如何带领团队“攻城略地”？优秀的架构师这样做

> **阿里妹导读：**架构师是一个既能掌控整体又能洞悉局部瓶颈并依据具体的业务场景给出解决方案的团队领导型人物。看似完美的“人格模型”背后，是艰辛的探索。今天，阿里巴巴技术专家九摩将多年经验，进行系统性地总结，帮助更多架构师在进阶这条路上走得更“顺畅”，姿态更“优雅”。

架构师职责

架构师不是一个人，他需要建立高效卓越的体系，带领团队去攻城略地，在规定的时间内完成项目。

架构师需要能够识别定义并确认需求，能够进行系统分解形成整体架构，能够正确地技术选型，能够制定技术规格说明并有效推动实施落地。

按 TOGAF 的定义，架构师的职责是了解并关注实际上关系重大但未变得过载的一些关键细节和界面，架构师的角色有：理解并解析需求，创建有用的模型，确认、细化并扩展模型，管理架构。

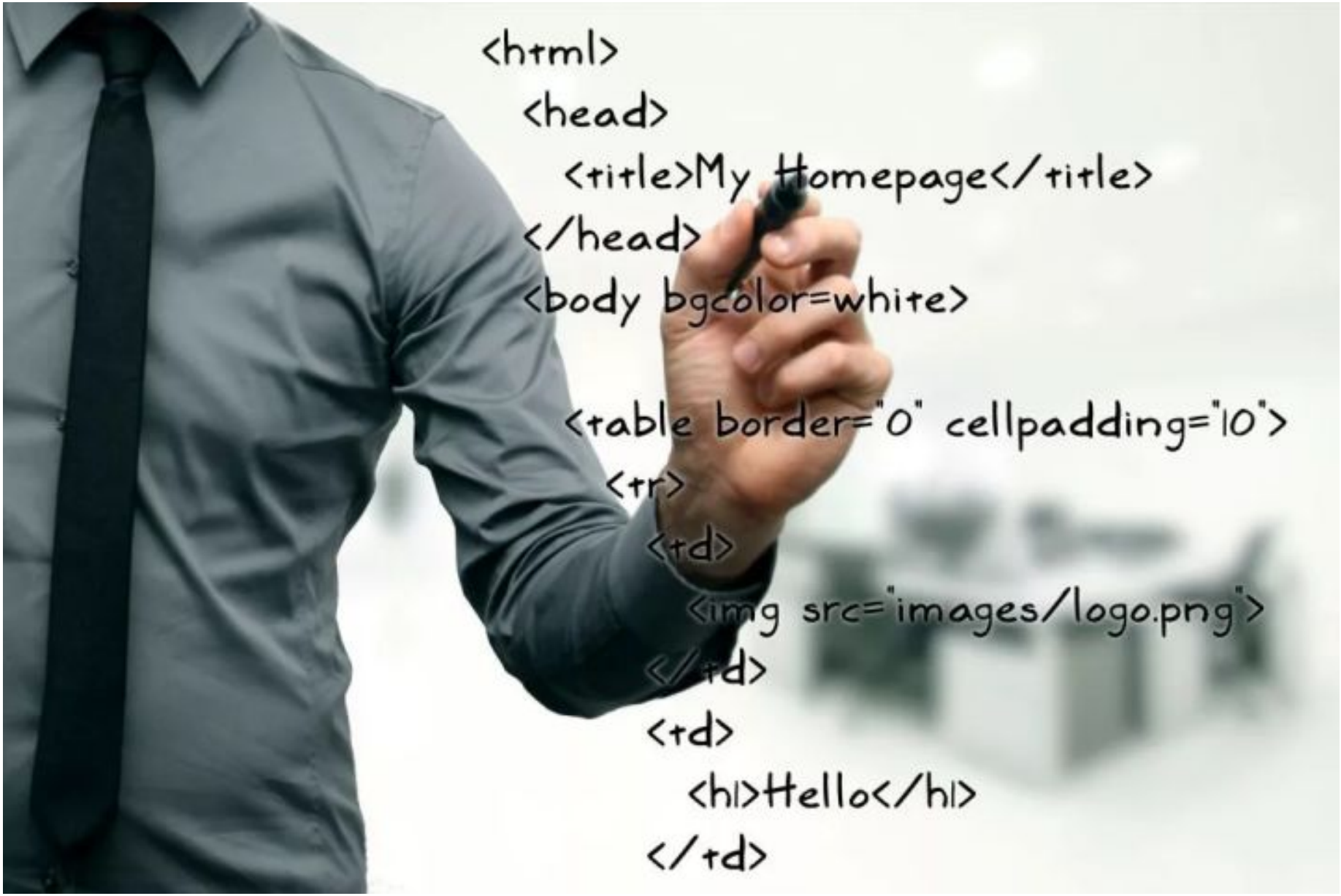
从业界来看对于架构师的理解可以大概区分为：

- 企业架构师：专注于企业总体 IT 架构的设计。
- IT 架构师-软件产品架构师：专注于软件产品的研发。

- IT 架构师-应用架构师：专注于结合企业需求，定制化 IT 解决方案；大部分需要交付的工作包括总体架构、应用架构、数据架构，甚至部署架构。

- IT 架构师-技术架构师：专注于基础设施，某种软硬件体系，甚至云平台，提交：产品建议、产品选型、部署架构、网络方案，甚至数据中心建设方案等。

阿里内部没有在职位 title 上专门设置架构师了，架构师更多是以角色而存在，现在还留下可见的 title 有两个：首席架构师和解决方案架构师，其中解决方案架构师目前在大部分 BU 都有设置，特别是在阿里云和电商体系。


<html>
<head>
<title>My Homepage</title>
</head>
<body bgcolor=white>
<table border="0" cellpadding="10">
<tr>
<td>
<img src="images/logo.png">
</td>
<td>
<h1>Hello</h1>
</td>

解决方案架构师

工作方式理解

- 了解和挖掘客户痛点，项目定义，现有环境管理；
- 梳理明确高阶需求和非功能性需求；
- 客户有什么资产，星环（阿里电商操作系统）/ 阿里云等有什么解决方案；
- 沟通，方案建议，多次迭代，交付总体架构；
- 架构决策。

职责

1.从客户视图来看：

- 坚定客户高层信心：利用架构和解决方案能力，帮忙客户选择星环 / 阿里云平台的信心。
- 解决客户中层问题：利用星环 / 阿里云平台服务+结合应用架构设计/解决方案能力，帮忙客户解决业务问题，获得业务价值。
- 引领客户 IT 员工和阿里生态同学：技术引领、方法引领、产品引领。

2.从项目视图看：

- 对接管理部门：汇报技术方案，进度；技术沟通。
- 对接客户 PM，项目 PM：协助项目计划，人员管理等。负责所有技术交付物的指导。
- 对接业务部门和需求人员：了解和挖掘痛点，帮忙梳理高级业务需求，指导需求工艺。
- 对接开发：产品支持、技术指导、架构指导。
- 对接测试：配合测试计划和工艺制定。配合性能测试或者非功能性测试。
- 对接运维：产品支持，运维支持。
- 对接配置&环境：产品支持。
- 其他：阿里技术资源聚合。

3.从阿里内部看：

- 销售方案支持；
- 市场宣贯；
- 客户需求Facade；
- 解决方案沉淀。

架构师职责明确了，那么有什么架构思维可以指导架构设计呢？请看下述的架构思维。

架构思维

自顶向下构建架构

要点主要如下：

1.首先定义问题，而定义问题中最重要的是定义客户的问题。定义问题，特别是识别出关键问题，关键问题是对客户有体感，能够解决客户痛点，通过一定的数据化来衡量识别出来，关键问题要优先给出解决方案。

2.问题定义务必加入时间维度，把手段/方案和问题定义区分开来。

3.问题定义中，需要对问题进行升层思考后再进行升维思考，从而真正抓到问题的本质，理清和挖掘清楚需求；要善用第一性原理思维进行分析思考问题。

4.问题解决原则：先解决客户的问题（使命），然后才能解决自己的问题（愿景）；务必记住不是强调我们怎么样，而是我们能为客户具体解决什么问题，然后才是我们变成什么，从而怎么样去更好得服务客户。

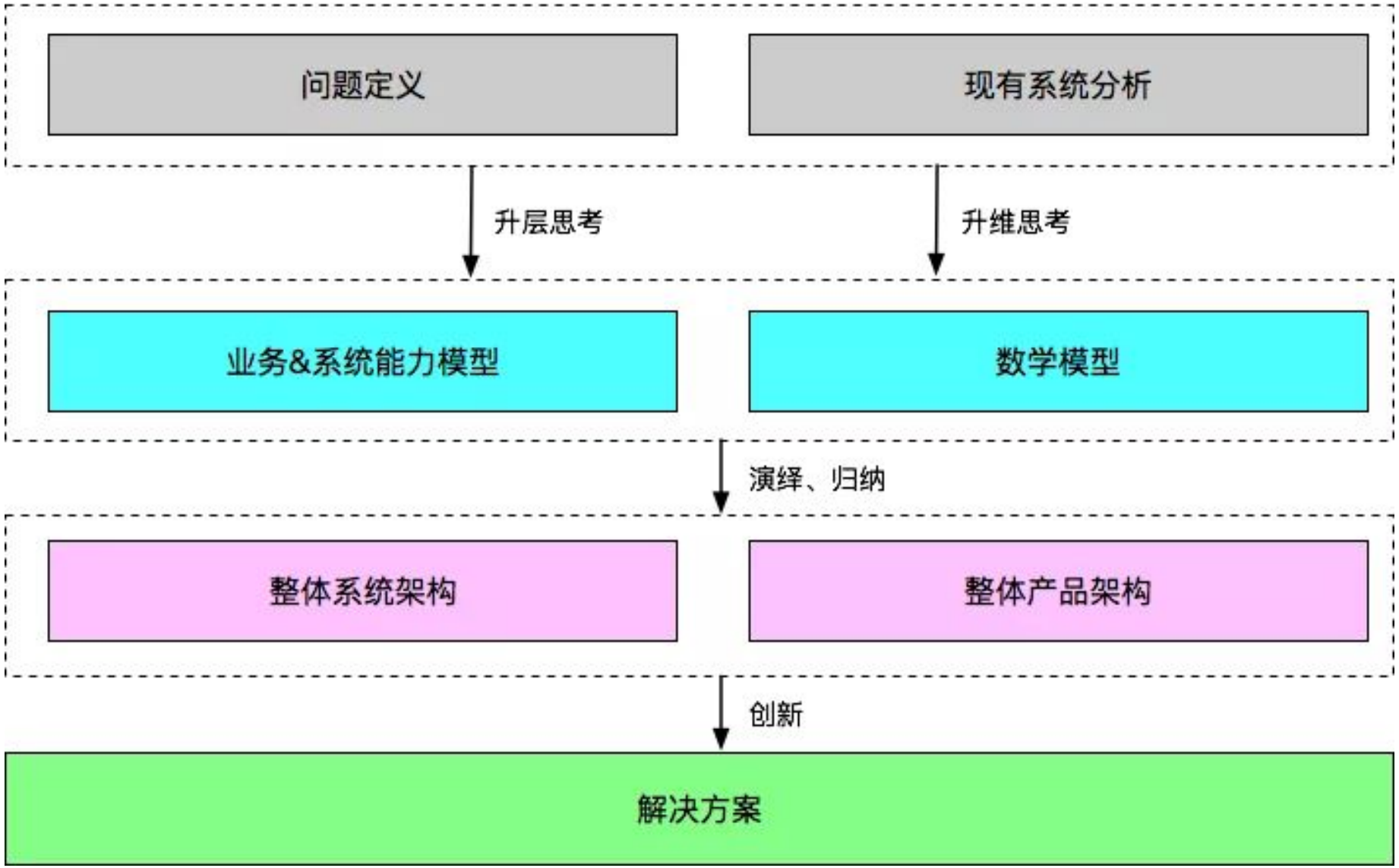
5.善用多种方法对客户问题进行分析，转换成我们产品或者平台需要提供的能力，比如仓储系统 WMS 可以提供哪些商业能力。

6.对我们的现有的流程和能力模型进行梳理，找到需要提升的地方，升层思考和升维思考真正明确提升部分。

7.定义指标，并能够对指标进行拆解，然后进行数学建模。

8.将抽象出来的能力诉求转换成技术挑战，此步对于技术人员来说相当于找到了靶子，可以进行方案的设计了，需要结合自底向上的架构推导方式。

9.创新可以是业务创新，也可以是产品创新，也可以是技术创新，也可以是运营创新，升层思考、升维思考，使用第一性原理思维、生物学（进化论--进化=变异+选择+隔离、熵增定律、分形和涌现）思维等哲科思维可以帮助我们在业务，产品，技术上发现不同的创新可能。可以说哲科思维是架构师的灵魂思维。


问题定义
现有系统分析
升层思考
升维思考
业务&系统能力模型
数学模型
演绎、归纳
整体系统架构
整体产品架构
创新
解决方案

自底向上推导应用架构

先根据业务流程，分解出系统时序图，根据时序图开始对模块进行归纳，从而得到粒度更大的模块，模块的组合 / 聚合构建整个系统架构。

基本上应用逻辑架构的推导有4个子路径，他们分别是：

1. 业务概念架构：业务概念架构来自于业务概念模型和业务流程；

2. 系统模型：来自于业务概念模型；

3. 系统流程：来自业务流程；

4. 非功能性的系统支撑：来自对性能、稳定性、成本的需要。

效率、稳定性、性能是最影响逻辑架构落地成物理架构的三大主要因素，所以从逻辑架构到物理架构，一定需要先对效率、稳定性和性能做出明确的量化要求。

自底向上重度依赖于演绎和归纳。

如果是产品方案已经明确，程序员需要理解这个业务需求，并根据产品方案推导出架构，此时一般使用自底向上的方法，而领域建模就是这种自底向上的分析方法。

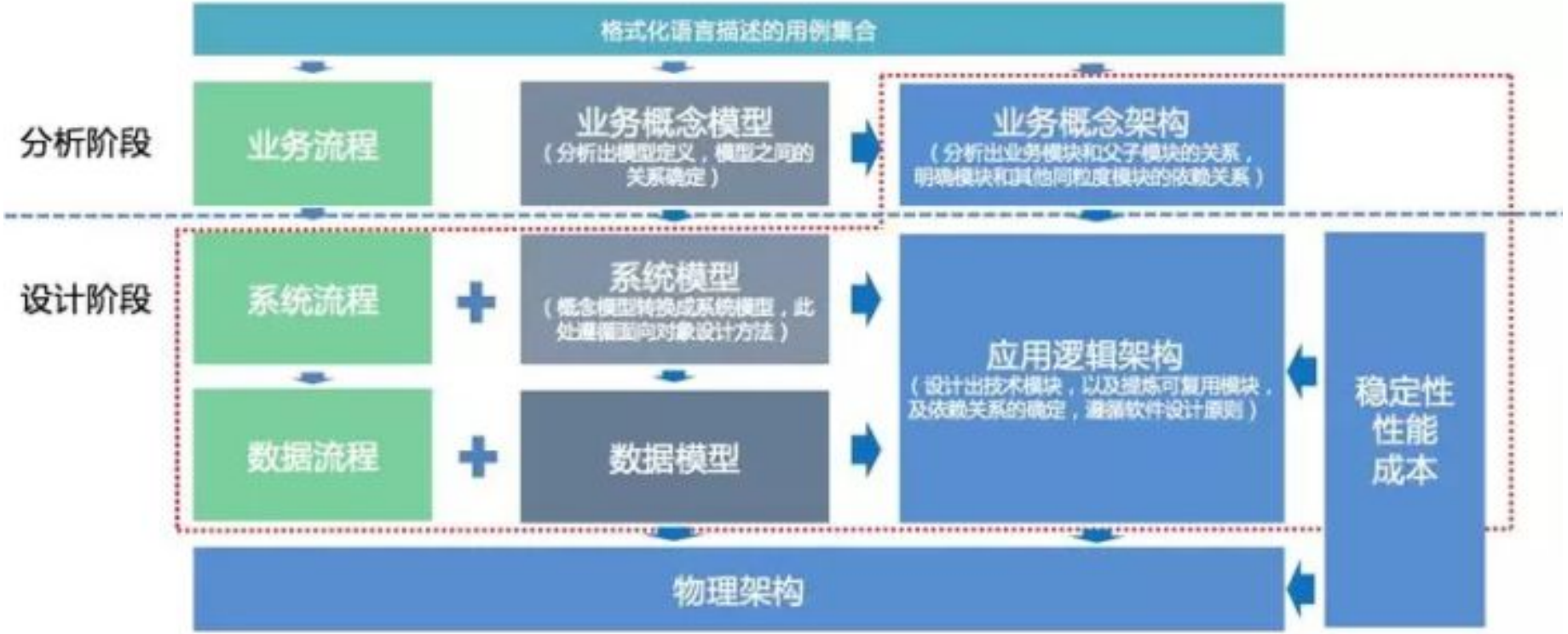
对于自底向上的分析方法，如果提炼一下关键词，会得到如下两个关键词：

1.演绎：演绎就是逻辑推导，越是底层的，越需要演绎：

- 从用例到业务模型就属于演绎；
- 从业务模型到系统模型也属于演绎；
- 根据目前的问题，推导出要实施某种稳定性措施，这是也是演绎。

2.归纳：这里的归纳是根据事物的某个维度来进行归类，越是高层的，越需要归纳：

- 问题空间模块划分属于归纳；
- 逻辑架构中有部分也属于归纳；
- 根据一堆稳定性问题，归纳出，事前，事中，事后都需要做对应的操作，是就是根据时间维度来进行归纳。



领域驱动设计架构

大部分传统架构都是基于领域模型分析架构，典型的领域实现模型设计可以参考DDD（领域驱动设计），详细可以参考《实现领域驱动设计》这本书，另外《UML和模式应用》在领域建模实操方面比较好，前者偏理论了解，后者便于落地实践。

