数据库缓存

mysql 查询缓存

查询缓存可以看做是 SQL 文本和查询结果的映射,第二次查询的 SQL 和第一次查询的 SQL 全相同,则会使用缓存

表的结构或数据发生改变时,查询缓存中的数据不再有效

配置：

query_cache_type

查询缓存类型,有 0、1、2 三个取值。0 则不使用查询缓存。1 表示始终使用查询缓存。2 表示按需使用查询缓存。

query_cache_size

默认情况下 query_cache_size 为 0,表示为查询缓存预留的内存为 0,则无法使用查询缓存

```
SET GLOBAL query_cache_size= 134217728
```

使用

querycache_type 为 1 时,亦可关闭查询缓存

```
SELECT SQL_NO_CACHE FROM my_table WHERE condition;
```

query_cache_type 为 2 时,可按需使用查询缓存

```
SELECT SQL_CACHE FROM my_table WHERE condition;
```

其他

查看命中次数

```
SHOW STATUS LIKE'Qcache_hits';
```

清理缓存

```
FLUSH QUERY CACHE‖清理查询缓存内存碎片 RESET QUERY CACHE‖从查询缓存中移出所有查询 FLUSH TABLES;//关闭所有打开的表,同时该操作将会清空查询

缓存中的内容
```

redis/memcache 缓存

- Redis,依赖客户端来实现分布式读写
- Memcache 本身没有数据冗余机制

- Redis 支持(快照、AOF),依赖快照进行持久化,aof 增强了可靠性的同时,对性能有所影响
- Memcache 不支持持久化,通常做缓存,提升性能;
- Memcache 在并发场景下,用 cas 保证一致性, redis 事务支持比较弱,只能保证事务中的每个操作连续执行
- Redis 支持多种类的数据类型
- Redis 用于数据量较小的高性能操作和运算上
- Memcache 用于在动态系统中减少数据库负载,提升性能;适合做缓存,提高性能

mysql 优化

数据表数据类型优化

- tinyint、 smallint、 bigint 考虑空间的问题,考虑范围的问题
- char、varchar
- enum 特定、固定的分类可以使用 enum 存储,效率更快
- ip 地址 ip2long() 转成长整型存储

索引优化

索引创建原则

- 索引不是越多越好,在合适的字段上创建合适的索引
- 复合索引的前缀原则

索引注意事项

- 复合索引的前缀原则
- like 查询%的问题
- 全表扫描优化
- or 条件索引使用情况
- 字符串类型索引失效的问题

SQL 语句的优化

优化查询过程中的数据访问

- 使用 Limit
- 返回列不用*

优化长难句的查询语句

- 变复杂为简单
- 切分查询
- 分解关联查询

优化特定类型的查询语句

- 优化 count()
- 优化关联查询
- 优化子查询
- 优化 Group by 和 distinct
- 优化 limit 和 union

存储引擎的优化

尽量使用 Inno DB 存储引擎

数据表结构设计的优化

分区操作

- 通过特定的策略对数据表进行物理拆分
- 对用户透明
- partition by

分库分表

- 水平拆分
- 垂直拆分

数据库服务器架构的优化

- 主从复制
- 读写分离
- 双主热备
- 负载均衡

负载均衡

- 通过 LVS 的三种基本模式实现负载均衡
- MyCat 数据库中间件实现负载均衡

web 服务器的负载均衡、请求分发

七层负载均衡实现

基于 URL 等应用层信息的负载均衡

Nginx 的 proxy 是它一个很强大的功能,实现了 7 层负载均衡

nginx 负载均衡

优点

功能强大,性能卓越,运行稳定
配置简单灵活

能够自动剔除工作不正常的后端服务器
上传文件使用异步模式
支持多种分配策略,可以分配权重,分配方式灵活

Nginx 负载均衡策略

内置策略: IP Hash、加权轮询
扩展策略:fair 策略、通用 hash、一致性 hash

加权轮询策略

首先将请求都分给高权重的机器,直到该机器的权值降到了比其他机器低,才开始将请求分给下一个高权重的机器

当所有后端机器都 down 掉时,Ngnx 会立即将所有机器的标志位清成初始状态,以避免造成所有的机器都处在 timeout 的状态

IP Hash 策略

Nginx 内置的另一个负载均衡的策略,流程和轮询很类似,只是其中的算法和具体的策略有些变化
IP Hash 算法是一种变相的轮询算法

fair 策略

根据后端服务器的响应时间判断负载情况,从中选出负载最轻的机器进行分流

通用 Hash、一致性 Hash 策略

通用 hash 比较简单,可以以 Nginx 内置的变量为 key 进行 hash,

一致性 hash 采用 Nginx 了内置的一致性 hash 环,支持 memcache

配置

```
http {
    upstream cluster {
        server srvl;
        server srv2;
        server srv3;
    }
    server {
        listen 80;
        location /
        proxy_pass http: //cluster;
    }}
```

四层负载均衡实现

通过报文中的目标地址和端口,再加上负载均衡设备设置的服务器选择方式,决定最终选择的内部服务器

LvS 实现服务器集群负载均衡有三种方式,NAT,DR 和 TUN