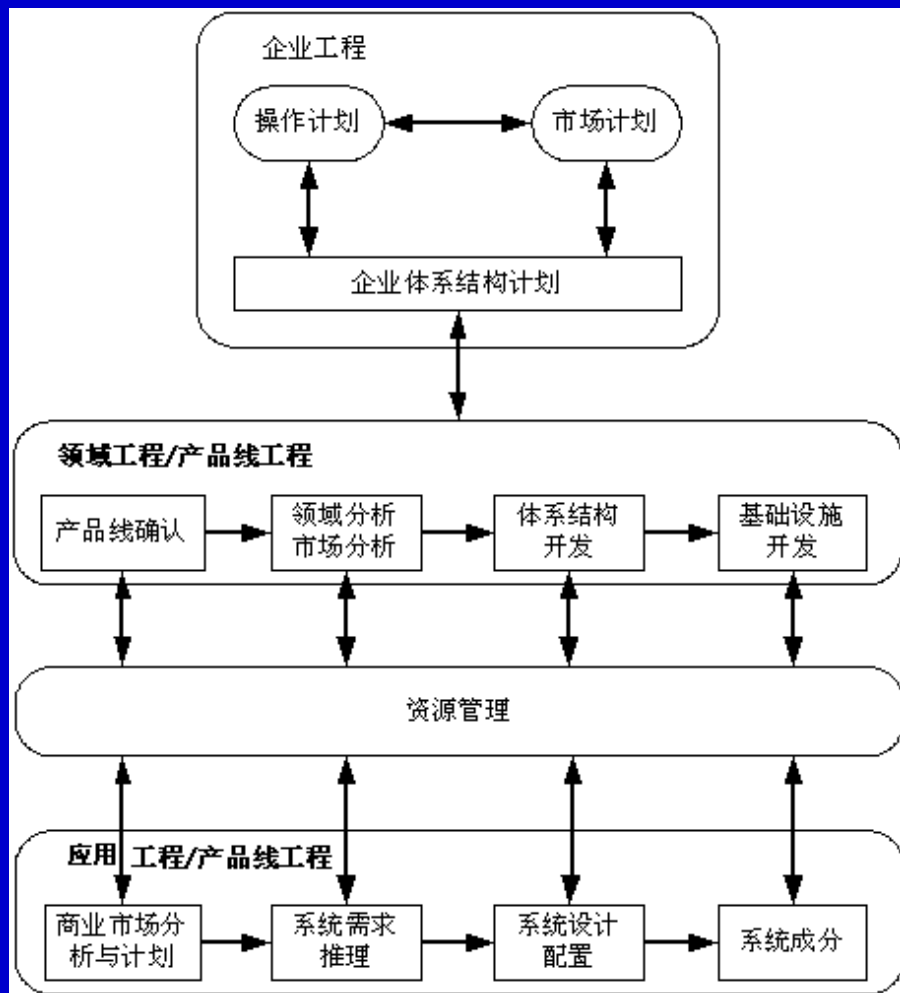


## ◇ 软件产品线的过程模型

### 2、SEI模型

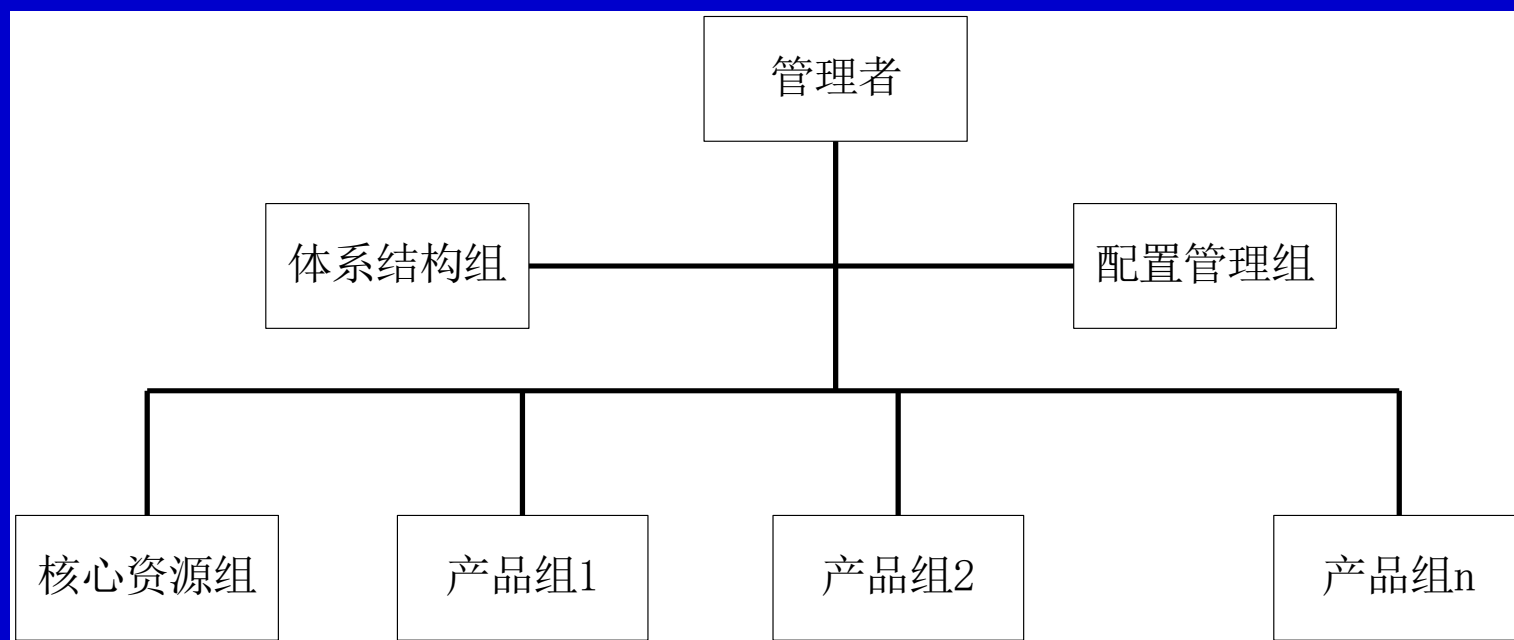
- ◎ 循环重复是产品线开发过程的特征，也是核心资源开发、产品线开发以及核心资源和产品之间协作的特征；
- ◎ 核心资源开发和产品开发没有先后之分；
- ◎ 管理活动协调整个产品线开发过程的各个活动，对产品线的成败负责；
- ◎ 核心资源开发和产品开发是两个互动的过程，三个活动和整个产品线开发之间也是双向互动的。

### ◇ 软件产品线的过程模型



### 3、三生命周期模型

### ◇ 软件产品线的组织结构





#### ◇ SEI产品线组织结构(1)

将产品线组织分为四个工作小组：

- ◎ **市场人员**是产品线和产品能力、客户需求之间的沟通桥梁；
- ◎ **核心资源组**负责体系结构和其他核心资源的开发；
- ◎ **应用组**负责交付给客户的系统的开发；
- ◎ **管理者**负责开发过程的协调、商务计划等。

#### ◇ SEI 产品线组织结构 (2)

设有**独立核心资源小组**的组织结构通常合适于至少由50到100人组成的较大型的软件开发组织，设立独立的核心资源小组可以使小组成员将精力和时间集中在核心资源的认真的设计和开发上，得到更通用的资源。

另外一种典型的组织结构**不设立独立的核心资源小组**，核心资源的开发融入各系统开发小组中，只是设立专人负责核心资源开发的管理。这种组织结构的重点不在核心资源的开发上，所以比较适合于组成产品线的产品**共性相对较少**，开发独立产品所需的工作量相对较大的情况。也是小型软件组织向软件产品线开发过渡时采用的一种方法。

#### ◇ Jan Bosch产品线组织结构(1)

◎ **开发部门**：所有的软件开发集中在一个部门，每个人都可承担领域工程和应用工程中适合的任务，简单、利于沟通，适用于不超过30人的组织。

◎ **业务部门**：每个部门负责产品线中一个和多个相似的系统，共性资源由需要使用它的一个和几个部门协作开发，整个团体都可享用。资源更容易共享，适用于30—100人的组织，主要缺点是业务部门更注重自己的产品而将产品线的整体利益放在第二位。

#### ◇ Jan Bosch产品线组织结构(2)

◎ **领域工程部门**：有一个专门的单位——领域工程部门负责核心资源库的开发和维护，其他业务单位使用这些核心资源来构建产品。这种结构可有效的降低通讯的复杂度、保持资源的通用性，适于超过100人的组织。缺点是难以管理领域工程部门和不同产品工程部门之间的需求冲突和因此导致的开发周期增长。

