全国计算机技术与软件专业技术资格（水平）考试

2025年5月系统架构设计师考试冲刺串讲

讲师：邹月平

系统架构设计师2025年上考试时间

题型	数量	考试时间（5月24日）	做题时间	分数
单选题	75题	8:30-12:30	240分钟	75分
案例题	1+4选2 共3题			75分
论文	4选1	14:30-16:30	120分钟	75分

1 系统架构设计师——计算机基础

系统工程

1、霍尔的三维结构。适合大型项目

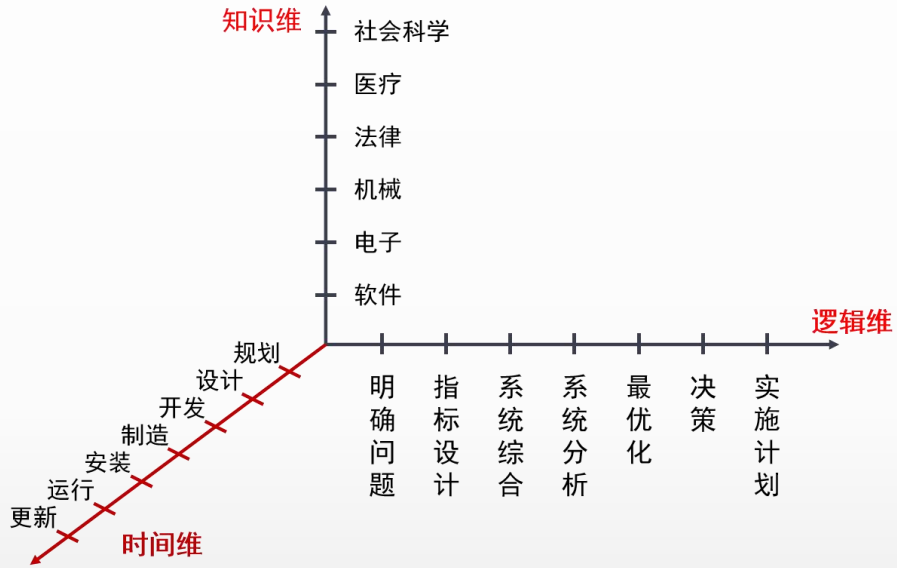
2、切克兰德方法有7个步骤：认识问题、根底定义、建立概念模型、比较及探寻、选择、设计与实施

、评估与反馈。

3、并行工程方法。目标是提高质量、降低成本、缩短产品开发周期和产品上市时间。

4、综合集成方法。定性与定量结合，科学理论与经验知识结合、宏观研究与微观研究统一，必须有大型计算机系统支持，不仅有管理信息系统、决策支持系统等功能，而且还要有综合集成的功能。

5、WSR系统方法：分别代表物理、事理、人理。WSR方法的步骤：理解意图，制定目标，调查分析，构造策略，选择方案，协调关系，实现构想。



霍尔的三维结构

系统工程

- 生命周期方法：

1、计划驱动方法

- 始终遵守规定流程
- 特别关注文档的完整性
- 特别关注需求的可追溯性
- 特别关注每种表示的事后验证

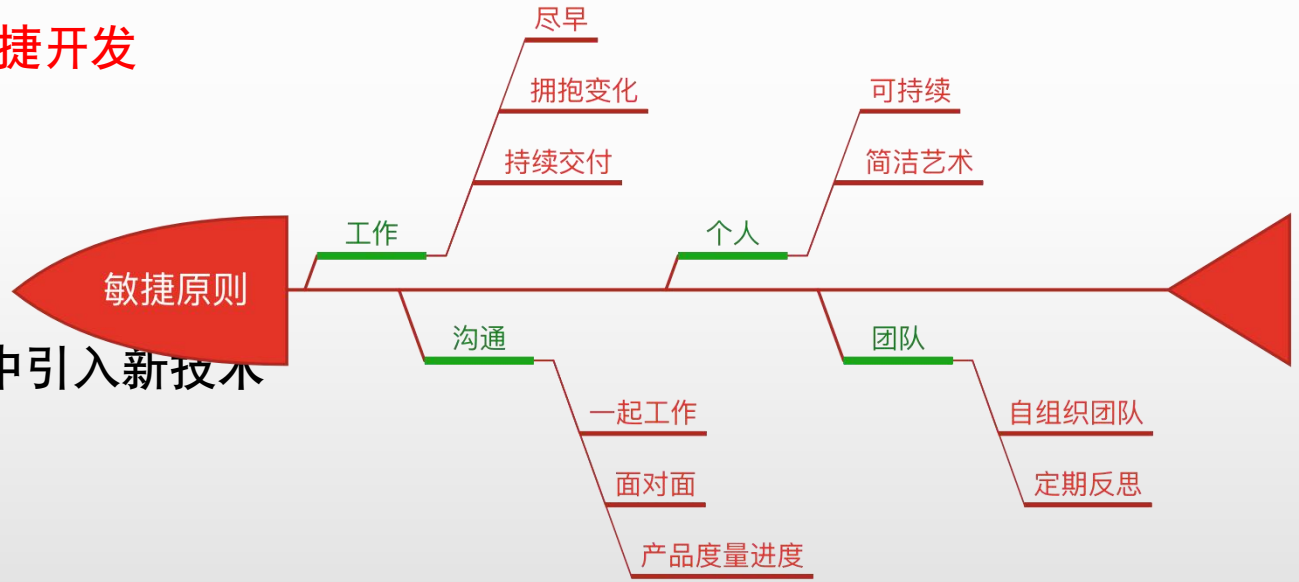
2、渐进迭代式开发

- 需求不清晰、不确定或者客户希望在系统中引入新技术
- 较小的、不太复杂的系统
- 灵活性
- 通过剪裁突出了产品开发的核心活动

3、精益开发

- 动态的、知识驱动的、以客户为中心的过程
- 向客户交付最大价值并使浪费活动最小化

4、敏捷开发



基于模型的系统工程MBSE

- **采用**形式化、图形化、关联化的**建模语言**及相应的**建模工具**，**改造**系统工程的**技术过程**，充分**利用**计算机、信息技术的**优势**，开展建模（含分析、优化、仿真）工作，为系统实现、验证奠定更为坚实的基础，从而**提升**整个研制过程的**效率**。
- MBSE过程的三个阶段产生三种图形。

2 系统架构设计师——操作系统

考点分析

- 进程管理：进程的互斥与同步定义、死锁问题
- 文件管理：文件的直接索引、一级间接索引、二级间接索引、位示图
- 存储管理：分页存储管理
- 设备管理：磁盘的调度

以上历年试题中的经典考题

3 系统架构设计师——通信与网络基础

通信技术

- 信息传输就是信源和信宿通过信道收发信息的过程。信源发出信息，发信机负责将信息转换成适合在信道上传输的信号，收信机将信号转化成信息发送给信宿。
- 在一条信道上只传输一路数据的情况下，只需要经过信源编码、信道编码、交织、脉冲成形、调制之后就可以发送到信道上进行传输了；但如果同时传递多路数据就需要复用技术和多址技术。

1 ）复用技术：是指在一条信道上同时传输多路数据的技术，如TDM时分复用、FDM频分复用和CDM码分复用等。ADSL 使用了FDM 的技术，语音的上行和下行占用了不同的带宽。

2 ）多址技术

多址技术是指在一条线上同时传输多个用户数据的技术，在接收端把多个用户的数据分离(TDMA 时分多址、FDMA 频分多址和CDMA 码分多址）。

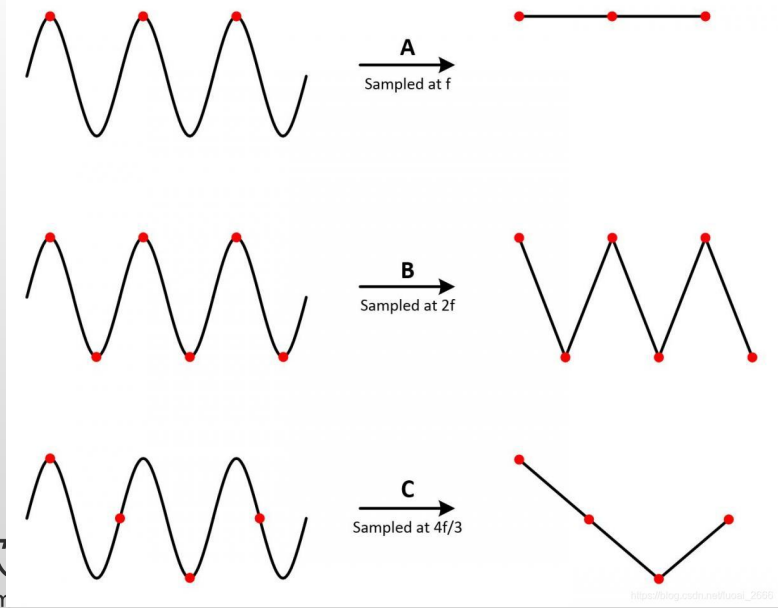
多路复用技术是多址技术的基础，多址技术还涉及信道资源分配算法，Walsh 码分配算法等。

通信技术

- 模拟信号到数字信号的转换过程： 采样---量化---编码
- 波特率（baud rate）：指的是每秒传送的符号或信号元素的数量。
- 比特率（bit rate）：信号每秒钟传输的数据的位数。

也就是每秒钟传输0和1的个数，单位是bps（bit per second）比如网速：100Mbps。

波特率和比特率的关系：一个符号可能表示一个或多个比特。



- 奈奎斯特抽样定理 ：要从抽样信号中无失真地恢复原信号，抽样频率应大于2倍信号最高频率。

网络协议

➢ OSI协议：

层的名称	主要功能	详细说明
应用层	处理网络应用	直接为端用户服务，提供各类应用过程的接口和用户接口。
表示层	数据表示	使应用层可以根据其服务解释数据的涵义。通常包括数据编码的约定、本地句法的转换、数据加密和解密、数据压缩和解压。
会话层	互连主机通信	负责管理远程用户或进程间的通信，通常包括通信控制、检查点设置、重建中断的传输链路、名字查找和安全验证服务。
传输层	端到端连接	实现发送端和接收端的端到端的数据分组传送，负责保证实现数据包无差错、按顺序、无丢失和无冗余的传输。其服务访问点为端口。
网络层	分组传输和路由选择	通过网络连接交换传输层实体发出的数据，解决路由器选择、网络拥塞、异构网络互联的问题。服务访问点为逻辑地址（网络地址）。
数据链路层	传输以帧为单位的信息	建立、维持和释放网络实体之间的数据链路，把流量控制合并在一起。
物理层	二进制为传输	通过一系列协议定义了通行设备的机械的、电气的、功能的、规程的特征。

