上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

智能信息系统建模

Intelligent Information System Modeling

Lecture2 技术架构及发展

蔡鸿明

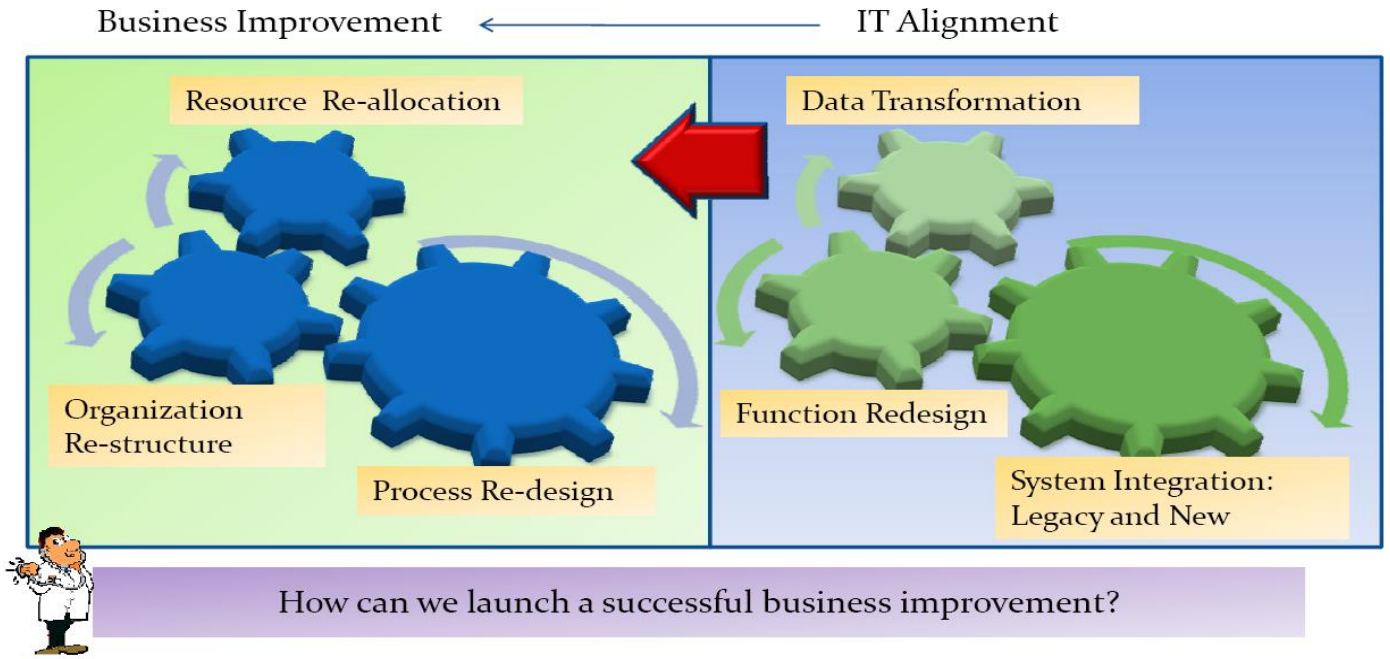
hmcai@sjtu.edu.cn

目录

- 1 软件的三种开发模式
  - 定制开发-套件实施-模型驱动架构
- 2 持续演化的软件技术架构
- 3 软件的部署运维模式
- 4 基于流程的软件过程
  - 开发实施过程-工具-角色职责
- 5 流程自动化RPA技术
- 6 小结

1 软件的三种开发模式

软件系统的开发实施过程覆盖了从业务问题到可运行系统之间的软件各阶段。



业务和IT对齐

当前，软件系统开发及实施方法可分为三种开发模式：

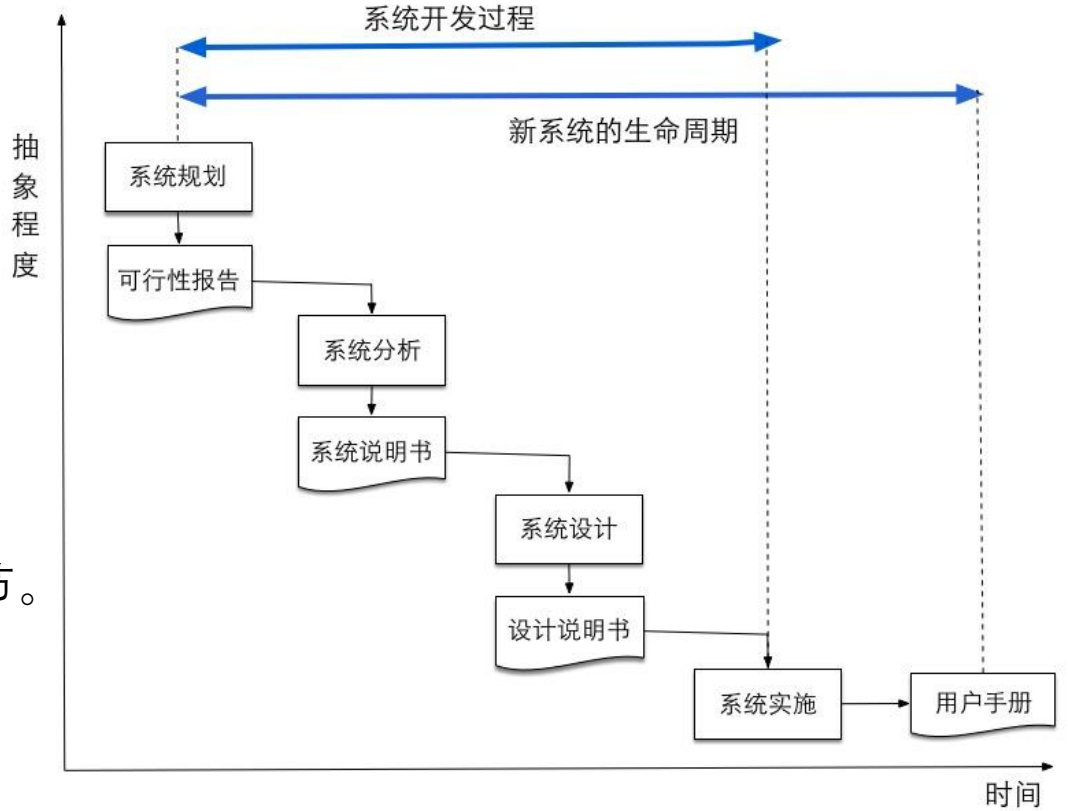
- 定制开发模式
- 套件实施模式
- 模型驱动架构模式（低代码）

1.1 基于定制开发模式的软件构造

- 软件系统的定制开发，是指软件提供商或软件开发人员为某企业或者组织的特定需求而定做的系统，通常为具有特殊业务需求的系统。
- 定制开发的软件系统一般具有以下特点：
  - 针对性强。每一个软件的开发都要经过细致的系统分析，针对不同企业的情况，编制最适用的程序。在编写软件的过程中，可以将管理者的最新管理思路或者最科学的管理模式融入到软件的数学模型中，从而大大提高了软件的科学价值，带给企业巨大的经济效益；
  - 使用方便。完全根据企业现有的工作流程编制程序，用户只需具备基本的计算机使用知识，就可以自如的操控软件，不必进行复杂的培训；
  - 效率更高。定制软件具有针对性强，完全按需定制开发，所以不像其他通用软件那样功能复杂，目标不统一，功能更加简洁，可根据使用需求随时进行调整，所以效率更高
  - 服务无忧。定制软件在使用过程中出现问题或需要优化升级时，全部由软件开发商上门解决，全方位服务让用户无后顾之忧。
- 定制开发是在不考虑时间、成本、质量等问题的情况下，可以解决客户提出的所有业务需求。

定制开发模式

- 根据用户的定制要求，按照软件工程要求，分需求分析、设计、开发、测试、部署等阶段
- 采用多种软件开发管理方法
- 理论上可以实现所有合理的用户需求，但开发周期、开发成本、软件质量差异很大
- 传统的软件开发大都属于该模式



传统软件的SDLC过程是适合定制开发模式的：

- 由于开发团队和用户的分离，
- 需求阶段需要以甲方主导，双方共同开展及确认，
- 测试阶段有时有用户甲方以及第三方测试团队的介入，
- 因此对于业务需求以及设计意图的把握是非常重要的环节。

定制开发模式

- **本质上是基于业务对齐IT的模式；**

- **优势：定制开发模式理论上可以满足所有的业务需求。**
  - **外包开发，运维服务。**
- **劣势：**
  - **人员：一般**完全按照用户现有的工作流程或者需求进行开发，但不同开发团队对于客户的需求把握和实现具有差异；
  - **周期：**根据最适用的软件要求仔细开发和分析，周期长；
  - **费用：**定制软件只为特定用途设计，复制能力有限，开发成本高。

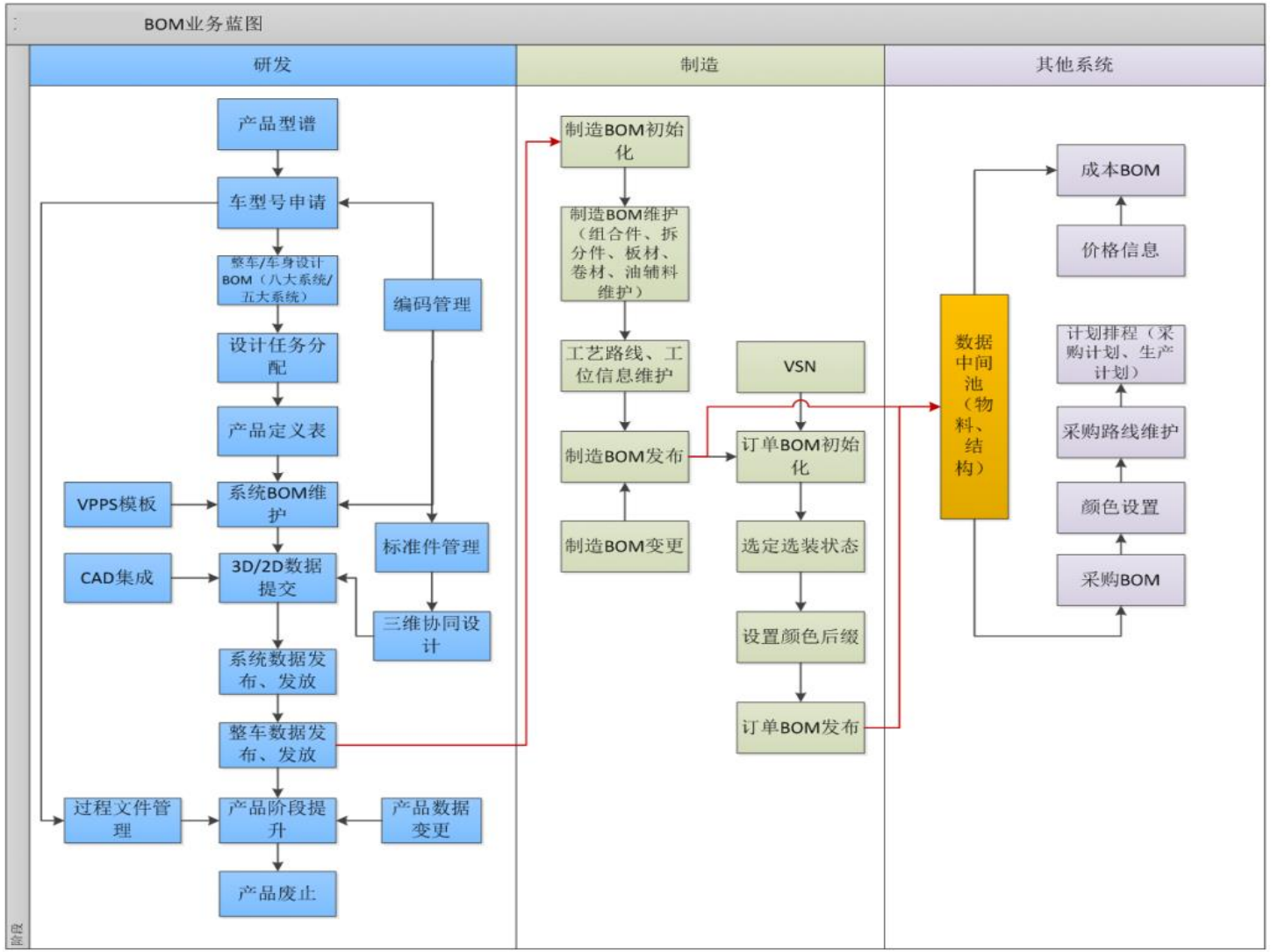
1.2 基于套件模式的软件系统实施

- 套件实施模式指的是基于一些商业化平台，通过业务模型导引下的数据和功能配置，搭建可用系统的过程。
  - 如SAP ERP、Oracle EBS（Electric Business Suite）、用友、金蝶
- 在这些软件系统套件基础上，建立覆盖业务需求的企业业务蓝图，指导软件系统的功能配置和界面设计，是设计及实施大型系统的重要策略。
- 设计业务蓝图的出发点是为了支持业务过程，促进客户、咨询顾问之间的沟通。

业务蓝图

- 企业业务蓝图即业务参考模型。业务蓝图的目的不是创建原型、生成代码或设计规范，而是将复杂业务进行流线式的过程描述。

业务蓝图实例



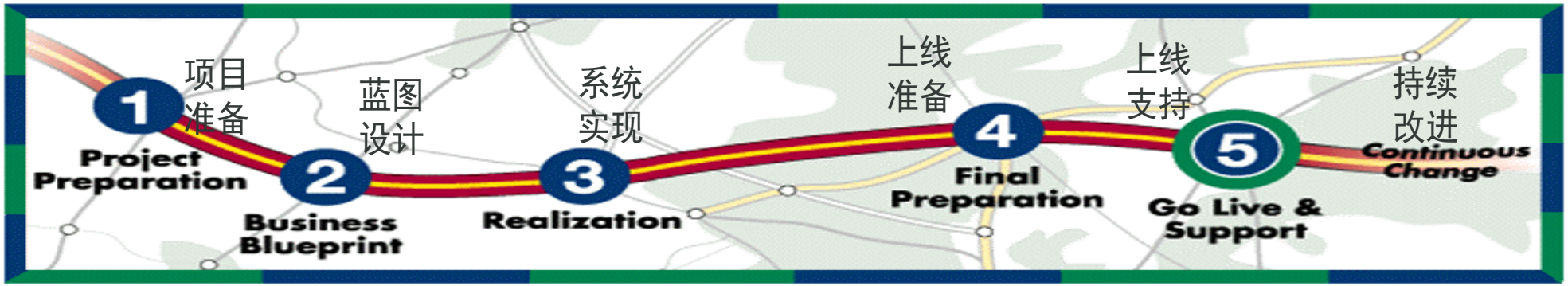
业务蓝图的发展

- 业务蓝图的发展过程较为曲折。
  - 70年代末80年代初，数据建模成为数据处理部门的首要任务，后来逐渐发展成为“企业建模”。
  - 数据建模的核心是在企业内部创建包括数据和过程的完整信息模型，其主要目的是开发和实现应用程序。
  - 事实上，在这些项目当中，许多模型其实并不能为应用程序的开发或者重构提供有益的帮助。
- 业务蓝图集中描述了企业业务的四个方面：数据、功能（任务）、组织、交互。
- 业务蓝图可以作为企业业务建模的起点，它隐藏了技术细节，以便业务用户可以集中全部注意力在业务过程问题上。
- 业务蓝图不仅面向业务处理，而且业务的描述表达方式适合几乎所有行业。

