## 1、什么是微服务？

微服务是将一个原本独立的系统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，服务和服务之间采用轻量级的通信机制进行协作。每个服务可以被独立的部署到生产环境。

## 2、单体系统的缺点：

修改一个小功能，就需要将整个系统重新部署上线，影响其他功能的运行；

功能模块互相依赖，强耦合，扩展困难。如果出现性能瓶颈，需要对整体应用进行升级，虽然影响性能的可能只是其中一个小模块；

## 3、单体系统的优点：

容易部署，程序单一，不存在分布式集群的复杂部署环境；

容易测试，没有复杂的服务调用关系。

## 4、微服务的优点：

不同的服务可以使用不同的技术；

隔离性。一个服务不可用不会导致其他服务不可用；

可扩展性。某个服务出现性能瓶颈，只需对此服务进行升级即可；

简化部署。服务的部署是独立的，哪个服务出现问题，只需对此服务进行修改重新部署；

## 5、微服务的缺点：

网络调用频繁。性能相对函数调用较差。

运维成本增加。系统由多个独立运行的微服务构成，需要设计一个良好的监控系统对各个微服务的运行状态进行监控。

## 6、分布式和微服务的区别

从概念理解，分布式服务架构强调的是服务化以及服务的分散化，微服务则更强调服务的专业化和精细分工；

从实践的角度来看，微服务架构通常是分布式服务架构，反之则未必成立。

一句话概括：分布式：分散部署；微服务：分散能力。

## 7、服务怎么划分？

横向拆分：按照不同的业务域进行拆分，例如订单、营销、风控、积分资源等。形成独立的业务领域微服务集群。

纵向拆分：把一个业务功能里的不同模块或者组件进行拆分。例如把公共组件拆分成独立的原子服务，下沉到底层，形成相对独立的原子服务层。这样一纵一横，就可以实现业务的服务化拆分。

要做好微服务的分层：梳理和抽取核心应用、公共应用，作为独立的服务下沉到核心和公共能力层，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求

总之，微服务的设计一定要 渐进式 的，总的原则是 服务内部高内聚，服务之间低耦合。

## 8、微服务设计原则

单一职责原则

意思是每个微服务只需要实现自己的业务逻辑就可以了，比如订单管理模块，它只需要处理订单的业务逻辑就可以了，其它的不必考虑。

服务自治原则

意思是每个微服务从开发、测试、运维等都是独立的，包括存储的数据库也都是独立的，自己就有一套完整的流程，我们完全可以把它当成一个项目来对待。不必依赖于其它模块。

轻量级通信原则

首先是通信的语言非常的轻量，第二，该通信方式需要是跨语言、跨平台的，之所以要跨平台、跨语言就是为了让每个微服务都有足够的独立性，可以不受技术的钳制。

接口明确原则

由于微服务之间可能存在着调用关系，为了尽量避免以后由于某个微服务的接口变化而导致其它微服务都做调整，在设计之初就要考虑到所有情况，让接口尽量做的更通用，更灵活，从而尽量避免其它模块也做调整。

## 9、微服务之间是如何通讯的？

1、RPC

优点：简单，常见。因为没有中间件代理，系统更简单

缺点：

只支持请求/响应的模式，不支持别的，比如通知、发布/订阅

降低了可用性，因为客户端和服务端在请求过程中必须都是可用的

2、消息队列

除了标准的基于RPC通信的微服务架构，还有基于消息队列通信的微服务架构，这种架构下的微服务采用发送消息（Publish Message）与监听消息（Subscribe Message）的方式来实现彼此之间的交互。

网易的蜂巢平台就采用了基于消息队列的微服务架构设计思路，如下图所示，微服务之间通过RabbitMQ传递消息，实现通信。

与上面几种微服务架构相比，基于消息队列的微服务架构并不多，案例也相对较少，更多地体现为一种与业务相关的设计经验，各家有各家的实现方式，缺乏公认的设计思路与参考架构，也没有形成一个知名的开源平台。因此，如果需要实施这种微服务架构，则基本上需要项目组自己从零开始去设计实现一个微服务架构基础平台，其代价是成本高、风险大。

优点:

把客户端和服务端解耦，更松耦合提高可用性，因为消息中间件缓存了消息，直到消费者可以消费

支持很多通信机制比如通知、发布/订阅等

缺点：

缺乏公认的设计思路与参考架构，也没有形成一个知名的开源平台

成本高、风险大

## 10、熔断器

雪崩效应：假如存在这样的调用链路，a服务->b服务->c服务，当c服务挂了的时候，b服务调用c服务会等待超时，a服务调用b服务也会等待超时，调用方长时间等不到相应而占用线程，如果有大量的请求过来，就会造成线程池打满，导致整个链路的服务奔溃。

为了解决分布式系统的雪崩效应，分布式系统引进了熔断器机制 。

当一个服务的处理用户请求的失败次数在一定时间内小于设定的阀值时，熔断器出于关闭状态，服务正常。

当服务处理用户请求失败次数在一定时间内大于设定的阀值时，说明服务出现故障，打开熔断器，这时所有的请求会快速返回失败的错误信息，不执行业务逻辑，从而防止故障的蔓延。

当处于打开状态的熔断器时，一段时间后出于半打开状态，并执行一定数量的请求，剩余的请求会执行快速失败，若执行请求失败了，则继续打开熔断器，若成功了，则将熔断器关闭。

## 11、服务网关

### 何为网关？

通俗一点的讲：网关就是要去别的网络的时候，把报文首先发送到的那台设备。稍微专业一点的术语，网关就是当前主机的默认路由。

网关一般就是一台路由器，有点像“一个小区中的一个邮局”，小区里面的住户互相是知道怎么走，但是要向外地投递东西就不知道了，怎么办？把地址写好送到本小区的邮局就好了。

