# 前言

一个成熟的大型网站系统架构并不是一开始就设计的非常完美的，世上没有完美的架构，也不是一开始就具备高性能、高可用、安全性等特性，而是随着用户量的不断增加、业务功能的扩展逐步完善演变过来的。在这个过程中，系统架构也因此也不断的演进、升级、迭代。从单一应用，到垂直拆分，到分布式服务，到SOA，以及现在火热的微服务架构 。如果一个软件开发人员，不了解软件架构的演进，会制约技术的选型和开发人员的生存、晋升空间。这里我列举了目前主要的四种软件架构以及他们的优缺点，希望能够帮助软件开发人员拓展知识面。

# 架构演变

## 单体架构(all in one)

单体架构是最简单的架构风格，所有的代码全都在一个项目中。这样研发团队的任何一个人都可以随时修改任意的一段代码，或者增加一些新的代码。开发人员也可以只在自己的电脑上就可以随时开发、调试、测试整个系统的功能。也不需要额外的一些依赖条件和准备步骤，我们就可以直接编译打包整个系统代码，创建一个可以发布的二进制版本。这种方式对于一个新团队的创立初期，需要迅速开始从 0 到 1，抓住时机实现产品最短时间推向市场，可以省去各种额外的设计，直接上手干活，争取了时间，因而是非常有意义的。单体架构的应用比较容易部署、测试，在项目的初期，单体应用可以很好地运行。然而，随着需求的不断增加，越来越多的人加入开发团队，代码库也在飞速地膨胀。慢慢地，单体应用变得越来越臃肿，可维护性、灵活性逐渐降低，维护成本越来越高。这种方式对于一个系统的长期稳定发展确实有很多缺点的：

|  |
| --- |
| * **项目复杂性高，代码严重耦合，代码维护困难**   单体架构的系统存在代码严重耦合的问题，以一个百万行级别的单体应用为例，整个项目包含的模块非常多、模块的边界模糊、依赖关系不清晰、代码质量参差不齐、混乱地堆砌在一起。可想而知整个项目非常复杂。每次修改代码都心惊胆战，甚至添加一个简单的功能，或者修改一个Bug都会带来隐含的缺陷，修改一处代码，可能会影响一大片的功能无法正常使用。   * **欠下一屁股技术债务**   随着时间推移、需求变更和人员更迭，会逐渐形成应用程序的技术债务，并且越积越多。“ 不坏不修”，这在软件开发中非常常见，在单体应用中这种思想更甚。已使用的系统设计或代码难以被修改，因为应用程序中的其他模块可能会以意料之外的方式使用它。   * **部署频率低**   随着代码的增多，构建和部署的时间也会增加。而在单体应用中，**每次功能的变更或缺陷的修复都会导致需要重新部署整个应用**。全量部署的方式耗时长、影响范围大、风险高，这使得单体应用项目上线部署的频率较低。而部署频率低又导致两次发布之间会有大量的功能变更和缺陷修复，出错率比较高。   * **局部功能的修改需要整个系统重新发布， 牵一发而动全身**   单体架构的系统变更对部署影响大，并且这个问题是所有的单体架构系统都存在的问题。系统作为一个单体部署，每次发布的部署单元就是一个新版本的整个系统，系统内的任何业务逻辑调整都会导致整个系统的重新打包，部署、停机、再重启，进而导致了系统的停机发布时间较长。每次发布上线都是生产系统的重大变更，这种部署模式大大提升了系统风险，降低了系统的可用性。   * **可靠性差**   某个应用Bug，例如死循环、内存溢出等，可能会导致整个应用的崩溃   * **扩展能力受限**   单体应用只能作为一个整体进行扩展，无法根据业务模块的需要进行伸缩。例如，应用中有的模块是计算密集型的，它需要强劲的CPU；有的模块则是IO密集型的，需要更大的内存。如果任何一个业务存在性能问题，那么都需要**考虑多部署几个完整的实例的集群**，或者再加上负载均衡设备，才能保证整个系统的性能可以支撑用户的使用。   * **影响开发效率**   单体架构的系统影响开发效率。如果一个使用 Java 的简单单体项目代码超过 100 万行，那么在一台笔记本电脑上修改了代码后执行自动编译，可能需要等待十分钟以上，并且内存可能不够编译过程使用，这是非常难以忍受的。或者说因为别的模块出现问题，导致整个系统无法运行，这也是非常难以忍受的。   * **系统启动慢**   单体架构模式打包后的部署结构可能过于庞大，导致业务系统启动很慢，进而也会影响系统的可用性。这一条也是所有单体架构的系统都有的问题。   * **阻碍技术创新，技术栈受限**   单体应用往往使用统一的技术平台或方案解决所有的问题，团队中的每个成员都必须使用相同的开发语言和框架，要想引入新框架或新技术平台会非常困难。 |

