24 | 微服务架构该如何落地?

2018-10-16 胡忠想

24 | 微服务架构该如何落地?
朗读人：胡忠想　11′08″ | 4.47M

专栏前面的文章我给你讲解了微服务架构的各个组成部分，以及实践过程中可能遇到的问题和对应的解决方案，到这里你应该对微服务架构有了一个完整的认识。那么在实际项目中，如何让一个团队把我们所学的微服务架构落地呢?

今天我就结合自己的经验，定位在中小规模团队，谈谈微服务架构到底该如何落地。

组建合适的技术团队

经过我前面的讲解，你应该认识到微服务架构相比于单体应用来说复杂度提升了很多，这其中涉及很多组件，比如注册中心、配置中心、RPC 框架、监控系统、追踪系统、服务治理等，每个组件都需要专门的人甚至专家把控才能 hold 住，不然微服务架构的落地就相当于空中楼阁，虚无缥缈。

所以想要落地微服务，首先需要合适的人，也就是组建一支合适的技术团队。你一定很容易想到，是不是只有架构师适合做微服务架构的开发？一定程度上，这是合理的，因为微服务架构所涉及的具体技术，比如 CAP 理论、底层网络可靠性保证、Netty 高并发框架等，都对技术的深

度要求比较高，一般有经验的架构师才能掌握，所以这个技术团队必须包含技术能力很强的架构师。但是还要考虑到微服务架构最后还是要落地到业务当中，既要满足业务的需求，也要防止一种情况的发生，那就是全部由架构人员组成技术团队，根据自己的设想，脱离了实际的业务场景，最后开发出来的架构中看不中用，业务无法实际落地，既打击了团队人员积极性，又对业务没有实际价值，劳民伤财。所以这支技术团队，也必须包含做业务懂业务的开发人员，只有他们了解业务的实际痛点以及落地过程中的难点，这样才能保证最后设计出的微服务架构是贴合业务实际的，并且最后是能够实际落地的。

从一个案例入手

当你的团队决定要对业务进行微服务架构改造时，要避免一上来就妄想将整个业务进行服务化拆分、追求完美。这种想法是很危险的，一切的技术改造都应当以给业务创造价值为宗旨，所以业务的稳定性要放在第一位，切忌好高骛远。

正确的方法是首先从众多业务中找到一个小的业务进行试点，前期的技术方案以满足这个小的业务需求为准，力求先把这个小业务的微服务架构落地实施，从中发现各种问题并予以解决，然后才可以继续考虑更大规模的推广。这样的话，即使微服务架构的改造因为技术方案不成熟，对业务造成了影响，也只是局限在一个小的业务之中，不会对整体业务造成太大影响。否则的话，如果因为微服务架构的改造给业务带来灾难性的后果，在许多技术团队的决策者来看，可能微服务架构的所带来的种种好处也不足以抵消其带来的风险，最后整个微服务架构的改造可能就夭折了。

回想一下微博业务的微服务改造，从 2013 年开始进行微服务架构的研发，到 2014 年用户关系服务开始进行微服务改造，再到 2015 年 Feed 业务开始进行微服务改造，从几个服务上线后经过春晚流量的考验后，逐步推广到上百个服务的上线，整个过程持续了两年多时间。虽然周期比较长，但是对于大流量的业务系统来说，稳定性永远是在第一位的，业务架构改造追求的是稳步推进，中间可以有小的波折，但对整体架构的演进方向不会产生影响。

做好技术取舍

我在搭建微服务架构的时候，其实做的最多的工作就是技术取舍。比如在开发 RPC 框架的时候，是选择自研呢还是采用开源 RPC 框架呢？如果自研的话，目前团队系统的主要语言是 Java，那么 RPC 框架是只支持 Java 语言就可以了，还是考虑到将来有可能需要支持其他语言呢？

我的经验就是一切以业务的实际情况为准，只要满足当前的需求就好，切忌好高骛远，尤其是对于技术能力很强的开发者来说，很容易陷入对技术的完美追求，投入过多精力在架构的雕花工作上，而忽视了眼下业务最实际的需求。尤其是在团队技术人力紧张，开发周期短的时候，更需要集中力量去满足业务最迫切的需求。而对于架构的完善以及一些附加功能的追求，可以在后面业务落地后逐步进行完善。

以微博的服务化框架 Motan 为例，因为微博平台的开发语言主要是 Java，所以最早 Motan 只支持 Java 语言。从 2017 年开始，有了跨语言服务化调用的需求，才在此基础上，对架构进行了升级，加入了对 Go、PHP 等语言的支持。而且在早期业务开始使用时，只开发了最基本的几个核心组件如 RPC 框架、注册中心和配置中心，以及简单的监控系统，而服务追踪系统、服务治理平台这些高级的功能都没有，后来随着重要业务进行微服务改造的越来越多，不断补充技术人力，才开始完善服务追踪系统以及服务治理平台。