交换方式

交换方式	特点	适用场景
电路交换	1．需建立专用的物理通信路径；2．延迟较低；3、可能导致资源浪费	适用于数据流量稳定且持续时间较长的应用，如传统的电话通信
报文交换	1、无需预先建立连接；2、报文整体传输；3、网络拥堵时，报文可能在节点中长时间存储，降低效率	适用于早期的电报和某些低速率数据网络
分组交换（主流）	1、数据被分割成小数据包独立传输；2、网络资源共享，提高资源利用率；3、能处理突发流量和不同类型的数据传输需求	适用于现代计算机网络，特别是互联网

4 系统架构设计师——信息安全

安全技术

- 加密技术：

类型	特征	算法
对称加密	又叫私钥加密算法，加密与解密密钥相同。适合大数据量（明文）加密。加密速度快。	DES、3DES、RC-5、IDEA、AES、SM1、SM4、SM7、祖冲之密码算法ZUC
非对称加密	又叫公钥加密算法，加密与解密密钥不同。适合少数据量加密。加密速度慢。身份认证抗抵赖性好。	RSA、ECC椭圆曲线、SM2、SM9

- 消息摘要与数字签名与数字证书

类型	特征
消息摘要	使用单向哈希函数生成摘要发送给接收方。常用算法MD5（密文长度128位）、SHA-256安全散列算法。
数字签名	利用公钥加密对消息摘要签名。其中发送方私钥用于签名而发送方公钥用于验证。
数字证书	用电子手段来证实一个用户的身份和对网络资源的访问权限。由数字证书认证机构CA和签发和管理。CA可以证明数字证书的持有者合法拥有证书中列出的公开密钥。

5 系统架构设计师——数据库基础

关系代数运算

运算符	含义	名词解释
∪	并	关系R与S的并是由属于R或属于S的元组构成的集合。
-	差	关系R与S的差是由属于R但不属于S的元组构成的集合
∩	交	关系R与S的交是由属于R同时又属于S的元组构成的集合。

运算符	含义	名词解释
σ	选择	取得关系R中符合条件的行。
π	投影	取得关系R中符合条件的列组成新的关系。
×	笛卡尔积	两个元组分别为n列和m列的关系R和S的笛卡尔积是一个（n+m）列的元组的集合。元组的前n列是关系R的一个元组，后m列是关系S的一个元组，记作R×S，如果R和S有相同的属性名，可在属性名前加关系名作为限定，以示区别。若R有K_1个元组，S有K_2个元组，则R和S的笛卡尔积有$K_1 \times K_2$个元组。
⋈	连接	自然连接：一种特殊的等值连接，它要求比较的属性列必须是相同的属性组，并且把结果中重复属性去掉。

函数依赖

超键	在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。例如学生关系模式中的（学号）、（身份证号）、（学号，姓名）、（身份证号、性别）等。
候选键	不含有多余属性的超键称为候选键，是超键的最小子集。例如，（学号）、（身份证号）
主键	用户挑选出来做元组标识的一个候选键称为主键。例如，我们通常选择（学号）作为主键。
外键	如果关系模式R中的某些属性集不是R的主键，而是关系模式S的主键，则这个属性集对模式R而言是外键。例如，（教师编号）是学生关系模式的外键。
主属性	含在任何一个候选键中的属性称为主属性，否则称为非主属性。
全码	关系模型的所有属性组是这个关系模式的候选键，称为全码。

规范化

类型	特征
1NF	若关系模式R的每一个分量是不可再分的数据项，则关系模式R属于第一范式。
2NF	若关系模式R∈1NF，且每一个非主属性完全依赖主键时，则关系式R是2NF（第二范式）。
3NF	即当2NF消除了非主属性对主键的传递函数依赖，则称为3NF。
BCNF	R∈3NF，且消除了主属性对候选键的部分和传递函数依赖。

- 第四范式（4NF）：关系模式R∈1NF，若对于R的每个非平凡多值依赖X→→Y且Y⊆X时,X必含有码，则关系模式 R(U,F)∈4NF。4NF是限制关系模式的属性间不允许有非平凡且非函数依赖的多值依赖。

6 系统架构设计师——软件工程

开发模型

瀑布模型	◘ 上一次的开发成果作为本活动输入。利用这一输入实施本活动。 ◘ 本次活动的成果输出给下次活动。 ◘ 对本次活动的成果实施评审。若成功得到确认，则继续下一项开发活动；否则返回前一项，甚至更前项活动。 ◘ 缺点：1、现实中软件的需求很难确定。2、需要很长时间才会得到软件的初始版本，如果需求改变可能损失巨大。
原型模型	◘ 应该注意： 1. 用户对系统模糊不清，无法准确回答目标系统的需求。 2. 要有一定的开发环境和工具支持。 3. 经过对原型的若干次修改，应收敛到目标范围内，否则可能会失败。 4. 对大型软件来说，原型可能非常复杂而难以快速形成，如果没有现成的,就不应考虑用原型法。
螺旋模型	◘ 瀑布模型与原型模型相结合，并加入两者所忽略的风险分析。 ◘ 螺旋模型沿着螺线进行若干次迭代，每次迭代都包括制订计划、风险分析、实施工程和客户评估4个方面的工作，特别适用于庞大、复杂并具有高风险的系统。
喷泉模型	◘ 无间隙：在各项活动之间无明显边界，如分析、设计和编码之间没有明显的界限。 ◘ 迭代：在系统某个部分常常被重复工作多次，相关对象在每次迭代中随之加入渐进的系统。

开发模型

快速应用开发	◘ 瀑布的高速变种，构件化开发。 ◘ 基本思想体现在以下4个方面： 1. 让用户更主动地参与到系统分析、设计和构造活动中来。 2. 将项目开发组织成一系列重点突出的研讨会，研讨会要让项目投资方、用户、系统分析师、设计人员和开发人员一起参与。 3. 通过一种迭代的构造方法，加速需求分析和设计阶段。 4. 让用户提前看到一个可工作的系统。 ◘ 局限性： 1. 并非所有应用都适合RAD。 2. 需求分析时间短需要各方配合。 3. 不适合新技术
RUP	◘ 特点：用例驱动、以架构为中心、迭代和增量的。 ◘ 工作流：业务建模、需求、分析与设计、实现、测试、部署、配置与变更管理、项目管理、环境 ◘ 阶段：初始、细化、构建、移交 ◘ 核心概念：角色、活动、制品、工作流

结构化方法

- 结构化设计：

模块的内聚从低到高排序

内聚类型	描述
偶然内聚	指一个模块内的各处理元素之间没有任何联系完全是通过上级模块的控制和调用来实现的
逻辑内聚	指模块内执行若干个逻辑上相似的功能，通过参数确定该模块完成哪一个功能
时间内聚	指把需要同时执行的动作组合在一起形成的模块
过程内聚	指一个模块完成多个任务，这些任务必须按照指定的过程执行
通信内聚	指模块内的所有处理元素都在同一个数据结构上操作，或者各处理使用相同的输入数据或者产生相同的输出数据
顺序内聚	指一个模块中的各个处理元素都密切相关于同一功能且必须顺序执行，前一功能元素的输出就是下一功能元素的输入
功能内聚	指模块内的所有元素共同作用完成一个功能，缺一不可

结构化方法

模块的耦合从低到高排序

耦合类别	描述
无直接耦合	指两个模块之间没有直接的关系，它们分别从属于不同模块的控制与调用，它们之间不传递任何消息。
数据耦合	指两个模块之间有调用关系，传递的是简单的数据值。
标记耦合	指两个模块之间传递的是数据结构
控制耦合	模块之间传递的信息中包含用于控制模块内部逻辑的信息
通信耦合	一组模块共用了一组输入信息，或者它们的输出需要整合以形成完整数据，即共享了输入或输出。
公共耦合	一组模块都访问同一个全局数据结构，则称之为公共耦合。公共数据环境可以是共享的通信区、内存的公共覆盖区、远程的公共存储区等。
内容耦合	当一个模块直接使用另一个模块的内部数据，或通过非正常入口转入另一个模块内部时

结构化方法

- 结构化详细设计的基本步骤如下：

① 分析并确定输入 / 输出数据的逻辑结构。

② 找出输入数据结构和输出数据结构中有对应关系的数据单元。

③ 按一定的规则由输入、输出的数据结构导出程序结构。

④ 列出基本操作与条件，并把它们分配到程序结构图的适当位置。

⑤ 用伪码写出程序。

结构化详细设计常用工具	
程序流程图	直观、清晰，易于学习掌握。
盒图（N-S图）	避免了流程图在描述程序逻辑时的随意性与灵活性。
过程设计语言	也称为结构化语言或伪代码，采用自然语言的词汇和结构化程序设计语言的语法，类似于编程语言。
问题分析图	允许递归使用。
判定表	适用于多个互相联系的条件和可能产生多种结果的问题。
判定树	适用于多个互相联系的条件和可能产生多种结果的问题，比判定表更直观。

面向对象方法

- 面向对象的设计原则

单一职责原则	设计目的单一的类。
开放-封闭原则	对扩展开放，对修改封闭。
里式替换原则	子类可以替换父类
依赖倒置原则	要依赖于抽象，不是具体实践。对接口进行编程，不要对实现编程。
接口隔离原则	使用多个专门的接口比使用单一的总接口好。
组合重用原则	尽量使用组合不是继承达到重用的目的。
迪米特原则 (最少知识)	一个对象应当对其他对象有尽可能少的了解。

