软件系统设计与体系结构

一.基于软件重用的设计方法学

1.软件重用

(1)定义

- 软件重用是一种由预先构造好的、以重用为目的而设计的软件构件来建立或组装软件系统的过程。

(2)重用方法

1. 代码复用
2. 设计复用
3. 分析复用
4. 测试信息复用

(3)优缺点

- 优点:在已有工作的基础上，充分利用以前系统开发中积累的知识和经验 ，将开发的重点转移到现有系统的特有构成成分。
- 缺点:
    - 过程方面：缺少支持软件重用的软件开发过程
    - 工程方面：缺乏支持实施软件重用的工具、没有构件重用库或库信息很快过时
    - 组织方面：管理者轻视
    - 资金方面：初期投入大

(4)原则

- 在已有工作的基础上，充分利用以前系统开发中积累的知识和经验 ，将开发的重点转移到现有系统的特有构成成分。

(5)判断题

1. 启动软件重用仅仅是为了引入合适的技术(X)
2. 软件重用可能会有用，但太昂贵(√)
3. 采用OO的程序设计语言会导致系统化的重用(X)
4. 只能将重用局限在代码构件上(X)
5. 开发人员要从数以万计的小构件中选取要重用的构件(X)
6. 只要建立可重用构件库就会促使构件重用(X)
7. 启动软件重用仅仅是为了引入合适的技术(X)

2.构件技术

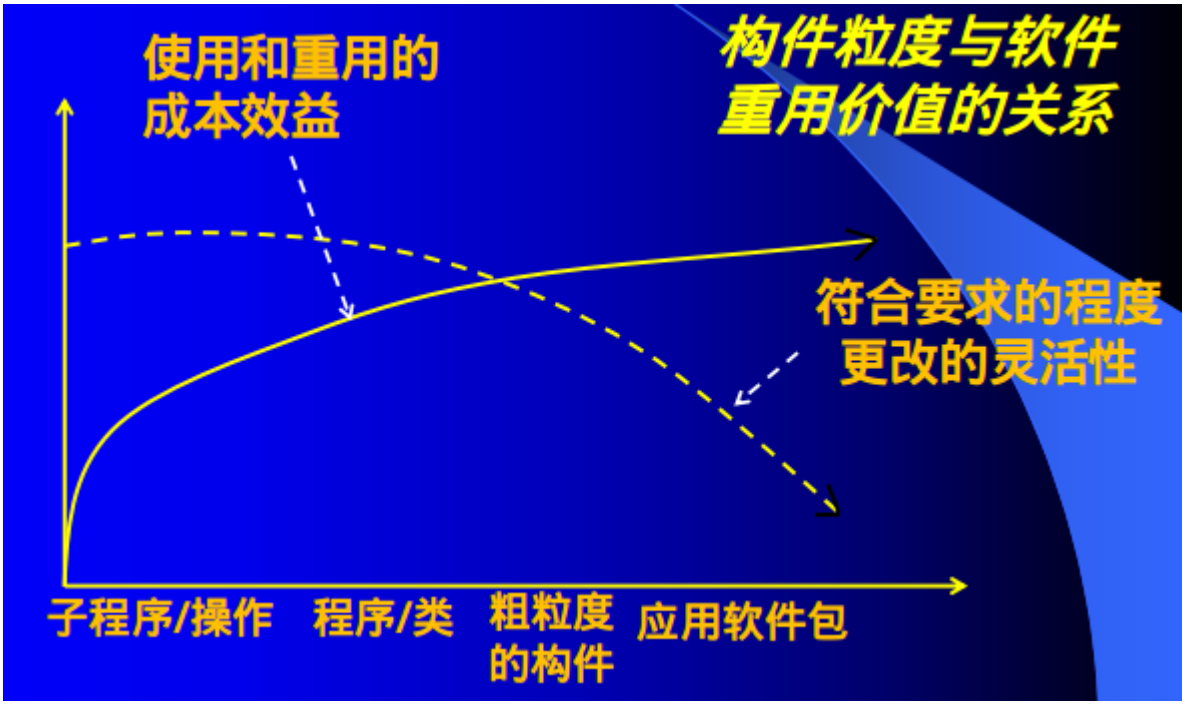
(1)定义

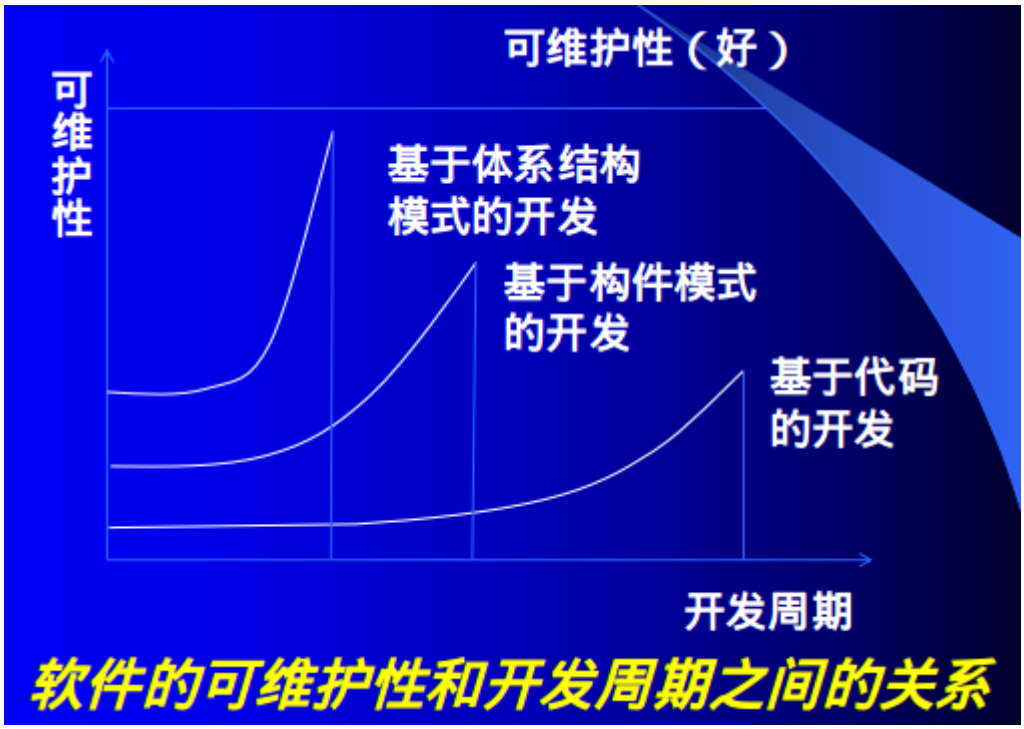
- 语义完整、语法正确和有重用价值的软件单元，是软件重用过程中可明确辨识的成分。

(2)获取途径

1. 从现有构件中获得（★提倡）
2. 通过遗留工程
3. 现成的商业构件
4. 开发新构件

(3)好处





(4)人员划分

- 通过实现构件库管理系统，软件开发人员被分为两类：构件开发人员和构件组装人员，实现了软件开发人员的合理分工。

二.软件体系架构SA的概念

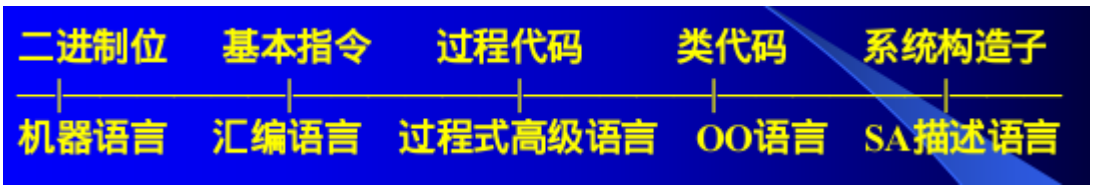
1.发展历程

(1)发展历程

- “无体系结构”设计阶段：汇编语言，规模小，无需建模
- 萌芽阶段——程序结构设计阶段：高级程序语言，SA的概念被明确
- 初级阶段——模块结构阶段：OO技术，从多角度对系统建模（如UML）
- 高级阶段——构件结构阶段：以Kruchten提出的“4+1”模型为标志

  尽管目前仍存在许多问题，但地位已经迅速上升。

(2)重用粒度的提升



- 重用粒度提升的原因：
  1. 软件系统规模和复杂度的提升
  2. 软件行业开发经验的积累及相关理论的提出

(3)SA的定义

- 尽管目前有很多对SA的定义，但我们倾向于更容易被理解的Garlan & Shaw 模型
- SA={Component, Connectors, Constrains} 软件体系架构=构件+连接件+约束（配置）

