主办方：msup | ARCHNOTES 架构

GIAC

全球互联网架构大会

GLOBAL INTERNET ARCHITECTURE CONFERENCE

海量数据和高并发下的Redis优化实践

钱文品　掌阅资深工程师

一、Key Value 缓存

```
# 缓存用户信息
def get_user(user_id):
    user = redis.get(user_id)  # 先查缓存
    if not user:
        user = db.get(user_id)
        redis.setex(user_id, ttl, user)  # 缓存数据
    return user

def save_user(user):
    redis.setex(user.id, ttl, user)
    db.save_async(user)  # 异步持久化

# 扩容
codislredis-cluster
```

一、Key Value 缓存

缓存模式

config set maxmemory 20gb

淘汰策略

config set maxmemory_policy allkeys-lru（淘汰范围 — 淘汰算法）

1. no-eviction 禁止写：不招人了
2. volatile-xxx 范围：临时工
3. allkeys-xxx 范围：所有的key（一律平等）
4. xxx-random 算法：摇号（CEO也可能被裁）
5. xxx-lru 算法：LRU（是考核最近一次成绩）
6. xxx-lfu 算法：LFU（考核平时的成绩）

二、分布式锁

用户积分（经验）体系

user_state = json.parse(redis.get(user_id))

user_state.exp += delta_for(user_state, input_event)

redis.set(user_id, jsonify(user_state))

加锁
set "lock:$user_id" owner_id nx ex=5 加 5s 的锁

owenr_id： 所有权标识，打上自己的印记（当前节点的线程）

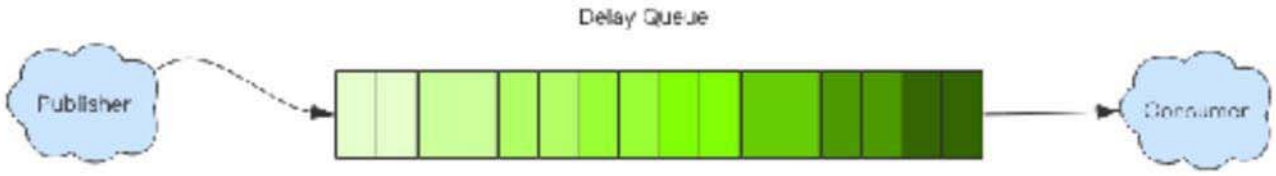
释放锁
del_if_equals lock:$user_id owner_id
```
if redis.call("get", KEYS[1]) == ARGV[1] then
    return redis.call("del", KEYS[1])
else
    return 0
end
```

为什么没有使用 redlock?

三 、延时队列

锁碰撞怎么办？使用延时队列延后处理（zset）

delay(5, task)



生产延时消息

zadd(queue-key, now_ts+5, task_json)

消费延时消息

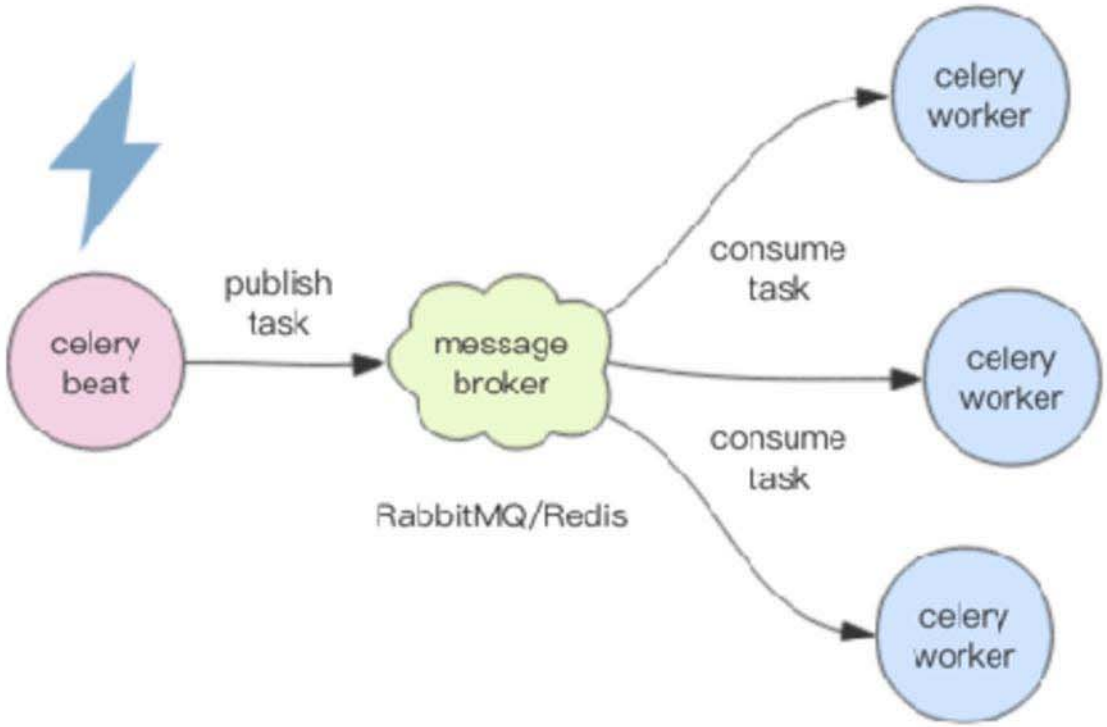
```
task_json = zrevrangebyscore(queue-key, now_ts, 0, 0, 1)
if task_json:
    grabbed_ok = zrem(queue-key, task_json)
    if grabbed_ok:
        process_task(task_json)
```