◎ **层次领域工程部门**：对于非常巨大和复杂的产品线可以设立多层（一般为两层）领域工程部门，不同层部门服务的范围不同。这种模型趋向臃肿，对新需求的响应慢。

◇ 软件产品线的建立方式

	演化方式	革命方式
基于现有产品集	基于现有产品体系结构开发产品线的体系结构 经演化现有构件的文件一次开发一个产品线构件	产品线核心资源的开发基于现有产品集的需求和可预测的、将来需求的超集
全新产品线	产品线核心资源随产品新成员的需求而演化	开发满足所有预期产品线成员的需求的产品线核心资源

#### ◇ 软件产品线的演化

从整体来看，软件产品线的发展过程有三个阶段，**开发阶段**、**配置分发阶段**和**演化阶段**。

引起产品线体系结构演化的原因：产品线与技术变化的协调、现有问题的改正、新功能的增加、对现有功能的重组以允许更多的变化等等。

产品线的演化包括产品线核心资源的演化、产品的演化和产品的版本升级。这样在整个产品线就出现了：核心资源的新旧版本、产品的新旧版本和新产品等。它们之间的协调是产品线演化研究的主要问题。

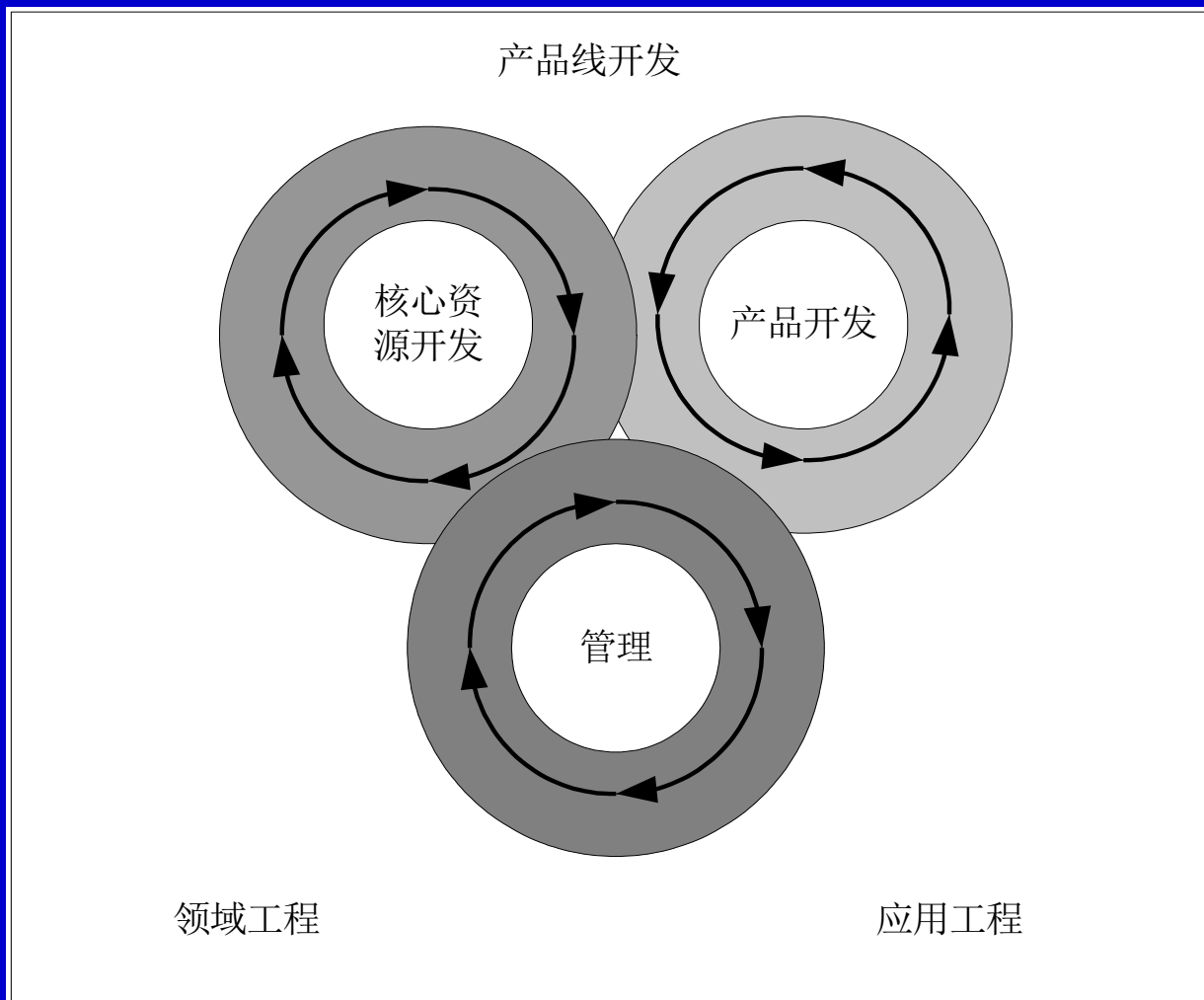
框架是封装了特定应用族抽象设计的抽象类的集合，框架又是一个模板，关键的方法和其他细节在框架实例中实现。

应用框架又称为通用应用，是为一个特定应用领域的软件系统提供可重用结构的一组相互协作的类的集合。

◎ 黑盒框架

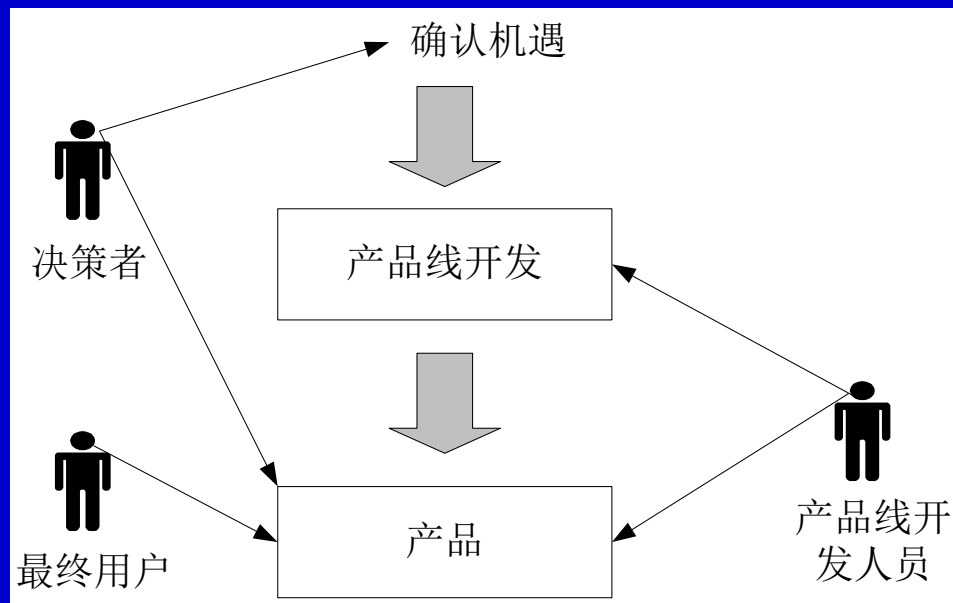
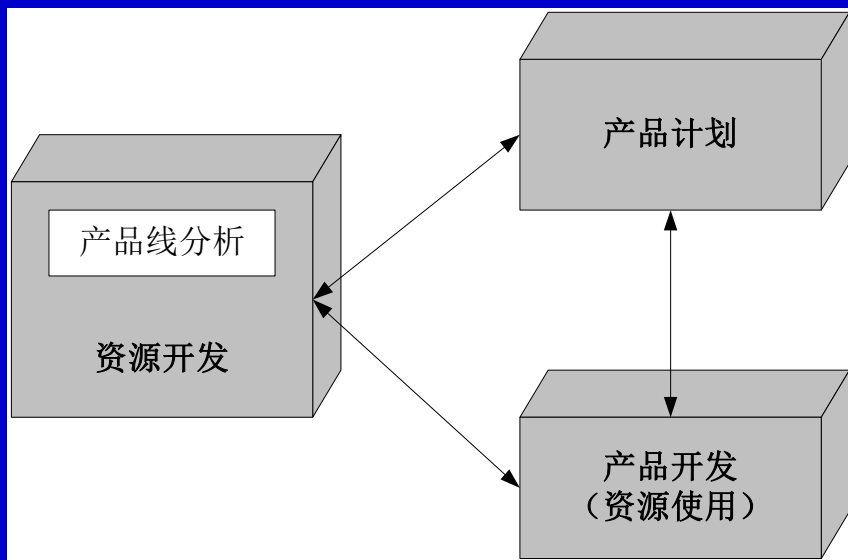
◎ 白盒框架