案例：SAP业务蓝图（业务参考模型）

- SAP提供的业务蓝图为每一个客户提供描述性的，但不必是标准化的解决方案，提供了各功能领域业务过程及其变量的广泛集合，为设计业务过程提供了很大范围。
- 业务蓝图包括企业的视图，其中包括样本业务对象，以及反映成功的案例中最佳操作的业务过程，其中的客户应用程序基础包括含有超过170个核心业务对象（采购单、销售订单、物料单等）和超过800个业务流程模型。
- 业务蓝图里描述了套件支持的业务案例，这使客户能很容易的看到并分析不同的业务解决方案。
- SAP业务蓝图将业务过程描述为事件驱动的过程链 (Event-driven Process Chains，EPC) 。

一种典型的套件系统实施方法（ASAP）



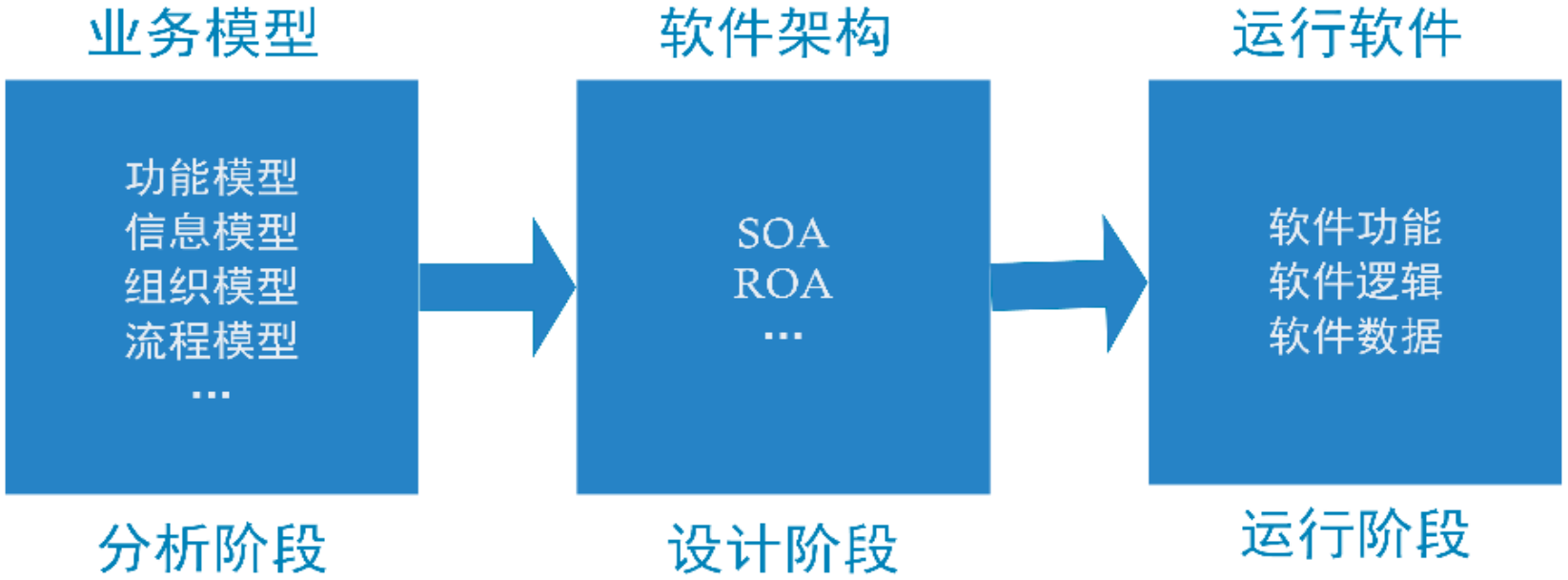
项目准备	蓝图设计	系统实现	上线准备	上线与支持	持续改进
• 项目规划 • 搭建项目组织 • 制定项目计划 • 制定管理章程 • 确定文档模板 • 确定责任分工 • 确定软硬件环境 • 软硬件准备 • 项目启动大会 • SAP概念培训 • 项目管理培训 • 文档模板培训 • 质量检查	• 阶段启动管理 • 业务现状调研 • 系统操作培训 • 业务模式确认 • 定义组织架构 • 业务流程设计 • 业务问题分析 • 数据整理启动 • 外围系统设计 • 制定开发计划 • 原型系统搭建 • 业务蓝图汇报 • 质量检查	• 阶段启动管理 • 技术参数设定 • 程序/接口开发 • 系统后台配置 • 单元测试 • 集成测试 • SAP权限设置 • 关键用户培训 • 最终用户培训 • 网络硬件审核 • 先期变革导入 • 质量检查	• 阶段启动管理 • 上线编程验收 • 制定上线计划 • 服务器检查 • 系统管理培训 • 上线动员大会 • 静态数据导入 • 库存动态盘点 • 订单Cut转 • 月结数据导出 • 动态数据导入 • 系统数据校准 • 质量检查	• 上线切换运行 • 生产系统支持 • 业务数据监控 • 月结编程验收 • 财务月结支持 • IT人员编制调整 • 自行维护启动 • 补充知识培训 • 业务蓝图修正 • 项目结束 • 上线总结	• 新业务需求清理 • 后续项目计划 • 业务数据监控 • 自身梯队培养

套件实施模式

- 本质上是基于IT对齐业务的模式；
- 优势：
  - 开发周期短、开发成本较低、软件质量较为稳定。
- 劣势：
  - 受软件已有功能和架构的限制，用户的部分特别需求往往难以得到完全满足。
    - 因此，很多情况下，最佳实践反而成为基于套件的软件系统实施饱受诟病的起点；
    - 一些用户认为这是实施团队要求企业业务适应应用系统，而不是进行开发以实现业务需求的借口。

1.3 业务模型驱动的软件架构模式

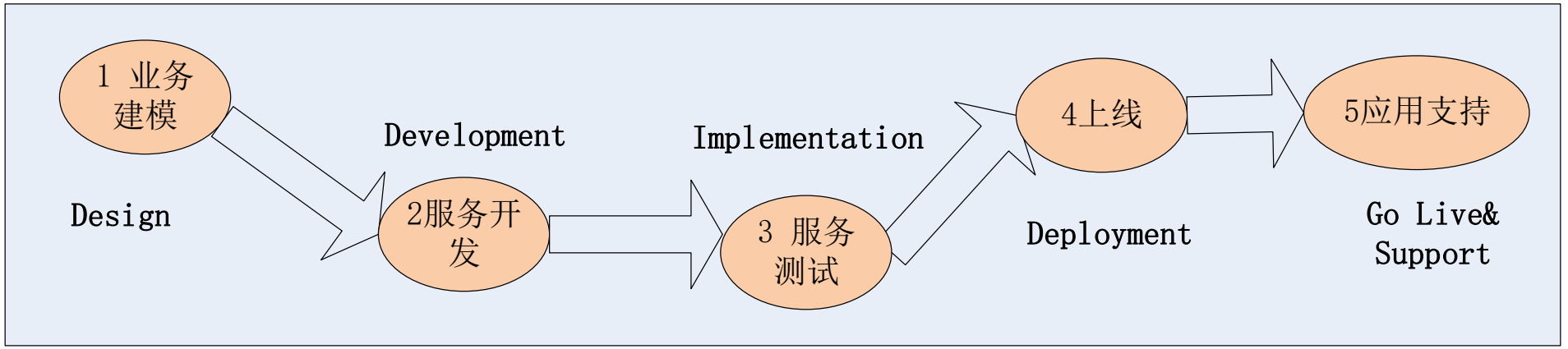
- 业务模型驱动的IT架构方式是当前软件系统分析与设计的重要发展方向。
- 模型驱动架构模式，是指在构建的业务模型基础上，通过模型驱动的转换，实现软件的架构，进而构造出可运行的软件。
- 在业务模型基础上构造出高度集成的软件系统，不仅会改善整体业务，而且可以使软件更容易调整以适应将来的业务变化。



SOA（Service Oriented Architecture）
ROA（Resource Oriented Architecture）
Micro-Service Architecture
Serverless Computing

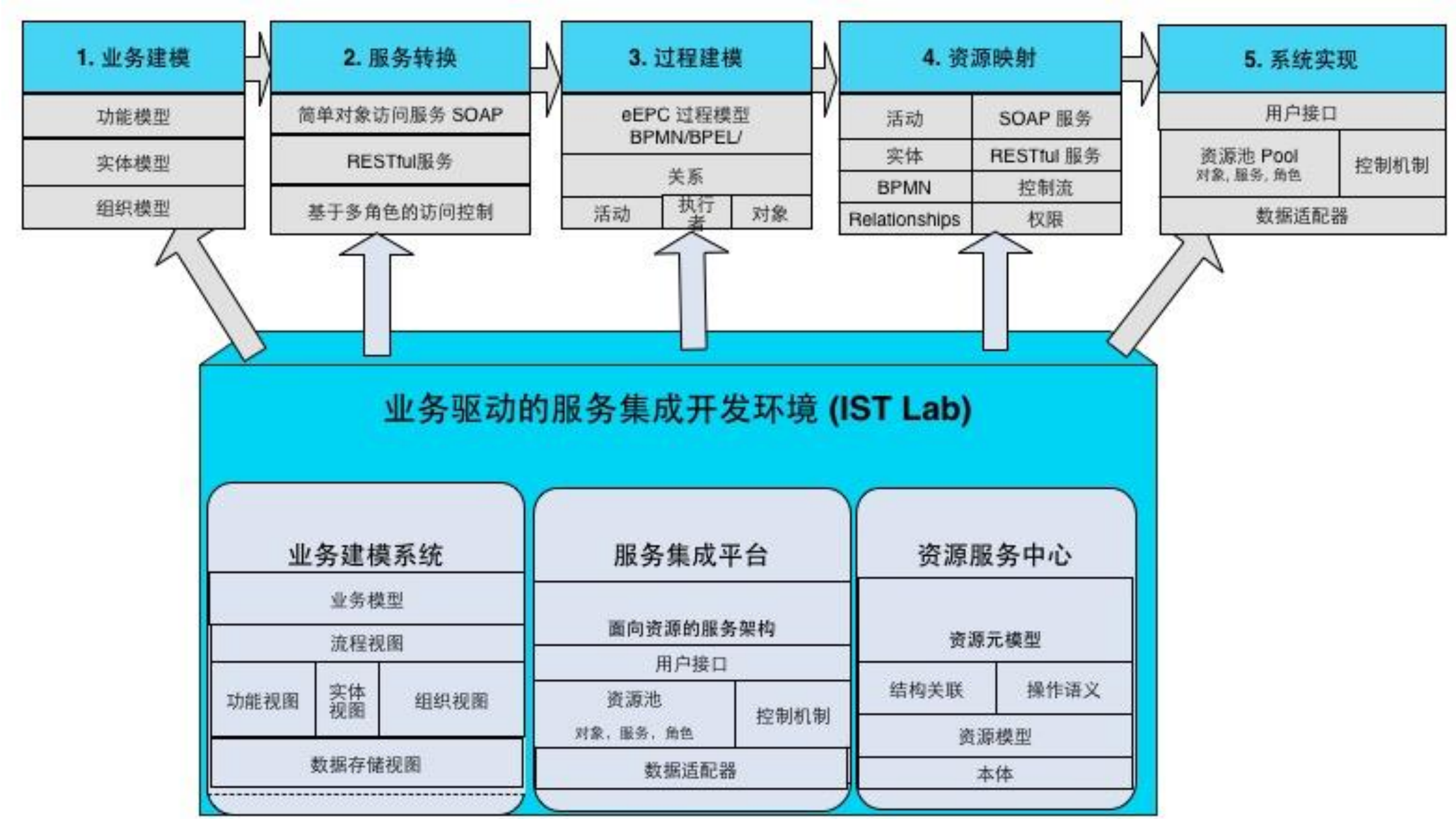
实施过程

- 一种业务模型驱动的软件架构的各方法阶段。



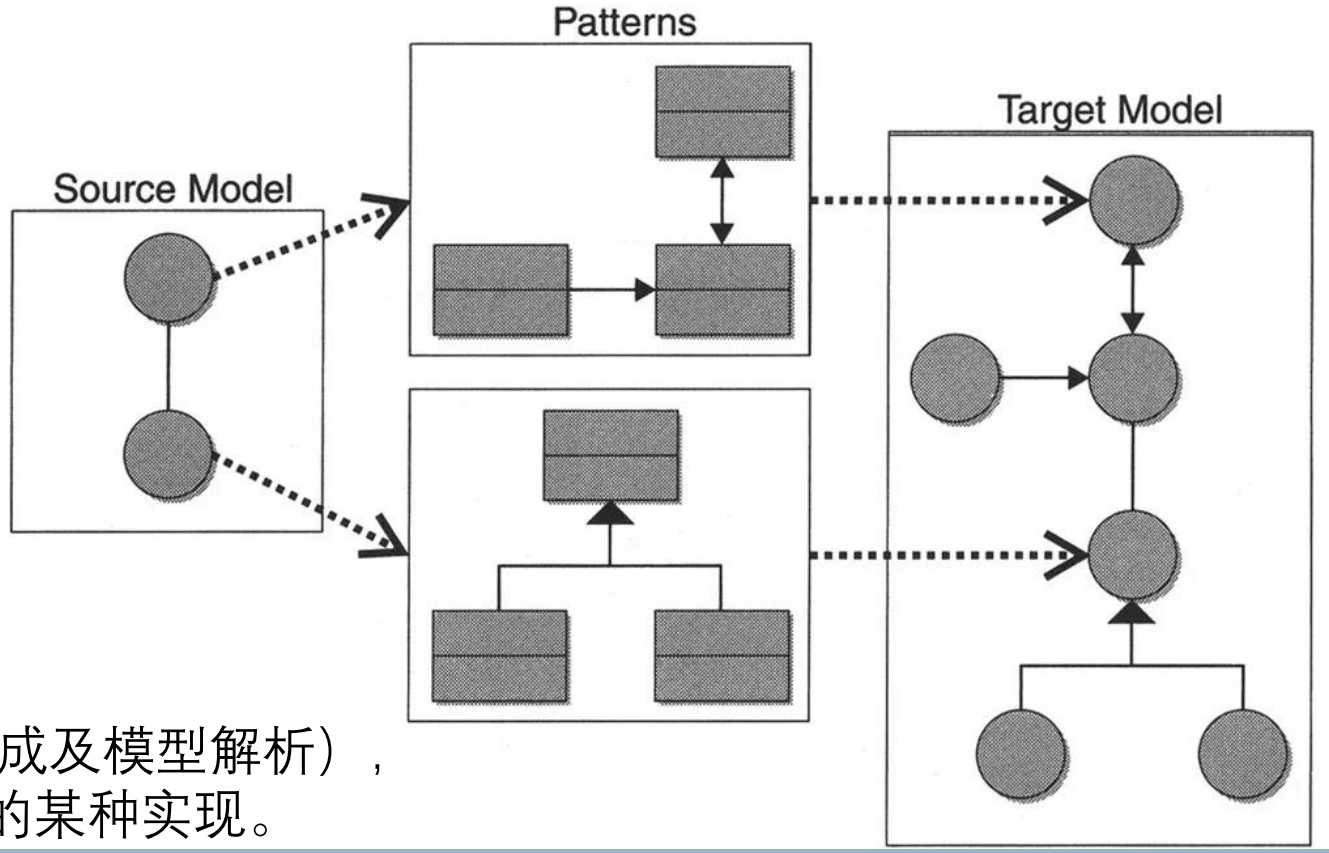
业务建模	创建并提交应用	应用服务测试	应用服务提交部署	服务提供
业务现状调研 业务框架确定 定义功能模型 定义组织角色 定义数据对象 建立过程模型 业务模型检查 模型向服务映射 服务存储 发布服务 质量检查	技术参数确定 服务流程定义 服务接口开发 服务组装及开发 界面开发 权限设定 单元测试 集成测试 提交应用 质量检查	测试计划 服务功能测试 服务性能测试 服务集成测试 压力测试 安全测试 服务测试总结 服务审 质量检查	制定上线计划 部署环境准备 设定服务水平 设定部署环境 设置虚拟机条件 部署应用 设置初始数据 质量检查	上线切换 用户数据维护 业务数据监控 综合管理 统计分析 持续改进计划 业务模型修正 新需求整理 后续项目计划 上线总结