领域划分设计步骤：

1.对用户需求场景分析，识别出业务全维度 Use Case；

2.分析模型鲁棒图，识别出业务场景中所有的实体对象。鲁棒图 —— 是需求设计过程中使用的一种方法（鲁棒性分析），通过鲁棒分析法可以让设计人员更清晰，更全面地了解需求。它通常使用在需求分析后及需求设计前做软件架构分析之用，它主要注重于功能需求的设计分析工作。需求规格说明书为其输入信息，设计模型为其输出信息。它是从功能需求向设计方案过渡的第一步，重点是识别组成软件系统的高级职责模块、规划模块之间的关系。鲁棒图包含三种图形：边界、控制、实体，三个图形如下：

3、领域划分，将所有识别出的实体对象进行分类；

4、评估域划分合理性，并进行优化。

基于数据驱动设计架构

随着 IoT、大数据和人工智能的发展，以领域驱动的方式进行架构往往满足不了需求或者达不到预期的效果，在大数据时代，在大数据应用场景，我们需要转变思维，从领域分析升维到基于大数据统计分析结果来进行业务架构、应用架构、数据架构和技术架构。这里需要架构师具备数理统计分析的基础和 BI 的能力，以数据思维来架构系统，典型的系统像阿里的数据分析平台采云间和菜鸟的数据分析平台 FBI。

上述四种思维，往往在架构设计中是融合使用的，需要根据业务或者系统的需求来选择侧重思维方式。

有了架构思维的指导，具体有没有通用 / 标准化的架构框架以更好的执行架构设计？请看常见的架构框架。下述的架构框架其实本身也包含了重要的一些架构思维。

常见架构框架

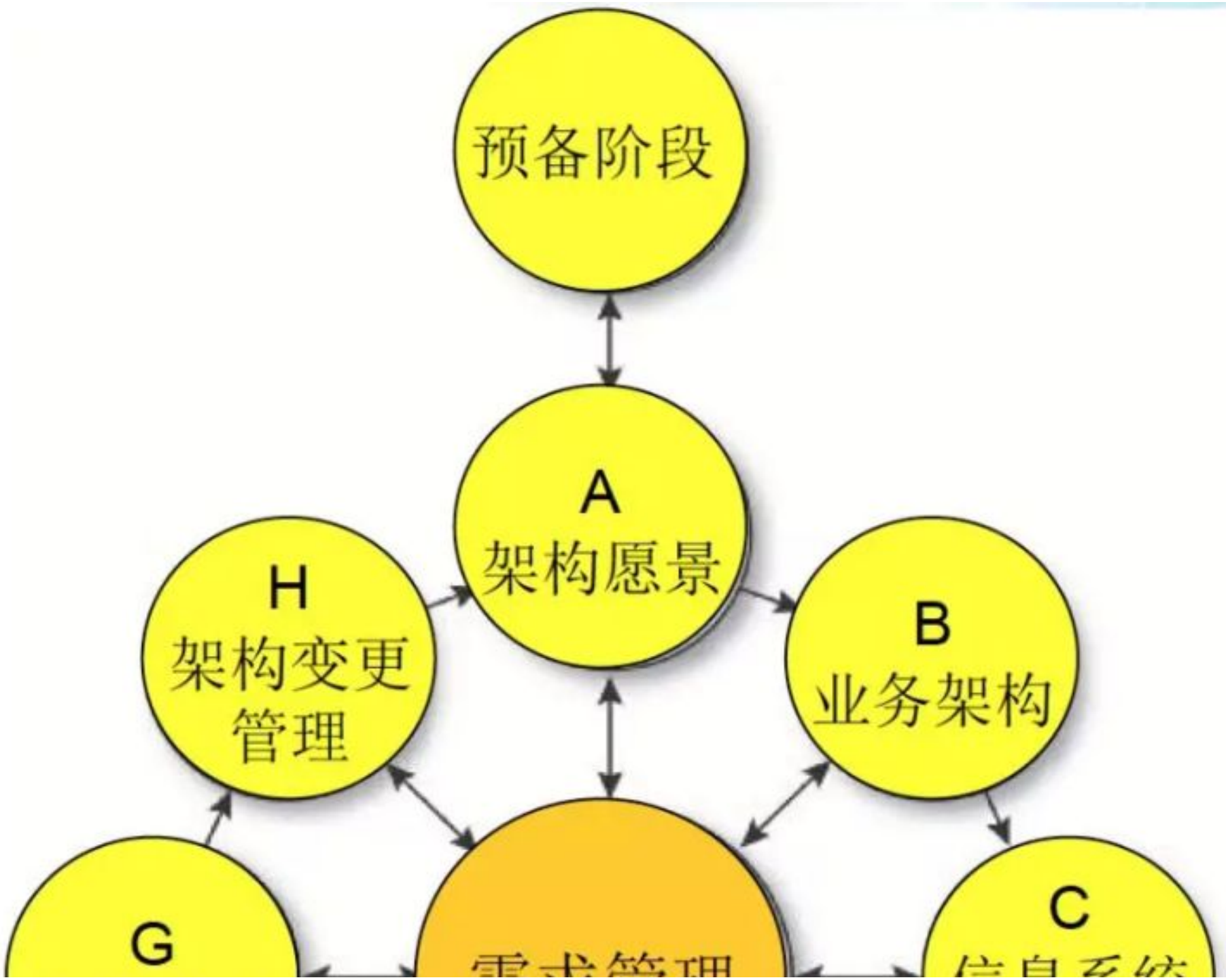
TOGAF

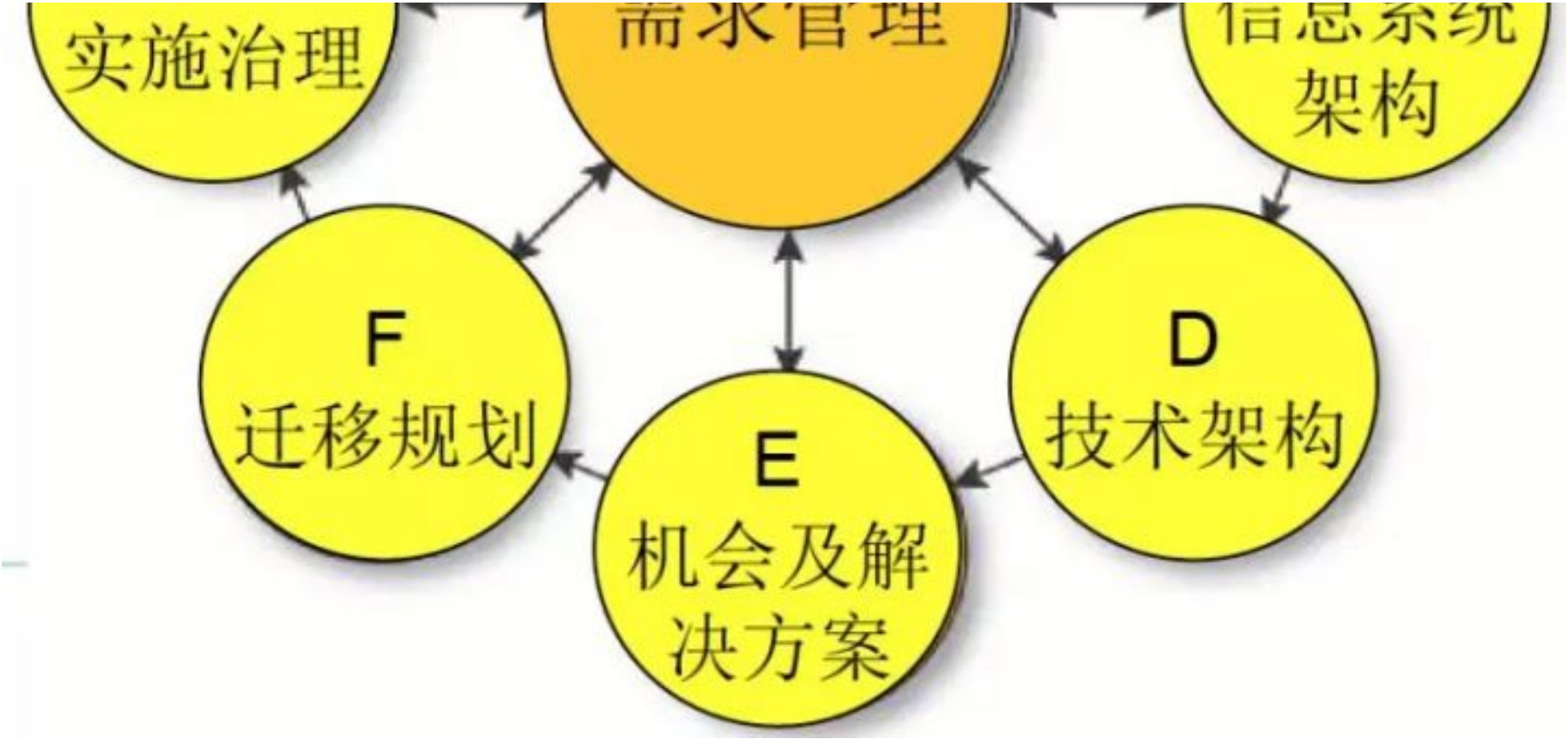
TOGAF 是 The Open Group Architecture Framework 的缩写，它由 The Open Group 开发，The Open Group 是一个非盈利的技术行业联盟，它不断更新和重申 TOGAF。

TOGAF 强调商业目标作为架构的驱动力，并提供了一个最佳实践的储藏库，其中包括 TOGAF 架构开发方法（ADM）、TOGAF 架构内容框架、TOGAF 参考模型、架构开发方法（ADM）指引和技术、企业连续统一体和 TOGAF 能力框架。

ADM

ADM是一个迭代的步骤顺序以发展企业范围的架构的方法。


预备阶段
A
架构愿景
H
架构变更
管理
B
业务架构
G
C



架构内容框架

- 提供了一套架构工作产品的详细模型，包括**交付物**，交付物内的**制品**，以及交付物代表的**架构构建块**（ABBs）。
  - 它驱使TOGAF的产出物有更强的一致性
  - 它提供了一个全面的架构产出清单
  - 它有利于更好地整合工作产品
  - 它提供了详细架构应如何予以说明的开放式标准
  - 它包括一个详细的元模型

TOGAF™

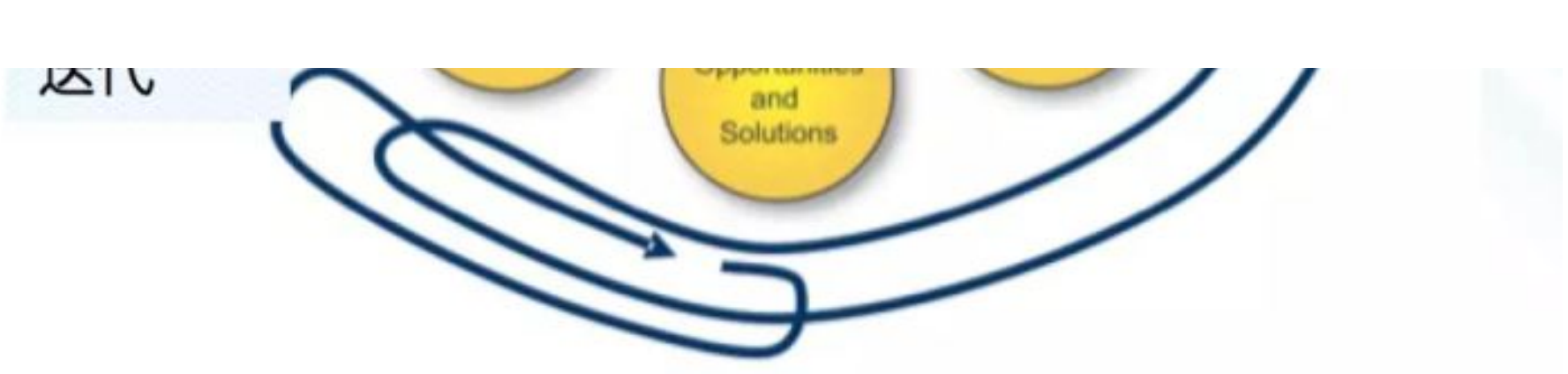
参考模型


质量
基础设施应用
业务应用
应用平台接口
质量
系统及网络管理
软件工程
安全
事务处理
位置与目录
国际操作
用户界面
数据交换
数据管理
图形及影像
质量
操作系统服务
网络服务

