

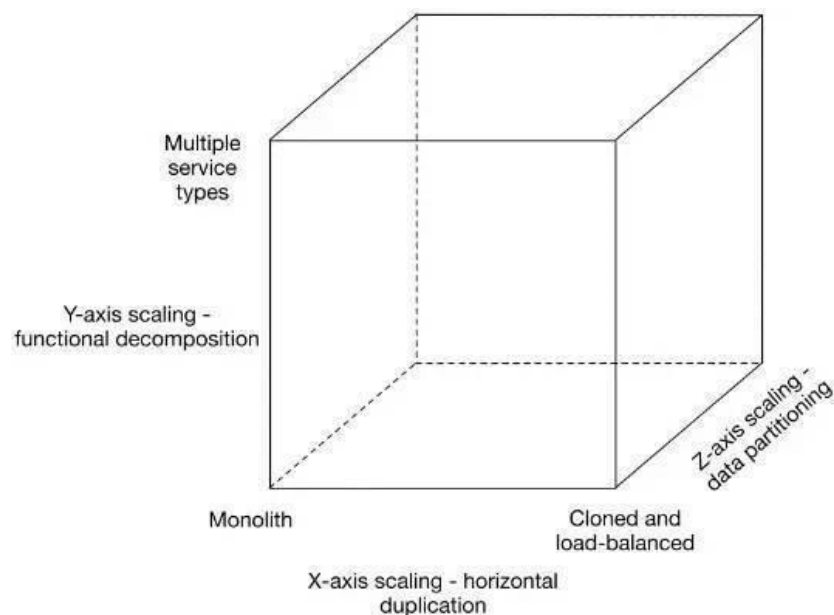
# 第03期-单体应用&SOA架构&微服务

2019年11月18日 13:08

目的：高性能、可用性、伸缩性、可扩展和安全性。

## 应用规模庞大时如何拆分？

当一个应用非常庞大时，是加 Web 服务器？买更高配置的数据库服务器？还是分库分表？这些都没问题，也就是常说的三维扩展模型，从这三个角度去分析问题，将大问题转换成小问题，分治思想。



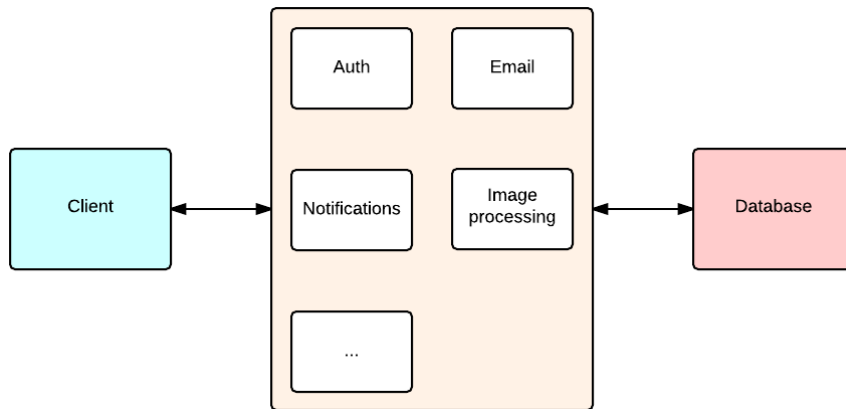
x轴，水平拆分，针对单个可部署的应用或子系统，增加服务器，使用负载均衡器，例如：web园  
y轴，功能分解，针对不同的业务模块进行拆分，例如：订单管理模块和产品管理模块  
z轴，数据分割，分库分表，对同一业务进行数据结构分离存储，然后负载，提高性能。

## 设计模式、框架、分层、架构

设计模式指导写出便于维护的代码，指导框架设计，框架提供开发方式和基础架子，分层便于开发人员维护和分工，架构是一整套解决方案的系统工程，架构主要解决每个系统如何运作。例如三层，MVC框架，微服务架构。

