**分布式中常见概念**

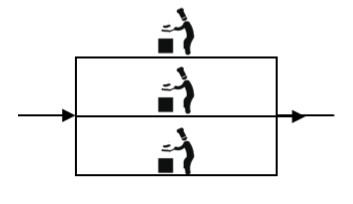
1. **集中式**

所谓集中式系统就是指由一台或多台主计算机组成中心节点，数据集中存储于这个中心节点中，并且整个系统的所有业务单元都集中部署在这个中心节点上，系统所有的功能均由其集中处理。也就是说，集中式系统中，每个终端或客户端及其仅仅负责数据的录入和输出，而数据的存储与控制处理完全交由主机来完成。

集中式系统最大的特点就是部署结构简单，由于集中式系统往往基于底层性能卓越的大型主机，因此无需考虑如何对服务进行多个节点的部署，也就不用考虑多个节点之间的分布式协作问题。

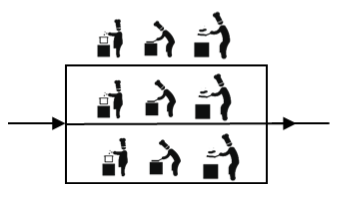
1. **分布式特点**
2. 、组件分布在网络计算机上
3. 、组件之间仅仅通过消息传递来通信并协调行动
4. 、分布性
5. 、对等性
6. 、并发性
7. 、缺乏全局时钟
8. 、故障总是会发生
9. 、处理单点故障（避免单点而对关键就是把这个功能从单机实现变为集群实现）
10. **分布式系统的意义**
11. 、**升级单机处理能力的性价比越来越低**
12. 、**单机处理能力存在瓶颈**
13. 、**出于稳定性和可用性的考虑**
14. **集群**

小饭店原来只有一个厨师，切菜洗菜备料炒菜全干。后来客人多了，厨房一个厨师忙不过来，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



1. **分布式**

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，备菜，备料，厨师和配菜师的关系是分布式，一个配菜师也忙不过来了，又请了个配菜师，两个配菜师关系是集群



1. **节点**

节点是指一个可以独立按照分布式协议完成一组逻辑的程序个体。在具体的项目中，一个节点表示的是一个操作系统上的进程。

1. **副本机制**

副本(replica/copy)指在分布式系统中为数据或服务提供的冗余。

数据副本指在不同的节点上持久化同一份数据，当出现某一个节点的数据丢失时，可以从副本上读取到数据。数据副本是分布式系统中解决数据丢失问题的唯一手段。

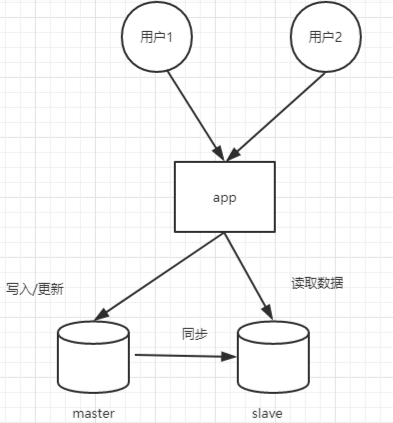
服务副本表示多个节点提供相同的服务，通过主从关系来实现服务的高可用方案

1. **中间件**

中间件位于操作系统提供的服务之外，又不属于应用，他是位于应用和系统层之间为开发者方便的处理通信、输入输出的一类软件，能够让用户关心自己应用的部分。

1. **分布式一致性**

在分布式系统中要解决的一个重要问题就是数据的复制。在我们的日常开发经验中，相信很多开发人员都遇到过这样的问题：在做数据库读写分离的场景中，假设客户端C1将系统中的一个值K由V1更新为V2，但客户端C2无法立即读取到K的最新值，需要在一段时间之后才能读取到。这很正常，因为数据库复制之间存在延时。



所谓的分布式一致性问题，是指在分布式环境中引入数据复制机制之后，不同数据节点之间可能出现的，并无法依靠计算机应用程序自身解决的数据不一致的情况。简单讲，数据一致性就是指在对一个副本数据进行更新的时候，必须确保也能够更新其他的副本，否则不同副本之间的数据将不一致。那么如何去解决这个问题？按照正常的思路，我们可能会想，既然是因为网络延迟导致的问题，那么我们可以把同步动作进行阻塞，用户2在查询的时候必须要等到数据同步完成以后再来做。但是这个方案带来的问题是性能会收到非常大的影响。如果同步的数据比较多或者比较频繁，那么因为阻塞操作可能将导致整个新系统不可用的情况；

