



# 系统架构设计

## - 负载均衡和高可用

张浦



# 目录

- **负载均衡技术介绍**
- 高可用的系统设计
- 高可用方案及实践



# 负载均衡技术介绍

- **什么是负载均衡？**
- 负载均衡算法原理
- 负载均衡应用模式

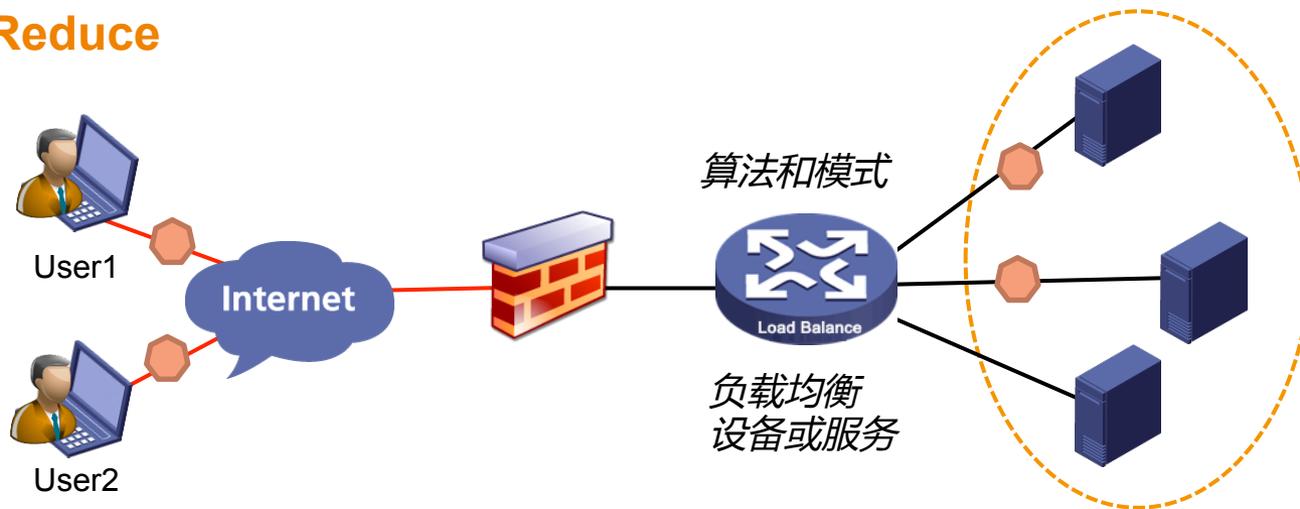
# 什么是负载均衡？

负载均衡 (load balance) 顾名思义，是把服务的并发请求均衡得负载到后端多个具有相同能力的服务进行处理分担。以廉价有效透明的方式扩展网络设备或服务的带宽，增加吞吐量，增强服务的整体处理能力，提供服务的灵活性和可用性。

两个方面：调度算法和应用模式

两个含义：

- 大量的并发访问或数据流量分担到多台节点设备上分别处理，减少用户等待响应的时间。典型：**WEB集群**
- 单个重负载的运算分担到多台节点设备上做并行处理，每个节点设备处理结束后，将结果汇总，返回给用户，系统处理能力得到大幅度提高。典型：**MapReduce**





# 负载均衡技术介绍

- 什么是负载均衡？
- **负载均衡算法原理**
- 负载均衡应用模式



# 负载均衡调度算法原理

负载均衡算法是负载均衡设备（包括虚拟设备或相关软件）在执行负载均衡调度，选择具体处理的后端服务的时候使用的**调度和分发的逻辑**。

流行和常用的部分负载均衡算法包括：

- 轮寻算法：Round Robin/Weight Round Robin Scheduling
- Hash算法: 随机数Hash，Sources Hashing Scheduling
- 一致性Hash算法：Consistency Hash Scheduling
- Session(一般用于WEB)，严格的说不算是算法
- 最少连接或请求数: (Weight)Least Connection/Request Scheduling
- 最大空闲：Most idle First(基于监控CPU,内存,带宽等综合评估)
- 平均最快响应：平均最快响应
- 最少流量：Least Traffic Scheduling

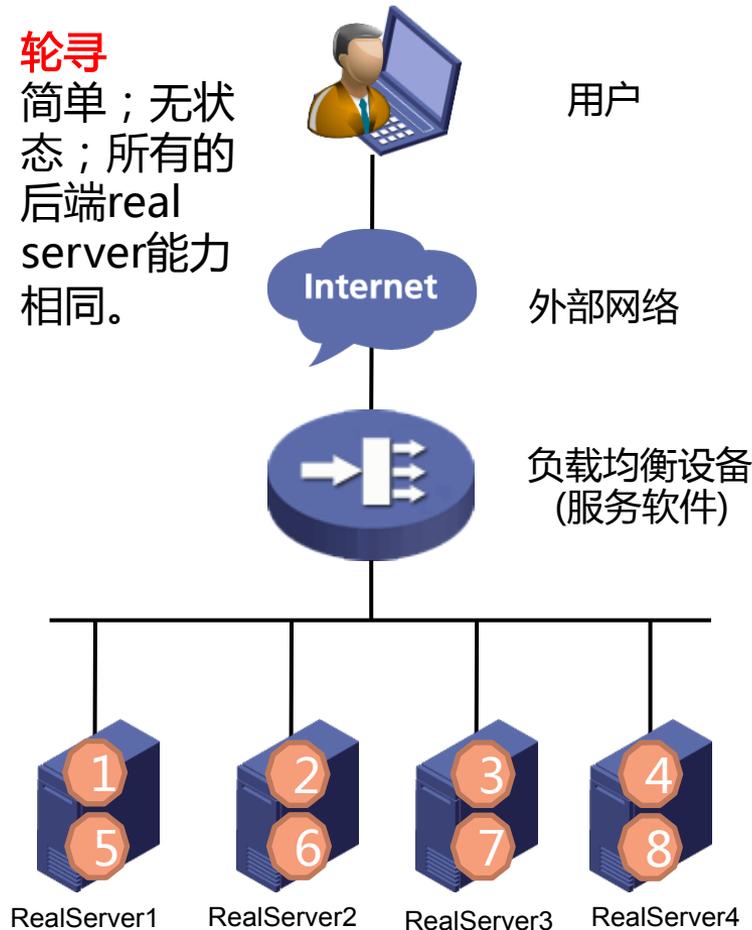
**注意：**负载均衡的算法只是规定了调度和分发的逻辑，在不同的负载均衡方案中都可能使用相同和类似的算法，它只是负载均衡方案的一部分。

# 轮寻算法

以依次轮叫的方式依次将请求调度不同的后端服务器(Real Server)。算法的优点是其简洁性，无状态。算法简单表示为： $i = (i + 1) \bmod n$

## 轮寻

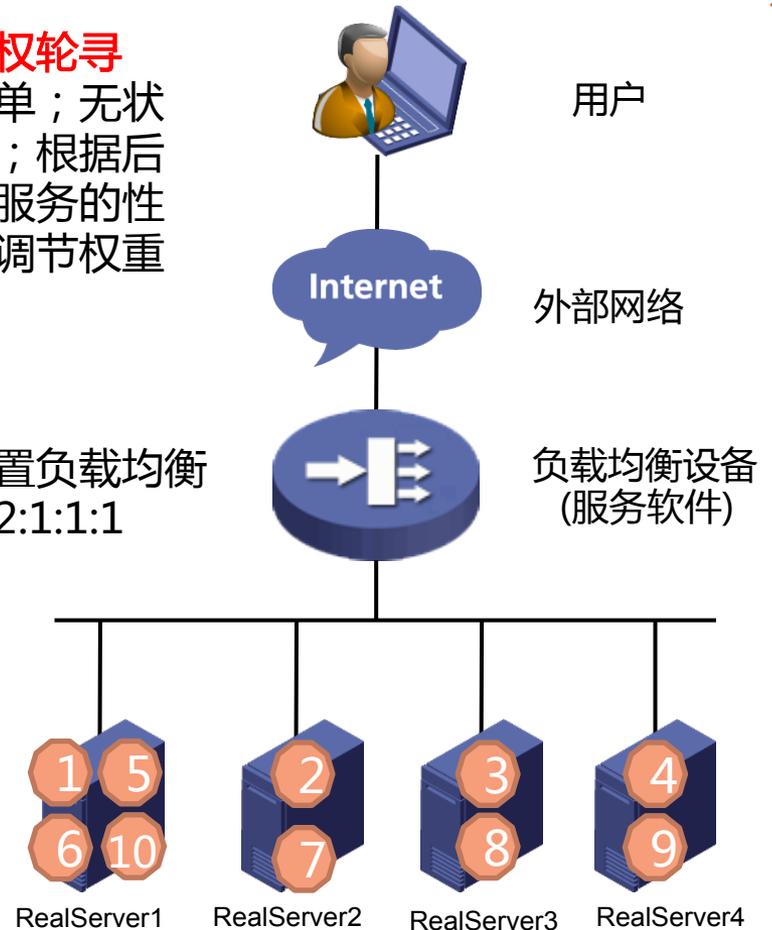
简单；无状态；所有的后端real server能力相同。



## 加权轮寻

简单；无状态；根据后端服务的性能调节权重

设置负载均衡按2:1:1:1



# Hash算法

Hash算法，又叫取余算法。一般是对请求报文中的某项数据(key，一般常用客户端来源IP)计算Hash值，然后按机器数量(n)取模。 $idx = Hash(key) \% n$

