

NEW PLAN

软考高级架构师

• — 段 新 征 程 •





大纲介绍





信息系统架构基本概念

信息系统架构(ISA)是指对**某一特定内容里的信息进行统筹、规划、设计、安排等一系列有机处理的活动**。为了更好地理解信息系统架构的定义，特作如下说明：

1. **架构是对系统的抽象**，它通过**描述元素、元素的外部可见属性及元素**之间的关系来反映这种抽象。因此，仅与内部具体实现有关的细节是不属于架构的，即定义强调元素的“外部可见”属性。
2. 架构由**多个结构**组成，结构是从功能角度来描述元素之间的关系的，具体的结构传达了架构某方面的信息，但是个别结构一般不能代表大型信息系统架构。
3. 任何软件都存在架构，但不一定有对该架构的具体表述文档。即架构可以独立于架构的描述而存在。如文档已过时，则该文档不能反映架构。
4. **元素及其行为的集合构成架构的内容**本现系统由哪些元素组成，这些元素各有哪些功能(外部可见)，以及这些元素间如何连接与互动。即在两个方面进行抽象：在静态方面，关注系统的大粒度(宏观)总体结构(如分层)；在动态方面，关注系统内关键行为的共同特征。
5. 架构**具有“基础”性**：它通常涉及**解决各类关键重复问题的通用方案**(复用性)，以及系统设计中影响深远(架构敏感)的各项重要决策(一旦贯彻，更改的代价昂贵)。
6. 架构**隐含有“决策”**，即架构是由架构设计师根据关键的功能和非功能性需求(质量属性及项目相关的约束)进行设计与决策的结果。



信息系统架构

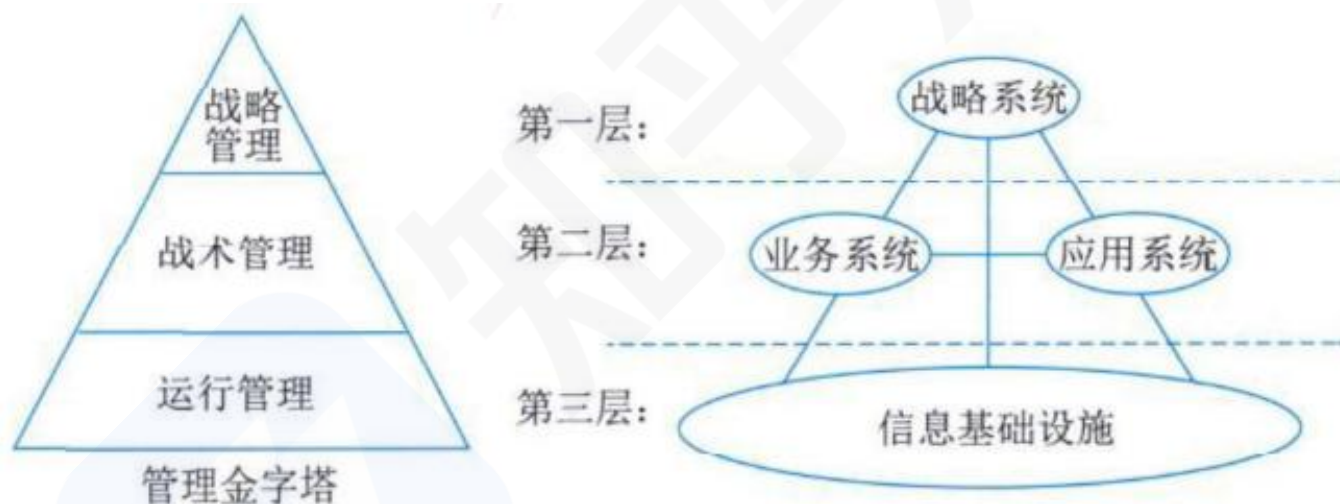
- 信息系统架构可分为**物理结构与逻辑结构**两种，物理结构是指不考虑系统各部分的实际工作与功能结构，只抽象地考察其硬件系统的空间分布情况。
逻辑结构是指信息系统各种功能子系统的综合体。
 - 物理结构一般分为**集中式与分布式两大类**。
- 在信息系统开发中，强调对各种子系统进行统一规划，并对各子系统进行综合，从逻辑结构来看分为以下几种综合方式。
 - 横向综合：**将同一管理层次的各种职能综合在一起**，例如，将运行控制层的人事和工资子系统综合在一起，使基层业务处理一体化。
 - 纵向综合：**把某种职能的各个管理层次的业务组织在一起**，这种综合沟通了上下级之间的联系，如工厂的会计系统和公司的会计系统综合在一起，它们都有共同之处，能形成一体化的处理过程。
 - 纵横综合：主要是从信息模型和处理模型两个方面来进行综合，做到信息集中共享，程序尽量模块化，注意提取通用部分，**建立系统公用数据库和统一的信息处理系统**。
- 信息系统常用四种架构模型
 - **单机应用模式**：是最简单的软件结构，是指运行在一台物理机器上的独立应用程序。单机系统本身也可以很复杂。
 - **客户机/服务器模式**：即两层、三层C/S、B/S模式、MVC模式等。
 - **面向服务架构(SOA)模式**。
 - **企业数据交换总线**：不同的企业应用之间进行信息交换的公共通道



信息系统架构-企业信息系统的总体框架

要在企业中建立一个有效集成的ISA，必须考虑企业中的四个方面：

- 战略系统
- 业务系统
- 应用系统
- 信息基础设施





信息系统架构

1.战略系统：是指企业中与战略制定、高层决策有关的管理活动和计算机辅助系统。

- 在ISA中战略系统由两个部分组成，其一是为以计算机为基础的高层决策支持系统，比如DSS(决策支持系统)，其二是企业的战略规划体系。
- 在ISA中设立战略系统有两重含义：一是它表示**信息系统对企业高层管理者的决策支持能力**；二是它表示企业战略规划**对信息系统建设的影响和要求**。

2.业务系统：是指企业中完成一定业务功能的各部分(物质、能量、信息和人)组成的系统。

作用：对企业现有业务系统、业务过程和业务活动进行建模，并在企业战略的指导下，采用业务流程管理(BPM)和业务流程重组(BPR)。

3.应用系统：即**应用软件系统**，指信息系统中的应用软件部分。如TPS(业务处理系统)、MIS(管理信息系统)等。包含两个基本组成部分：**内部功能实现部分**和**外部界面部分**。

4.**企业信息基础设施(EII)**：是指根据企业当前业务和可预见的发展趋势，及对信息采集、处理、存储和流通的要求，构筑由信息设备、通信网络、数据库、系统软件和支持性软件等组成的环境。这里可以将企业信息基础设施分成三部分：**技术基础设施、信息资源设施和管理基础设施**。

- 技术基础设施：由**计算机、网络、系统软件、支持性软件、数据交换协议**等组成；
- 信息资源设施：由**数据与信息本身、数据交换的形式与标准、信息处理方法**等组成；
- 管理基础设施：指企业中**信息系统部门的组织结构、信息资源设施管理人员的分工、企业信息基础设施的管理方法与规章制度**等。



信息系统架构

- 企业背景：假设我们有一家零售公司，名为ABC，该公司经营多家实体店和在线销售渠道。ABC希望通过建立一个有效的ISA来实现以下目标：
 - 提高客户体验，包括在线购物和实体店购物。
 - 优化库存管理，以减少库存成本。
 - 加强供应链协调，以确保产品及时到达。
 - 实现销售数据分析，以更好地了解客户需求。
- 战略系统：首先，ABC需要定义战略系统。这涉及到制定信息技术战略，以确保其业务目标与信息技术目标一致。例如，他们决定将数字化客户体验和供应链协调作为优先战略，因为这与他们的业务目标密切相关。
- 业务系统：接下来，ABC需要确定其关键业务系统，这些系统将支持其战略目标。这可能包括销售点系统、在线销售平台、库存管理系统和供应链管理系统。这些业务系统应该能够高效地共享信息，以支持整个企业的决策和操作。
- 应用系统：在业务系统之上，ABC需要选择和实施适当的应用系统。例如，他们可能需要一个统一的数据仓库，用于存储销售、库存和供应链数据，以便进行分析。他们还可能需要实施电子商务平台，以提供在线购物体验。这些应用系统应该与业务系统集成，以实现数据流畅和协调。
- 信息基础设施：最后，ABC需要建立稳健的信息基础设施来支持其应用系统。这包括网络基础设施、数据库管理系统、云服务等。他们还需要考虑数据安全和隐私问题，确保客户数据受到保护。
- 通过综合考虑这四个方面的，ABC可以建立一个有效集成的ISA，有助于实现他们的战略目标，提高客户体验，优化库存管理，加强供应链协调，并进行销售数据分析。ISA的设计和实施将使企业能够更好地适应市场变化，提高竞争力。



