# 背景

面对大量用户访问、高并发请求，海量数据，可以使用高性能的服务器、大型数据库，存储设备，高性能Web服务器，采用高效率的编程语言比如(Go，Scala)等，当单机容量达到极限时，我们需要考虑业务拆分和分布式部署，来解决大型网站访问量大，并发量高，海量数据的问题。

从单机网站到分布式网站，很重要的区别是业务拆分和分布式部署，将应用拆分后，部署到不同的机器上，实现大规模分布式系统。分布式和业务拆分解决了，从集中到分布的问题，但是每个部署的独立业务还存在单点的问题和访问统一入口问题，为解决单点故障，我们可以采取冗余的方式。将相同的应用部署到多台机器上。解决访问统一入口问题，我们可以在集群前面增加负载均衡设备，实现流量分发。

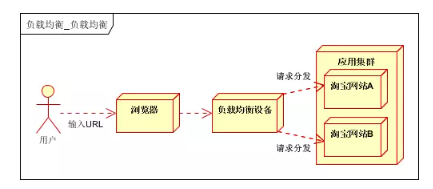
负载均衡（Load Balance），意思是将负载（工作任务，访问请求）进行平衡、分摊到多个操作单元（服务器，组件）上进行执行。**是解决高性能，单点故障（高可用），扩展性（水平伸缩）的终极解决方案**。可以理解为，负载均衡是高可用和高并发共同使用的一种技术。

在实际应用中，在 Web 服务器集群之前总会有一台负载均衡服务器，负载均衡设备的任务就是作为 Web 服务器流量的入口，挑选最合适的一台 Web 服务器，将客户端的请求转发给它处理，实现客户端到真实服务端的透明转发。

最近几年很火的「云计算」以及分布式架构，本质上也是将后端服务器作为计算资源、存储资源，由某台管理服务器封装成一个服务对外提供，客户端不需要关心真正提供服务的是哪台机器，在它看来，就好像它面对的是一台拥有近乎无限能力的服务器，而本质上，真正提供服务的是后端的集群。

# 原理

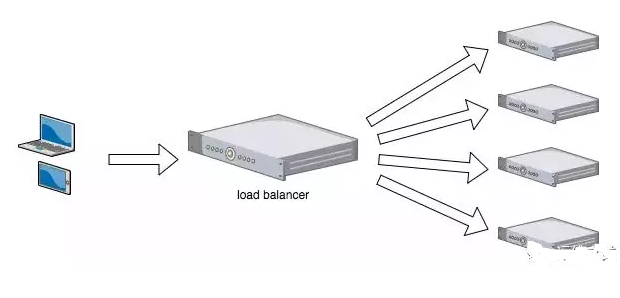
系统的扩展可分为纵向（垂直）扩展和横向（水平）扩展。纵向扩展，是从单机的角度通过增加硬件处理能力，比如CPU处理能力，内存容量，磁盘等方面，实现服务器处理能力的提升，不能满足大型分布式系统（网站），大流量，高并发，海量数据的问题。因此需要采用横向扩展的方式，通过添加机器来满足大型网站服务的处理能力。比如：一台机器不能满足，则增加两台或者多台机器，共同承担访问压力。这就是典型的集群和负载均衡架构：如下图：



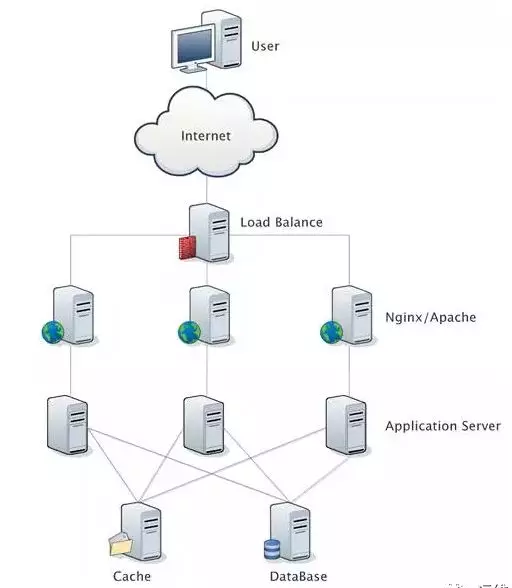
**应用集群：**将同一应用部署到多台机器上，组成处理集群，接收负载均衡设备分发的请求，进行处理，并返回相应数据。

**负载均衡设备：**将用户访问的请求，根据负载均衡算法，分发到集群中的一台处理服务器。（一种把网络请求分散到一个服务器集群中的可用服务器上去的设备）

软件负载解决的两个核心问题是：选谁、转发，其中最著名的是 LVS（Linux Virtual Server）。



一个典型的互联网应用架构：



# 作用

负载均衡的作用（解决的问题）：

1、解决**并发压力**，提高应用处理性能（增加吞吐量，加强网络处理能力）；

2、提供**故障转移**，实现**高可用**；

3、通过添加或减少服务器数量，提供网站伸缩性（扩展性）；

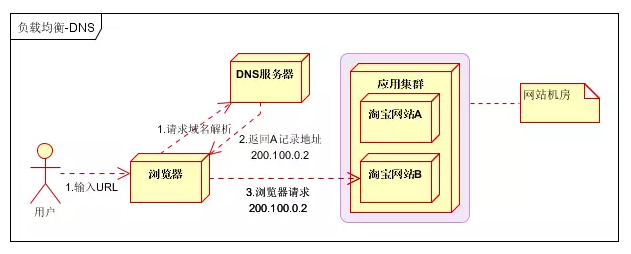
4、安全防护；（负载均衡设备上做一些过滤，黑白名单等处理）

# 分类

根据实现技术不同，可分为DNS负载均衡，HTTP负载均衡，IP负载均衡，链路层负载均衡等。

## DNS负载均衡

最早的负载均衡技术，利用域名解析实现负载均衡，在DNS服务器，配置多个A记录，这些A记录对应的服务器构成集群。大型网站总是部分使用DNS解析，作为第一级负载均衡。如下图：



