服务架构的进化之路

就像生产力和生产关系一样，生产力达到了某一个水平，生产关系才会顺其自然发生改变来满足新的生产力的要求。同样，在程序的开发过程中，服务架构的演变是一个随着业务规模的增长，技术的发展和用户需求变化而不断推进的过程，大致经历了单体架构、垂直架构、分布式架构、微服务架构、云原生架构以及服务网格架构等阶段。所有的服务架构并没有绝对的好坏之分，符合需求的架构才是最好的架构。

接下来结合背景和当时的需求与技术条件，我来大致介绍一下每个架构的特点

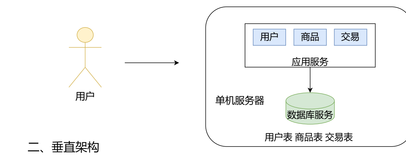
一、单体架构

背景：快速将开发的系统投入市场进行检验，迅速响应变化要求，前期用户访问量很少

特点：将应用程序的所有功能模块打包成一个独立的单元进行部署和运行

优势：开发简单，部署方便，所有代码都在一个项目中，便于管理和维护

局限性：随着业务的发展，代码量不断增加，项目变得庞大而复杂，难以维护和扩展；迭代缓慢，一个小的改动都要重新部署整个应用



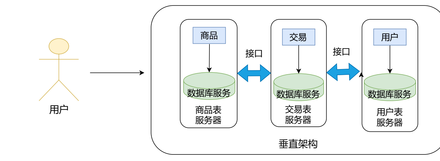
二、垂直架构

背景：用户量需求大幅提升，单体服务架构性能不足，需要团队分工

特点：按照功能将单体拆分成多个独立的应用，每个应用负责特定的业务领域如用户服务，订单服务，商品服务

优势：前后端分离独立，数据库读写分离，扩展性提高，开发效率提高，降低了各个部分之间的关联程度

局限性：调用关系变得复杂，增加了开发和维护难度，数据同步



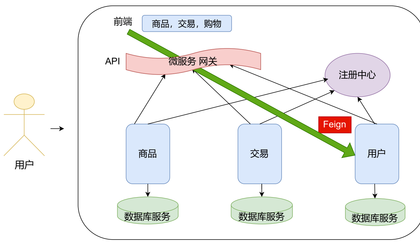
三、微服务（大拆小）

背景：业务日趋复杂，需求变化频繁，快速迭代，系统维护成本高昂

特点：将应用程序拆分成一组小型的，自治的服务，每个服务都独立运行，通过HTTP、RPC等进行交互，每个小单位都可以独立交付迭代

优势：灵活性高，扩展性高；各个服务之间联合度低，非常适合团队协作，没有严格规范，提高开发效率

局限性：架构复杂，从纯软件层面解决架构问题系统的复杂性进一步提高，对开发人员要求更高，需要掌握更多技术和工具



注册中心：组件之间交互时不能通过传统的紧耦合的方式，必须要通过轻量的接口方式调用。在注册中心找到接口地址，再进行调用

Feign:方便用户对发起调用，做了一层包装和代理

微服务网关：提供一个统一的HTTP出口地址供前端使用

API网关：弥补微服务流量请求代理低，流量请求代理力度加大

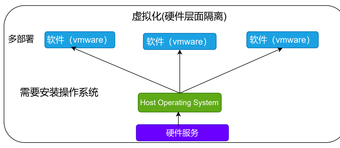
四、云原生架构

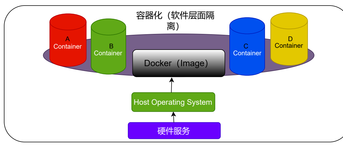
背景：虚拟化，容器化技术长足发展，Kubernetes赢得容器战争

特点：软硬一体，合力应对架构问题

优势：通过容器使服务的部署和运维变得简单

流程：应用和服务可以打包为Docker镜像，通过Kubernetes来发布和部署镜像。Docker理解为能运行应用和服务的最小的操作系统，里面放着运行代码。把整个"操作系统"打包为镜像后就可以分发到需要部署相关服务的服务器上，直接启动Docker镜像就可以运行



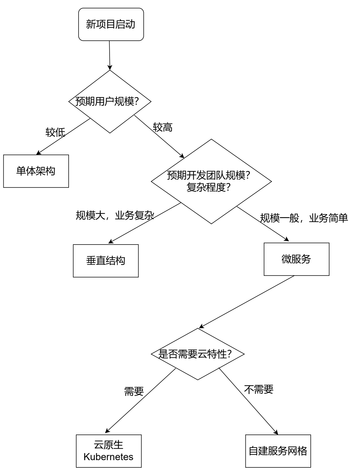


五、补充

SOA：昙花一现，在单体系统的缺陷在大型项目中变得越来越突出时，SOA自治能力强，隔离性强，技术异构能力强的特点得到开发人员注意，但因为规范过于严格，不接地气，最终曲高和寡，不得民心而被微服务取代。

无服务：存在一些无状态的应用，并不需要专门的服务器，多数Web资讯类网站，小程序，公共API服务。但响应的性能不太好，不适合信息管理系统，网络游戏等应用。如果说微服务架构时分布式系统这条路的极致，那无服务架构，也许就是'不分布式'的云端系统这条路的起点

总结一下架构的选择树



最后，引用周志明老师《凤凰架构》中一句话做结尾：

软件开发的未来不会只存在某一种"最先进的"架构风格，多种具针对性的架构风格并存，是软件产业更有生命力的形态。