Lua 脚本消灭竞争浪费

四 、分布式定时任务

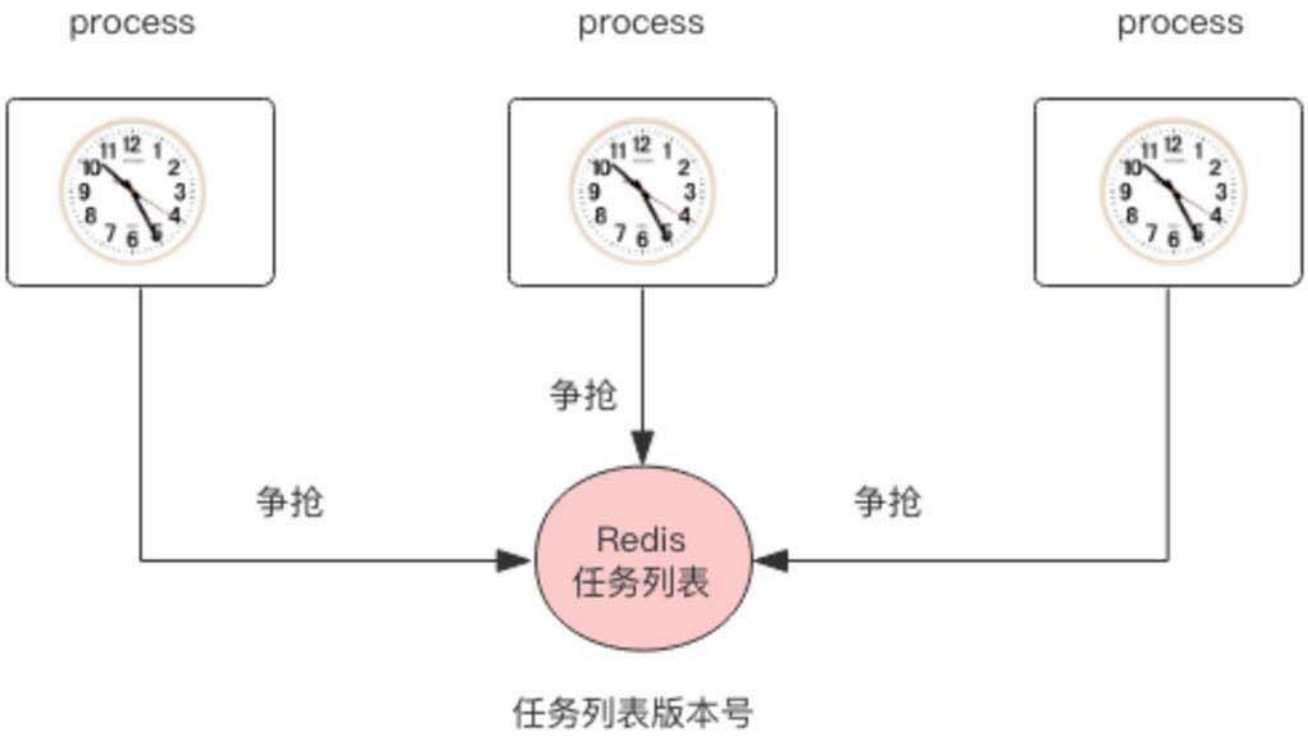
master-workers 模式

python celery（master单点）



四 、分布式定时任务

multi-master(worker)模式



四 、分布式定时任务

Redis 中存储任务信息（定时信息）

hset tasks name trigger_rule

加载任务列表，开启定时器

hgetall tasks

任务到点，争抢任务（机器时间务必同步）

set task_lock_${name} true nx ex=5

任务变更（滚动升级）、watch 任务变化

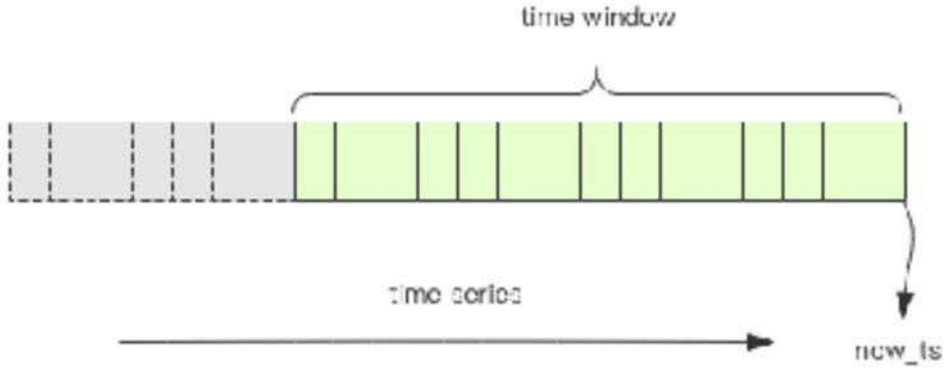
set tasks_version $new_version

get tasks_version

五 、频率控制

UGC 发帖频率限制（一段时间内的数量）

1小时最多5贴，zset 记录时间序列



hist_key="ugc:$user_id"