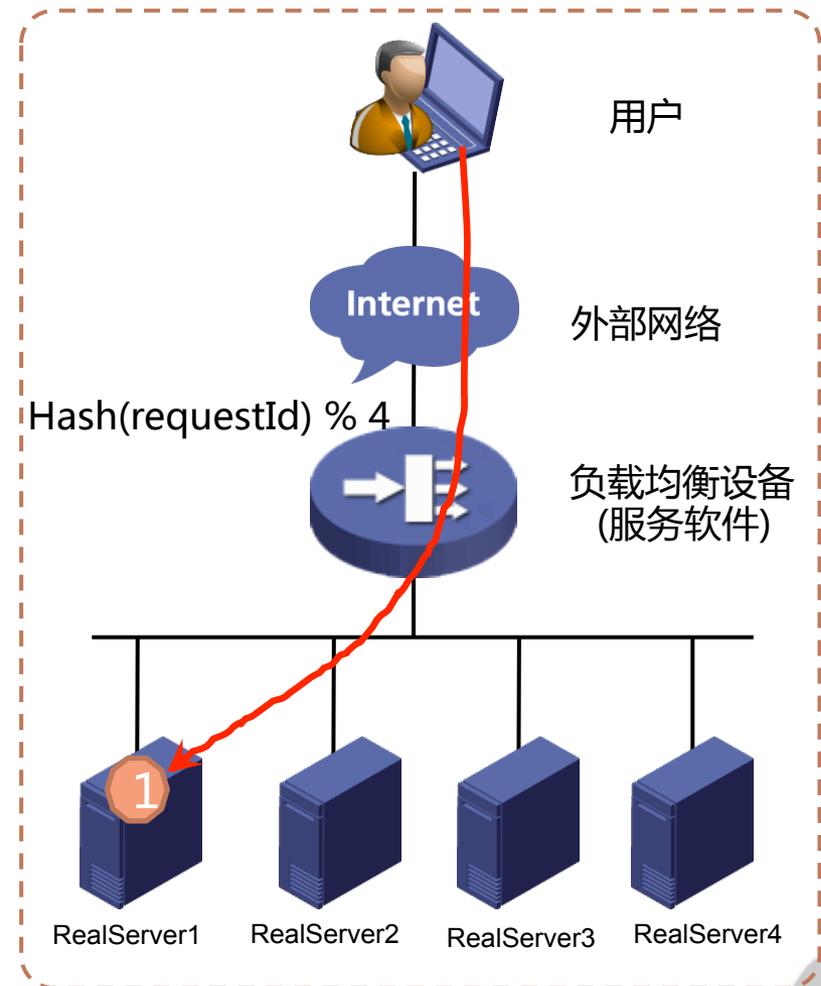
KEY选择常用实践用法：

## 1、请求时间或随机数

简单，具有一定分散性，但不稳定，一般用于要求不高的负载均衡场景。

## 2、来源IP

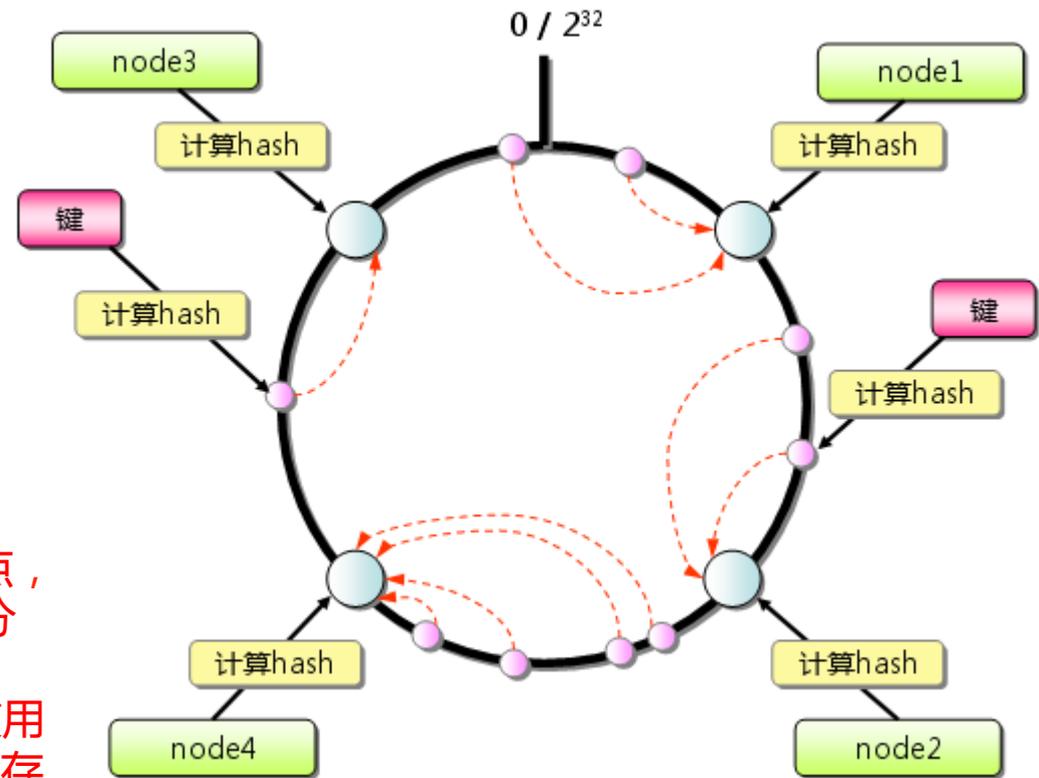
简单。如果客户的分布比较广，这种方式分散性较好。但如果较多的客户请求来源于同一IP（公司网络通过路由器上网），分散效果较差。大多负载均衡设备都支持这种算法，nginx和LVS等软件也支持。



# 一致性Hash算法

一致性Hash算法最常用于分布式缓存定位，但同时也可以再系统或程序中用于负载均衡，该算法本来的意义就在于分散负载和快速定位。

算法逻辑：首先求出服务器（节点）的哈希值，并将其配置到  $0 \sim 2^{32}$ （正整数范围）的圆上。然后用同样的方法求出存储数据的键的哈希值，并映射到圆上。然后从数据映射到的位置开始顺时针查找将数据保存到找到的第一个服务器上。如果超过  $2^{32}$  仍然找不到服务器，就会保存到第一台服务器上。



实践：

- 1 个实体节点可以虚拟  $N$  个虚拟节点，如 (160, 200) 让节点更为均匀的分别在环上。
- 并非负载均衡设备和系统软件可以使用这些算法，实践开发中，如分表，缓存定位都有较多的应用场景。

# Session

会话方式，也叫粘性模式，严格的说这不是负载均衡算法，只能是负载均衡算法的扩展补充，转用于基于会话的请求。一般结合轮训等其他算法一起使用。

## 问题：

基于会话的请求（如：带用户认证的WEB系统）不能简单的使用轮训或其他算法把同一用户会话的请求转发到不同的后端服务，造成会话丢失和混乱。

## 解决方案：

首次请求记录用户的SessionID，然后再通过轮训等算法选择后端服务器，如果用户后续同一SessionID的请求，则无需再选择服务器，直接转发给前面SessionID对应的后端服务器。

## 特点：

- 解决WEB集群会话问题最简单实用的方案。避免多播复制Session或采用分布式缓存。
- 负载均衡分散性会有一些的不确定性，这与用户的操作有关
- 更好的利用单个服务器本身的缓存。