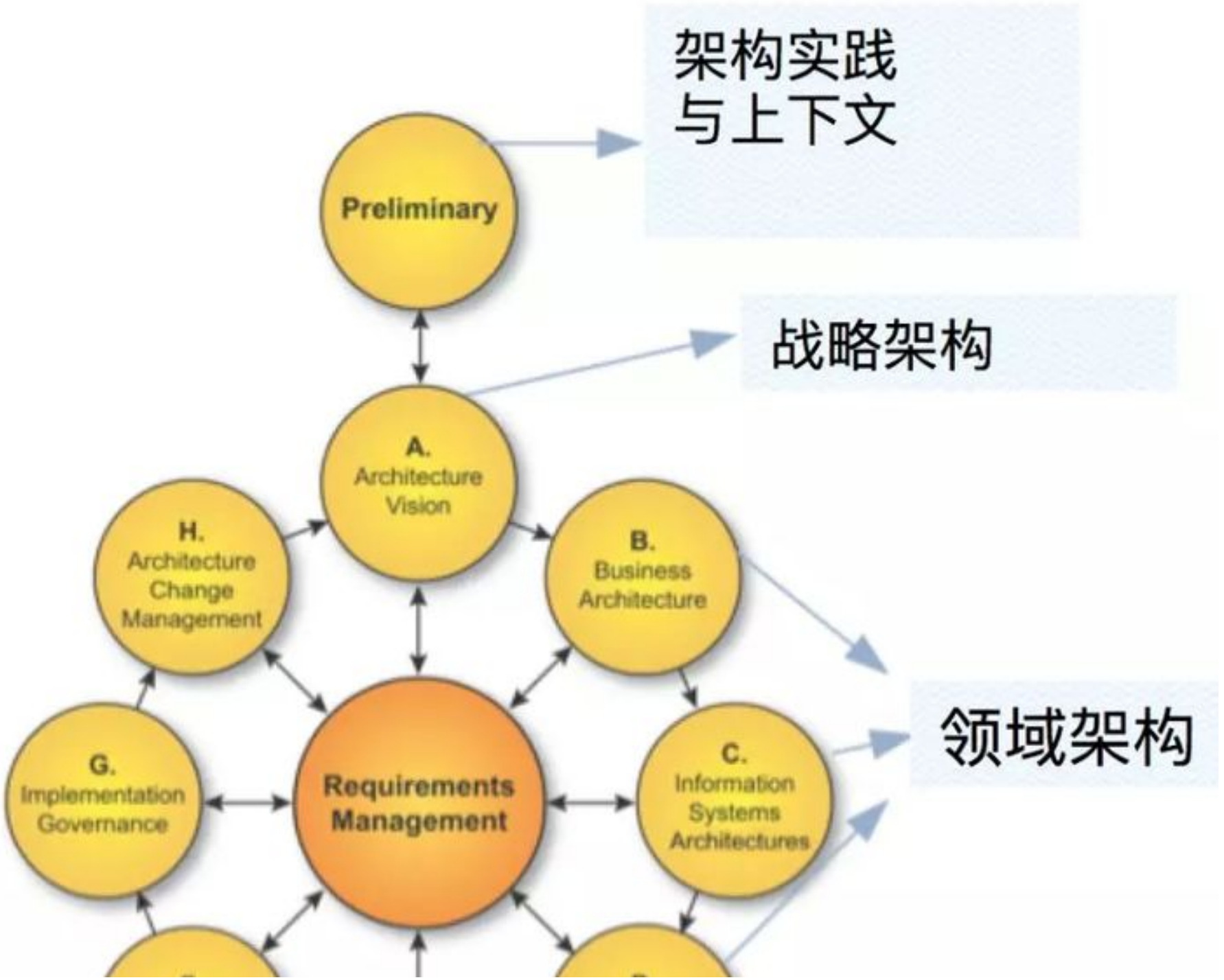
ADM指引和技术

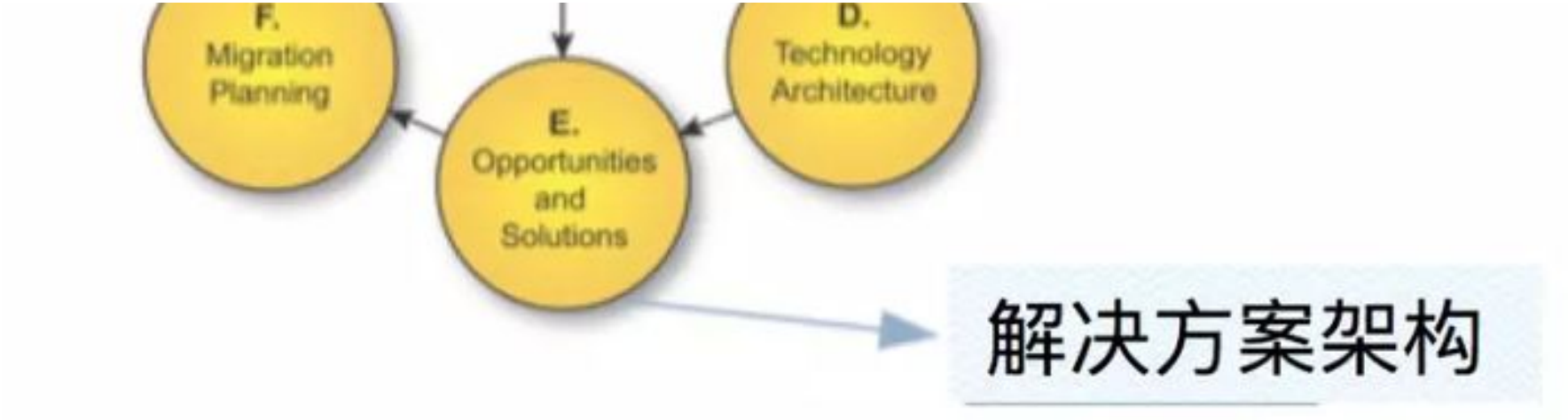
1、架构迭代阶段：

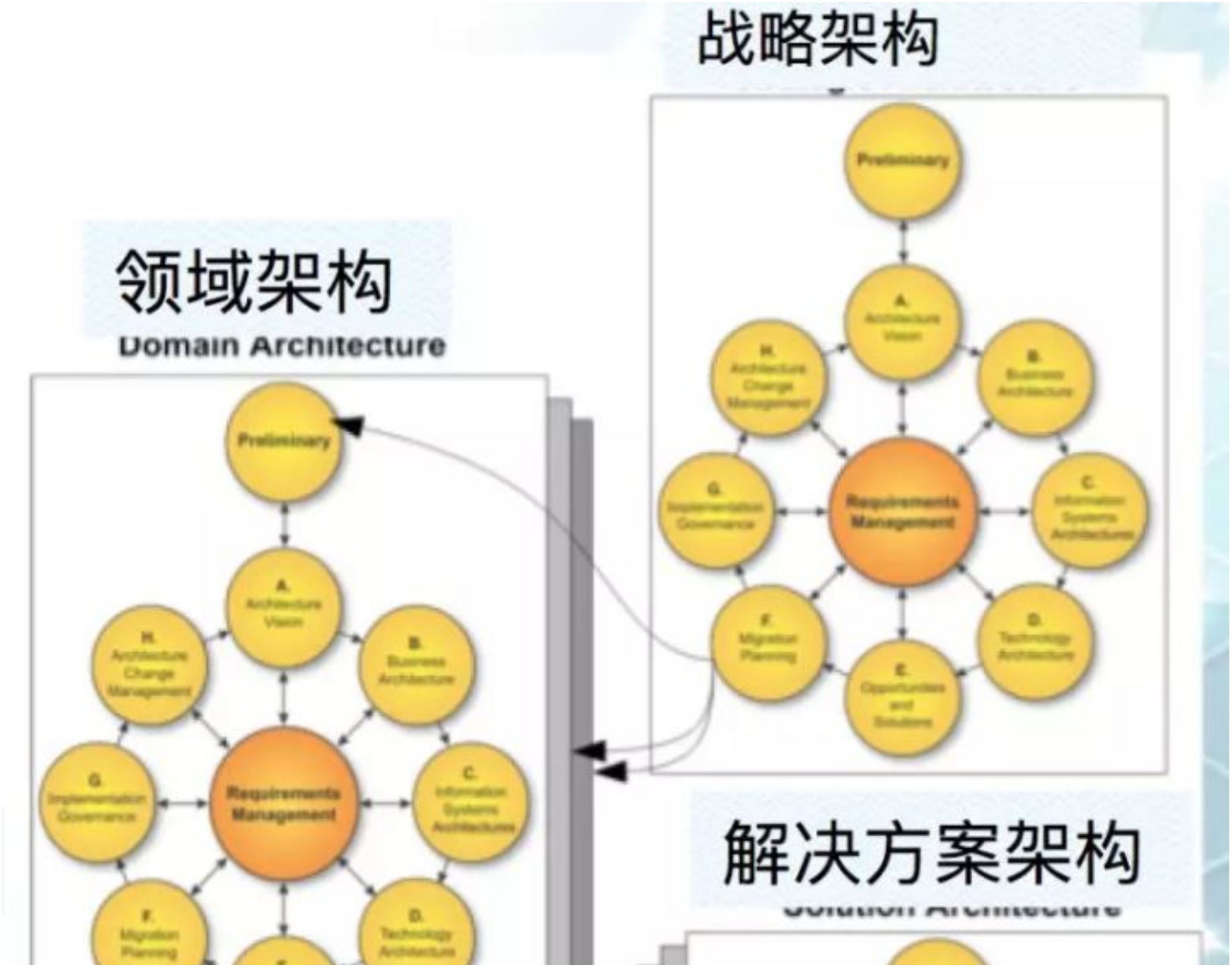

架构
上下文
迭代
架构
治理
迭代
架构
定义
迭代
过渡
规划
迭代
Preliminary
A.
Architecture
Vision
B.
Business
Architecture
C.
Information
Systems
Architectures
D.
Technology
Architecture
E.
Opportunities
F.
Migration
Planning
G.
Implementation
Governance
H.
Architecture
Change
Management
Requirements
Management

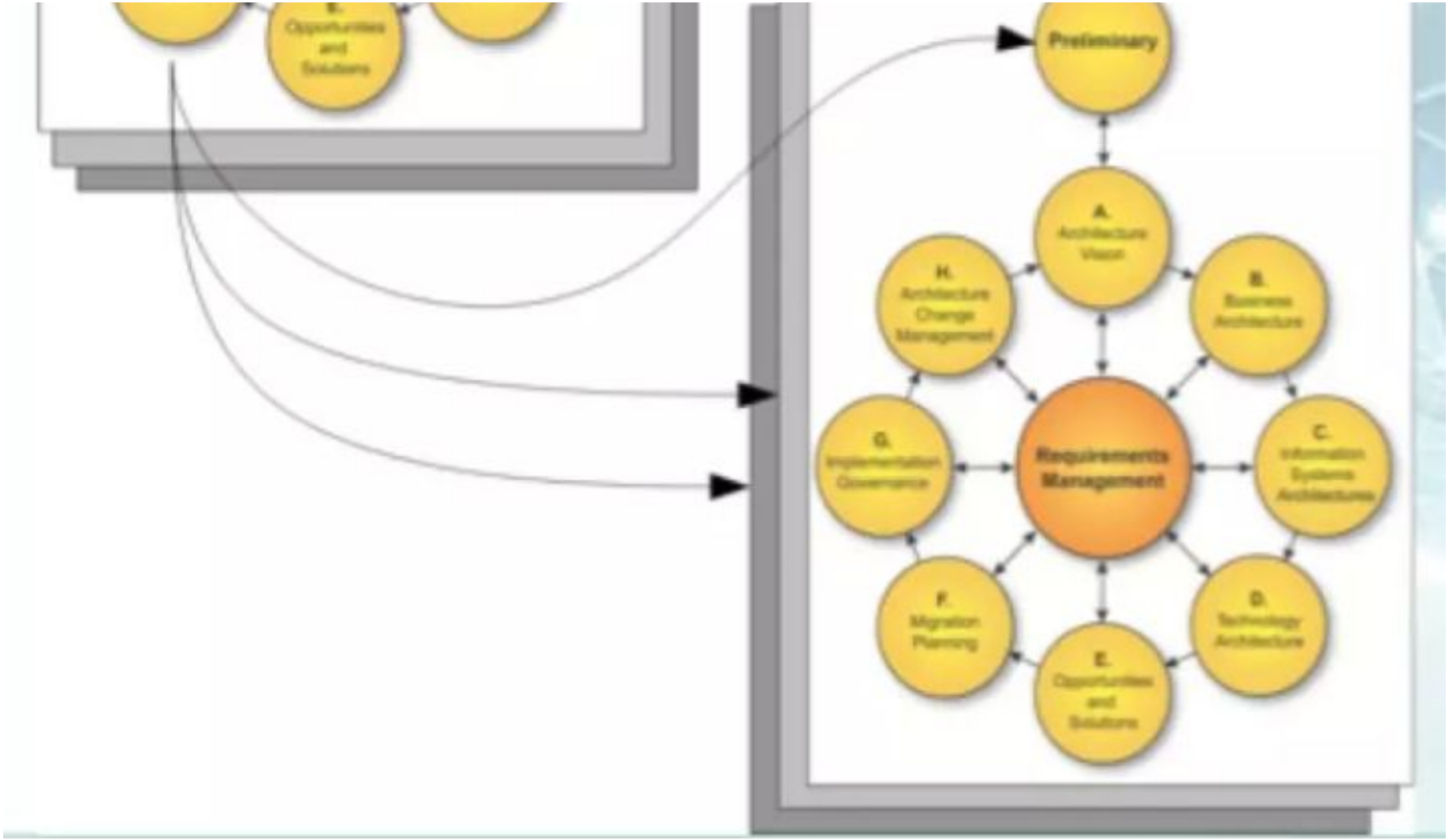


2、在不同水平运用ADM：


架构实践
与上下文
Preliminary
战略架构
A.
Architecture
Vision
H.
Architecture
Change
Management
B.
Business
Architecture
G.
Implementation
Governance
Requirements
Management
C.
Information
Systems
Architectures
领域架构


F.
Migration
Planning
D.
Technology
Architecture
E.
Opportunities
and
Solutions
解决方案架构


战略架构
领域架构
Domain Architecture
Preliminary
A.
Architecture Vision
B.
Business Architecture
C.
Information Systems Architectures
D.
Technology Architecture
E.
Opportunities and Solutions
F.
Migration Planning
G.
Implementation Governance
H.
Architecture Change Management
Requirements Management
解决方案架构


Preliminary
A.
Architecture Vision
B.
Business Architecture
C.
Information Systems Architectures
D.
Technology Architecture
E.
Opportunities and Solutions
F.
Migration Planning
G.
Implementation Governance
H.
Architecture Change Management
Requirements Management

3、利益相关者分类：

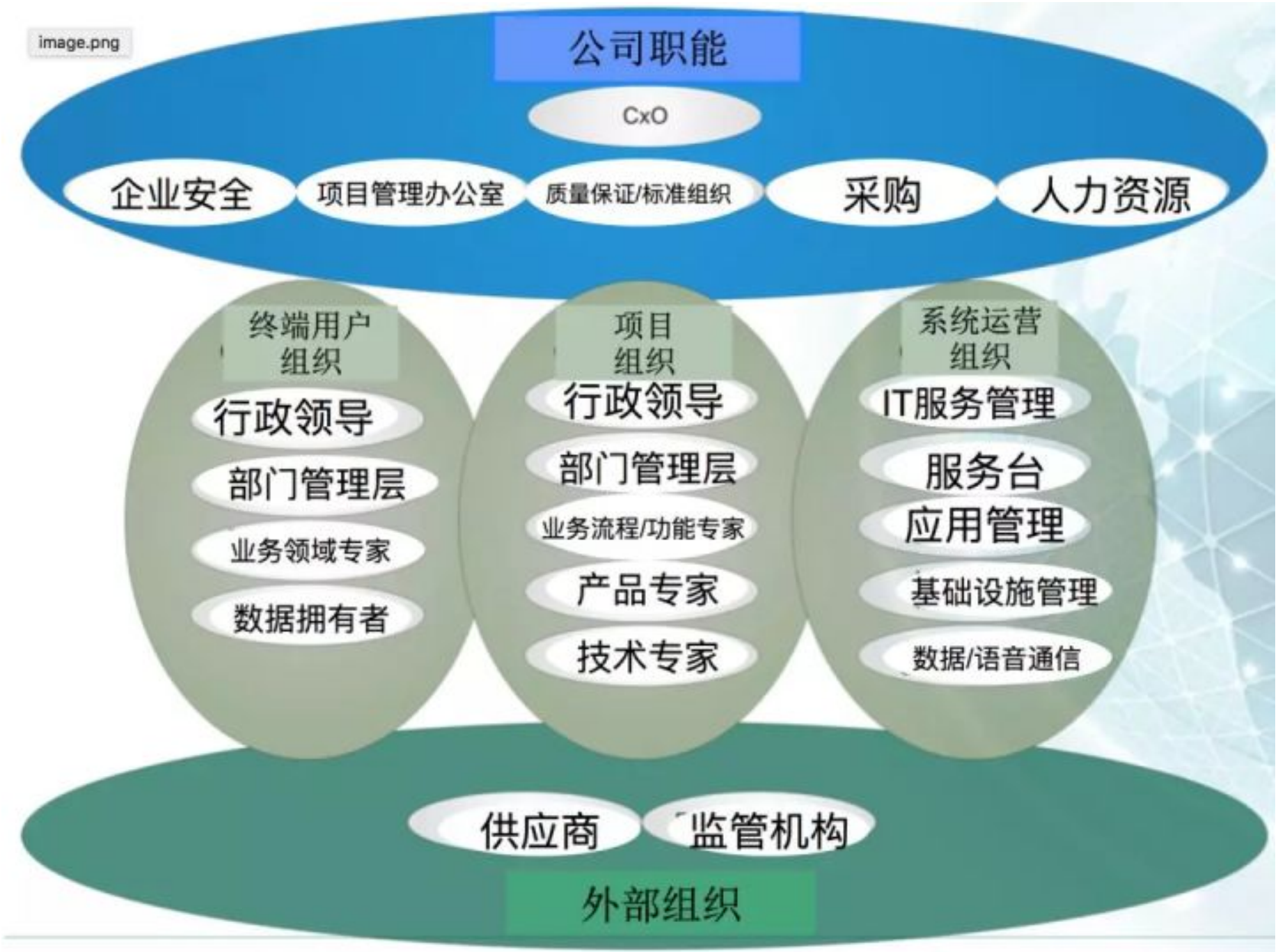
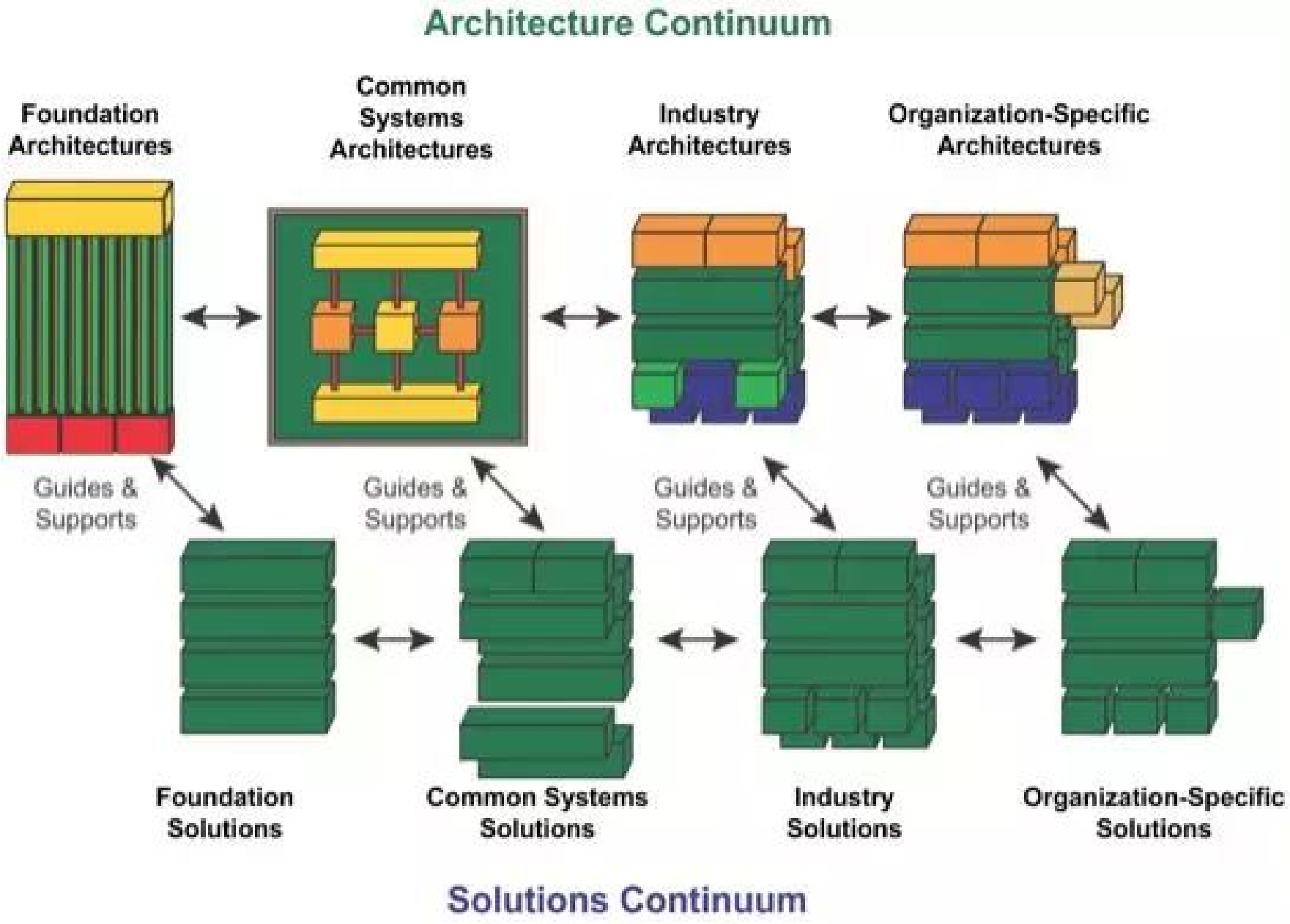


