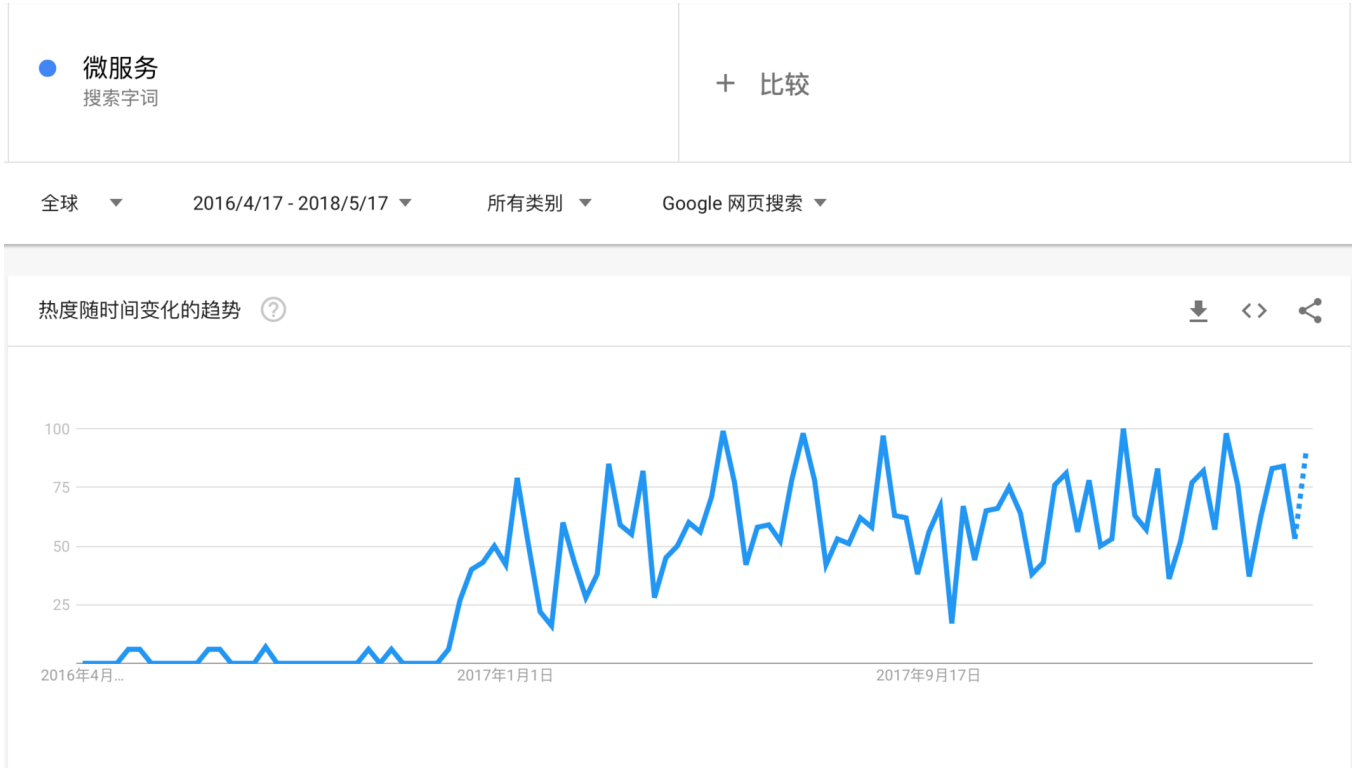
01 | 到底什么是微服务?

2018-08-23 胡忠想

01 | 到底什么是微服务?
朗读人：胡忠想　08′58″ | 4.12M

从谷歌的搜索指数来看，微服务的热度在进入 2017 年后突然爆发，国内各大会议和论坛的相关讨论也如雨后春笋般层出不穷，各大一线互联网公司也纷纷将这一技术引入并在实际业务中落地。



然而据我所知，国内不少中小规模的技术团队对微服务的概念都不甚了解，对该不该引入微服务也不置可否。还有一些技术团队，没有考虑实际业务场景，只是为了追求技术热点，盲目引入微服务，但又缺乏相应的技术掌控能力，最后影响了业务的稳定性。