支持平台


1. 业务建模
功能模型
实体模型
组织模型
2. 服务转换
简单对象访问服务 SOAP
RESTful服务
基于多角色的访问控制
3. 过程建模
eEPC 过程模型
BPMN/BPEL/
关系
活动
执行者
对象
4. 资源映射
活动
SOAP 服务
实体
RESTful 服务
BPMN
控制流
Relationships
权限
5. 系统实现
用户接口
资源池 Pool
对象, 服务, 角色
控制机制
数据适配器
业务驱动的服务集成开发环境 (IST Lab)
业务建模系统
业务模型
流程视图
功能视图
实体视图
组织视图
数据存储视图
服务集成平台
面向资源的服务架构
用户接口
资源池
对象，服务，角色
控制机制
数据适配器
资源服务中心
资源元模型
结构关联
操作语义
资源模型
本体

业务模型驱动架构模式讨论

- 模型驱动架构模式介于定制开发和套件实施的两种模式之间，具有一定灵活性又能保证软件质量，是当前应用的主要模式。
- 是缩小业务需求和技术实现之间的差距，实现业务技术融合互动的有效途径。
- 从业务与IT对齐角度，模型驱动开发MDD和低代码LCDP都是这种模式的具体化实现技术。



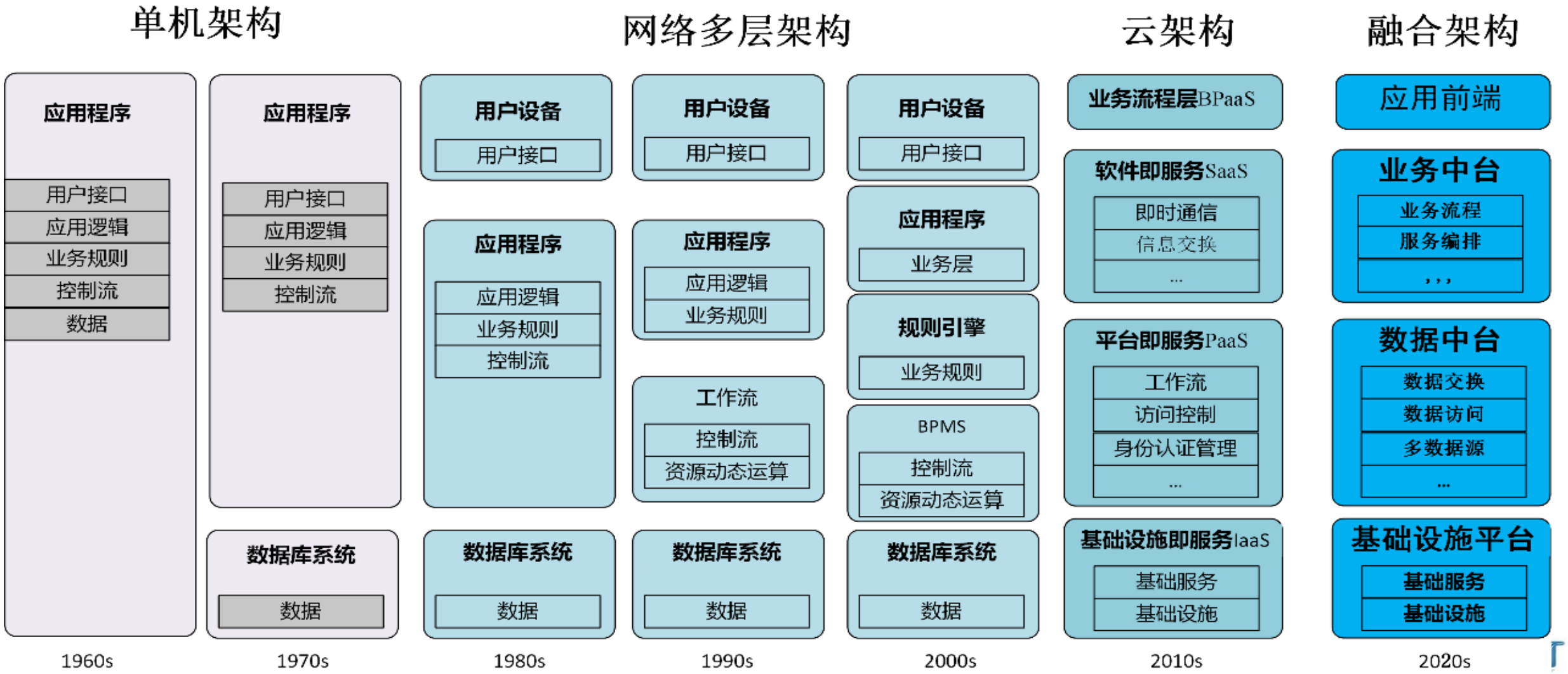
低代码LCDP技术的两种主要实现方法（代码生成及模型解析），基于元模型的模型解析方法可以归为这种思想的某种实现。

目录

- 1 软件的三种开发模式
  - 定制开发-套件实施-模型驱动架构
- 2 持续演化的软件技术架构
- 3 软件的部署运维模式
- 4 基于流程的软件过程
  - 开发实施过程-工具-角色职责
- 5 流程自动化RPA技术
- 6 小结

2.1 软件架构的演化阶段

软件架构的演化可以分为几个阶段:

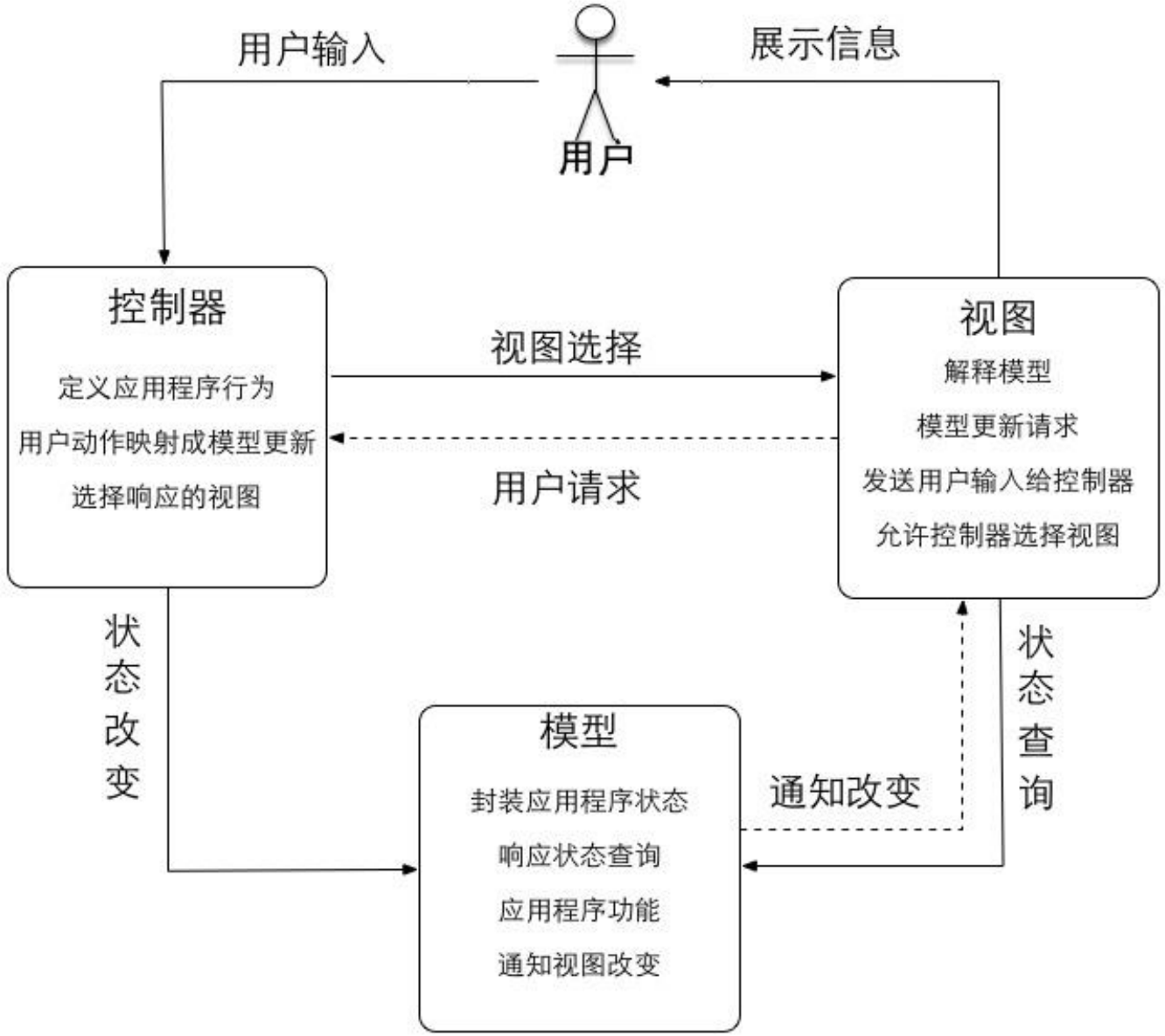


软件架构演化的主要阶段

- **主机系统**
  - 单机系统：所有的数据、控制逻辑，业务规则，应用逻辑和用户界面都在同一系统中。
  - 数据库系统：数据和功能分离导致了数据库系统的出现，应用系统往往分为数据库及其应用端，往往以主机系统为主。
- **网络架构**
  - 三层架构：网络的发展，进一步导致了多客户端系统的产生，系统往往分为C/S或B/S的客户端前端、应用服务端，以及数据库系统三层架构。
  - 工作流系统：体现过程控制以及资源调度的工作流引擎出现，导致了业务逻辑进一步分解为动态的工作流，以及静态的业务规则和应用逻辑。
  - 多层架构：应用服务大量出现的复用，使得工作流引擎演化为支持服务调用和流程全生命周期的BPM，包含多层灵活架构的软件基本成型。前端主要体现为网页、移动端、界面等应用前端，业务端主要体现为服务、规则、流程等应用逻辑，数据端体现为数据库、文件等多种数据源。
- **云架构**
  - 数据、平台、前端等层次概念逐渐模糊，软硬件的功能分配也不再明显，基础设施开始纳入整体软件，由基础设施层（IaaS）、平台层（PaaS）、服务层（SaaS）、业务流程层（BPaaS）等的分层划分逐渐扩展，体现为高度的灵活性和复杂的耦合性。
  - aPaaS（ability as a service），aPaaS 体系相比其他架构就是多出来了一个开发和部署应用程序的能力，即 aPaaS 赋予了原来平台进一步的软件服务体系开发和部署的能力。
- **融合架构**
  - 中台：数据中台、业务中台、技术中台；
  - 边缘计算，雾计算，云边端架构

基于MVC模式的多层架构

互联网条件下，基于MVC模式划分由于简洁清晰，通用性强而成为当前多层软件架构的主流。



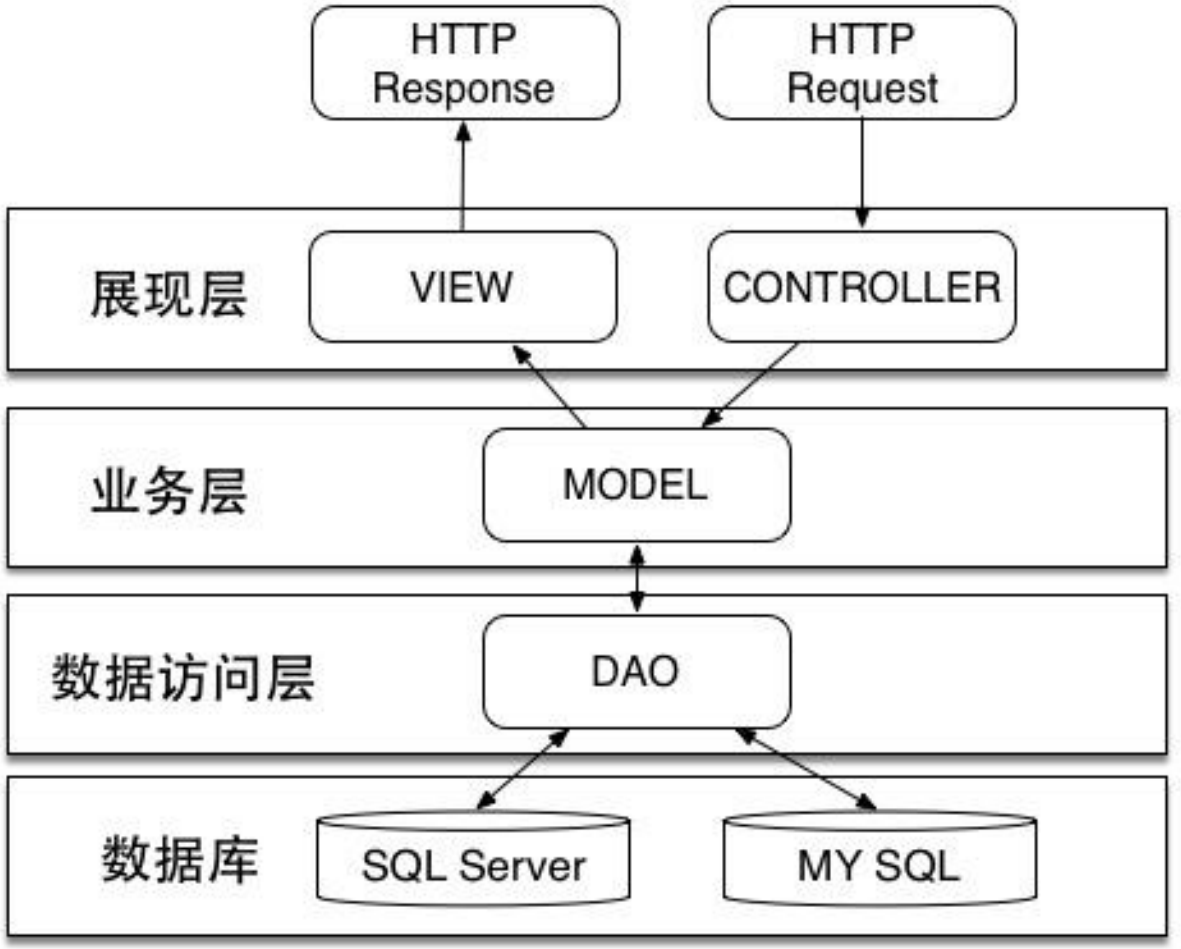
MVP（Model-View-Presenter）

MVVM（Model-View-ViewModel）

经典多层架构

经典多层架构四层包括：

- 展现层：接受用户的输入和展示数据或处理结果。
- 业务层：包含工作流控制逻辑，但不包含业务逻辑的处理，相对功能单一。
- 数据访问层：包含整个业务系统的业务逻辑，实现数据的存取。
- 数据库：提供整个业务系统的数据内容及管理。