总结：所以我们没有办法找到一种能够满足数据一致性、又不影响系统运行的性能的方案，所以这个地方就诞生了一个一致性的级别：

1. 强一致性：这种一致性级别是最符合用户直觉的，它要求系统写入什么，读出来的也会是什么，用户体验好，但实现起来往往对系统的性能影响大
2. 弱一致性：这种一致性级别约束了系统在写入成功后，不承诺立即可以读到写入的值，也不久承诺多久之后数据能够达到一致，但会尽可能地保证到某个时间级别（比如秒级别）后，数据能够达到一致状态
3. 最终一致性：最终一致性是弱一致性的一个特例，系统会保证在一定时间内，能够达到一个数据一致的状态。这里之所以将最终一致性单独提出来，是因为它是弱一致性中非常推崇的一种一致性模型，也是业界在大型分布式系统的数据一致性上比较用的多的模型
4. **分布式事务**

这是一个老生常谈的问题，我们都知道事务就是一些列操作的原子性保证，在单机的情况下，我们能够依靠本机的数据库连接和组件轻易做到事务的控制，但是分布式情况下，业务原子性操作很可能是跨服务的，这样就导致了分布式事务，例如A和B操作分别是不同服务下的同一个事务操作内的操作，A调用B，A如果可以清楚的知道B是否成功提交从而控制自身的提交还是回滚操作，但是在分布式系统中调用会出现一个新状态就是超时，就是A无法知道B是成功还是失败，这个时候A是提交本地事务还是回滚呢？其实这是一个很难的问题，如果强行保证事务一致性，可以采取分布式锁，但是那样会增加系统复杂度而且会增大系统的开销，而且事务跨越的服务越多，消耗的资源越大，性能越低，所以最好的解决方案就是避免分布式事务。还有一种解决方案就是重试机制，但是重试如果不是查询接口，必然涉及到数据库的变更，如果第一次调用成功但是没返回成功结果，那调用方第二次调用对调用方来说依然是重试，但是对于被调用方来说是重复调用，例如A向B转账，A-100,B + 100，这样会导致A扣了100，而B增加200。这样的结果不是我们期望的，因此需在要写入的接口做幂等设计。多次调用和单次调用是一样的效果。通常可以设置一个唯一键，在写入的时候查询是否已经存在，避免重复写入。但是幂等设计的一个前提就是服务是高可用，否则无论怎么重试都不能调用返回一个明确的结果调用方会一直等待，虽然可以限制重试的次数，但是这已经进入了异常状态了，甚至到了极端情况还是需要人肉补偿处理。其实根据CAP和BASE理论，不可能在高可用分布式情况下做到一致性，一般都是最终一致性保证。

1. **CAP理论**

一个分布式系统不可能同时满足一致性（C：Consistency）、可用性（A：Availability）和分区容错性（P：Partition tolerance）这三个基本需求，最多只能同时满足其中两项。

1. 一致性

所有节点上的数据时刻保持同步

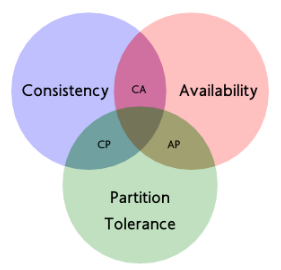
1. 可用性

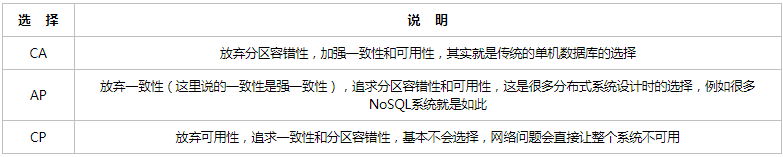
所有节点上的数据时刻保持同步

1. 分区容错性

分区容错性约束了一个分布式系统具有如下特性：分布式系统在遇到任何网络分区故障的时候，仍然需要能够保证对外提供满足一致性和可用性的服务，除非是整个网络环境都发生了故障

系统应该持续提供服务，即时系统内部（某个节点分区）有消息丢失。比如交换机失败、网址网络被分成几个子网，形成脑裂；服务器发生网络延迟或死机，导致某些server与集群中的其他机器失去联系