对于该不该引入微服务，以及微服务体系需要哪些技术，目前并没有适合中小团队的架构实践落地的指引。因此我结合自己在微博多年的业务实践，总结出了一套微服务落地经验，从基础理论到架构实践，再结合业界最新趋势分析，希望能帮助中小规模团队了解微服务的本质以及对业务的价值，从而做出正确的判断。

我们先来看看维基百科是如何定义微服务的。微服务的概念最早是在 2014 年由 Martin Fowler 和 James Lewis 共同提出，他们定义了微服务是由单一应用程序构成的小服务，拥有自己的进程与轻量化处理，服务依业务功能设计，以全自动的方式部署，与其他服务使用 HTTP API 通讯。同时，服务会使用最小规模的集中管理 （例如 Docker）技术，服务可以用不同的编程语言与数据库等。

这个理论的定义看着有点晕？没关系，接下来我来帮你理解到底什么是微服务？

单体应用

在开聊微服务之前，我先要你和介绍下单体应用。如果你不知道单体应用的痛，那也不会深刻理解微服务的价值。

早些年，各大互联网公司的应用技术栈大致可分为 LAMP（Linux + Apache + MySQL + PHP）和 MVC（Spring + iBatis/Hibernate + Tomcat）两大流派。无论是 LAMP 还是

MVC，都是为单体应用架构设计的，其优点是学习成本低，开发上手快，测试、部署、运维也比较方便，甚至一个人就可以完成一个网站的开发与部署。

以 MVC 架构为例，业务通常是通过部署一个 WAR 包到 Tomcat 中，然后启动 Tomcat，监听某个端口即可对外提供服务。早期在业务规模不大、开发团队人员规模较小的时候，采用单体应用架构，团队的开发和运维成本都可控。

然而随着业务规模的不断扩大，团队开发人员的不断扩张，单体应用架构就会开始出现问题。我估计经历过业务和团队快速增长的同学都会对此深有感触。从我的角度来看，大概会有以下几个方面的问题。

- 部署效率低下。以我实际参与的项目为例，当单体应用的代码越来越多，依赖的资源越来越多时，应用编译打包、部署测试一次，甚至需要 10 分钟以上。这也经常被新加入的同学吐槽说，部署测试一次的时间，都可以去楼下喝杯咖啡了。

- 团队协作开发成本高。以我的经验，早期在团队开发人员只有两三个人的时候，协作修改代码，最后合并到同一个 master 分支，然后打包部署，尚且可控。但是一旦团队人员扩张，超过 5 人修改代码，然后一起打包部署，测试阶段只要有一块功能有问题，就得重新编译打包部署，然后重新预览测试，所有相关的开发人员又都得参与其中，效率低下，开发成本极高。

- 系统高可用性差。因为所有的功能开发最后都部署到同一个 WAR 包里，运行在同一个 Tomcat 进程之中，一旦某一功能涉及的代码或者资源有问题，那就会影响整个 WAR 包中部署的功能。比如我经常遇到的一个问题，某段代码不断在内存中创建大对象，并且没有回收，部署到线上运行一段时间后，就会造成 JVM 内存泄露，异常退出，那么部署在同一个 JVM 进程中的所有服务都不可用，后果十分严重。

- 线上发布变慢。特别是对于 Java 应用来说，一旦代码膨胀，服务启动的时间就会变长，有些甚至超过 10 分钟以上，如果机器规模超过 100 台以上，假设每次发布的步长为 10%，单次发布需要就需要 100 分钟之久。因此，急需一种方法能够将应用的不同模块的解耦，降低开发和部署成本。

想要解决上面这些问题，服务化的思想也就应运而生。

什么是服务化?