面向对象方法

- 设计模式：

创建型模式	用于创建对象	工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。共五种。口诀：单抽元件（建）厂
结构型模式	处理类或对象的组合	适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。共七种。 口诀：外侨（桥）组员（元）戴（代）配饰。
行为型模式	描述类与对象怎样交互、怎样分配职责	策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。共十一种。口诀：观摩（模）对（迭）策，责令解放（访），戒（介）忘台（态）。

测试阶段

- 测试阶段

单元测试	测试模块的功能不符合 / 不满足期望的情况和编码错误。
集成测试	对组装起来的模块同时进行测试，明确该程序结构组装的正确性，发现和接口有关的问题。
系统测试	黑盒测试，结束标志是测试工作已满足测试目标所规定的需求覆盖率，并且测试所发现的缺陷都已全部归零。
性能测试	负载测试：是测试当负载逐渐增加时，系统各项性能指标的变化情况。
	压力测试：通过确定一个系统的瓶颈或者不能接受的性能点，来获得系统能提供的最大服务级别的测试。
验收测试	软件产品投入正式交付前的测试工作。测试软件满足用户要求和标准。

净室软件工程

- 净室软件工程（Cleanroom Software Engineering，CSE ）是一种在软件开发过程中强调在软件中建立正确性的需要的方法。

7 系统架构设计师——架构设计与评估

软件架构设计与生命周期

软件架构（Software Architecture）设计主要关注软件构件的结构、属性和交互作用 ， 并通过多种视图全面描述特定系统的架构。

- 需求分析阶段：研究SA模型转换和追踪性。
- 设计阶段：包括SA模型描述、ADL、多视图表示等。
- 实现阶段：研究基于SA的开发过程支持和寻求向实现过渡的途径。
- 构件组装阶段：支持构件互联和解决体系结构失配问题。
- 部署阶段：基于SA模型选择合理部署方案。
- 后开发阶段：涉及动态软件体系结构和体系结构恢复与重建。

基于体系结构（架构）的软件设计

- 基于体系结构（架构）的软件设计(Architecture-Based Software Design，ABSD)方法。是体系结构驱动，即指构成体系结构的商业、质量和功能需求的组合驱动的。在基于体系结构的软件设计方法中，采用视角与视图来描述软件架构，采用用例来描述功能需求，采用质量场景来描述质量需求。
- ABSD方法是一个自顶向下，递归细化的方法，软件系统的体系结构通过该方法得到细化，直到能产生软件构件和类。
- ABSD方法有三个基础：

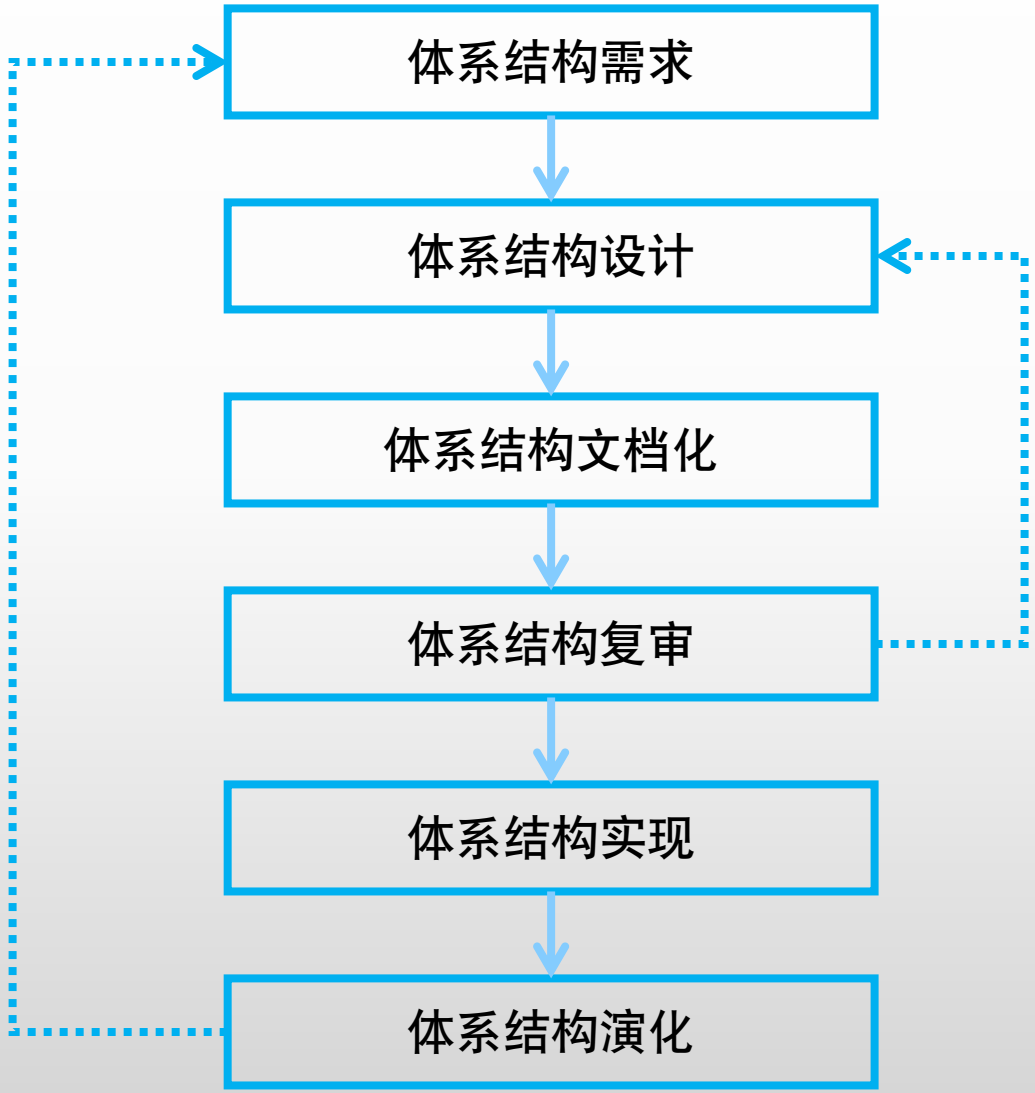
① 第一个基础是功能的分解。在功能分解中，ABSD方法使用已有的基于模块的内聚和耦合技术。