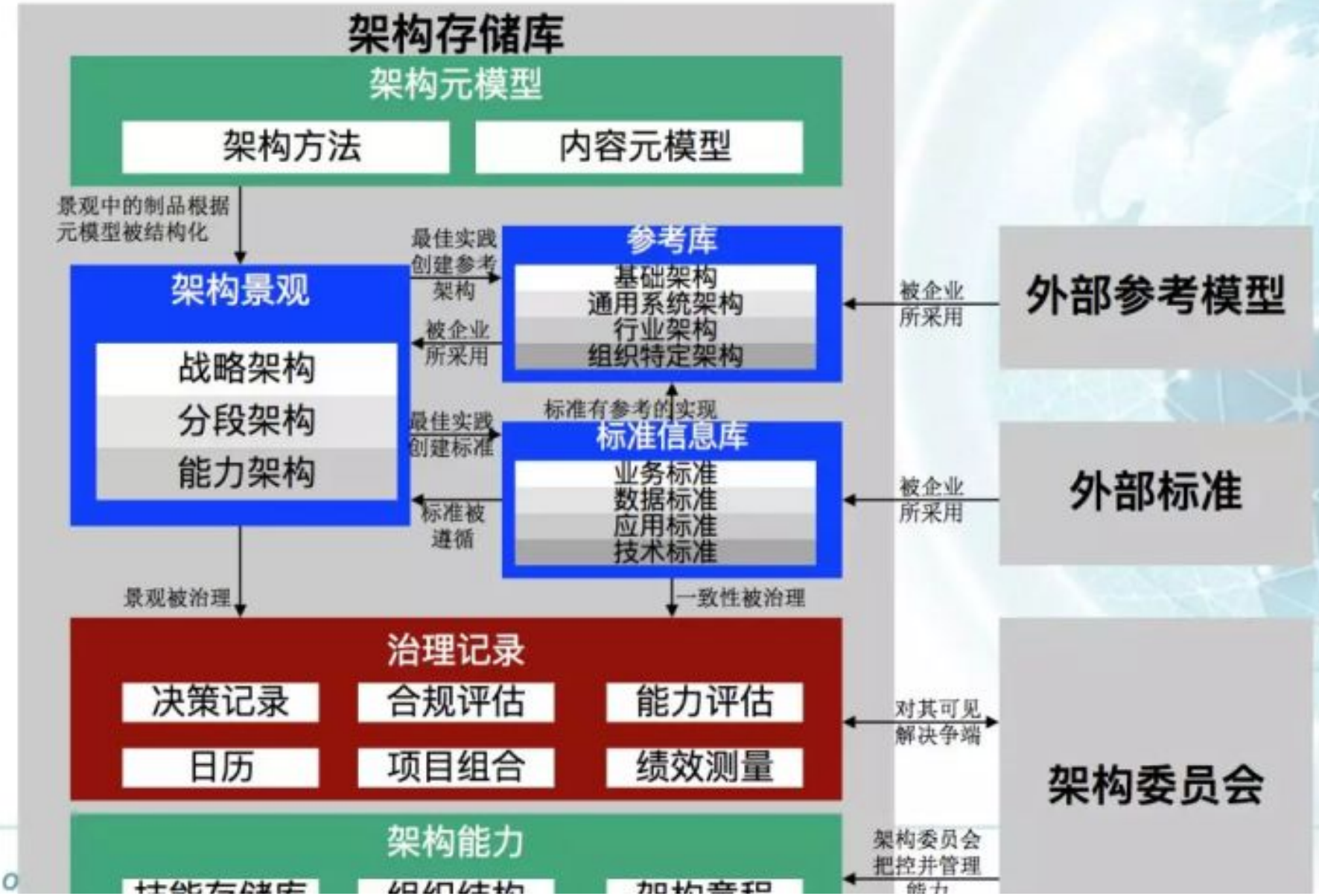
image.png
公司职能
CxO
企业安全
项目管理办公室
质量保证/标准组织
采购
人力资源
终端用户组织
行政领导
部门管理层
业务领域专家
数据拥有者
项目组织
行政领导
部门管理层
业务流程/功能专家
产品专家
技术专家
系统运营组织
IT服务管理
服务台
应用管理
基础设施管理
数据/语音通信
供应商
监管机构
外部组织

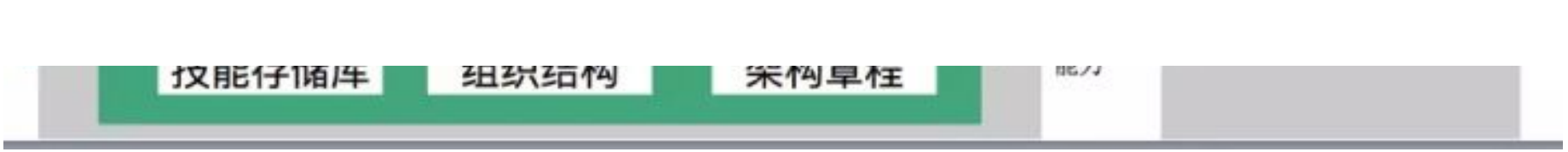
企业连续统一体

架构指导及支持解决方案：基础 通用系统 行业组织特定

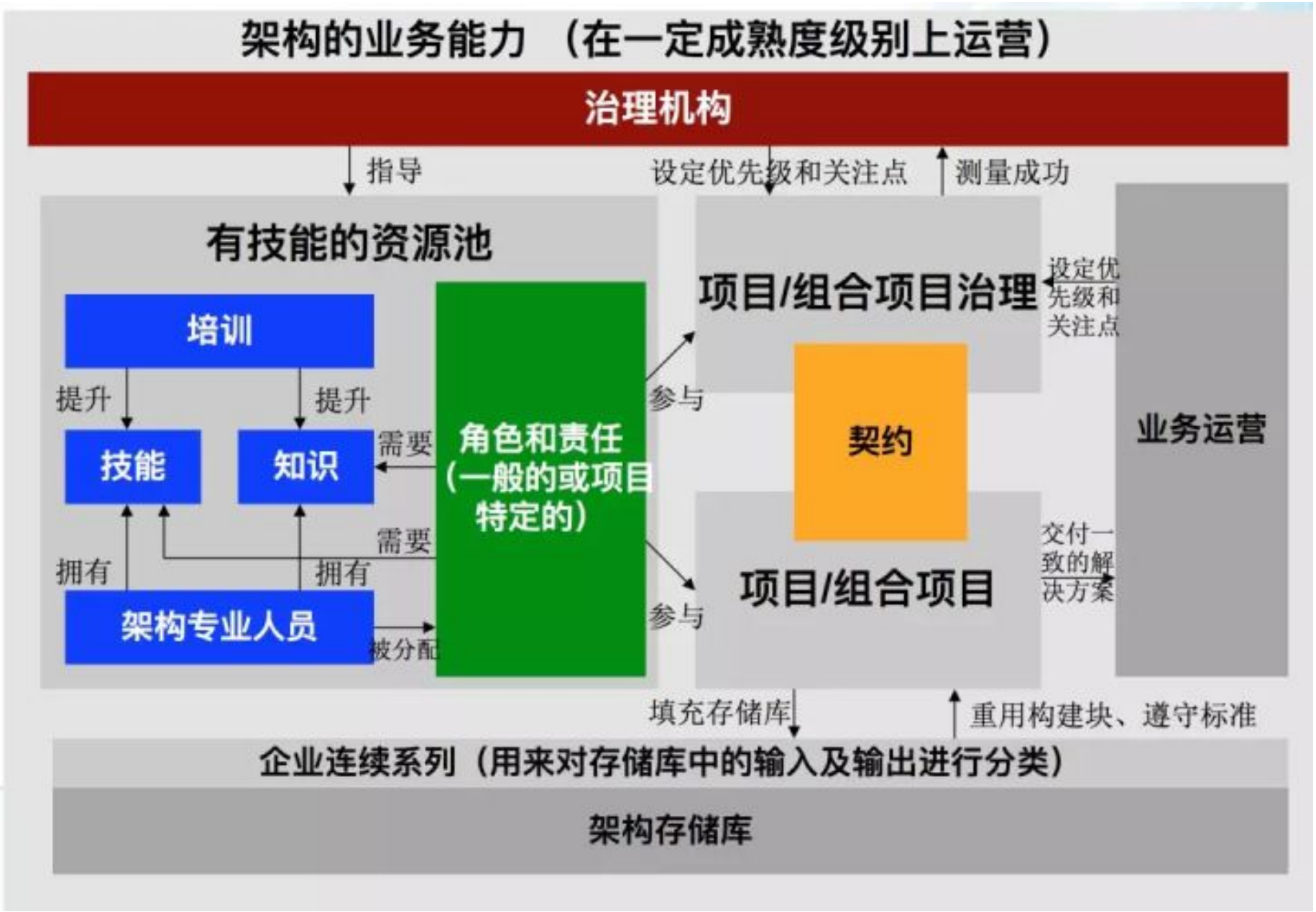

Architecture Continuum
Foundation Architectures
Common Systems Architectures
Industry Architectures
Organization-Specific Architectures
Guides & Supports
Guides & Supports
Guides & Supports
Guides & Supports
Foundation Solutions
Common Systems Solutions
Industry Solutions
Organization-Specific Solutions
Solutions Continuum

架构存储库


架构存储库
架构元模型
架构方法
内容元模型
景观中的制品根据元模型被结构化
架构景观
战略架构
分段架构
能力架构
最佳实践创建参考架构
被企业所采用
参考库
基础架构
通用系统架构
行业架构
组织特定架构
被企业所采用
外部参考模型
标准有参考的实现
最佳实践创建标准
标准信息库
业务标准
数据标准
应用标准
技术标准
标准被遵循
被企业所采用
外部标准
景观被治理
一致性被治理
治理记录
决策记录
合规评估
能力评估
日历
项目组合
绩效测量
对其可见解决争端
架构委员会
架构能力
架构委员会把控并管理



能力框架



(更多内容可以参考《TOGAF标准9.1版本》或者

https://www.opengroup.org/togaf)

Zachman

第一个最有影响力的框架方法论就是 Zachman 框架，它是 John Zachman 首次在1987年提出的。

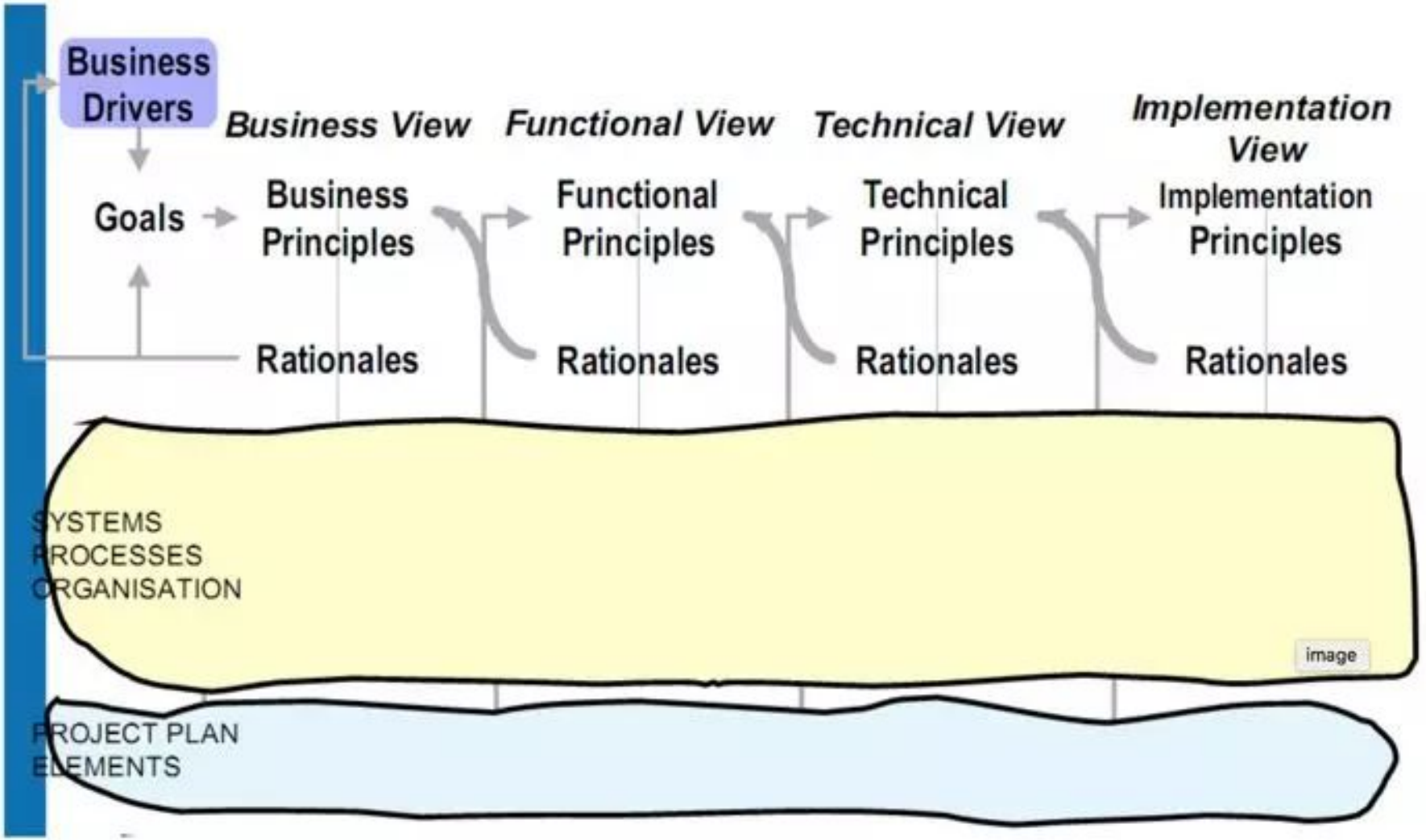
Zachman 框架模型分两个维度：横向维度采用6W（what、how、where、who、when、why）进行组织，纵向维度反映了 IT 架构层次，从上到下（Top-Down），分别为范围模型、企业模型、系统模型、技术模型、详细模型、功能模型。横向结合 6W，Zachman 框架分别由数据、功能、网络、人员、时间、动机分别对应回答What、How、Where、Who、When 与 Why 这六个问题。

	数据（什么？）	功能（怎样？）	网络（哪里？）	角色（谁？）	时间（何时？）	动机（为何？）
目标范围	列出对业务至关重要的元素	列出业务执行的流程	列出与业务运营有关的地域分布要求	列出对业务重要的组织部门	列出对业务重要的事件及时间周期	列出企业目标、战略
业务模型	实体关系图（包括M: M关系、N-ary关系、归因关系）	业务流程模型（物理数据流程图）	物流网络（节点和链接）	基于角色的组织层次图，包括相关技能规定、安全保障问题。	业务主进度表	业务计划
信息系统模型	数据模型（聚合体、完全规格化）	关键数据流程图、应用架构	分布系统架构	人机界面架构（角色、数据、入口）	相依关系图、数据实体生命历程（流程结构）	业务标准模型
技术模型	数据架构（数据库中的表格列表及属性）、遗产数据图	系统设计：结构图、伪代码	系统架构（硬件、软件类型）	用户界面（系统如何工作）、安全设计	"控制流"图（控制结构）	业务标准设计
详细展现	数据设计（反向规格化）、物理存储器设计	详细程序设计	网络架构	屏显、安全机构（不同种类数据源的开放设定）	时间、周期定义	程序逻辑的角色说明
功能系统	转化后的数据	可执行程序	通信设备	受训的人员	企业业务	强制标准

ITSA

ITSA诞生于1986年的惠普，世界最早的企业架构框架(IT战略与架构)。建模原则就是“Everything you need, and nothing you don’t”，只放你要的东西。