# 最少连接或请求数算法

Least Connection Scheduling，调度设备或服务记录后端服务器接受请求的计数，每次请求总是发给计数最小的服务器处理。

**问题：**  
客户端每次请求连接持续时间不均，部分后端服务器压力大。轮训或HASH等静态调度算法无法解决该问题。

**使用场景：**

适合长时处理的请求服务，  
如FTP，云存储服务等

最小连接调度是一种动态调度算法，它通过服务器当前所活跃的连接数来估计服务器的负载情况

## 负载均衡算法—最少连接数

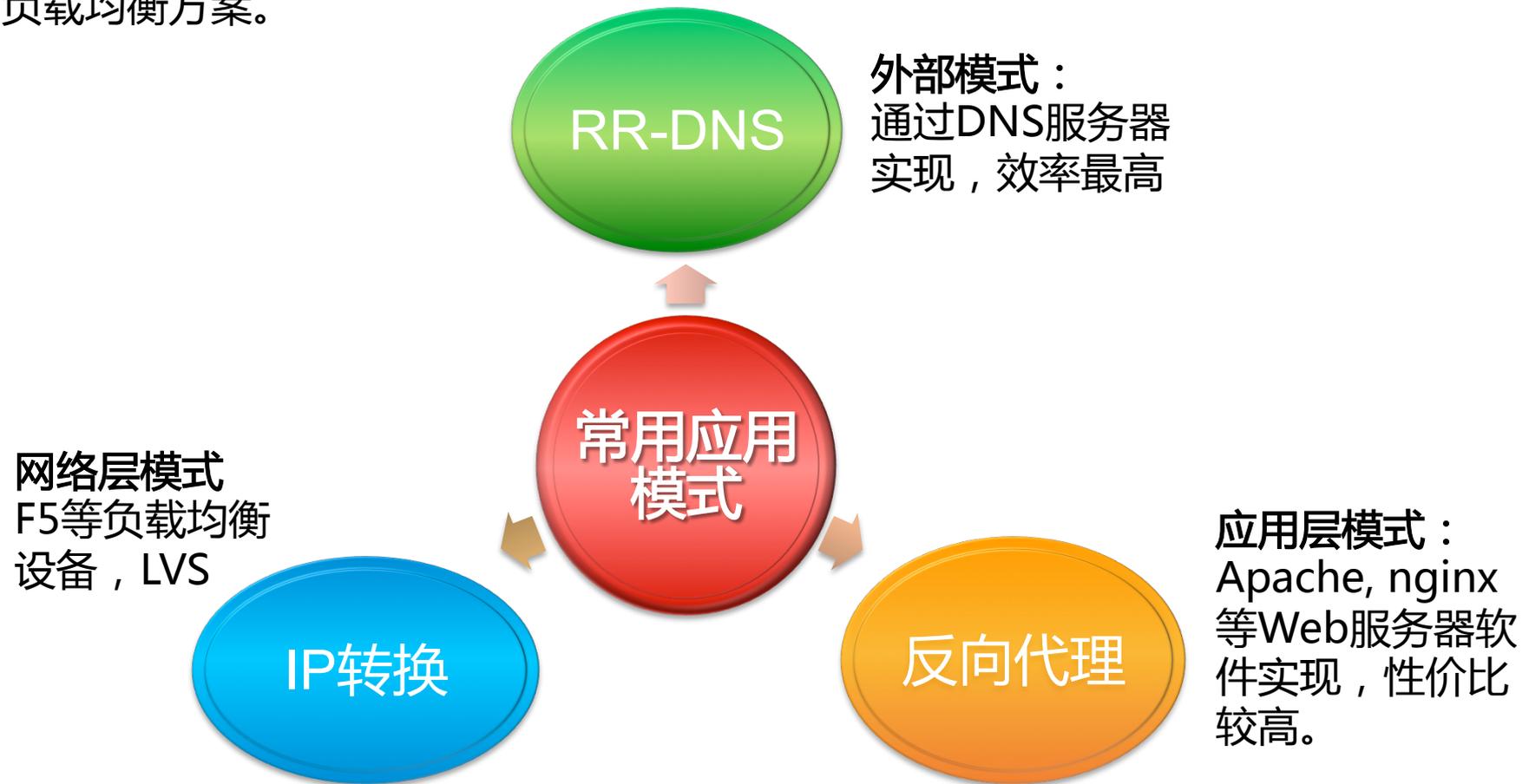


# 负载均衡技术介绍

- 什么是负载均衡？
- 负载均衡算法原理
- **负载均衡应用模式**

# 负载均衡应用模式

负载均衡模式主要是在整体方案中选择从服务网络的那个层次或那个产品来实现负载均衡方案。



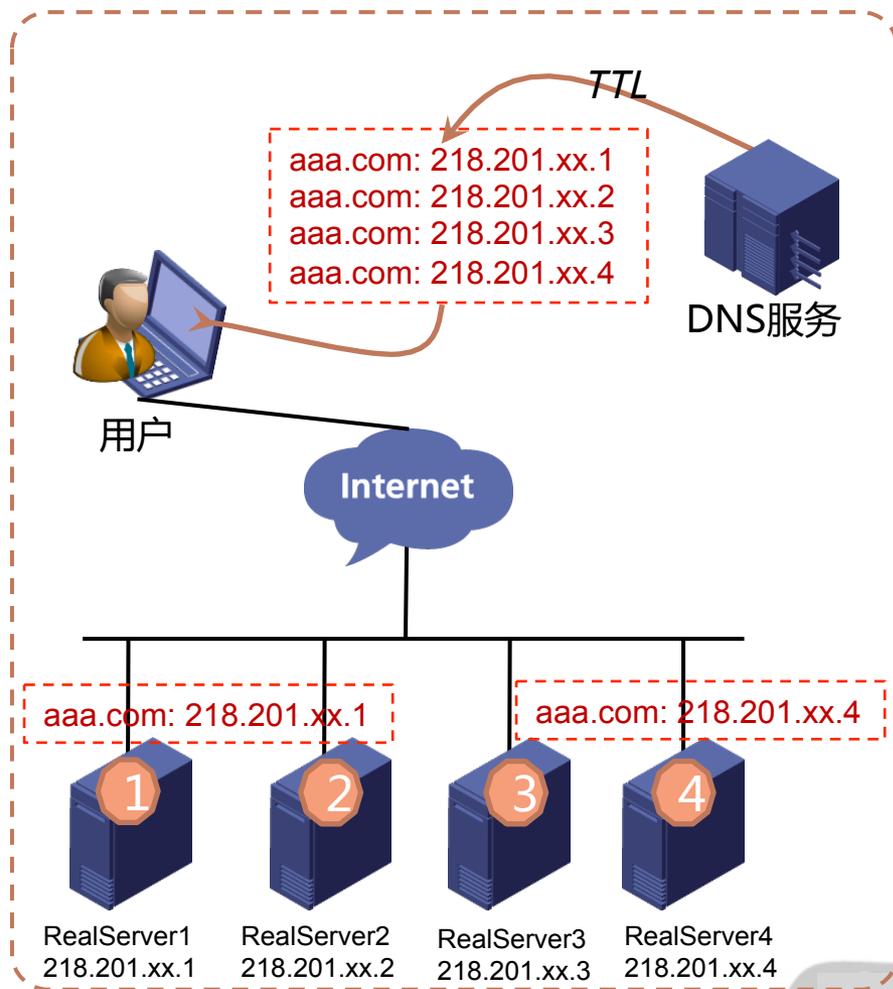


# RR-DNS

DNS轮训模式的原理是利用DNS服务器支持同一域名配置多个独立IP指向，然后轮训解析指向IP实现多次访问的调度和分发，实现负载均衡。。

## 特点：

- 负载均衡实现与后端服务完全没有关系，有DNS在本地解析指向实现轮训调度。这个方面来看性能是最佳的。
- DNS服务无法检测到后端服务器是否正常，在TTL失效前，会一直指向失效的服务器，这就要求在实践生成中，必须解决后端服务器的高可用问题。
- 一般的第三方DNS服务提供商都支持该功能，但如果更新频率高或附带更新逻辑，一般会在系统内自建DNS服务，然后在注册为公共DNS服务。





# 反向代理-什么是反向代理?

对反向代理的解释可以通过我们了解的正向代理来对比说明。

- 正向代理：用户通过代理服务访问internet, 把internet返回的数据转发给用户。

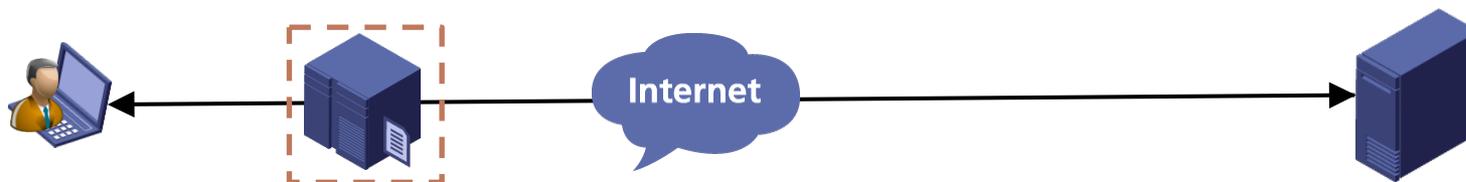
Internal Client ——(request-url)——> Forward Proxy ——> Internet

- 反向代理：与正向代理相对来说，是接受internet上用户的请求，转发给内部的多台服务器处理，完成后转发后端服务器的返回给对应的用户。

/————> Internal Server1

Internet —————> Reverse Proxy Server —————> Internal Server2

\————> internal serverN



正向代理：  
放到近客户测，对于整个网络请求，它的角色实际是客户端，代理客户对外的访问  
(request)