## 单体应用架构

所有功能都部署在一个web容器中运行的系统就叫做单体架构，这是一种比较传统的架构风格。



优点：开发简单，容易测试，直接打包部署简单，复用是基于面向对象的接口类。

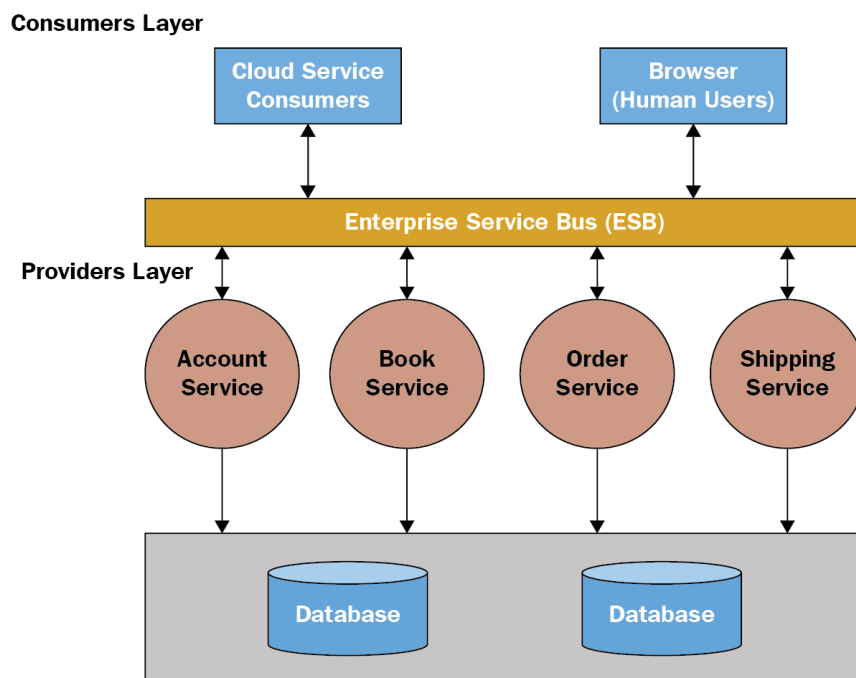
缺点：编译时间长，回归测试周期长，部署速度慢，开发效率低，边界模糊，扩展性差，不易修改，技术栈受限。

## SOA 面向服务的架构

面向服务的架构（SOA）是一个被滥用的词汇，不同人有不同的理解。不过作为一个通用标准，SOA 意味着需要从结构上将应用尽可能解耦成多个服务（通常为 HTTP 服务），这些服务要能区分成不同的类型，如子系统或分层，还可实现异构系统的隔离，最终使企业总线对外提供服务。