信息系统架构设计方法-TOGAF

- TOGAF(The Open Group Architecture Framework,TOGAF)是一种**开放式企业架构框架标准**，它为标准、方法论和企业架构专业人员之间的沟通提供一致性保障
- 该框架旨在通过以下**四个目标**帮助企业组织和解决所有关键业务需求：**确保从关键利益相关方到团队成员的所有用户都使用相同的语言、避免被“锁定”到企业架构的专有解决方案、节省时间和金钱更有效地利用资源、实现可观的投资回报。**
- TOGAF框架核心思想：模块化架构、内容框架、扩展指南、架构风格。
- TOGAF的关键是**架构开发方法(Architecture Development Method,ADM)**，为开发企业架构所需要执行各个步骤(10个阶段)以及它们之间的关系进行详细的定义。
 - ADM方法是由一组**按照架构领域的架构开发顺序而排列成一个环的多个阶段**所构成。
 - ADM三个级别的迭代：基于ADM整体的迭代、多个开发阶段间的迭代、在一个阶段内部的迭代。将ADM全生命周期十个阶段主要活动：

A D M 阶段	A D M 阶段内的活动
准备阶段	为实施成功的企业架构项目做好准备，包括定义组织机构、特定的架构框架、架构原则和工具
需求管理	完成需求的识别、保管和交付，相关联的 A D M 阶段则按优先级顺序对需求进行处理 TOGAF 项目的每个阶段，都是建立在业务需求之上并且需要对需求进行确认



信息系统架构设计方法-ADM

阶段 A：架构愿景	设置 TOGAF 项目的范围、约束和期望。创建架构愿景，包括： ● 定义利益相关者； ● 确认业务上下文环境； ● 创建架构工作说明书； ● 取得上级批准
阶段 B：业务架构 阶段 C：信息系统架构（应用&数据） 阶段 D：技术架构	从业务、信息系统和技术三个层面进行架构开发，在每一个层面分别完成以下活动： ● 开发基线架构描述； ● 开发目标架构描述； ● 执行差距分析
阶段 E：机会和解决方案	进行初步实施规划，并确认在前面阶段中确定的各种构建块的交付物形式； ● 确定主要实施项目； ● 对项目分组并纳入过渡架构； ● 决定途径（制造/购买/重用、外包、商用、开源）； ● 评估优先顺序； ● 识别相依性
阶段 F：迁移规划	对阶段 E 确定的项目进行绩效分析和风险评估，制订一个详细的实施和迁移计划
阶段 G：实施治理	定义实施项目的架构限制； ● 提供实施项目的架构监督； ● 发布实施项目的架构合同； ● 监测实施项目以确保符合架构要求
阶段 H：架构变更管理	提供持续监测和变更管理的流程，以确保架构可以响应企业的需求并且将架构对于业务的价值最大化

信息系统架构设计方法-信息化总体架构方法

- 实现信息化就要构筑和完善6个要素(开发利用信息资源，建设国家信息网络，推进信息技术应用，发展信息技术和产业，培育信息化人才，制定和完善信息化政策)的国家信息化体系。
- 完整的信息化内涵包括四方面内容：信息网络体系、信息产业基础、社会运行环境、效用积累过程。信息化建设指品牌利用现代信息技术来支撑品牌管理的手段和过程。信息化建设包括了企业规模，企业在电话通信、网站、电子商务方面的投入情况，在客户资源管理、质量管理体系方面的建设成就等。
- 信息化主要体现以下6种特征：易用性、健壮性、扩展性、安全性、整合性、移动性。
- 信息化架构一般有两种模式，一种是数据导向架构，一种是流程导向架构。对于数据导向架构重点是在数据中心，BI商业智能等建设中使用较多，关注数据模型和数据质量;对于流程导向架构，SOA本身就是关键方法和技术，关注端到端流程整合，以及架构对流程变化的适应度。两种架构并没有严格的边界，而是相互配合和补充。
 - 数据导向架构研究的是数据对象和数据对象之间的关系，这个是首要的内容。在这个完成后仍然要开始考虑数据的产生、变更、废弃等数据生命周期，这些自然涉及的数据管理的相关流程
 - 流程导向架构关注的是流程，架构本身的目的是为了端到端流程整合服务。因此研究切入点会是价值链分析，流程分析和分解，业务组件划分。



信息系统架构设计方法

信息系统的生命周期可以分为下面五个阶段：

- 系统规划阶段(做不做)：任务是**对组织的环境、目标及现行系统的状况进行初步调查**，根据组织目标和发展战略确定信息系统的发展战略，对建设新系统的需求做出分析和预测，制时考虑建设新系统所受的各种约束，**研究建设新系统的必要性和可能性。根据需要与可能，给出制建系统的备选方案。**输出：**可行性研究报告、系统设计任务书。**
- 系统分析阶段(做什么)：任务是根据系统设计任务书所确定的范围，对现行系统进行详细调查，描述现行系统的业务流程，指出现行系统的局限性和不足之处，**确定新系统的基本目标和逻辑功能要求，即提出新系统的逻辑模型。**系统分析阶段又称为逻辑设计阶段。这个阶段是整个系统建设的关键阶段，也是信息系统建设与一般工程项目的重要区别所在。输出：系统说明书。
- 系统设计阶段(怎么做)：系统分析阶段的任务是回答系统“做什么”的问题，而系统设计阶段要回答的问题是“怎么做”。**该阶段的任务是根据系统说明书中规定的功能要求，具体设计实现逻辑模型的技术方案，也就是设计新系统的物理模型。**这个阶段又称为物理设计阶段，可分为总体设计(概要设计)和详细设计两个子阶段。输出：系统设计说明书(概要设计、详细设计说明书)。
- 系统实施阶段(开始做)：是将设计的系统付诸实施的阶段。这一阶段的任务包括**计算机等设备的购置、安装和调试、程序的编写和调试、人员培训、数据文件转换、系统调试与转换等。**这个阶段的特点是几个互相联系、互相制约的任务同时展开，必须精心安排、合理组织。系统实施是按实施计划分阶段完成的，每个阶段应写出实施进展报告。系统测试之后写出系统测试分析报告。输出：**实施进展报告、系统测试分析报告。**
- 系统运行和维护阶段(做完后要干什么)：系统投入运行后，需要经常进行维护和评价，记录系统运行的情况，根据一定的规则对系统进行必要的修改，评价系统的工作质量和经济效益。

信息系统架构案例分析-价值驱动的体系结构

■ 价值模型核心的特征可以简化为三种基本形式：

- **价值期望值**：表示对某一特定功能的需求，包括内容(功能)、满意度(质量)和不同级别质量的实用性。
 - **反作用力**：系统部署实际环境中，实现某种价值期望值的难度，通常期望越高难度越大，即反作用力。
 - **变革催化剂**：表示环境中导致价值期望值发生变化的某种事件，或者是导致不同结果的限制因素。
- 反作用力和变革催化剂称为限制因素**，把这三个统称为价值驱动因素。

■ 体系结构挑战是**因为一个或多个限制因素使得满足一个或多个期望值变得更困难。**

■ 制定系统的体系结构策略始于：

- 识别合适的价值背景并对其进行优先化。
- 在每一背景中定义效用曲线和优先化期望值。
- 识别和分析每一背景中的反作用力和变革催化剂。
- 检测限制因素使满足期望值变难的领域。