② 第二个基础是通过选择体系结构风格来实现质量和商业需求。

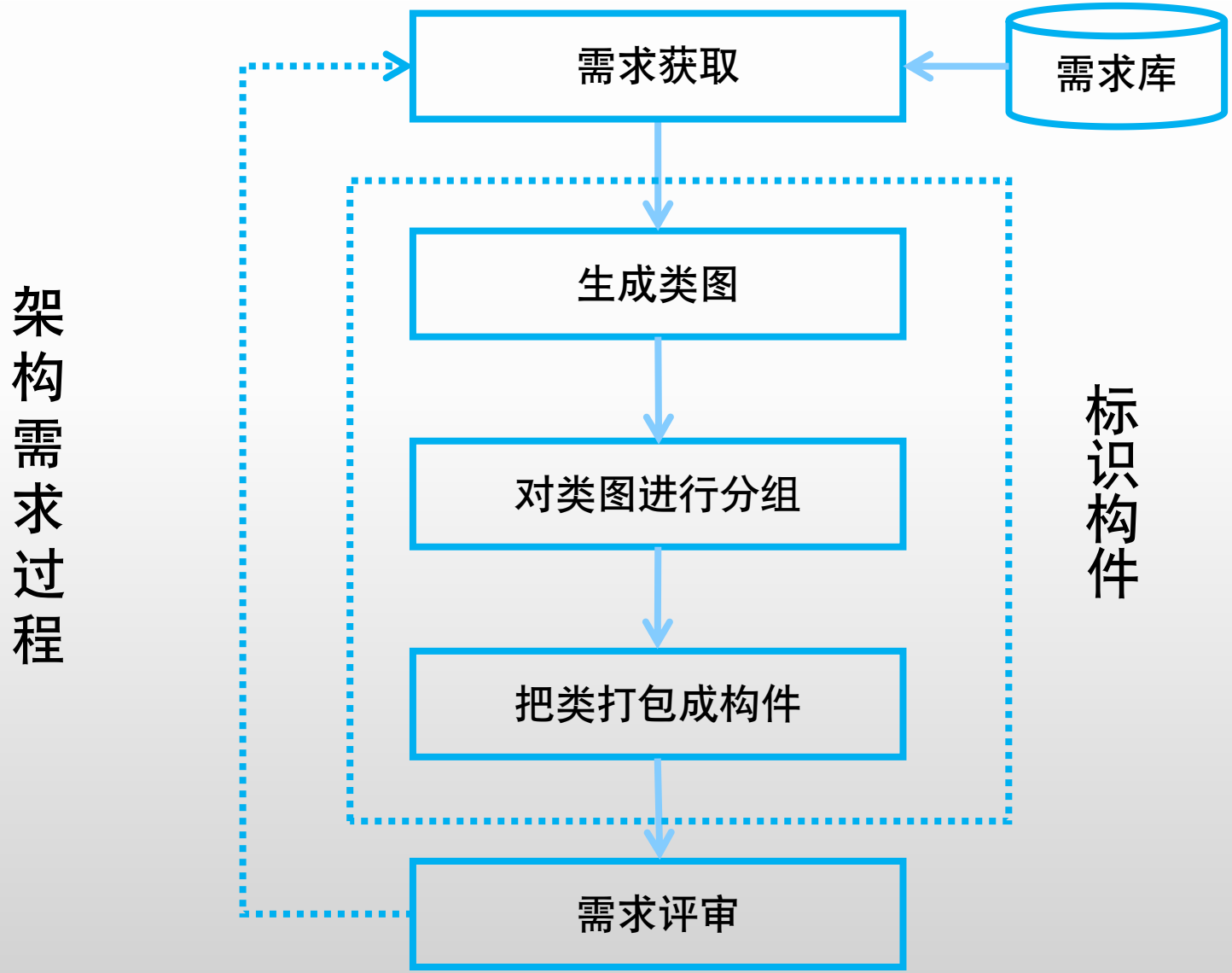
③ 第三个基础是软件模板的使用。软件模板利用了一些软件系统的结构。

基于体系结构的开发模型ABSDM

体系结构六大过程

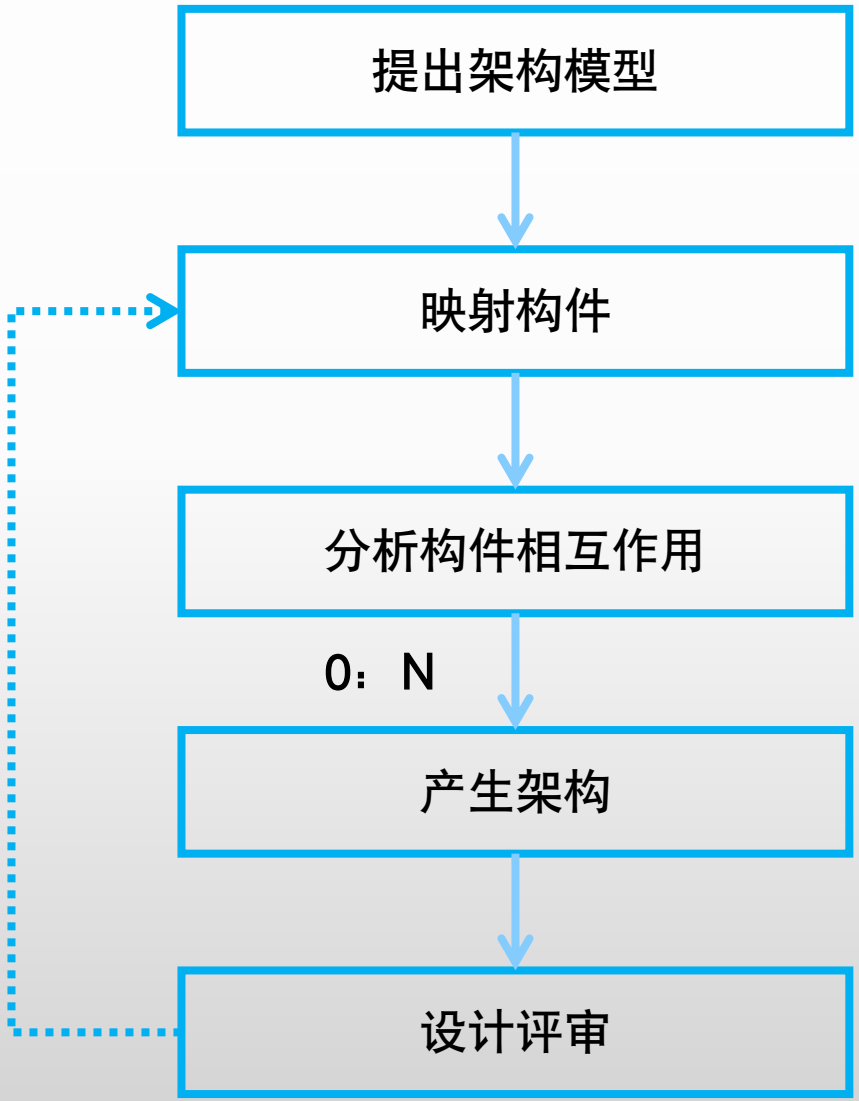


基于体系结构的开发模型ABSDM



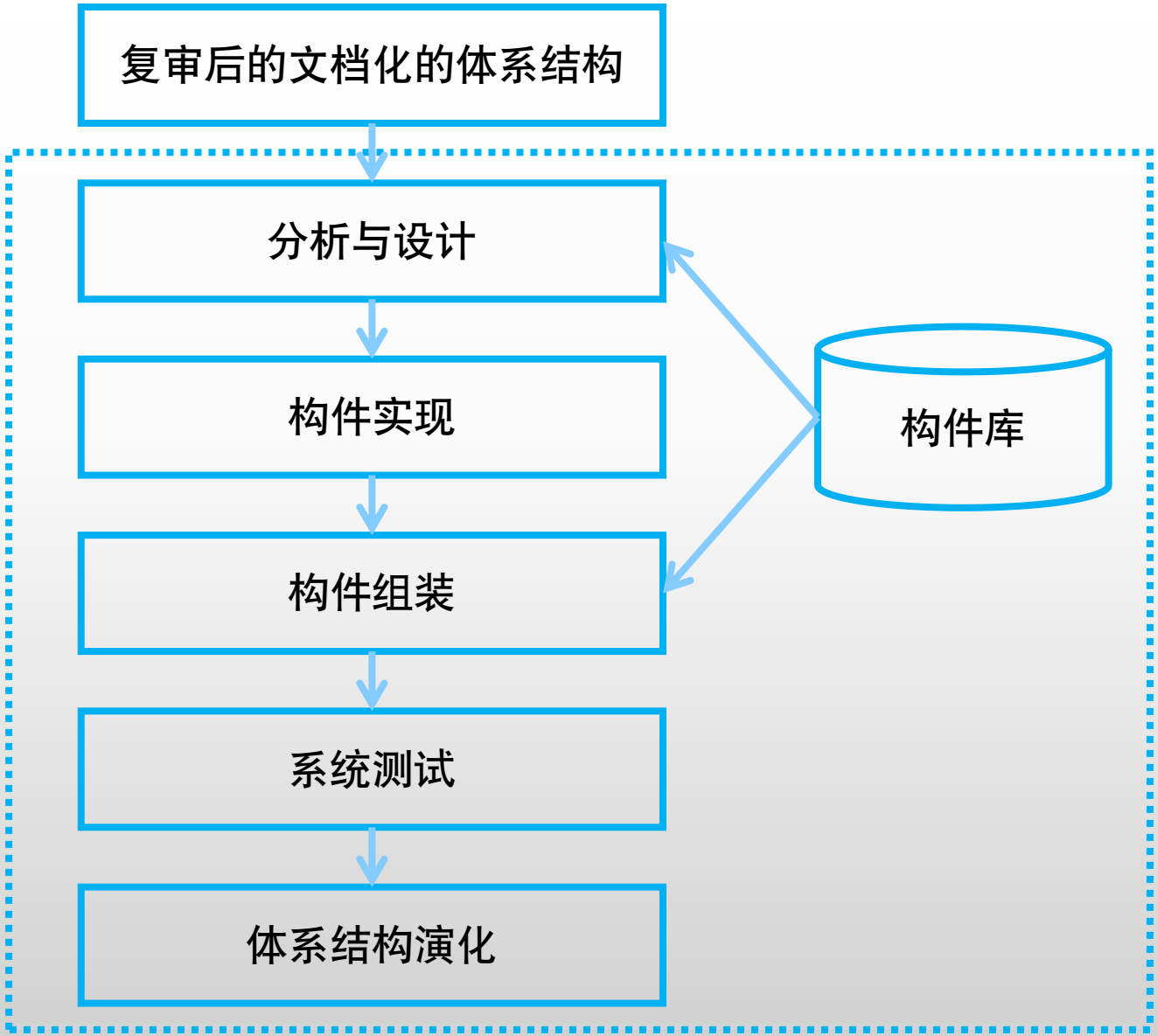
基于体系结构的开发模型ABSDM

架构设计过程

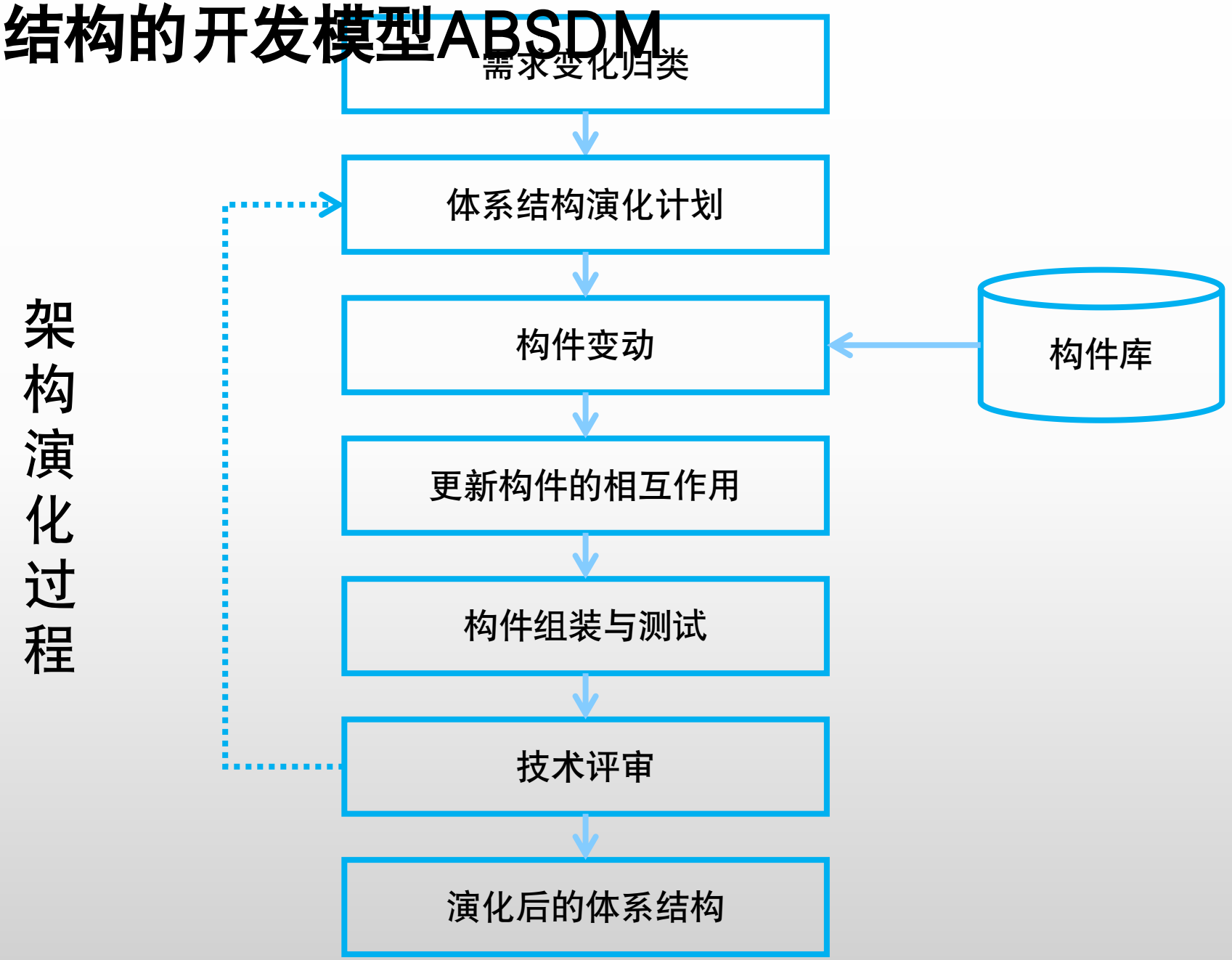


基于体系结构的开发模型ABSDM

架构实现过程



基于体系结构的开发模型ABSDM



架构风格	风格子类	关键词和场景	
数据流风格	批处理风格	整体，适用于离线数据分析和大规模数据处理	构件顺序处理、交互性不好
	管道-过滤器风格	流式，可以实现实时数据处理和流式计算	
调用/返回风格	主程序/子程序风格	主程序直接调用子程序	
	面向对象风格	显式调用，调用对象的方法，须知道它的标识和名称	
	层次结构风格	分层，调用下层层次	
	客户机/服务器风格	基于资源不对等，实现共享	
数据共享风格	仓库（数据库）体系结构风格	业务功能驱动系统相应的进程执行	
	黑板系统风格	中央数据结构的当前状态触发系统相应的进程执行，共享知识库，协同处理	
	超文本风格	网状链接、互联网领域。	
虚拟机风格	解释器风格	灵活、自定义，逐行解释源代码，效率低。	
	基于规则的系统风格	灵活、自定义基于事先定义的规则和条件，效率低	
独立构件风格	进程通信风格	显式进程间通信，实现分布式系统	
	事件系统体系结构风格	隐式调用、解耦、构件放弃了对计算的控制权	

软件系统质量属性

软件系统质量属性（Quality Attribute）是一个系统的可测量或者可测试的属性，用来描述系统满足利益相关者（Stakeholders）需求的程度。

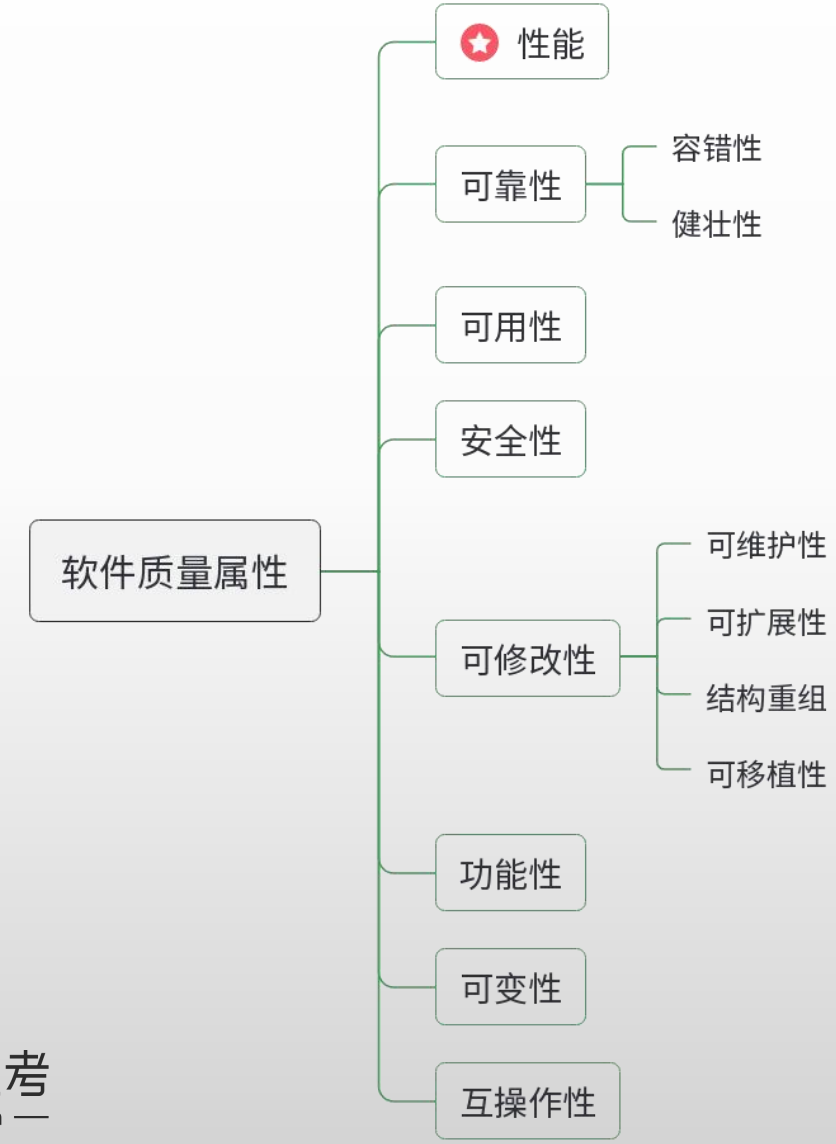
开发期质量属性

- 易理解性
- 可扩展性
- 可重用性
- 可测试性
- 可维护性
- 可移植性

运行期质量属性

- 性能
- 安全性
- 可伸缩性
- 互操作性
- 可靠性
- 可用性
- 鲁棒性

面向架构评估的质量属性



质量属性场景描述

- 刺激源（Source）：这是某个生成该刺激的实体(人、计算机系统或者任何其他刺激器)。
- 刺　激（Stimulus）：该刺激是当刺激到达系统时需要考虑的条件。
- 环境（Environment）：该刺激在某些条件内发生。当激励发生时，系统可能处于过载、 运行或者其他情况。
- 制品（Artifact）：某个制品被激励。这可能是整个系统，也可能是系统的一部分。
- 响应（Response）：该响应是在激励到达后所采取的行动。
- 响应度量（Measurement）： 当响应发生时，应当能够以某种方式对其进行度量，以对需求进行测试。

系统架构评估方法

基于调查问卷（检查表）方式

问卷调查。

基于度量方式：

制定一些 定量指标来度量架构，如代码行数。

基于场景的方式

架构权衡分析法 (Architecture Tradeoff Analysis Method,ATAM)

软件架构分析方法 (Software Architecture Analysis Method,SAAM)

成本效益分析法 (the Cost Benefit Analysis Method,CBAM)

8 系统架构设计师——法律法规

