关于分布式和集群不得不说的故事

--------其实 ta 们不一样

分布式是个啥，集群是个啥？听着都好熟悉的样子。

一、概念扫盲，简单的说：

分布式：一个业务分拆多个子业务，部署在不同的服务器上

集群：同一个业务，部署在多个服务器上

没有懂，再举个例子：

某个小饭店原来只有一个厨师，切菜、备料、炒菜全干。后来客人多了，一个厨师忙不过来，又请了个厨师，两个厨师都能炒一样的菜，这两个厨师的关系是**集群**。

最后，为了让厨师专心炒菜，把菜做到极致，又请了个配菜师负责切菜，备料，厨师只负责炒，厨师和配菜师的关系是**分布式**。

还没懂，再看个正经描述符，说说单机结构，集群结构，分布式结构：（这三者的介绍内容来自网络）

【链接： https://www.zhihu.com/question/20004877/answer/282033178 】

【来源：知乎，版权归原作者】

单机结构

我想大家最最最熟悉的就是单机结构，一个系统业务量很小的时候所有的代码都放在一个项目中就好了，然后这个项目部署在一台服务器上就好了。整个项目所有的服务都由这台服务器提供。这就是单机结构。

那么，单机结构有啥缺点呢？我想缺点是显而易见的，单机的处理能力毕竟是有限的，当你的业务增长到一定程度的时候，单机的硬件资源将无法满足你的业务需求。此时便出现了集群模式，往下接着看。

集群结构

集群模式在程序猿界有各种装逼解释，有的让你根本无法理解，其实就是一个很简单的玩意儿，且听我一一道来。

单机处理到达瓶颈的时候，你就把单机复制几份，这样就构成了一个“集群”。集群中每台服务器就叫做这个集群的一个“节点”，所有节点构成了一个集群。每个节点都提供相同的服务，那么这样系统的处理能力就相当于提升了好几倍（有几个节点就相当于提升了这么多倍）。

但问题是用户的请求究竟由哪个节点来处理呢？最好能够让此时此刻负载较小的节点来处理，这样使得每个节点的压力都比较平均。要实现这个功能，就需要在所有节点之前增加一个“调度者”的角色，用户的所有请求都先交给它，然后它根据当前所有节点的负载情况，决定将这个请求交给哪个节点处理。这个“调度者”有个牛逼了名字——负载均衡服务器。（**负载均衡内容备注在后面**）

集群结构的好处就是系统扩展非常容易。如果随着你们系统业务的发展，当前的系统又支撑不住了，那么给这个集群再增加节点就行了。但是，当你的业务发展到一定程度的时候，你会发现一个问题——无论怎么增加节点，貌似整个集群性能的提升效果并不明显了。这时候，你就需要使用微服务结构了。

分布式结构

先来对前面的知识点做个总结。

从单机结构到集群结构，你的代码基本无需要作任何修改，你要做的仅仅是多部署几台服务器，每台服务器上运行相同的代码就行了。但是，当你要从集群结构演进到微服务结构的时候，之前的那套代码就需要发生较大的改动了。所以对于新系统我们建议，系统设计之初就采用微服务架构，这样后期运维的成本更低。但如果一套老系统需要升级成微服务结构的话，那就得对代码大动干戈了。所以，对于老系统而言，究竟是继续保持集群模式，还是升级成微服务架构，这需要你们的架构师深思熟虑、权衡投入产出比。OK，下面开始介绍所谓的分布式结构。

分布式结构就是将一个完整的系统，按照业务功能，拆分成一个个独立的子系统，在分布式结构中，每个子系统就被称为“服务”。这些子系统能够独立运行在 web 容器中，它们之间通过 RPC 方式通信。（**RPC 注释在后面**）

举个例子，假设需要开发一个在线商城。按照微服务的思想，我们需要按照功能模块拆分成多个独立的服务，如：用户服务、产品服务、订单服务、后台管理服务、数据分析服务等等。这一个个服务都是一个个独立的项目，可以独立运行。如果服务之间有依赖关系，那么通过 RPC 方式调用。

这样的好处有很多：

1）系统之间的耦合度大大降低，可以独立开发、独立部署、独立测试，系统与系统之间的边界非常明确，排错也变得相当容易，开发效率大大提升。

2）系统之间的耦合度降低，从而系统更易于扩展。我们可以针对性地扩展某些服务。假设这个商城要搞一次大促，下单量可能会大大提升，因此我们可以针对性地提升订单系统、产品系统的节点数量，而对于后台管理系统、数据分析系统而言，节点数量维持原有水平即可。