需要明确的一点是，对于一个分布式系统而言，分区容错性是一个最基本的要求。因为既然是一个分布式系统，那么分布式系统中的组件必然需要被部署到不同的节点，否则也就无所谓分布式系统了，因此必然出现子网络。而对于分布式系统而言，网络问题又是一个必定会出现的异常情况，因此分区容错性也就成为了一个分布式系统必然需要面对和解决的问题。因此系统架构师往往需要把精力花在如何根据业务特点在C（一致性）和A（可用性）之间寻求平衡。

1. **BASE理论**

BASE是Basically Available（基本可用）、Soft state（软状态）和Eventually consistent（最终一致性）三个短语的缩写。BASE理论是对CAP中一致性和可用性权衡的结果，其来源于对大规模互联网系统分布式实践的总结，是基于CAP定理逐步演化而来的。

BASE理论的核心思想是：即使无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。接下来看一下BASE中的三要素：

1. 基本可用

基本可用是指分布式系统在出现不可预知故障的时候，允许损失部分可用性----注意，这绝不等价于系统不可用。比如：

（1）响应时间上的损失。正常情况下，一个在线搜索引擎需要在0.5秒之内返回给用户相应的查询结果，但由于出现故障，查询结果的响应时间增加了1~2秒

（2）系统功能上的损失：正常情况下，在一个电子商务网站上进行购物的时候，消费者几乎能够顺利完成每一笔订单，但是在一些节日大促购物高峰的时候，由于消费者的购物行为激增，为了保护购物系统的稳定性，部分消费者可能会被引导到一个降级页面

1. 软状态

软状态指允许系统中的数据存在中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延时

1. 最终一致性

最终一致性强调的是所有的数据副本，在经过一段时间的同步之后，最终都能够达到一个一致的状态。因此，最终一致性的本质是需要系统保证最终数据能够达到一致，而不需要实时保证系统数据的强一致性。

总的来说，BASE理论面向的是大型高可用可扩展的分布式系统，和传统的事物ACID特性是相反的，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，并允许数据在一段时间内是不一致的，但最终达到一致状态。但同时，在实际的分布式场景中，不同业务单元和组件对数据一致性的要求是不同的，因此在具体的分布式系统架构设计过程中，ACID特性和BASE理论往往又会结合在一起。

1. **分布式环境各种问题**
2. 通信异常

从集中式向分布式演变的过程中，必然引入网络因素，由于网络本身的不可靠性，因此也引入了额外的问题。分布式系统需要在各个节点之间进行网络通信，因此每次网络通信都会伴随着网络不可用的风险，网络光纤、路由器或是DNS等硬件设备或是系统不可用都会导致最终分布式系统无法顺利完成一次网络通信。另外，即使分布式系统各个节点之间的网络通信能够正常进行，其延时也会大于单机操作。通常我们认为现代计算机体系结构中，单机内存访问的延时在纳秒数量级（通常是10ns），而正常的一次网络通信的延迟在0.1~1ms左右（相当于内存访问延时的105倍），如此巨大的延时差别，也会影响到消息的收发过程，因此消息丢失和消息延迟变得非常普遍

1. 网络分区

当网络由于发生异常情况，导致分布式系统中部分节点之间的网络延时不断增大，最终导致组成分布式系统的所有节点中，只有部分节点之间能够正常通信，而另一些节点则不能----我们将这个现象称为网络分区

1. 三态

上面两点，我们已经了解到在分布式环境下，网络可能会出现各式各样的问题，因此分布式系统的每一次请求与响应，存在特有的三态概念，即成功、失败、超时。在传统的单机系统中，应用程序在调用一个函数之后，能够得到一个非常明确的响应：成功或失败。而在分布式系统中，由于网络是不可靠的，虽然在绝大部分情况下，网络通信也能够接受到成功或失败的响应，当时当网络出现异常的情况下，就可能会出现超时现象，通常有以下两种情况：