著作权、商标权、专利权

知识产权		取得条件	地域性	保护期限	计算起点	生效日	展期	登记机构
著作权	文学、文艺、影视作品、计算机软件作品（包括：人身权和著作财产权。著作权的署名权、修改权、完整权保护期不受限制）		无	50年	自然人：作者死后 单位：首次发表后	作品完成日	不予展期	中国版权保护中心
商标权	显著性的文字、图形、字母、数字		有	10年	核准注册日	核准注册日	可以展期	国家知识产权局
专利权	发明专利	新颖性、创造性、实用性	有	20年	申请日（邮寄以寄出邮戳日）	授予日（公告日）	不予展期	国家知识产权局
	实用新型			10年				
	外观设计	新颖性、美感、适用于工业应用		15年				

职务作品与单位作品

区分关键	职务作品	单位作品
是否由单位组织	否	是
作品体现的意志	个人意志	单位的意志
单位在过程中发挥的作用	只要最后的工作成果	把控整个过程
承担责任的主体	个人/单位	单位
举例	● 员工为公司开发的内部管理软件 ● 员工完成公司软件系统开发的低代码平台	● 公司出版的宣传手册由公司员工编写。 ● 《需求规格说明书》

一般职务作品与特殊职务作品

种类	一般职务作品	特殊职务作品
含义	除单位作品外，公民为完成单位工作任务而又未主要利用单位物质技术条件创作的作品。	主要是利用单位的物质技术条件制作并由单位承担责任的工程设计图、产品设计图、地图、计算机软件等职务作品，或另有约定。
归属	原则上归个人	原则上归单位
权利行使	① 单位在其业务范围内优先使用。 ② 作品完成2年内未经单位同意，作者不得许可第三人以与单位使用的相同方式使用该作品。 ③ 作品完成2年内经过单位同意，作者许可第三人以与单位使用的相同方式使用该作品，所获报酬，由作者与单位按约定的比例分配，(作品完成两年的期限，自作者向单位交付作品之日计算)。	个人享有署名权、获得报酬权、奖励权

9 系统架构设计师——案例分析

案例方向

- 案例方向：
- 案例一：架构风格+架构评估
- 案例二：软件工程系统建模工具
- 案例三：嵌入式（不推荐做）
- 案例四：数据库设计与性能优化技术
- 案例五：Web架构设计（云大物综合题）

Redis

- 缓存问题

（1）缓存穿透。大量请求访问了没有缓存的key，即大量的key在redis里是不存在的，从而导致请求直接访问数据库，数据库压力增大。

原因	解决方案
恶意攻击，造成大量访问不存在的key。	● 针对比较少的请求来源ip，主动限制其访问次数，或者拉入黑名单。 ● 应用程序来检查key的合法性，提前拒绝不合法的请求。 ● 使用布隆过滤器。
大量请求访问数据库里有但redis没有的key。例如新业务刚刚上线，此时Redis是空的。	● 预热Redis，运行一个批处理脚本，将可能会大量访问的数据预先加载到Redis，业务再“开张”。 ● 在最前端进行流量控制，逐步释放进来请求。给出一段时间，让Redis逐步加载热数据。 ● 如果数据库里也没有的key，也要在Redis中设置key，使其值为null或空。

Redis

（2）缓存雪崩。大量请求访问到缓存中的key，这些Key是存在的，但同时到了过期时间，从而导致请求直接访问数据库，数据库压力增大。

原因	解决方案
redis故障。比如redis宕机，网络出现抖动等。	● 使用主从复制提高可用性， 使用cluster集群方案降低故障时影响的范围。 ● 如果出现故障，则可以采取服务降级、熔断、限流等措施。
大量的key采用了相同的过期时间。例如在同一时刻设置了大量的key，但过期时间都是5分钟。	● 过期时间加上一个随机值，使得众多key均匀过期。