云-边-端协同计算架构

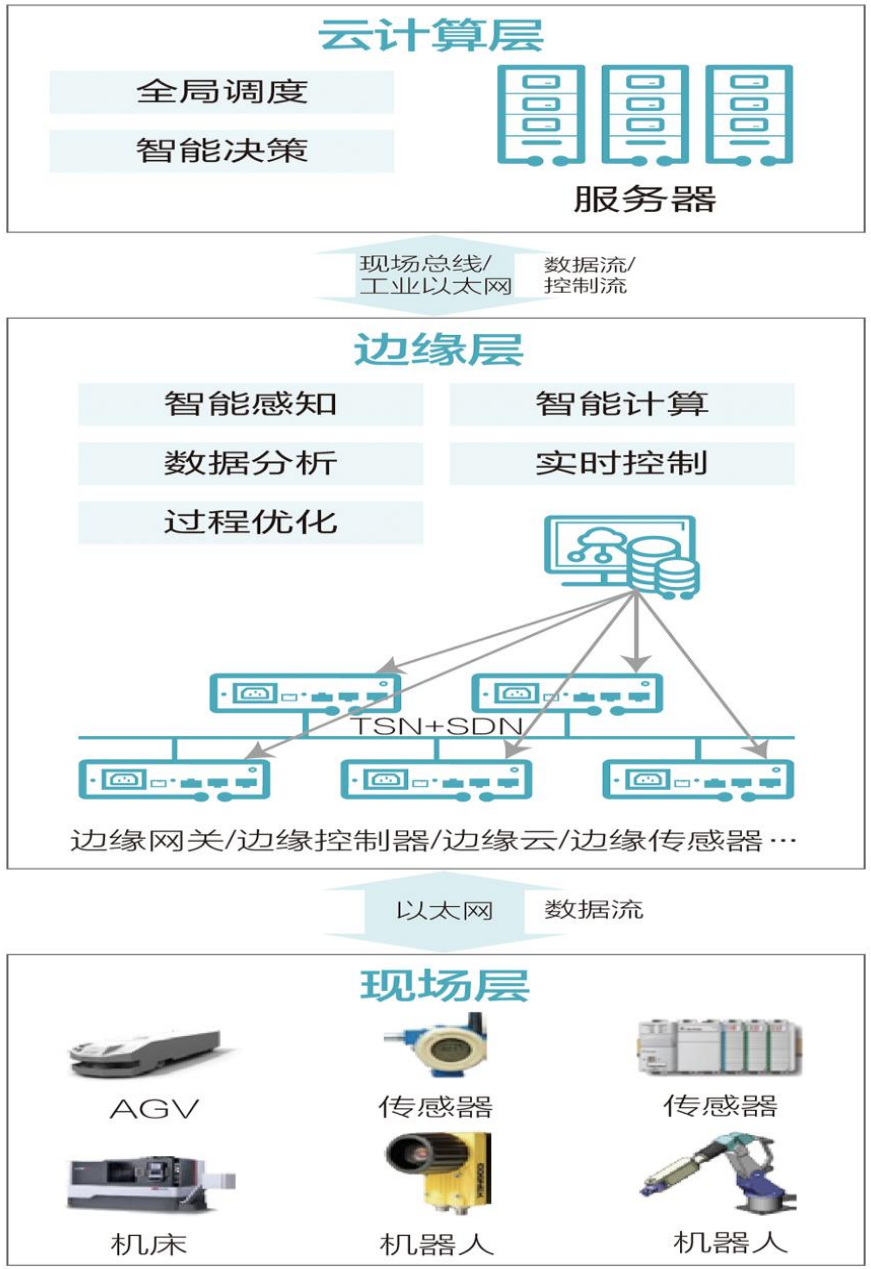
云-边-端协同计算架构

边缘侧与中心云的协同，云中心管理多个边缘节点，云边协同将放大云计算与边缘计算的应用价值：

- 云计算擅长全局性、非实时、长周期的大数据处理与分析，能够在长周期维护、业务决策支撑等领域发挥优势;
- 边缘计算适用于局部性、实时、短周期数据的处理与分析，能更好地支撑本地业务的实时智能化决策与执行;
- 边缘计算既靠近执行单元，也是云端所需数据的采集和初步处理单元，更好地支撑云端应用;
- 云计算通过大数据分析优化输出的业务规则或模型可以下发到边缘侧，边缘计算基于新的业务规则或模型运行。

✓ **架构要点：**

✓ 可以考虑从计算资源丰富程度，数据全局或者局部，算法代价等角度，分配业务功能。反之类似。



2.2 技术架构演进

垂直分层架构
有一定的模块化
成本低，但是二次开发困难

基于中央Bus（ESB）以及
复杂协议实现
软件开发全生命周期管理

细粒度、高密度部署
原子、自治、柔性、可靠性高

- “单体”只是表明系统中主要的过程调用都是进程内调用，不会发生进程间通信，仅此而已。
- 在许多介绍微服务的书籍和技术资料中常常会把这种架构风格称作“巨石系统”，并常常作为架构设计的一个反模式出现。
- 事实上，从实用意义来说，单体架构从来并不是反模式。
- 对于小型系统——即由单台机器就足以支撑其良好运行的系统，单体系统不仅易于开发、易于测试、易于部署，且由于系统中各个功能、模块、方法的调用过程都是进程内调用，因此也是运行效率最高的一种架构风格。

单体系统

- 尽管巨石系统会给人一种不可拆分、难以扩展的隐含意味，但这用来形容单体架构其实有失偏颇：
  - 从分层的角度而言，不管是分布式架构还是单体架构他们都有横向切分（分层架构、六边形架构）以及垂直切分（微服务、业务模块）
  - 从扩展的角度而言，在负载均衡器之后同时部署若干个相同的单体系统副本也可以达到分摊流量压力的效果。
- 单体系统的不足，必须基于特定的场景，需要满足某些非功能性需求，以及应用部署特定要求的前提下才有讨论的价值。

SOA架构

- SOA架构是一种具体地、系统性地解决分布式服务主要问题的架构模式，其核心如下：

• 依靠 SOAP 协议族（WSDL、UDDI 和一大票 WS-*协议）来完成服务的发布、 SOAP	• 利用企业服务总线（ESB）的消息管道来实现各个子系统之间的通信交互 ESB	• 通过业务流程编排（BPM）将各个子系统的功能编排在一起从而实现流程 BPM
• 使用服务数据对象（SDO）来访问和表示数据 SDO	• 使用服务组件架构（SCA）来定义服务封装的形式和服务运行的容器 SCA	• SOA不只是架构风格，更是一套软件设计的基础平台，拥有制定技术标准的组织 Open CSA

SOA架构评价

- SOA架构是一种具体地、系统性地解决分布式服务主要问题的架构模式。
- 系统性指的是 SOA 的宏大理想，它的终极目标是希望总结出一套自上而下的软件研发方法论，希望做到企业只需要跟着 SOA 的思路，就能够一揽子解决掉软件开发过程中的全部问题，譬如该如何挖掘需求、如何将需求分解为业务能力、如何编排已有服务、如何开发测试部署新的功能，等等。
- 不仅关注技术，还关注研发过程中涉及到的需求、管理、流程和组织。
- SOA 在 21 世纪最初的十年里曾经盛行一时，但是成也具体、败也具体，过于严格的规范定义带来了过度的复杂性，而构建在 SOAP 基础之上的 ESB、BPM、SCA、SDO 等诸多上层建筑，进一步加剧了这种复杂性。
- 过于精密的流程和理论使得需要懂得复杂概念的专业人员才能够驾驭SOA架构，从而造成了非常严重的技术壁垒，因此自SOA 诞生的那一天起，就已经注定了它只能是少数系统阳春白雪式的精致奢侈品。
- 它可以实现多个异构大型系统之间的复杂集成交互，却很难作为一种具有广泛普适性的软件架构风格来推广。

SaaS成熟度模型

从可配置性，高性能，可伸缩性三个特性区分的成熟度分级。

- Level1: Ad-hoc / Custom（定制开发）

为每个客户定制一套软件，并为其部署。每个客户使用一个独立的数据库实例和应用服务器实例。数据库中的数据结构和应用的代码可能都根据客户需求做过定制化修改。（多次开发）

- Level2：Configurable（可配置）

通过不同的配置满足不同客户的需求，而不需要为每个客户进行特定定制，以降低定制开发的成本。但是，软件的部署架构没有太大的变化，依然为每个客户独立部署一个运行实例。只是每个运行实例运行的是同一份代码，通过配置的不同来满足不同客户的个性化需求。可配置性的比较通用的实现方式，就是通过MetaData（元数据）来实现。（一次开发多次部署）

- Level3： Multi-Tenant Single-Instance (高性能的多租户架构)

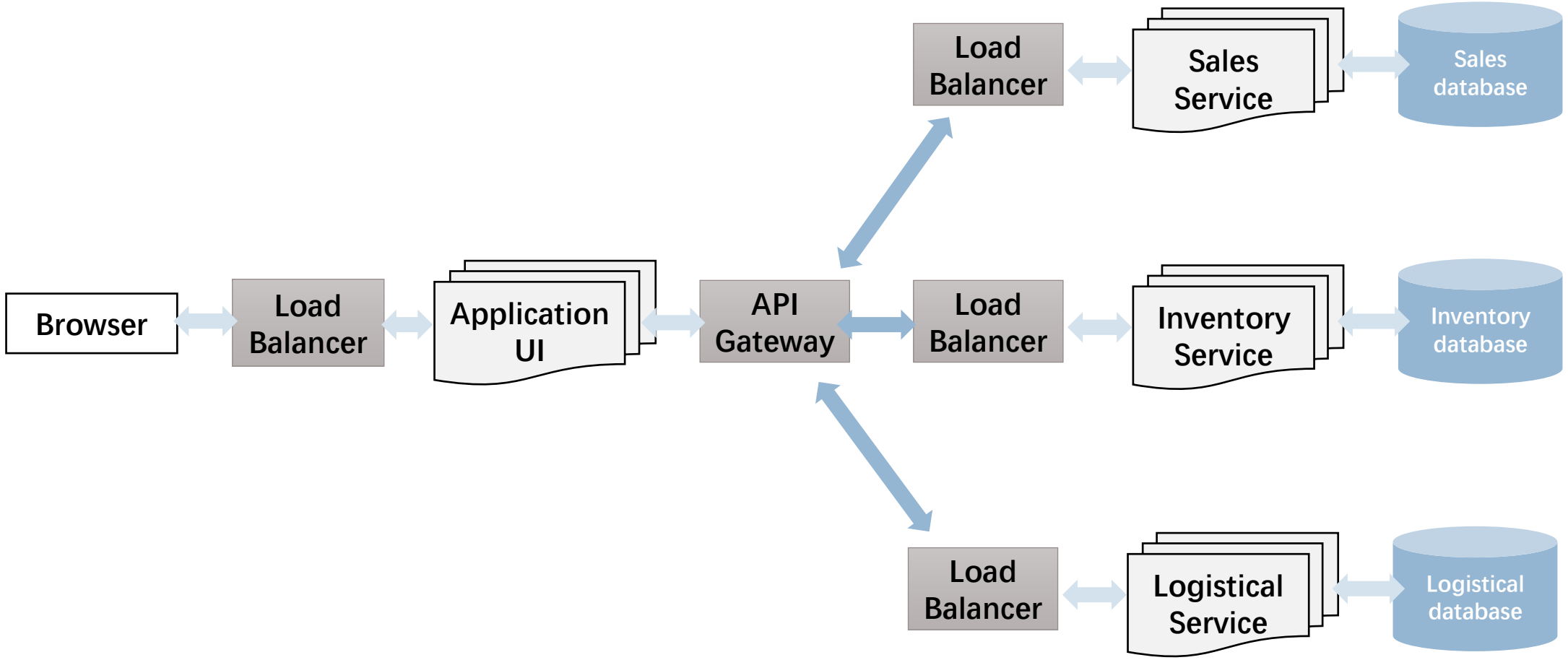
多租户单实例（Multi-Tenant）的应用架构才是通常真正意义上的SaaS应用架构，可以有效降低SaaS应用的硬件及运行维护成本，最大化地发挥SaaS应用的规模效应。（一次开发一次部署）

- Level4： Multi-Tenant Multi-Instance（可伸缩性的多租户架构)

将第三级的系统扩展。最终用户首先通过接入Tenant Load Balance层，再被分配到不同的实例上。通过多实例来分担大量用户的访问，让应用实现近似无限的水平扩展。第四级成熟度模型实现中，最复杂的是对原有单个Instance的数据库服务器，实现其数据的水平拆分。

一种基于微服务架构的实现方式

- Level 4+(Utopia)
- **以负载均衡等方式管理服务端多版本软件的多个实例，对外仍是统一的interface；**



2.3 以业务为中心的微服务架构

- 从SOA技术到SaaS，实现技术的标准化、重用性、松耦合以及互操作，便成为互联网软件发展的重要助力。
- 随着各类企业的业务协同的需要，信息互通互联内容和范围的增加，各种应用的集成需求开始突出。SOA架构没有定义组件粒度，在现实世界容易过度耦合，面对快速变化且不确定用户的需求，显得力不从心。
- 然而开发和运维的隔离，使得开发团队关注软件的需求满足得以交付，运维团队关注性能稳定方便运维，软件变更的影响很大，两个阶段团队难以协同。

软件开发和运维分离的困扰

微服务架构

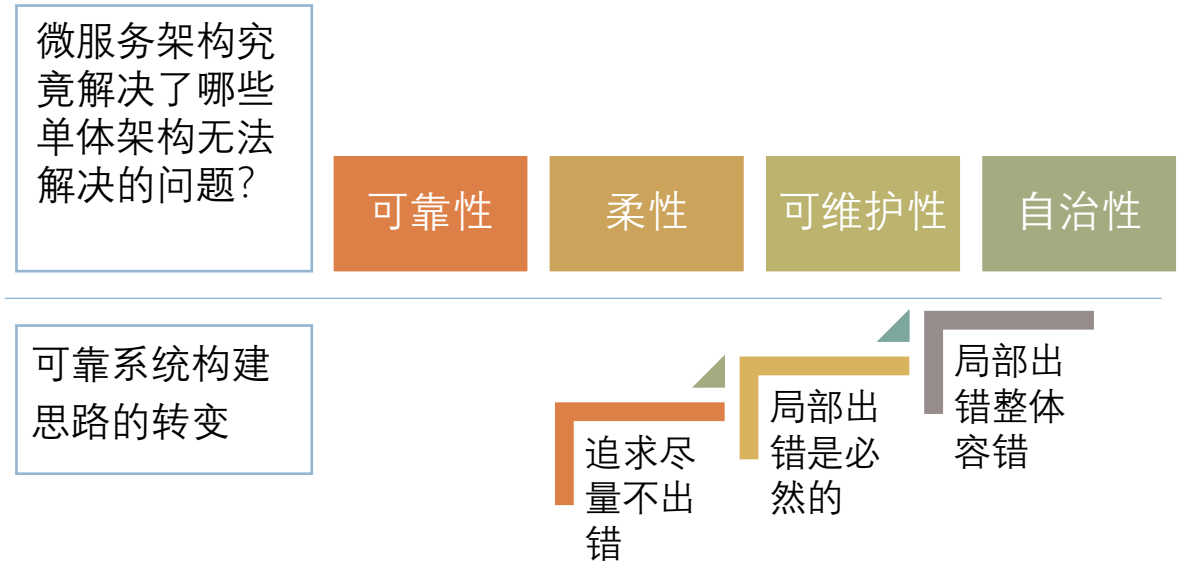
- 微服务是一种通过多个小型服务组合来构建单个应用的架构风格，这些服务围绕业务能力而非特定的技术标准来构建。
  - 各个服务可以采用不同的编程语言，不同的数据存储技术，运行在不同的进程之中。
  - 服务采取轻量级的通信机制和自动化的部署机制实现通信与运维。
- 相较于SOA，微服务追求的是更加自由的架构风格，摒弃了几乎所有 SOA 里可以抛弃的约束和规定，提倡以“实践标准”代替“规范标准”，
  - 实践标准，需要解决什么问题，就引入什么工具；
  - 团队熟悉什么技术，就使用什么框架。
  - 一个简单服务，并不见得就会同时面临分布式中所有的问题，也就没有必要背上 SOA 包罗万象的沉重技术包袱。
- 正是因为没有了统一的规范和约束，以前 SOA 所解决的那些分布式服务的问题都重新出现了，所有的问题在微服务中不再有统一的解决方案，而是百家争鸣。
- 设想：服务间远程调用有多少解决方案，服务发现又有多少解决方案。

微服务架构的特点

- **微服务是一种思想：开发时便考虑到运维需要，运维时实现软件灵活变更。**
- **微服务是一种架构风格**
  - **将大型的复杂业务系统分为多个微服务，微服务以业务为中心，仅关注一件任务，也可以表述为一个小的业务能力；**
  - **微服务间是松耦合的，每个微服务可以被独立部署。**
  - **微服务不是API，也不是SOA，而可以看成服务计算的一种优化实现方式。**