那么，怎么区分是“本小区”和“外地小区”的呢？根据IP地址 + 掩码。如果是在一个范围内的，就是本小区（局域网内部），如果掩不住的，就是外地的。

例如，你的机器的IP地址是：192.168.0.2/24，网关是192.168.0.1

如果机器访问的IP地址范围是：192.168.0.1~192.168.0.254的，说明是一个小区的邻居，你的机器就直接发送了（和网关没任何关系）。如果你访问的IP地址不是这个范围的，则就投递到192.168.0.1上，让这台设备来转发。

参考：https://www.zhihu.com/question/362842680/answer/951412213

### 何为API网关

假设你正在开发一个电商网站，那么这里会涉及到很多后端的微服务，比如会员、商品、推荐服务等等。

那么这里就会遇到一个问题，APP/Browser怎么去访问这些后端的服务? 如果业务比较简单的话，可以给每个业务都分配一个独立的域名(https://service.api.company.com)，但这种方式会有几个问题:

每个业务都会需要鉴权、限流、权限校验等逻辑，如果每个业务都各自为战，自己造轮子实现一遍，会很蛋疼，完全可以抽出来，放到一个统一的地方去做。

如果业务量比较简单的话，这种方式前期不会有什么问题，但随着业务越来越复杂，比如淘宝、亚马逊打开一个页面可能会涉及到数百个微服务协同工作，如果每一个微服务都分配一个域名的话，一方面客户端代码会很难维护，涉及到数百个域名

每上线一个新的服务，都需要运维参与，申请域名、配置Nginx等，当上线、下线服务器时，同样也需要运维参与，另外采用域名这种方式，对于环境的隔离也不太友好，调用者需要自己根据域名自己进行判断。

另外还有一个问题，后端每个微服务可能采用了不同的协议，比如HTTP、AMQP、自定义TCP协议等，但是你不可能要求客户端去适配这么多种协议，这是一项非常有挑战的工作，项目会变的非常复杂且很难维护。

更好的方式是采用API网关（也叫做服务网关），实现一个API网关接管所有的入口流量，类似Nginx的作用，将所有用户的请求转发给后端的服务器，但网关做的不仅仅只是简单的转发，也会针对流量做一些扩展，比如鉴权、限流、权限、熔断、协议转换、错误码统一、缓存、日志、监控、告警等，这样将通用的逻辑抽出来，由网关统一去做，业务方也能够更专注于业务逻辑，提升迭代的效率。

通过引入API网关，客户端只需要与API网关交互，而不用与各个业务方的接口分别通讯，但多引入一个组件就多引入了一个潜在的故障点，因此要实现一个高性能、稳定的网关，也会涉及到很多点。

网关层通常以集群的形式存在。并在服务网关层前通常会加上Nginx 用来负载均衡。

## 12、服务网关基本功能：

智能路由：接收外部一切请求，并转发到后端的对外服务。注意：我们只转发外部请求，服务之间的请求不走网关，这就表示全链路追踪、内部服务API监控、内部服务之间调用的容错、智能路由不能在网关完成；当然，也可以将所有的服务调用都走网关，那么几乎所有的功能都可以集成到网关中，但是这样的话，网关的压力会很大，不堪重负。

权限校验：网关可以做一些用户身份认证，权限认证，防止非法请求操作API 接口，对内部服务起到保护作用

API监控：监控经过网关的请求，以及网关本身的一些性能指标（gc等）

限流：与监控配合，进行限流操作

API日志统一收集：类似于一个aspect切面，记录接口的进入和出去时的相关日志

当然，网关实现这些功能，需要做高可用，否则网关很可能成为架构的瓶颈，最常用的网关组件Zuul、Nginx

## 13、服务配置统一管理

在微服务架构中，需要有统一管理配置文件的组件，例如：SpringCloud Config组件、阿里的Diamond、百度的Disconf、携程的Apollo等

## 14、服务链路追踪

在微服务架构中，必须实现分布式链路追踪，去跟进一个请求到底有哪些服务参与、参与顺序，是每个请求链路清晰可见，便于问题快速定位。

常用链路追踪组件有Google的Dapper、Twitter 的Zipkin，以及阿里Eagleeye。

## 15、微服务框架

市面常用微服务框架有：Spring Cloud 、Dubbo 、kubernetes

从功能模块上考虑，Dubbo缺少很多功能模块，例如网关、链路追踪等

从学习成本上考虑，Dubbo 版本趋于稳定，稳定完善、可以即学即用，难度简单，Spring cloud 基于Spring Boot，需要先掌握Spring Boot ，例外Spring cloud 大多为英文文档，要求学习者有一定的英文阅读能力

从开发风格考虑，Dubbo倾向于xml的配置方式，Spring cloud 基于Spring Boot ，采用基于注解和JavaBean配置方式的敏捷开发

从开发速度上考虑，Spring cloud 具有更高的开发和部署速度

从通信方式上考虑，Spring cloud 基于HTTP Restful 风格，服务于服务之间完全无关、无耦合。Dubbo 基于远程调用，对接口、平台和语言有强依赖性，如果需要实现跨平台，需要有额外的中间件。

所以Dubbo专注于服务治理；Spring Cloud关注于微服务架构生态。

其他

Spring Cloud基础知识

Spring Cloud基础知识

什么是Service Mesh？

Service Mesh 是微服务时代的 TCP/IP 协议。

————————————————

版权声明：本文为CSDN博主「程序员大彬」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/Tyson0314/article/details/127815190