**优点：**

1、使用简单：负载均衡工作，交给DNS服务器处理，省掉了负载均衡服务器维护的麻烦；

2、提高性能：可以支持基于地址的域名解析，解析成距离用户最近的服务器地址，可以加快访问速度，改善性能；

**缺点：**

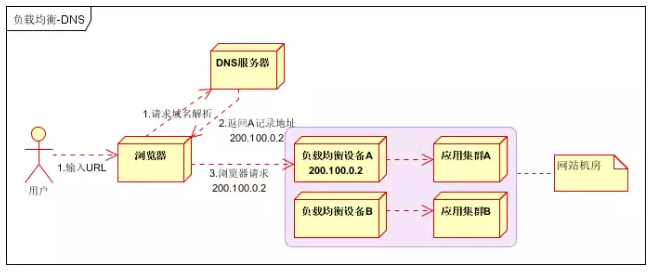
1、可用性差：DNS解析是多级解析，新增/修改DNS后，解析时间较长；解析过程中，用户访问网站将失败；

2、扩展性低：DNS负载均衡的控制权在域名商那里，无法对其做更多的改善和扩展；

3、维护性差：也不能反映服务器的当前运行状态；支持的算法少；不能区分服务器的差异（不能根据系统与服务的状态来判断负载）

实践建议：

**将DNS作为第一级负载均衡**，A记录对应着内部负载均衡的IP地址，通过内部负载均衡将请求分发到真实的Web服务器上。一般用于互联网公司，复杂的业务系统不合适使用。如下图：

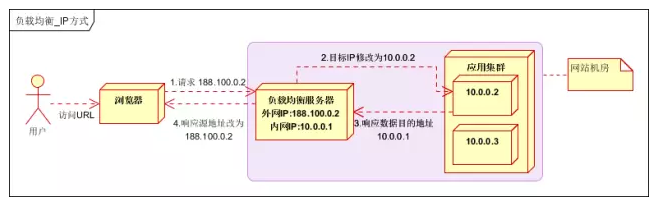


## IP负载均衡

在网络层通过修改请求目标地址进行负载均衡。

用户请求数据包，到达负载均衡服务器后，负载均衡服务器在操作系统内核进程获取网络数据包，根据负载均衡算法得到一台真实服务器地址，然后将请求目的地址修改为获得的真实ip地址，不需要经过用户进程处理。

真实服务器处理完成后，响应数据包回到负载均衡服务器，负载均衡服务器，再将数据包源地址修改为自身的ip地址，发送给用户浏览器。如下图：



IP负载均衡，真实物理服务器返回给负载均衡服务器，存在两种方式：

（1）负载均衡服务器在修改目的ip地址的同时修改源地址。将数据包源地址设为自身盘，即源地址转换（snat）。

（2）将负载均衡服务器同时作为真实物理服务器集群的网关服务器。

**优点：**

在内核进程完成数据分发，比在应用层分发性能更好；

**缺点：**

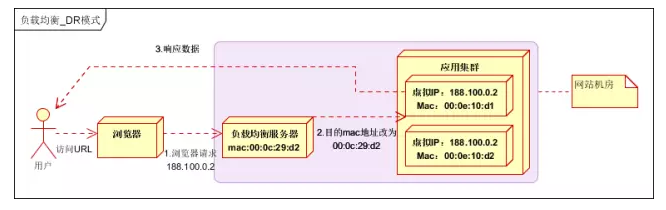
所有请求响应都需要经过负载均衡服务器，集群最大吞吐量受限于负载均衡服务器网卡带宽；

## 链路层负载均衡

在通信协议的数据链路层修改mac地址，进行负载均衡。

数据分发时，不修改ip地址，指修改目标mac地址，配置真实物理服务器集群所有机器虚拟ip和负载均衡服务器ip地址一致，达到不修改数据包的源地址和目标地址，进行数据分发的目的。

实际处理服务器ip和数据请求目的ip一致，不需要经过负载均衡服务器进行地址转换，可将响应数据包直接返回给用户浏览器，避免负载均衡服务器网卡带宽成为瓶颈。也称为直接路由模式（DR模式）。如下图：



**优点：**性能好；

**缺点：**配置复杂；

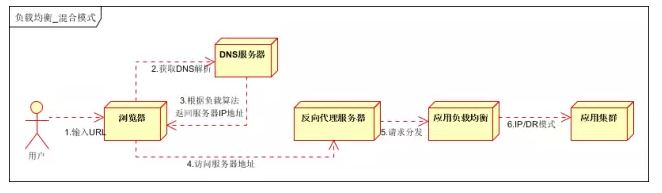
实践建议：DR模式是目前使用最广泛的一种负载均衡方式。

## 混合型负载均衡

由于多个服务器群内硬件设备、各自的规模、提供的服务等的差异，可以考虑给每个服务器群采用最合适的负载均衡方式，然后又在这多个服务器群间再一次负载均衡或群集起来以一个整体向外界提供服务（即把这多个服务器群当做一个新的服务器群），从而达到最佳的性能。将这种方式称之为混合型负载均衡。

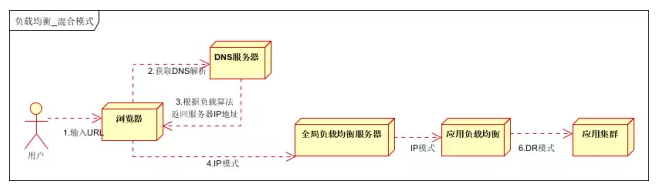
此种方式有时也用于单台均衡设备的性能不能满足大量连接请求的情况下。是目前大型互联网公司，普遍使用的方式。

方式一，如下图：



以上模式适合有动静分离的场景，反向代理服务器（集群）可以起到缓存和动态请求分发的作用，当是静态资源缓存在代理服务器时，则直接返回到浏览器。如果动态页面则请求后面的应用负载均衡（应用集群）。

方式二，如下图：



以上模式，适合动态请求场景。

因混合模式，可以根据具体场景，灵活搭配各种方式，以上两种方式仅供参考。

## ----

负载均衡就是一种计算机网络技术，用来在多个计算机（计算机集群）、网络连接、CPU、磁碟驱动器或其它资源中分配负载，以达到最佳化资源使用、最大化吞吐率、最小化响应时间、同时避免过载的目的。那么，这种计算机技术的实现方式有多种。

大致可以分为以下几种，其中最常用的是四层和七层负载均衡：

## 二层负载均衡

负载均衡服务器对外依然提供一个 VIP（虚IP），集群中不同的机器采用相同IP地址，但机器的MAC地址不一样。当负载均衡服务器接受到请求之后，通过改写报文的目标MAC地址的方式将请求转发到目标机器实现负载均衡。

