微服务是什么

< 上一节 下一节 >

微服务（MicroServices）最初是由 Martin Fowler 于 2014 年发表的论文 《MicroServices》 中提出的名词，它一经提出就成为了技术圈的热门话题。

微服务，我们可以从字面上去理解，即“微小的服务”，下面我们从“服务”和“微小”两个方面进行介绍。

1) 所谓“服务”，其实指的是项目中的功能模块，它可以帮助用户解决某一个或一组问题，在开发过程中表现为 IDE（集成开发环境，例如 Eclipse 或 IntelliJ IDEA）中的一个工程或 Moudle。

2) “微小”则强调的是单个服务的大小，主要体现为以下两个方面：

- 微服务体积小，复杂度低：一个微服务通常只提供单个业务功能的服务，即一个微服务只专注于做好一件事，因此微服务通常代码较少，体积较小，复杂度也较低。
- 微服务团队所需成员少：一般情况下，一个微服务团队只需要 8 到 10 名人员（开发人员 2 到 5 名）即可完成从设计、开发、测试到运维的全部工作。

微服务架构

微服务架构是一种系统架构的设计风格。与传统的单体式架构（ALL IN ONE）不同，微服务架构提倡将一个单一的应用程序拆分成多个小型服务，这些小型服务都在各自独立的进程中运行，服务之间使用轻量级通信机制（通常是 HTTP RESTFUL API）进行通讯。

通常情况下，这些小型服务都是围绕着某个特定的业务进行构建的，每一个服务只专注于完成一项任务并把它做好，即“专业的人做专业的事”。

每个服务都能够独立地部署到各种环境中，例如开发环境、测试环境和生产环境等，每个服务都能独立启动或销毁而不会对其他服务造成影响。

这些服务之间的交互是使用标准的通讯技术进行的，因此不同的服务可以使用不同数据存储技术，甚至使用不同的编程语言。

微服务架构 vs 单体架构

在当今的软件开发领域中，主要有两种系统架构风格，那就是新兴的“微服务架构”和传统的“单体架构”。

单体架构是微服务架构出现之前业界最经典的软件架构类型，许多早期的项目采用的也都是单体架构。单体架构将应用程序中所有业务逻辑都编写在同一个工程中，最终经过编译、打包，部署在一台服务器上运行。

在项目的初期，单体架构无论是在开发速度还是运维难度上都具有明显的优势。但随着业务复杂度的不断提高，单体架构的许多弊端也逐渐凸显出来，主要体现在以下 3 个方面：

- 随着业务复杂度的提高，单体应用（采用单体架构的应用程序）的代码量也越来越大，导致代码的可读性、可维护性以及扩展性下降。
- 随着用户越来越多，程序所承受的并发越来越高，而单体应用处理高并发的能力有限。
- 单体应用将所有的业务都集中在同一个工程中，修改或增加业务都可能会对其他业务造成一定的影响，导致测试难度增加。

由于单体架构存在这些弊端，因此许多公司和组织都开始将将它们的项目从单体架构向微服务架构转型。

下面我们就来对比下微服务架构和单体架构到底有什么不同。

不同点	微服务架构	单体架构
团队规模	微服务架构可以将传统模式下的单个应用拆分为多个独立的服务，每个微服务都可以单独开发、部署和维护。每个服务从设计、开发到维护所需的团队规模小，团队管理成本小。	单体架构的应用程序通常需要一个大型团队，围绕一个庞大的应用程序工作，团队管理的成本大。
数据存储方式	不同的微服务可以使用不同的数据存储方式，例如有的用 Redis，有的使用 MySQL。	单一架构的所有模块共享同一个公共数据库，存储方式相对单一。
部署方式	微服务架构中每个服务都可以独立部署，也可以独立于其他服务进行扩展。如果部署得当，基于微服务的架构可以帮助企业提高应用程序的部署效率。	采用单体架构的应用程序的每一次功能更改或 bug 修复都必须对整个应用程序重新进行部署。
开发模式	在采用微服务架构的应用程序中，不同模块可以使用不同的技术或语言进行开发，开发模式更加灵活。	在采用单体架构的应用程序中，所有模块使用的技术和语言必须相同，开发模式受限。
故障隔离	在微服务架构中，故障被隔离在单个服务中，避免系统的整体崩溃。	在单体架构中，当一个组件出现故障时，故障很可能会在进程中蔓延，导致系统全局不可用。
项目结构	微服务架构将单个应用程序拆分为多个独立的小型服务，每个服务都可以独立的开发、部署和维护，每个服务都能完成一项特定的业务需求。	单体架构的应用程序，所有的业务逻辑都集中在同一个工程中。