这里我就不谈一些官方的、教条主义的概念了。在我看来，用通俗的话来讲，服务化就是把传统的单机应用中通过 JAR 包依赖产生的本地方法调用，改造成通过 RPC 接口产生的远程方法调用。一般在编写业务代码时，对于一些通用的业务逻辑，我会尽力把它抽象并独立成为专门的模块，因为这对于代码复用和业务理解都大有裨益。

在过去的项目经历里，我对此深有体会。以微博系统为例，微博既包含了内容模块，也包含了消息模块和用户模块等。其中消息模块依赖内容模块，消息模块和内容模块又都依赖用户模块。当这三个模块的代码耦合在一起，应用启动时，需要同时去加载每个模块的代码并连接对应的资源。一旦任何模块的代码出现 bug，或者依赖的资源出现问题，整个单体应用都会受到影响。

为此，首先可以把用户模块从单体应用中拆分出来，独立成一个服务部署，以 RPC 接口的形式对外提供服务。微博和消息模块调用用户接口，就从进程内的调用变成远程 RPC 调用。这样，用户模块就可以独立开发、测试、上线和运维，可以交由专门的团队来做，与主模块不耦合。进一步的可以再把消息模块也拆分出来作为独立的模块，交由专门的团队来开发和维护。

可见通过服务化，可以解决单体应用膨胀、团队开发耦合度高、协作效率低下的问题。

什么是微服务？

从 2014 年开始，得益于以 Docker 为代表的容器化技术的成熟以及 DevOps 文化的兴起，服务化的思想进一步演化，演变为今天我们所熟知的微服务。

那么微服务相比于服务化又有什么不同呢？

在我看来，可以总结为以下四点：

- 服务拆分粒度更细。微服务可以说是更细维度的服务化，小到一个子模块，只要该模块依赖的资源与其他模块都没有关系，那么就可以拆分为一个微服务。
- 服务独立部署。每个微服务都严格遵循独立打包部署的准则，互不影响。比如一台物理机上可以部署多个 Docker 实例，每个 Docker 实例可以部署一个微服务的代码。
- 服务独立维护。每个微服务都可以交由一个小团队甚至个人来开发、测试、发布和运维，并对整个生命周期负责。
- 服务治理能力要求高。因为拆分为微服务之后，服务的数量变多，因此需要有统一的服务治理平台，来对各个服务进行管理。

继续以前面举的微博系统为例，可以进一步对内容模块的功能进行拆分，比如内容模块又包含了 feed 模块、评论模块和个人页模块。通过微服务化，将这三个模块变成三个独立的服务，每个服务依赖各自的资源，并独立部署在不同的服务池中，可以由不同的开发人员进行维护。当评论服务需求变更时，只需要修改评论业务相关的代码，并独立上线发布；而 feed 服务和个人页服务不需要变更，也不会受到发布可能带来的变更影响。

由此可见，微服务化给服务的发布和部署，以及服务的保障带来了诸多好处。

总结

今天，我介绍了微服务的发展由来，它是由单体应用进化到服务化拆分部署，后期随着移动互联网规模的不断扩大，敏捷开发、持续交付、DevOps 理论的发展和实践，以及基于 Docker 容器化技术的成熟，微服务架构开始流行，逐渐成为应用架构的未来演进方向。

总结来说，微服务架构是将复杂臃肿的单体应用进行细粒度的服务化拆分，每个拆分出来的服务各自独立打包部署，并交由小团队进行开发和运维，从而极大地提高了应用交付的周期，并被各大互联网公司所普遍采用。

思考题

你在业务开发中是否也遇到过因单体应用过度膨胀所带来的问题呢？你觉得针对这些问题微服务能解决吗？

欢迎你在留言区写下自己的思考，与我一起讨论。

精选留言

WolvesLeader 2

老师，问一下，一般和数据库交互的dao层，是单独拿出来做成服务好还是不做好，还有一些公共的模块是不是也要做成服务呢？谢谢啦

2018-08-23

乐杰 1

可以部分解决，返回有涉及关联数据查询问题，如文件和用户关联，查看文件列表时，需要显示用户名称，而文件服务只存储用户ID，那么此时如何取用户名好，不带来性能问题

2018-08-23

lwt 1

一个应用包含太多功能，每次改动一块的功能，测试需要整体测试，周期很长。
同一时期很多项目都需要去改同一个应用，一个项目上线后，其他项目的需要合并这个项目的代码，然后整体回归测试。
因为应该集成的功能很多，新人很难在短时间内熟悉整个应用。

2018-08-23

李 0

你好，就目前的趋势来看，微服务架构未来会是springcloud还是阿里的db会胜出

2018-08-23