（1）由于网络原因，该请求并没有被成功地发送到接收方，而是在发送过程中就发生了消息丢失现象

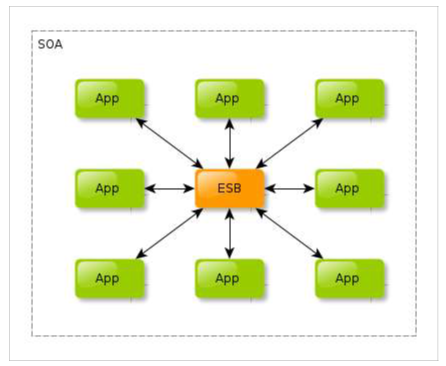
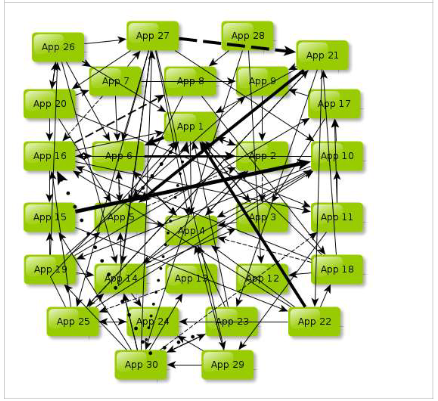
（2）该请求成功地被接收方接收后，进行了处理，但是在将响应反馈给发送方的过程中，发生了消息丢失现象

1. 节点故障

节点故障则是分布式环境下另一个比较常见的问题，指的是组成分布式系统的服务器节点出现的宕机或"僵死"现象，通常根据经验来说，每个节点都有可能出现故障，并且每天都在发生

1. **SOA架构**

SOA全称（Service Oriented Architecture），中文意思为“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跟SOA相提并论的还有一个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；



1. **微服务架构**

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。组件表示一个可以独立更换和升级的单元，就像PC中的CPU、内存、显卡、硬盘一样，独立且可以更换升级而不影响其他单元。如果我们把PC作为组件以服务的方式构建，那么这台PC只需要维护主板和一些必要的外部设备。CPU、内存、硬盘都是以组件方式提供服务，PC需要调用CPU做计算处理，只需要知道CPU这个组件的地址即可。

微服务的特征

1. 通过服务实现组件化
2. 按业务能力来划分服务和开发团队
3. 去中心化
4. 基础设施自动化（develops、自动化部署）
5. **领域驱动设计及业务驱动划分**

领域驱动设计（DDD,Domain-Driven Design），软件开发不是一蹴而就的事情，我们不可能在不了解产品（或行业领域）的前提下进行软件开发，在开发前，通常需要进行大量的业务知识梳理，然后才到软件设计的层面，最后才是开发。而在业务知识梳理的过程中，我们必然会形成某个领域知识，根据领域知识来一步步驱动软件设计，就是领域驱动设计的基本概念。

领域驱动设计其实我们可以简单认为是一种指导思想，是一种软件开发方法，通过DDD可以将系统解构更加合理，最终满足高内聚低耦合的本质。

1. **什么是分布式架构下的高可用设计**
2. 避免单点故障
3. 负载均衡技术（failover/选址/硬件负载/软件负载/去中心化的软件负载（gossip(redis-cluster)））
4. 热备（linuxHA）
5. 多机房（同城灾备、异地灾备）
6. 应用的高可用性
7. 故障监控（系统监控（cpu、内存）/链路监控/日志监控）自动预警
8. 应用的容错设计、（服务降级、限流）自我保护能力
9. 数据量（数据分片、读写分离
10. 可伸缩设计
11. 垂直伸缩 提升硬件能力
12. 水平伸缩 增加服务器
13. 加速静态内容访问的CDN
14. **大型网站发展史**

（1）从一个小网站发展起来，一台服务器，应用程序、数据库、文件等所有资源都在一台服务器上

（2）网站业务的发展，一台服务器逐渐不能满足需求，因此要将应用和数据分离，应用和数据分离后使用三台服务器：应用服务器、文件服务器和数据库服务器

（3）网站进一步发展，数据库压力太大导致访问延迟，因此使用缓存该改善网站性能（记住，使用缓存是改善网站性能的第一步），网站使用的缓存分为两种：缓存在应用服务器上的本地缓存和缓存在专门的分布式缓存服务器上的远程缓存

（4）使用缓存，数据库访问压力得到有效缓解，但是在网站访问高峰期应用服务器还是成为了整个网站的瓶颈。这种时候要理解，不要企图去更换更强大的服务器，对大型网站而言，不管多么强大的服务器，都满足不了网站持续增长的业务需求，因此可以通过增加服务器的方式改善负载压力，再通过负载均衡调度服务器，将来自用户浏览器的访问请求分发到应用服务器集群中的某台服务器上