微服务的特点

微服务具有以下特点：

- 服务按照业务来划分，每个服务通常只专注于某一个特定的业务、所需代码量小，复杂度低、易于维护。
- 每个微服都可以独立开发、部署和运行，且代码量较少，因此启动和运行速度较快。
- 每个服务从设计、开发、测试到维护所需的团队规模小，一般 8 到 10 人，团队管理成本小。
- 采用单体架构的应用程序只要有任何修改，就需要重新部署整个应用才能生效，而微服务则完美地解决了这一问题。在微服架构中，某个微服务修改后，只需要重新部署这个服务即可，而不需要重新部署整个应用程序。
- 在微服务架构中，开发人员可以结合项目业务及团队的特点，合理地选择语言和工具进行开发和部署，不同的微服务可以使用不同的语言和工具。
- 微服务具备良好的可扩展性。随着业务的不断增加，微服务的体积和代码量都会急剧膨胀，此时我们可以根据业务将微服务再次进行拆分；除此之外，当用户量和并发量的增加时，我们还可以将微服务集群化部署，从而增加系统的负载能力。
- 微服务能够与容器（Docker）配合使用，实现快速迭代、快速构建、快速部署。
- 微服务具有良好的故障隔离能力，当应用程序中的某个微服发生故障时，该故障会被隔离在当前服务中，而不会波及到其他微服务造成整个系统的瘫痪。
- 微服务系统具有链路追踪的能力。

微服务框架

微服务架构是一种系统架构风格和思想，想要真正地搭建一套微服务系统，则需要微服务框架的支持。随着微服务的流行，很多编程语言都相继推出了它们的微服务框架，下面我们就来简单列举下。

Java 微服务框架

市面上的 Java 微服务框架主要有以下 5 种：

- Spring Cloud：它能够基于 REST 服务来构建服务，帮助架构师构建出一套完整的微服务技术生态链。
- Dropwizard：用于开发高性能和 Restful 的 Web 服务，对配置、应用程序指标、日志记录和操作工具都提供了开箱即用的支持。
- Restlet： 该框架遵循 RST 架构风格，可以帮助 Java 开发人员构建微服务。

- Spark：最好的 Java 微服务框架之一，该框架支持通过 Java 8 和 Kotlin 创建微服务架构的应用程序。
- Dubbo：由阿里巴巴开源的分布式服务治理框架。

Go 语言微服务框

Go 语言中的微服务框架较少，使用的较多的是 GoMicro，它是一个 RPC 框架，具有负载均衡、服务发现、同步通信、异步通讯和消息编码等功能。

Phyton 微服务框架

Phyton 中的微服务框架主要有 Flask、Falcon、Bottle、Nameko 和 CherryPy 等。

NodeJS微服务框架

Molecular 是一种使用 NodeJS 构建的事件驱动架构，该框架内置了服务注册表、动态服务发现、负载均衡、容错功能和内置缓存等组件。

< 上一节　　下一节 >

推荐阅读

Linux MySQL安装过程（详解版）
C++汉诺塔递归算法完全攻略
C++ STL map容器详解
学好数据结构，你已然超越了99%的程序员！
JS String（字符串）对象
SELinux安全上下文查看方法（超详细）
Python format()格式化输出方法详解
MySQL TRUNCATE：清空表记录
数据库逻辑结构设计阶段（非常重要）
C语言qsort()：快速排序