## 三层负载均衡

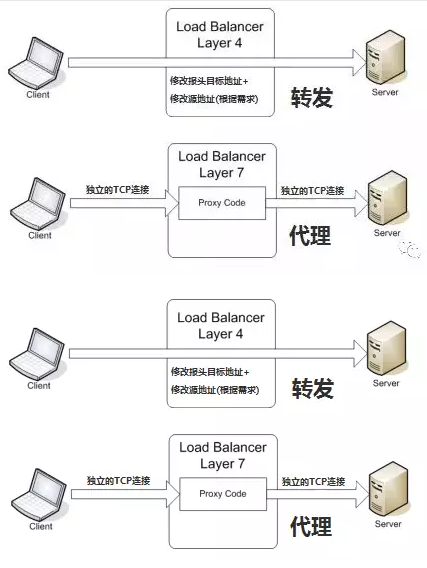
和二层负载均衡类似，负载均衡服务器对外依然提供一个VIP（虚IP），但集群中不同的机器采用不同的IP地址。当负载均衡服务器接受到请求之后，根据不同的负载均衡算法，通过IP将请求转发至不同的真实服务器。

## 四层负载均衡

四层负载均衡工作在OSI模型的传输层，由于在传输层，只有TCP/UDP协议，这两种协议中除了包含源IP、目标IP以外，还包含源端口号及目的端口号。四层负载均衡服务器在接受到客户端请求后，以后通过修改数据包的地址信息（IP+端口号）将流量转发到应用服务器。

## 七层负载均衡

七层负载均衡工作在OSI模型的应用层，应用层协议较多，常用HTTP、Radius、DNS 等。七层负载就可以基于这些协议来负载。这些应用层协议中会包含很多有意义的内容。比如同一个Web服务器的负载均衡，除了根据IP加端口进行负载外，还可根据七层的URL、浏览器类别、语言来决定是否要进行负载均衡。



图：四层和七层负载均衡

对于一般的应用来说，有了Nginx就够了。**Nginx可以用于七层负载均衡**。但是对于一些大的网站，一般会采用**DNS+四层负载+七层负载**的方式进行多层次负载均衡。

# 算法

负载均衡服务器在决定将请求转发到具体哪台真实服务器时，是通过负载均衡算法来实现的。负载均衡算法可以分为两类：静态负载均衡算法和动态负载均衡算法。

静态负载均衡算法包括：轮询、比率、优先权。

动态负载均衡算法包括：最少连接数、最快响应速度、观察方法、预测法、动态性能分配、动态服务器补充、服务质量、服务类型、规则模式。

常用的负载均衡算法有，轮询，随机，最少链接，源地址散列，加权等方式；

## 轮询

轮询（Round Robin）：顺序循环将请求一次顺序循环地连接每个服务器。当其中某个服务器发生第二到第7层的故障，BIG-IP 就把其从顺序循环队列中拿出，不参加下一次的轮询，直到其恢复正常。

以轮询的方式依次请求调度不同的服务器；实现时，一般为服务器带上权重；这样有两个好处：针对服务器的性能差异可分配不同的负载；当需要将某个结点剔除时，只需要将其权重设置为0即可；

将所有请求，依次分发到每台服务器上，适合服务器硬件同相同的场景。

**优点：**服务器请求数目相同；实现简单、高效；易水平扩展

**缺点：**服务器压力不一样，不适合服务器配置不同的情况；请求到目的结点的不确定，造成其无法适用于有写的场景（缓存，数据库写）

**应用场景：**数据库或应用服务层中只有读的场景

## 随机

随机方式：请求随机分布到各个结点；在数据足够大的场景能达到一个均衡分布。

**优点：**使用简单；易水平扩展

**缺点：**不适合机器配置不同的场景；同Round Robin，无法用于有写的场景

**应用场景：**数据库负载均衡，也是只有读的场景

## 最少链接

将请求分配到连接数最少的服务器（目前处理请求最少的服务器）。

**优点：**根据服务器当前的请求处理情况，动态分配；

**缺点：**算法实现相对复杂，需要监控服务器请求连接数；

## Hash（源地址散列）

源地址散列：根据IP地址进行Hash计算，得到IP地址。

**优点：**将来自同一IP地址的请求，同一会话期内，转发到相同的服务器；实现会话粘滞。

**缺点：**目标服务器宕机后，会话会丢失；

根据 key 来计算需要落在的结点上，可以保证一个同一个键一定落在相同的服务器上。

**优点：**相同 key 一定落在同一个结点上，这样就可用于有写有读的缓存场景

**缺点：**在某个结点故障后，会导致哈希键重新分布，造成命中率大幅度下降

**解决：**一致性哈希 or 使用 keepalived 保证任何一个结点的高可用性，故障后会有其它结点顶上来

**应用场景：**缓存，有读有写

## 一致性哈希

在服务器一个结点出现故障时，受影响的只有这个结点上的 key，最大程度的保证命中率；如 twemproxy 中的 ketama方案；生产实现中还可以规划指定子 key 哈希，从而保证局部相似特征的键能分布在同一个服务器上；

**优点：**结点故障后命中率下降有限

**应用场景：**缓存

## 根据键的范围来负载

根据键的范围来负载，前 1 亿个键都存放到第一个服务器，1~2 亿在第二个结点。

**优点：**水平扩展容易，存储不够用时，加服务器存放后续新增数据

**缺点：**负载不均；数据库的分布不均衡；

（数据有冷热区分，一般最近注册的用户更加活跃，这样造成后续的服务器非常繁忙，而前期的结点空闲很多）

**适用场景：**数据库分片负载均衡

## 根据键对服务器结点数取模来负载

根据键对服务器结点数取模来负载；比如有 4 台服务器，key 取模为 0 的落在第一个结点，1 落在第二个结点上。

**优点：**数据冷热分布均衡，数据库结点负载均衡分布；

**缺点：**水平扩展较难；

**适用场景：**数据库分片负载均衡

## 纯动态结点负载均衡

根据 CPU、IO、网络的处理能力来决策接下来的请求如何调度。

**优点：**充分利用服务器的资源，保证个结点上负载处理均衡

**缺点：**实现起来复杂，真实使用较少

## 不用主动负载均衡

使用消息队列转为异步模型，将负载均衡的问题消灭；负载均衡是一种推模型，一直向你发数据，那么将所有的用户请求发到消息队列中，所有的下游结点谁空闲，谁上来取数据处理；转为拉模型之后，消除了对下行结点负载的问题。