综上所述，单体架构模式比较适用于规模较小的系统和规模较小的团队，特别是需要快速推出原型实现，以质量换速度的场景。

## 垂直架构(分布式架构)

**分布式：多个人完成一件事情**

**集群：多个人做同样的事情**

**分布式和集群的关系**

**比如说你开了一家饭店，饭店里有一个厨师，这个厨师又得配菜又得炒菜，这时候万一这个厨师生病了来不了了干不了活，这个时候你的饭店是不是就不可用了，对外提供不了吃饭服务了。这个时候呢你又招了一个厨师，这两个厨师都做的是一模一样的菜，那这两个厨师之间的关系就是集群。为了让厨师能专注的去炒菜，这个时候呢你又招了一个配菜员，配菜员和厨师之间的关系就是分布式。**

针对单体架构的不足，为了适应大型项目的开发需求，许多公司将一个单体系统按业务垂直拆分为若干个子系统，系统之间通过网络交互来完成用户的业务处理，每个系统可分布式部署，这种架构称为分布式架构。也就说将一个大系统拆分为多个子系统。

**特点**：

1、按业务垂直拆分成一个一个的单体系统，此架构也称为垂直架构。

2、系统与系统之间的存在数据冗余，耦合性较大，如上图中三个项目都存在客户信息。

3、系统之间的接口多为实现数据同步，如上图中三个项目要同步客户信息。

**优点**：

1、通过垂直拆分，每个子系统变成小型系统，功能简单，前期开发成本低，周期短。

2、每个子系统可按需伸缩。

3、每个子系统可采用不同的技术。

**缺点**：

1、子系统之间存在数据冗余、功能冗余，耦合性高。

2、按需伸缩粒度不够，对同一个子系统中的不同的业务无法实现，比如订单管理和用户管理。

## SOA架构

SOA是一种面向服务的架构，基于分布式架构，它将不同业务功能按服务进行拆分，并通过这些服务之间定义良好的接口和协议联系起来。

**特点**：

1、基于SOA的架构思想，将重复公用的功能抽取为组件，以服务的方式向各各系统提供服务。

2、各个系统与服务之间采用webservice、rpc等方式进行通信。

3、ESB企业服务总线作为系统与服务之间通信的桥梁。

**优点**：

1、将重复的功能抽取为服务，提高开发效率，提高系统的可重用性、可维护性。

2、可以针对不同服务的特点按需伸缩。

3、采用ESB减少系统中的接口耦合。

**缺点**：

1、系统与服务的界限模糊，会导致抽取的服务的**粒度过大**，系统与服务之间耦合性高。

2、虽然使用了ESB，但是服务的接口协议不固定，种类繁多，不利于系统维护。

## 微服务架构

基于SOA架构的思想，为了满足移动互联网对大型项目及多客户端的需求，对服务层进行**细粒度**的拆分，所拆分的每个服务只完成某个特定的业务功能，比如订单服务只实现订单相关的业务，用户服务实现用户管理相关的业务等等，**服务的粒度很小，所以称为微服务架构**。

**特点**：

1、服务层按业务拆分为一个一个的微服务。

2、微服务的职责单一。

3、微服务之间采用RESTful、RPC等轻量级协议传输。

<https://www.jianshu.com/p/2accc2840a1b>

4、有利于采用前后端分离架构。

**优点**：

1、服务拆分粒度更细，有利于资源重复利用，提高开发效率。

2、可以更加精准的制定每个服务的优化方案，按需伸缩。

3、适用于互联网时代，产品迭代周期更短。

**缺点**：

1、开发的复杂性增加，因为一个业务流程需要多个微服务通过网络交互来完成。

2、微服务过多，服务治理成本高，不利于系统维护。