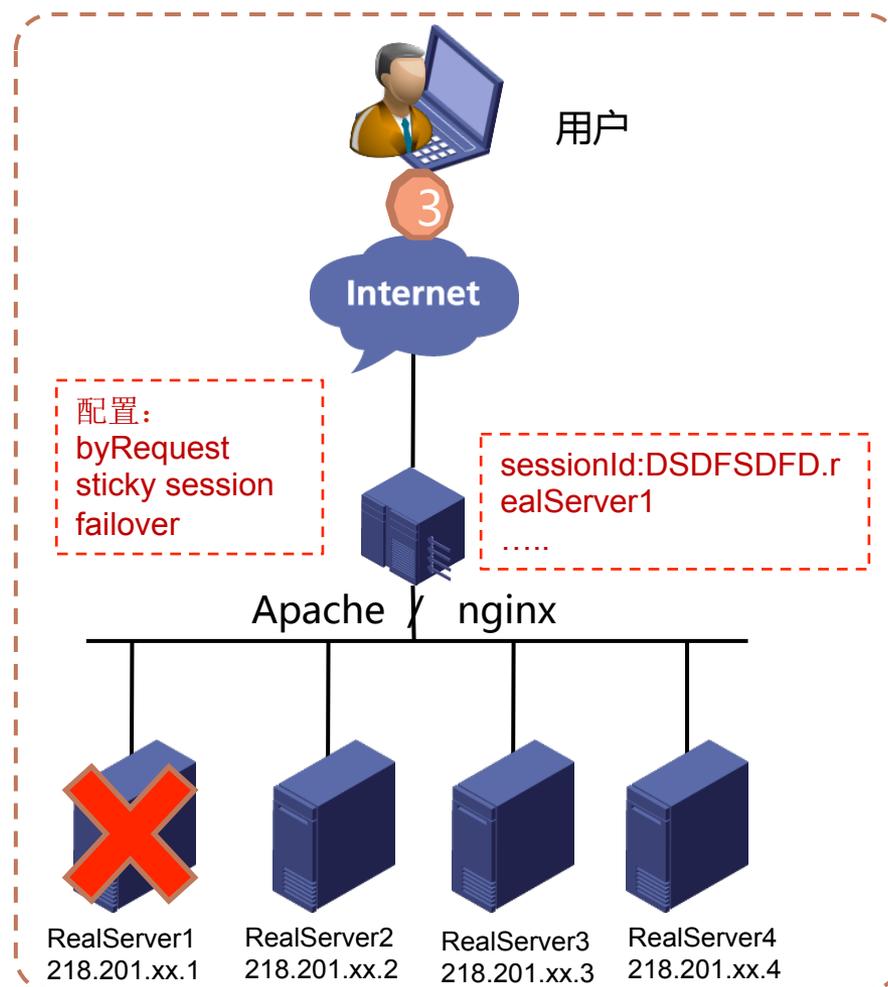
反向代理：  
放到近服务测，对于整个网络请求，它的角色实际是服务器，代理接受 ( accept ) 所有用户的请求。

# 反向代理-应用模式

利用反向代理软件实现负载均衡是性价比较高的模式。

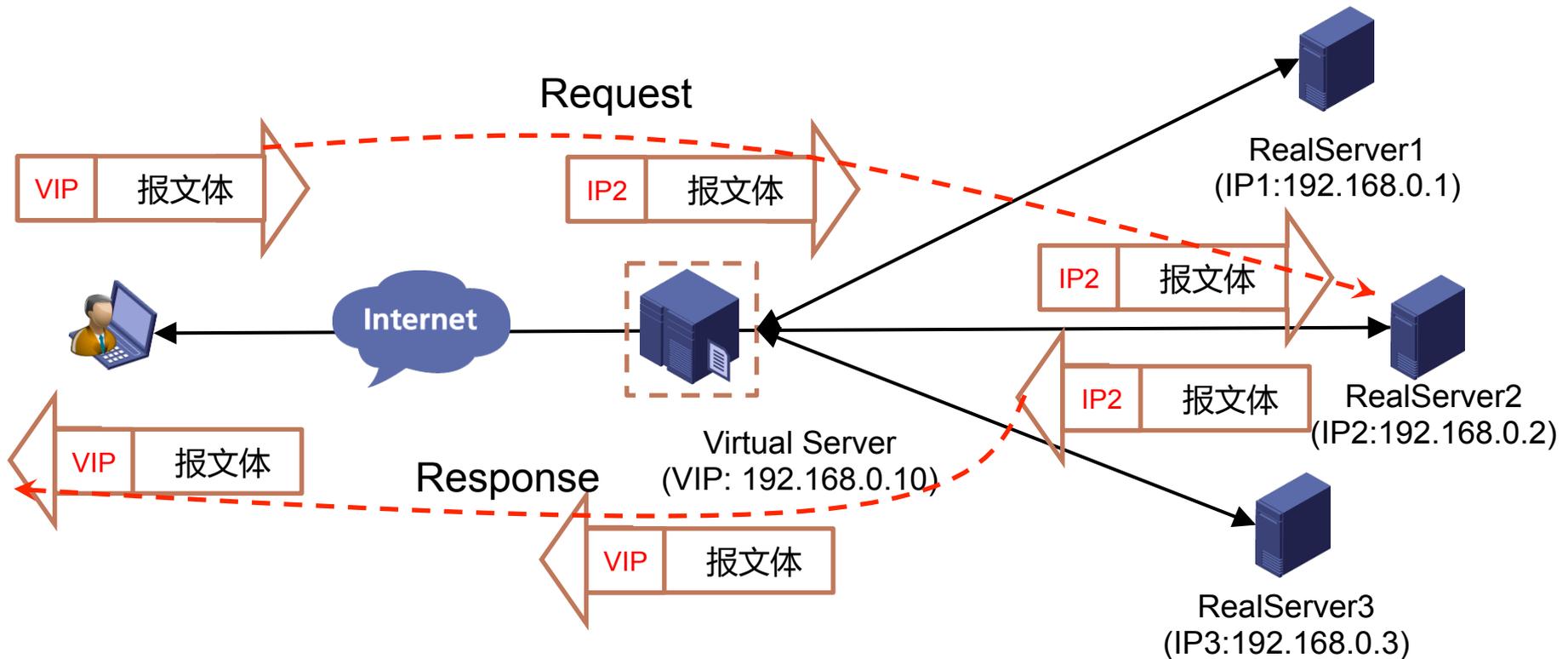
- 优点：网络，设备和程序都是透明，建设简单，能力较强，成本低廉，效率较好。
  - 缺点：完全软实现，性能依赖代理软件节点。当并发量达到一定程度，代理服务节点是系统的瓶颈。
1. 支持负载均衡调度算法：轮训，最小流量，最小连接
  2. 支持会话粘性转发模式
  3. 支持故障转移

使用场景：中小型WEB应用的负载均衡和高可用。



# IP转换-原理

IP转换模式的负载均衡一般是在网络的IP层实现，通过报文改写的方式实现VIP到多个内部IP的转发调度，以达到负载均衡的效果。



特点：网络层方案，效率较高，稳定性较好；可与操作系统内核结合；工业级模式和方案；大部分商业设备和产品都以该方式为主；LVS的基本原理也类同。



# IP转换-LVS

LVS ( Linux Virtual Server ) ,中国人 ( 98年 ) 写的工业级的负载平衡调度解决方案。也是目前业界最流行的软件方式实现负载均衡的模式之一。LVS也是利用IP转发的原理实现大多数有商业产品实现的能力，并做了部分优化，主要有三种模式的应用。

1. 通过NAT实现虚拟服务器 ( VS/NAT )
2. 通过IP隧道实现虚拟服务器 ( VS/TUN )
3. 通过直接路由实现虚拟服务器 ( VS/DR )

SourceForge (sourceforge.net)