2.SA的构成要素

(1)软件构件

- 强调三点：
  - 构件的粒度
  - 构件内部
  - 端口
- 补充一点：
  - 还要关注构件的进化能力，它是系统进化的基础。

(2)连接件

- 用来建立构件间的交互，以及支配这些交互规则的体系结构的构造模块。
- 常见的连接件：

体系结构	连接件名称
“客户机/服务器”	通信协议或通信机制
“管道-过滤器”	管道（Pipe）

(3)约束(配置)

- 描述了体系结构的构件与连接件的连接图，它一般是对象连接时的规则，或指明构件连接的势态或条件。

三.SA风格

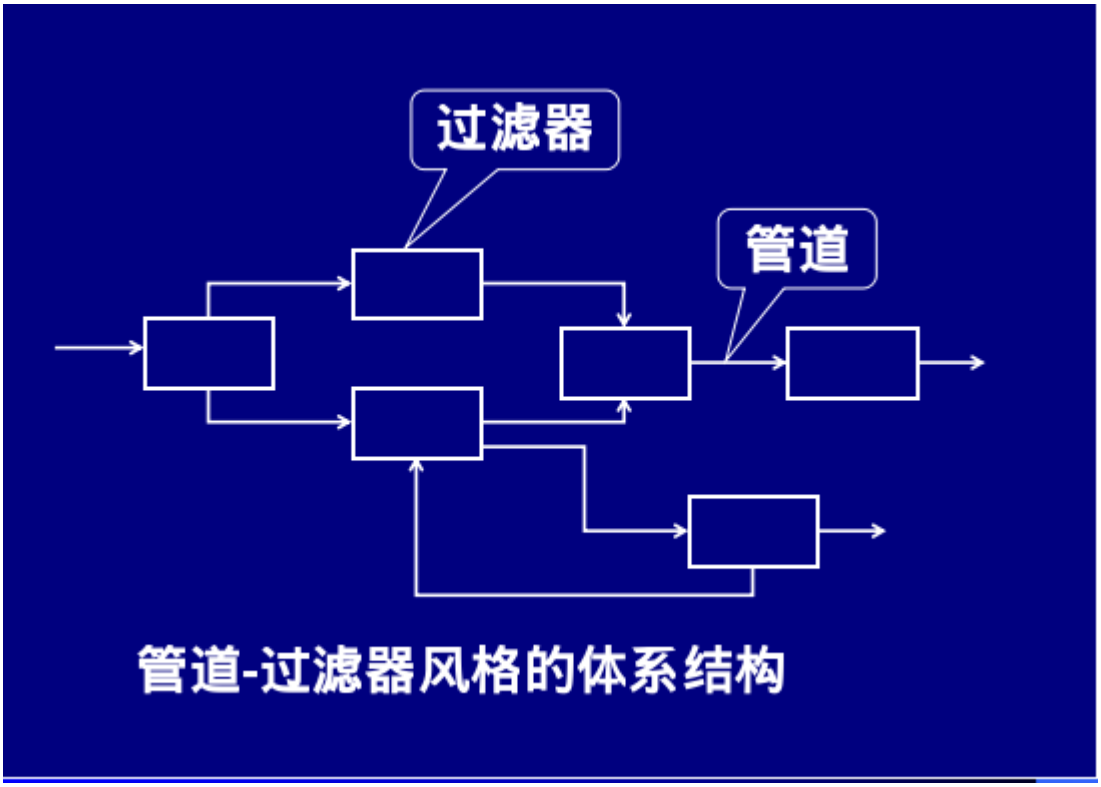
1.SA定义

- 一般意义上，SA模式=SA风格，没有本质区别，因此可以从理解SA模式来理解SA风格。
- 模式——抽取重复问题，描述解决方案的核心。
- SA模式：意图+上下文+问题+解决方案
- 两者也有少许区别：
  - SA风格描述系统总体框架，而SA模式描述得更广
  - SA风格相对独立，而模式彼此依赖性较强
  - SA风格侧重抽取总体结构，模式更面向问题

2.经典风格的优缺点、应用场景

(1)管道-过滤器

- 过滤器——过滤器接收数据输入，进行转化后输出。
- 管道——负责连接各个过滤器，负责它们之间的交互。
- 有三种常见变种:
  1. 流水线模式——批处理、Unix shell、编译器;
  2. 有界管道——限制数据量(带宽);
  3. 类型管道——限制数据类型通过。



- 优点：
  1. 软构件具有隐蔽性、高内聚、低耦合★
  2. 支持并行执行★
  3. 易于理解——系统行为=多个过滤器的行为
  4. 支持软件重用
  5. 易维护、增强系统性能简单(体现在增删中)
  6. 易于分析吞吐量、死锁等属性
- 缺点：
  1. 系统性能下降，且增加编写过滤器的复杂性★
  2. 导致进程成为批处理的结构
  3. 不适合处理交互的应用

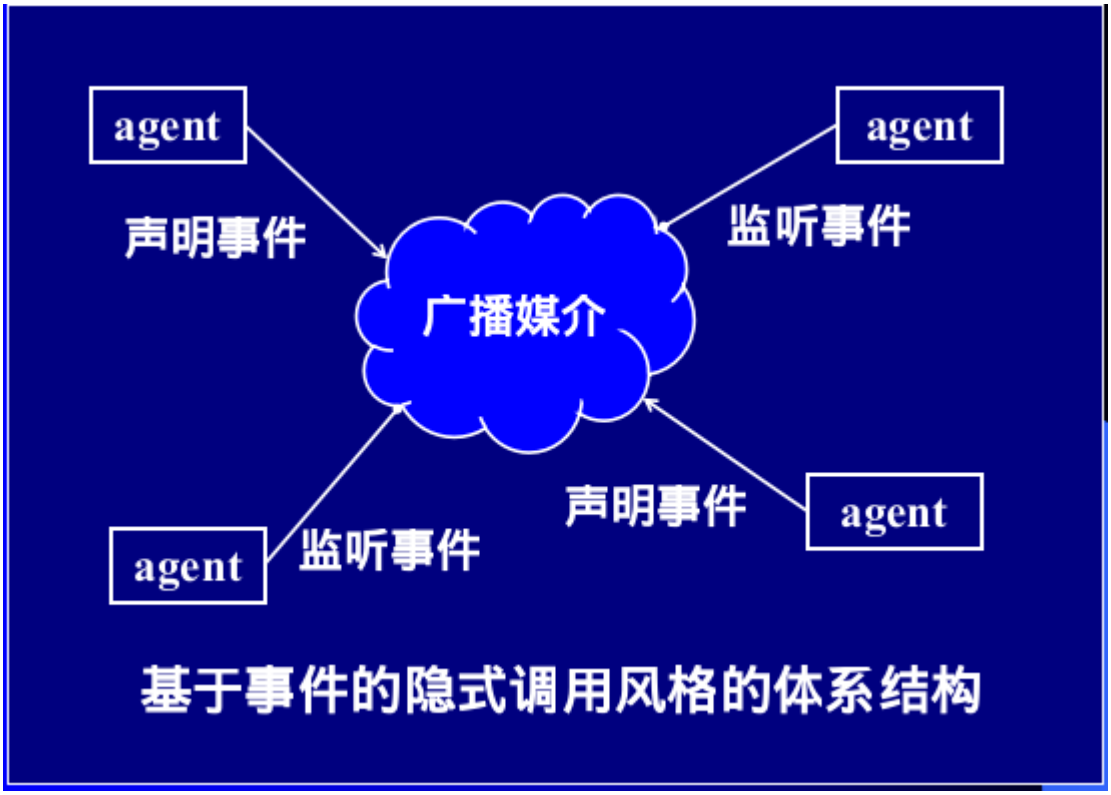
(2)数据抽象和面向对象风格

- 这种风格建立在数据抽象和面向对象的基础上。

- 优点：
  1. 易分解——可将数据存取操作分解为一些交互的代理程序的集合 ★
  2. 易维护——改变一个对象内部的表示，不会影响其他对象(封装)
- 缺点：
  1. 过程调用依赖于对象标识的确定★
  2. 不同对象的操作关联性弱，尤其是发生在多个对象同时访问一个对象时。