3）服务的复用性更高。比如，当我们将用户系统作为单独的服务后，该公司所有的产品都可以使用该系统作为用户系统，无需重复开发。

备注内容：

1　RPC，即 **Remote Procedure Call**（远程过程调用），说得通俗一点就是：调用远程计算机上的服务，就像调用本地服务一样。

RPC 可基于 HTTP 或 TCP 协议，Web Service 就是基于 HTTP 协议的 RPC，它具有良好的跨平台性，但其性能却不如基于 TCP 协议的 RPC。有两方面因素会直接影响 RPC 的性能，一是传输方式，二是序列化。

众所周知，TCP 是传输层协议，HTTP 是应用层协议，而传输层较应用层更加底层，在数据传输方面，越底层越快，因此，在一般情况下，TCP 一定比 HTTP 快。

【 http://www.importnew.com/20327.html 】

【 http://www.importnew.com/22003.html 】

2 负载均衡

先理解一下所谓的“均衡”，不能狭义地理解为分配给所有实际服务器一样多的工作量，因

为多台服务器的承载能力各不相同，这可能体现在硬件配置、网络带宽的差异，也可能因为某台服务器身兼多职，我们所说的“均衡”，也就是希望所有服务器都不要过载，并且能够最大程序地发挥作用。

负载均衡实现的几种方式：

1)http 重定向

当 http 代理（比如浏览器）向 web 服务器请求某个 URL 后，web 服务器可以通过 http 响应头信息中的 Location 标记来返回一个新的 URL。这意味着 HTTP 代理需要继续请求这个新的 URL，完成自动跳转。

2) DNS 负载均衡

DNS 负责提供域名解析服务，当访问某个站点时，实际上首先需要通过该站点域名的 DNS 服务器来获取域名指向的 IP 地址，在这一过程中，DNS 服务器完成了域名到 IP 地址的映射，同样，这样映射也可以是一对多的，这时候，DNS 服务器便充当了负载均衡调度器，它就像 http 重定向转换策略一样，将用户的请求分散到多台服务器上，但是它的实现机制完全不同。

相比 http 重定向，基于 DNS 的负载均衡完全节省了所谓的主站点，或者说 DNS 服务器已经充当了主站点的职能。但不同的是，作为调度器，DNS 服务器本身的性能几乎不用担心。因为 DNS 记录可以被用户浏览器或者互联网接入服务商的各级 DNS 服务器缓存，只有当缓存过期后才会重新向域名的 DNS 服务器请求解析。也说是 DNS 不存在 http 的吞吐率限制，理论上可以无限增加实际服务器的数量。

3) 反向代理负载均衡

相比前面的 HTTP 重定向和 DNS 解析，反向代理的调度器扮演的是用户和实际服务器中间人的角色：

1、任何对于实际服务器的 HTTP 请求都必须经过调度器

2、调度器必须等待实际服务器的 HTTP 响应，并将它反馈给用户（前两种方式不需要经过调度反馈，是实际服务器直接发送给用户。

4) IP 负载均衡(LVS-NAT)

因为反向代理服务器工作在 HTTP 层，其本身的开销就已经严重制约了可扩展性，从而也限制了它的性能极限。那能否在 HTTP 层面以下实现负载均衡呢？

NAT 服务器:它工作在传输层，它可以修改发送来的 IP 数据包，将数据包的目标地址修改为实际服务器地址。

从 Linux2.4 内核开始，其内置的 Neftilter 模块在内核中维护着一些数据包过滤表，这些表包含了用于控制数据包过滤的规则。可喜的是，Linux 提供了 iptables 来对过滤表进行插入、修改和删除等操作。更加令人振奋的是，Linux2.6.x 内核中内置了 IPVS 模块，它的工作性质类型于 Netfilter 模块，不过它更专注于实现 IP 负载均衡。

5) 直接路由(LVS-DR)

NAT 是工作在网络分层模型的传输层（第四层），而直接路由是工作在数据链路层（第二层），貌似更屌些。它通过修改数据包的目标 MAC 地址（没有修改目标 IP），将数据包转发到实际服务器上，不同的是，实际服务器的响应数据包将直接发送给客户羰，而不经过调度器。

6) ip 隧道

基于 IP 隧道的请求转发机制：将调度器收到的 IP 数据包封装在一个新的 IP 数据包中，转交给实际服务器，然后实际服务器的响应数据包可以直接到达用户端。目前 Linux 大多支持，可以用 LVS 来实现，称为 LVS-TUN，与 LVS-DR 不同的是，实际服务器可以和调度器不在同一个 WANt 网段，调度器通过 IP 隧道技术来转发请求到实际服务器，所以实际服务器也必须拥有合法的 IP 地址。

【链接： http://blog.csdn.net/u014686399/article/details/71156342 这部分内容收集整理于该链接，里面有对以上几种原理的优缺点有描述，这里省略了】
【链接： https://www.zhihu.com/question/22610352 该链接有图描述，更形象】