Linux的门户网站

RealPlayer提供音频视频服务

世界上最大的PC制造商之一采用了两个LVS  
集群系统

红旗Linux和中软

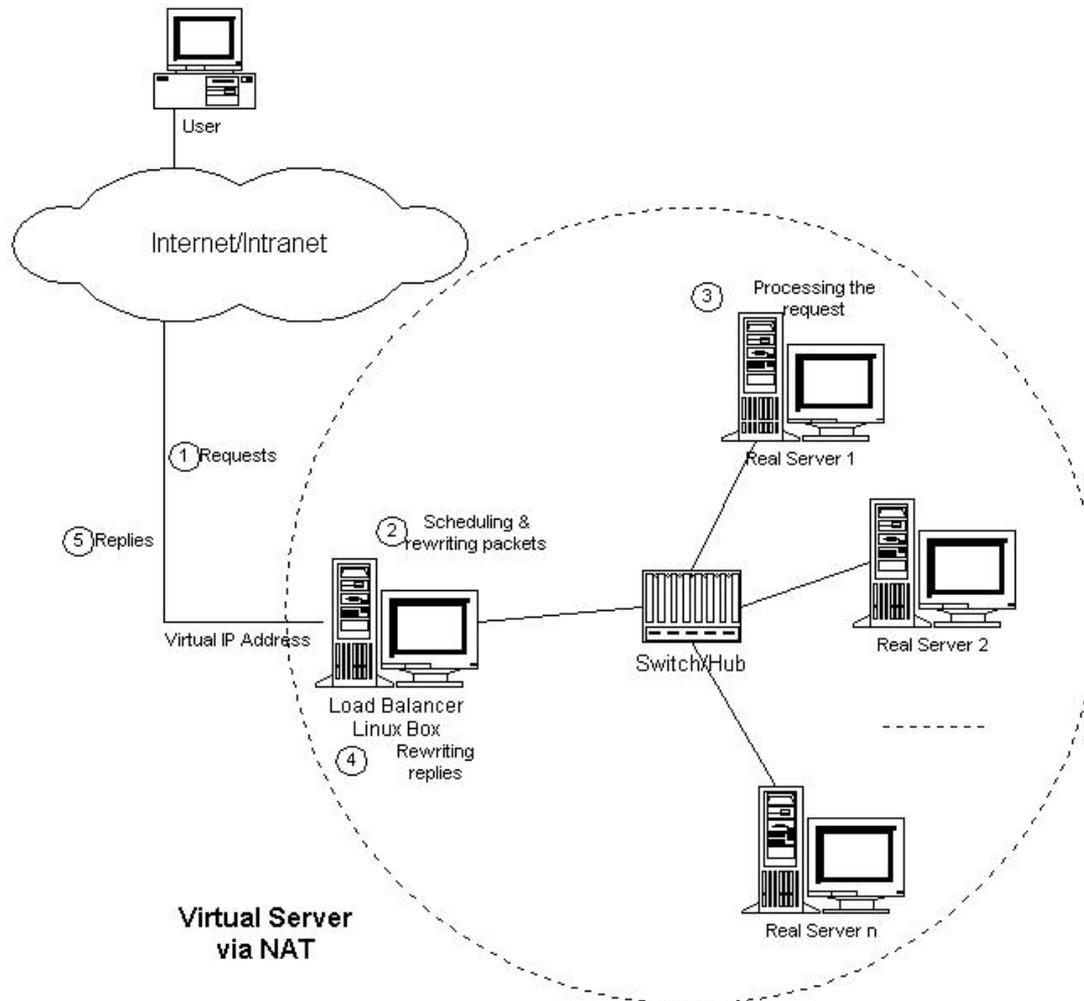
英国150所以上的大学提供Web Cache服务

## 性能

LVS服务器集群系统具有良好的伸缩性，可支持几百万个并发连接。配置100M网卡，采用VS/TUN或VS/DR调度技术，集群系统的吞吐量可高达1Gbits/s；如配置千兆网卡，则系统的最大吞吐量可接近10Gbits/s。

# IP转换-LVS

## 通过NAT实现虚拟服务器 (VS/NAT)



优点：

- 可以运行任何支持TCP/IP的操作系统
- 只需要配置LVS服务VIP，无需配置Real Servers，配置简单

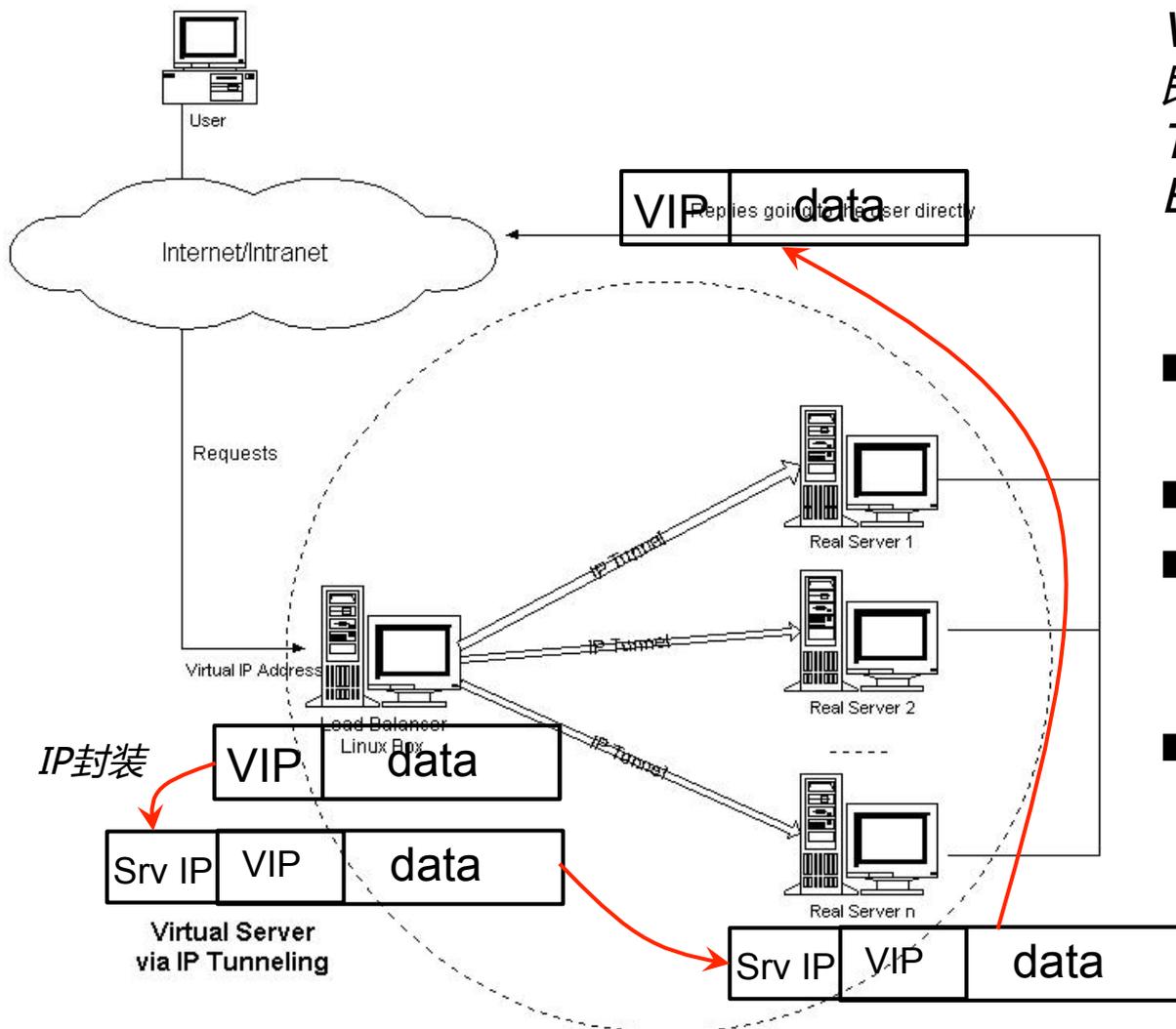
缺点：

- LVS对请求和回复报文修改报文头，所有请求和回复都通过LVS服务，很容易成为瓶颈。



# IP转换-LVS

## 通过IP隧道实现虚拟服务器 (VS/TUN)



VS/TUN技术对服务器有要求，即所有的服务器必须支持“IP Tunneling”或者“IP Encapsulation”协议

- LVS服务需要与所有Real Server建立VPN网络。
- 可以实现端口转发
- 请求报文通过LVS，由LVS对原始报文进行再次封装后转发给后端服务。
- 响应报文直接返回到客户端，无需通过LVS转发。