zadd(hist_key, ts, uuid) # 记录时间序列

zremrangebyscore(hist_key, 0, now_ts - 3600) # 移除时间窗口外的动作

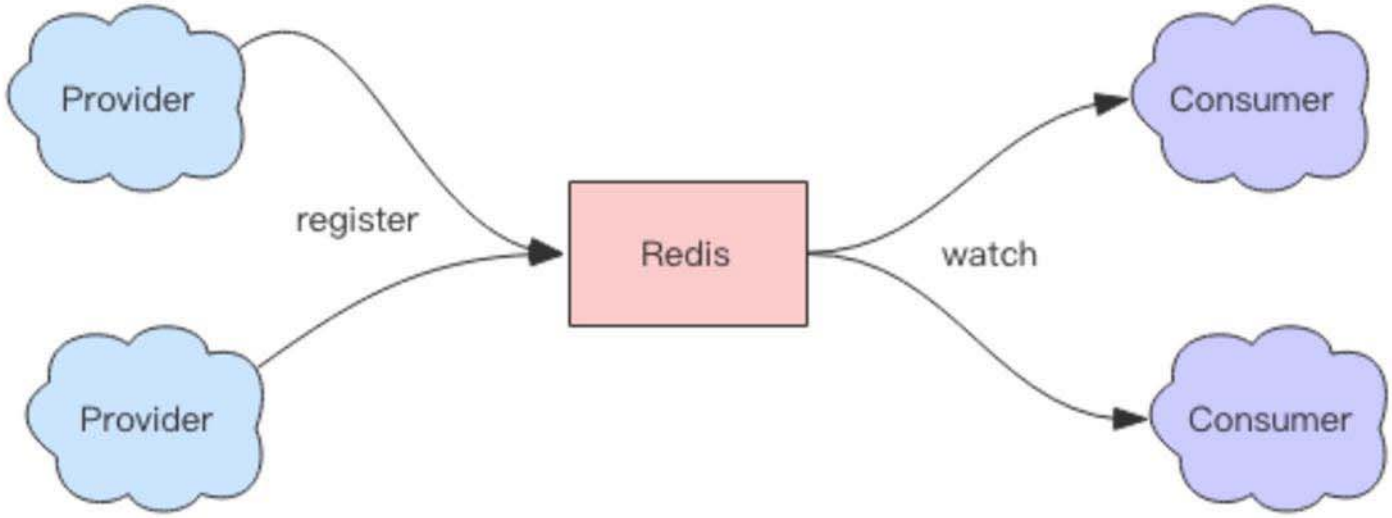
count=zcard(hist_key) # 统计窗口内的动作数量

expire(hist_key, 3600) # 设置时间序列的整体过期时间

事务管道原子执行

六、服务发现

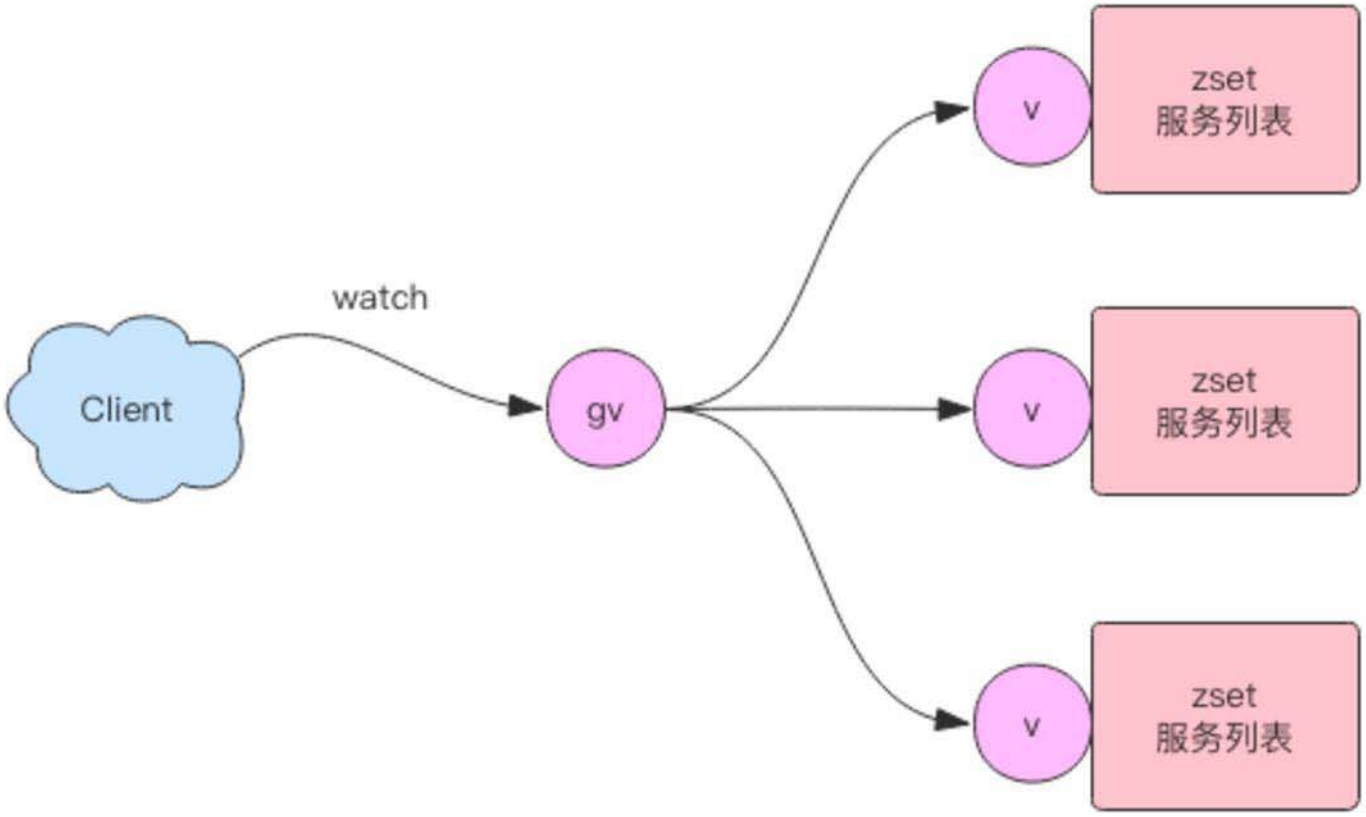
注册取消服务、获取服务列表、监听服务列表变更



六 、服务发现

```
# 注册服务列表（zset）
zadd $service_key heartbeat_ts addr

# 获取服务列表
zrange $service_key 0 -1

# 正常停机 vs 异常停机处理(单独的线程)
zrem $service_key addr
zremrangebyscore $service_key 0 now_ts - 10   # 10s 过期

# 监听服务列表变更（版本号）
incr version:$service_key  # 单个服务列表版本号
incr gversion  # 全局服务列表版本号
```

六、服务发现



七 、位图

签到系统

0 未签到 1 已签到 2 补签

hset sign:$user_id 2019-01-01 1

hset sign:$user_id 2019-01-05 1

hset sign:$user_id 2019-01-10 2

...