(3)基于事件的隐式调用风格

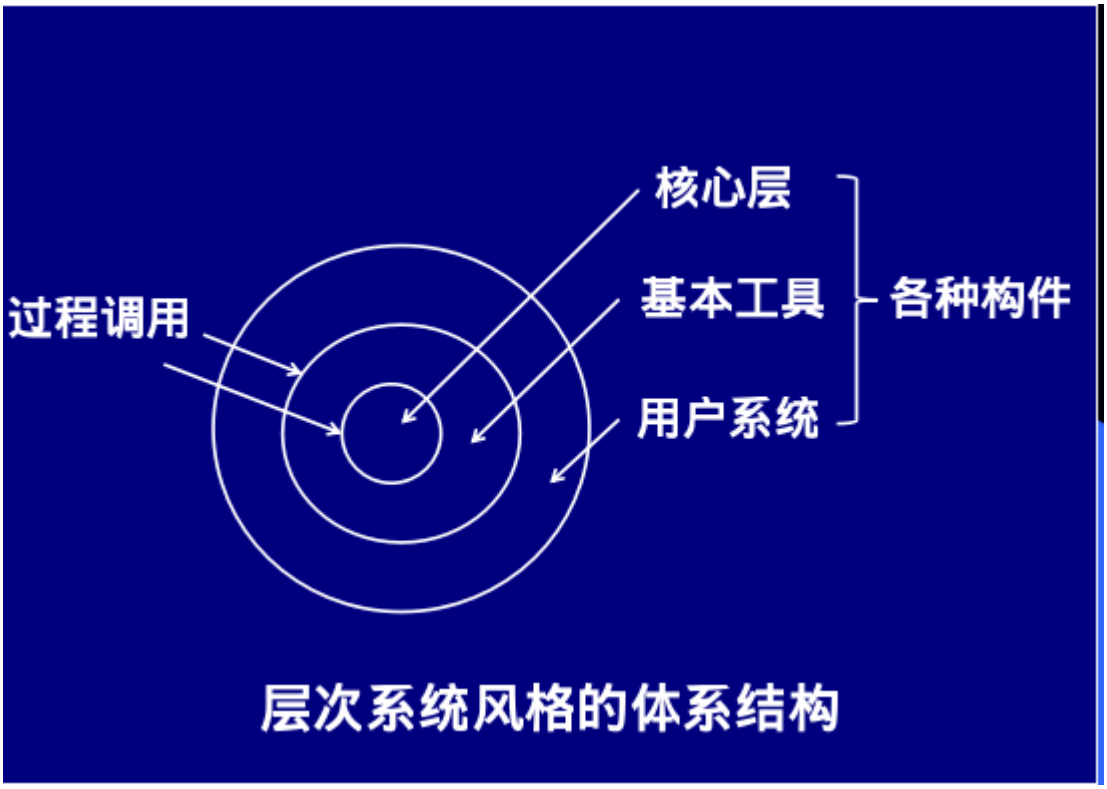
- 构件不直接调用一个过程，而是触发或广播一个或多个事件。
- 一个事件的触发=>另一个模块中所有过程的调用
- 构成要素：
  - 构件——是一些模块，既可以是一些过程，又可以是一些事件的集合。
  - 连接件——触发信息、广播信息
  - 约束——触发模块与过程执行模块间的联系规则



- 优点：
  1. 为软件重用提供了强大的支持★
  2. 为改进带来了方便。
- 缺点：
  1. 构件放弃了对系统计算的控制★
  2. 数据交换成问题(对于利用共享数据进行交互不利)
  3. 很难对系统的正确性进行推理(需要人的参与)
- 应用领域：
  1. 编辑器中支持语法检查——触发报警★
  2. 数据库系统中用于检查一致性约束条件等

(4)层次系统风格

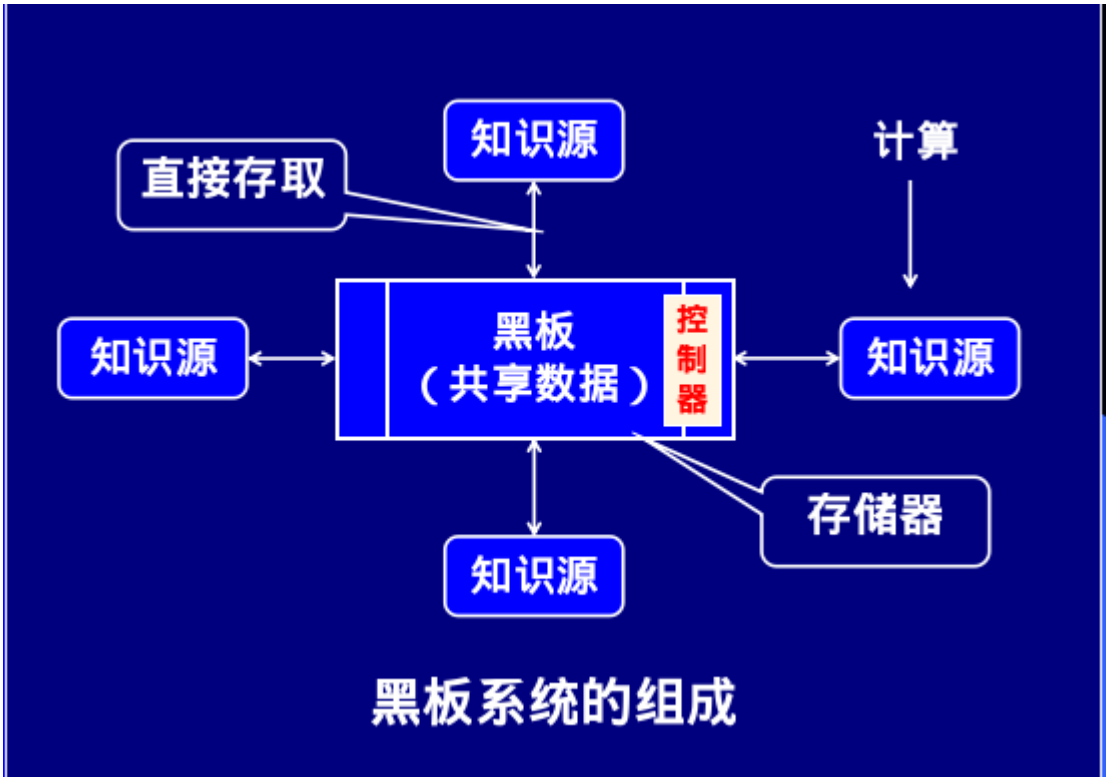
- 每一层为它的上层提供服务，并作为下层的客户，不能跨层调用。
- 由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样*为软件重用提供了强大的支持*。



- 优点：
  1. 支持基于抽象程度递增的系统设计★
  2. 支持重用★
  3. 支持功能增强，改实现不改接口
- 缺点：
  1. 一些系统不易被分层★
  2. 很难找到一个合适的、正确的层次抽象方法
- 应用领域：
  1. 分层通信协议(OSI-ISO)
  2. OS
  3. 数据库系统
  4. TCP/IP协议栈

(5)仓库风格和黑板风格

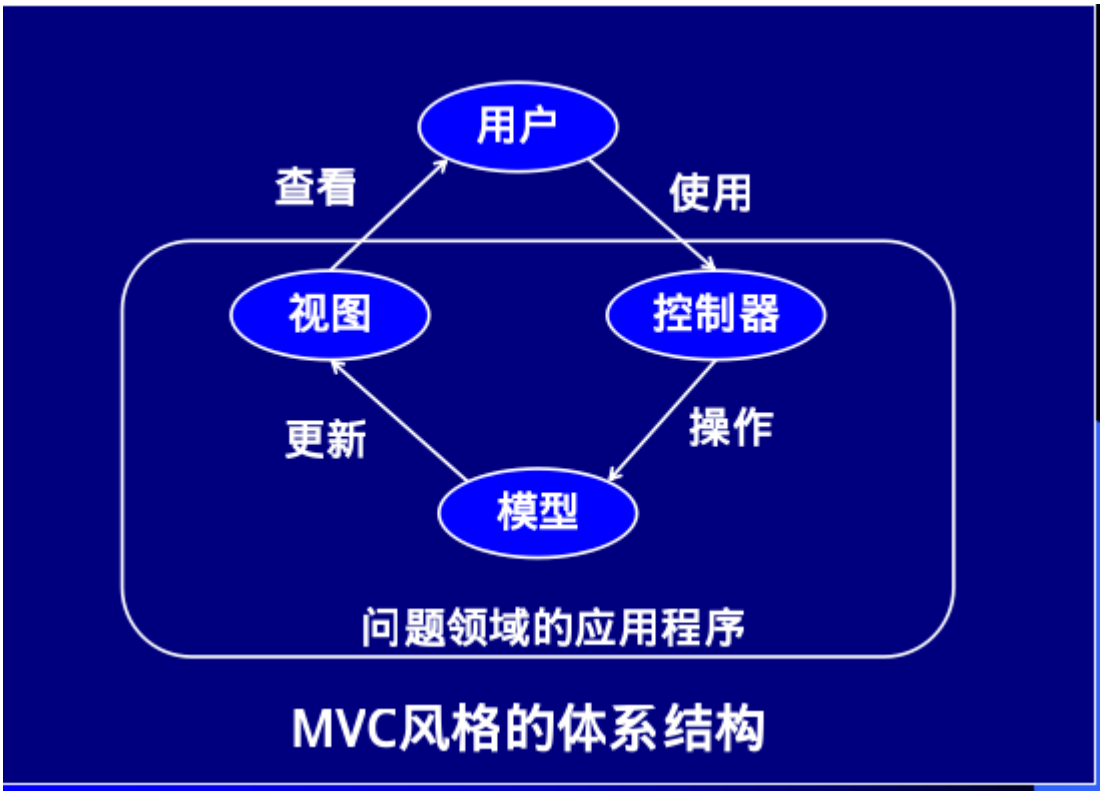
- 两种构件组成：
  - 一个中央数据结构——当前状态
  - 一批独立构件的集合——与中央数据结构交互
- 两类控制策略：
  1. 传统数据库(12306)
  2. 黑板体系结构(BBS)