■ 优化体系结构需要权衡：重要性、程度、后果、隔离。



信息系统架构案例分析-Web服务在HL7上的应用

- HL7(Health Level Seven): 是美国国家标准化协会 (ANSI) 认可的标准化开发组织中的一个，是一种国际标准，用于医疗信息交换和数据集成。它是一种用于在不同医疗信息系统之间共享健康信息的协议和数据标准。HL7标准的目标是**确保不同医疗设施和系统之间可以有效、安全地共享患者的医疗信息，从而提高医疗保健服务的质量和效率。**
- 具体的应用和举例如下：
 - 患者数据交换： HL7标准可用于在医院、诊所、实验室和其他医疗机构之间共享患者的基本信息，如患者身份、病历、诊断、处方等。这有助于医疗专业人员快速获取患者的历史数据，提高诊断和治疗的准确性。
 - 医疗设备集成： HL7还可用于将医疗设备（如心电图仪、监护设备、实验室仪器等）连接到医疗信息系统，使实时数据可以自动传输到患者的电子病历中。这可以提高患者监测的效率，并减少数据输入错误。
 - 医疗账单和索赔处理： 医疗保健提供商和保险公司可以使用HL7标准来共享患者的账单和索赔信息。这有助于减少错误，加速报销流程，降低了医疗账单处理的成本。
 - 公共卫生监测： 在流行病爆发或疫情监测中，HL7标准可以用于收集和共享患者的健康数据，以便及时采取措施来阻止疾病的传播。
 - 移动医疗应用： 许多移动医疗应用程序使用HL7标准来与医疗信息系统集成，以提供患者访问其健康记录、预约医生和接收医疗建议的功能。
- 总之，HL7是一种关键的标准，用于在医疗保健领域促进信息共享和集成，有助于提高患者照顾的质量、安全性和效率。这个标准对于医疗信息技术和电子健康记录系统的发展至关重要。



信息系统架构案例分析-以服务为中心的企业整合

- 以服务为中心的企业整合（Service-Centric Enterprise Integration）是一种商业战略和IT战略的方法，其重点是将企业内部和外部的服务资源整合在一起，以实现更高的效率、协同工作和客户满意度。这种整合方法旨在提供更灵活、响应更快、更客户导向的业务模式。
- 举例：一个大型电信公司，它采用了以服务为中心的企业整合战略：
- 服务划分：该电信公司将其业务划分为一系列可重用的服务，包括订阅计划管理、账单生成、客户支持、网络管理等。
- 服务目录：公司创建了一个服务目录，其中包含所有可用服务的详细信息和接口说明。这个目录对内部员工和合作伙伴都可见。
- 客户体验优化：通过整合各种服务，该公司能够提供更快速、更个性化的客户服务。例如，当客户订阅新服务或需要技术支持时，系统可以自动调用相关服务，减少了处理时间，提高了客户满意度。
- 合作伙伴集成：该电信公司与其他供应商和合作伙伴合作，通过共享服务来扩展其业务。这种整合允许不同的系统和组织协同工作，提供更多价值。
- 以服务为中心的企业整合有助于提高业务的敏捷性、效率和创新的能力。它可以应用于各种行业，从电信和金融到制造和医疗保健，以实现更灵活、客户导向的业务模式。



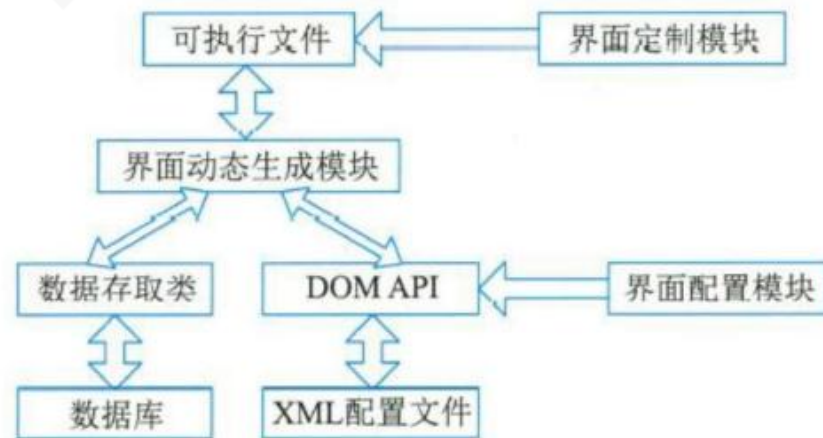
层次式架构设计





表现层框架设计

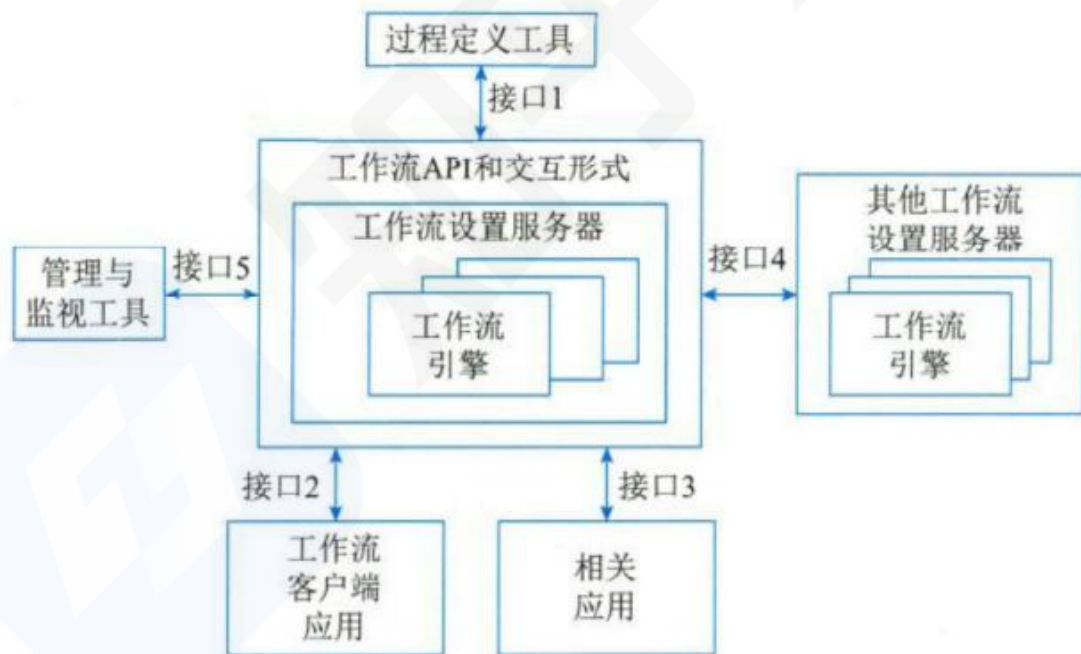
- 软件层次式体系结构是**最通用**的架构，也被叫作**N层架构模式**。大部分的应用会分成**表现层(或称为展示层)**、**中间层(或称为业务层)**、**数据访问层(或称为持久层)**和**数据层**。
- 一般会使用XML来设计表现层，现在更多的是使用动态拖拽生成表现层。
- UIP提供了一个**扩展的框架**，用于**简化用户界面与商业逻辑代码的分离的方法**，可以用它来写复杂的用户界面导航和工作流处理，并且它能够复用在不同的场景、并可以随着应用的增加而进行扩展。
- 使用UIP框架的应用程序把表现层分为了以下几层。
 - User Interface Components（用户界面组件）：**这个组件就是原来的表现层**，用户看到的和进行交互都是这个组件，它负责获取用户的数据并且返回结果。
 - User Interface Process Components（用户界面过程组件）：这个组件用于协调用户界面的各部分，使其配合后台的活动，例如导航和工作流控制，以及状态和视图的管理。**用户看不到这一组件，但是这些组件为用户界面组件提供了重要的支持功能。**
- 表现层动态生成设计：基于XML的界面管理技术可实现灵活的**界面配置(静态)**、**界面动态生成**和**界面定制(动态)**。其思路是用XML生成配置文件及界面所需的元数据，按不同需求生成界面元素及软件界面。





中层框架设计

- 组件设计：业务逻辑组件分为**接口和实现**类两个部分。接口用于定义业务逻辑组件，定义业务逻辑组件必须实现的方法是整个系统运行的核心。增加业务逻辑组件的接口，是为了提供更好的解耦，控制器无须与具体的业务逻辑组件耦合，而是面向接口编程。
- workflow设计：业务流程的全部或部分自动化，在此过程中，文档、信息或任务按照一定的过程规则流转，实现组织成员间的协调工作以达到业务的整体目标。它解决的主要问题是：使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者是促使此目标的实现。





