链滴

多研究些架构，少谈些框架

作者：bnvnvnv

原文链接：https://ld246.com/article/1531124963504

来源网站：链滴

2017-6-9 曹祖鹏 github.com/JoeCao/JoeCao.github.io/issues/3

微服务架构和SOA区别

微服务现在辣么火，业界流行的对比的却都是所谓的Monolithic单体应用，而大量的系统在十几年前是已经是分布式系统了，那么微服务作为新的理念和原来的分布式系统，或者说SOA（面向服务架构是什么区别呢？

我们先看相同点：

● 需要Registry，实现动态的服务注册发现机制；

● 需要考虑分布式下面的事务一致性，CAP原则下，两段式提交不能保证性能，事务补偿机制需要考；

● 同步调用还是异步消息传递，如何保证消息可靠性？SOA由ESB来集成所有的消息；

● 都需要统一的Gateway来汇聚、编排接口，实现统一认证机制，对外提供APP使用的RESTful接口；

● 同样的要关注如何再分布式下定位系统问题，如何做日志跟踪，就像我们电信领域做了十几年的信跟踪的功能；

那么差别在哪？

● 是持续集成、持续部署？对于CI、CD（持续集成、持续部署），这本身和敏捷、DevOps是交织在起的，我认为这更倾向于软件工程的领域而不是微服务技术本身；

● 使用不同的通讯协议是不是区别？微服务的标杆通讯协议是RESTful，而传统的SOA一般是SOAP，过目前来说采用轻量级的RPC框架Dubbo、Thrift、gRPC非常多，在Spring Cloud中也有Feign框架标准RESTful转为代码的API这种仿RPC的行为，这些通讯协议不应该是区分微服务架构和SOA的核心别；

● 是流行的基于容器框架还是虚拟机为主？Docker和虚拟机还是物理机都是架构实现的一种方式，不核心区别；

微服务架构的精髓在切分

● 服务的切分上有比较大的区别，SOA原本是以一种“集成”技术出现的，很多技术方案是将原有企内部服务封装为一个独立进程，这样新的业务开发就可重用这些服务，这些服务很可能是类似供应链CRM这样的非常大的颗粒；而微服务这个“微”，就说明了他在切分上有讲究，不妥协。无数的案例明，如果你的切分是错误的，那么你得不到微服务承诺的“低耦合、升级不影响、可靠性高”之类的势，而会比使用Monolithic有更多的麻烦。

● 不拆分存储的微服务是伪服务：在实践中，我们常常见到一种架构，后端存储是全部和在一个数据中，仅仅把前端的业务逻辑拆分到不同的服务进程中，本质上和一个Monolithic一样，只是把模块之的进程内调用改为进程间调用，这种切分不可取，违反了分布式第一原则，模块耦合没有解决，性能受到了影响。

分布式设计第一原则 — “不要分布你的对象”

● 微服务的“Micro”这个词并不是越小越好，而是相对SOA那种粗粒度的服务，我们需要更小更合的粒度，这种Micro不是无限制的小。

如果我们将两路（同步）通信与小/微服务结合使用，并根据比如“1个类=1个服务”的原则，那么我实际上回到了使用Corba、J2EE和分布式对象的20世纪90年代。遗憾的是，新生代的开发人员没有使分布式对象的经验，因此也就没有认识到这个主意多么糟糕，他们正试图重复历史，只是这次使用了技术，比如用HTTP取代了RMI或IIOP。

微服务和Domain Driven Design

一个简单的图书管理系统肯定无需微服务架构。既然采用了微服务架构，那么面对的问题空间必然是较宏大，比如整个电商、CRM。

如何拆解服务呢?

使用什么样的方法拆解服务？业界流行1个类=1个服务、1个方法=1个服务、2 Pizza团队、2周能重完成等方法，但是这些都缺乏实施基础。我们必须从一些软件设计方法中寻找，面向对象和设计模式用的问题空间是一个模块，而函数式编程的理念更多的是在代码层面的微观上起作用。

Eric Evans 的《领域驱动设计》这本书对微服务架构有很大借鉴意义，这本书提出了一个能将一个大题空间拆解分为领域和实体之间的关系和行为的技术。目前来说，这是一个最合理的解决拆分问题的案，透过限界上下文（Bounded Context，下文简称为BC）这个概念，我们能将实现细节封装起来让BC都能够实现SRP（单一职责）原则。而每个微服务正是BC在实际世界的物理映射，符合BC思路微服务互相独立松耦合。

微服务架构是一件好事，逼着大家关注设计软件的合理性，如果原来在Monolithic中领域分析、面向象设计做不好，换微服务会把这个问题成倍的放大。

以电商中的订单和商品两个领域举例，按照DDD拆解，他们应该是两个独立的限界上下文，但是订单肯定是包含商品的，如果贸然拆为两个BC，查询、调用关系就耦合在一起了，甚至有了麻烦的分布式务的问题，这个关联如何拆解？BC理论认为在不同的BC中，即使是一个术语，他的关注点也不一样在商品BC中，关注的是属性、规格、详情等等（实际上商品BC这个领域有价格、库存、促销等等，他作为单独一个BC也是不合理的，这里为了简化例子，大家先认为商品BC就是商品基础信息），而订单BC中更关注商品的库存、价格。所以在实际编码设计中，订单服务往往将关注的商品名称、价格等属性冗余在订单中，这个设计解脱了和商品BC的强关联，两个BC可以独立提供服务，独立数据存储

小结

微服务架构首先要关注的不是RPC/ServiceDiscovery/Circuit Breaker这些概念，也不是Eureka/Docer/SpringCloud/Zipkin这些技术框架，而是服务的边界、职责划分，划分错误就会陷入大量的服务的相互调用和分布式事务中，这种情况微服务带来的不是便利而是麻烦。

DDD给我们带来了合理的划分手段，但是DDD的概念众多，晦涩难以理解，如何抓住重点，合理的用到微服务架构中呢?

我认为如下的几个架构思想是重中之重

- 充血模型
- 事件驱动

之后将为大家详细介绍这两个设计思路。!