（5）虽然使用缓存可以使大部分数据不走数据库，但是缓存没有命中、缓存过期的数据还是会走数据库，网站达到一定规模之后，数据库读写压力还是会很大，成为网站的瓶颈。此时可以使用数据库读写分离来改善数据库负载压力，应用服务器写数据走写库，应用服务器读数据走读库，目前大部分主流数据库都提供主从热备功能，通过配置两台数据库主从关系，可以将一台数据库服务器的数据更新同步到另一台服务器上

（6）随着网站业务继续发展，用户规模不断增大，由于中国复杂的网络环境，不同地区用户访问网站时，速度差别也极大。因此可以使用反向代理和CDN，一方面加快用户访问速度，另一方面减轻后端服务器的负载压力，因为反向代理和CDN的基本原理都是缓存

（7）数据库经过读写分离后，由一台服务器拆分为两台服务器，但还是不能满足网站业务量的需求，因此可以使用分布式数据库，主要拆分手段是业务分库，将不同的业务数据部署在不同的物理服务器上

（8）大型网站为了应对日益复杂的业务场景，可以使用分而治之的手段将整个网站的业务拆分成不同的应用，每个应用独立部署，可以通过超级链建立关系，也可以通过消息队列进行数据分发

大型网站发展到这里，基本上大多数的技术问题都得以解决

1. **相关知识点**
2. 高性能网站的关键：控制住并发的量。只要能做到这点，很多棘手的数据问题也就不是什么问题了
3. 不要企图通过技术解决所有问题，业务的问题也可以通过业务手段去解决
4. 计算机软件发展的 一个重要目标和驱动力是降低软件的耦合性，事物之间关系越少，就越少彼此影响，越可以独立发展
5. 异步架构是典型的生产者消费者模式
6. 使用异步队列有几个好处
7. 、提高系统的可用性
8. 、加快网站的访问速度
9. 、消除并发访问高峰
10. 网站伸缩性就是指通过不断向集群中加入服务器的手段来缓解不断上升的用户并发访问压力和不断增长的数据存储需求
11. 衡量架构伸缩性的指标
12. 、是否可用多台服务器架构集群
13. 、是否容易向服务器中添加新的服务器
14. 、加入的服务器是否可以提供和原服务器无差别的服务
15. 、集群中容纳的总服务器是否有限制
16. 反应系统忙闲的重要指标Load

System Load也称为Load，即系统负载，指当前正在被CPU执行和等待被CPU执行的线程数之和，是反映系统忙闲程度的重要指标。多核CPU下，完美情况应该是所有CPU都在用，没有线程等待处理。Load值低于CPU数目，表示CPU有空闲，资源存在浪费；Load值高于CPU数目，表示进程在等待CPU调度，资源存在不足

1. 浏览器访问优化手段
2. 、减少http请求，合并CSS、JS、图片，不要发起多次http请求去拿这些数据
3. 、使用浏览器缓存，存储静态资源，Cache-Control、Expires、Pragma、Last\_Modified都是和缓存的HTTP HEADER
4. 、启用压缩，有效减低通信传输的数据量
5. 、CSS放在上面、JS放在下面，因为JS下载会立即执行，可能阻塞页面加载速度
6. 、减少Cookie传输
7. 缓存使用的几点细节
8. 、频繁修改的数据不要写缓存，一般读写比例至少2:1以上才会做缓存，即一次写入至少有两次以上的读取
9. 、如果访问没有热点，大部分数据的访问没有集中在小部分数据上，那么做缓存就没有意义，因为缓存有失效机制，大部分数据还没有被再次访问就被挤出缓存了
10. 、容忍一定时间的数据不一致，除非数据更新时立刻通知缓存，不过这也会带来开销与事物一致性的问题
11. 、使用分布式缓存集群以提高缓存可用性
12. 、新启动的缓存没有任何数据，在重建缓存的过程中，系统性能与数据库负载都不太好，因此要根据项目、根据业务，将一部分数据在启动时就加载好，这就是缓存预热
13. 、对缓存做无效参数并设置失效时间，避免不恰当的业务或恶意攻击频繁调用接口查询数据库，一旦某一个Key值数据库里面查不到数据就进入无效缓存，一段时间内再次访问这个Key值无数据返回
14. 消息队列具有很好的削峰作用（前面提过），不过注意需要适当修改业务流程进行配合