**优点：**通过消息队列的缓冲，保护后端系统，请求剧增时不会冲垮后端服务器；水平扩展容易，加入新结点后，直接取 queue 即可；

**缺点：**不具有实时性；

**应用场景：**不需要实时返回的场景；比如，12036 下订单后，立刻返回提示信息：您的订单进去排队了...等处理完毕后，再异步通知；

## 加权

在轮询，随机，最少链接，Hash等算法的基础上，通过加权的方式，进行负载服务器分配。

**优点：**根据权重，调节转发服务器的请求数目；

**缺点：**使用相对复杂；

## 比率（Ratio）

给每个服务器分配一个加权值为比例，根椐这个比例，把用户的请求分配到每个服务器。当其中某个服务器发生第 2 到第 7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。

## 优先权（Priority）

给所有服务器分组，给每个组定义优先权，BIG-IP 用户的请求，分配给优先级最高的服务器组（在同一组内，采用轮询或比率算法，分配用户的请求）；当最高优先级中所有服务器出现故障，BIG-IP 才将请求送给次优先级的服务器组。这种方式，实际为用户提供一种热备份的方式。

## 最少的连接方式（Least Connection）

传递新的连接给那些进行最少连接处理的服务器。当其中某个服务器发生第 2 到第 7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。

## 最快模式（Fastest）

传递连接给那些响应最快的服务器。当其中某个服务器发生第二到第7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。

## 观察模式（Observed）

连接数目和响应时间以这两项的最佳平衡为依据为新的请求选择服务器。当其中某个服务器发生第二到第 7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。

## 预测模式（Predictive）

BIG-IP 利用收集到的服务器当前的性能指标，进行预测分析，选择一台服务器在下一个时间片内，其性能将达到最佳的服务器相应用户的请求。(被 BIG-IP 进行检测)

## 动态性能分配(Dynamic Ratio-APM)

BIG-IP 收集到的应用程序和应用服务器的各项性能参数，动态调整流量分配。

## 动态服务器补充(Dynamic Server Act.)

当主服务器群中因故障导致数量减少时，动态地将备份服务器补充至主服务器群。

## 服务质量(QoS）

按不同的优先级对数据流进行分配。

## 服务类型(ToS)

按不同的服务类型（在 Type of Field 中标识）负载均衡对数据流进行分配。

## 规则模式

针对不同的数据流设置导向规则，用户可自行。

# 硬件负载均衡

采用硬件的方式实现负载均衡，一般是单独的负载均衡服务器，价格昂贵，一般土豪级公司可以考虑，业界领先的有两款，F5和A10（还有Citrix Netscaler）。

使用硬件负载均衡，主要考虑一下几个方面：

（1）功能考虑：功能全面支持各层级的负载均衡，支持全面的负载均衡算法，支持全局负载均衡；

（2）性能考虑：一般软件负载均衡支持到5万级并发已经很困难了，硬件负载均衡可以支持

（3）稳定性：商用硬件负载均衡，经过了良好的严格的测试，从经过大规模使用，在稳定性方面高；

（4）安全防护：硬件均衡设备除具备负载均衡功能外，还具备防火墙，防DDOS攻击等安全功能；

（5）维护角度：提供良好的维护管理界面，售后服务和技术支持；

（6）土豪公司：F5 Big Ip 价格：15w~55w不等；A10 价格：55w-100w不等；

缺点：

（1）价格昂贵；

（2）扩展能力差。

硬件负载均衡稳定性更强，双机或集群的效果更佳，可以应对高并发、高吞吐的网络环境中。在策略配置方面，**可以实现深度的健康检查方法，而不是简单的ping或tcp的方式，而是可以针对业务层进行健康检查，整体的策略调度更灵活、配置更方便，在七层负载方面更具优势**。

**推荐使用：**

核心系统必须使用硬件负载均衡设备；

测试系统和一般系统可以使用软件负载均衡设备。

# 负载均衡工具

硬件负载均衡性能优越，功能全面，但价格昂贵，一般适合初期或者土豪级公司长期使用。因此软件负载均衡在互联网领域大量使用。常用的软件负载均衡软件有Nginx、LVS、HaProxy等。

Nginx/LVS/HAProxy是目前使用最广泛的三种负载均衡软件。

## LVS

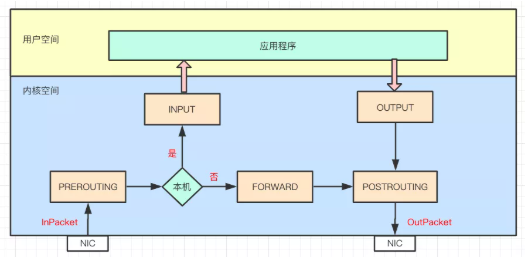
### 概述

LVS（Linux Virtual Server），也就是Linux虚拟服务器，是一个由章文嵩博士发起的自由软件项目。使用 LVS 技术要达到的目标是：通过LVS提供的负载均衡技术和Linux操作系统实现一个高性能、高可用的服务器群集，它具有良好可靠性、可扩展性和可操作性。从而以低廉的成本实现最优的服务性能。

**LVS 主要用来做四层（传输层）负载均衡**。

### netfilter

LVS是基于Linux内核中netfilter框架实现的负载均衡系统，netfilter 其实很复杂也很重要，平时我们说的Linux防火墙就是netfilter，不过我们平时操作的都是iptables，iptables 只是用户空间编写和传递规则的工具而已，真正工作的是netfilter。通过下图可以简单了解下netfilter的工作机制：



netfilter是内核态的Linux防火墙机制，作为一个通用、抽象的框架，提供了一整套的hook函数管理机制，提供诸如数据包过滤、网络地址转换、基于协议类型的连接跟踪的功能。

通俗点讲，就是netfilter提供一种机制，可以在数据包流经过程中，根据规则设置若干个关卡（hook函数）来执行相关的操作。netfilter总共设置了 5 个点，包括：PREROUTING、INPUT、FORWARD、OUTPUT、POSTROUTING