### ◇ 产品线基本活动

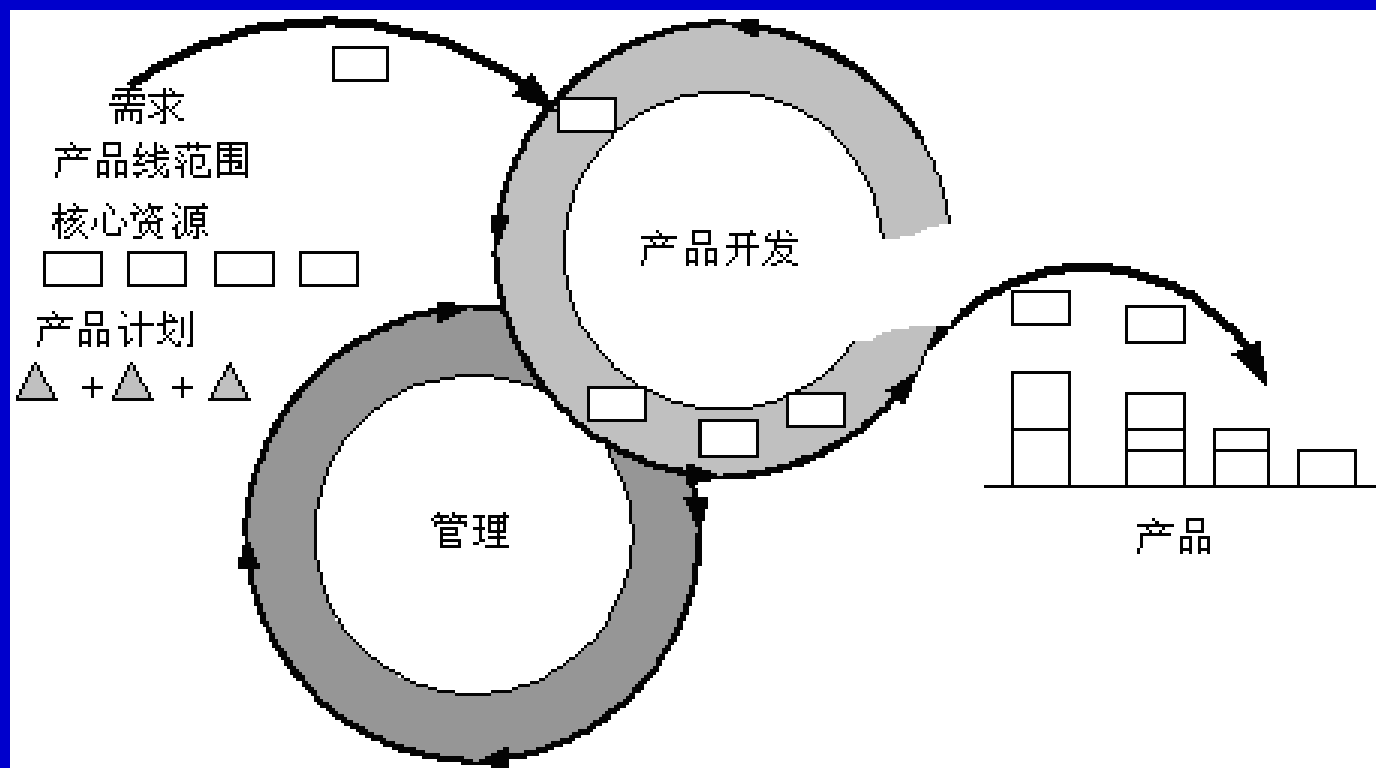




### ◇ 产品线分析



### ◇ 产品开发



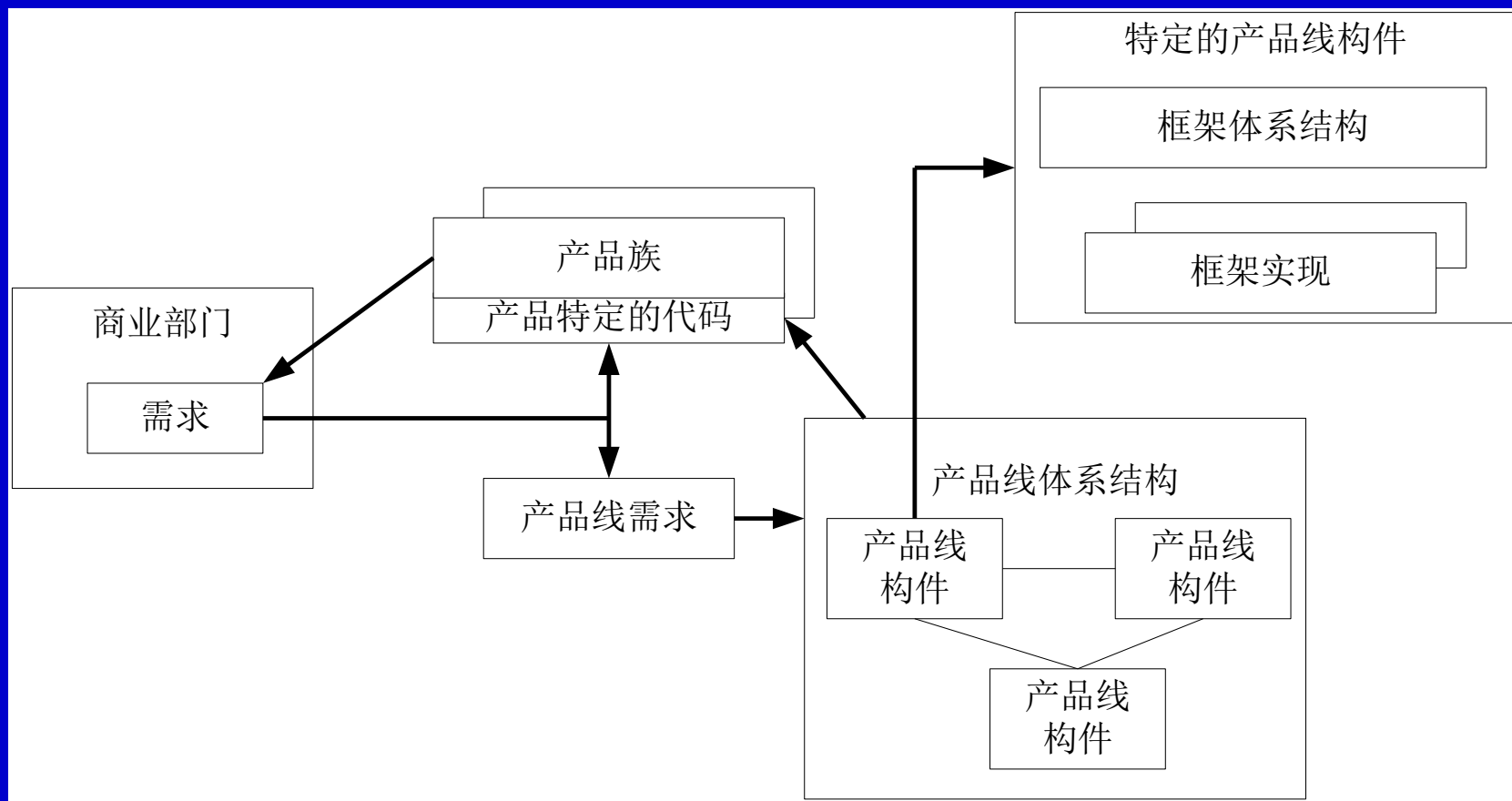
#### ◇ 产品线体系结构简介

- ◎ 软件产品线体系结构指一个软件开发组织为一组相关应用或产品建立的公共体系结构。
- ◎ 同领域模型一样，软件产品线体系结构中也可以分为共性部分和个性部分。
- ◎ 产品线体系结构是产品线核心资源的早期和主要部分，在产品线的生命周期中，产品线体系结构应该保持相对小和缓慢的变化以便在生命周期中尽量保持一致。产品线体系结构要明确定义核心资源库中软件构件集合及其相关文档。