Architectural coherence part1



Architectural coherence part2

Business Strategy

Business Processes

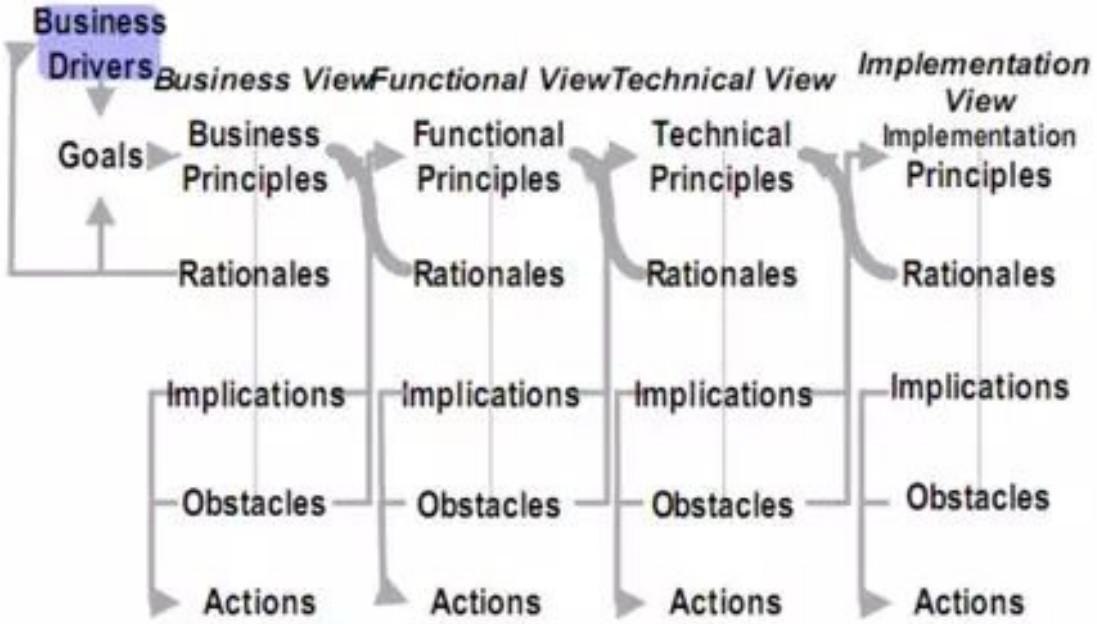
Data

Applications

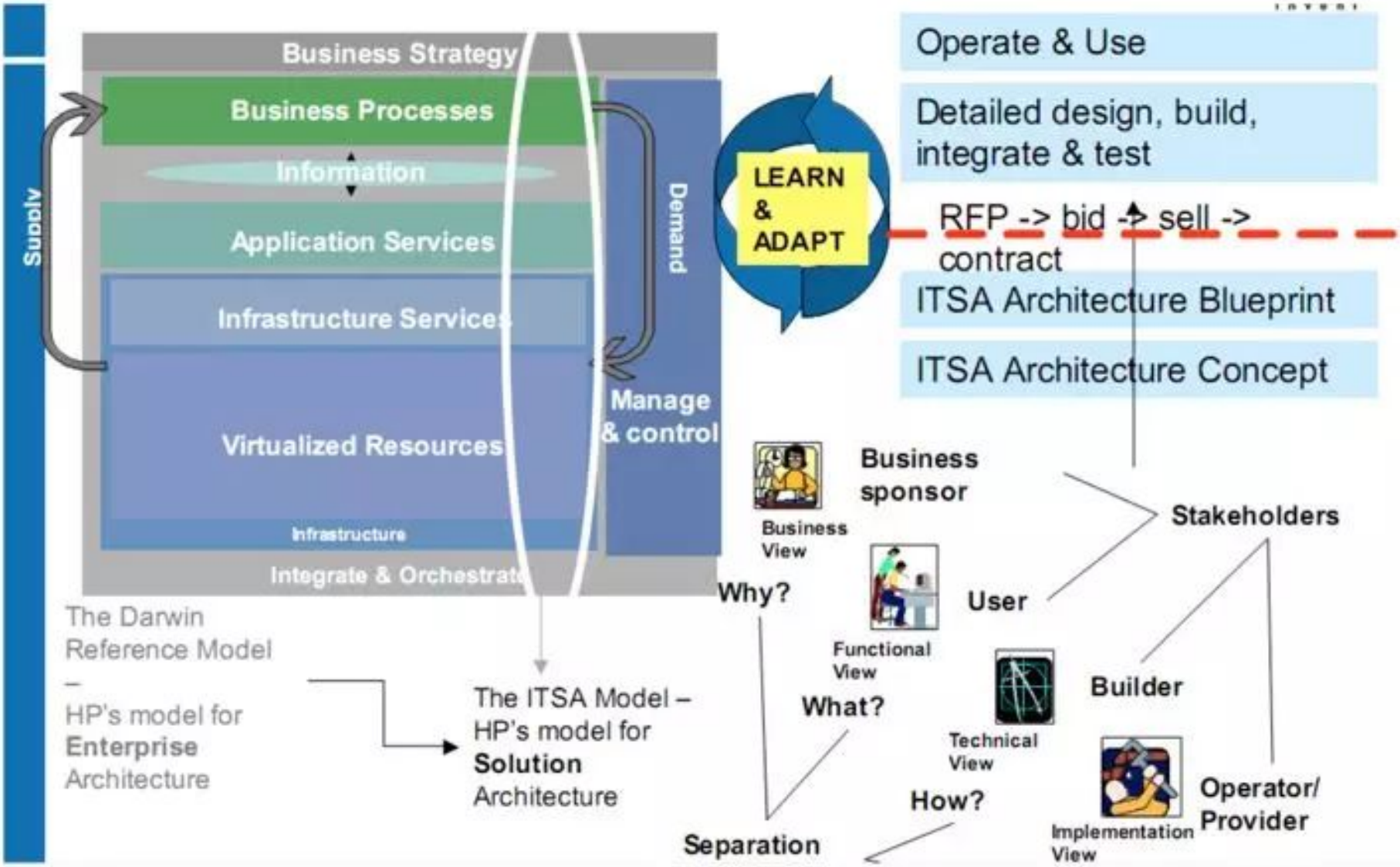
Generic User Services

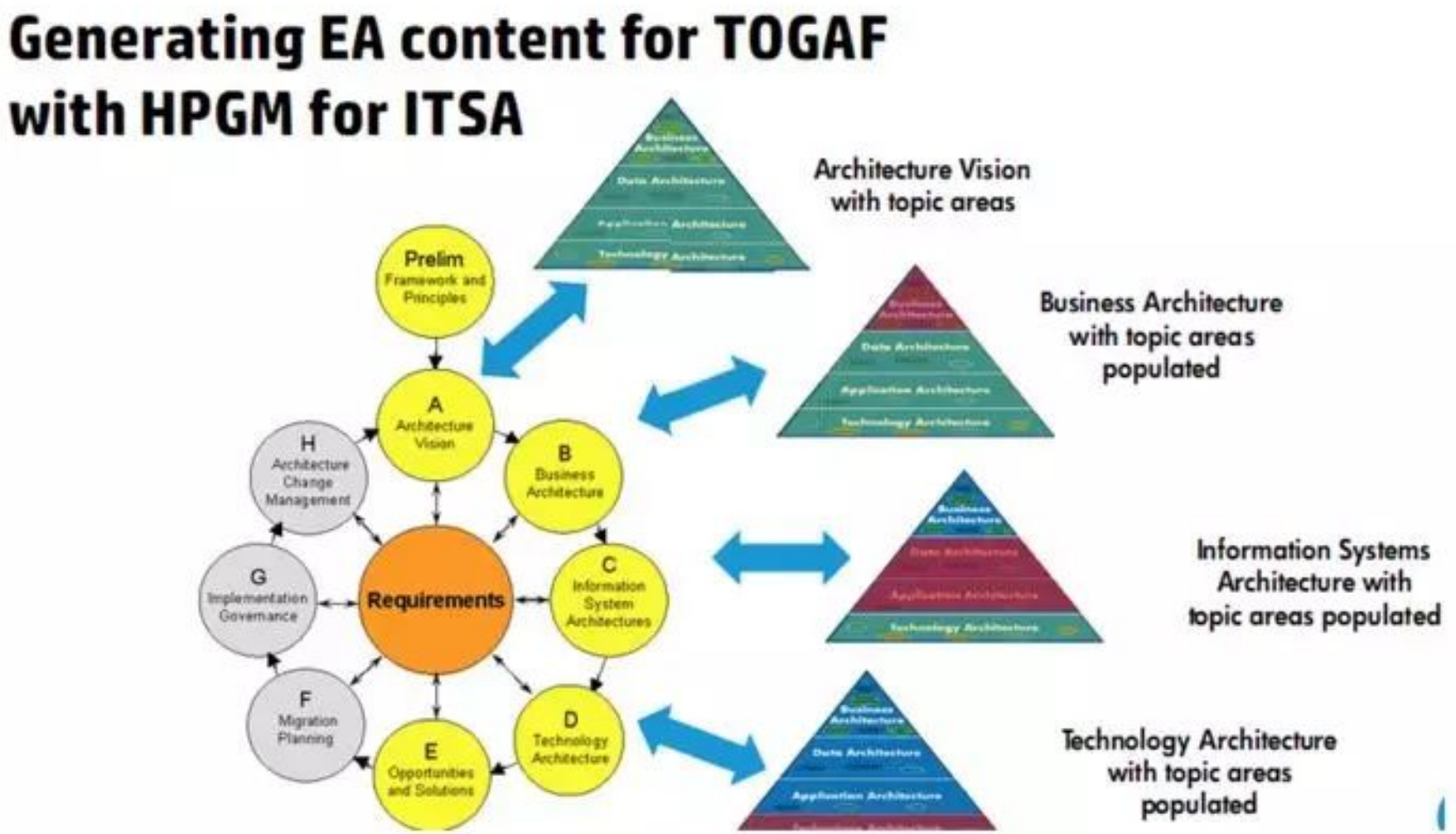
Technical Infrastructure

- Architects to help stakeholders use the EA elements to shape the solution
- A Governing Body (Steering Group, Policy Board) to manage architectural compliance



SA and EA


Supply
Business Strategy
Business Processes
Information
Application Services
Infrastructure Services
Virtualized Resources
Infrastructure
Integrate & Orchestrate
Demand
Manage & control
LEARN & ADAPT
Operate & Use
Detailed design, build, integrate & test
RFP -> bid -> sell -> contract
ITSA Architecture Blueprint
ITSA Architecture Concept
Business sponsor
Business View
Stakeholders
Why?
User
Functional View
What?
Builder
Technical View
How?
Operator/ Provider
Implementation View
Separation
The Darwin Reference Model – HP's model for Enterprise Architecture
The ITSA Model – HP's model for Solution Architecture

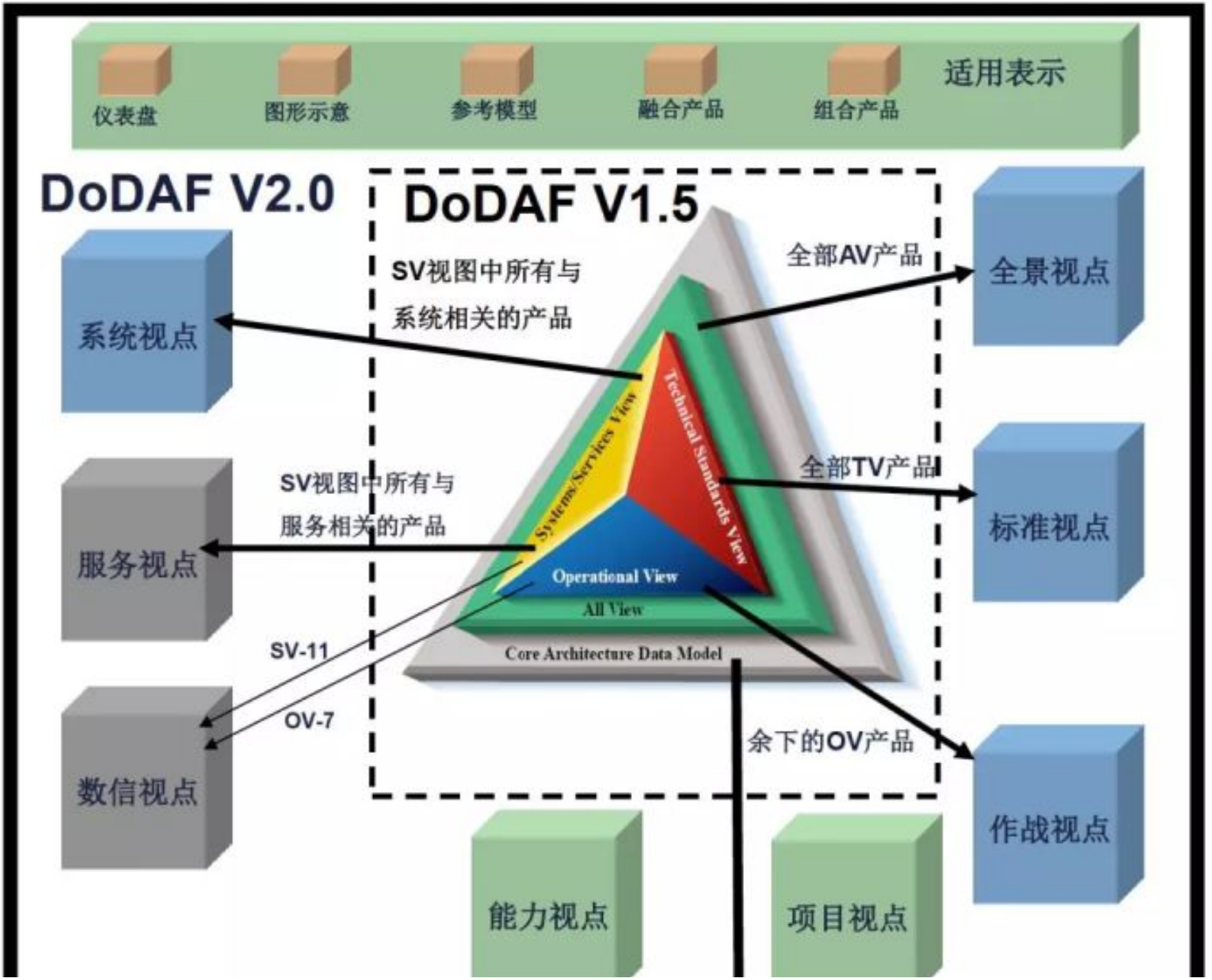


DODAF

DODAF 是美国国防部架构框架，是一个控制"EA开发、维护和决策生成"的组织机制，是统一组织"团队资源、描述和控制EA活动"的总体结构。

DODAF 涵盖 DoD 的所有业务领域，定义了表示、描述、集成 DoD 范围内众多架构的标准方法，确保架构描述可比较、评估，提供了对 FoS (系统族)和 SoS (体系)进行理解、比较、集成和互操作共同的架构基础，提供开发和表达架构描述的规则和指南,但不指导如何实现。

DODAF 核心是8个视点和52个模型。


仪表盘
图形示意
参考模型
融合产品
组合产品
适用表示
DoDAF V2.0
DoDAF V1.5
系统视点
SV视图中所有与
系统相关的产品
全部AV产品
全景视点
Technical Standards View
Systems/Services View
Operational View
All View
Core Architecture Data Model
SV视图中所有与
服务相关的产品
全部TV产品
服务视点
标准视点
SV-11
OV-7
数信视点
余下的OV产品
作战视点
能力视点
项目视点

1.全景视点 AV

与所有视点相关的体系结构描述的顶层概貌。提供有关体系结构描述的总体信息，诸如体系结构描述的范围和背景。范围包括体系结构描述的专业领域和时间框架。背景由构成体系结构描述背景的相互关联各种条件组成，包括条令，战术、技术和程序，相关目标和构想的描述，作战概念(CONOPS)，想定和环境条件。

AV-1：概要信息	描述项目的构想、目标、任务、计划、行动、事件、条件、度量、效果（结果）和引申的目标。
AV-2：集成词典	存放体系结构数据的仓库，定义了在体系结构数据和表示中使用的所有术语。

2.能力视点CV

能力视点(CV)集中反映了与整体愿景相关的组织目标，这些愿景指在特定标准和条件下进行特定行动过程或是达成期望效果的能力，它们综合使用各种手段和方式来完成一组任务。

CV 为体系结构描述中阐述的能力提供了战略背景和相应的高层范围，比作战概念图中定义的基于想定的范围更全面。

这些模型是高层的，用决策者易于理解的术语来描述能力，以便沟通能力演进方面战略构想。

CV-1：能力愿景	军事转型的总体构想，提供所描述能力的战略背景和高层范围。
CV-2：能力分类	规定所有能力的能力等级，在若干体系结构描述中引用。
CV-3：能力阶段	在不同时间点或特定时间段中计划实现的能力。在不考虑执行者和特定区域解决方案的情况下，按照行动、条件、期望效果、遵循的规则、资源消耗与生产以及度量等方面对能力进行的阶段划分。
CV-4：能力依赖	计划的能力和能力逻辑分组定义之间的依赖性。
CV-5：能力-组织开发映射	能力需求的完成显示了特定能力阶段计划能力的部署和相互联系。CV-5显示了根据执行者和特定区域及其相关概念所计划的该阶段解决方案。
CV-6：能力-作战行动映射	能力与其支撑的作战行动之间的映射关系。
CV-7：能力-服务映射	能力与实现该能力的服务之间的映射关系。

3.作战视点OV

作战视点(OV)集中反映了完成 DoD 使命的机构、任务或执行的行动以及彼此间必须交换的信息。描述信息交换的种类、频度、性质，信息交换支持哪些任务和活动。

OV-1：高层作战概念图	用图形或文本描述的高层作战概念。
0V-2：作战资源流描述	描述作战活动之间交换的资源流。
0V-3：作战资源流矩阵	描述交换的资源及其相关属性。
0V-4：组织机构关系图	各组织机构之间的隶属关系、指挥关系、协同关系、任务或其他关系。
0V-5a：作战活动分解树	以树形结构组织在一起的各种能力和作战活动。
0V-5b：作战活动模型	能力和行动及其在行动、输入和输出中的关系；其他的还能显示成本、执行者或其他相关信息。
OV-6a：作战规则模型	描述作战活动的三个模型之一。它确定作战活动应该遵守的业务规则。
0V-6b：作战状态转换模型	描述作战活动的三个模型之一。它确定响应事件（通常指非常短暂的行动）的业务流程（行动）。
OV-6c：作战事件跟踪描述	描述作战活动的三个模型之一。它描述想定或事件发展过程中执行作战活动的先后顺序。

4.服务视点 SvcV

服务视点(SvcV)集中反映了为作战行动提供支撑的系统、服务和相互交织的功能。DoD 流程包括作战、业务、情报和基础设施功能。SvcV 功能和服务资源及要素可以链接到 0V 中的体系结构数据。这些系统功能和服务资源支撑作战行动，促进信息交换。

SvcV-1：服务背景描述	对服务、服务项及其相互关系的确定。
SvcV-2：服务资源流描述	对服务之间交换的资源流的描述。
SvcV-3a：系统-服务矩阵	描述系统和服务之间的支持关系，即一个服务由哪些系统来提供，一个系统提供哪些服务。
SvcV-3b：服务-服务矩阵	描述服务之间的关系，如服务之间的接口、规划的与现有的接口的对比。
SvcV-4：服务功能描述	描述服务的功能和提供服务时所需的数据流。
SvcV-5：作战活动-服务跟踪矩阵	描述服务与作战活动之间的支持关系，即一个作战活动由哪些服务来支持，一个服务可以支持哪些作战活动。
SvcV-6：服务资源流矩阵	描述服务间交换的服务资源流要素和交换属性的具体细节。
SvcV-7：服务度量矩阵	描述服务在指定时间段的度量指标。
SvcV-8：服务演进描述	规划服务未来演进发展的路线图，确定每一个关键阶段应该提供哪些服务。
SvcV-9：服务技术和技能预测	期望能在某个时间框架范围中获得的并将影响未来服务发展的新兴技术、软/硬件产品和技能。
SvcV-10a：服务规则模型	描述服务功能的三个模型之一。描述服务执行时必需遵守的业务规则。
SvcV-10b：服务状态转换描述	描述服务功能的三个模型之一。描述一个服务有哪些状态，以及服务因响应事件而导致服务从一个状态如何转换到另一个状态的。
SvcV-10c：服务事件跟踪描述	描述服务功能的三个模型之一。描述服务之间如何交互、协作，共同完成某个作战任务或活动的。

5.系统视点 SV

系统视点(SV)集中反映支持作战行动中的自动化系统、相互交联和其他系统功能的信息。随着对面向服务环境和云计算的重视，在 DoDAF 的未来版本中也许不会有系统视点。