Redis

（3）缓存击穿。少量热点的key缓存时间失效了，使得请求直接访问数据库。

原因	解决方案
热点的key设置了太短的过期时间。例如秒杀业务下的“库存数量”。	● 将key设置较长的过期时间。对于非常重要的key，则设置永久有效。但需要解决好与数据库中的key的一致性问题。 ● 使用分布式锁。如果热点key失效了，要控制好访问后端数据库的流量。只允许一个请求去访问数据库，取出最新的key，存放到Redis，其他请求则必须等待。但分布式锁也要防止出现异常的情况。

Redis

- 缓存与数据库同步：

策略名称	工作方式	数据不一致问题解决方案
Cache-Aside (旁路缓存)	读：先读缓存，没有再读数据库并更新缓存 写：先写数据库再删除缓存	消息队列、订阅binlog日志、分布式锁等
Read-Through (读穿透) 与 Write-Through (写穿透)	读：同Cache-Aside 写： ● 缓存有，更新缓存同步更数据库 ● 缓存无，直接更新数据库后返回	消息队列、分布式锁、延迟双删等
Write-Back (写回)	写：先更新缓存并标记，再异步更新数据库	重试机制、消息队列、持久化缓存、数据恢复机制、分布式锁

HBase

- HBase特征：

01

HBase是一个稀疏、多维度、排序的映射表，这张表的索引是行键、列族、列限定符和时间戳

02

每个值是一个未经解释的字符串，没有数据类型

03

用户在表中存储数据，每一行都有一个可排序的行键和任意多的列

04

表在水平方向由一个或者多个列族组成，一个列族中可以包含任意多个列，同一个列族里面的数据存储在一起

05

列族支持动态扩展，可以很轻松地添加一个列族或列，无需预先定义列的数量以及类型，所有列均以字符串形式存储，用户需要自行进行数据类型转换

06

HBase中执行更新操作时，并不会删除数据旧的版本，而是生成一个新的版本，旧有的版本仍然保留

MongoDB

- MongoDB的特征：

- 高性能。提供 JSON、XML等可嵌入数据快速处理功能；提供文档的索引功能，相对传统数据库而言，大大提高查询速度。
- 丰富的查询语言。为数据聚合、结构文档、地理空间提供丰富的查询功能。
- 高可用性。提供数据冗余处理和故障自动转移的功能。
- 水平扩展能力。通过集群将数据分布到多台机器，而不是只提升单个节点的性能，具体处理分为主从和权衡两种处理模式。

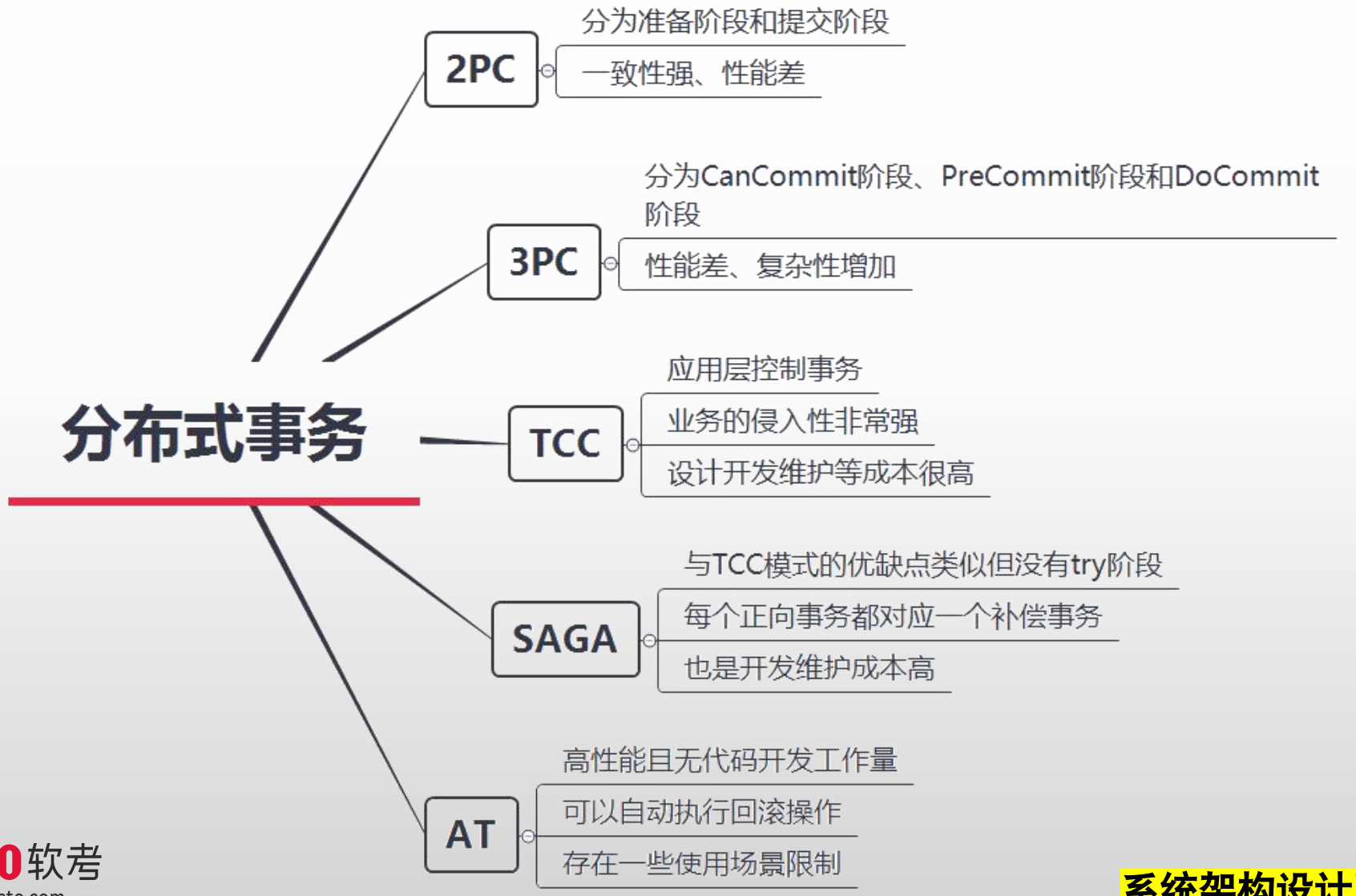
数据库与数据仓库

内容	数据库	数据仓库
数据内容	当前值	历史的、存档的、归纳的、计算的
数据目标	面向业务操作人员，重复处理	面向主题域、分析应用
数据特性	动态变化，按字段更新	静态、不能直接更新，只能定时添加、刷新
数据结构	高度结构化、复杂、适合操作计算	简单、适合分析
使用频率	高	中、低
数据访问量	每个事务只访问少量的记录	有的事务可能需要访问大量的记录
对响应时间的要求	以秒为单位计算	以秒、分钟甚至小时为计算单位

数据库与数据湖

特性	数据库	数据湖
定义与用途	组织化的数据集合，长期存储、管理和检索结构化数据	集中式存储库，存储结构化、半结构化和非结构化数据，用于高级分析和机器学习
数据结构与处理	主要存储结构化数据，写入模式处理	存储多样化数据，读取模式处理，无需预定义数据结构
存储与成本	使用关系型或非关系型数据库存储，成本相对灵活	使用分布式文件系统或云存储服务，存储成本相对较低
敏捷性与灵活性	配置和修改相对不够灵活，适应大规模数据变化或新需求时间较长	高度敏捷性，快速配置和重置数据模型、查询和应用，处理多样化数据具有优势
安全性与治理	具有保护数据不被未授权访问和破坏的机制，完善的数据管理功能	需要采取更严格的安全措施，建立有效的数据治理机制确保数据质量、安全性和合规性

分布式事务


分布式事务
2PC
分为准备阶段和提交阶段
一致性强、性能差
3PC
分为CanCommit阶段、PreCommit阶段和DoCommit阶段
性能差、复杂性增加
TCC
应用层控制事务
业务的侵入性非常强
设计开发维护等成本很高
SAGA
与TCC模式的优缺点类似但没有try阶段
每个正向事务都对应一个补偿事务
也是开发维护成本高
AT
高性能且无代码开发工作量
可以自动执行回滚操作
存在一些使用场景限制

软件工程

状态图	活动图
用来描述一个特定的对象所有可能的状态，以及由于各种事件的发生而引起的状态之间的转移和变化。	将进程或其他计算的结构展示为计算内部一步步的控制流和数据流，主要用来描述系统的动态视图。
状态图主要描述行为的结果。	活动图主要描述行为的动作。
用于对系统的动态方面建模。	用于对系统的动态方面建模。

软件工程

流程图	活动图
描述处理过程。主要控制结构是顺序、分支和循环，各个处理过程之间有严格的顺序和时间关系。	描述的是对象活动的顺序关系所遵循的规则，它着重表现的是系统的行为，而非系统的处理过程。
不能表示并发活动的情形	能够表示并发活动的情形
面向过程	面向对象

软件工程

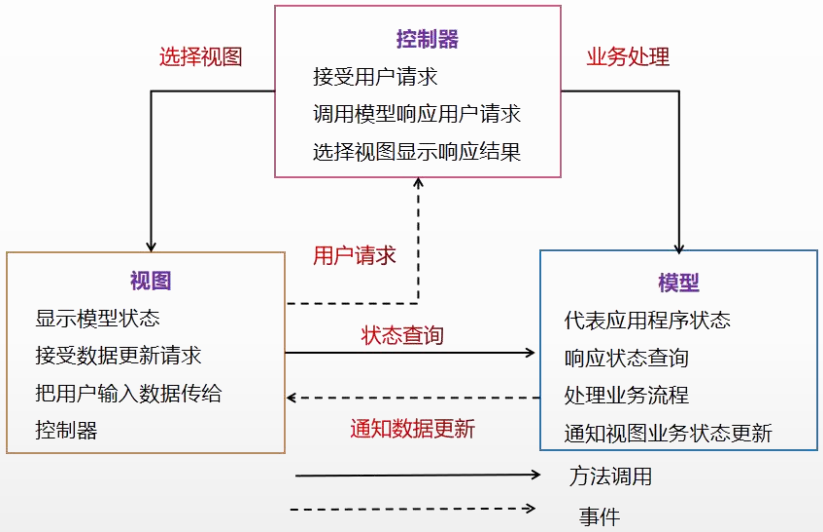
数据流图	系统流程图
数据流图中的处理过程可并行	系统流程图在某个时间点只能处于一个处理过程
数据流图展现系统的数据流	系统流程图展现系统的控制流
数据流图展现全局的处理过程，过程之间遵循不同的计时标准	系统流程图中处理过程遵循一致的计时标准。