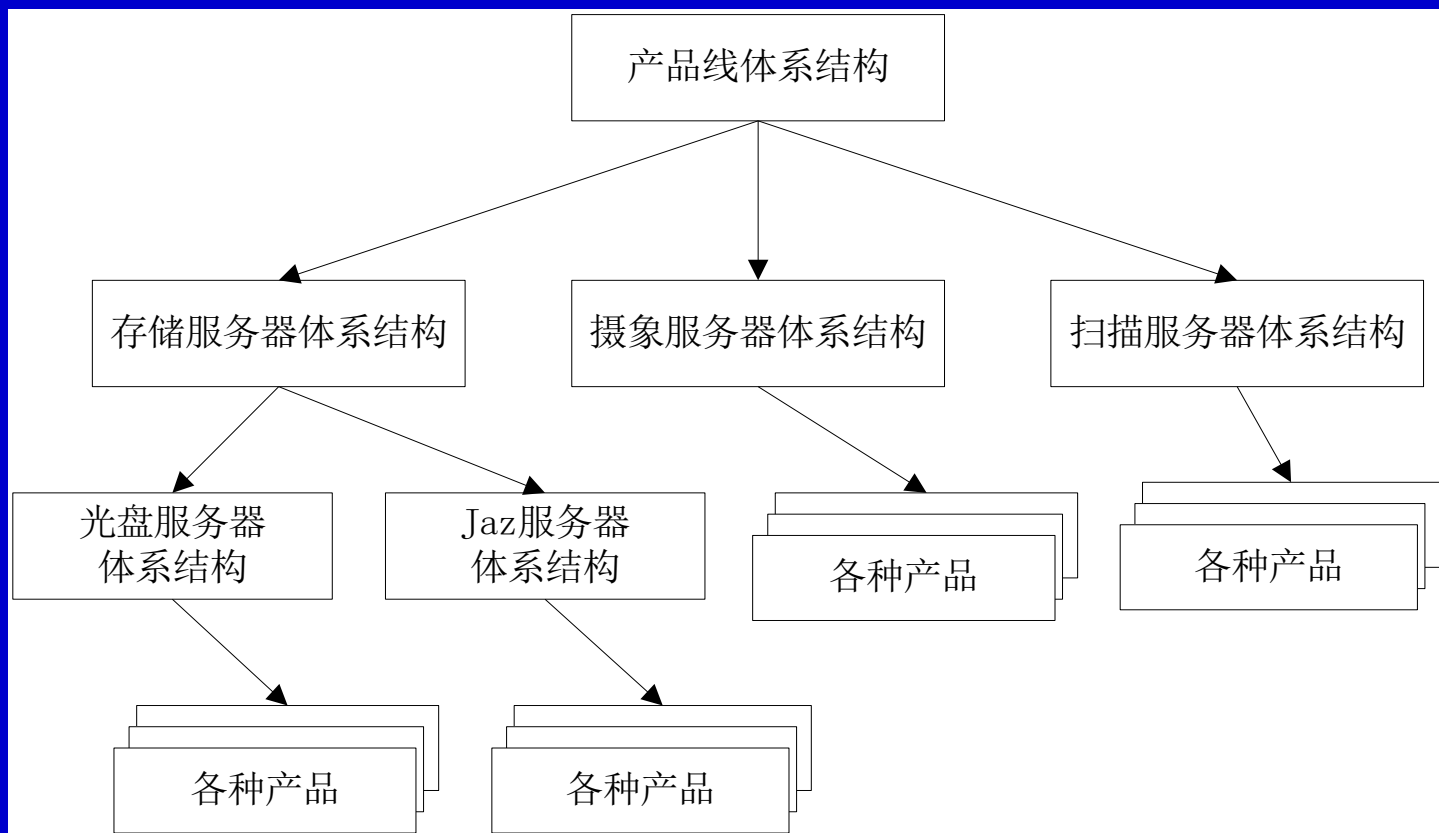
#### ◇ 产品线体系结构的标准化和定制

◎ 为适应应用的规模增大、复杂度提高，软件技术不断发展，相继出现了中间件技术、软件产品线等。

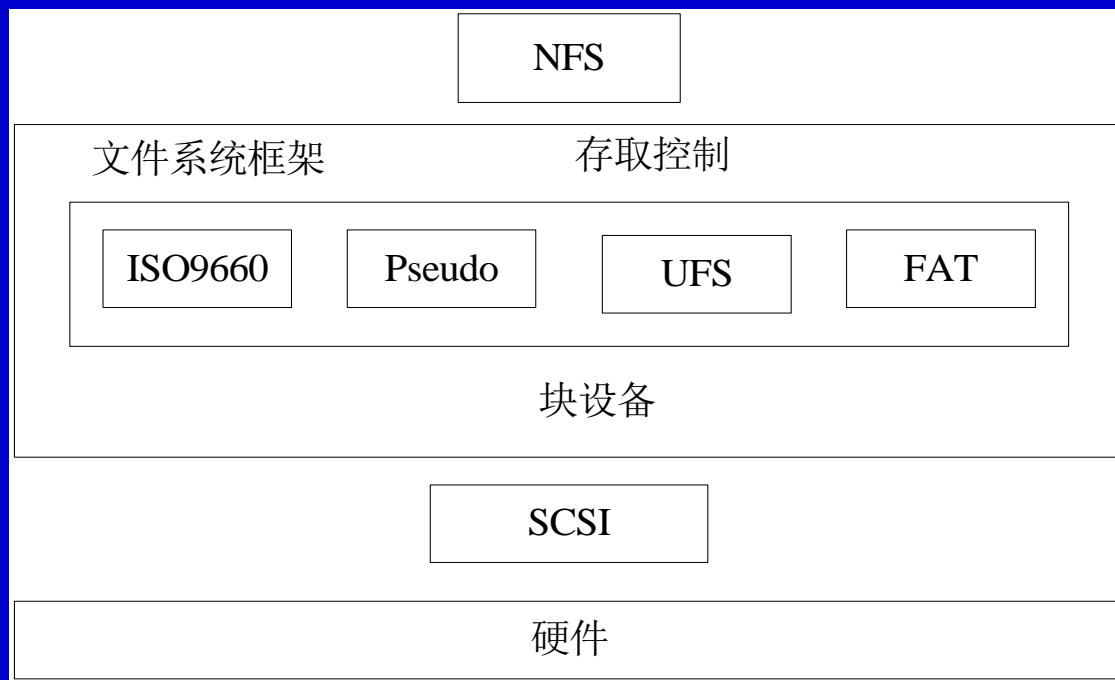
◎ 体系结构风格是一个使产品和产品线具有良好的可移植性的结构，产品和产品线通过最小的修改就可移植到一个新的平台上。