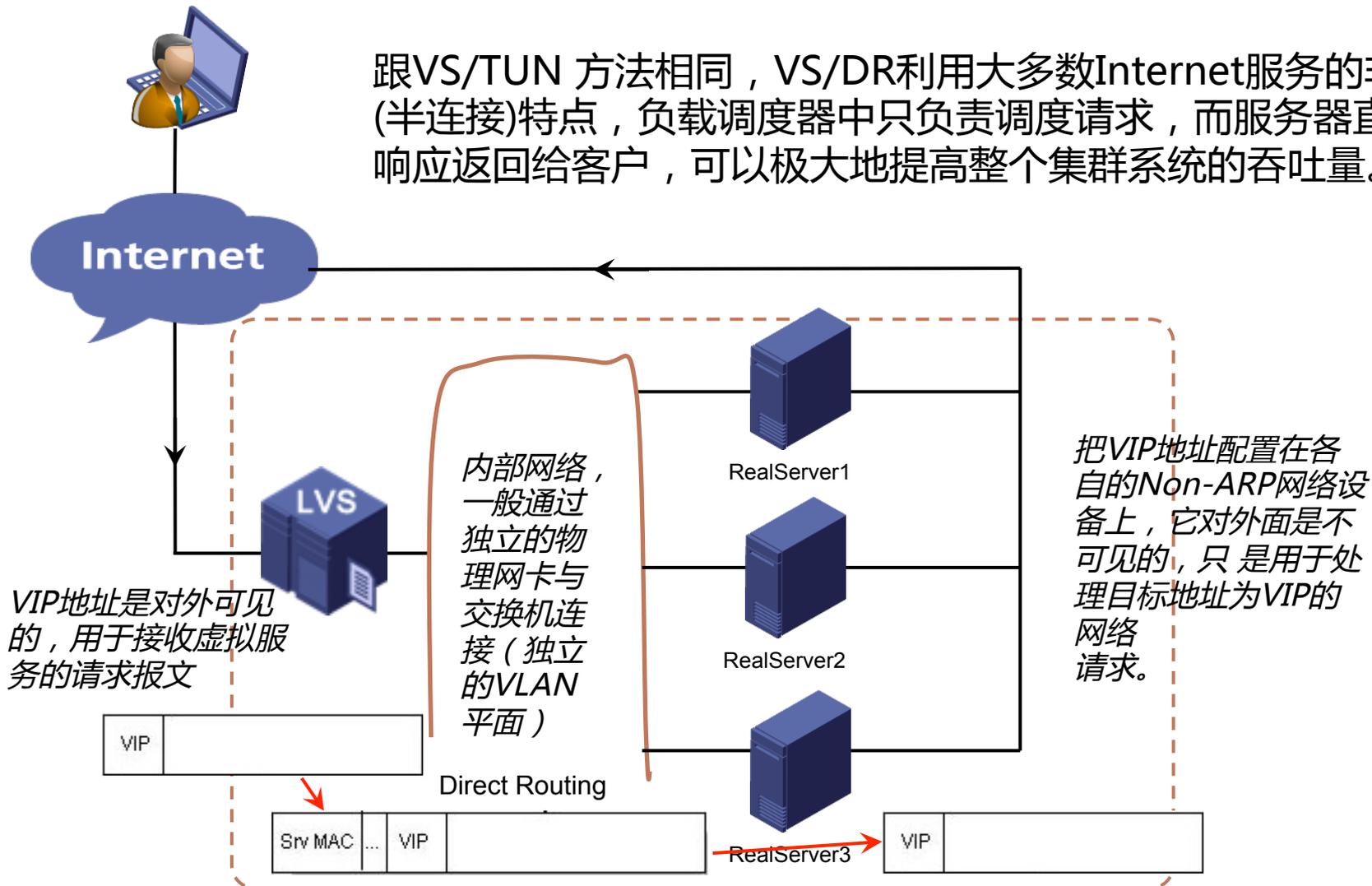
发现VIP是配置在本地IP隧道上，直接处理报文数据



# IP转换-LVS

## 通过直接路由实现虚拟服务器 (VS/DR)

跟VS/TUN 方法相同，VS/DR利用大多数Internet服务的非对称(半连接)特点，负载调度器中只负责调度请求，而服务器直接将响应返回给客户，可以极大地提高整个集群系统的吞吐量。





# 目录

- 负载均衡技术介绍
- 高可用的系统设计
  - 什么是系统高可用性？
  - 高可用的模式
  - 系统高可用设计
  - 高可用设计案例
- 高可用方案及实践

# 什么是系统高可用性？

系统“高可用性”（High Availability）通常来描述一个IT系统经过专门的设计，减少计划和非计划停工时间，保持其服务的高度持续可用性。

系统可用性定义： $MTTF / (MTTF + MTTR) * 100\%$

MTTF：mean time to failure，平均失效前时间，也就是正常运行的时间

MTTR：mean time to restoration，平均恢复前时间，也就是故障时间

系统高可用



影响系统可用性的因素？

- 电源冗余
- 服务器内部架构（包括电子元件是否有冗余等）
- 网卡是否高可用
- 网线是否高可用
- 网络设备是否高可用
- 网络环境是否高可用
- 服务器是否有冗余
- 存储是否高可用
- 支持软件是否高可用
- 应用及服务是否高可用
- ...

综合因素

# 高可用的模式

系统高可用性的常用设计模式包括三种，包括：

## 主备(Active-Standby)

**工作原理：**主机工作，备机处于监控准备状况；当主机宕机时，备机接管主机的一切工作，待主机恢复正常后，按使用者的设定以自动（热备）或手动（冷备）方式将服务切换到主机上运行。一般需要人工干预才能回复初始状态。

**典型案例：**Postgres主备方案

## 互备(Active-Active)

**工作原理：**两台主机（A标记为主，B标记为备）同时运行各自的服务工作且相互监测情况，当任一主机（A）宕机时，另一台主机（B，启用并标记为主）立即接管它的一切工作，保证工作实时可用

**典型案例：**Mysql双机热备

## 集群(Cluster)

**工作原理：**多台具有相同能力的服务同时对外提供透明服务，所有服务之间都是Active-Active关系，并分担处理服务请求，一般通过总控节点或集群软件进行高可用的控制。

**典型案例：**apache+tomcat Web集群；KeepAlived+LVS应用集群等

## 高可用与负载均衡的关系！？

从实际技术应用上看：负载均衡+故障转移≈高可用集群



# 系统高可用设计

以**需求为基础**，考虑影响系统**高可用性的因素**，**切合实际**，综合成本，时间和资源等条件，设计满足高可用需求和指标的系统方案。

需求为基础：

高可用性的设计是以实际应用需求为基础准则的。只有系统的失效足以影响到系统用户的需求时才会成为设计和考虑的范畴。

例如：在一个高访问量的内容型网站系统中，1秒内可以自回复的失败或错误一般是不会被客户感知的；但如果是一个用于大型数据分析的集群服务，1秒的中断可能导致后续分析全部失败，则是不可接受的。

切合实际：

高可用设计切忌生搬硬套（考虑所有的因素），应切合实际，只考虑符合实际情况和常识。例如：某100人级公司内部的信息分享系统，是可以容忍几个小时的非计划停机的，如果去考虑购买多台服务器，搭建高可用集群，考虑网络，设备和电源（如果公司基础设施本不具备的情况）等高可用，保证7\*24小时高可用，就有点得不偿失。

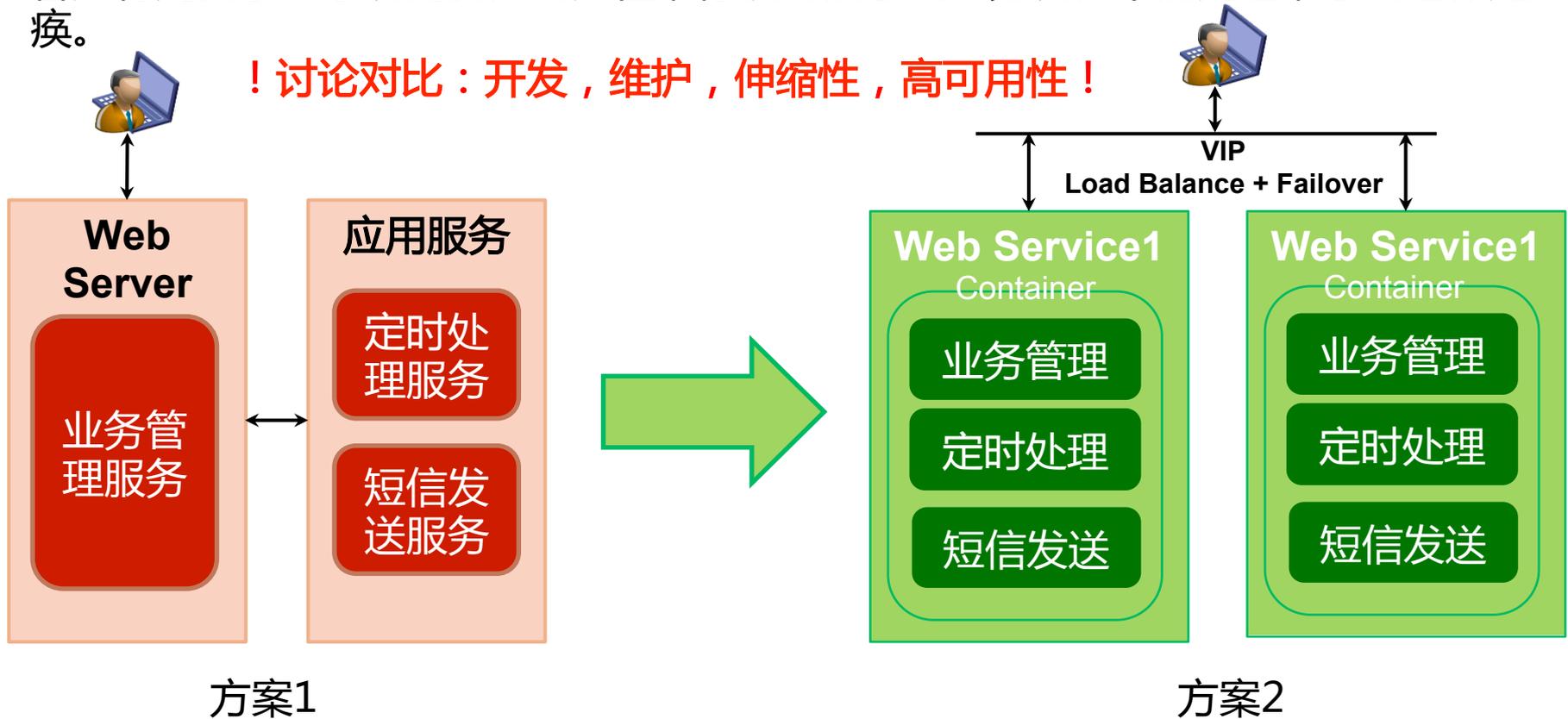
架构设计就像找老婆：没有最好的，只有最合适的，高可用设计也如是

高可用的设计核心手段：去单点

# 高可用设计案例

需求：某公司为客户（100员工）开发一套业务管理系统，用于客户公司内部实现管理和业务流程的自动化。同时提供短信发送功能定时下发每天的待办业务和提醒，客户特别要求该系统需要高可用性，停机可能造成业务流程不能处理，造成运作瘫痪。

**！讨论对比：开发，维护，伸缩性，高可用性！**



设计思路：服务独立，互不影响，宕机后只影响系统部分能力

设计思路：考虑高可用性和扩展性

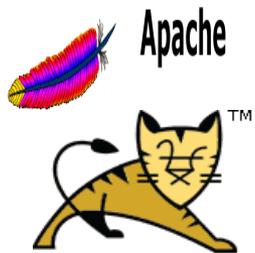


# 目录

- 负载均衡技术介绍
- 高可用的系统设计
- 高可用方案及实践
  - Apache+Tomcat 高可用WEB集群
  - Keepalived +LVS 高可用应用集群
  - MySql双机互备