PREROUTING：刚刚进入网络层，还未进行路由查找的包，通过此处；

INPUT：通过路由查找，确定发往本机的包，通过此处；

FORWARD：经路由查找后，要转发的包，在POST\_ROUTING之前；

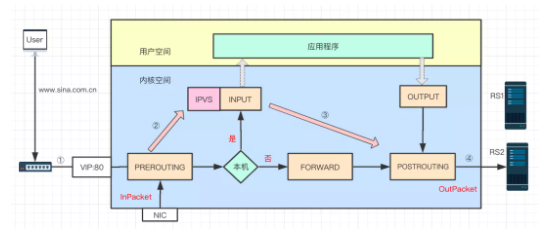
OUTPUT：从本机进程刚发出的包，通过此处；

POSTROUTING：进入网络层已经经过路由查找，确定转发，将要离开本设备的包，通过此处；

当一个数据包进入网卡，经过链路层之后进入网络层就会到达PREROUTING，接着根据目标IP地址进行路由查找，如果目标IP是本机，数据包继续传递到INPUT上，经过协议栈后根据端口将数据送到相应的应用程序；应用程序处理请求后将响应数据包发送到OUTPUT上，最终通过POSTROUTING后发送出网卡。如果目标IP不是本机，而且服务器开启了forward参数，就会将数据包递送给FORWARD上，最后通过POSTROUTING后发送出网卡。

### 原理

LVS是基于netfilter框架，主要工作于INPUT链上，在INPUT上注册ip\_vs\_in HOOK函数，进行IPVS主流程，大概原理如图所示：



当用户访问www.sina.com.cn时，用户数据通过层层网络，最后通过交换机进入LVS服务器网卡，并进入内核网络层。

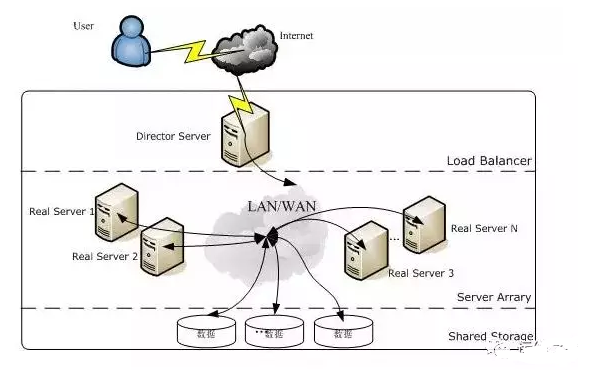
进入PREROUTING后经过路由查找，确定访问的目的VIP是本机IP地址，所以数据包进入到INPUT链上。

IPVS是工作在INPUT链上，会根据访问的vip+port判断请求是否IPVS服务，如果是则调用注册的IPVS HOOK函数，进行IPVS相关主流程，强行修改数据包的相关数据，并将数据包发往POSTROUTING链上。

POSTROUTING上收到数据包后，根据目标IP地址（后端服务器），通过路由选路，将数据包最终发往后端的服务器上。

### 架构

LVS架设的服务器集群系统由三个部分组成：最前端的负载均衡层（Loader Balancer），中间的服务器群组层，用Server Array表示，最底层的数据共享存储层，用Shared Storage表示。在用户看来所有的应用都是透明的，用户只是在使用一个虚拟服务器提供的高性能服务。



LVS的各个层次的详细介绍：

**Load Balancer层：**位于整个集群系统的最前端，有一台或者多台负载调度器（Director Server）组成，LVS模块就安装在Director Server上，而Director的主要作用类似于一个路由器，它含有完成LVS功能所设定的路由表，通过这些路由表把用户的请求分发给Server Array层的应用服务器（Real Server）上。同时，在Director Server上还要安装对Real Server服务的监控模块Ldirectord，此模块用于监测各个Real Server服务的健康状况。在Real Server不可用时把它从LVS路由表中剔除，恢复时重新加入。

**Server Array层：**由一组实际运行应用服务的机器组成，Real Server可以是Web服务器、Mail服务器、FTP服务器、DNS服务器、视频服务器中的一个或者多个，每个Real Server之间通过高速的LAN或分布在各地的WAN相连接。在实际的应用中，Director Server也可以同时兼任Real Server的角色。

**Shared Storage层：**是为所有Real Server提供共享存储空间和内容一致性的存储区域，在物理上一般由磁盘阵列设备组成，为了提供内容的一致性，一般可以通过NFS网络文件系统共享数据，但NFS在繁忙的业务系统中，性能并不是很好，此时可以采用集群文件系统，例如Redhat的GFS文件系统、Oracle提供的OCFS2文件系统等。

从整个LVS结构可以看出，Director Server是整个LVS的核心，目前用于Director Server 的操作系统只能是Linux和FreeBSD，Linux 2.6内核不用任何设置就可以支持LVS功能，而 FreeBSD作为Director Server的应用还不是很多，性能也不是很好。对于Real Server，几乎可以是所有的系统平台，Linux、windows、Solaris、AIX、BSD系列都能很好地支持。

### 工作模式

开源 LVS 版本有 3 种工作模式，每种模式工作原理截然不同，说各种模式都有自己的优缺点，分别适合不同的应用场景，不过最终本质的功能都是能实现均衡的流量调度和良好的扩展性。主要包括以下三种模式：

DR 模式

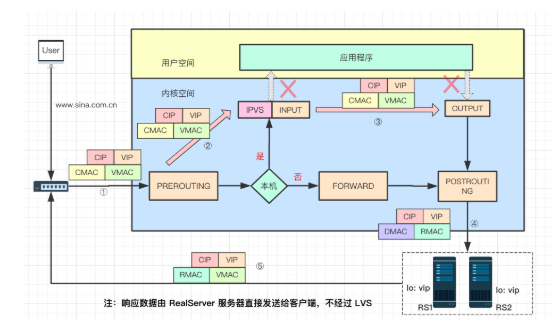
NAT 模式

Tunnel 模式

另外必须要说的模式是 FullNAT，这个模式在开源版本中是模式没有的，代码没有合并进入内核主线版本。

#### DR模式

下边会针对 DR 模式的具体实现原理，详细的阐述 DR 模式是如何工作的。



其实 DR 是最常用的工作模式，因为它的强大的性能。下边以一次请求和响应数据流的过程来描述 DR 模式的具体原理。

（一）实现原理过程

① 当客户端请求 www.sina.com.cn 主页，经过 DNS 解析到 IP 后，向新浪服务器发送请求数据，数据包经过层层网络到达新浪负载均衡 LVS 服务器，到达 LVS 网卡时的数据包：源 IP 是客户端 IP 地址 CIP，目的 IP 是新浪对外的服务器 IP 地址，也就是 VIP；此时源 MAC 地址是 CMAC，其实是 LVS 连接的路由器的 MAC 地址（为了容易理解记为 CMAC），目标 MAC 地址是 VIP 对应的 MAC，记为 VMAC。