产品化思维	• 开发人员需要关心整个产品的全部方面：需求、设计、开发、运维、运营、反馈，而不是简单地当作一个功能点的开发
强终端弱管道	• 在SOA架构中大量的功能（消息编码、业务规则转换、事务一致性、认证授权）都交给了ESB和ws-*协议去完成，这将太多功能都压在了管道上，而在微服务中提倡让微服务本身去实现对应的功能，管道只负责通信即可
容错性设计	• 承认服务会出错，并且去积极解决出错了该怎么办（检查，隔离，熔断，降级），从而实现局部出错但整体容错
数据去中心化	• 每个服务都有一个全局实体对应的一部分视图，就像盲人摸象，并且只关注和自己业务相关的视图同时各自存储而不是存在同一个地方

- 微服务架构有如下好处：
  - 使大型的复杂应用程序可以持续交付和持续部署。
  - 每个服务都相对较小并容易维护。
  - 服务可以独立部署。
  - 服务可以独立扩展。
  - 微服务架构可以实现团队的自治。
  - 更容易实验和采纳新的技术。
  - 更好的容错性



微服务架构主要特性

- 基于服务的应用组件化
- 围绕业务能力来组织微服务
- 按产品而非项目划分微服务
- 强调智能端点和管道的扁平化
- 分散式管理
- 分散式数据
- 基础设施自动化
- 基于故障的设计
- 演进的设计

微服务架构的实现要点：

- 按业务分解服务;
- 通过资源规划和容器编排实现自动化部署;
- 服务通讯和管理:
- 数据去中心化:
- 服务发现和负载均衡;
- 分段升级;
- 安全互联及权限认证等

通过结合容器Docker等技术，微服务架构实现开发、测试、生产环境的统一。

从单体结构到微服务架构的变化

单体架构	微服务架构
整体部署	拆分部署
紧耦合	松耦合
基于整个系统的扩展	基于独立服务，按需扩展
集中式管理	分布式管理
应用无依赖关系管理	微服务间较强的依赖关系管理
局部修改，整体更新	局部修改，局部更新
故障全局性	故障隔离，非全局
代码不易理解，难维护	代码易于理解维护
开发效率低	开发效率高
资源利用率低	资源利用率高
重，慢	轻，快

微服务架构的使用要点

微服务架构带来的问题主要在于：

- 开发人员技能要求高
- 接口的数目和匹配问题
- 分布式系统复杂性
- 可测性的挑战
- 异步机制
- 代码重复
- 运维开销及成本增加

因此，微服务只有当系统复杂度持续增加，业务对技术的支撑需求和预期加强时，考虑微服务才是合理的。

目录

- 1 软件的三种开发模式
  - 定制开发-套件实施-模型驱动架构
- 2 持续演化的软件技术架构
- 3 软件的部署运维模式
- 4 基于流程的软件过程
  - 开发实施过程-工具-角色职责
- 5 流程自动化RPA技术
- 6 小结

3 服务应用部署模式

将服务应用部署到生产环境时有以下四种主要的部署模式：

- 使用特定编程语言的发布包格式的模式部署服务
- 使用将服务部署为虚拟机的模式部署服务
- 使用将服务部署为容器的模式部署服务
- 使用Serverless部署模式部署服务

（1）编程语言特定的发布包格式

- 部署服务的第一种方式是使用特定于编程语言的软件包部署服务；
- 使用此方式时，生产环境中部署的内容，以及服务运行时管理的内容，都是特定语言发布包中的服务。
- 对于不同的编程语言，发布包的格式不同，例如：
  - Java语言的场景下，发布包的格式是JAR文件或WAR文件
  - Node.js的场景下，发布包的格式是源代码和模块的目录
  - GoLang的场景下，发布包的格式是特定于操作系统某个路径的可执行文件

编程语言特定的发布包格式

- 将服务作为特定于编程语言的发布包进行部署有以下好处：
  - 部署快速：只需要将服务复制到主机并启动
  - 资源的高效利用：多个服务实例共享机器及其操作系统
- 尽管有上述好处，但是该部署模式有如下几个显著的缺点：
  - 缺乏对技术栈的封装
  - 无法约束服务实例消耗的资源
  - 同一台主机上运行多个服务实例时缺少隔离
  - 无法自动判定放置服务实例的位置

（2）将服务部署为虚拟机

- 部署服务的第二种方式是将服务部署为虚拟机，使用此方式时会将服务打包成虚拟机镜像然后部署到生产环境中，每个服务实例都是一个虚拟机。
- 将服务部署为虚拟机有以下好处：
  - 虚拟机镜像封装了技术栈
  - 服务实例之间互相隔离
  - 可以使用成熟的云计算基础设施
- 尽管有上述好处，但是该部署模式有如下几个显著的缺点：
  - 资源利用效率较低
  - 部署速度相对较慢
  - 系统管理需要额外的开销

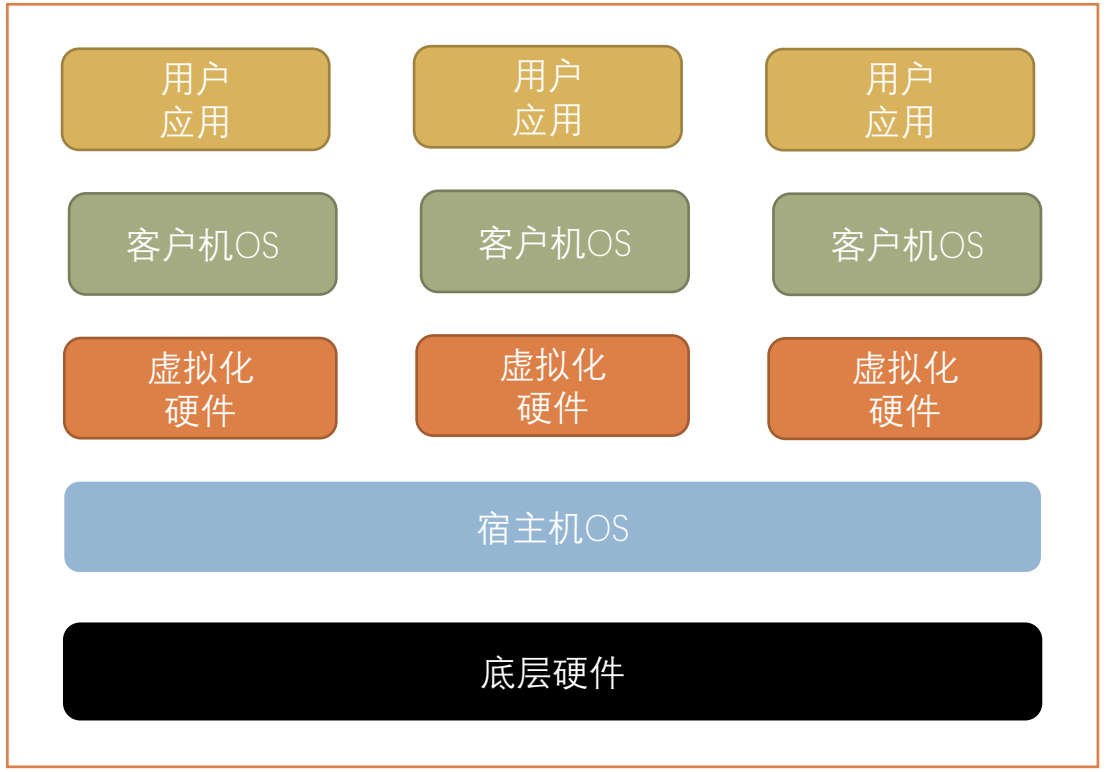
（3）将服务部署为容器

- 部署服务的第三种方式是将服务部署为容器，使用此方式时会将服务打包成容器镜像，然后基于Kubernetes容器编排框架将服务部署到生产环境中，每个服务实例都是一个容器。
- 将服务部署为容器有以下好处：
  - 容器镜像封装了技术栈
  - 容器服务实例是隔离的
  - 容器服务实例的资源受到限制
  - 容器镜像可以快速地构建且体积较小
  - 容器服务实例可以快速地启动
- 尽管有上述好处，但是该部署模式有如下缺点：
  - 开发者需要承担容器镜像的管理工作
  - 开发者需要承担容器运行所需要的底层容器编排框架的管理工作

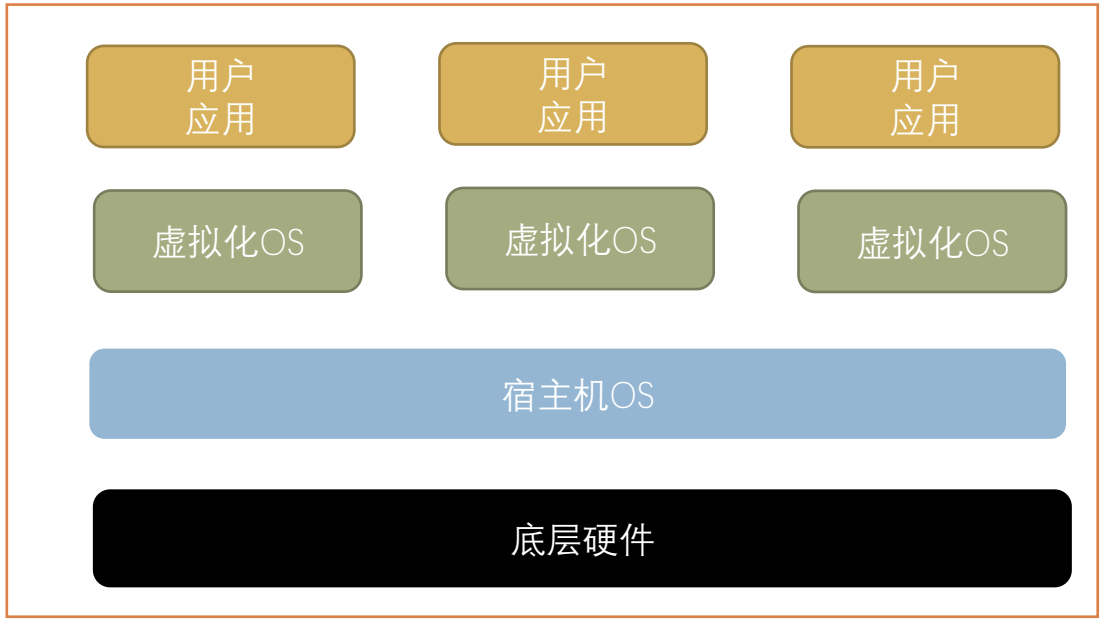
虚拟机及容器比较

- 相较于虚拟机的硬件虚拟化机制，容器是一种更现代、更轻量级的操作系统虚拟化机制，容器可以看作是一个沙箱，其中运行了一个或多个进程，沙箱将它们与其他容器中的进程隔离。

虚拟机（硬件虚拟化）



容器（操作系统虚拟化）



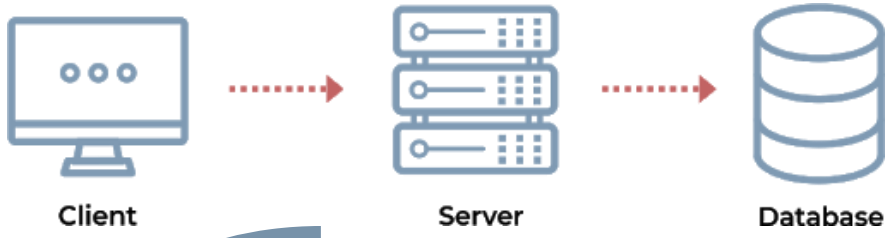
（4）Serverless部署模式

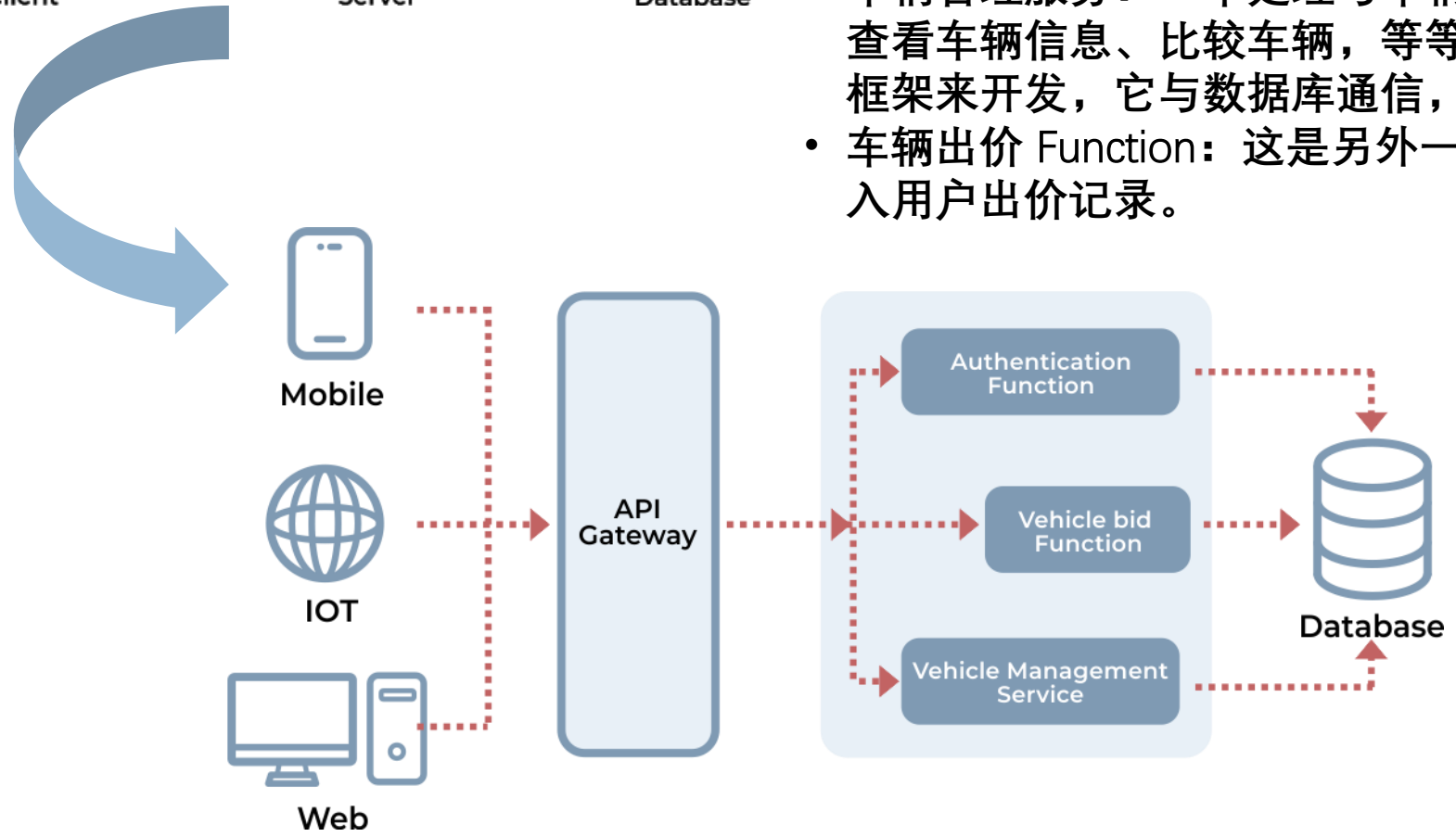
- 虽然上述的三种部署模式不同，但是它们还是具有一些共同的特征：
  - 用户必须预先准备一些计算资源，例如物理机、虚拟机或容器，即使它们处于闲置状态，用户也总是需要为某些虚拟机或容器付费；
  - 用户必须负责系统管理，无论运行什么类型的计算资源，用户必须承担为操作系统、软件以及容器编排框架升级或打补丁的工作。
- Serverless使得用户不再需要预先准备计算资源也不再需要系统管理。
- Serverless将底层基础架构从开发人员中分离出来，实现了底层资源的透明化，基本上虚拟化了运行时和运营管理，使得用户执行给定的任务而不必担心服务器、虚拟机或底层计算资源的配置。