# Apache + Tomcat实现高可用WEB集群

Apache + Tomcat是J2EE领域最常见和低成本的高可用集群实现方式，同时也是应用最广泛的WEB-HA实现方式之一



APACHE：负载均衡和故障转移；TOMCAT：后端服务

Apache\_proxy通过反向代理与TOMCAT结合，包括：  
AJP代理（二进制，长连接，支持粘性会话）  
http直接代理（http直接转发，不支持粘性会话）

- 方案一: No-Session集群
- 方案二: Sticky-Session集群
- 方案三: Replicate-Session 集群
- 方案四: MSM-Session 集群



# 方案一: No-Session集群

**场景**：平台接口服务（不考虑SESSION问题，每次请求都是原子操作），如：Web Service接口服务（SOAP, REST）

## 优点

- 配置简单，只需配置 Apache Proxy, Tomcat 无需做任务特殊的配置。
- 横向扩展节点容易，支持故障转移和恢复

**问题**：不支持会话同步，只支持无状态的服务。

## Apache 核心配置

```
<VirtualHost  
#.....  
ProxyRequest  
ProxyPreserveHost  
# apache+tomcat  
ProxyPass / http://127.0.0.1:8080/  
ProxyPassReverse / http://127.0.0.1:8080/  
  
<Proxy balancer://mycluster>  
BalancerMember http://127.0.0.1:8080 lbmethod=byrequests  
BalancerMember http://127.0.0.1:8080 lbmethod=bytraffic  
BalancerMember http://127.0.0.1:8080 lbmethod=bybusyness  
ProxySet lbmethod=byrequests  
</Proxy>  
</VirtualHost>
```

**PROXY代理负载均衡算法：**

**参数：lbmethod**

lbmethod=byrequests 轮训

lbmethod=bytraffic 最小流量

lbmethod=bybusyness 最小连接



# 方案二: Sticky-Session集群

**场景：**内部管理或业务系统（弱HA），需要考虑Session有效性的负载均衡场景。

**优点：**粘性会话？解决Session有效性问题，简单高效；最大程度的利用后端服务的缓存。

## Apache-粘性会话(sticky session)

实现一个用户的同一会话的多次请求使用同一台后端服务器支持，简单的说就是前端用户通过浏览器打开界面到关闭浏览器之间的所有请求都由一台TOMCAT服务，这种方式只实现了会话级别的负载均衡。

### 原理：

1. 用户首次请求APACHE时，APACHE通过负载均衡算法找出服务的后端TOMCAT服务器，修改SESSIONID为SESSIONID.route，route表示后端TOMCAT的唯一标志。
2. 在TOMCAT处理完成后，RESPONSE ( SET-COOKIES ) 给客户端浏览器。
3. 客户端浏览器下次请求的时候带上该SESSIONID，APACHE通过请求中的SESSIONID的route选择对应的后端TOMCAT进行服务，实现复制均衡的同时保证SESSION可以。



# 方案三: Replicate-Session 集群

**场景**：中小型业务/管理系统（强HA），支持Session的复制。

**优点**：粘性会话 + Session多播复制，简单解决中小应用中的高可用和负载均衡

**缺点**：多播复制限制了节点数不能过多，效率较低。

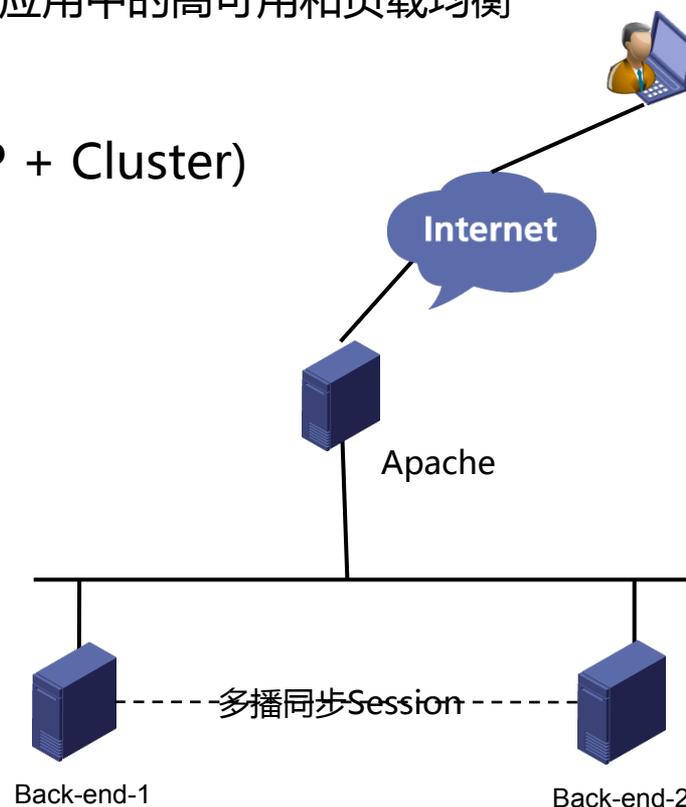
**配置**：Apache ( Proxy + AJP ) + Tomcat(AJP + Cluster)

## Apache配置

请参考方案二，配置完全相同。

## TOMCAT配置

在方案二中，加入Tomcat的Cluster配置，提供后端服务间的Session负载能力。

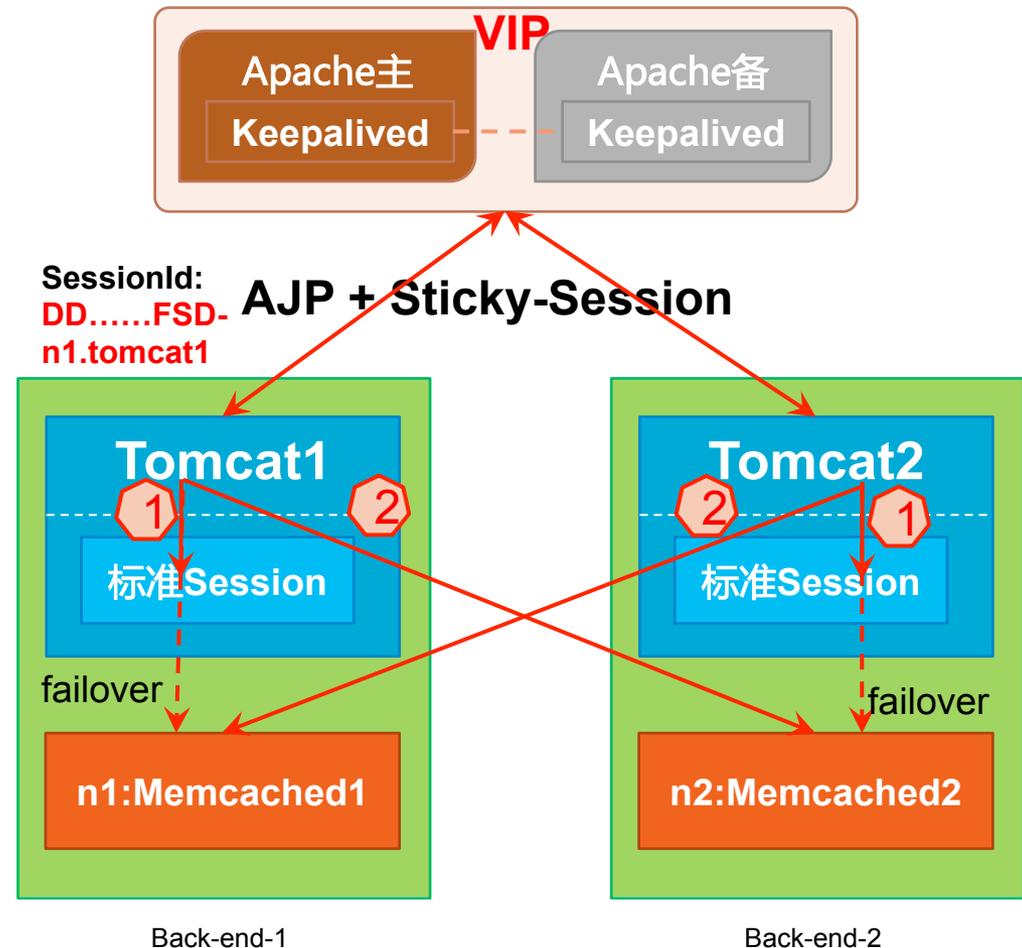


# 方案四: MSM-Session 集群

**场景：**大中型WEB系统。在架构的各层都考虑了高可用。是比较完善的廉价解决方案之一。支持多个TOMCAT节点，节点的扩容的非常方便。可以使用在对可靠性要求比较高的WEB业务系统。

## 方案说明：

- Apache端使用AJP方式连接后端TOMCAT，启用sticky，实现会话级别的负载均衡。
- APACHE端配置支持后端TOMCAT节点的故障转移。
- 可选的APACHE通过keepalived实现2台apache的主备配置，实现apache服务器的高可用
- TOMCAT端使用memcached session manager实现SESSION的共享存储和访问。
- memcached session manager采用sticky方式配置，实现memcached的failover，确保memcached高可用。