② 数据包到达网卡后，经过链路层到达 PREROUTING 位置（刚进入网络层），查找路由发现目的 IP 是 LVS 的 VIP，就会递送到 INPUT 链上，此时数据包 MAC、IP、Port 都没有修改。

③ 数据包到达 INPUT 链，INPUT 是 LVS 主要工作的位置。此时 LVS 会根据目的 IP 和 Port 来确认是否是 LVS 定义的服务，如果是定义过的 VIP 服务，就会根据配置的 Service 信息，从 RealServer 中选择一个作为后端服务器 RS1，然后以 RS1 作为目标查找 Out 方向的路由，确定一下跳信息以及数据包要通过哪个网卡发出。最后将数据包通过 INET\_HOOK 到 OUTPUT 链上（Out 方向刚从四层进入网络层）。

④ 数据包通过 POSTROUTING 链后，从网络层转到链路层，将目的 MAC 地址修改为 RealServer 服务器 MAC 地址，记为 RMAC；而源 MAC 地址修改为 LVS 与 RS 同网段的 selfIP 对应的 MAC 地址，记为 DMAC。此时，数据包通过交换机转发给了 RealServer 服务器（注：为了简单图中没有画交换机）。

⑤ 请求数据包到达 RealServer 服务器后，链路层检查目的 MAC 是自己网卡地址。到了网络层，查找路由，目的 IP 是 VIP（lo 上配置了 VIP），判定是本地主机的数据包，经过协议栈后拷贝至应用程序（比如这里是 nginx 服务器），nginx 响应请求后，产生响应数据包。以目的 VIP 为 dst 查找 Out 路由，确定吓一跳信息和发送网卡设备信息，发送数据包。此时数据包源、目的 IP 分别是 VIP、CIP，而源 MAC 地址是 RS1 的 RMAC，目的 MAC 是下一跳（路由器）的 MAC 地址，记为 CMAC（为了容易理解，记为 CMAC）。然后数据包通过 RS 相连的路由器转发给真正客户端。

从整个过程可以看出，DR 模式 LVS 逻辑非常简单，数据包通过路由方式直接转发给 RS，而且响应数据包是由 RS 服务器直接发送给客户端，不经过 LVS。我们知道一般请求数据包会比较小，响应报文较大，经过 LVS 的数据包基本上都是小包，上述几条因素是 LVS 的 DR 模式性能强大的主要原因。

（二）优缺点和使用场景

DR 模式的优点

a. 响应数据不经过 lvs，性能高

b. 对数据包修改小，信息保存完整（携带客户端源 IP）

DR 模式的缺点

a. lvs 与 rs 必须在同一个物理网络（不支持跨机房）

b. rs 上必须配置 lo 和其它内核参数

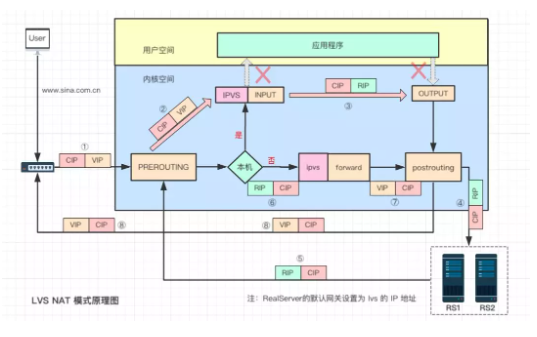
c. 不支持端口映射

DR 模式的使用场景

如果对性能要求非常高，可以首选 DR 模式，而且可以透传客户端源 IP 地址。

#### NAT模式

lvs 的第 2 种工作模式是 NAT 模式，下图详细介绍了数据包从客户端进入 lvs 后转发到 rs，后经 rs 再次将响应数据转发给 lvs，由 lvs 将数据包回复给客户端的整个过程。



（一）实现原理与过程

① 用户请求数据包经过层层网络，到达 lvs 网卡，此时数据包源 IP 是 CIP，目的 IP 是 VIP。

② 经过网卡进入网络层 prerouting 位置，根据目的 IP 查找路由，确认是本机 IP，将数据包转发到 INPUT 上，此时源、目的 IP 都未发生变化。

③ 到达 lvs 后，通过目的 IP 和目的 port 查找是否为 IPVS 服务。若是 IPVS 服务，则会选择一个 RS 作为后端服务器，将数据包目的 IP 修改为 RIP，并以 RIP 为目的 IP 查找路由信息，确定下一跳和出口信息，将数据包转发至 output 上。

④ 修改后的数据包经过 postrouting 和链路层处理后，到达 RS 服务器，此时的数据包源 IP 是 CIP，目的 IP 是 RIP。

⑤ 到达 RS 服务器的数据包经过链路层和网络层检查后，被送往用户空间 nginx 程序。nginx 程序处理完毕，发送响应数据包，由于 RS 上默认网关配置为 lvs 设备 IP，所以 nginx 服务器会将数据包转发至下一跳，也就是 lvs 服务器。此时数据包源 IP 是 RIP，目的 IP 是 CIP。

⑥ lvs 服务器收到 RS 响应数据包后，根据路由查找，发现目的 IP 不是本机 IP，且 lvs 服务器开启了转发模式，所以将数据包转发给 forward 链，此时数据包未作修改。

⑦ lvs 收到响应数据包后，根据目的 IP 和目的 port 查找服务和连接表，将源 IP 改为 VIP，通过路由查找，确定下一跳和出口信息，将数据包发送至网关，经过复杂的网络到达用户客户端，最终完成了一次请求和响应的交互。