通俗点来讲，SOA提倡将不同应用程序的业务功能封装成服务并宿主起来，通常以接口和契约的形式暴露并提供给外界应用访问（通过交换消息），达到不同系统可重用的目的。

SOA是一个组件模型，它能将不同的服务通过定义良好的接口和契约联系起来，服务是SOA的基石。



SOA体系下，服务之间通过企业服务总线（Enterprise Service Bus）通信，许多业务逻辑在中间层（消息的路由、转换和组织）。

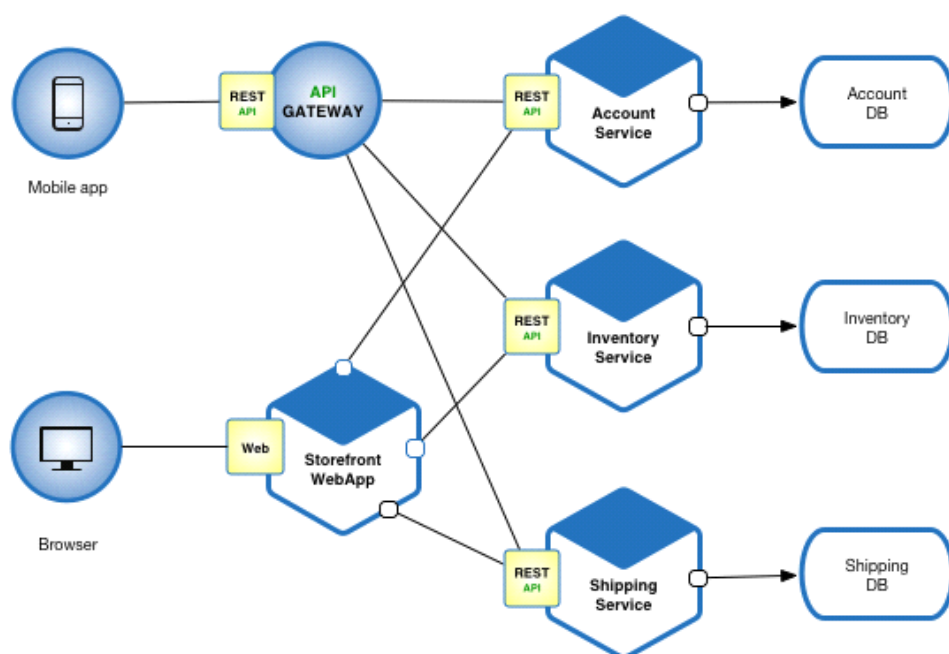
SOA的web服务一般采用文本消息，基于复杂的消息格式（SOAP）和消息定义（xsd），使用 WSDL 对服务进行描述。

框架：.NET Remoting、webService 和 WCF

## 微服务架构

微服务是从 SOA 派生而来的，但 SOA 与微服务是不同的架构，去中心化的思想，业务逻辑被拆分成一系列小而松散耦合的分布式组件，每个组件都被称为微服务，而每个微服务都在整体架构中执行着单独的任务，或负责单独的功能，每个微服务可能会被一个或多个其他微服务调用，每个微服务可能有自己的数据库。

但 SOA 与微服务是不同的架构。例如大型中心代理、企业级中心编排引擎和企业服务总线（ESB）都是典型的 SOA 功能。在微服务社区中，大多数情况下这都是 SOA 的反模式设计。实际上，有争议认为微服务架构只是正确实现 SOA 之后的产物。



优点：细粒度，单独开发部署，多团队协作，降低复杂度，数据存储隔离，可替换，技术栈不受限。

缺点：运维要求较高 DevOps、分布式固有的复杂性、接口调整成本高。

微服务之间的通信：同步消息 REST 接口，异步消息 AMQP、STOMP 和 MQTT 分布式队列。

消息格式：JSON、XML、Thrift、ProtoBuf 和 Avro。

服务描述：Swagger、RAML 和 Thrift IDL

常见方式：点对点方式 – 直接调用服务、API-网关方式和 消息代理方式。

## 微服务到底能有多小（微）？

在开发微服务时，大小不应该是重点，重点是要创建低耦合的服务，以便能独立地开发、部署和扩展每个服务。当然在确定和设计微服务时，只要它们之间没有太多直接依赖，就应该使它们尽可能地小。比大小更重要的还有必要的内

部一致性和对其他服务的依赖。

## 微服务和 SOA 的区别

微服务架构强调的第一个重点就是业务系统需要彻底的组件化和服务化。微服务不再强调传统 SOA 架构里面比较重的 ESB 企业服务总线，同时 SOA 的思想进入到单个业务系统内部实现真正的组件化。

微服务相比于 SOA 更加精细，微服务更多的以独立的进程的方式存在，互相之间并无影响；微服务提供的接口方式更加通用化，例如 HTTP RESTful 方式，各种终端都可以调用，无关语言、平台限制；微服务更倾向于分布式去中心化的部署方式，在互联网业务场景下更适合。