SV-1：系统接口描述	对系统、系统项目及其相互关系的确定。
SV-2：系统资源流描述	描述系统之间交换的资源流。
SV-3：系统-系统矩阵	描述系统之间的关系，例如，系统之间的接口，计划的和现有的接口对比。
SV-4：系统功能描述	描述系统的功能及执行功能时所需的系统数据流。
SV-5a：作战活动-系统功能跟踪矩阵	描述系统功能与作战活动之间的支持关系，即一个作战活动由哪些系统功能来支持，一个系统功能可以支持哪些作战活动。
SV-5b：作战活动-系统跟踪矩阵	描述系统与作战活动之间的支持关系，即一个作战活动由哪些系统来支持，一个系统可以支持哪些作战活动。
SV-6：系统资源流矩阵	描述系统之间交换的系统资源流及其要素。
SV-7：系统度量矩阵	描述系统在指定时间段的度量指标。
SV-8：系统演进描述	规划系统未来演进发展的路线图，确定每一个关键阶段系统应该具备的功能。
SV-9：系统技术和技能预测	列出未来某个时间段内可能会影响系统开发的新兴技术、软/硬件产品和技能。
SV-10a：系统规则模型	描述系统功能的三个模型之一。描述系统功能执行时必需遵守的业务规则。
SV-10b：系统状态转换描述	描述系统功能的三个模型之一。描述一个系统有哪些状态，以及系统因响应事件而导致系统从一个状态如何转换到另一个状态的。
SV-10c：系统事件跟踪描述	描述系统功能的三个模型之一。描述系统之间如何交互、协作，共同完成某个作战任务或活动的。

6.数信视点 DIV

数据和信息视点(DIV)，简称数信视点，反映了体系结构描述中的业务信息需求和结构化的业务流程规则。

描述体系结构描述中与信息交换相关的信息，诸如属性、特征和相互关系。
必要时，本视点模型中用到的数据需要由多个架构团队来共同考虑。

DIV-1：概念数据模型	所需的高层数据概念及其相互关系。
DIV-2：逻辑数据模型	数据需求和有组织的业务流程（行动）规则的文档。相当于 DoDAF V1.5 中的OV-7。
DIV-3：物理数据模型	逻辑数据模型实体的物理执行格式，例如，消息格式、文档结构、物理模式。相当于DoDAFV 1.5中的SV-11。

7.标准视点 StdV

标准视点(StdV)是用来管控系统各组成部分或要素的编排、交互和相互依赖的规则的最小集。其目的是确保系统能满足特定的一组操作需求。

标准视点提供技术系统的实施指南，以工程规范为基础，确立通用的积木块，开发产品线。

包括一系列技术标准、执行惯例、标准选项、规则和规范，这些标准在特定体系结构描述中可以组成管控系统和系统/服务要素的文件(profile)。

StdV-1：标准概览	**列出解决方案要遵从的标准、条例条令、规章制度。**
StdV-2：标准预测	列出未来某个时间段内可能发布的标准、条例条令，而这些标准、条例条令对解决方案具有潜在影响。

8.项目视点 PV

项目视点(PV)集中反映了项目是如何有机地组织成一个采办项目的有序组合。
描述多个采办项目之间关联关系，每个采办项目都负责交付特定系统或能力。

PV-1：项目组合关系	**描述组织机构和项目之间的依赖关系以及项目组合管理所要求的组织结构。**
PV-2：项目时间进度	项目的时间进度计划，说明关键里程碑和互相依赖性。
PV-3：项目-能力映射	描述项目与能力的支持关系，说明一个项目提供了哪些能力，某个能力是由哪些项目共同提供了。

TOGAF，Zachman，ITSA 和 DODAF 是非常不错的架构框架，尤其前两者应用很广泛，TOGAF 还有专门的架构认证。当我们掌握了这些框架，我们是不是需要一些架构原则来指导更具体的设计？请看下文。

架构原则

设计原则就是架构设计的指导思想，它指导我们如何将数据和函数组织成类，如何将类链接起来成为组件和程序。反向来说，架构的主要工作就是将软件拆解为组件，设计原则指导我们如何拆解、拆解的粒度、组件间依赖的方向、组件解耦的方式等。

设计原则有很多，我们进行架构设计的主导原则是 OCP（开闭原则），在类和代码的层级上有：SRP（单一职责原则）、LSP（里氏替换原则）、ISP（接口隔离原则）、DIP（依赖反转原则）；在组件的层级上有：REP（复用、发布等同原则）、CCP（共同闭包原则）、CRP（共同复用原则），处理组件依赖问题的三原则：无依赖环原则、稳定依赖原则、稳定抽象原则。

1.OCP（开闭原则）：设计良好的软件应该易于扩展，同时抗拒修改。这是我们进行架构设计的主导原则，其他的原则都为这条原则服务。

2.SRP（单一职责原则）：任何一个软件模块，都应该有且只有一个被修改的原因，“被修改的原因”指系统的用户或所有者，翻译一下就是，任何模块只对一个用户的价值负责，该原则指导我们如何拆分组件。

举个例子，CTO 和 COO 都要统计员工的工时，当前他们要求的统计方式可能是相同的，我们复用一套代码，这时 COO 说周末的工时统计要乘以二，按照这个需求修改完代码，CTO 可能就要过来骂街了。当然这是个非常浅显的例子，实际项目中也有很多代码服务于多个价值主体，这带来很大的探秘成本和修改风险，另外，当一份代码有多个所有者时，就会产生代码合并冲突的问题。

3.LSP（里氏替换原则）：当用同一接口的不同实现互相替换时，系统的行为应该保持不变。该原则指导的是接口与其实现方式。

你一定很疑惑，实现了同一个接口，他们的行为也肯定是一致的呀，还真不一定。假设认为矩形的系统行为是：面积=宽*高，让正方形实现矩形的接口，在调用 setW 和 setH 时，正方形做的其实是同一个事情，设置它的边长。这时下边的单元测试用矩形能通过，用正方形就不行，实现同样的接口，但是系统行为变了，这是违反 LSP 的经典案例。

4.ISP（接口隔离原则）：不依赖任何不需要的方法、类或组件。该原则指导我们的接口设计。当我们依赖一个接口但只用到了其中的部分方法时，其实我们已经依赖了不需要的方法或类，当这些方法或类有变更时，会引起我们类的重新编译，或者引起我们组件的重新部署，这些都是不必要的。所以我们最好定义个小接口，把用到的方法拆出来。

5.DIP（依赖反转原则）：指一种特定的解耦（传统的依赖关系创建在高层次上，而具体的策略设置则应用在低层次的模块上）形式，使得高层次的模块不依赖于低层次的模块的实现细节，依赖关系被颠倒（反转），从而使得低层次模块依赖于高层次模块的需求抽象。

跨越组建边界的依赖方向永远与控制流的方向相反。该原则指导我们设计组件间依赖的方向。

依赖反转原则是个可操作性非常强的原则，当你要修改组件间的依赖方向时，将需要进行组件间通信的类抽象为接口，接口放在边界的哪边，依赖就指向哪边。

6.REP（复用、发布等同原则）：软件复用的最小粒度应等同于其发布的最小粒度。直白地说，就是要复用一段代码就把它抽成组件，该原则指导我们组件拆分的粒度。

7.CCP（共同闭包原则）：为了相同目的而同时修改的类，应该放在同一个组件中。CCP 原则是 SRP 原则在组件层面的描述。该原则指导我们组件拆分的粒度。

对大部分应用程序而言，可维护性的重要性远远大于可复用性，由同一个原因引起的代码修改，最好在同一个组件中，如果分散在多个组件中，那么开发、提交、部署的成本都会上升。

8.CRP（共同复用原则）：不要强迫一个组件依赖它不需要的东西。CRP 原则是 ISP原则在组件层面的描述。该原则指导我们组件拆分的粒度。

相信你一定有这种经历，集成了组件 A，但组件 A 依赖了组件 B、C。即使组件 B、C 你完全用不到，也不得不集成进来。这是因为你只用到了组件 A 的部分能力，组件 A 中额外的能力带来了额外的依赖。如果遵循共同复用原则，你需要把 A 拆分，只保留你要用的部分。

REP、CCP、CRP 三个原则之间存在彼此竞争的关系，REP 和 CCP 是黏合性原则，它们会让组件变得更大，而 CRP 原则是排除性原则，它会让组件变小。遵守REP、CCP 而忽略 CRP，就会依赖了太多没有用到的组件和类，而这些组件或类的变动会导致你自己的组件进行太多不必要的发布；遵守 REP、CRP 而忽略 CCP，因为组件拆分的太细了，一个需求变更可能要改 n 个组件，带来的成本也是巨大的。

除了上述设计原则，还有一些重要的指导原则如下：

1.N+1设计：系统中的每个组件都应做到没有单点故障；

2.回滚设计：确保系统可以向前兼容，在系统升级时应能有办法回滚版本；

3.禁用设计：应该提供控制具体功能是否可用的配置，在系统出现故障时能够快速下线功能；

4.监控设计：在设计阶段就要考虑监控的手段，便于有效的排查问题，比如引入traceId、业务身份 Id 便于排查监控问题；

5.多活数据中心设计：若系统需要极高的高可用，应考虑在多地实施数据中心进行多活，至少在一个机房断电的情况下系统依然可用；

6.采用成熟的技术：刚开发的或开源的技术往往存在很多隐藏的 bug，出了问题没有很好的商业支持可能会是一个灾难；

7.资源隔离设计：应避免单一业务占用全部资源；

8.架构水平扩展设计：系统只有做到能水平扩展，才能有效避免瓶颈问题；

9.非核心则购买的原则：非核心功能若需要占用大量的研发资源才能解决，则考虑购买成熟的产品；

10.使用商用硬件：商用硬件能有效降低硬件故障的机率；

11.快速迭代：系统应该快速开发小功能模块，尽快上线进行验证，早日发现问题大大降低系统交付的风险；

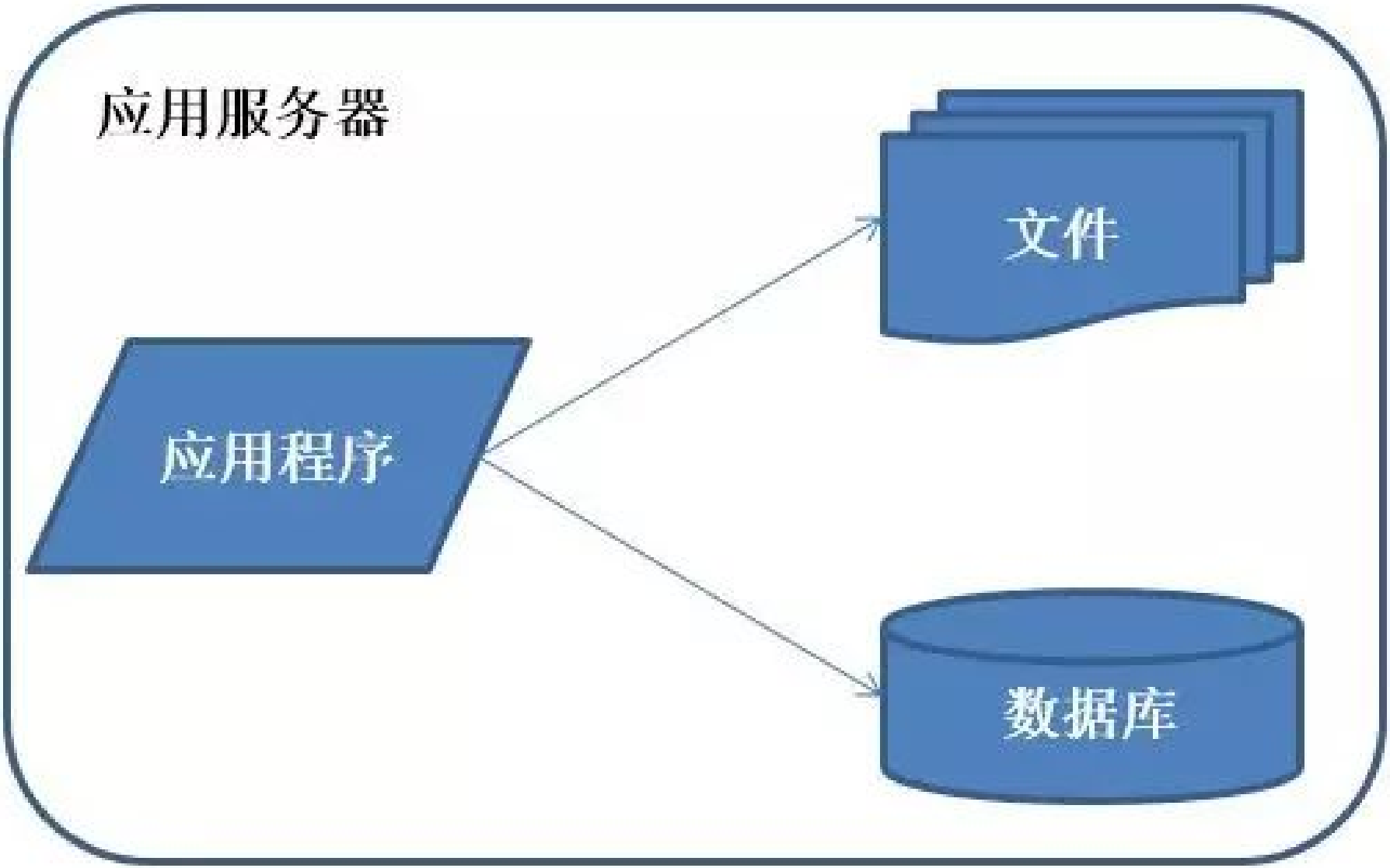
12.无状态设计：服务接口应该做成无状态的，当前接口的访问不依赖于接口上次访问的状态。

架构师知道了职责，具备很好的架构思维，掌握了通用的架构框架和方法论，使用架构原则进行架构设计，不同的业务和系统要求不一样，那么有没有针对不同场景的系统架构设计？下文就针对分布式架构演进、单元化架构、面向服务 SOA 架构、微服务架构、Serverless 架构进行介绍，以便于我们在实际运用中进行参考使用。