NAT 模式双向流量都经过 LVS，因此 NAT 模式性能会存在一定的瓶颈。不过与其它模式区别的是，NAT 支持端口映射，且支持 windows 操作系统。

（二）优点、缺点与使用场景

NAT 模式优点

a. 能够支持 windows 操作系统

b. 支持端口映射。如果 rs 端口与 vport 不一致，lvs 除了修改目的 IP，也会修改 dport 以支持端口映射。

NAT 模式缺点

a. 后端 RS 需要配置网关

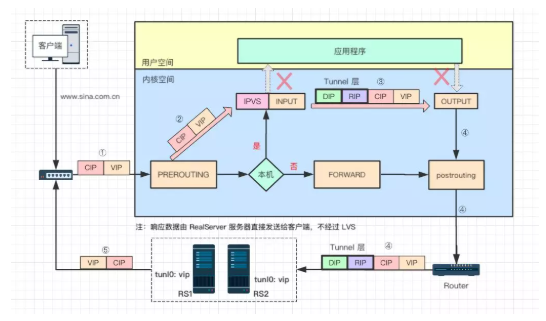
b. 双向流量对 lvs 负载压力比较大

NAT 模式的使用场景

如果你是 windows 系统，使用 lvs 的话，则必须选择 NAT 模式了。

#### Tunnel模式

Tunnel 模式在国内使用的比较少，不过据说腾讯使用了大量的 Tunnel 模式。它也是一种单臂的模式，只有请求数据会经过 lvs，响应数据直接从后端服务器发送给客户端，性能也很强大，同时支持跨机房。下边继续看图分析原理。



（一）实现原理与过程

① 用户请求数据包经过多层网络，到达 lvs 网卡，此时数据包源 IP 是 cip，目的 ip 是 vip。

② 经过网卡进入网络层 prerouting 位置，根据目的 ip 查找路由，确认是本机 ip，将数据包转发到 input 链上，到达 lvs，此时源、目的 ip 都未发生变化。

③ 到达 lvs 后，通过目的 ip 和目的 port 查找是否为 IPVS 服务。若是 IPVS 服务，则会选择一个 rs 作为后端服务器，以 rip 为目的 ip 查找路由信息，确定下一跳、dev 等信息，然后 IP 头部前边额外增加了一个 IP 头（以 dip 为源，rip 为目的 ip），将数据包转发至 output 上。

④ 数据包根据路由信息经最终经过 lvs 网卡，发送至路由器网关，通过网络到达后端服务器。

⑤ 后端服务器收到数据包后，ipip 模块将 Tunnel 头部卸载，正常看到的源 ip 是 cip，目的 ip 是 vip，由于在 tunl0 上配置 vip，路由查找后判定为本机 ip，送往应用程序。应用程序 nginx 正常响应数据后以 vip 为源 ip，cip 为目的 ip 数据包发送出网卡，最终到达客户端。

Tunnel 模式具备 DR 模式的高性能，又支持跨机房访问，听起来比较完美了。不过国内运营商有一定特色性，比如 RS 的响应数据包的源 IP 为 VIP，VIP 与后端服务器有可能存在跨运营商的情况，有可能被运营商的策略封掉。Tunnel 在生产环境确实没有使用过，在国内推行 Tunnel 可能会有一定的难度吧！

（二）优点、缺点与使用场景

Tunnel 模式的优点

a. 单臂模式，对 lvs 负载压力小

b. 对数据包修改较小，信息保存完整

c. 可跨机房（不过在国内实现有难度）

Tunnel 模式的缺点

a. 需要在后端服务器安装配置 ipip 模块

b. 需要在后端服务器 tunl0 配置 vip

c. 隧道头部的加入可能导致分片，影响服务器性能

d. 隧道头部 IP 地址固定，后端服务器网卡 hash 可能不均

e. 不支持端口映射

Tunnel 模式的使用场景

理论上，如果对转发性能要求较高，且有跨机房需求，Tunnel 可能是较好的选择。

## Nginx

### 概述

Nginx（发音同 engine x）是一个网页服务器，它能反向代理 HTTP、HTTPS,、SMTP、POP3、IMAP的协议链接，以及一个负载均衡器和一个HTTP缓存。

**Nginx主要用来做七层（应用层）负载均衡**。

并发性能：官方支持每秒 5 万并发，实际国内一般到每秒 2 万并发，有优化到每秒 10 万并发的。具体性能看应用场景。

### 特点

模块化设计：良好的扩展性，可以通过模块方式进行功能扩展。

高可靠性：主控进程和 worker 是同步实现的，一个 worker 出现问题，会立刻启动另一个 worker。

内存消耗低：一万个长连接（keep-alive）,仅消耗 2.5 MB 内存。

支持热部署：不用停止服务器，实现更新配置文件，更换日志文件、更新服务器程序版本。

并发能力强：官方数据每秒支持 5 万并发；

功能丰富：优秀的反向代理功能和灵活的负载均衡策略

### 工作模式

一个 master 进程，生成一个或者多个 worker 进程。但这里 master 是使用 root 身份启动的，因为 Nginx 要工作在 80 端口。而只有管理员才有权限启动小于低于 1023 的端口。master 主要是负责的作用只是启动 worker，加载配置文件，负责系统的平滑升级。其它的工作是交给 worker。那当 worker 被启动之后，也只是负责一些 web 最简单的工作，而其它的工作都是由 worker 中调用的模块来实现的。

模块之间是以流水线的方式实现功能的。流水线，指的是一个用户请求，由多个模块组合各自的功能依次实现完成的。比如：第一个模块只负责分析请求首部，第二个模块只负责查找数据，第三个模块只负责压缩数据，依次完成各自工作。来实现整个工作的完成。

它们是如何实现热部署的呢？是这样的，我们前面说 master 不负责具体的工作，而是调用 worker 工作，它只是负责读取配置文件，因此当一个模块修改或者配置文件发生变化，是由 master 进行读取，因此此时不会影响到 worker 工作。在 master 进行读取配置文件之后，不会立即把修改的配置文件告知 worker 。而是让被修改的 worker 继续使用老的配置文件工作，当 worker 工作完毕之后，直接当掉这个子进程，更换新的子进程，使用新的规则。

## HAProxy

HAProxy 也是使用较多的一款负载均衡软件。HAProxy 提供高可用性、负载均衡以及基于 TCP 和 HTTP 应用的代理，支持虚拟主机，是免费、快速并且可靠的一种解决方案。特别适用于那些负载特大的 Web站点。运行模式使得它可以很简单安全的整合到当前的架构中，同时可以保护你的web服务器不被暴露到网络上。