# 目录

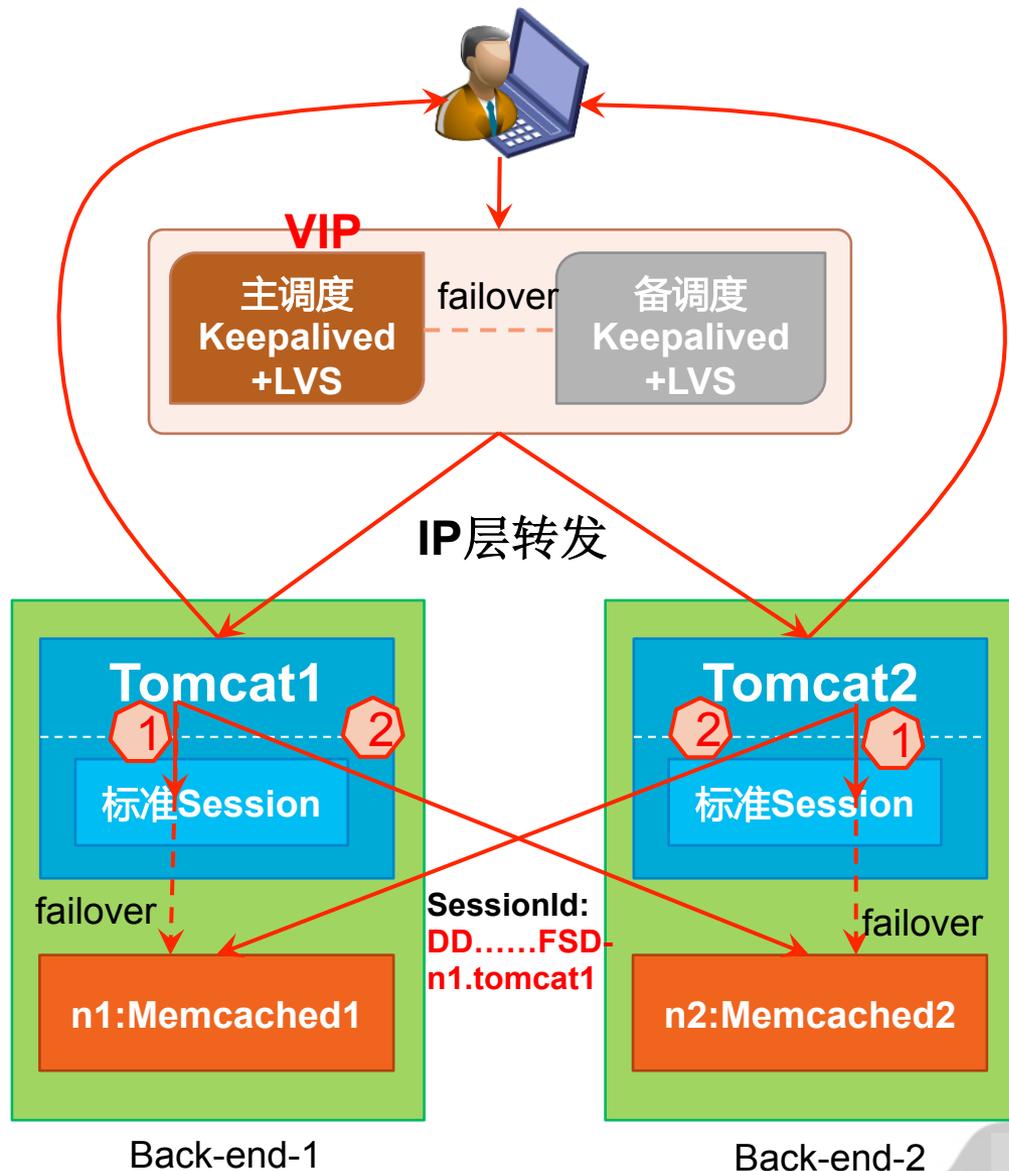
- 负载均衡技术介绍
- 高可用的系统设计
- 高可用方案及实践
  - Apache+Tomcat 高可用WEB集群
  - Keepalived +LVS 高可用应用集群
  - MySql双机互备

# Keepalived + LVS 高可用应用服务集群

**场景：**实用于大中型应用服务的负载均衡和高可用。如：SOA 平台服务，网站，SNS应用，云存储引擎，消息中间件集群等。

**特点：**

- 负载均衡和高可用效率和稳定性较好（内核态程序，IP层转发，半连接）。
- MSM-Session同步方式解决后端服务的性能和高扩展性。
- 应用较广的工业级解决方案。





# 目录

- 负载均衡技术介绍
- 高可用的系统设计
- 高可用方案及实践
  - Apache+Tomcat 高可用WEB集群
  - Keepalived +LVS 高可用应用集群
  - MySQL双机互备

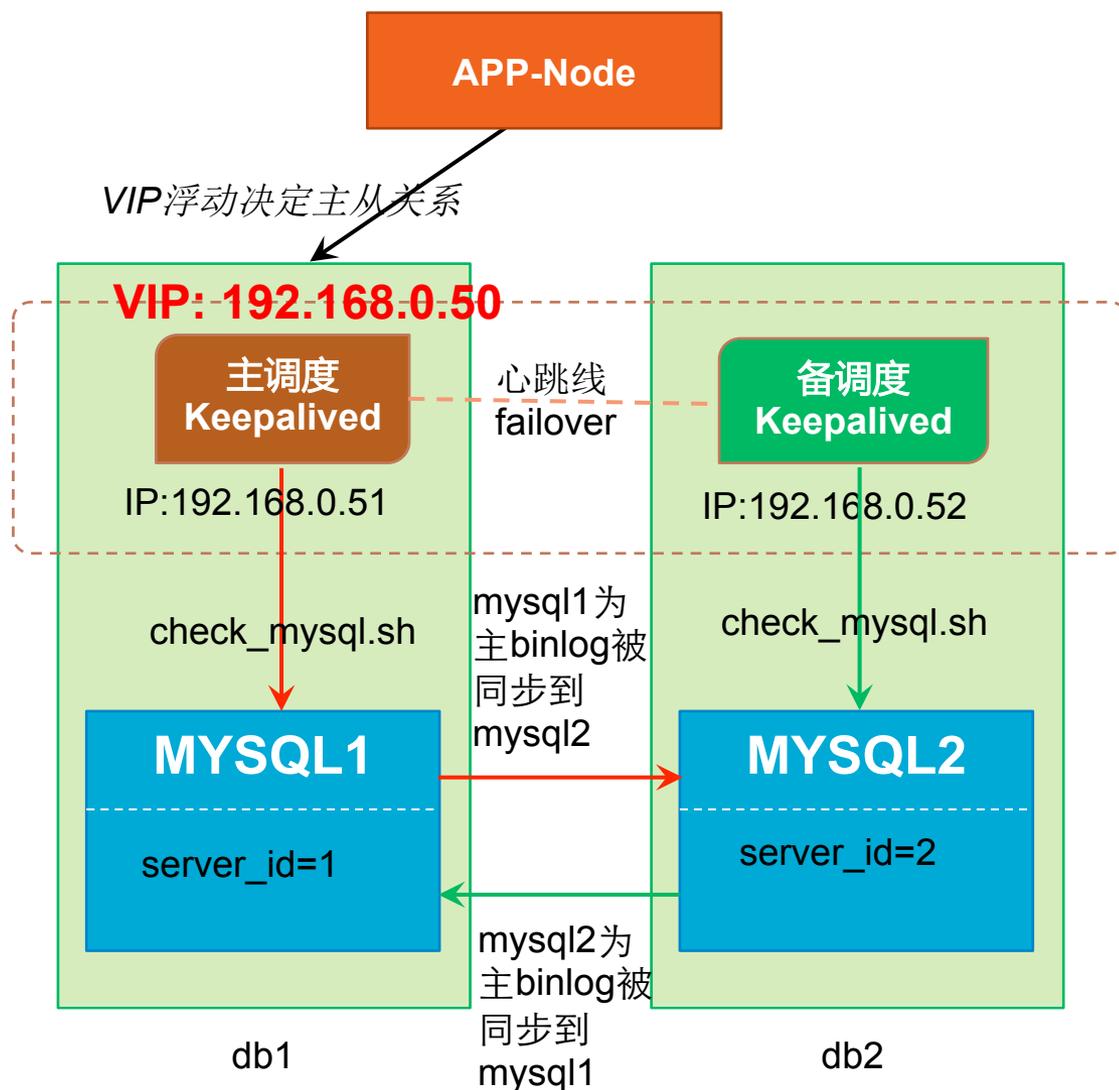


# MySQL双机互备

- 两台linux做双机  
MASTER-MASTER高  
可用
- 采用MYSQL5.6.x的半  
同步实现数据复制和同  
步
- 使用keepalived来监控  
MYSQL和提供VIP及浮  
动。

小经验:

- *Keepalived*: 主备都配置  
为*BACKUP*, 使用优先权  
重来决定初始主。
- *Mysql*: 不要在*my.cnf*中配  
置同步的*master/slave*参数,  
打开*bin-log*就可以。





# Q&A