- 优点：
  1. 便于多客户共享大量数据★
  2. 添加知识源和扩展黑板数据结构都很方便
- 缺点：
  1. 要保证数据结构的完整和一致——系统复杂度增加(同步/加锁)★
  2. 不同知识源代理对共享数据结构要一致——修改难(通过安装控件保持更新)

(6)模型-视图-控制器风格MVC

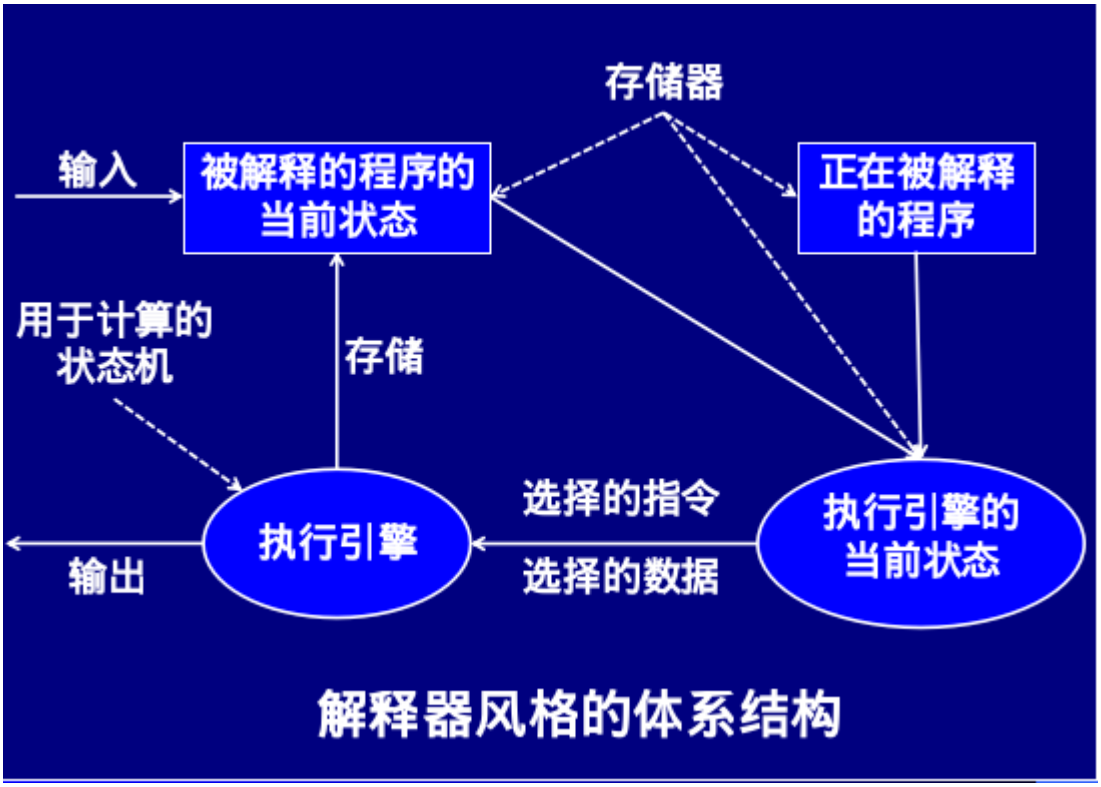
- MVC风格将交互式应用划分为3种构件：
  1. 模型(Model)
  2. 视图(View)
  3. 控制器(Controller)
- 最重要的一点：用户只操作控制器构件，看视图构件的显示，不去碰模型构件里面的应用程序核心代码。



- 优点：
  1. 改变界面影响小，易于演化，可维护性好★
  2. 分而治之，简化设计，保证了可扩展性
  3. 易于改变，动态机制良好
- 缺点：
  - 仅局限在应用软件的用户界面开发领域中★
- 应用领域：
  1. Windows应用程序的文档视图结构
  2. Java Swing应用程序等

(7)解释器风格

- 创建了一个软件虚拟出来的机器。该系统的4个构件：
  - 1个状态机——执行引擎
  - 3个存储器
    - 正在被解释的程序
    - 被解释的程序的当前状态
    - 执行引擎的当前状态



- 优点：
  1. 对未实现的硬件进行仿真★
  2. 提升程序设计语言的跨平台能力
  3. 有助于应用程序的可移植性
- 缺点：
  1. 额外的间接层次带来了系统性能的下降★
- 应用领域：
  1. 虚拟机★
  2. 程序设计语言的编译器
  3. 基于规则的系统
  4. 脚本语言

题目

1. “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”是那种SA风格？

   基于事件的隐式调用风格

2. 采用OO的程序设计语言会导致系统化的重用(X系统化)
3. 不应该将重用只局限在代码构件上(√)

比较

数据流风格	共享数据	抽象数据类型	隐式调用	管道-过滤器
算法变更	--	-	+	+
数据表示变更	--	++	-	--
功能变更	+	-	++	+
性能	++	-（*）	--	--
重用	--	++	-	+

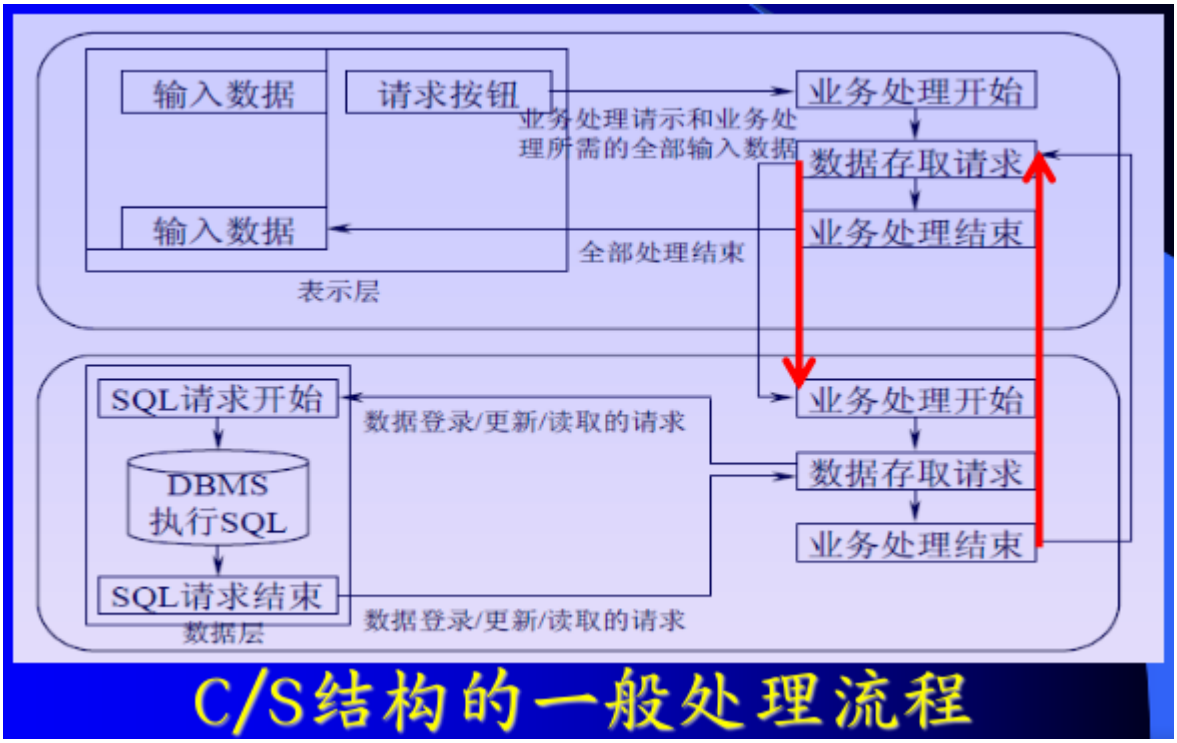
3.B/S 架构、C/S两层、三层架构

(1)B/S风格

- WWW浏览器技术——客户端
- 应用程序以网页形式存放于Web服务器上
- 用户通过键入URL，调用应用程序，对数据库进行操作
- 优点：
  - 浏览器与Web服务器真正的连接时间很短，因此Web服务器可以为更多的用户提供服务。
- 缺点：
  - 数据的动态交互性不强（尤其是一些需要及时响应的，经常交互的）——会“卡”,怎么办？————混合SA，即二层C/S+B/S