中层框架设计

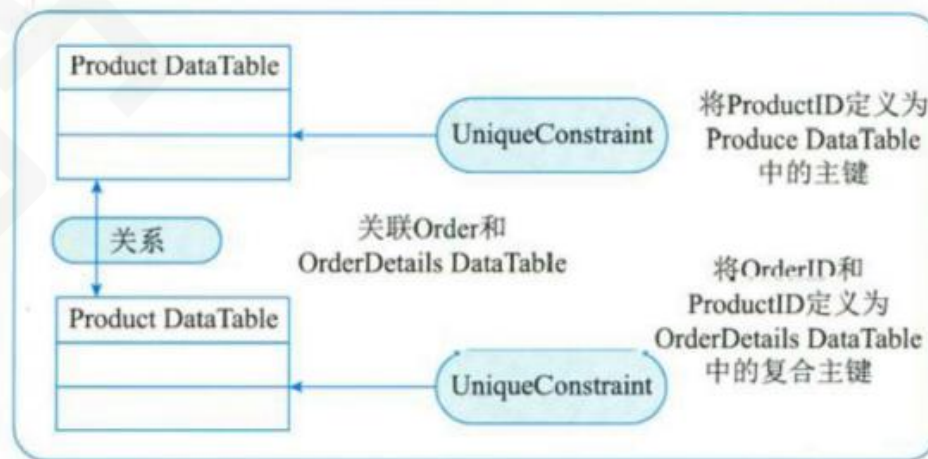
- interface 1: **过程定义导入/导出接口**。这个接口的特点是：转换格式和API调用，从而支持过程定义信息间的互相转换。
- interface 2: **客户端应用程序接口**。通过这个接口工作流机可以与任务表处理器交互，代表用户资源来组织任务。然后由任务表处理器负责，从任务表中选择、推进任务项。由任务表处理器或者终端用户来控制应用工具的活动。
- interface 3: **应用程序调用接口**。允许工作流机直接激活一个应用工具，来执行一个活动。典型的是调用以后台服务为主的应用程序，没有用户接口。当执行活动要用到的工具，需要与终端用户交互，通常是使用客户端应用程序接口来调用那个工具，这样可以为用户安排任务时间表提供更多的灵活性。
- interface 4: **工作流机协作接口**。其目标是定义相关标准，以使不同开发商的工作流系统产品相互间能够进行无缝的任务项传递。
- interface 5: **管理和监视接口**。提供的功能包括用户管理、角色管理、审查管理、资源控制、过程管理和过程状态处理器等。
- 用工作流的思想组织业务逻辑，优点是：将应用逻辑与过程逻辑分离，在不修改具体功能的情况下，通过修改过程模型改变系统功能，完成对生产经营部分过程或全过程的集成管理，可有效地把人、信息和应用工具合理地组织在一起，发挥系统的最大效能。



中层框架设计

- 实体设计：业务逻辑层**实体提供对业务数据及相关功能(在某些设计中)的状态编程访问**。业务逻辑层实体可以使用**具有复杂架构的数据**来构建，这种数据通常来自数据库中的多个相关表。
- 在应用程序中表示业务逻辑层实体的方法有很多(从以数据为中心的模型到更加面向对象的表示法)，如XML、通用DataSet、有类型的DataSet等。
- 如下左图是业务实体用XML表示，右图所示为用于Order业务逻辑层实体的通用DataSet对象。此DataSet对象具有两个Data Table 对象，分别保存订单信息和订单详细信息。每个DataTable具有一个对应的UniqueConstraint对象，用于标识表中的主键。此外，该DataSet 还有一个Relation对象，用于将订单详细信息与订单相关联。

```
<?xml version="1.0"?>
<Product xmlns="urn:aUniqueNamespace">
  <ProductID>1</ProductID>
  <ProductName>Chai</ProductName>
  <QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit>
  <UnitPrice>18.00</UnitPrice>
  <UnitsInStock>39</UnitsInStock>
  <UnitsOnOrder>0</UnitsOnOrder>
  <ReorderLevel>10</ReorderLevel>
</Product>
```



中层框架设计

- 业务框架位于系统架构的中间层，是实现系统功能的核心组件。采用容器的形式，便于系统功能的开发、代码重用和管理。下图便是在吸收了SOA 思想之后的一个三层体系结构的简图。
- 业务层采用业务容器的方式存在于整个系统当中，采用此方式可以大大降低业务层和相邻各层的耦合，表示层代码只需要将业务参数传递给业务容器，而不需要业务层多余的干预。如此一来，可以有效地防止业务层代码渗透到表示层。
- 在业务容器中，业务逻辑是按照Domain Model—Service—Control思想来实现的。
 - Domain Model：是领域层业务对象，它仅仅包含业务相关的属性。
 - Service：是业务过程实现的组成部分，是应用程序的不同功能单元，通过在这些服务之间定义良好的接口和契约联系起来。
 - Control：服务控制器，是服务之间的纽带，不同服务之间的切换就是通过它来实现的。