通过异步处理，将短时间高并发产生的事物消息存储在消息队列中，从而可以削平高峰期的并发事物。不过要注意点，由于数据写入消息队列后立即返回给用户，数据在后续的业务校验、写数据库等操作可能失败，因此在使用消息队列进行业务异步处理后，，需要适当修改业务流程进行配合，如订单提交之后，订单数据写入消息队列，不能立即返回给用户订单提交成功，需要在消息队列的订单消费者进程真正处理完该订单的，甚至商品出库后，再通过电子邮件或者SMS消息通知用户订单成功，避免交易纠纷

1. CDN和反向代理的基本原理都是缓存，区别在于CDN部署在网络提供商的机房，使用户在请求网站服务时，可以从距离自己最近的网络提供商机房获取数据；反向代理则部署在网站的中心机房，当用户请求到达中心机房时，首先访问的是反向代理服务器，如果反向代理服务器中缓存着用户请求的资源，就将其直接返回给用户。使用CDN和反向代理的目的是一样的：
2. 、尽早返回速度给用户，加快用户访问网站的速度
3. 、减轻后端服务器的负载压力
4. 应用服务器集群的Session管理

单机环境下，Session可由部署在服务器上的Web容器管理，在使用负载均衡的集群环境中，由于负载均衡服务器可能会将请求分发到集群任何一台应用服务器上，所以保证每次请求依然能够获得正确的Session比单机时要复杂得多。集群环境下，Session管理有以下手段：

1. 、Session复制

在集群中的几台服务器之间同步Session对象，使得每台服务器上都保存着所有用户的Session信息，这样任何一台机器宕机都不会导致Session数据的丢失，而服务器使用Session时，也只需要从本机获取即可。

这种方案虽然简单，从本机读取Session信息也很快，但只能使用在集群规模比较小的情况下。当集群规模较大时，集群服务器之间需要大量的通信进行Session复制，占用服务器和网络的大量资源，系统不堪负担。而且由于所有用户的Session在每台服务器都有备份，在大量用户访问的情况下，甚至会出现服务器内存不够Session使用的情况。

大型网站的核心应用集群就是数千台服务器，同时在线用户达到数千万，并不适用这种方案。

1. 、Session绑定

Session绑定可以利用负载均衡的原地址Hash算法，负载均衡服务器总是将来源于同一IP的请求分发到同一台服务器。这样在整个会话期间，用户所有请求都在同一台服务器上处理，即Session绑定在某台特定服务器上，保证Session总能在这台服务器上获取，这种方法又被称为会话粘滞。

但是Session绑定的方案不符合对系统高可用的需求，因为一旦某台服务器宕机，那么该机器上的Session也就不复存在了，用户请求切换到其他机器后因为没有Session而无法完成业务处理。因此虽然大部分负载均衡服务器都提供原地址负载均衡算法，但很少有网站利用这个算法进行Session管理

1. 、利用Cookie记录Session

早期的企业应用使用C/S架构，一种管理Session的方式是将Session记录在客户端，每次请求服务器的时候，将Session放在请求中发送给服务器，服务器处理完请求后再将修改过的Session响应给客户端。网站没有客户端，但是可以利用浏览器支持的Cookie记录Session。

利用Cookie记录Session也有一些缺点：

（1）受到Cookie大小限制，能记录的信息有限

（2）每次请求和响应都携带Cookie，影响性能

（3）如果用户关闭Cookie，访问就会不正常

不过因为Cookie简单易用且可用性高，支持应用服务器的线性伸缩，而大部分应用需要记录的Session信息有比较小，因此事实上，许多网站或多 或少都会使用Cookie记录Session

1. 、Session服务器

利用独立部署的Session服务器（集群）统一管理Session，应用服务器每次读写Session时，都访问Session服务器。

这种解决方案事实上是将应用服务器的状态分离，分为无状态的应用服务器和有状态的Session服务器，然后针对这两种服务器的不同特性分别设计其架构。

对于有状态的Session服务器，一种比较简单的方法是利用分布式缓存、数据库等，在这些产品的基础上进行包装，使其符合Session的存储和访问要求；如果业务场景对Session管理有比较高的要求，那么就要利用Session服务集成单点登录（SSO）、用户服务等功能，则需要开发专门的Session服务管理平台。