(2)二层C/S风格

- C/S体系结构的三个主要组成部分：
  1. 数据库服务器
  2. 客户应用程序
  3. 网络
- 服务器、客户应用程序的主要任务：
  1. 服务器——DB的安全性、控制并发、全局数据完整性规则、备份与恢复
  2. 客户应用程序——提供交互界面、提交用户请求并接收反馈、处理存在于客户端的数据
  3. 网络通信软件——完成数据库服务器和客户应用程序之间的数据传输



- 优点：
    1. 灵活
    2. 可扩展
    3. 通过合理地分解任务，可以使应用成本降低
- 缺点：
    1. 开发成本较高
    2. 客户端程序设计复杂
    3. 软件维护很麻烦

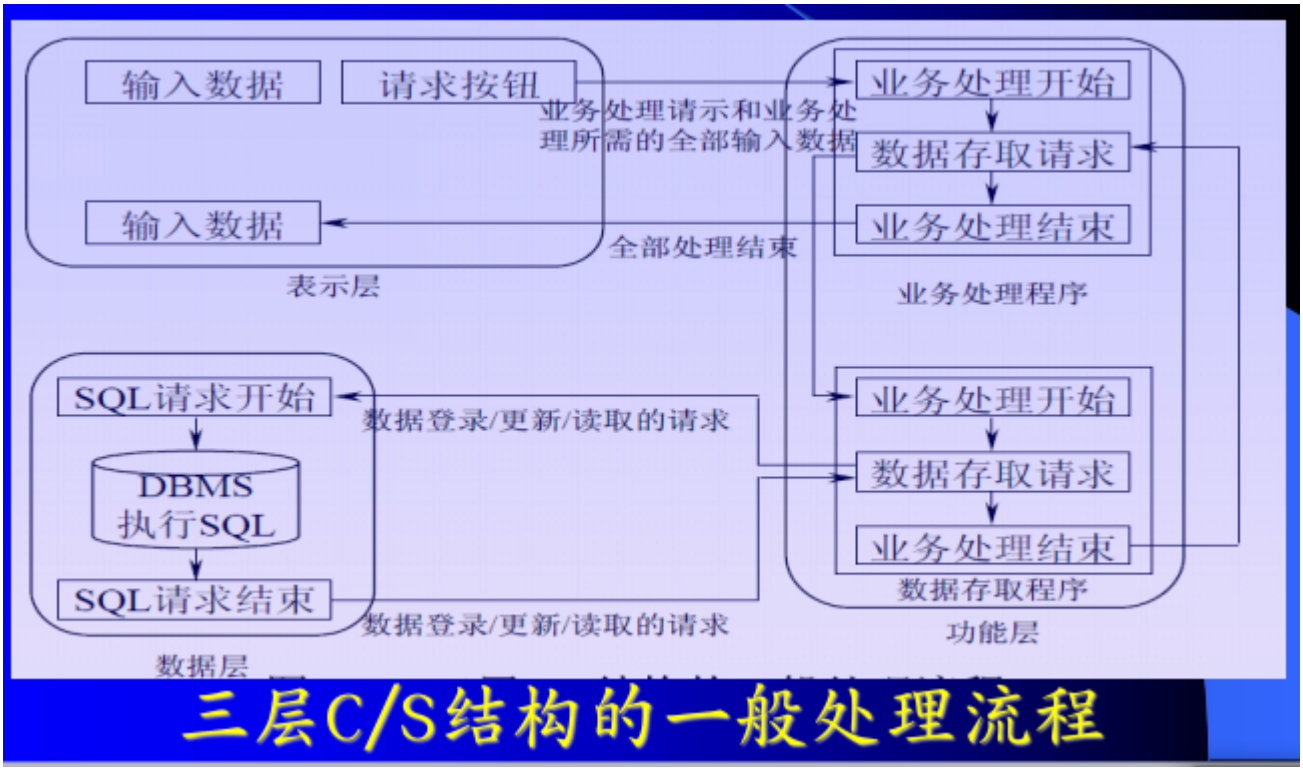
(3)三层C/S风格

①二层**C/S**风格的局限性

- 二层C/S结构以单一服务器、局域网为核心的，难以扩展至大型企业广域网和Internet
- 软硬件结合以及集成能力有限
- 服务器负荷太重，性能低
- 数据安全性差，因为客户端程序可以访问数据库服务器，所以其他程序也可以使用类似的方法访问。

②三层**C/S**风格的三个层次

1. 表示层：应用的用户接口，担负着用户与应用间对话的功能。
2. 功能层：应用的本体，是程序中具体的业务处理逻辑。
3. 数据层：DBMS数据库管理系统，负责对数据库的读写。



③优点

1. 提高系统和软件的可维护性和可扩展性
2. 可灵活选用相应的平台和硬件系统
3. 高效地进行开发，且便于维护
4. 为严格的安全管理奠定了坚实的基础

> 基本上符合软件设计关注的四大质量属性：可修改性、性能、安全性、可用性

4.异构结构风格

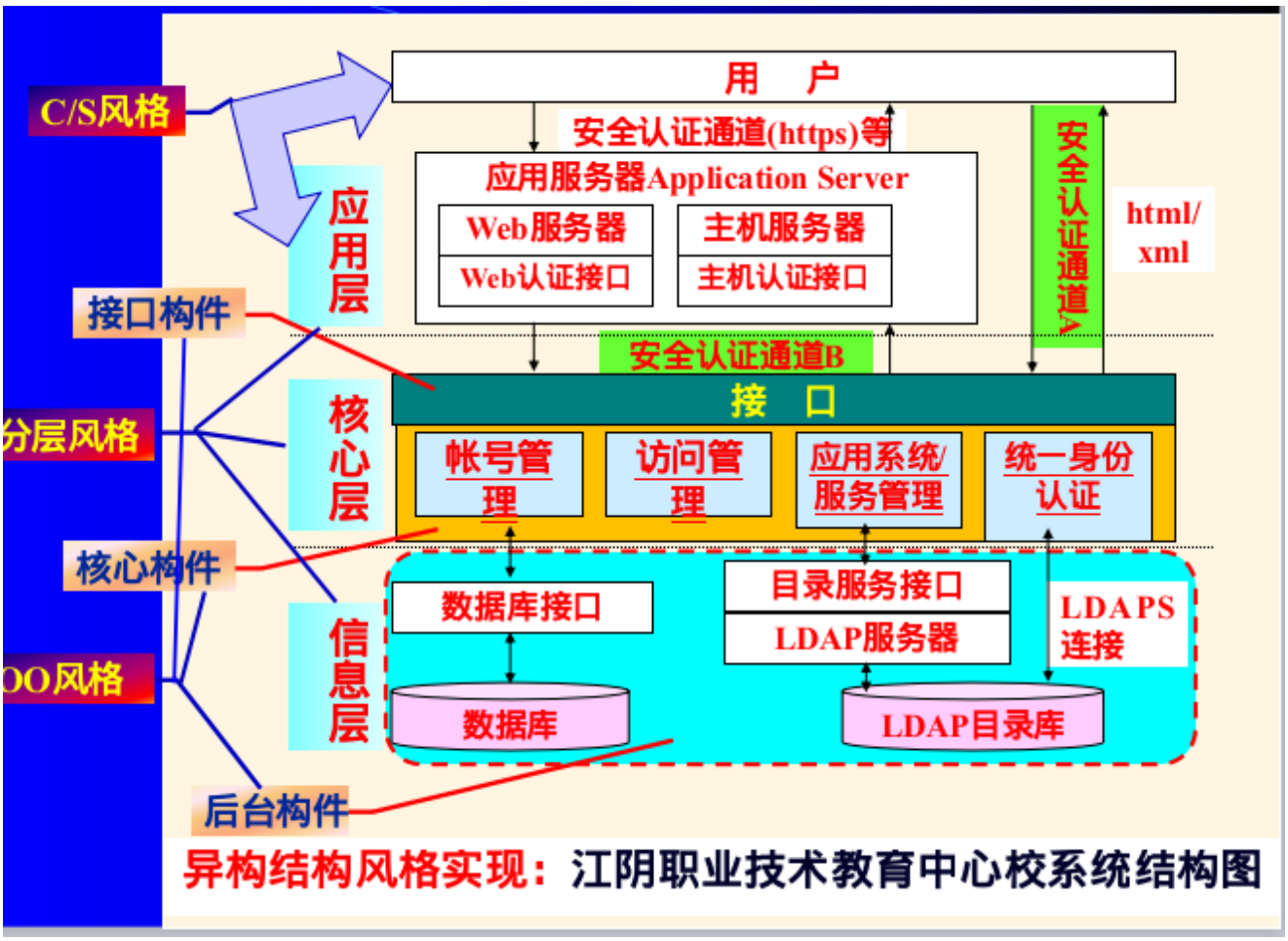
(1)使用异构风格的原因

1. 不同的结构都同时具有优缺点
2. 关于软件包、框架、通信的标准的变动
3. 遗留工程与遗留代码的使用
4. 解释或表示习惯上的不同

(2)典型的异构体系结构——C/S与B/S的混合SA

1. 企业内部C/S结构+企业外部B/S结构(“内外有别”)
2. 修改数据C/S结构+查询数据B/S结构(“改查有别”)