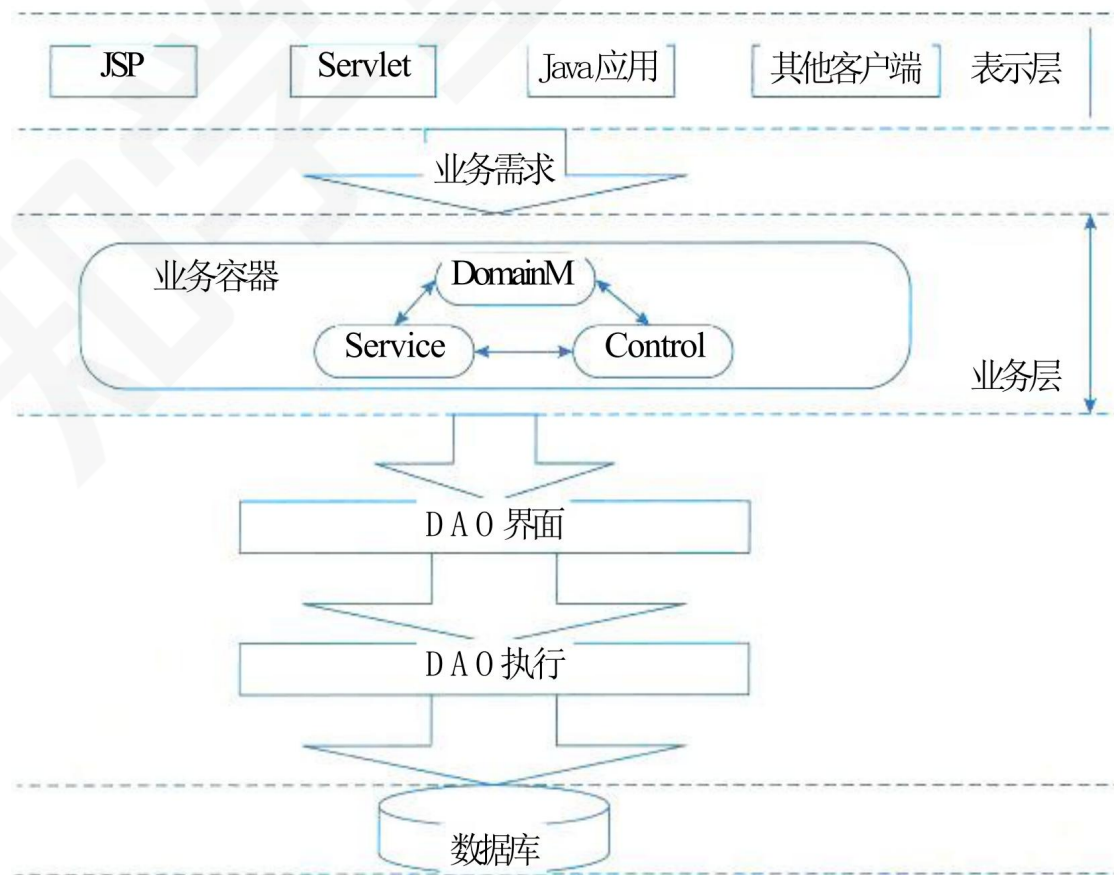


图13-10业务框架在整个系统架构中的位置



数据访问层设计

5种数据访问模式：

- **在线访问**：会占用一个数据库连接，读取数据，每个数据库操作都会通过这个连接不断地与后台的数据源进行交互。比如使用pl/sql或者navicat连接数据库
- **Data Access Object**：是标准J2EE设计模式之一，开发人员常常用这种模式将底层数据访问操作与高层业务逻辑分离开。
- **Data Transfer Object**：是经典EJB设计模式之一。DTO本身是这样一组对象或是数据的容器，它需要跨不同的进程或是网络的边界来传输数据。这类对象本身应该不包含具体的业务逻辑，并且通常这些对象内部只能进行一些诸如内部一致性检查和基本验证之类的方法，而且这些方法最好不要再调用其他的对象行为。
- **离线数据模式**是**以数据为中心**，数据从数据源获取之后，将按照某种预定义的结构存放在系统中，成为应用的中心。离线，对数据的各种操作独立于各种与后台数据源之间的连接或是事务
- **对象/关系映射**(Object/Relation Mapping, O/R Mapping)：大多数应用中的数据都是依据关系模型存储在关系型数据库中；而很多应用程序中的数据在开发或是运行时则是以对象的形式组织起来的。那么，对象/关系映射就提供了这样一种工具或是平台，能够帮助将**应用程序中的数据转换成关系型数据库中的记录**；或是**将关系数据库中的记录转换成应用程序中代码便于操作的对象**



数据访问层设计

- **工厂模式**在数据库访问层的应用：首先**定义一个操纵数据库的接口DataAccess**，然后根据**数据库的不同，由类工厂决定实例化哪个类**。因为DataAccess的具体实现类有一些共同的方法，所以先从DataAccess实现一个抽象AbstractDataAccess类，包含一些公用方法。然后，分别为SQL Server、Oracle和MySQL数据库编写三个数据访问的具体实现类。现在已经完成了所要的功能，下面需要创建一个Factory类，来实现自动数据库切换的管理。这个类很简单，主要的功能就是根据数据库类型，返回适当的数据库操纵类。
- **事务处理设计**：JavaBean中使用**JDBC**方式进行事务处理：在JDBC中，打开一个连接对象Connection时，默认是auto-commit模式，每个SQL语句都被当作一个事务，即每次执行一个语句，都会自动地得到事务确认。为了能将多个SQL语句组合成一个事务，要将auto-commit模式屏蔽掉。在auto-commit模式屏蔽掉之后，如果不调用commit()方法，SQL语句不会得到事务确认。在最近一次commit()方法调用之后的所有SQL会在方法commit()调用时得到确认。
- **连接对象管理设计（数据库连接池）**：通过**资源池**解决资源频繁分配、释放所造成的问题。有了这个连接池，下面就可以提供一套自定义的分配、释放策略。当客户请求数据库连接时，首先看连接池中是否有未分配出去的连接。如果存在空闲连接则把连接分配给客户，并标记该连接为已分配。若连接池中没空闲连接，就在已经分配出去的连接中，寻找一个合适的连接给客户，此时该连接在多个客户间复用。当客户释放数据库连接时，可以根据该连接是否被复用，进行不同的处理。如果连接没有使用者，就放入到连接池中，而不是被关闭。



数据架构规划与设计

XML文档分为两类：

- 一类是**以数据为中心**的文档，这种文档通常用于存储和传输结构化数据，数据元素的顺序通常不是关键因素，而更关注数据本身。一个典型的例子是使用XML来表示电子商务产品目录的数据。左图中，数据元素（如产品ID、名称和价格）都是规则的，并且存储了有关产品的结构化信息。**顺序不重要，只要能够访问和解析这些数据。**
- 另一类是**以文档为中心**的文档，这种文档用于发布描述性信息，内容可能比较零散，元素之间的顺序可能很重要。一个典型的例子是使用XML来表示一篇新闻文章，右图中，元素的顺序很重要，因为它们定义了文章的结构，如标题、作者和内容。**这种XML文档用于在网页上发布描述性信息，文章的内容和结构通常由文档的作者或编辑精心安排。**

```
<products>
  <product>
    <id>101</id>
    <name>Laptop</name>
    <price>799.99</price>
  </product>
  <product>
    <id>102</id>
    <name>Smartphone</name>
    <price>299.99</price>
  </product>
</products>
```

```
<article>
  <title>Breaking News</title>
  <author>John Doe</author>
  <content>
    <paragraph>This is a breaking news story...</paragraph>
    <paragraph>More details to follow...</paragraph>
  </content>
</article>
```



数据架构规划与设计

经提出的XML文档的存储方式有两种：

- 基于文件的存储方式。基于文件的存储方式是指将XML文档按其原始文本形式存储，主要存储技术包括操作系统文件库、通用文档管理系统和传统数据库的列。这种存储方式需维护某种类型的附加索引，以建立文件之间的层次结构。基于文件的存储方式的特点：无法获取XML文档中的结构化数据;通过附加索引可以定位具有某些关键字的XML文档，一旦关键字不确定，将很难定位;查询时，只能以原始文档的形式返回，即不能获取文档内部信息;文件管理存在容量大、管理难的缺点。
- 数据库存储方式。数据库在数据管理方面具有管理方便、存储占用空间小、检索速度快、修改效率高和安全性好等优点。一种比较自然的想法是采用数据库对XML文档进行存取和操作，这样可以利用相对成熟的数据库技术处理XML文档内部的数据。数据库存储方式的特点：能够管理结构化和半结构化数据;具有管理和控制整个文档集合本身的能力;可以对文档内部的数据进行操作;具有数据库技术的特性，如多用户、并发控制和一致性约束等;管理方便，易于操作。



物联网层次架构设计

物联网可以分为三个层次，**底层是用来感知数据的感知层**，即利用传感器、二维码、RFID等设备随时随地获取物体的信息。**第二层是数据传输处理的网络层**，即通过各种传感网络与互联网的融合，将对象当前的信息实时准确地传递出去。**第三层则是与行业需求结合的应用层**，即通过智能计算、云计算等将对象进行智能化控制。

- 感知层：用于识别物体、采集信息。感知层包括二维码标签和识读者、RFID标签和读写器、摄像头、GPS、传感器、M2M终端、传感器网关等，主要功能是识别对象、采集信息，与人体结构中皮肤和五官的作用类似。**感知层解决的是人类世界和物理世界的获取数据问题。**
- 网络层：用于传递信息和处理信息。网络层包括通信网与互联网的融合网络、网络管理中心、信息中心和智能处理中心等。网络层将感知层获取的信息进行传递和处理，类似于人体结构中的神经中枢和大脑。**网络层解决的是传输和预处理感知层所获得数据的问题。**
- 应用层：实现广泛智能化。应用层是物联网与行业专业技术的深度融合，结合行业需求实现行业智能化，这类似于人们的社会分工。物联网应用层利用经过分析处理的感知数据，为用户提供丰富的特定服务。**应用层解决的是信息处理和人机交互的问题。**



层次是架构案例分析

电子商务网站(网上商店PetShop)

