链滴

组件化、模块化、集中式、分布式、服务化、面向服务的架构、微服务架构、CAP 定论、BASE

作者：xixiaoming

原文链接：https://ld246.com/article/1530068519087

来源网站：链滴

组件化和模块化

中心思想都是**分而治之**。目的都是将一个庞大的系统拆分成多个组件或者说是模块

组件化

将一个大的软件系统按照分离关注点的形式，拆分成多个独立的组件，主要目的就是**减少耦合**。一个立的组件可以是一个软件包、web服务、web资源或者是封装了一些函数的模块。这样，独立出来的件可以单独维护和升级而不会影响到其他的组件。

模块化

目的在于将一个程序按照其功能做拆分，分成相互独立的模块，以便于每个模块只包含与其功能相关内容，模块之间通过接口调用。将一个大的系统模块化之后，每个模块都可以被**高度复用**。

集中式与分布式

集中式

一个主机带多个终端。终端没有数据处理能力，仅负责数据的录入和输出。而运算、存储等全部在主上进行。采用采用单机部署。

分布式

分布式意味着可以采用更多的普通计算机组成分布式集群对外提供服务。把不同的模块部署到不同的器上，各个模块之间通过**远程服务调用**(RPC)等方式进行通信。

服务化

服务化架构使搭建分布式系统成为了可能。服务化是一种粗粒度、松耦合的以服务为中心的架构，服之间通过定义明确的协议和接口进行通信。这里说到的“服务”，本质上来说，就是指“**RPC**”。但在一个复杂的业务环境，如何管理和协同这些大量的RPC才是最麻烦的事情。所以，一般提到的“服化”更多指的是对**RPC的管理**。服务化一般关注服务注册，服务协调，服务可用性，服务通讯协议和容交换等。

微服务架构

通过将功能分散到各个离散的服务中以实现对解决方案的解耦。重点就是业务系统需要彻底的组件化服务化。微服务不再强调传统SOA架构里面比较重的ESB企业服务总线。微服务把所有的“思考”逻包括路由、消息解析等放在服务内部，去掉一个大一统的ESB，服务间轻通信，是比SOA更彻底的拆。

CAP定论

一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项。

C一致性即更新操作成功并返回客户端完成后，所有节点在同一时间的数据完全一致。

A可用性服务一直可用，而且是正常响应时间。

P分区容错性即分布式系统在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性和可性的服务。

1. 对于多数大型互联网应用的场景，一般保证满足P和A，舍弃C（一致性无法保证，退而求其次保证终一致性）。虽然某些地方会影响客户体验，但没达到造成用户流失的严重程度。如原来同步架构的候如果没有库存，就马上告诉客户库存不足无法下单。但在微服务框架下订单和库存可能是两个微服对应两个数据库，用户下单时订单服务是立即生成的，很可能过了一会系统通知你订单被取消掉（最一致性）。就像抢购“小米手机”一样，几十万人在排队，排了很久告诉你没货了，明天再来吧。
2. 对于涉及到钱财这样不能有一丝让步的场景，C必须保证。网络发生故障宁可停止服务，这是保证C，舍弃P。
3. 还有一种是保证CP，舍弃A。例如网络故障事只读不写。

BASE

BASE 是 Basically Available(基本可用)、Soft state(软状态)和 Eventually consistent (最终一致性)个短语的缩写。是对CAP中AP的一个扩展

1. 基本可用:分布式系统在出现故障时，允许损失部分可用功能，保证核心功能可用。
2. 软状态:允许系统中存在中间状态，这个状态不影响系统可用性，这里指的是CAP中的不一致。
3. 最终一致:最终一致是指经过一段时间后，所有节点数据都将会达到一致。

BASE解决了CAP中理论没有网络延迟，在BASE中用软状态和最终一致，保证了延迟后的一致性。BAS和 ACID 是相反的，它完全不同于ACID的强一致性模型，而是通过牺牲强一致性来获得可用性，并许数据在一段时间内是不一致的，但最终达到一致状态。