**dandyhuang**

# 系统设计

## 低耦合

## 高内聚

## 高并发

### 高并发系统有三把斧子：缓存、熔断和限流、其实还有预热



## 依赖倒置

## 轻重分离

## 故障隔离

## 渐进式设计

## 过度设计

## 破窗效应

## 重构

## 互不信任原则

## 协程

## 技术债

# 服务治理

## 过载保护

## 系统降级

## 拨测监控

## 有损服务

## 故障屏蔽

## 异步削峰

## 版本兼容

## 服务发现

### 服务发现组件架构

* 高性能

• 不应该成为使用方服务发现过程中的性能瓶颈

* 高可用

• 组件故障，不会影响调用方的使用

* 数据一致性

• 对于有数据存储的组件，数据的最终一致性还是强一致性，看场景要求。单不能容忍数据丢失

### 对外提供的服务能力

* 服务注册和反注册
* 服务发现

• 调用方可以快速寻址、就近调度、负载均衡和开发简单

• 客户端发现

• 需要把请求发送到服务发现系统的负载均衡器，负载均衡器查询服务注册表，并将请求转发到可用的服务实例。常见的案例有Nginx，优点是服务发现逻辑对使用方是透明的；缺点是所有请求都需要经过负载均衡器，负载均衡器的性能可能会成为服务的瓶颈

• 服务端发现

• 客户端决定相应服务实例的网络位置，并且对请求实现负载均衡。客户端查询服务注册表，后者是一个可用服务实例的数据库；然后使用负载均衡算法从中选择一个实例，并发出请求。常见案例有L5和CMLB，优点是并发性能高；缺点是对使用方来说是代码侵入式的，需要开发调用组件API

* 服务健康检查

### 接入层服务发现

* lVS

• NAT的模式

• 请求和响应报文都必须经过调度器地址重写，调度器的处理能力将成为瓶颈(内网和外网地址的转化等)

• TUN模式

• ip隧道转发

• TGW，即收敛了外网IP，又隐藏了实际业务机器地址，且具备负载均衡、可扩展和可管理的能力

• 用户通过DNS域名解析和GSLB调度，获取到最合适(运营商、地区就近)的VIP

• 户发往VIP的请求会通过IP隧道分发到RS（真实的业务服务器）

• DR

• 改写请求报文的MAC地址，同一物理网段上

• 服务注册

• 需要在TGW申请VIP，并在RS上部署IP隧道

### web server接入

* nginx

• 高性能

• 限于机器IO和配置，其负载度和稳定度差LVS还有几个等级

• 高可用

• Nginx处理所有流量，没有现成的双机热备方案，常用解决方案是在其前端接入DNS或LVS做多机器部署来对其故障机器容灾

• 在对后端保护上，可以有限频（令牌桶和漏桶算法）、限连接数等策略。公司的tnginx还集成了L5，使其服务发现、负载均衡和容灾能力更强大。

### 逻辑层服务发现

* CMLB和L5

• LVS是基于IP层的技术，适用于接入层，成本较高；CMLB和L5则是基于应用层的技术，适用于内部服务发现和负载均衡

• 高性能

• 调用方通过API访问本地Agent数据，直接与服务提供方进行通信，并发性能高。

• 高可用

• Agent有缓存，非业务系统关键路径，CMLB或L5的Server故障不影响业务。但L5的API和Agent是本机网络通信，虽然本地有缓存，但需要定时请求Agent去刷新配置，正常情况下本机网络通信耗时很短，对性能的影响很小。

• 数据一致性

• 最终一致性：CMLB和L5都使用MySql存储静态配置数据，CMLB的Agent定时轮询Server同步数据，L5则是轮询和推送相结合。由于Agent和Server的数据同步时机存在偏差，在服务方发生变更时，Agent之间看到的服务方会在一小段时间内不一致：A. 无状态路由，如加权轮叫，由于服务方任意机器都提供相同服务，所以基本无影响B. 一致性hash寻址，会出现二义性，即一个key在调用方A寻址到的服务方IP和另外一个调用方B寻址到的服务方IP不一样。如果服务方机器间提供服务不一样，可能会存在问题，比如服务方是缓存系统，会导致缓存没有命中或数据不一致

## 服务注册

## 平滑重启

## 故障重放

## 熔断、限流

https://zhuanlan.zhihu.com/p/72128345

### 令牌桶

* 令牌桶算法能够在限制数据的平均传输速率的同时还允许某种程度的突发传输。因为令牌的投放速度是固定的，并且有令牌桶可以暂存没有被拿走的令牌。

### 漏桶

* 漏桶算法能够强行限制数据的传输速率。因为水从桶里漏出的速度是固定的。

# 网络通信

## 断线重连

## 会话保持

## 帧同步

## 状态同步

## 消息队列

### 为什么使用消息队列

* 解耦

• 代码高内聚、低耦合

* 冗余
* 扩展性
* 灵活性和峰值处理
* 顺序保证
* 缓冲

## 远程过程调用

## 协议栈

## 惊群效应

## 拥塞控制

# 运维监控

## 灰度发布

## abtest

## ab实验

## 回滚

## 服务监控

## 快照

## 日志

# 服务异常

## 脑裂

## 缓存穿透

## 数据倾斜

## 雪崩效应

## 毛刺

## 重放攻击

## 网络孤岛

# 架构设计

## 集群

## 主从备份

## 分布式系统

### cap、base理论、ACID性质raft、paxos、akf

## 分布式集群

## 平行扩容

## 服务发现

## 负载均衡

## 缓存

## 路由算法Load Balance

### 一致性哈希

l5一致性hash是带虚拟节点的一致性hash算法的一种实现，算法大致过程如下： 1.服务器列表会配置到一个文件中 (eg: 1.2.3.4:11211 100, 5.6.7.8:11211 100, 9.8.7.6:11211 100) 2.把每个字符串的服务器地址hash成unsigned int 3.把上面的整数放到0~232的区间中，形成一个环，叫continuum 4.用一个数据结构，把hash的整数和ip一一对应起来 5.当需要确定某一个key的位置时，首先对key进行hash，会得到另一个unsigned int un，然后在环continuum上查找大于un的下一个数值。若找到，就把key保存到这台服务器上 6.如果un很大，接近了232，那么把这个key映射到第一个节点上

* 哈希算法好坏的四个定义:

• 平衡性

• 单调性

• 新增或者删除缓冲加入系统中，原有的请求因保证分配到原有的或新的缓冲中。

• 分散性

• 分布式环境中，终端有可能看不到所有缓冲。相同的内容被不同的终端映射到不同的缓冲中。

• 负载

* 子主题 2

### 随机

### 轮询

### lRU

### 加权

## 幂等性

# 系统性能

## 缓存

## 分片

## 压测

### 零钱痛确实是这么压测的。一是流量区分，比如请求头内带某个标识，区分真实流量和压测流量二是数据隔离，这需要借助第一步的流量区分，同时为专门为压测准备相应的测试数据，比如虚拟账号，虚拟城市等，不能影响到正常的业务数据统计，统计时候，1是给订单单独打标，2是可以固定的sql或者系统进行清除数据三是时间节点的选择，一般情况是压线上机器，选择合适的时间点，避免流量高峰期作压测

https://help.aliyun.com/document\_detail/91580.html?spm=a2c4g.11174283.6.658.1eae1b3a59r84I

## 子主题 4

## 子主题 5