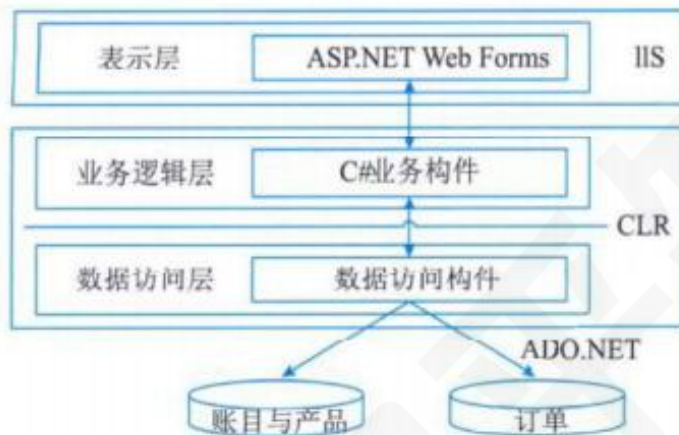


图 13-15 Net 中标准的 BS 分层式结构



图 13-16 PetShop 2.0 的体系架构

从图13-16中可以看到，并没有明显的数据库访问层设计。这样的设计虽然提高了数据库访问的性能，但也同时导致了业务逻辑层与数据库访问的职责混乱。

PetShop 3.0纠正了此前层次不明的问题，将数据库访问逻辑作为单独的一层独立出来。

层次是架构案例分析

PetShop 4.0基本上延续了3.0的结构，但在性能上作了一定的改进，引入了缓存和异步处理机制，同时又充分利用了ASP.Net 2.0的新功能MemberShip。可以看到，在数据访问层中，完全采用了“面向接口编程”思想。抽象出来的IDAL模块，脱离了与具体数据库的依赖，从而使得整个数据访问层有利于数据库迁移。DALFactory模块专门管理DAL对象的创建，便于业务逻辑层访问。SQLServerDAL 和OracleDAL模块均实现IDAL模块的接口，其中包含的逻辑就是对数据库的Select、Insert、Update和Delete操作。因为数据库类型的不同，对数据库的操作也有所不同，代码也会因此有所区别。

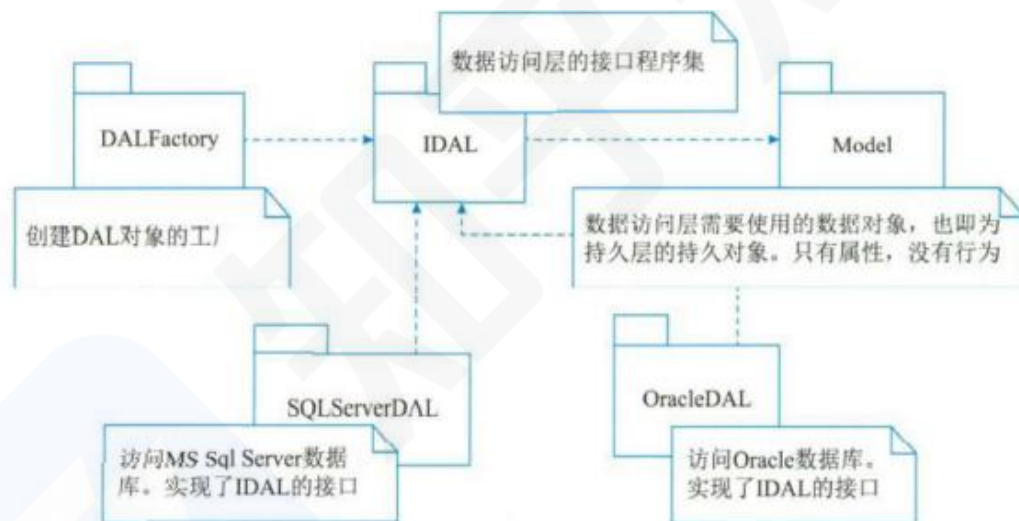


图 13-19 数据访问层的模块结构图

层次是架构案例分析

此外，抽象出来的IDAL模块，除了解除了向下的依赖之外，对于其上的业务逻辑层同样仅存在弱依赖关系，如图13-20所示。图13-20中，**BLL是业务逻辑层的核心模块**，它包含了整个系统的核心业务。在业务逻辑层中，不能直接访问数据库，而必须通过数据访问层。注意，图13-20中对数据访问业务的调用，是通过**接口模块IDAL来完成的**。既然与具体的数据访问逻辑无关，则层与层之间的关系就是松散耦合的。如果此时需要修改数据访问层的具体实现，只要不涉及IDAL的接口定义，那么业务逻辑层就不会受到任何影响。毕竟，具体实现的SQLServerDAL和OracaIDAL根本就与业务逻辑层没有半点关系。