层次式体系结构

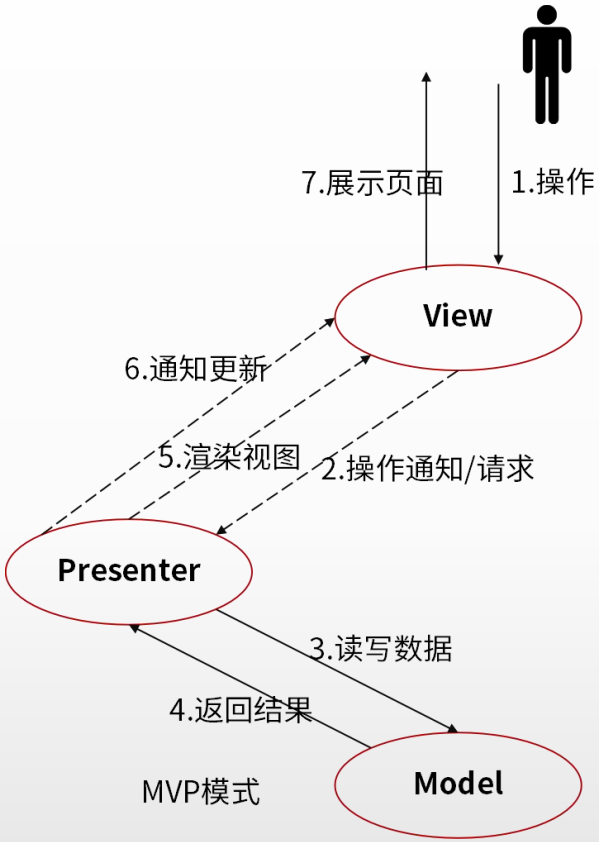
- 优点：
  - 开发人员可以只关注整个结构中的其中某一层；
  - 可以很容易的用新的实现来替换原有层次的实现；
  - 可以降低层与层之间的依赖；
  - 有利于标准化；
  - 利于各层逻辑的复用；
  - 扩展性强。不同层负责不同的层面；
  - 安全性高。用户端只能通过逻辑层来访问数据层，减少了入口点，把很多危险的系统功能都屏蔽了。
  - 项目结构更清楚，分工更明确，有利于后期的维护和升级。
- 缺点：
  - 严格的分层可能导致性能问题，具体取决于层数。
  - 建立清晰的分层架构并不总是很容易。

层次式体系结构

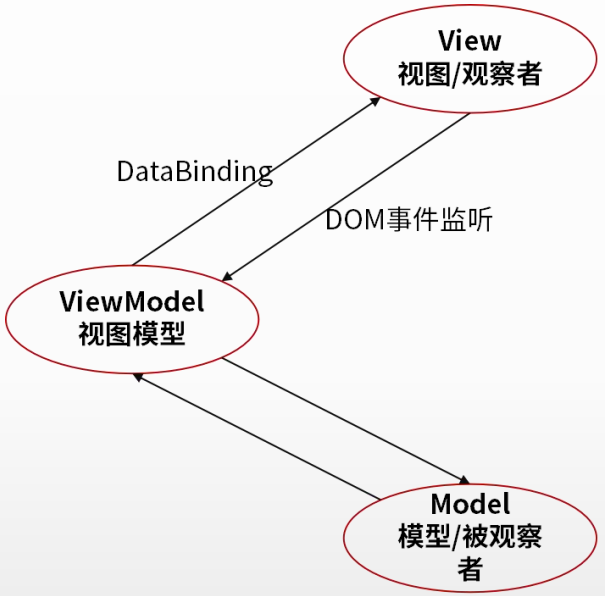
- 表现层设计



MVC模式



MVP模式



MVVM模式

层次式体系结构

- 使用MVC模式来设计表现层，可以有以下的优点：

（1）允许多种用户界面的扩展。在MVC模式中，视图与模型没有必然的联系，都是通过控制器发生关系，这样如果要增加新类型的用户界面，只需要改动相应的视图和控制器即可，而模型则不需发生改动。

（2）易于维护。控制器和视图可以随着模型的扩展而进行相应的扩展，只要保持一种公共的接口，控制器和视图的旧版本也可以继续使用。

（3）功能强大的用户界面。用户界面与模型方法调用组合起来，使程序的使用更清晰，可将友好的界面发布给用户。

- 使用MVP 模式来设计表现层，可以有以下的优点。

（1）模型与视图完全分离，可以修改视图而不影响模型。

（2）可以更高效地使用模型，因为所有的交互都发生在一个地方—-Presenter 内部。

（3）可以将一个Presenter 用于 多个视图，而不需要改变Presenter 的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁。

（4）如果把逻辑放在Presenter 中，就可以脱离用户接口来测试这些逻辑（单元测试）。

- MVVM 中，View 与Model 的交互通过ViewModel 来实现。ViewModel 是MVVM 的核心，它通过DataBinding 实现 View 与Model 之间的双向绑定，其内容包括数据状态处理、数据绑定及数据转换。

层次式体系结构

- 数据访问层设计：

在线访问

占用一个数据库连接，通过连接与后台数据库交互。

离线数据模式

数据从数据源获取之后，存放在系统中。

DataAccess Object

将底层数据访问操作与高层业务逻辑分离开。

对象 / 关系映射 ORM

将应用程序中的数据转换成关系型数据库中的记录，或反之。

Data Transfer Object

DTO 本身不包含任何业务逻辑或持久化逻辑。它的作用仅仅是数据的传输和临时存储。

SOA

- SOA主要协议和规范

UDDI	统一描述、发现和集成，提供了一种服务发布、查找和定位的方法，是服务的信息注册规范。
WSDL	Web服务描述语言是对服务进行描述的语言。
SOAP	简单对象访问协议定义了服务请求者和服务提供者之间的消息传输规范。
REST	表述性状态转移REST，是一种设计风格： （1）资源：以资源为中心，一个资源可以对应多个URI（统一资源标识），一个URI只能对应 一种资源。 （2）表述：资源的状态，可以有HTML、 JSON 、XML 、纯文本等。 （3）状态转移：应用状态保存在客户端，资源状态保存在服务端。 （4）超链接：在页面中嵌入链接和其他资源建立联系，包含在应用状态中，由客户端维护保存。

SOA

- SOA设计原则：无状态、单一实例、明确定义的接口、自包含和模块化、粗粒度、服务之间的松耦合性、重用能力、互操作性、兼容和策略声明。
- SOA的实现方式有：服务注册表、企业服务总线
- 企业服务总线ESB 的核心功能如下：

- 提供位置透明性的消息路由和寻址服务。
- 提供服务注册和命名的管理功能。
- 支持多种消息传递范型（如请求 / 响应、发布 / 订阅等）。
- 支持多种可以广泛使用的传输协议。
- 支持多种数据格式及其相互转换。
- 提供日志和监控功能。

微服务

- API网关在微服务中的作用：

（1）封装系统架构：封装了整个系统的架构，为客户端提供统一的接口。

（2）路由和转发：根据请求的URL、头部信息或请求体内容，将请求路由到相应的微服务实例。

（3）负载均衡：对请求进行负载均衡，将请求分配到多个微服务实例上。这有助于提高系统的可伸缩性和可用性。

（4）聚合响应：对于需要多个微服务协同完成的请求，API网关可以并行调用这些服务，并聚合它们的响应返回给客户端。

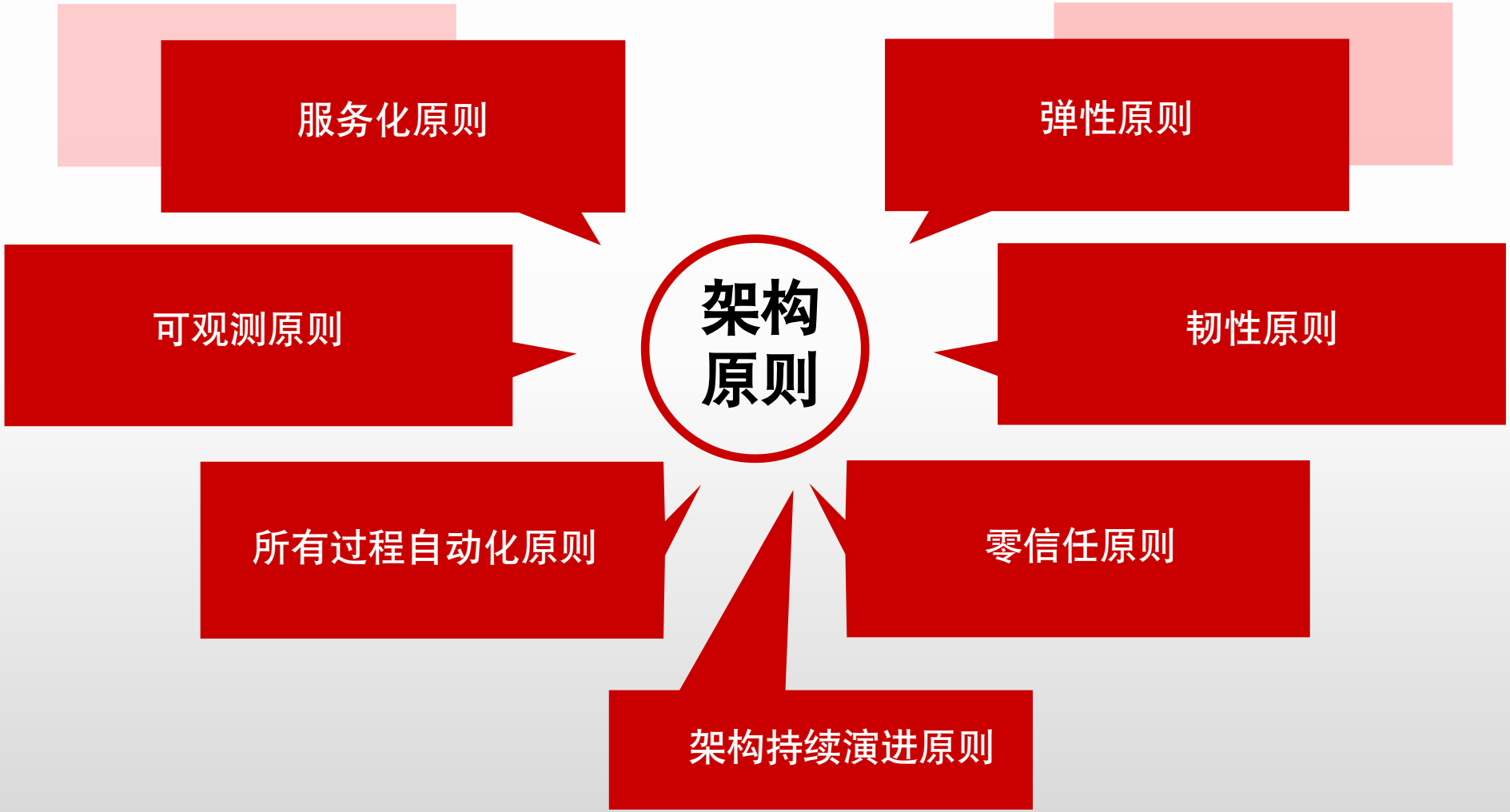
（5）认证与授权：在请求到达微服务之前，API网关可以执行身份验证和授权，确保只有合法的请求才能通过。