### ◇ 背景介绍

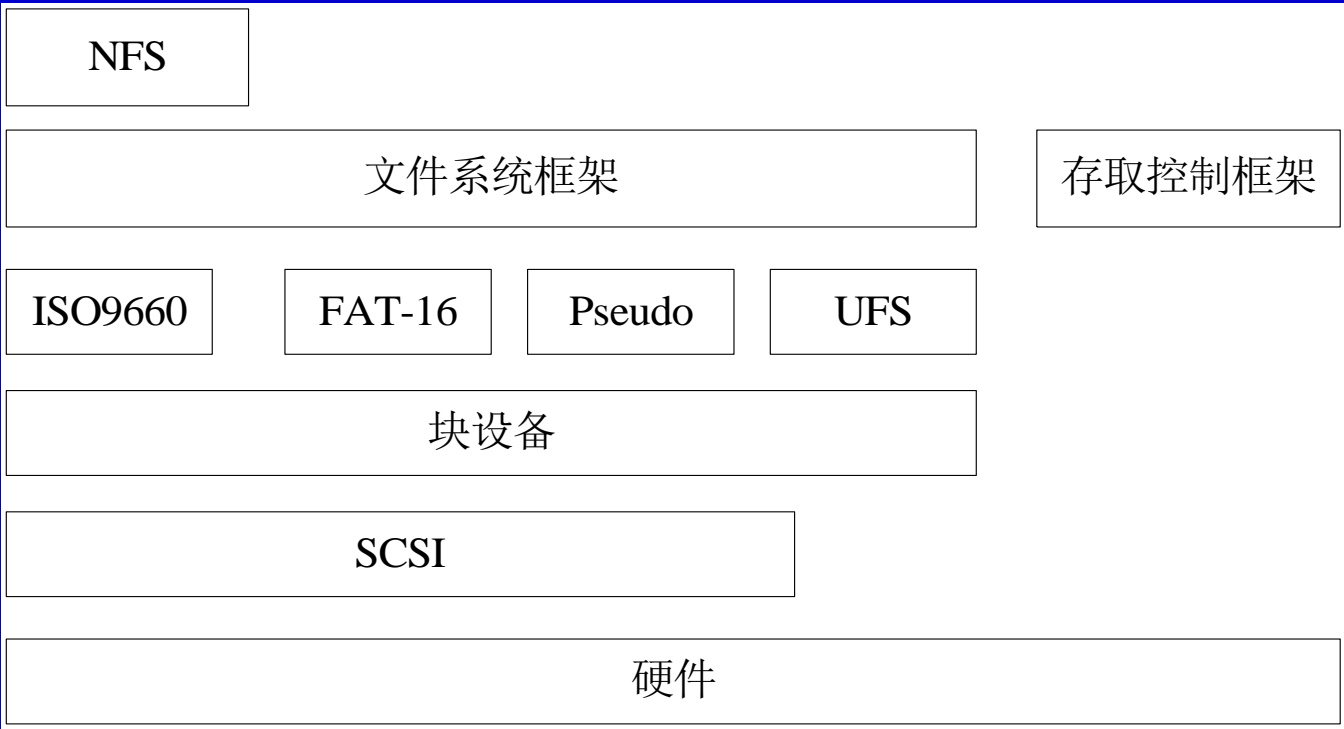


### ◇ 背景介绍



第一代文件系统框架

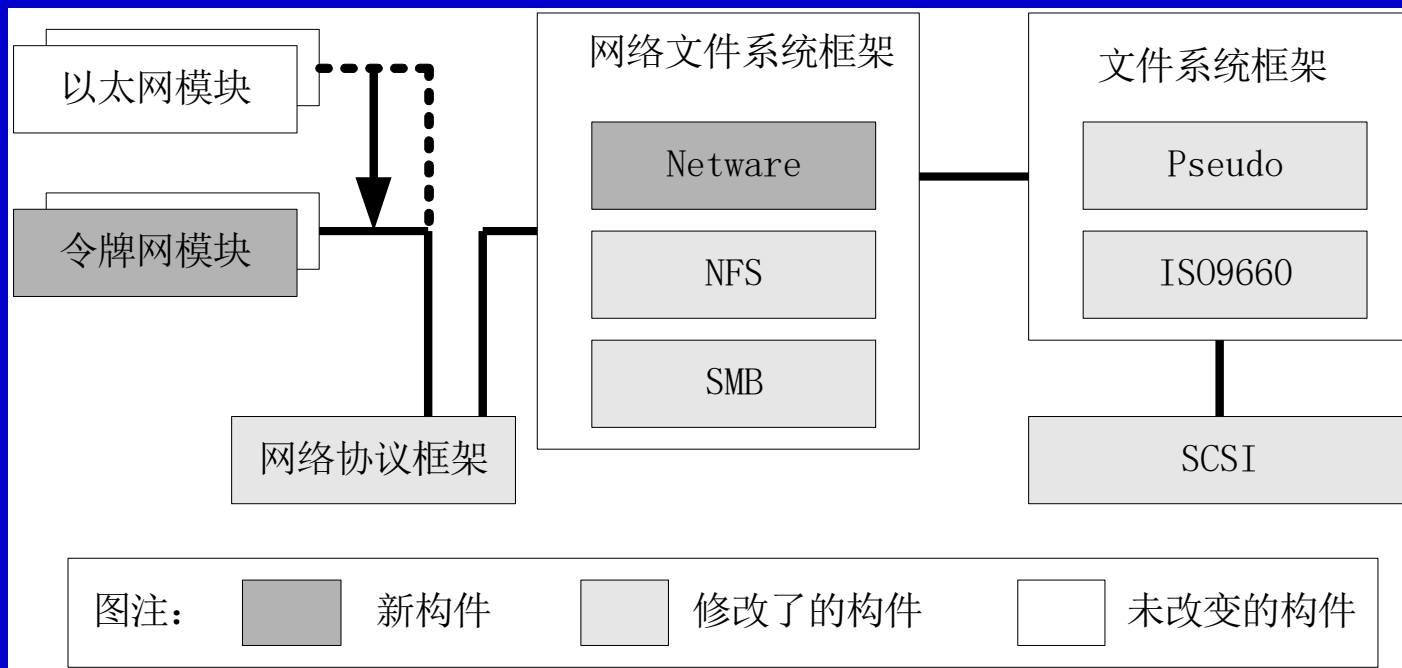
### ◇ 背景介绍



第二代文件系统框架

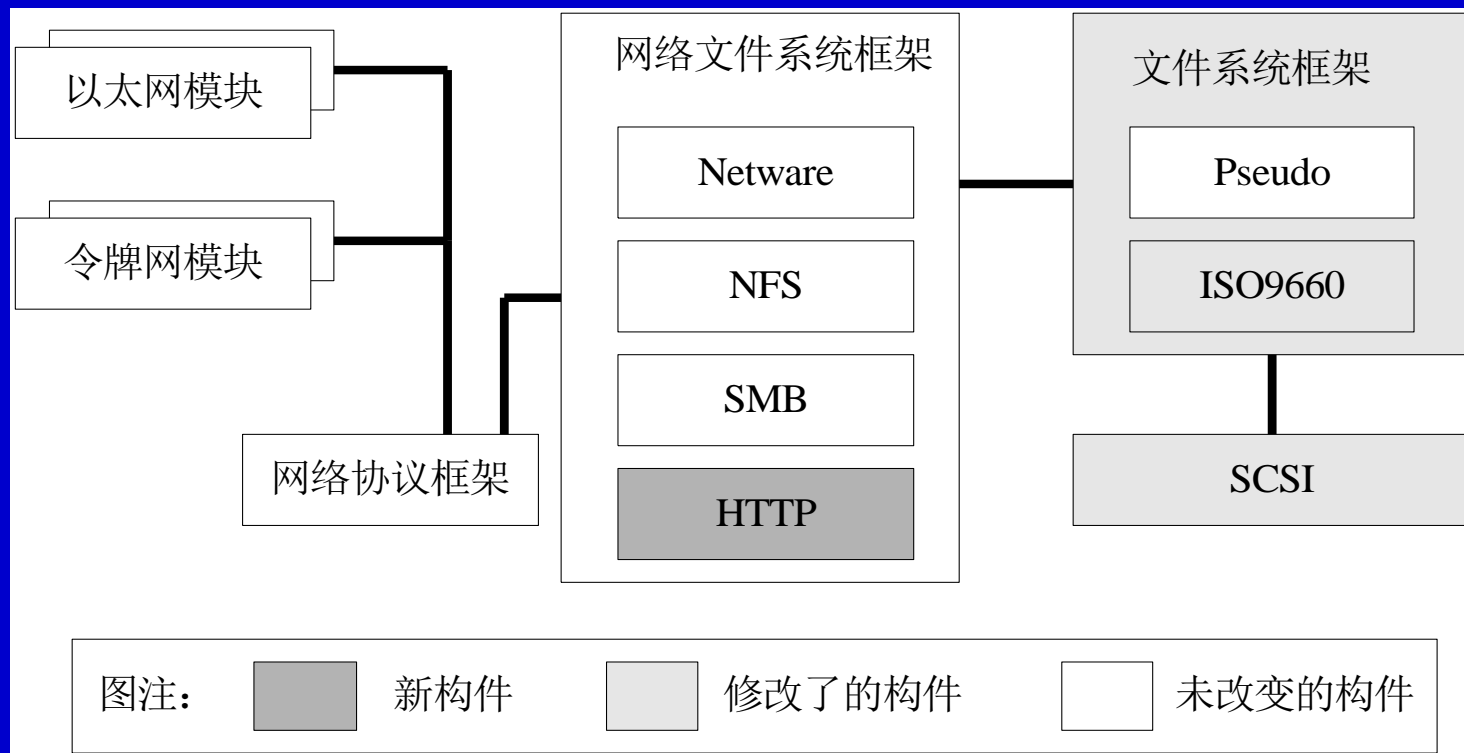


### ◇ 两代产品的各种发行版本



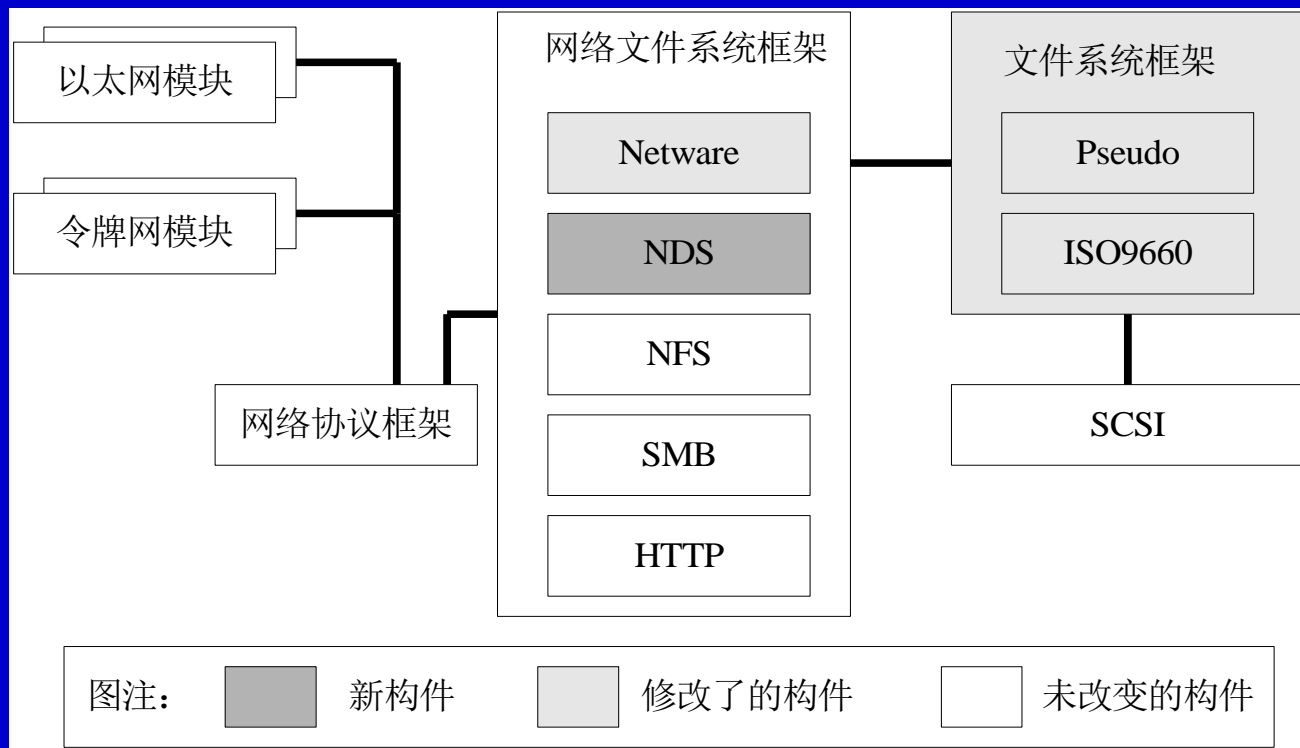
第一代产品的第二个版本

### ◇ 两代产品的各种发行版本



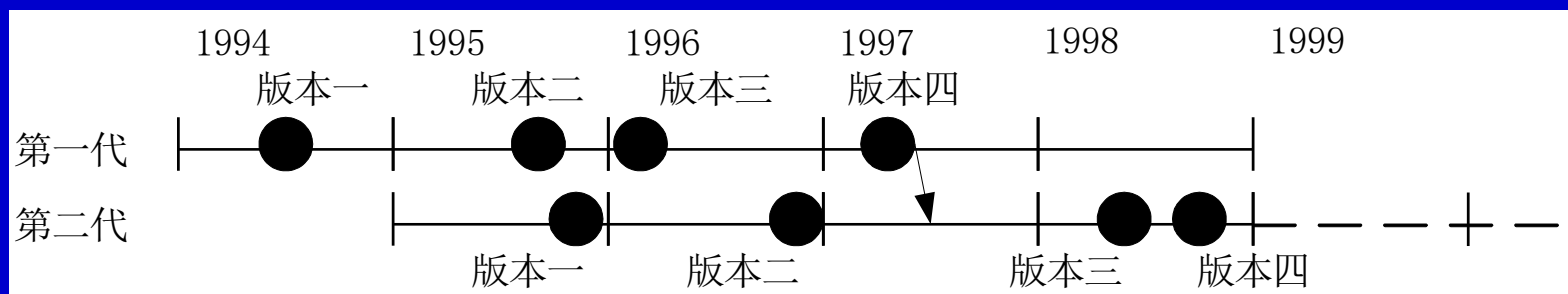