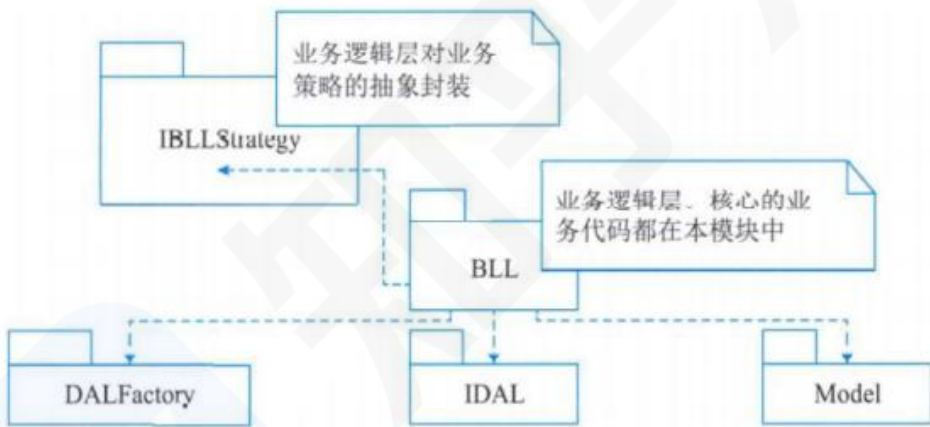


图 13-20 业务逻辑层的模块结构图

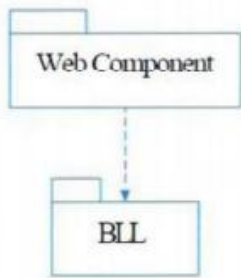


图13-21 表示层的模块结构



层次是架构案例分析

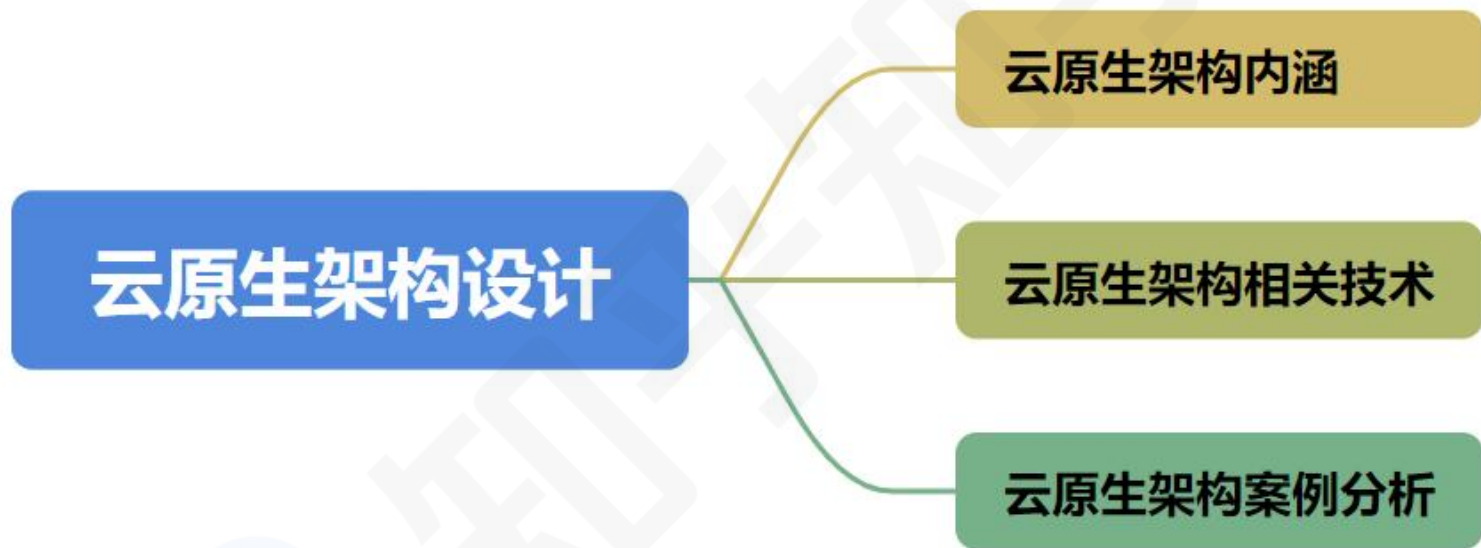
基于物联网架构的电子小票服务系统

采用感知层、网络层和应用层的3层物联网体系架构模型：





云原生架构设计





云原生架构内涵

- 云原生架构是**基于云原生技术**的一组架构原则和设计模式的集合，旨在将**云应用中的非业务代码部分进行最大化的剥离**，从而让云设施接管应用中原有的大量非功能特性(如弹性、韧性、安全、可观测性、灰度等)，使业务不再有非功能性业务中断困扰的同时，具备轻量、敏捷、高度自动化的特点。
- 云原生的代码通常包括三部分：业务代码、三方软件、处理非功能特性的代码。从业务代码中剥离大量非功能性特性(不会是所有，比如易用性还不能剥离)到IaaS(基础设施即服务)和PaaS(平台即服务)中。
- 具备云原生架构的应用可以**最大程度利用云服务和提升软件交付能力，进一步加快软件开发**。其特点包括代码结构发生巨大变化、非功能性特性大量委托、高度自动化的软件交付。
- 云原生架构原则
 - 服务化原则：拆分为微服务架构、小服务架构，分别迭代。
 - 弹性原则：系统的部署规模可以随着业务量的变化而自动伸缩。
 - 可观测原则：通过日志、链路跟踪和度量等手段。
 - 韧性原则：当软件所依赖的软硬件组件出现各种异常时，软件表现出来的抵御能力。
 - 所有过程自动化原则：一方面标准化企业内部的软件交付过程，另一方面在标准化的基础上进行自动化，通过配置数据自描述和面向终态的交付过程。
 - 零信任原则：默认情况下不应该信任网络内部和外部的任何人/设备/系统，需要基于认证和授权重构访问控制的信任基础，以身份为中心。
 - 架构持续演进原则：云原生架构本身也必须是一个具备持续演进能力的架构。



云原生架构内涵

主要架构模式

- **服务化架构模式**：典型模式是**微服务和小服务**模式。通过服务化架构，把**代码模块关系和部署关系进行分离**，每个接口可以部署不同数量的实例，单独扩缩容，从而使得整体的部署更经济。
- **Mesh化架构模式**：把**中间件框架(如RPC、缓存、异步消息等)**从业务进程中分离，让中间件SDK与业务代码进一步解耦，从而使得中间件升级对业务进程没有影响。分离后在业务进程中只保留很“薄”的Client部分。
- **Serverless模式**：将“部署”这个动作从运维中“收走”，使开发者不用关心应用运行地点、操作系统、网络配置、CPU性能等，也就是把应用的整个运行都委托给云。
- **存储计算分离模式**：在云环境中，推荐把各类暂态数据(如session)、结构化和非结构化持久数据都采用云服务来保存，从而实现存储计算分离。
- **分布式事务模式**：大颗粒度的业务需要访问多个微服务，必然带来分布式事务问题，否则数据就会出现不一致。架构师需要根据不同的场景选择合适的分布式事务模式。
- **可观测架构**：可观测架构包括Logging、Tracing、Metrics 三个方面，其中Logging提供多个级别的详细信息跟踪，由应用开发者主动提供;Tracing提供一个请求从前端到后端的完整调用链路跟踪对于分布式场景尤其有用; Metrics则提供对系统量化的多维度度量。
- **事件驱动架构**：本质上是一种应用/组件间的集成架构模式。可用于服务解耦、增强服务韧性、数据变化通知等场景中。



云原生架构相关技术

- 容器技术：容器作为标准化软件单元，它将应用及其所有依赖项打包，使应用不再受环境限制，在不同计算环境间快速、可靠地运行。通过容器技术，企业可以充分发挥云计算弹性优势，降低运维成本。
- Kubernetes已经成为容器编排的事实标准，被广泛用于自动部署，扩展和管理容器化应用。Kubernetes提供了分布式应用管理的核心能力，包括：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。
- 云原生微服务：微服务模式将后端单体应用拆分为松耦合的多个子应用，每个子应用负责一组子功能。这些子应用称为“微服务”，多个“微服务”共同形成了一个物理独立但逻辑完整的分布式微服务体系。这些微服务相对独立，通过解耦研发、测试与部署流程，提高整体迭代效率。
- 微服务设计约束：
 - 微服务个体约束：功能在业务域划分上应是相互独立的，低耦合、单一职责。
 - 微服务与微服务之间的横向关系：。主要从微服务的可发现性和可交互性处理服务间的横向关系，一般需要服务注册中心。
 - 微服务与数据层之间的纵向约束：在微服务领域，提供数据存储隔离原则，即数据是微服务的私有资产，对于该数据的访问都必须通过当前微服务提供的API来访问。
 - 全局视角下的微服务分布式约束：故障发现时效性和根因精确性始终是开发运维人员的核心诉求



云原生架构相关技术

要微服务技术：

- Apache Dubbo作为源自阿里巴巴的一款开源高性能RPC框架，特性包括基于透明接口的RPC、智能负载均衡、自动服务注册和发现、可扩展性高、运行时流量路由与可视化的服务治理。
- Spring Cloud作为开发者的主要微服务选择之一，为开发者提供了分布式系统需要的配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性Token、全局锁、决策竞选、分布式会话与集群状态管理等能力和开发工具。
- Eclipse MicroProfile作为Java 微服务开发的基础编程模型，它致力于定义企业Java微服务规范，MicroProfile提供指标、API文档、运行状况检查、容错与分布式跟踪等能力，使用它创建的云原生微服务可以自由地部署在任何地方，包括服务网格架构。
- Tars是腾讯将其内部使用的微服务框架，包含一整套开发框架与管理平台，兼顾多语言、易用性、高性能与服务治理，理念是让开发更聚焦业务逻辑，让运维更高效。
- SOFAShark是由蚂蚁金服开源的一套用于快速构建金融级分布式架构的中间件，也是在金融场景里的最佳实践。
- Dapr(分布式应用运行时)是微软新推出的一种可移植的、无服务器的、事件驱动的运行时代，它使开发人员可以轻松构建弹性，无状态和有状态微服务，这些服务运行在云和边缘上，并包含多种语言和开发框架。