除此之外，在做技术选型的时候，还要考虑到团队的实际掌控能力，尤其是对一些新技术方案的引入要尤其慎重。如果没有合适的人能够掌控这些技术，那么贸然引入新技术，一旦业务受影响时，如果没有人能有效干预，这对业务来说是灾难性的后果。

微博在做注册中心选型的时候，没有选取当时很火的 Zookeeper 的一个重要原因就是，它底层依赖的是 HBase 存储，当时团队中还没有有经验的运维和开发人员；但团队对 Redis 十分了解，所以基于 Redis 存储，自研了一套注册中心，完全能够满足需求，并且又没有引入技术不可控因素。

采用 DevOps

微服务架构带来的不光是业务开发模式的改变，对测试和运维的影响也是根本性的。以往在单体应用架构时，开发只需要整体打包成一个服务，交给测试去做自动化测试、交给运维去部署发布就可以了。但是微服务架构下，一个单体应用被拆分成多个细的微服务，并且需要独自开发、测试和上线，如果继续按照之前的单体应用模式运维，那么测试和运维的工作量相当于成倍的增加。因此迫切需要对以往的开发、测试和运维模式进行升级，从我的经验来看，最好的方案就是采用 DevOps，对微服务架构进行一站式开发、测试、上线和运维。

在单体应用架构下，开发、测试和运维这三者角色的区分是十分比较明显的，分属于不同的部门。而在微服务架构下，由于服务被拆分得足够细，每个服务都需要完成独立的开发、测试和运维工作，有自己完整的生命周期，所以需要将一个服务从代码开发、单元测试、集成测试以及服务发布都自动化起来。这样的话，测试人员就可以从众多微服务的测试中解放出来，着重进行自动化测试用例的维护；运维人员也可以从众多微服务的上线发布工作中解放出来，着重进行 DevOps 体系工具的建设。而每个服务的开发负责人，需要对服务的整个生命周期负责，无论是在代码检查阶段出现问题，还是测试阶段和发布阶段出现问题，都需要去解决。

统一微服务治理平台

以前我们只需要关心一个大的单体应用的健康状况，所以团队可以针对大的单体应用专门监控。但进行微服务改造后，拆分出几个甚至上百个服务之后，再靠传统的运维方案去管理，就会显得力不从心了。

而且微服务架构下会衍生出许多新的问题，比如 RPC 调用超时、注册中心获取失败、服务容量不足等，有些问题需要开发介入去定位分析，而有些问题需要运维介入，十分混乱。

微博在进行微服务改造初期，就面临着诸多问题，比如某一个微服务的容量不足了，需要进行扩容，而它所依赖的服务也需要进行扩容，但这种依赖关系只有业务的开发人员清楚，运维人员其实并不知晓详情。还有就是某个服务依赖的另一个服务出现故障，需要紧急降级，而此时如果运维人员操作的话并不知道哪个开关，虽然开发知晓，但开发实际上又没有线上服务器的操作权限。

所以，这时就迫切需要一个微服务治理平台，能够将微服务的服务治理以及各种运维操作都统一管理起来，并且打破开发和运维之间的隔阂，给予同样的权限，让服务的开发人员真正做到对自己的服务负责，不仅要对自己的服务情况了如指掌，还需要能对自己的服务进行治理和运维。

基于此，也就需要开发和运维深入合作，发挥各自专业的特长，将微服务治理的功能以及之前运维系统的基础功能结合在一起，打造成“一站式”微服务治理平台。

总结

今天我给你讲解了微服务架构如何在业务中进行落地，总结来讲就是，首先你必须组建一支合适的技术团队，这其中不仅要包含资深的架构师，还需要包含业务的开发者。在选择业务进行微服务架构改造时，不能追大求全，正确的做法应当是先以一个适当规模的业务进行微服务改造，走完整个微服务架构落地的过程，从而找出问题，不断打磨到成熟可用的状态，再推广到更多更重要的业务当中。在改造的过程中，要做好技术取舍，以团队人员的实际情况以及业务的实际需求为准绳，切忌追新立异，避免给业务引入不可控因素，留下“架构债”。同时，微服务架构的过程，也是团队组织变革的过程，传统意义上的开发、测试和运维明确的分割线会被打破，出现一种 DevOps 工程师的角色，他需要对服务全生命周期负责。为了做到这一点，就需要一个统一的微服务治理平台，融合服务治理和运维的各种功能。

实际上，每个团队都有各自不同的情况，但只要秉承上面这些基本准则，就可以走出一条适合自己团队的微服务架构路线出来，这其中没有高低之分，适合自己的才是最好的。

思考题

传统单体应用下进行测试只需要启动单体应用部署一个测试环境即可进行集成测试，但经过微服务改造后，一个功能依赖了多个微服务，每个微服务都有自己的测试环境，这个时候该如何进行集成测试呢？

欢迎你在留言区写下自己的思考，与我一起讨论。

写留言

通过留言可与作者互动