第一代产品的第三个版本

### ◇ 两代产品的各种发行版本



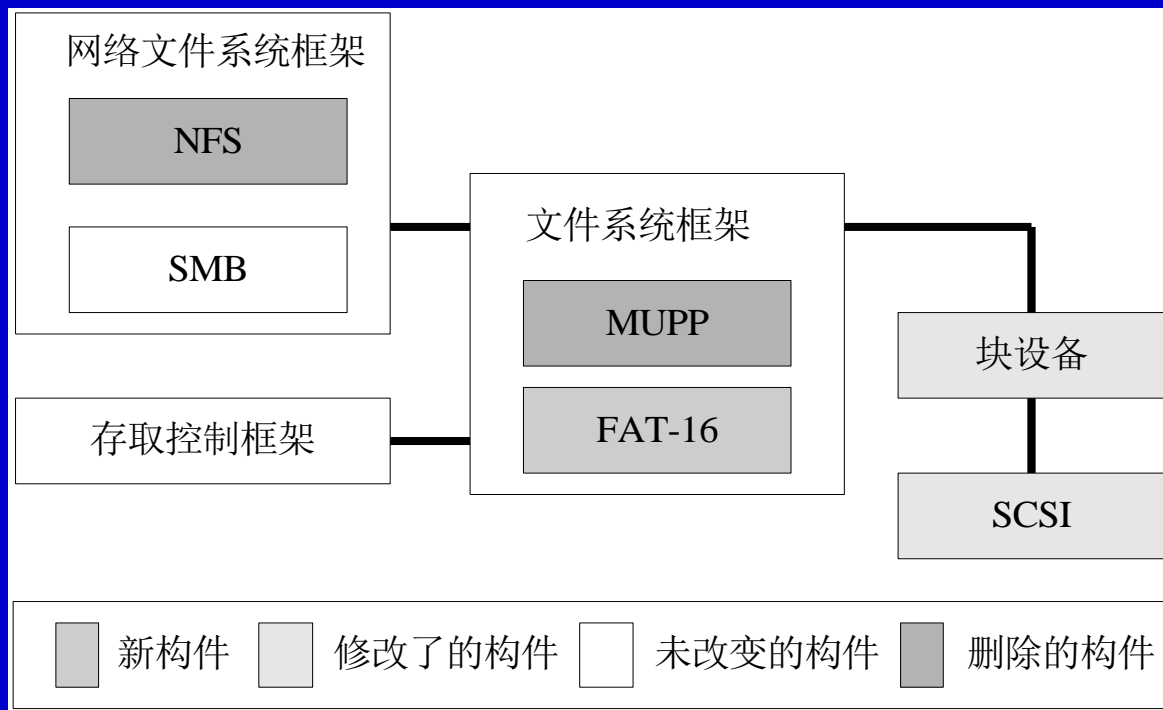
第一代产品的第四个版本

#### ◇ 两代产品的各种发行版本



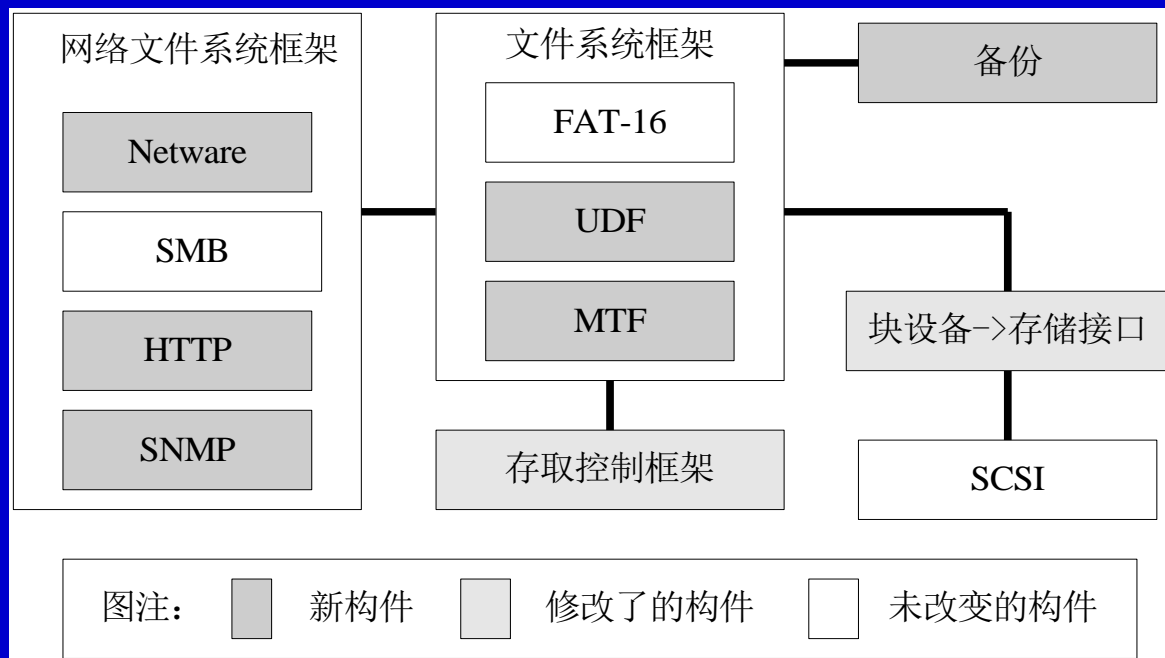
文件系统框架的时间线

#### ◇ 两代产品的各种发行版本



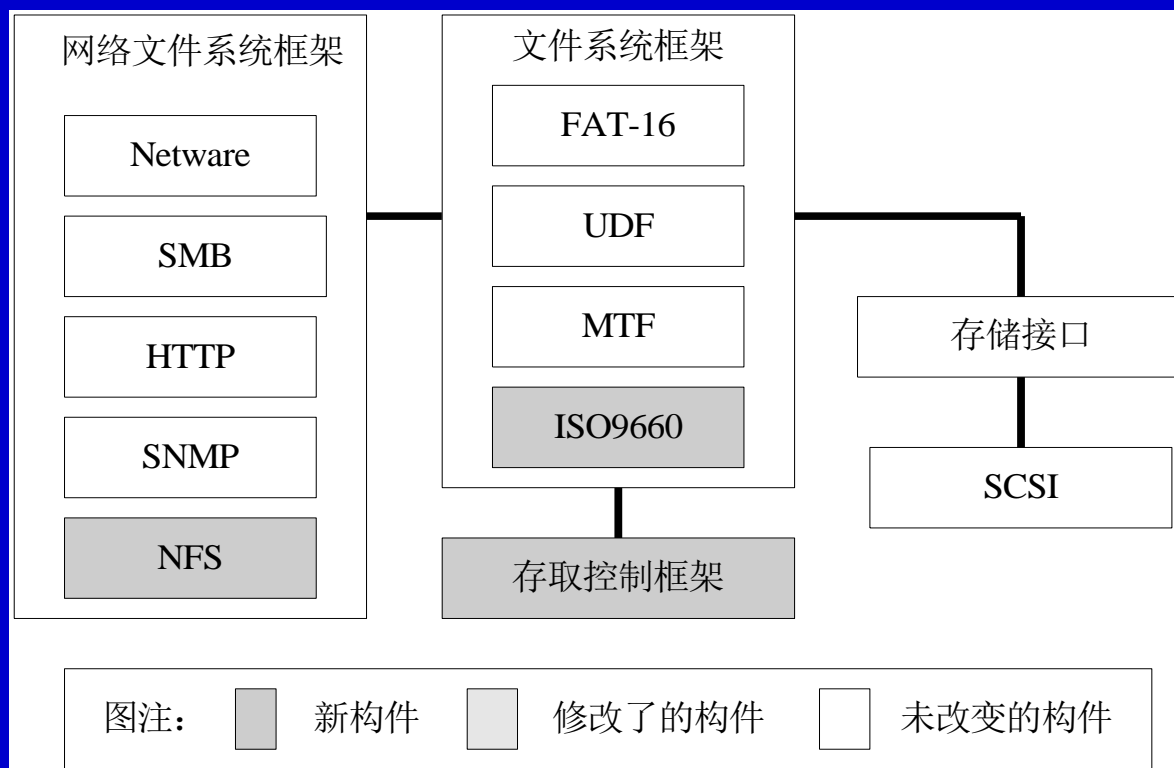
第二代产品的第二个版本

### ◇ 两代产品的各种发行版本



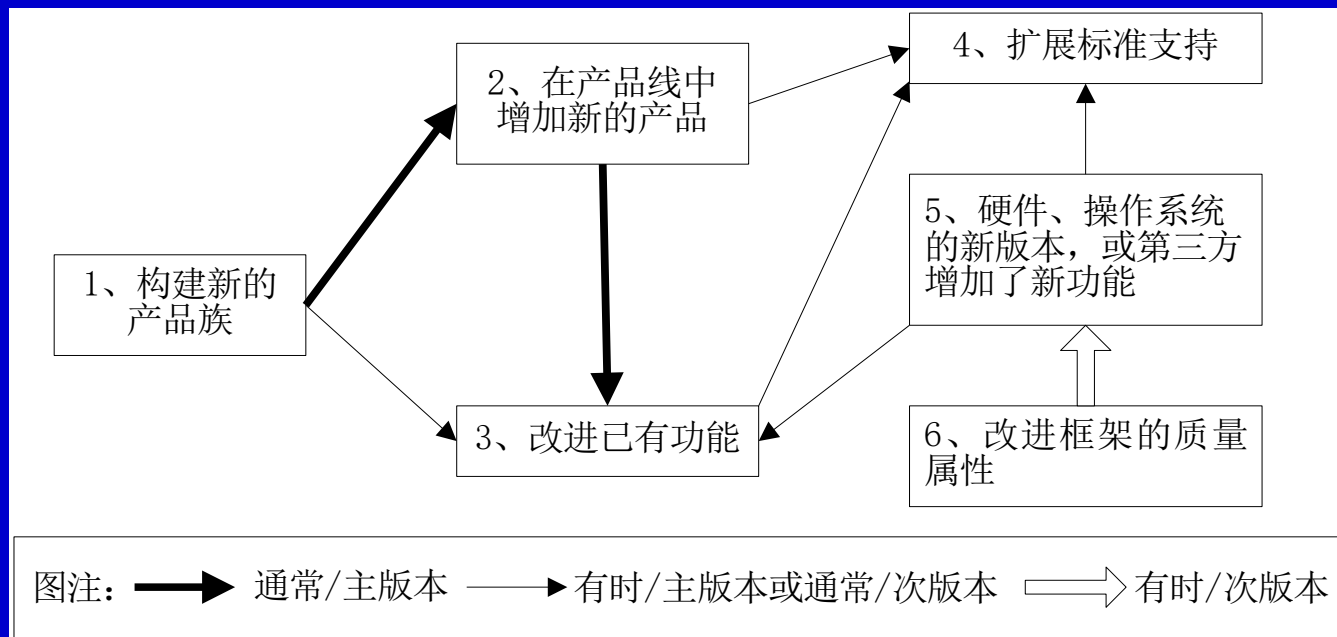
第二代产品的第三个版本

### ◇ 两代产品的各种发行版本



第二代产品的第四个版本