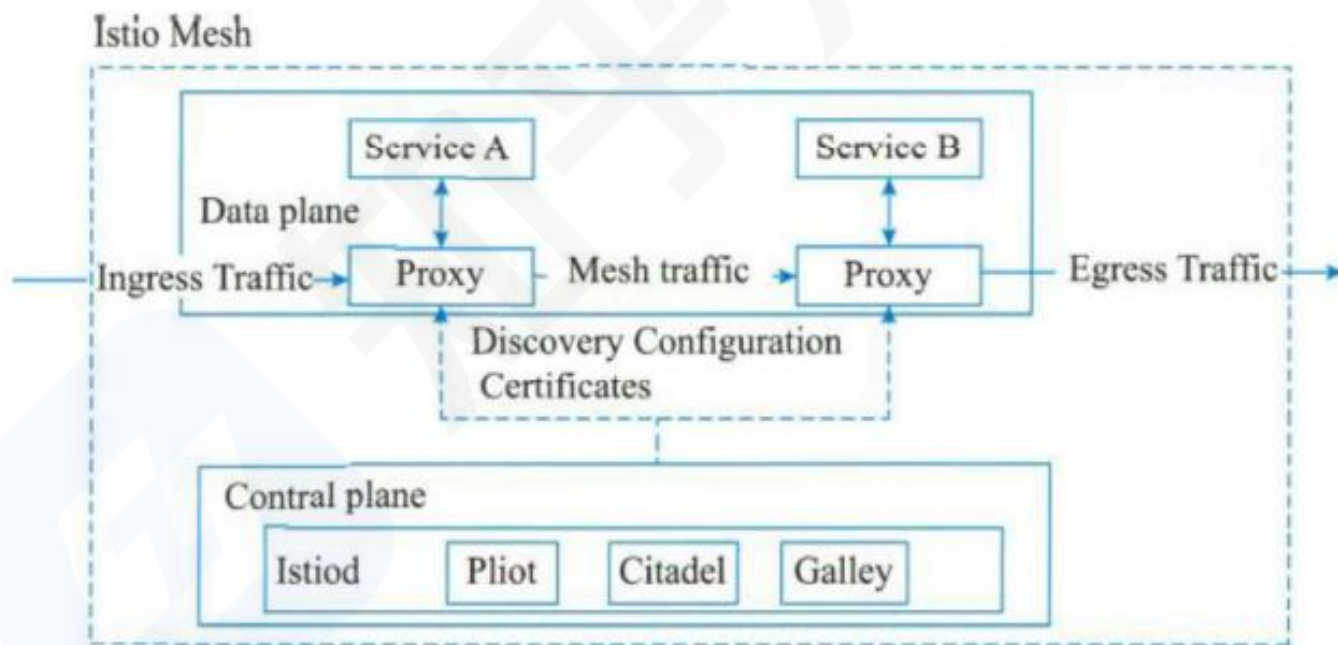
云原生架构相关技术

- **无服务器技术(Serverless)**因为屏蔽了服务器的各种运维复杂度，让开发人员可以将**更多精力用于业务逻辑设计与实现**，而逐渐成为云原生主流技术之一。， Serverless计算包含以下特征：
 - **全托管的计算服务**，客户只需要编写代码构建应用，无需关注同质化的、负担繁重的基于服务器等基础设施的开发、运维、安全、高可用等工作；
 - **通用性**，结合云BaaSAPI的能力，能够支撑云上所有重要类型的应用；
 - 自动弹性伸缩，让用户无需为资源使用提前进行容量规划；
 - **按量计费**，让企业使用成本得有效降低，无需为闲置资源付费。
- **函数计算(FaaS)**是Serverless中最具代表性的产品形态。通过把应用逻辑拆分多个函数，每个函数都通过事件驱动的方式触发执行。
- **无服务器技术关注点**：计算资源弹性调度、负载均衡和流控、安全性。



云原生架构相关技术

- **服务网格(ServiceMesh)是分布式应用在微服务软件架构之上发展起来的新技术**，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。这个解耦意味着**开发者无需关注微服务相关治理问题而聚焦于业务逻辑本身**，提升应用开发效率并加速业务探索和创新。
- 在这张架构图中，服务A调用服务B的所有请求，都被其下的服务代理截获，代理服务A完成到服务B的服务发现、熔断、限流等策略，而这些策略的总控是在控制平面(ControlPlane)上配置。



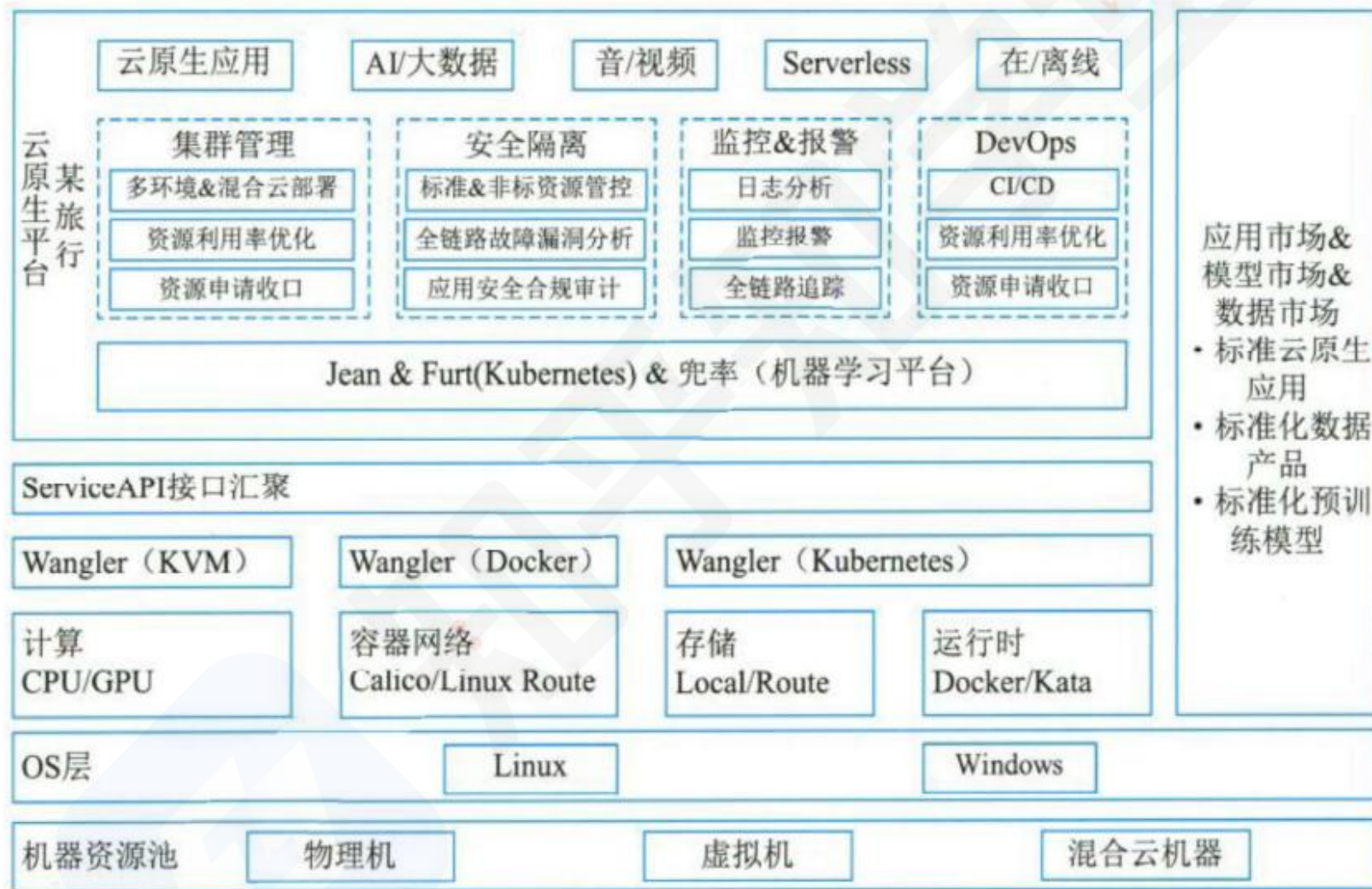


云原生架构案例分析

- **某旅行公司云原生改造**：某公司主体由两个公司合并之后，出现了以下2个问题：**技术体系不同，需要整合为一体；节假日高并发流量。**
- 改造第一阶段，某旅行技术团队为了提升集群资源利用率，降低资源使用成本。利用云原生思维重构部分技术体系，将多套旧有系统合并、收拢到一套以云原生应用为核心的私有云平台上，同时将IDC、物理网络、虚拟网络、计算资源、存储资源等通过IaaS、PaaS等，实现虚拟化封装、切割再投产的自动化流程。随着服务器集群规模的扩大，部分机器开始频繁出现故障。此时，保障服务稳定性成了第二阶段改造的首要任务。
- 第二阶段基于**公有云、私有云和离线专属云集群**等新型动态计算环境，某旅行公司的技术团队帮助业务构建和运行具有弹性的云原生应用，促进业务团队开始使用声明式API，同时通过不可变基础设施、服务网格和容器服务，来构建容错性好、易于管理和观察的应用系统，并结合平台可靠的自动化恢复、弹性计算来完成整个服务稳定性的提升。
- 第三阶段**通过基础组件、服务的云原生改造、服务依赖梳理和定义等方式**，使应用不再需要考虑底层资源、机房、运行时间和供应商等因素。此外，还利用标准的云原生应用模型，实现了服务的跨地域、跨云自动化灾备、自动部署，并向云原生场景下的DevOps 演进。架构图如下：



云原生架构案例分析





云原生架构案例分析

云原生技术助力某汽车公司数字化转型实践：

- 战略性构建**容器云平台**。通过平台实现对某云行App、二手车、在线支付、优惠券等核心互联网应用承载。以多租户的形式提供弹性计算、数据持久化、应用发布等面向敏捷业务服务，并实现高水平资源隔离。标准化交付部署，快速实现业务扩展，满足弹性要求。
- **数字混合云交付**。采用**私有云+公有云**的混合交付模式，按照服务的敏态/稳态特性和管控要求划分部署，灵活调度公有云资源来满足临时突发或短期高TPS业务支撑的需求。
- **深度融合微服务治理体系**，实现架构的革新和能力的沉淀，逐步形成支撑数字化应用的业务中台



T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