> 都是对安全性和动态交互性能的权衡



四.SA描述

1.SA描述方法及典型描述语言

> SA风格是对经典应用系统的特征抽象,SA描述是对真正软件系统的具体展现

- 目前通用的描述SA的方法——非形式化的图和文本
  - 缺点：
    1. 不能描述系统构件之间的接口
    2. 难于进行形式化分析和模拟
    3. 缺乏相应的支持工具帮助设计师完成设计工作
    4. 不能分析其一致性和完整性等特性

(1)SA的描述方法

- SA的描述方法——软件体系结构描述语言（Architecture Description Language，ADL）
- SA的三个最基本的构成元素=>ADL的三个基本构成要素
  1. 构件：基本计算单元、数据存储单元
  2. 连接件：用来建立构件间的交互，以及支配这些交互规则的体系结构构造模块
  3. 体系结构配置：描述1和2的连接图，提供信息来确定构件是否正确连接、接口是否匹配、连接件构成的通信是否正确，并说明实现要求行为的组合语义——澄清系统结构
- ★好的ADL能够起到承上启下的作用

- 需求分析=>SA设计（ADL）=>软件设计
- SA设计——桥梁
- SA描述——效果图（建模）
- ADL——画效果图的工具（建模工具）
- 一个好的ADL——可以让SA描述方便地转换为其它设计文档，同时可利用需求分析成果直接生成系统的体系结构说明。

(2)典型SA描述语言

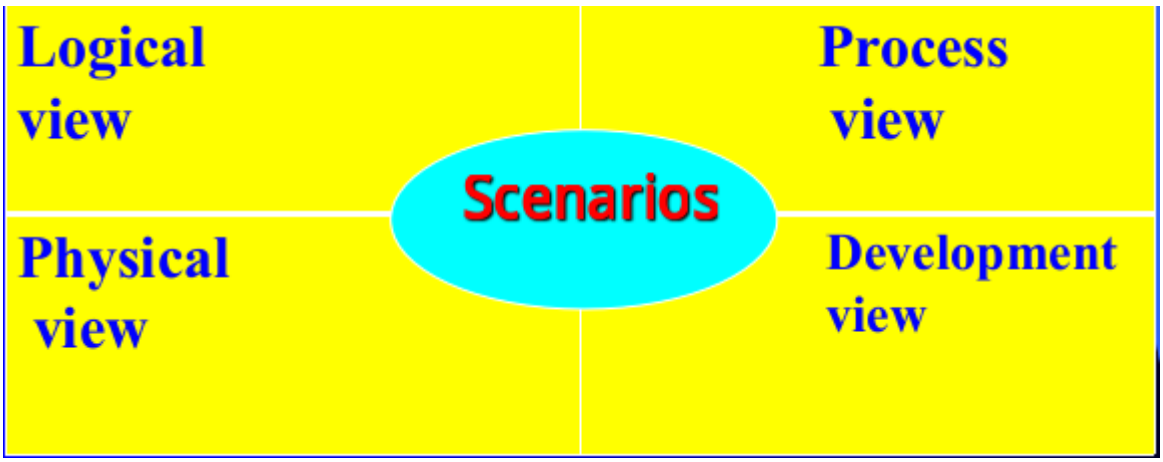
- 主要的SA描述语言

1. UniCon——支持异构的构件和连接件类型，并提供了关于体系结构的高层编译器
2. C2——支持构件重置与GUI重用
3. Wright——连接件=协议，协议刻画构件行为专注于抽象行为的精确表达，以及为架构师提供结构化表达系统信息的方法
4. ACME——是一种交互式ADL，它旨在为开发工具与环境提供交互格式。

> 总之，这些ADL都强调了体系结构的不同侧面，大部分ADL都是与领域相关，不利于对不同领域的体系结构进行分析。多种ADL并存的局面正在演化为几种通用的ADL。

2. 4+1模型

(1)组成



(2)作用

①4+1模型

- 使得有不同需求的人员能够得到他们想要了解的SA的某一方面。

②逻辑视图

- 当采用面向对象的设计方法时，逻辑试图即是对象模型。

③过程视图

- 描述系统的并发和同步方面的设计。

④物理视图

- 描述软件与硬件之间的映射关系，反映系统在分布方面的设计。

⑤开发视图

- 描述软件在开发环境下的静态组织结构。

⑥场景视图

- 场景是用例的集合，这些用例对SA加以说明。

(3)关注点

- 逻辑视图

  SA的组成以实现功能需求，即提供什么样的服务（OO的分解：子系统、模块名称）

- 过程视图

  非功能需求，过程分解（并发、进程、通讯、容错...）

- 物理视图

  从软件到硬件的映射（硬件部署）

- 开发视图

  子系统组件的类型（开发所得部件的显示）

- 场景视图

  汇总，用例的实例（软件产品对架构的需求）

3.题目

1. 在“4+1”视图模型中，__的体系结构关注的是从软件到硬件的映射；__描述了系统的动态结构A.逻辑视图 B.开发视图 C.进(/过)程视图 D.场景视图 E.物理视图

   (E,C)

2. 开发人员要从数以万计的小构件中选取要重用的构件(之前有过，X)
3. 由于考虑到提升通用性，所以被重用的代码构件速度太低(√)
4. 只要建立可重用构件库就会促使构件重用(X)

4.XML

(1)与HTML的区别

- 目标 :HTML的设计目标是显示数据，焦点是数据外观，而XML的设计目标是描述数据，焦点是数据的内容，它的显示形式靠CSS或XSL帮助完成。
- 语法：HTML的标记不是所有的都需要成对出现，XML则要求所有的标记必须成对出现；HTML标记不区分大小写，XML则大小敏感，即区分大小写。
- 更新：XML允许粒度更新，不必在XML文档每次有局部改变时都发送整个文档的内容，只有改变的元素才必须从服务器发送到客户机，而HTML却不支持这样的功能
- 标签的定义：XML的标记由架构或文档的作者定义，并且是无限制的。HTML的标记则是预定义的;HTML 作者只能使用当前 HTML 标准所支持的标记。

(2)语法

	HTML的语法（较随意）	XML的语法(简明)
有起始标签是否必须出现结束标签/关闭标签	未必，例如HTML中的标签	是
标签大小写意义是否相同	相同	不同
标签嵌套顺序是否不能错	无所谓	是
是否必须要有根元素	无所谓	是
属性值是否必须加“”	加不加都可以	是
空格是否被保留	裁减为一个空格	均被保留
所有的特性是否必须有值	未必，如nowrap特性	是

(3)XML语法找错

- `</vendor>`大小写不对
- 第五行没有关闭标签`</Name>`
- `</Price>`位置不对

五.SA设计

1.设计原理

(1)耦合和内聚原理★★★

- 模块间的耦合度是指模块之间的依赖关系，包括控制关系、调用关系、数据传递关系。

- 内聚性是模块内部的相互依赖程度
- 耦合度等级划分：

耦合度	描述
非直接耦合	两模块之间的联系完全是通过主模块的控制和调用来实现的
数据耦合	一个模块访问另一模块，通过简单数据参数来交换信息
标记耦合	一组模块通过参数表传递记录信息，这个记录不是简单变量
控制耦合	一个模块通过传递开关、标志、名字等控制信息控制另一模块
外部耦合	一组模块都访问同一全局简单变量
公共耦合	一组模块都访问同一个公共数据环境
内容耦合	一个模块直接修改另一模块的数据，或直接转入另一个模块

- 内聚度等级划分：

内聚类型	描述
功能内聚	完成一个单一功能，各部分协同工作，缺一不可
顺序内聚	处理元素相关，必须顺序执行
通信内聚	所有处理元素集中在一个数据结构的区域上
过程内聚	处理元素相关，必须按照特定的次序执行
瞬时内聚	所包含的任务必须在同一时间间隔内执行(如初始化模块)
逻辑内聚	完成逻辑上相关的一组任务
偶然内聚	完成一组没有关系或松散关系的任务