Serverless部署模式

- Serverless的全称是Serverless computing——无服务器运算，又被称为函数即服务（Function-as-a-Service，FaaS），它是云计算的一种模型。
- Serverless以平台即服务（PaaS）为基础提供了一个事件驱动型架构，终端客户不需要部署、配置或管理服务器服务，终端客户仅仅需要上传符合指定接口定义的代码给Serverless平台，如：AWS Lambda。
- Serverless平台在有事件发生时，如：HTTP请求、定时执行，Web服务请求，自动运行充足的微服务实例来处理对应的事件，根据所花费的时间和消耗的内存，用户仅仅需要为处理事件所需要的资源付费，而无需为任何服务器或者管理付费。
- 从基础架构角度看，它有不同的抽象层，如物理机、虚拟机和容器，开发人员可以和这些抽象层进行互动，而不用担心服务器的基础架构或者管理。当触发代码的预定义事件发生时， Serverless平台执行任务。

Serverless和微服务的结合





原来的单体应用程序被拆分成了多个服务器端组件：

- API 网关：所有服务的入口点和反向代理。来自客户端的请求会先到达网关，网关根据路由规则将请求重定向到特定的服务。
- 认证 Function：这是一个用于管理用户认证（登录）的 Function（Function，FaaS）。
- 车辆管理服务：一个处理与车辆相关操作的微服务，如列出车辆、查看车辆信息、比较车辆，等等。这个服务可以使用任意的语言或框架来开发，它与数据库通信，并且独立运行。
- 车辆出价 Function：这是另外一个 Function，也与数据库通信，录入用户出价记录。

Serverless部署模式

- 使用Serverless部署模式有以下好处：
  - **敏捷：**由于开发人员在使用服务器时不部署、管理或扩展服务器，因此组织可以放弃基础设施管理，用户不需要再负责底层的系统管理，从而可以专注于开发应用程序，这极大地减少了操作开销。
  - **弹性：**Serverless平台本身会运行处理事件及负载所需要的适合的实例数，用户无需预测所需容量。升级和添加计算资源不再依赖于运维团队，Serverless应用程序可以快速、无缝地自动扩展，以适应流量峰值；反之，当并发用户数量减少时这些应用程序也会自动缩小规模。
  - **基于使用情况定价：**与典型的IaaS云按分钟或小时收费不同，Serverless平台或者说FaaS仅仅收取处理每个事件所消耗的资源的费用。
  - **安全：** Serverless架构提供了安全保障。由于不再管理服务器，DDoS 攻击的威胁性要小得多，而且无服务器功能的自动扩展功能有助于降低此类攻击的风险
  - **应用性强：**无服务器与微服务架构高度兼容，这也带来了显著的好处。

Serverless讨论

由于Serverless平台会动态运行用户的代码，因此它需要花费时间来配置应用程序实例和启动应用程序，并且其本质上是基于有限事件与请求的编程模型，不适用于部署长时间运行的批处理服务。

因此Serverless部署模式往往有如下缺点：

- 启动延迟；
- 厂商锁定，对服务器缺乏控制；
- 性能优化局限于代码内部；
- 执行时间限制（AWS Lambda 的执行时间限制为 15 分钟）；
- 成本不可预测；
- 开发环境和生产环境不一样；
- 测试和调试更为复杂。

目录

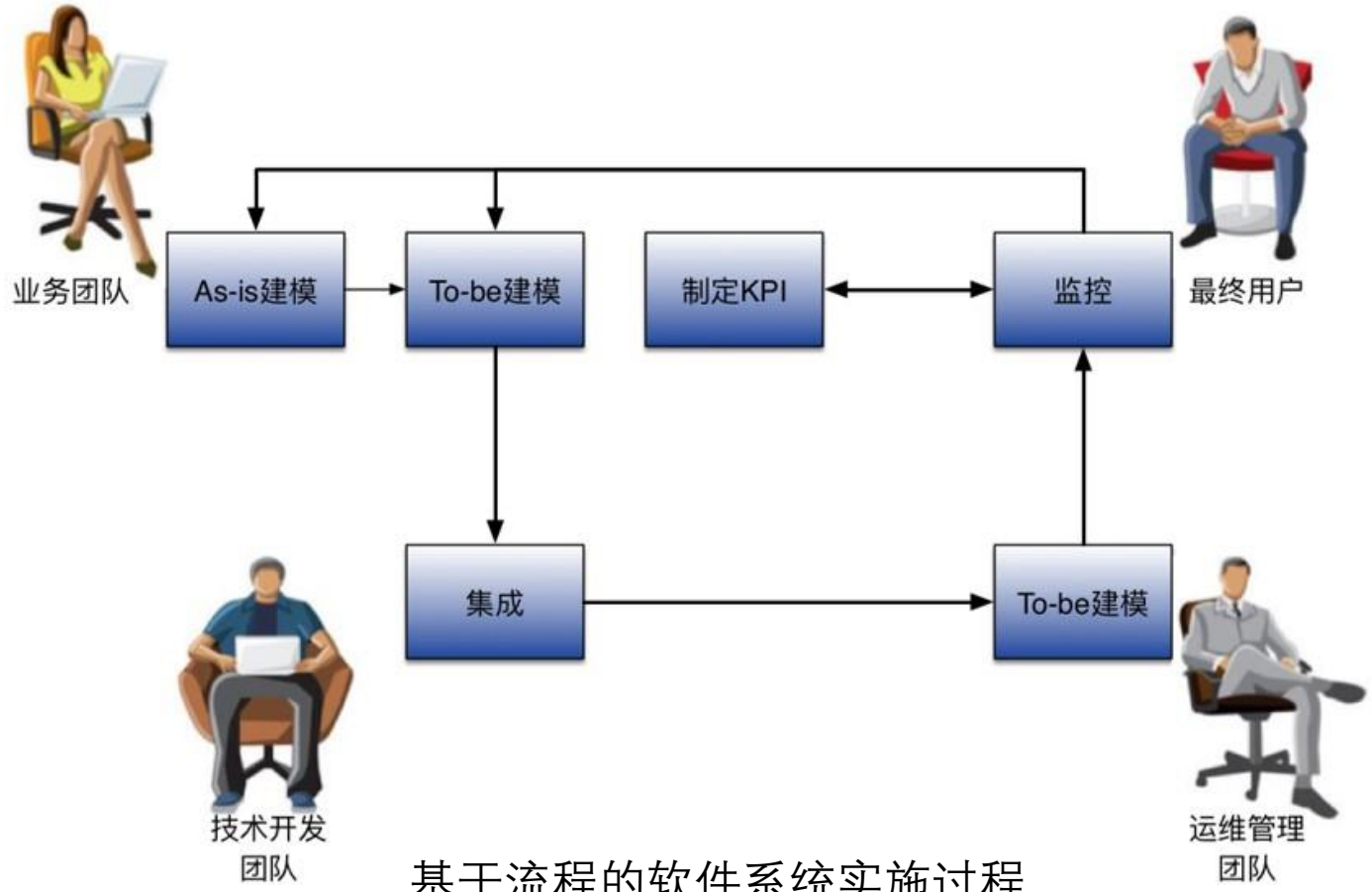
- 1 软件的三种开发模式
  - 定制开发-套件实施-模型驱动架构
- 2 持续演化的软件技术架构
- 3 软件的部署运维模式
- 4 基于流程的软件过程
  - 开发实施过程-工具-角色职责
- 5 流程自动化RPA技术
- 6 小结

4 基于流程的软件过程

- 互联网环境下的软件过程体现出覆盖全生命周期的持续集成框架；
- 虽然业务模型包括数据、组织、规则、流程等部分。但互联网环境下，随着软件系统逐渐重点关注以用户使用过程为核心的业务过程，流程驱动的软件系统实施方法重要性越发突出。
- 从流程为核心，给出了基于流程的软件开发及实施运维过程的主要阶段以及相应的角色。
  - 包括信息建模、技术架构、系统开发及测试、系统部署、运维管理、项目管理等部分；
  - 各主要阶段相应的角色。

4.1 基于流程的软件实施过程

- 流程类软件的系统开发实施，其核心是围绕业务流程生命周期的模型构建及转换,也是基于业务流程的管理活动，也提供了软件实施步骤和任务划分的基础。
- 涉及四大群体。



基于流程的软件系统实施过程

流程实施的四个阶段

（1）流程设计及建模

- 流程设计包括已有流程的识别和未来流程的设计，设计流程的关注领域包括了标准操作流程、服务水平协议SLA、任务交接机制等的设计。将已识别和定义业务流程的过程设想实现为可视化的流程模型。

（2）基于流程的服务集成

- 主要在流程基础上面向系统的开发和配置。因此往往不限于业务流程的建模和转换，为使得模型可执行，可能涉及到页面、数据、服务等要素的技术要素的集成。

（3）系统部署及执行

- 流程执行涉及到流程模型的自动化实现。
- 自动化流程的一个方法就是开发或者购买一个工作流引擎或者流程服务器，开发相关应用程序，执行流程中所需任务。实现业务流程自动化执行的关键主要包括流程标准，业务规则，服务互操作。

（4）系统监控及优化

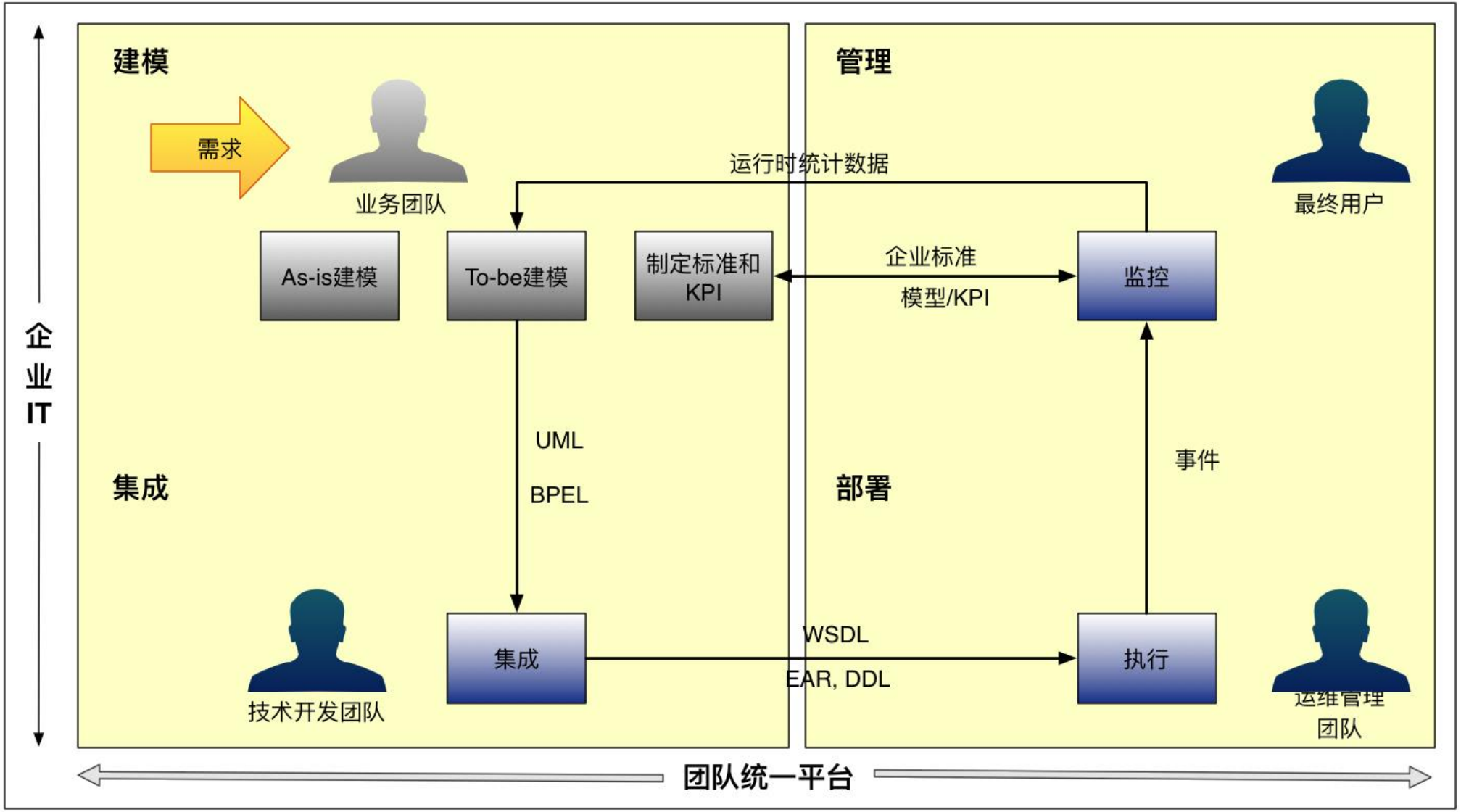
- 流程监控包括跟踪单个流程以获取它们的状态信息,也可以获取多个流程的性能统计。
- 流程优化主要从建模或监控阶段获取流程性能信息，识别潜在或实际的瓶颈，节省成本的潜在可能，在流程设计时应用这些信息，作为持续改进依据。

业务流程管理生命周期的工具支持

基于WebSphere套件支持业务流程的管理

- 具体包含了四个主要组件，对应四个阶段
- 阶段 1: 建模组件
  - 采用WebSphere Business Modeler收集需求，设计，建模，仿真和优化业务模型，作为软件或服务的分析及设计基础。
- 阶段 2: 装配组件
  - 在Modeler创建的流程模型基础上，采用WebSphere Integration Developer建立模型和组件、资源的映射关联，实现系统配置。
- 阶段 3: 部署组件
  - 采用WebSphere Process Server对所有基于SOA的流程自动化。
- 阶段 4: 管理组件
  - 采用WebSphere Business Monitor实现流程运行的动态管理。