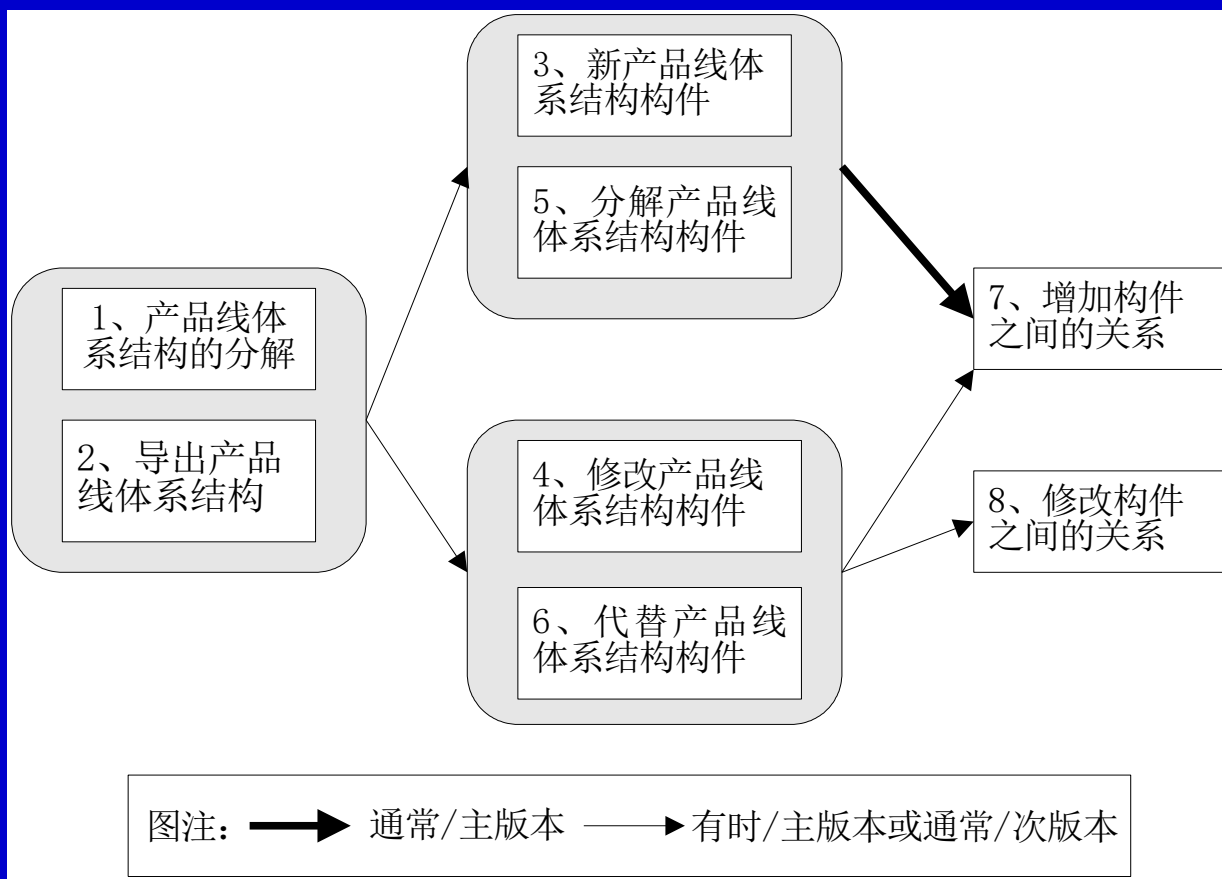
### ◇ 需求和演化的分类



需求分类之间的关系

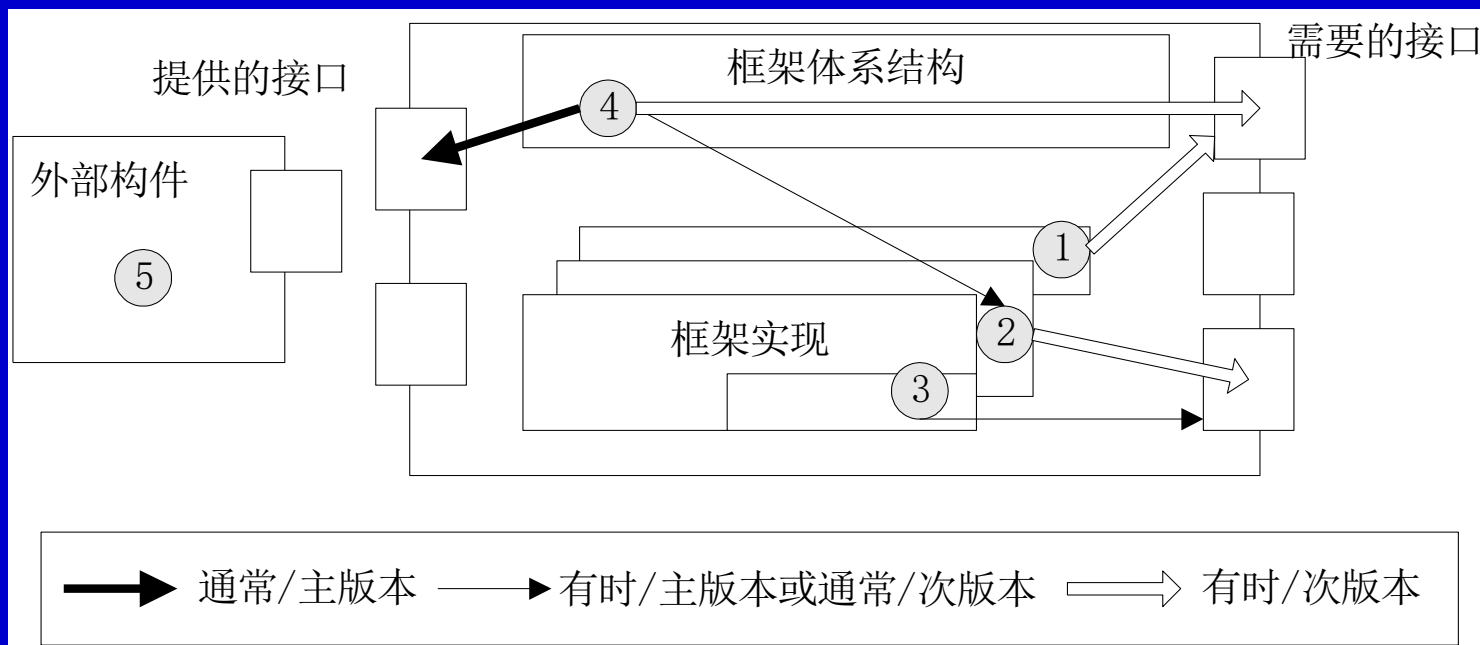


### ◇ 需求和演化的分类



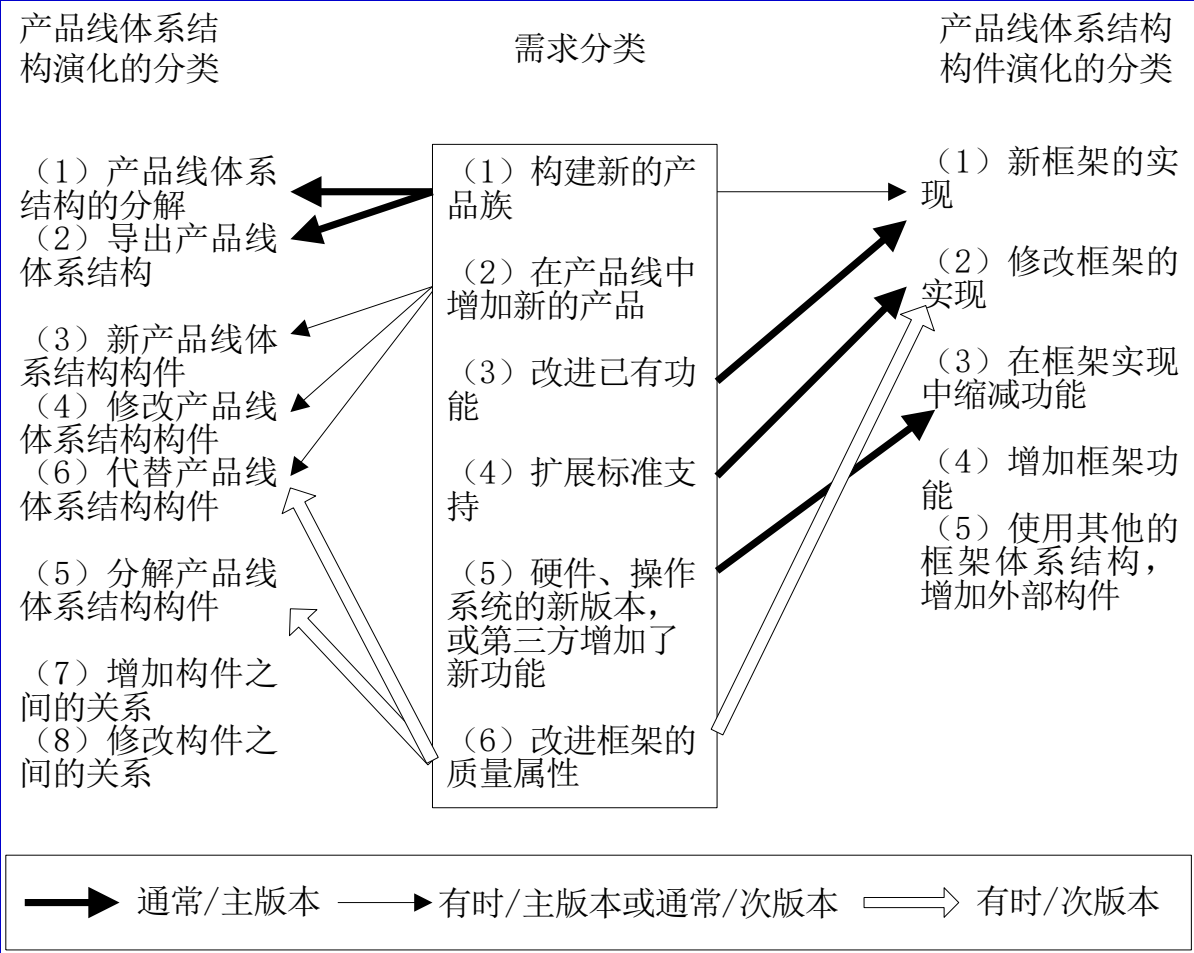
产品线体系结构分类之间的关系

### ◇ 需求和演化的分类



### 产品线体系结构构件演化分类

### ◇ 需求和演化的分类



各种演化之间的关系

A公司是一家相对较大的软件和硬件企业，专业从事网络设备的开发。从单一的产品开始，现在，已经延伸到包括摄相服务器、扫描服务器、光盘服务器以及其他的存储服务器在内的产品。公司原来的产品都是一个一个地开发，每个软件组织一个项目组。为了适应快速变化的市场，降低开发成本，公司想引入产品线方法。然而，软件产品线开发涉及了一个软件开发组织的多个产品，选择了软件产品线意味着要承担由此带来的许多风险。

所以，公司的CTO王总决定在弄清三个问题之后再做决定，首先就是本公司的业务范围是否适合使用产品线方法，其次是如何在原有产品的基础上建立产品线，最后是成功实施产品线的主要因素是什么？

[问题1]

请用100字以内文字说明A公司是否适合采用产品线方法？为什么？

[问题2]

请用200字以内文字说明如何在原有产品的基础上建立产品线？

[问题3]

请用150字以内文字说明成功实施产品线的主要因素是什么？