> 习题
>
> 模块间的耦合度是指模块之间的依赖关系，其中不包括哪个关系？D
>
> A.控制关系　　B.调用关系　　C.数据传递关系　　D.继承关系
>
> 耦合度最高的是内容耦合，应该尽量采用。(X尽量避免)
>
> 使用设计模式，将不用再考虑软件体系结构的现有的解决方案，因为模式自身就包含解决方案。(X)
>
> SA设计的重要性体现在它包括了早期的设计决定，体现了系统的全局结构，极大地影响整个系统的质量。(√)

(2)抽象原理★★

- 抽象是人们透过事务繁杂的表面现象，揭示事物本质特征的方法，也是软件体系结构设计中要用到的基本原理。
- 抽象可分为两类：

1. 过程抽象：任何一个具体的操作序列，若他们完成一项逻辑意义上的功能，则其使用者都可以把它看做一个单一的逻辑概念。
2. 数据抽象：将数据类型和施加于该类型对象的上的操作作为整体来定义，并限定了对象的值只能通过使用这些操作修改和观察。

- 抽象是封装的基础，有助于处理系统复杂性，有助于减少部件耦合，有助于接口与实现分离。

(3)封装原理★★

- 封装是将事物的属性和行为结合在一起，并且保护事物内部信息不受破坏的一种方式。封装使不同的抽象之间有了明确的界限。
- 封装有助于非功能特性的实现，例如可变性和可重用性。
- 封装由两方面组成：
  1. 内部构成
  2. 操作服务
- 封装与信息隐藏之间有着密切的联系，封装为信息隐藏提供了支持。

(4)信息隐藏原理★★

- 信息隐藏对用户隐藏了部件的实现细节。因此，可以用来更好的处理系统的复杂性和减少各模块之间的耦合。

(5)模块化原理★

- 模块化主要关心的是如何将一个软件系统分解成多个子系统和部件。
- 在体系结构设计中，如果注重了模块化的概念就可以限制更改设计所造成的影响范围。

(6)注意点分离原理★

- 不同的和无关联的责任应该出现在系统不同的部件中，让他们相互独立的分离开来。
- 相互协作完成某一个特定任务的部件也应该和在其他任务中执行的计算部件分离开来。
- 如果一个部件在不同的环境下扮演着不同的角色，在部件中这些角色应该独立且相互分离。

(7)接口和实现分离原理★★

- 在软件体系结构中，任何一个部件都包括了两个部分：接口和实现。
  - 接口部分给出了部件所提供的功能的定义，并对功能的使用方法进行了规范。部件的客户可以访问接口。
  - 实现部分包括了实际代码，对部件的客户不可用。
- 接口和实现分离的部件更容易在系统中进行改变，尤其是不影响接口的改变，比如性能的提升。

(8)分而治之原理★

- 将大问题分解成小问题，经常用作注意点分离原理的方法。(横向分割)

(9)层次化原理★

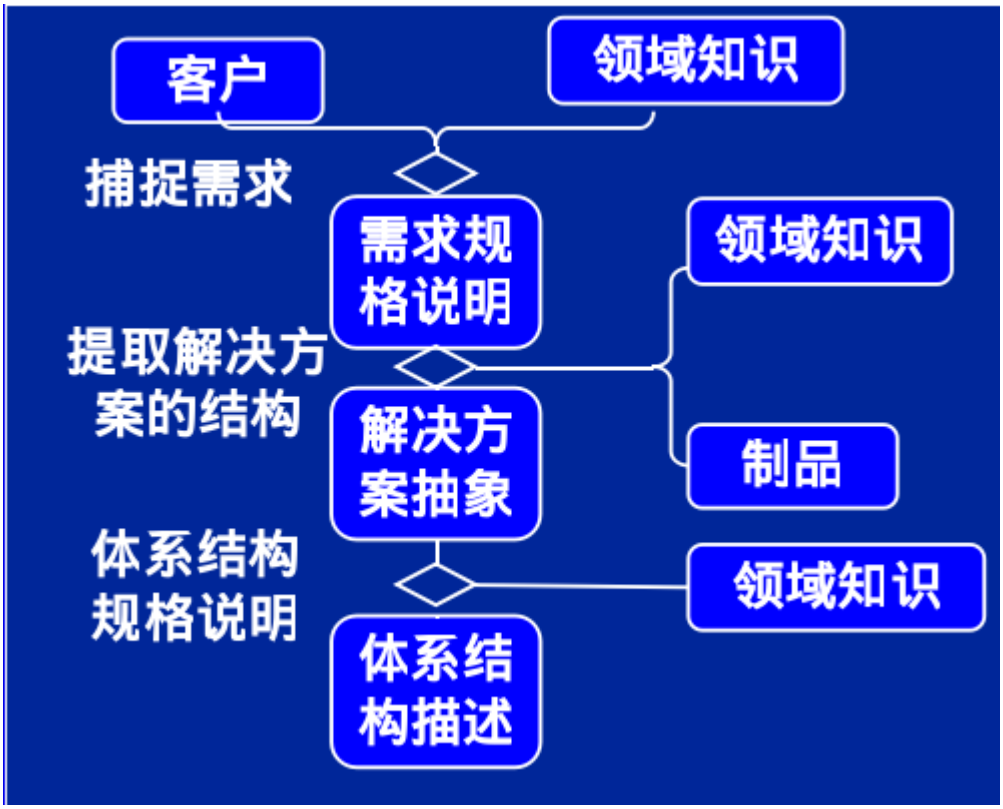
- 纵向分割复杂的问题。原理和优缺点同层次系统风格。

> 题目
>
> 一个模块内包含的信息，对于不需要这些信息的模块是不能访问的，这体现了多个设计原理的共同要求。(√)

> 耦合度最高的是内容耦合，应该尽量采用。(X)
>
> 对接口和实现分离原理的贯彻，可以保证用户在使用方式与习惯不变的情况下，享受运行性能的提升。(√改实现不改接口)

2.元模型

- 元模型是对各种体系结构设计模型的抽象。



- 需求规格说明用来表示规格说明，该规格说明描述了所要开发的体系结构的系统需求。
- 解决方案抽象定义了体系结构中(子)结构的概念表示。
- 体系结构描述定义了软件体系结构的规格说明。

> 题目
>
> 各种不同的体系结构设计方法都可以描述成元模型的实例，只是在过程的顺序及概念的特定内容上有所不同。(√)
>
> 元模型中，需求规格说明关注SA中子系统和模块的需求，而解决方案抽象关注SA的系统需求。(X相反)需求规格说明关注SA的系统需求，解决方案抽象关注SA中子系统和模块的需求。
>
> 用例的使用在IBM RUP中被推崇备至，整个RUP流程都被称作是“用例驱动”(Use-Case Driven)的。(√)

六.基于SA的软件开发过程

1.SA设计在开发过程中的位置地位

- 软件体系结构设计是软件需求与软件设计之间的桥梁。

2.F16例子对导出SA需求有什么借鉴

1. 分析体系结构需求的三个来源，即系统的质量目标、系统的业务目标、将在该系统上工作的人员的业务目标（人员目标很重要）。
2. 需求来自于许多系统相关人员，尤其是直接使用者。
3. 应综合考虑质量属性（需求要具体）。

3.瓦萨号

- 导出体系结构需求——
  - 体系结构需求的三个来源——质量目标★、业务目标、人员目标
  - 特定质量场景
  - 系统相关人员中应有评估、监理专家团队
- 进行SA设计——
  - 体系结构的构造（功能构造）
  - 验证——首选质量场景
  - 开发过程分析

> 基于体系结构的软件开发过程六个步骤：
>
> 1. 导出体系结构需求
> 2. 设计体系结构
> 3. 文档化体系结构
> 4. 分析体系结构
> 5. 实现体系结构
> 6. 维护体系结构
>
> 2 、 3 、 4步可以重复执行
>
> 每个步骤都包括：输入（包括收集此信息的手段）、验证活动、输出。
>
> 瓦萨号的例子告诉我们——
>
> 妄想实现所有需求，只会产生脆弱的一无是处的架构。另外，比起仅仅满足客户所要求的功能，软件的成功更为重要。
>
> 举例说明——设计出来的新系统虽然满足了技术上的规范，但并没有达到客户可接受的程度。用户总是抱怨用户界面运行缓慢，并且新的数据文件所占用的磁盘空间太大。是什么导致了用户期望与产品实际性能之间的期望差异？ ——对质量属性的权衡取舍工作没做好