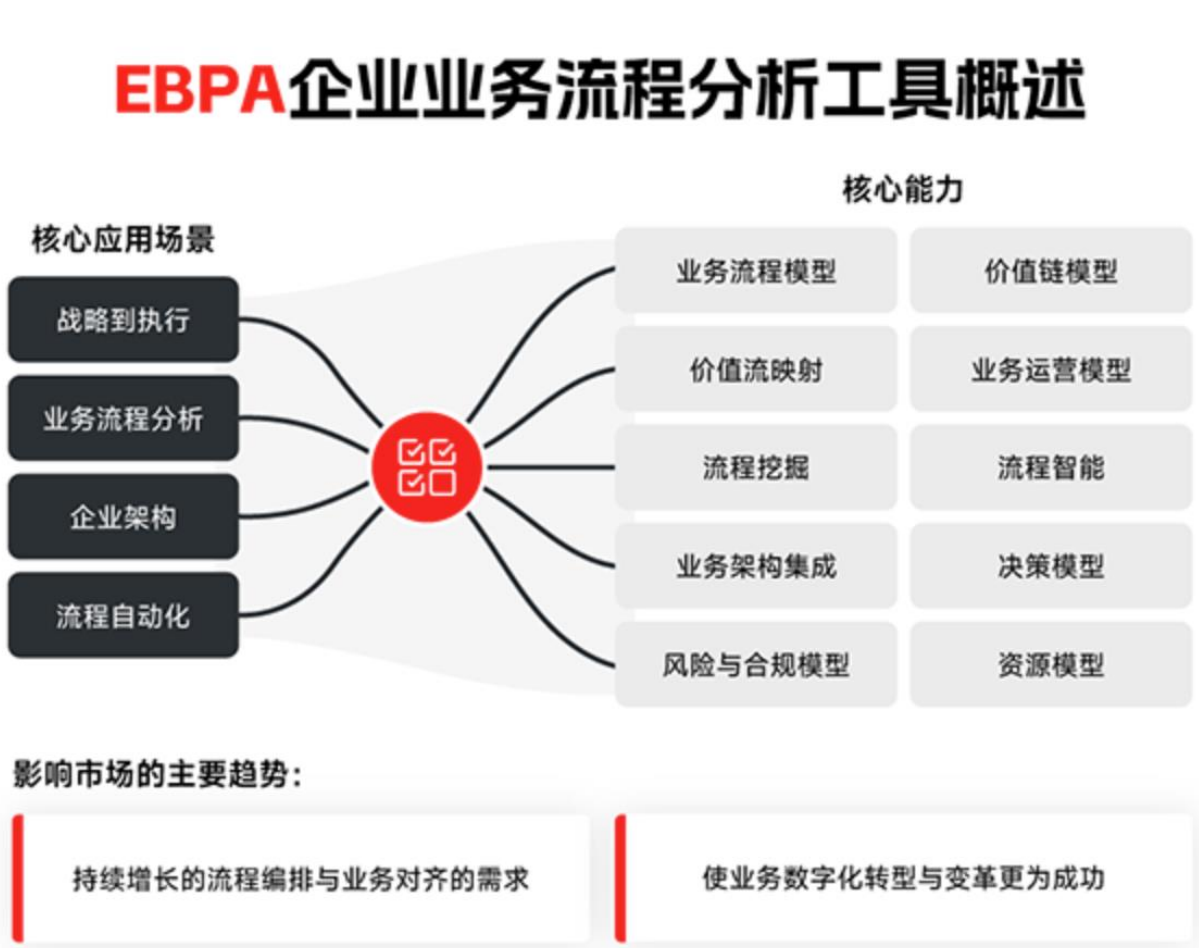
流程应用各阶段的模型传递



面向不同团队的信息建模、系统集成、软件部署、监控管理及优化等交互过程，包括模型交互以及任务接口的划分和交接。

4.2 BPM工具

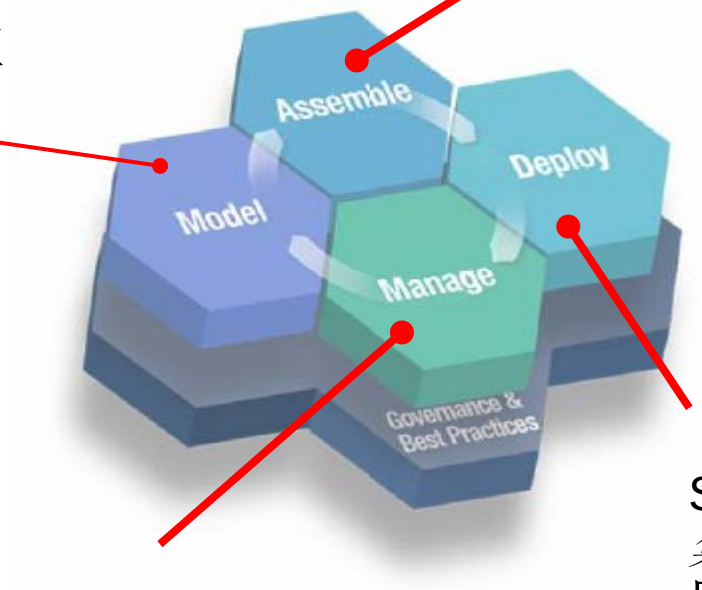
- 业务流程管理（BPM）工具是一种软件应用程序，用于帮助组织管理、优化和自动化其业务流程。它提供了一种集成的平台，使组织能够设计、模拟、执行、监控和优化其业务流程。



WebSphere系列业务流程工具

Stage 2 — **WebSphere Integration Developer**
发现,组装,测试
基于标准的开发环境

Stage 1 — **WebSphere Business Modeler**
收集需求
设计，建模，仿真和优化业务模型



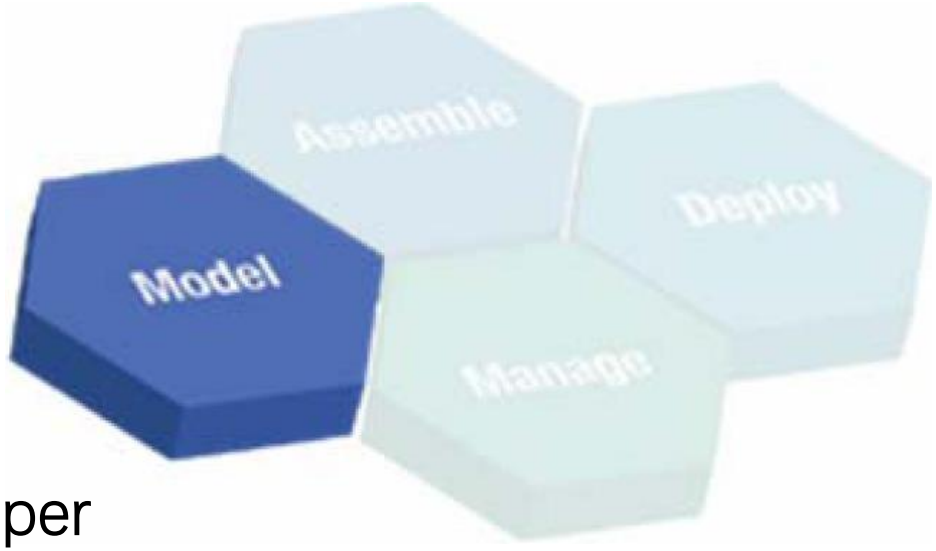
Stage 3 — **WebSphere Process Server**
集成部署策略和工作流
用系统自动化步骤和信息流来支持人力密集型流程步骤

Stage 4 — **WebSphere Business Monitor**
业务监控用以协同活动分析和流程优化
业务流程的实时可见性,
以实现流程干预和持续改进

阶段 1: 建模

- 记录在案的，可审计的模型
  - 创建新的业务流程模型并将流程记录
  - 符合规范的要求
- 流程的仿真和分析
  - 对当前流程模型仿真市场变化以降低风险
    - 评估成本，资源和周期对模型的影响
  - 在投入生产前预测风险
- 定义关键性能指标
  - 测量是管理的关键
  - 有效利用技术资源
- 导出文件用于WebSphere Integration Developer
  - 可靠的实施转移
    - 拉近 IT 和业务用户间的差距

WebSphere Business Modeler



阶段 2: 装配

- 创建流程模型，建立模型和组件、资源的映射关联
  - 业务流程执行语言(BPEL) 描述来自Modeler的模型
    - BPEL是一种描述商业活动的抽象高级语言
- 开发 SOA 组件
  - 将组件和服务组合到复合应用程序中，为WebSphere Process Server 的执行
  - Dynamic BPEL processes for flexibility and responsiveness based on runtime values
- 用于使生产率最大化的，易于使用的工具
  - 一个集成框架: Eclipse
- 简化开发
  - 重用组件和资源

WebSphere Integration Developer

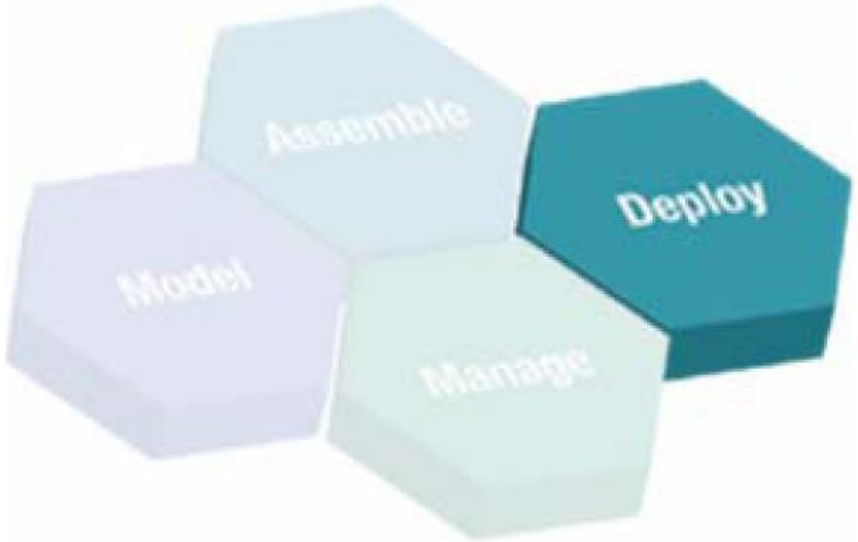
阶段 3: 部署

集成运行时

- 对所有基于SOA的流程自动化
  - 重用已有服务
  - 创建未来可重用的新服务
- 可靠的、可扩展的、安全的、开放的标准

变更时，迅速改变流程行为以跟上业务需求

- 业务规则：灵活的流程
  - 动态控制流程执行
- 状态机：即时行为改变
- 接口映射和关系
- 支持流程集成的所有方面
  - 流程编排
  - 人工任务：动态资源的使用
  - 服务：构建流程流而不用担心信息资源的细节
  - 业务对象：实时流程数据
  - 配适器：将人，应用程序，信息结合到一起
  - 公共事件基础设施：监控活动的心跳

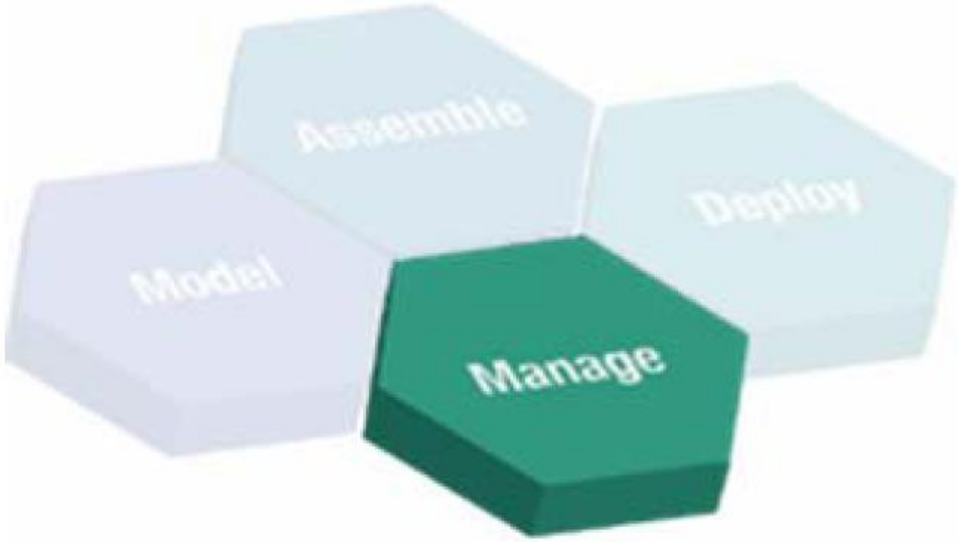


WebSphere Process Server

阶段 4: 管理

- **持续流程改进**
  - 实时监控流程实例
  - 实时数据导出给Modeler，实现闭环
- **查看实时性能**
  - 关键性能指标记分卡
  - 跟踪成本、时间和资源
  - 负载均衡
  - 从portlets创建Web页面: dashboards
  - 业务活动监控以一个dashboard展示一个Web页面
  - 无需编码，修改dashboards
- **干预部署的流程**
  - 设置情境触发器和通知
  - 动态响应这些警报

WebSphere Business Monitor



4.3 软件开发相关角色及职责

阶段	任务	定制开发模式角色	套件实施模式角色	业务驱动架构角色
业务规划阶段	确定系统的作用域、确保项目可行性、制定进度表和资源分配计划，并进行项目其余部分的预算	业务分析员	业务顾问	业务顾问 产品经理
分析阶段	了解新系统的商业需求和处理要求并做好文档	业务分析员	业务顾问	业务咨询师 系统架构师
设计阶段	根据分析阶段的需求定义和制定的决策，设计好设计方案	系统设计人员	技术顾问	系统架构师
实施阶段	监理、测试和安装可靠的工作软件系统，培训用户并使其收益于系统的使用	系统开发人员 系统测试人员	技术顾问	系统实施人员 系统测试人员
运行维护阶段	保持系统的有效运行	运维人员	运维人员	运维人员
应用		最终用户	最终用户	最终用户

各类角色及职责

- 业务类人员
  - 业务咨询师，在整个软件系统开发过程的工作，便是开展分析企业业务，进行业务建模，建立系统开发及实施的业务蓝图。
- 开发类人员
  - 系统架构师的主要职责是在业务模型的基础上，完成业务转换，实现业务架构的技术解决方案。
  - 系统开发人员主要采用业务工程开展软件系统开发和实施的可以更快的、更简便有效的实现软件系统。
  - 系统测试人员往往也是项目开发人员，但随着互联网的发展，系统测试往往在系统运行中体现，系统测试的职责变得更为复杂。因此，除了传统的功能、性能测试人员之外，系统运维人员从系统日志，业务人员从反馈数据都可以加入系统的测试验证人员行列。
- 运维类人员
  - 系统运维人员在软件系统运行中开展管理及维护，是系统的管理者。具体可分为网络管理、数据库管理、信息安全管理等细分角色，以保证软件系统的稳定运行。

软件管理类角色

项目经理

- 项目经理是为项目的成功策划和执行负总责的人。项目经理是项目团队的领导者，项目经理首要职责是在预算范围内按时优质地领导项目小组完成全部项目工作内容，并使客户满意。

产品经理

- 产品经理的核心是软件产品，产品经理的核心任务是在产品规划的框架下，实现需求和技术的有效对接，保证软件产品和需求的一致性。实现产品战略规划、产品需求分析、产品技术实现。

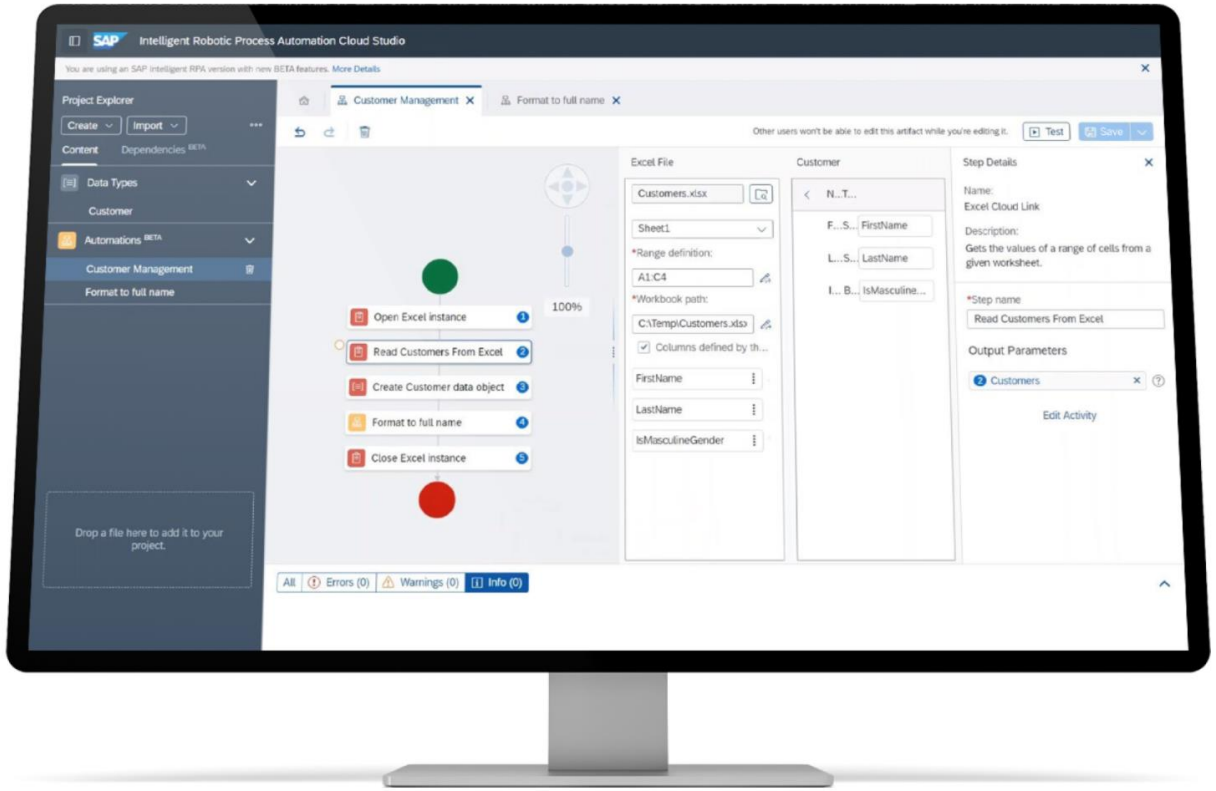
目录

- 1 软件的三种开发模式
  - 定制开发-套件实施-模型驱动架构
- 2 持续演化的软件技术架构
- 3 软件的部署运维模式
- 4 基于流程的软件过程
  - 开发实施过程-工具-角色职责
- 5 流程自动化RPA技术
- 6 小结