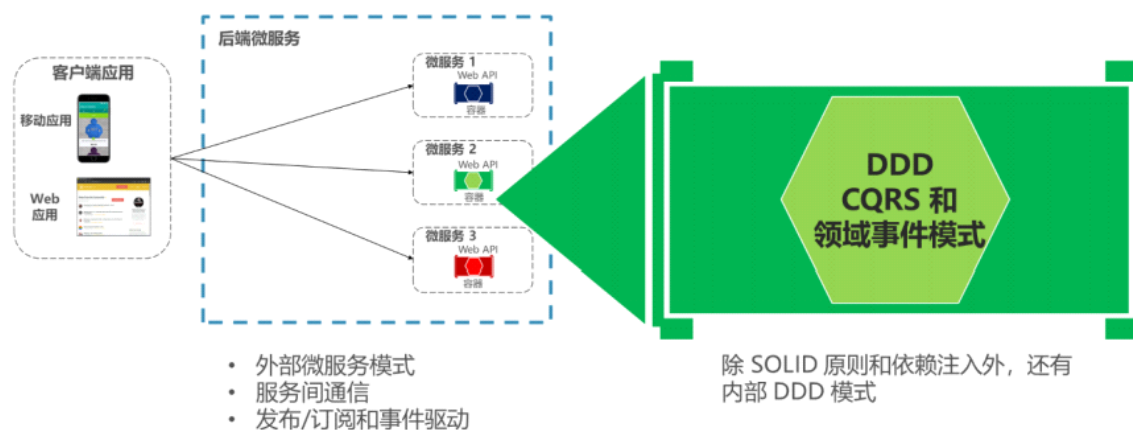
### SOA和微服务架构的区别？

## DDD与微服务

DDD是建模方法，微服务是一种架构方式，前者强调问题领域分析方法，后者重在技术架构。

### 运用四色建模法进行领域分析

如何确定边界是设计和定义微服务的关键任务，DDD 模式可以帮助我们理解领域的复杂性。对于每个限界上下文中的领域模型，识别和定义用来模型化该领域的实体、值对象和聚合。构建并提炼在一个定义上下文边界里的领域模型。很明显这是微服务的形式。那些边界内组件最终会成为微服务，即使在某些情况下，一个 BC 或业务微服务可以由多个物理的服务组成。DDD 和微服务都是关于边界的。



## 分布式与集群的区别

分布式：业务模块分别部署到不同的机器，通过API Gateway 实现模块的负载均衡，分布式需要做好事务管理。  
集群：不同服务器部署同一套服务对外访问，实现服务的负载均衡，集群模式需要做好 session 共享。

## 分布式是否属于微服务？

答案是肯定的，微服务的意思也就是将模块拆分成一个独立的服务单元通过接口来实现数据的交互，微服务与分布式的细微差别是，微服务的应用不一定是分散在多个服务器上，他也可以是同一个服务器的不同容器里。

## 单体架构过时了吗？

当然不是，单体应用由于代码集中化，所以开发快速、管理简单，在一个应用功能不多代码不太复杂的时候，没有拆分项目的必要，此时用单体架构简单高效成本也更低。

## 微服务为什么一定要用 docker 容器吗？

物理机、虚拟机、容器三者的优缺点比较后发现，容器最适合做微服务，因为启动快，治理性好，又开源，所以定制化就多了，其实容器并非 docker 一家，但 docker 又是这么一个相对来说比较好的容器实现，加之 docker 趋势大增，所以 docker 就比较广泛应用了，应用的人越多就会更加普及，趋势是这样，但未来结果谁知道呢。

## 架构总结

设计原则千万条，高内聚低耦合第一条，架构设计不规范，开发运维两行泪！架构要考虑需求，切合实际场景分析，切勿过度设计，切勿设计不足，每家公司需求都不同，架构有方法论，有经验可谈，且勿天马行空，切勿为了技术而技术，为了分层而架构。