（6）限流与熔断：通过限流和熔断机制，API网关可以保护微服务免受流量洪峰和故障的影响，提高系统的稳定性和可用性。

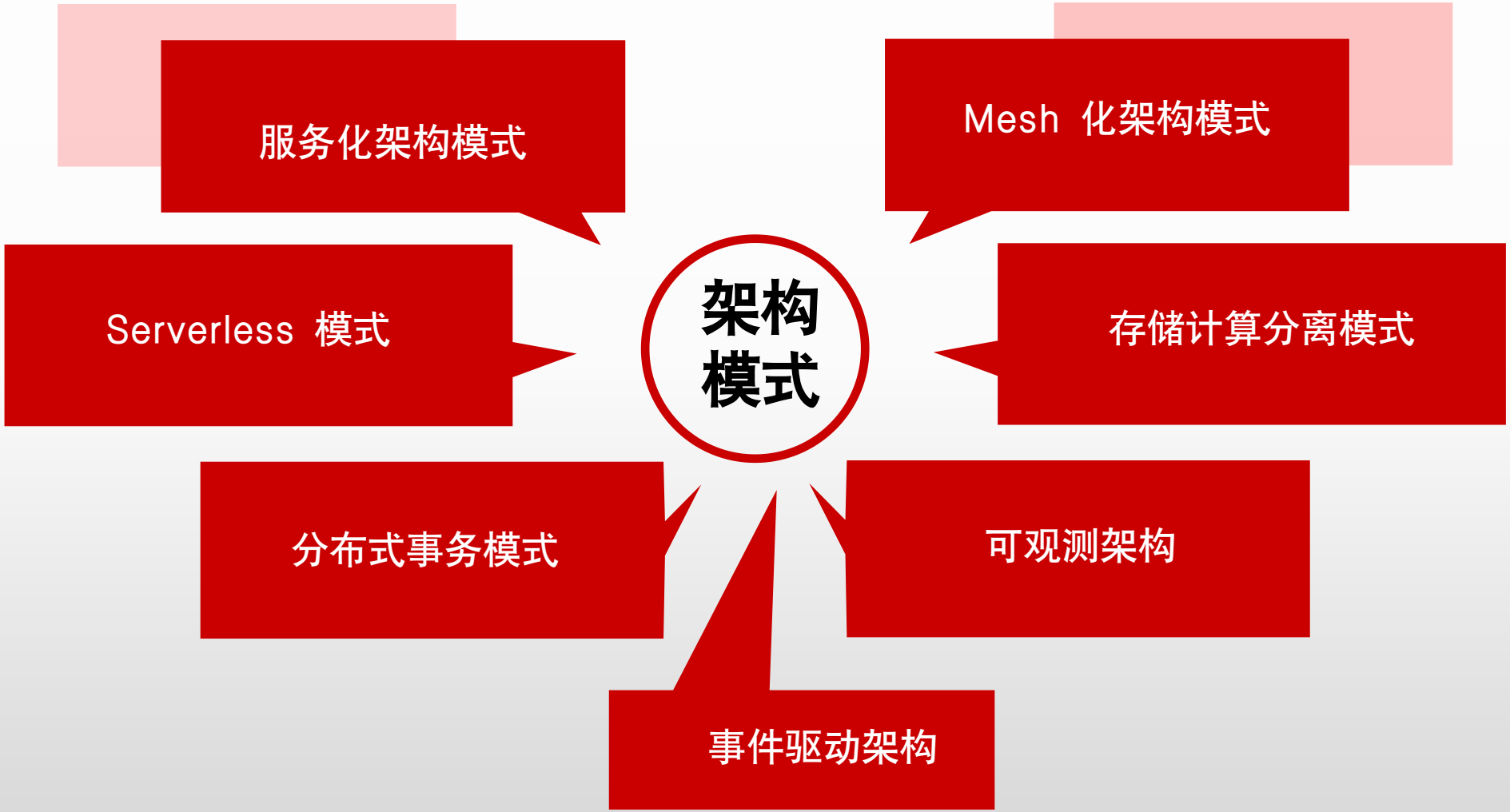
（7）协议转换：充当协议转换器的角色，将来自客户端的请求转换为微服务可以理解的协议，并将微服务的响应转换回客户端可以理解的协议。

（8）缓存：API网关可以对常见的请求进行缓存，减少对后端微服务的调用，提高系统的性能和响应速度。

云原生



云原生


架构模式
服务化架构模式
Mesh 化架构模式
Serverless 模式
存储计算分离模式
分布式事务模式
可观测架构
事件驱动架构

云原生

➢ 云原生关键技术：

- Docker 容器基于操作系统虚拟化技术，共享操作系统内核、轻量、没有资源损耗、秒级启动，极大提升了系统的应用部署密度和弹性。更重要的是，Docker 提出了创新的应用打包规范一-Docker 镜像，解耦了应用与运行环境，使应用可以在不同计算环境一致、可靠地运行。
- 容器编排标准Kubernetes 的能力：资源调度、应用部署与管理、自动修复、服务发现与负载均衡、弹性伸缩、声明式API、可扩展性架构、可移植性
- 无服务器技术：全托管的计算服务、通用性、自动弹性伸缩、按量计费
- 服务网格：服务网格（ServiceMesh）旨在将那些微服务间的连接、安全、流量控制和可观测等通用功能下沉为平台基础设施，实现应用与平台基础设施的解耦。

大数据

- Lambda架构常用技术：

数据流	分布式消息队列Kafka	
批处理层	数据存储	HDFS或HBase
	数据查询	HIVE
	数据计算	MapReduce或spark
加速层	数据存储	Hbase或内存型数据库
	数据查询	Impala
	数据计算	Storm或flink
服务层	批处理层与加速层的技术、数据合并算法	

- Kappa架构通常使用Kafka作为中转，常用的技术参考Lambda架构加速层

大数据

对比内容	Lambda架构	Kappa架构
复杂度与开发、维护成本	需要维护两套系统(引擎),复杂度高，开发、维护成本高	只需要维护一套系统(引擎),复杂度低，开发、维护成本低
计算开销	需要一直运行批处理和实时计算，计算开销大	必要时进行全量计算，计算开销相对较小
实时性	满足实时性	满足实时性
历史数据处理能力	批式全量处理，吞吐量大，历史数据处理能力强	流式全量处理，吞吐量相对较低，历史数据处理能力相对较弱

10 系统架构设计师——论文

论文

- 论文方向：

1、云原生

2、微服务

3、Devops

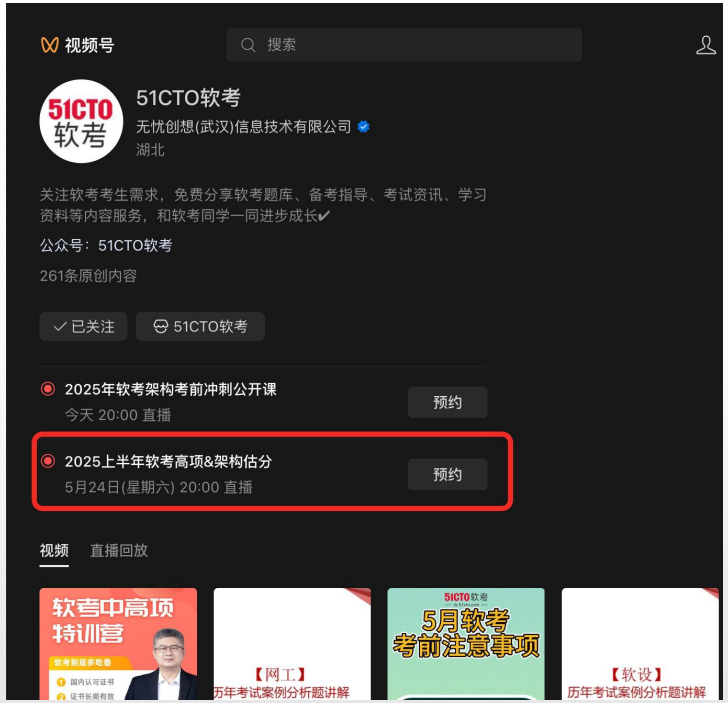
4、论软件设计方法及应用

5、企业集成

6、论系统架构演化

预祝各位考生5月软考考试顺利！

软考刷题小程序



点头像预约5月24日晚8点估分直播