5 RPA技术概述

- RPA (Robotic Process Automation)技术指的是应用软件机器人实现过程自动化。
- RPA创建并部署具有启动和操作其他软件能力的软件机器人，可以轻松编程以跨应用程序执行基本重复任务的软件。
- RPA往往应用在一些相对基本任务的自动化，这些系统在各种应用程序中运行，就像人类工人一样。

- 软件或机器人可以通过多个步骤和应用程序学习工作流，
  例如获取收到的表单、发送接收消息、检查表单的完整性、将表单归档到文件夹中以及使用表单名称、归档日期更新电子表格等。
- RPA软件旨在减轻员工完成重复、简单任务的负担。



RPA 技术框架

RPA通过非侵入的方式模拟人工操作。不会侵入或影响已有的软件系统，具有良好的嵌入性，且对非技术的业务人员友好。

- 设计器

  主要用于设计和编写RPA应用流程，类似于传统软件开发用到的IDE。在设计器中还可以进行代码调试、应用发布、导入第三方包或库等操作。

- 控制台（Robot）

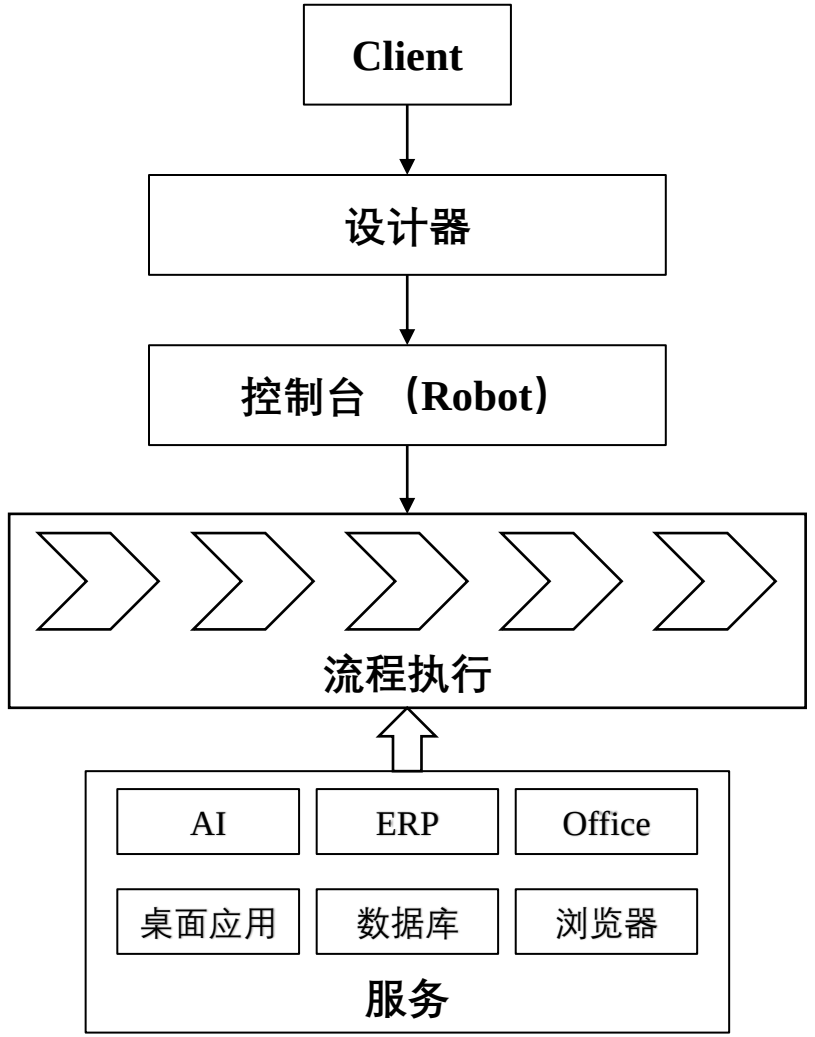
  用于执行设计器编排好的RPA应用流程，它按照既定步骤完成某个流程的自动执行，执行期间无须人为干预。

- 流程执行

  RPA流程是一系列封装好的应用组件和代码的集合。它规范了机器人的执行步骤和操作范围，确保机器人执行流程过程中的准确性。

- 服务

  提供各类流程执行需要的后台服务和应用组件，如AI服务、ERP服务、桌面应用等。



RPA使用特点

Manual & Repetitive

Consolidate and manipulate data from multiple data sources, e.g. MS Excel, vendor portal, SAP systems

High Volume

Handling process steps hundreds of times a day, like copy & paste e.g. data migrations, approvals

Multiple Systems

Access multiple applications during process execution, e.g. web application, ERP, non-SAP systems

- Capital Expenditure Approvals
- Invoice Approvals
- Account Opening
- Relocation
- Visa Permits
- HR Employee Self Services
- Managing Contingent Workers
- Master Data Record Update
- Document Approval
- Lead Generation Workflows
- Return Order Process
- Incident Management
- Procurement Data Collection

理解RPA

- 机器人过程自动化（RPA）主要用于帮助完成办公室类型的功能，这些功能通常需要能够按特定顺序完成多种类型的任务。
- 它创建并部署了一个能够启动和操作其他软件的软件机器人。
- 从某种意义上说，其基本概念与传统的制造自动化类似，后者侧重于完成工作流的一部分，甚至是一项任务，并创建一个专门从事这项工作的机器人。
- 办公室工作通常需要同样的重复工作，但由于它是跨平台和应用程序操作数据的，因此不需要物理机器人。
- 虽然公司经常寻求自动化来简化流程和降低劳动力成本，但在某些情况下，自动化出现了问题。

RPA领域

改进商务操作

- 消除流程中的人工、枯燥、容易出错的任务
- 调整资源到高价值的任务
- 允许并行执行和即时扩展
- 避免长时间以及昂贵的开发

缩减处理时间

- 改进流程的整体性能
- 节省成本
- 缩减流程执行时间

改善服务性能

- 减少人工错误
- 改善用户体验
- 通过机器人不间断的运行以获得效率和速度
- 调动人员投向有价值的错误校核任务

改善一致性

- 提升分析能力通过文档化路径分析
- 导入验证以及自动化报告功能

构造人工智能入口

- 理解非结构化数据
- 导入自处理机器人
- 异常处理历史
- 导入新模式和创新方法
- 避免遗留问题

RPA的优势

- 与深度学习不同，机器人过程自动化中使用的软件机器人是由员工在程序员的帮助下编程完成特定工作流中的任务。该软件不会自行学习，也不会寻求调整新的效率或新的见解，如大数据分析或AI软件。
- 相反，RPA的工作方式就像员工的数字助手，它清除了占用每个上班族一天一部分时间的繁重、简单的任务。
- 因此，RPA是一种比人工智能驱动的系统或企业软件更简单的产品，该系统或企业软件旨在将所有数据带入平台。这也使得它比AI或大数据系统相对便宜。
- 这种简单性和相对便宜性可以使RPA成为许多公司更具吸引力的解决方案，特别是如果公司有遗留系统的话。RPA旨在与大多数遗留应用程序兼容，与其他企业自动化解决方案相比更易于实施。

典型RPA平台产品

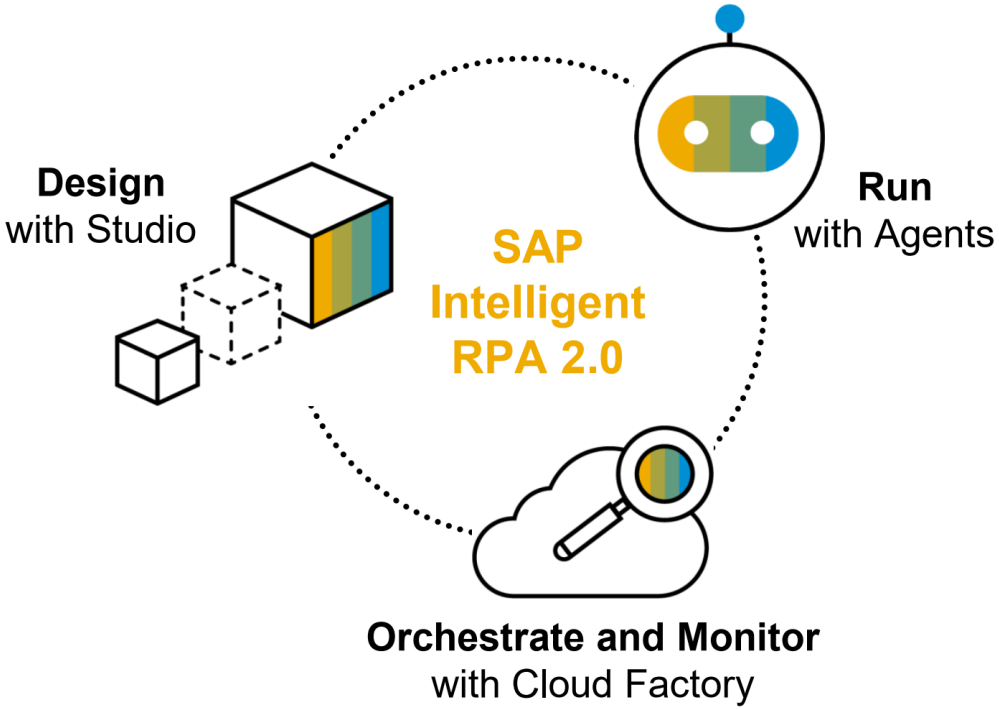
SAP Intelligent Robotic Process Automation

SAP Intelligent RPA Cloud Studio

SAP Intelligent RPA Cloud Studio web应用程序提供了一种低代码和无代码的方法，使您能够设计机器人并自动化应用程序。

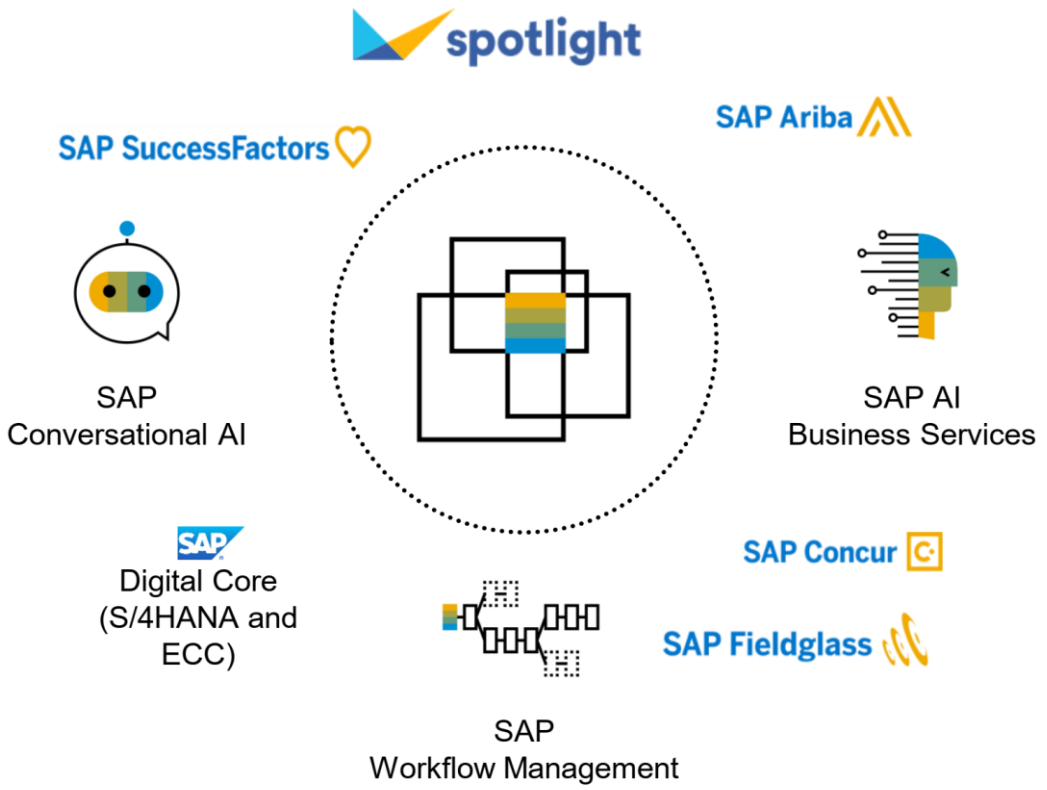
通过低代码方法，开发人员和业务用户可以使用Javascript手工编码知识构建和运行应用程序机器人。

通过无代码的方法，用户可以通过一个直观的界面和一个简单的工作流设计进行指导。cloud studio中提供的易于使用的机器人构建功能允许专家开发人员和业务用户构建和编辑云项目，而无需编写一行代码。



基于工作流的应用集成

- 通过SAP专用连接器简化UI集成
- 使用SAP和第三方应用程序自动化数据收集
- 在SAP Intelligent RPA bot Store上提供150多个免费预建机器人，缩短自动化时间
- 通过与SAP应用程序集成以加速机器人开发
- 使用嵌入式人工智能和智能服务自动化处理非结构化信息
- 与SAP 工作流管理平台集成
- 与SAP AI Business Services和SAP Conversational AI实现机器学习和人工智能结合使用
- 使用技术连接器实现安全的应用程序驱动



RPA技术的应用前景

- 随着合规性和监管备案要求的提高，金融业银行、保险公司和投资管理公司已成为RPA的早期采用者。
- 许多繁重的后台功能，如确保提交最新的了解客户表格或在贷款申请中包含最近的信用检查，都是RPA的理想选择。从员工身上消除这一负担使他们能够专注于高回报的任务。
- 更重要的是，软件可以比人类更快地清除这些基本的归档和数据操作功能，从而缩短总体处理时间。
- 当然，RPA不仅仅局限于金融领域的使用。任何处理数据和归档的行业都可以从机器人过程自动化中获益。
- 当软件能够在不需要繁重复杂的实现的情况下降低成本并提高效率时，它将在几乎任何行业找到用户和有用的应用程序。

目录

- 1 软件的三种开发模式
  - 定制开发-套件实施-模型驱动架构
- 2 持续演化的软件技术架构
- 3 软件的部署运维模式
- 4 基于流程的软件过程
  - 开发实施过程-工具-角色职责
- 5 流程自动化RPA技术
- 6 小结

5 小结

- 从软件中的两个核心问题（业务IT对齐，开发运维分离）出发，讨论了软件技术架构中的现状和趋势。
- 从业务IT对齐出发，软件开发模式的定制开发-套件实施-模型驱动架构比较分析；
- 从开发运维结合出发，阐述了从单体架构到SOA架构再到微服务架构的软件演进历程，以及各个架构的优缺点，并重点阐述了微服务架构的主要特点。
- 阐述了面向软件开发实施的流程全生命周期过程，相关的工具支撑，以及相关主要角色的职责和任务。
- 面向互联网应用，阐述了服务平台的要点和构架思路；