转载：https://blog.csdn.net/zpoison/article/details/80729052

1.SOA架构和微服务架构的区别

首先SOA和微服务架构一个层面的东西，而对于ESB和微服务网关是一个层面的东西，一个谈到是架构风格和方法，一个谈的是实现工具或组件。

1.SOA（Service Oriented Architecture）“面向服务的架构”：他是一种设计方法，其中包含多个服务， 服务之间通过相互依赖最终提供一系列的功能。一个服务 通常以独立的形式存在与操作系统进程中。各个服务之间 通过网络调用。

2.微服务架构:其实和 SOA 架构类似,微服务是在 SOA 上做的升华，微服务架构强调的一个重点是“业务需要彻底的组件化和服务化”，原有的单个业务系统会拆分为多个可以独立开发、设计、运行的小应用。这些小应用之间通过服务完成交互和集成。

微服务架构 = 80%的SOA服务架构思想 + 100%的组件化架构思想 + 80%的领域建模思想

2.ESB和微服务API网关。

1.ESB（企业服务总线），简单 来说 ESB 就是一根管道，用来连接各个服务节点。为了集 成不同系统，不同协议的服务，ESB 做了消息的转化解释和路由工作，让不同的服务互联互通；

2.API网关:API网关是一个服务器，是系统的唯一入口。从面向对象设计的角度看，它与外观模式类似。API网关封装了系统内部架构，

为每个客户端提供一个定制的API。它可能还具有其它职责，如身份验证、监控、负载均衡、缓存、请求分片与管理、静态响应处理。API网关方式的核心要点是，所有的客户端和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。通常，网关也是提供REST/HTTP的访问API。服务端通过API-GW注册和管理服务。

3.SOA架构特点：

系统集成：站在系统的角度，解决企业系统间的通信问 题，把原先散乱、无规划的系统间的网状结构，梳理成 规整、可治理的系统间星形结构，这一步往往需要引入 一些产品，比如 ESB、以及技术规范、服务管理规范； 这一步解决的核心问题是【有序】

系统的服务化：站在功能的角度，把业务逻辑抽象成 可复用、可组装的服务，通过服务的编排实现业务的 快速再生，目的：把原先固有的业务功能转变为通用 的业务服务，实现业务逻辑的快速复用；这一步解决 的核心问题是【复用】

业务的服务化：站在企业的角度，把企业职能抽象成 可复用、可组装的服务；把原先职能化的企业架构转变为服务化的企业架构，进一步提升企业的对外服务能力；“前面两步都是从技术层面来解决系统调用、系统功能复用的问题”。第三步，则是以业务驱动把一个业务单元封装成一项服务。这一步解决的核心问题是【高效】

4.微服务架构特点：

1.通过服务实现组件化

开发者不再需要协调其它服务部署对本服务的影响。

2.按业务能力来划分服务和开发团队

开发者可以自由选择开发技术，提供 API 服务

3.去中心化

每个微服务有自己私有的数据库持久化业务数据

每个微服务只能访问自己的数据库，而不能访问其它服务的数据库

某些业务场景下，需要在一个事务中更新多个数据库。这种情况也不能直接访问其它微服务的数据库，而是通过对于微服务进行操作。

数据的去中心化，进一步降低了微服务之间的耦合度，不同服务可以采用不同的数据库技术（SQL、NoSQL等）。在复杂的业务场景下，如果包含多个微服务，通常在客户端或者中间层（网关）处理。

4.基础设施自动化（devops、自动化部署）

的Java EE部署架构，通过展现层打包WARs，业务层划分到JARs最后部署为EAR一个大包，而微服务则打开了这个黑盒子，把应用拆分成为一个一个的单个服务，应用Docker技术，不依赖任何服务器和数据模型，是一个全栈应用，可以通过自动化方式独立部署，每个服务运行在自己的进程中，通过轻量的通讯机制联系，经常是基于HTTP资源API，这些服务基于业务能力构建，能实现集中化管理（因为服务太多啦，不集中管理就无法DevOps啦）。

5.主要区别：

功能	SOA	微服务
组件大小	大块业务逻辑	单独任务或小块业务逻辑
耦合	通常松耦合	总是松耦合
公司架构	任何类型	小型、专注于功能交叉团队
管理	着重中央管理	着重分散管理
目标	确保应用能够交互操作	执行新功能、快速拓展开发团队

6.Dubbo服务的最佳实践

分包

服务接口、请求服务模型、异常信息都放在api里面，符合重用发布等价原则，共同重用原则

api里面放入spring 的引用配置。 也可以放在模块的包目录下。

粒度

尽可能把接口设置成粗粒度，每个服务方法代表一个独立的功能，而不是某个功能的步骤。否则就会涉及到分布式事务

服务接口建议以业务场景为单位划分。并对相近业务做抽象，防止接口暴增

不建议使用过于抽象的通用接口 T T<泛型>，接口没有明确的语义，带来后期的维护

版本

每个接口都应该定义版本，为后续的兼容性提供前瞻性的考虑

version （maven -snapshot）

建议使用两位版本号，因为第三位版本号表示的兼容性升级，只有不兼容时才需要变更服务版本

当接口做到不兼容升级的时候，先升级一半或者一台提供者为新版本，再将消费全部升级新版本，然后再将剩下的一半提供者升级新版本

预发布环境

推荐用法

在provider端尽可能配置consumer端的属性

比如timeout、retires、线程池大小、LoadBalance

配置管理员信息

application上面配置的owner 、 owner建议配置2个人以上。因为owner都能够在监控中心看到

配置dubbo缓存文件

注册中心的列表

服务提供者列表

参考文献：

http://www.uml.org.cn/zjjs/201708083.asp

https://zhidao.baidu.com/question/1899225333752310100.html

http://blog.sina.com.cn/s/blog_493a84550102wq50.html