常见架构

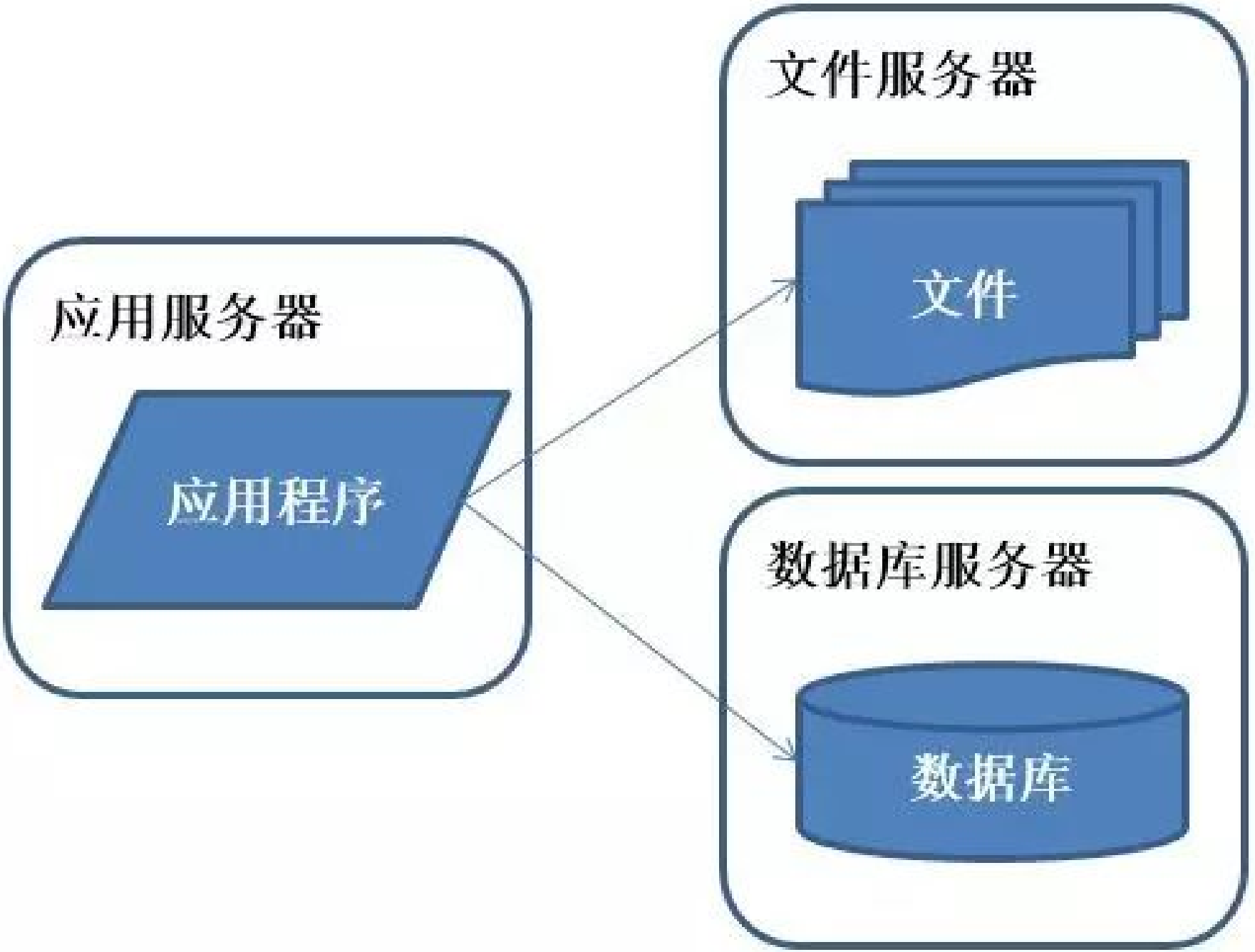
分布式架构演进

初始阶段架构


应用服务器
文件
应用程序
数据库

特征：应用程序，数据库，文件等所有资源都放在一台服务器上。

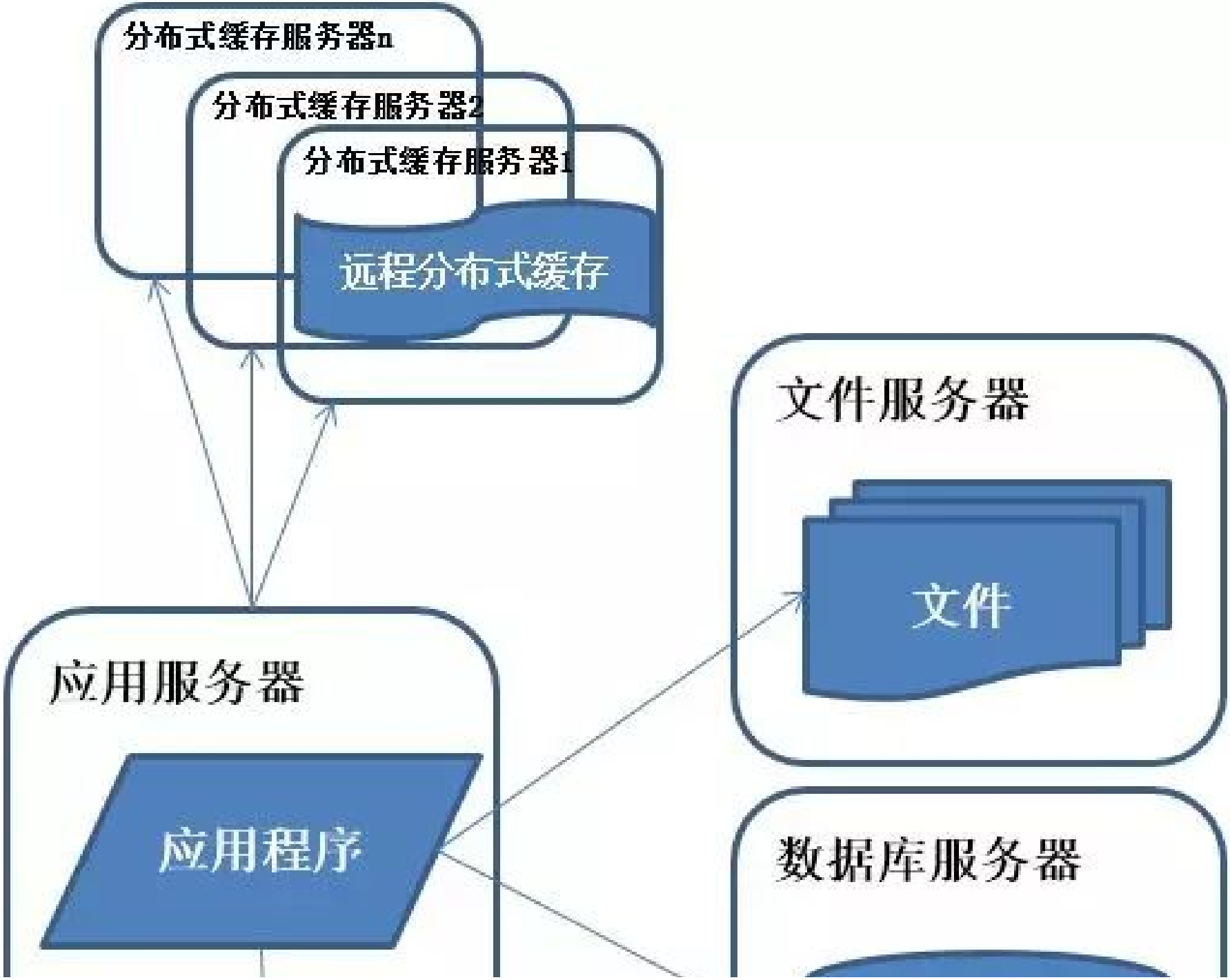
应用服务和数据服务以及文件服务分离


文件服务器
文件
应用服务器
应用程序
数据库服务器
数据库

说明：好景不长，发现随着系统访问量的再度增加，webserver 机器的压力在高峰期会上升到比较高，这个时候开始考虑增加一台 webserver。

特征：应用程序、数据库、文件分别部署在独立的资源上。

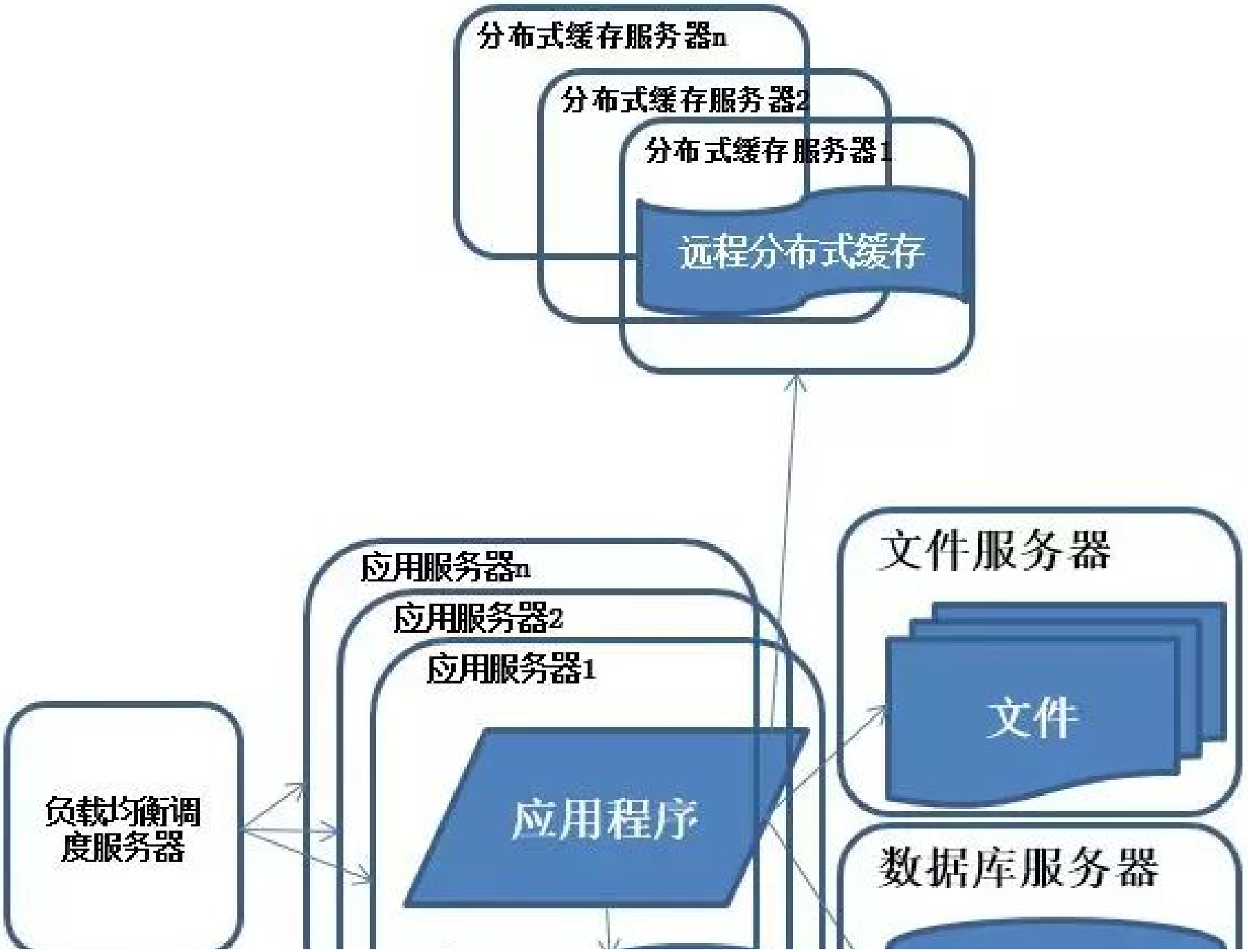
使用缓存改善性能


分布式缓存服务器n
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
文件服务器
文件
应用服务器
应用程序
数据库服务器

说明：系统访问特点遵循二八定律，即80%的业务访问集中在20%的数据上。缓存分为本地缓存 远程分布式缓存，本地缓存访问速度更快但缓存数据量有限，同时存在与应用程序争用内存的情况。

特征：数据库中访问较集中的一小部分数据存储在缓存服务器中，减少数据库的访问次数，降低数据库的访问压力。

使用“应用服务器”集群

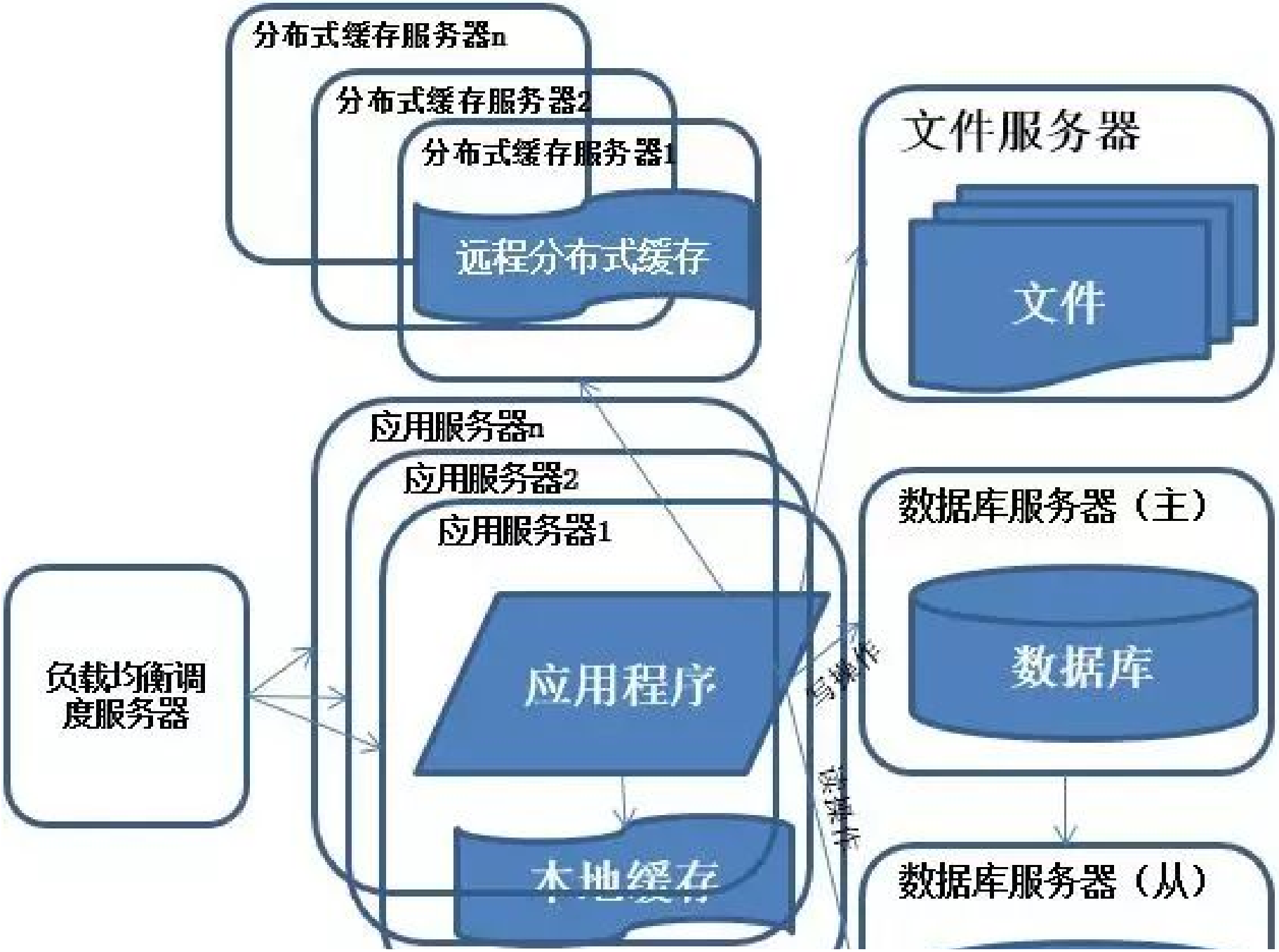

分布式缓存服务器n
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
应用服务器n
应用服务器2
应用服务器1
应用程序
负载均衡调度服务器
文件服务器
文件
数据库服务器

说明：在做完分库分表这些工作后，数据库上的压力已经降到比较低了，又开始过着每天看着访问量暴增的幸福生活了。突然有一天，发现系统的访问又开始有变慢的趋势了，这个时候首先查看数据库，压力一切正常，之后查看 webserver，发现 apache 阻塞了很多的请求， 而应用服务器对每个请求也是比较快的，看来是请求数太高导致需要排队等待，响应速度变慢。

特征：多台服务器通过负载均衡同时向外部提供服务，解决单台服务器处理能力和存储空间上限的问题。

描述：使用集群是系统解决高并发、海量数据问题的常用手段。通过向集群中追加资源，提升系统的并发处理能力，使得服务器的负载压力不再成为整个系统的瓶颈。

数据库读写分离

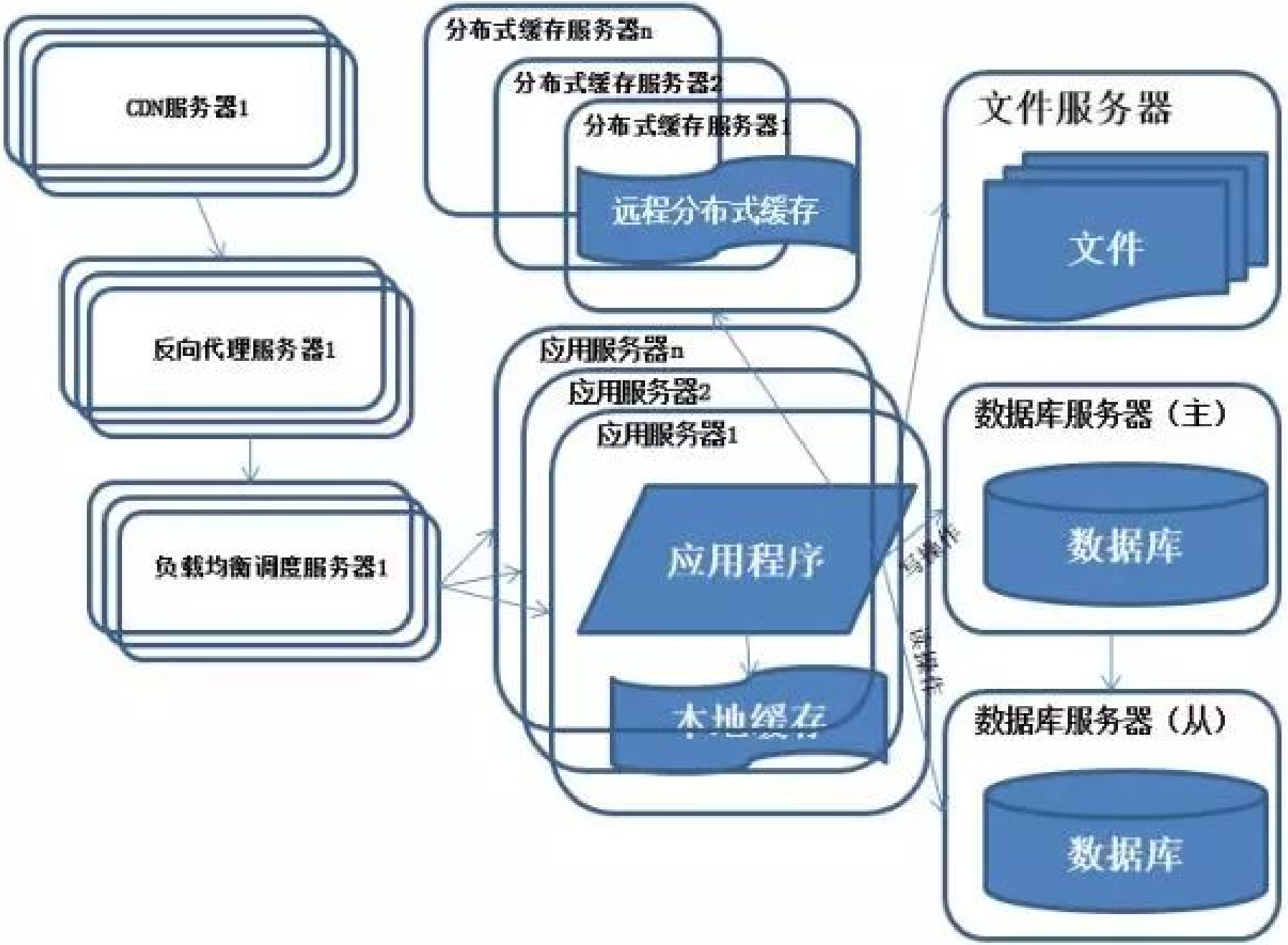

分布式缓存服务器n
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
文件服务器
文件
应用服务器n
应用服务器2
应用服务器1
负载均衡调度服务器
应用程序
本地缓存
写操作
读操作
数据库服务器（主）
数据库
数据库服务器（从）

说明：享受了一段时间的系统访问量高速增长的幸福后，发现系统又开始变慢了，这次又是什么状况呢，经过查找，发现数据库写入、更新的这些操作的部分数据库连接的资源竞争非常激烈，导致了系统变慢。

特征：数据库引入主备部署。

描述：把数据库划分为读库和写库，通过引入主从数据库服务，读和写操作在不同的数据库服务处理，读库可以有多个，通过同步机制把写库的数据同步到读库，对于需要查询最新写入数据场景，可以通过在缓存中多写一份，通过缓存获得最新数据。