空间浪费严重

七、位图

位存储

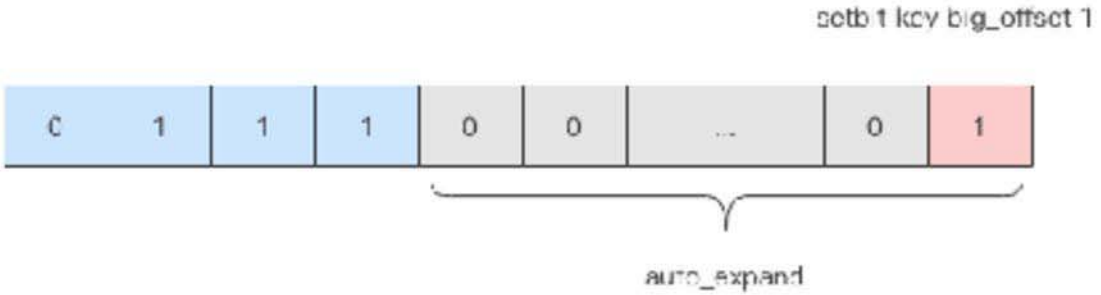
00 未签到 01 已签到 10 补签 11 保留

一个月的签到信息空间占用

28*2~31*2 bits 7～8个字节

空间效果显著

30G => 10G



位图的底层是字符串

位图会自动扩展（dangerous）

咆哮位图

八 、模糊计数

签到日活、月活统计、文章阅读数（去重）

set 空间浪费严重

Redis HyperLogLog

pfadd sign_uv_${day} user_id

pfcount sign_uv_${day}

稀疏I密集位图 空间占用最多12k

概率计数 误差率 0.81%

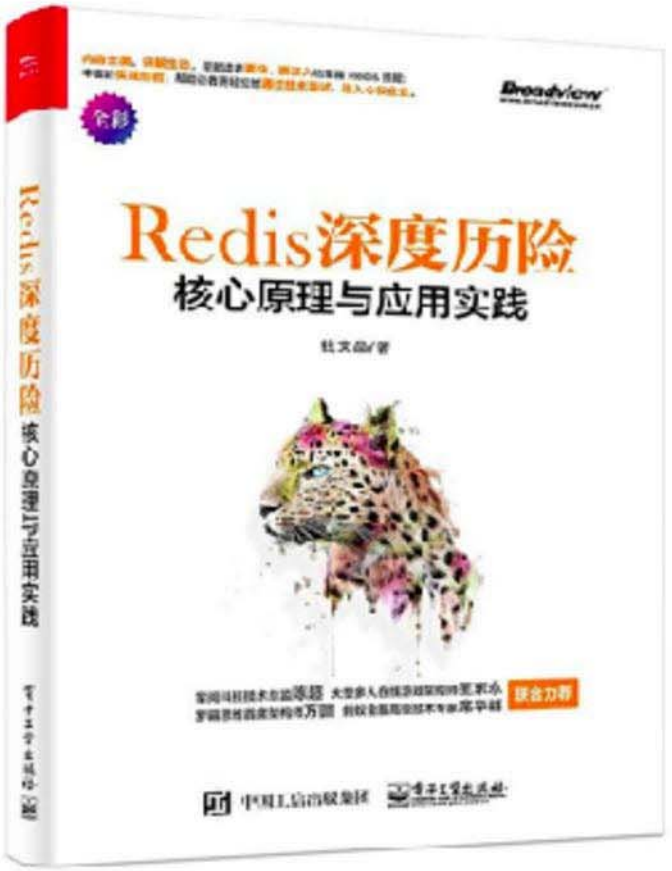
不支持 remove 反向操作、不支持 contains 查询操作

九 、布隆过滤器

```
# 新上子系统的缓存穿透问题（大量新用户）
# 缓存穿透攻击（使用不存在的 user_id）
def get_user_state0(user_id):
    state = cache.get(user_id)
    if not state:
        state = db.get(user_id) or {}
        cache.set(user_id, state)
    return state

def save_user_state0(user_id, state):
    cache.set(user_id, state)
    db.set_async(user_id, state)

# 大量无效的 db 查询
```

九 、布隆过滤器

```
# 空间换时间，可以挡住 99% 的无效 db 查询
# 相当于一个压缩的 set
def get_user_state(user_id):
    exists = bloom_redis.exists(bloom_key, user_id)
    if not exists:
        return {}
    return get_user_state0(user_id)

def save_user_state(user_id, state):
    save_user_state0(user_id, state)
    bloom_redis.add(bloom_key, user_id)
# 基于Redis的原生位图实现：https://github.com/seomoz/pyreBloom
# 基于Redis Module实现：https://github.com/RedisBloom/RedisBloom
# 布谷鸟过滤器
```



微信搜索【码洞】关注后回复 Redis 获取文字稿