七.SA评估

1.质量属性

- 产品的易用程度，执行速度，可靠性；当发生异常情况时，系统如何处理。这些被称为软件质量属性(或质量因素)的特性，是系统非功能（也叫非行为）部分的需求。
- 质量属性——是一个组件或一个系统的非功能性特征，提供了测量和分析质量的上下文。
- 具体的质量属性包括了性能、可靠性、可用性、安全性、可修改性、功能性、可变性、可集成性、互操作性等等。

> 对各质量属性的区分（用户）

- 有效性——指的是在预定的启动时间中，系统真正可用并且完全运行时间所占的百分比
    - 工作日期间，在当地时间早上6点到午夜，系统的有效性至少达到9 9 . 5 %，在下午4点到6点，系统的有效性至少可达到9 9 . 9 5 %
- 效率——是用来衡量系统如何优化处理器、磁盘空间或通信带宽的（ Davis 1993）。如果系统用完了所有可用的资源，那么用户遇到的将是性能的下降，这是效率降低的一个表现。
    - 在预计的高峰负载条件下， 1 0 %处理器能力和1 5 %系统可用内存必须留出备用。
- 灵活性——就像我们所知道的可扩充性、增加性、可延伸性和可扩展性一样，灵活性表明了在产品中增加新功能时所需工作量的大小。
    - 一个至少具有6个月产品支持经验的软件维护程序员可以在一个小时之内为系统添加一个新的可支持硬拷贝的输出设备。
- 完整性——完整性（或安全性）主要涉及：防止非法访问系统功能、防止数据丢失、防止病毒入侵并防止私人数据进入系统。
    - 只有拥有查账员访问特权的用户才可以查看客户交易历史。
- 互操作性——表明了产品与其它系统交换数据和服务的难易程度。
    - 化学制品跟踪系统应该能够从C h e m i D r a w和C h e m - S t r u c t工具中导入任何有效化学制品结构图。
- 可靠性——是软件无故障执行一段时间的概率（Musa, Iannino and Okumoto 1987）。健壮性和有效性有时可看成是可靠性的一部分。
    - 由于软件失效引起实验失败的概率应不超过5‰
- 健壮性——指的是当系统或其组成部分遇到非法输入数据、相关软件或硬件组成部分的缺陷或异常的操作情况时，能继续正确运行功能的程度。
    - 所有的规划参数都要指定一个缺省值，当输入数据丢失或无效时，就使用缺省值数据。
- 可用性——也称为“易用性”和“人类工程”，它所描述的是许多组成“用户友好”的因素。可用性衡量准备输入、操作和理解产品输出所花费的努力。
    - 一个培训过的用户应该可以在平均3分钟或最多5分钟时间以内，完成从供应商目录表中请求一种化学制品的操作。

对各质量属性的区分（开发者和维护者）

- 可维护性——表明了在软件中纠正一个缺陷或做一次更改的简易程度。可维护性取决于理解软件、更改软件和测试软件的简易程度，可维护性与灵活性密切相关。
    - 在接到来自联邦政府修订的化学制品报告的规定后，对于现有报表的更改操作必须在一周内完成。
- 可移植性——是度量把一个软件从一种运行环境转移到另一种运行环境中所花费的工作量。
    - 该应用软件系统各个模块都可以在x86机器上运行，且其中的大部分功能模块也可以在iMac机器上运行，剩下的少量功能模块经修改后，也可在iMac机器上运行。
- 可重用性——从软件开发的长远目标上看，可重用性表明了一个软件组件除了在最初开发的系统中使用之外，还可以在其它应用程序中使用的程度。
    - 设计单机版媒体播放器系统时，还要保证核心的播放模块可以在将来开发的联机版媒体播放器系统中使用。
- 可测试性——指的是测试软件组件或集成产品时查找缺陷的简易程度。

> - 随着图形引擎功能的不断增强，我们需要对它进行多次测试，所以作出了如下的设计目标：“一个模块的最大循环复杂度不能超过2 0。”

2.几种评估方式以及各自的特点

1. 基于调查问卷、检查表
   - 自由灵活，但因人而异——主观性强
2. 基于场景★★
   - SEI at CMU的评估方法
3. 基于度量★
   - 能提供更为客观和量化的质量评估，不过需要在SA设计基本完成之后进行——较客观

- 评估过程=论证SA对关键场景的支持程度

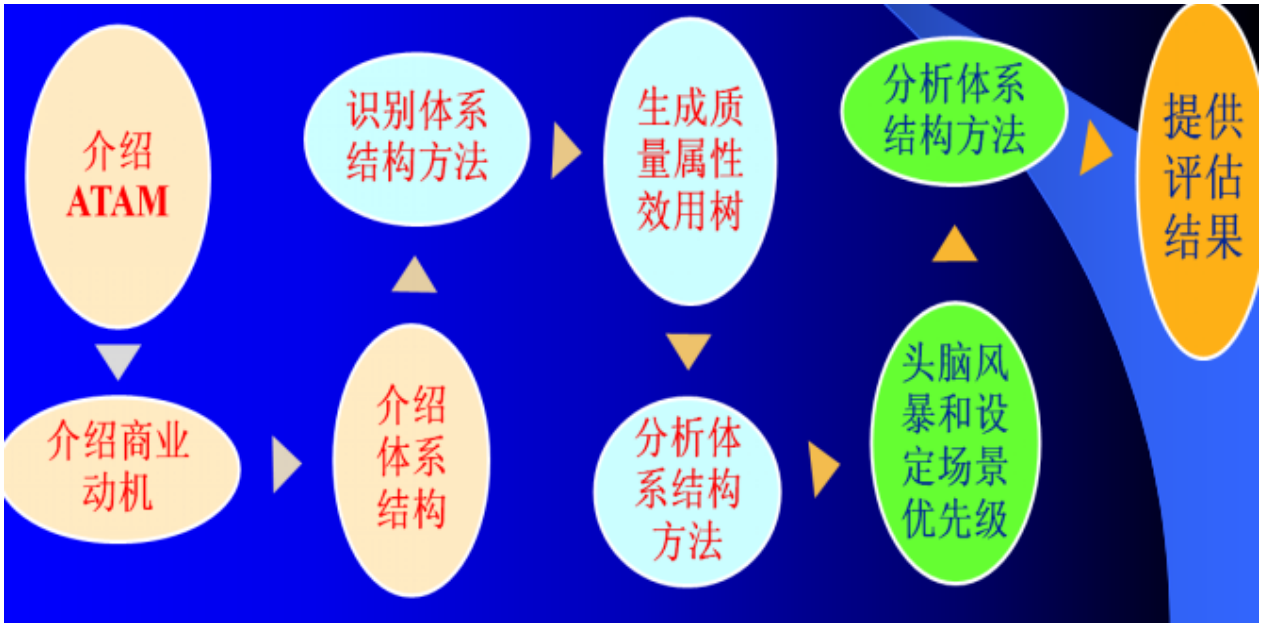
3.SAAM/ATAM

(1)SAAM软件体系结构分析法

- SAAM法仅仅考虑场景和体系结构的关系，也不涉及太多的步骤和独特的技术。是一种理想的入门方法。
- SAAM法实质上是对检查表法的进一步完善，它首先把要检查的事物的属性列举出来，再对各个属性进行"检查"，使思路更广，目标更明确。
- SAAM分析评估体系结构的过程包括6个步骤:
  1. 通过风险承担者协商讨论，开发一些任务场景，体现系统所支持的各种活动。(场景生成)
  2. 用一种易于理解的、合乎语法规则的体系结构描述SA，体现系统的计算构件、数据构件以及构件之间的关系。(体系结构描述)
  3. 区分直接场景（不需要修改SA的场景）和间接场景，设置优先级（从重要到次要）。(场景的分类和优先级确定)
  4. 对于间接场景，要估计修改SA以实现它的代价。(间接场景的单独评估)
  5. 通过对场景交互（多个间接场景要求更改同一个组件）的分析，能得出系统中所有场景对系统中的构件所产生的影响的列表。(对场景关联的评估)
  6. 对场景和场景间的交互作一个总体的权衡和评价，设置权值，得出总体评价。(形成总体评估)

(2)ATAM体系结构权衡分析法

- ATAM是评价软件构架的一种综合全面的方法。这种方法不仅可以揭示出构架满足特定质量目标的情况，而且（因为它认识到了构架决策会影响多个质量属性）可以使我们更清楚地认识到质量目标之间的联系——即如何权衡诸多质量目标。
- 体系结构的敏感点和权衡点
  - 敏感点：会被某些体系结构元素显著影响的系统模型的属性值。
  - 权衡点：系统内与几个敏感点都相关的地方。
- ATAM的评估步骤


介绍
ATAM
介绍商业
动机
介绍
体系
结构
识别体系
结构方法
生成质
量属性
效用树
分析体
系结构
方法
头脑风
暴和设
定场景
优先级
分析体系
结构方法
提供
评估
结果