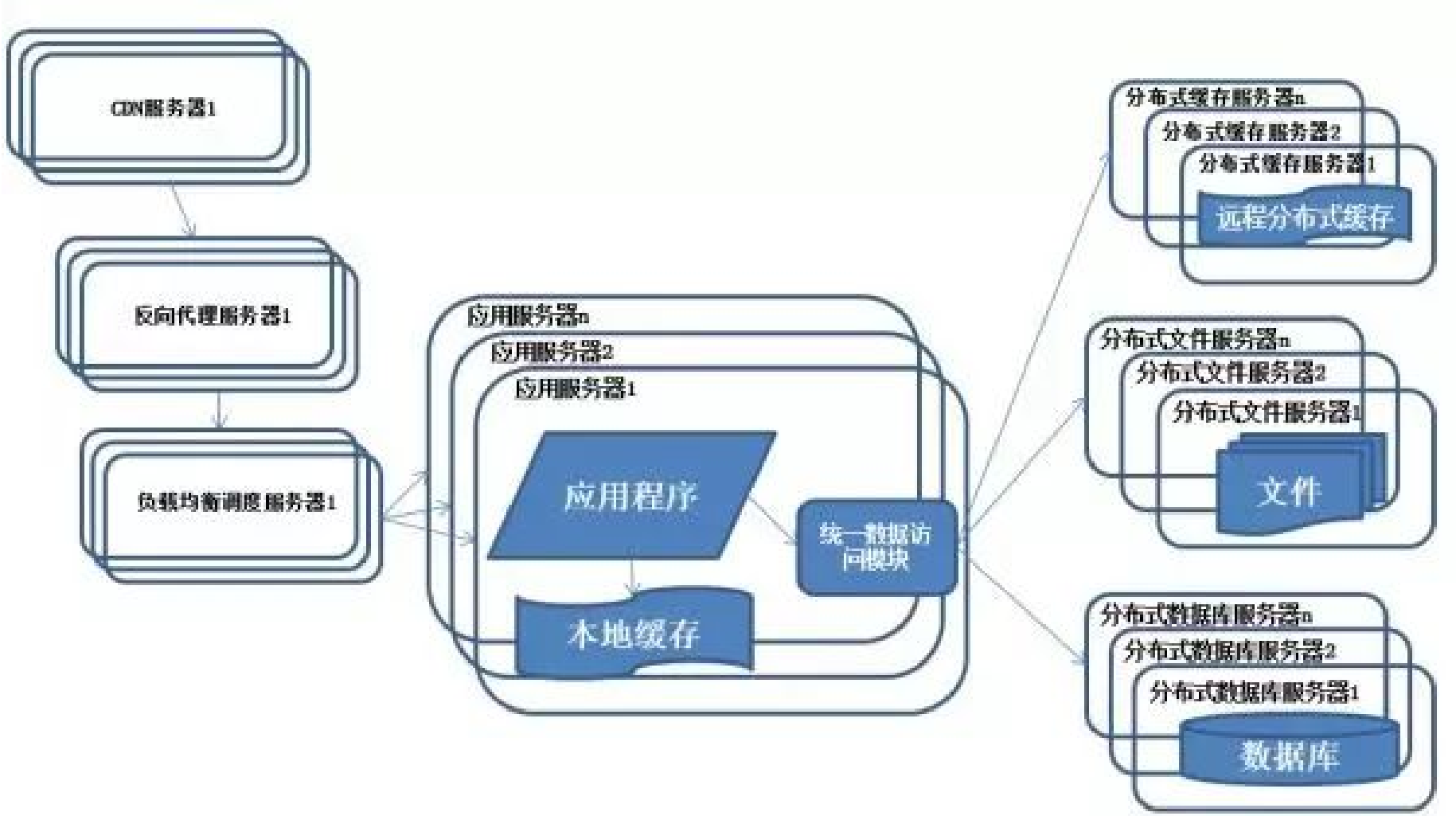
反向代理和CDN加速


CDN服务器1
反向代理服务器1
负载均衡调度服务器1
分布式缓存服务器n
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
应用服务器n
应用服务器2
应用服务器1
应用程序
本地缓存
文件服务器
文件
数据库服务器（主）
数据库
数据库服务器（从）
数据库

特征：采用CDN和反向代理加快系统的访问速度。

描述：为了应付复杂的网络环境和不同地区用户的访问，通过 CDN 和反向代理加快用户访问的速度，同时减轻后端服务器的负载压力。CDN 与反向代理的基本原理都是缓存。

“分布式文件”系统 和 “分布式数据库”



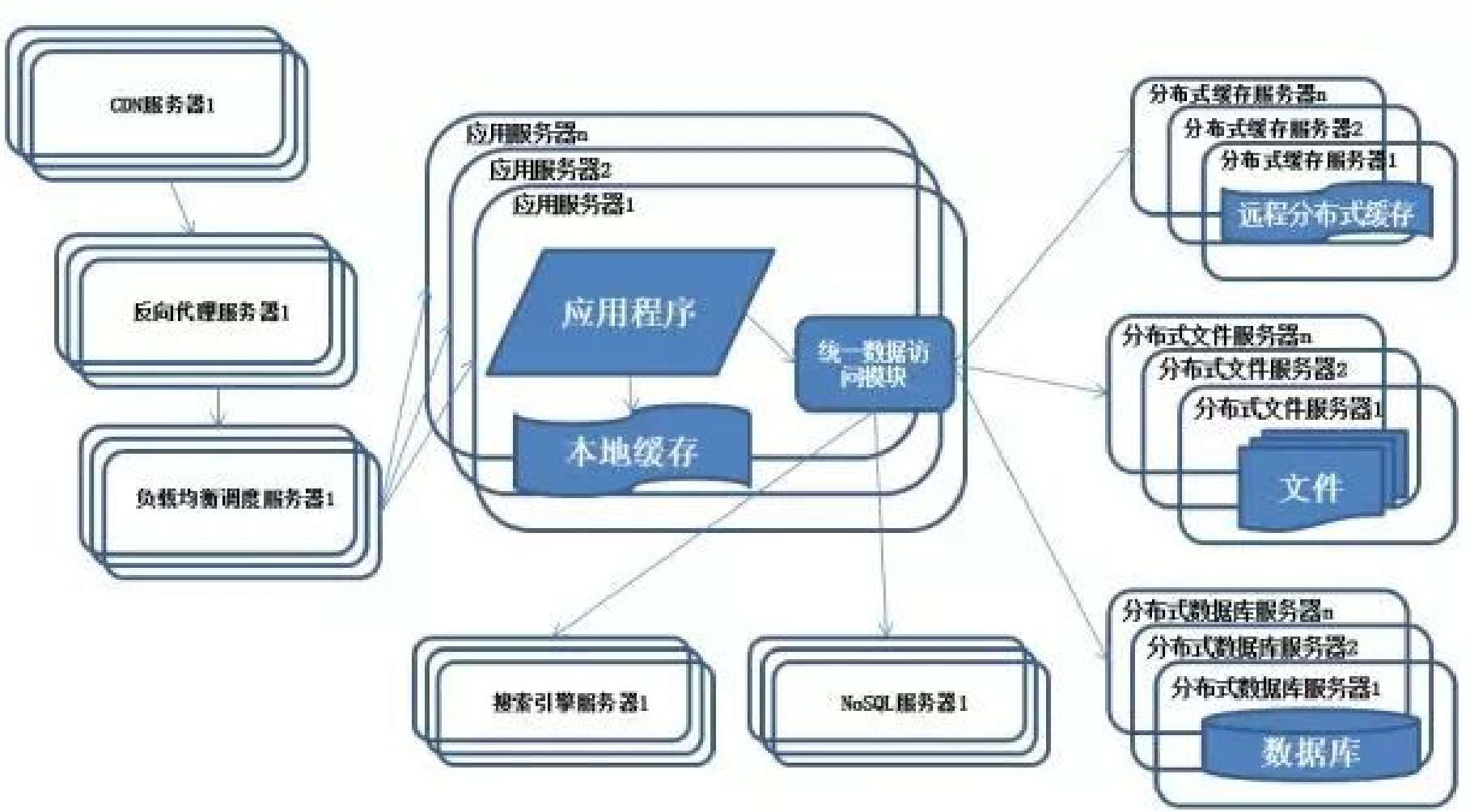
说明：随着系统的不断运行，数据量开始大幅度增长，这个时候发现分库后查询仍然会有些慢，于是按照分库的思想开始做分表的工作

特征：数据库采用分布式数据库，文件系统采用分布式文件系统。

描述：任何强大的单一服务器都满足不了大型系统持续增长的业务需求，数据库读写分离随着业务的发展最终也将无法满足需求，需要使用分布式数据库及分布式文件系统来支撑。

分布式数据库是系统数据库拆分的最后方法，只有在单表数据规模非常庞大的时候才使用，更常用的数据库拆分手段是业务分库，将不同的业务数据库部署在不同的物理服务器上。

使用 NoSQL 和搜索引擎

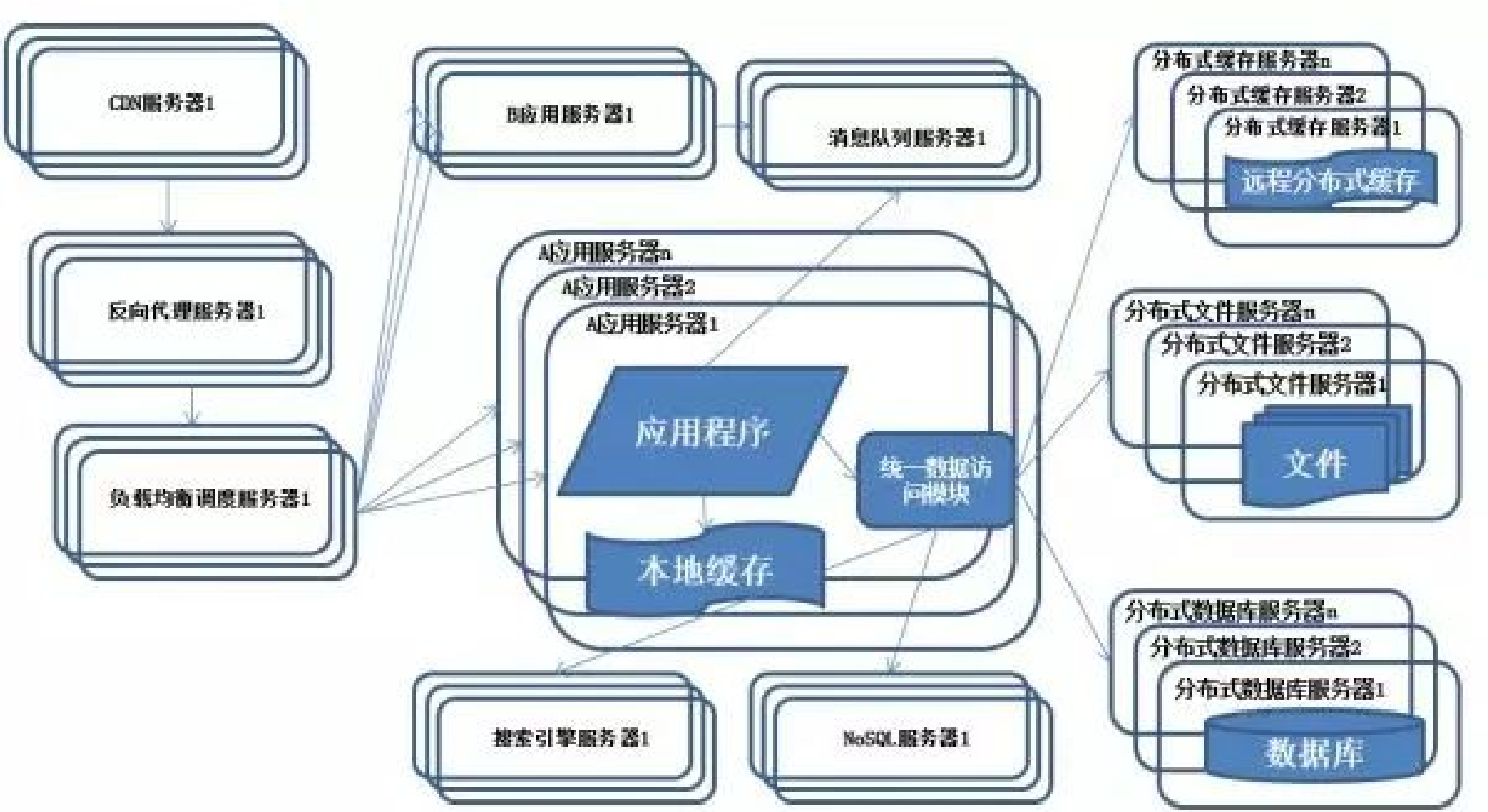


特征：系统引入 NoSQL 数据库及搜索引擎。

描述：随着业务越来越复杂，对数据存储和检索的需求也越来越复杂，系统需要采用一些非关系型数据库如 NoSQL 和分数据库查询技术如搜索引擎。

应用服务器通过统一数据访问模块访问各种数据，减轻应用程序管理诸多数据源的麻烦。

业务拆分



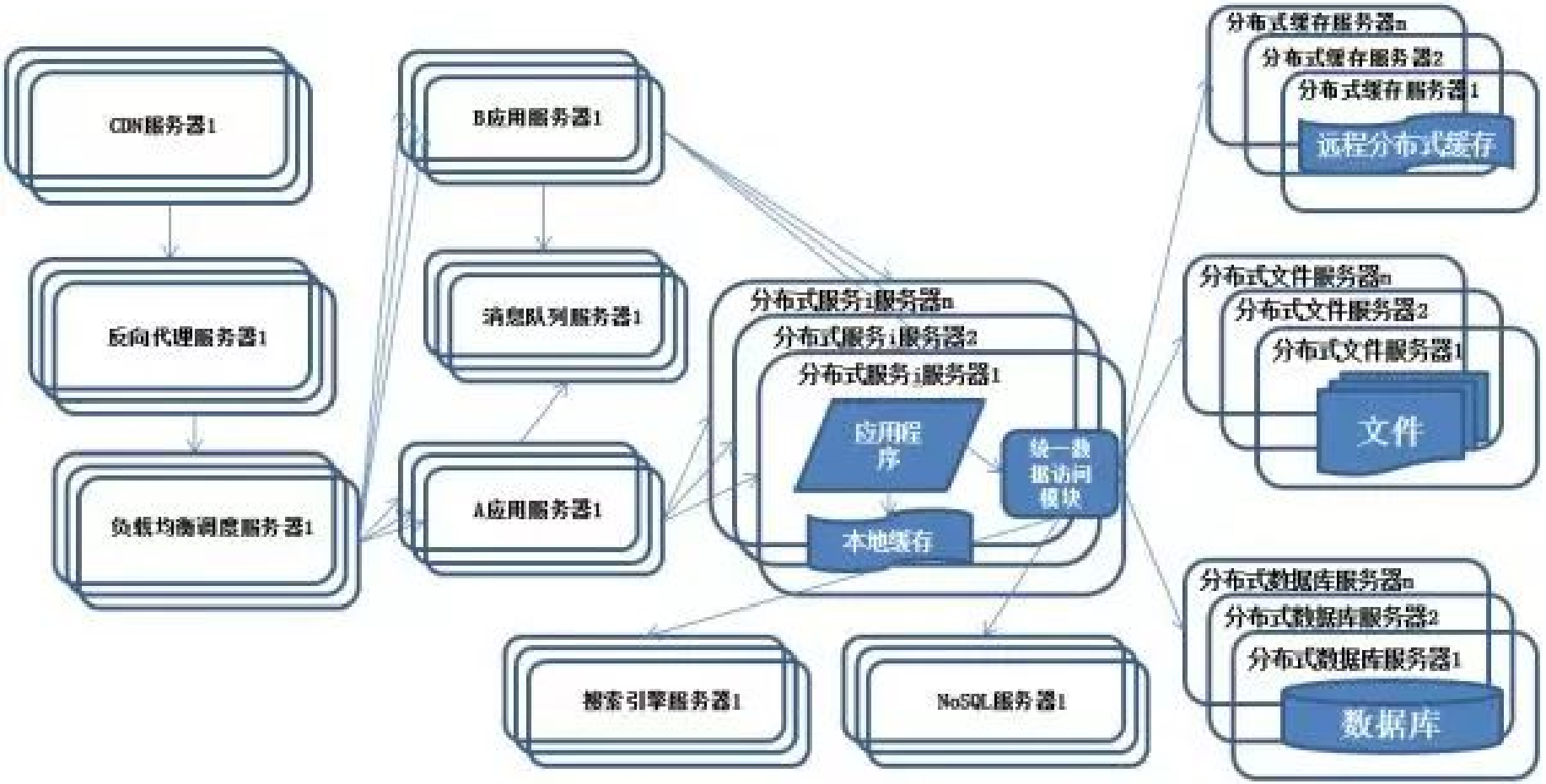
特征：系统上按照业务进行拆分改造，应用服务器按照业务区分进行分别部署。

描述：为了应对日益复杂的业务场景，通常使用分而治之的手段将整个系统业务分成不同的产品线，应用之间通过超链接建立关系，也可以通过消息队列进行数据分发，当然更多的还是通过访问同一个数据存储系统来构成一个关联的完整系统。

纵向拆分：将一个大应用拆分为多个小应用，如果新业务较为独立，那么就直接将其设计部署为一个独立的 Web 应用系统纵向拆分相对较为简单，通过梳理业务，将较少相关的业务剥离即可。

横向拆分：将复用的业务拆分出来，独立部署为分布式服务，新增业务只需要调用这些分布式服务横向拆分需要识别可复用的业务，设计服务接口，规范服务依赖关系。

分布式服务



特征：公共的应用模块被提取出来，部署在分布式服务器上供应用服务器调用。

描述：随着业务越拆越小，应用系统整体复杂程度呈指数级上升，由于所有应用要和所有数据库系统连接，最终导致数据库连接资源不足，拒绝服务。