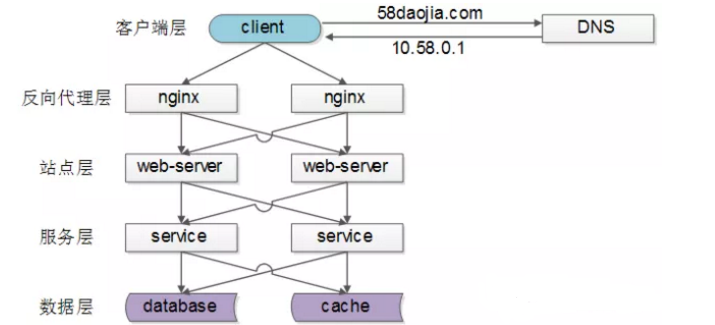
HAProxy 是一个使用 C 语言编写的自由及开放源代码软件，其提供高可用性、负载均衡，以及基于 TCP 和 HTTP 的应用程序代理。

**Haproxy 主要用来做七层负载均衡**。

## Twemproxy

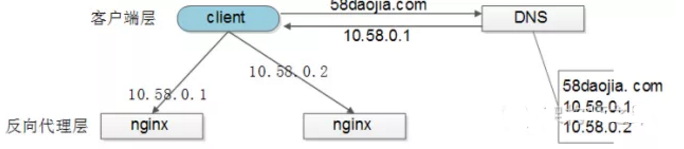
Twemproxy用来做redis的结点的分片、redis的存储受限与单个结点的内存容量，数据量大到需要分片，使用twemproxy可做到对业务层透明的分片；twemproxy也是使用的单线程reactor模型，一个twemproxy后端接再多的redis结点，其能够支撑的TPS不会超过单个redis结点的处理能力，使用时需要启动多个twemproxy对外提供查询结点。

# 互联网分层



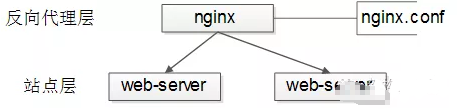
常见互联网分布式架构如上，分为客户端层、反向代理nginx层、站点层、服务层、数据层。可以看到，每一个下游都有多个上游调用，只需要做到，每一个上游都均匀访问每一个下游，就能实现“将请求/数据均匀分摊到多个操作单元上执行”。

## 客户端层->反向代理层



客户端层到反向代理层的负载均衡，是通过**“DNS轮询”**实现的：DNS-server对于一个域名配置了多个解析ip，每次DNS解析请求来访问DNS-server，会轮询返回这些ip，保证每个ip的解析概率是相同的。这些ip就是nginx的外网ip，以做到每台nginx的请求分配也是均衡的。

## 反向代理层->站点层



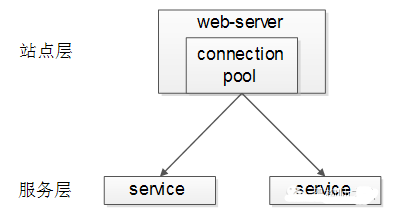
反向代理层到站点层的负载均衡，是通过“nginx”实现的。通过修改nginx.conf，可以实现多种负载均衡策略：

1）请求轮询：和DNS轮询类似，请求依次路由到各个web-server

2）最少连接路由：哪个web-server的连接少，路由到哪个web-server

3）ip哈希：按照访问用户的ip哈希值来路由web-server，只要用户的ip分布是均匀的，请求理论上也是均匀的，ip哈希均衡方法可以做到，同一个用户的请求固定落到同一台web-server上，此策略适合有状态服务，例如session（58沈剑备注：可以这么做，但强烈不建议这么做，站点层无状态是分布式架构设计的基本原则之一，session最好放到数据层存储）。

## 站点层->服务层



站点层到服务层的负载均衡，是通过“**服务连接池**”实现的。

上游连接池会建立与下游服务多个连接，每次请求会“随机”选取连接来访问下游服务。

## 数据层

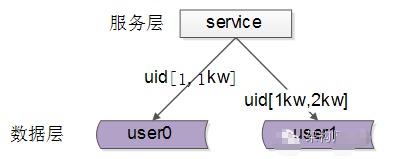
在数据量很大的情况下，由于数据层（db，cache）涉及数据的水平切分，所以数据层的负载均衡更为复杂一些，它分为“数据的均衡”与“请求的均衡”。

数据的均衡是指：水平切分后的每个服务（db，cache），数据量是差不多的。

请求的均衡是指：水平切分后的每个服务（db，cache），请求量是差不多的。

业内常见的水平切分方式有这么几种：

### 按照range水平切分



每一个数据服务，存储一定范围的数据，上图为例：

user0服务，存储uid范围1-1kw

user1服务，存储uid范围1kw-2kw

这个方案的好处：

（1）规则简单，service只需判断一下uid范围就能路由到对应的存储服务

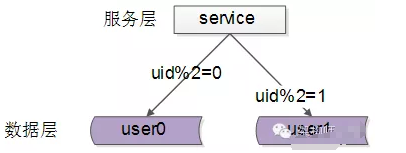
（2）数据均衡性较好

（3）比较容易扩展，可以随时加一个uid[2kw,3kw]的数据服务

不足：

（1）请求的负载不一定均衡，一般来说，新注册的用户会比老用户更活跃，大range的服务请求压力会更大

### 按照id哈希水平切分



每一个数据服务，存储某个key值hash后的部分数据，上图为例：

user0服务，存储偶数uid数据

user1服务，存储奇数uid数据

这个方案的好处：

（1）规则简单，service只需对uid进行hash能路由到对应的存储服务

（2）数据均衡性较好

（3）请求均匀性较好

不足：

（1）不容易扩展，扩展一个数据服务，hash方法改变时候，可能需要进行数据迁移

## 总结

负载均衡（Load Balance）是分布式系统架构设计中必须考虑的因素之一，它通常是指，将请求/数据【均匀】分摊到多个操作单元上执行，负载均衡的关键在于【均匀】。

（1）【客户端层】到【反向代理层】的负载均衡，是通过“DNS轮询”实现的

（2）【反向代理层】到【站点层】的负载均衡，是通过“nginx”实现的

（3）【站点层】到【服务层】的负载均衡，是通过“服务连接池”实现的

（4）【数据层】的负载均衡，要考虑“数据的均衡”与“请求的均衡”两个点，常见的方式有“按照范围水平切分”与“hash水平切分”

# 数据库负载均衡

# 小结

（1）一般硬件的负载均衡也要做双机高可用，因此成本会比较高。

（2）互联网公司一般使用开源软件，因此大部分应用采用软件负载均衡；部分采用硬件负载均衡。

比如某互联网公司，目前是使用几台F5做全局负载均衡，内部使用Nginx等软件负载均衡。