目录

[1. 多线程 2](#_Toc4139)

[2. Mysql 5](#_Toc22847)

[3. Redis： 6](#_Toc30779)

[数据加载在内存中，查询单线程，多个客户端连接顺序执行。 6](#_Toc19979)

[1. 持久化： 6](#_Toc4097)

[1. 从节点发起psync命令，发起命令之前会和主节点建立socket长连接 7](#_Toc11485)

[2. 主节点收到psync命令，执行bgsave生成rdb快照数据 7](#_Toc26435)

[3. 主节点send rdb到从节点 7](#_Toc6544)

[4. 从节点清空老数据并加载主节点rdb文件 7](#_Toc24791)

[5. 主节点后续有新的命令会实时send buff发送命令到从节点 7](#_Toc10347)

[6. 从节点执行buffer里的写明了到内存 7](#_Toc4518)

[7. 主节点通过socket长连接持续把写明了发送给从节点，保证主从数据一致性 7](#_Toc3455)

[1. RedisK-V底层设计原理 8](#_Toc19716)

[2. Redis渐进式rehash及动态扩容机制 8](#_Toc10294)

[3. Redis核心编码结构 8](#_Toc28814)

[4. 亿级用户日活统计BitMap实战及源码分析 8](#_Toc21543)

[4. Zookeeper 18](#_Toc21548)

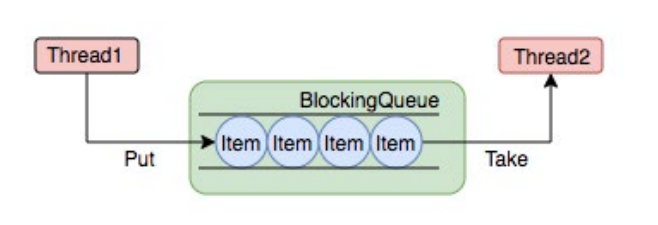
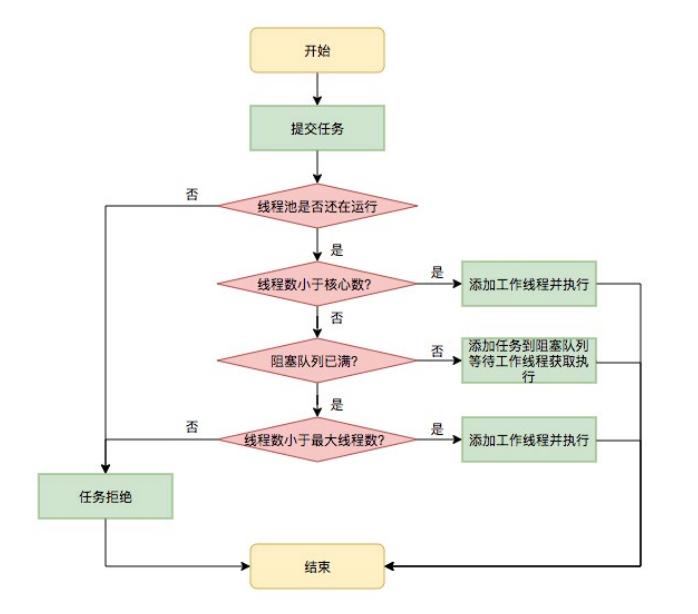
[5. Dubbo 19](#_Toc29089)

[6. 分布式 19](#_Toc9960)

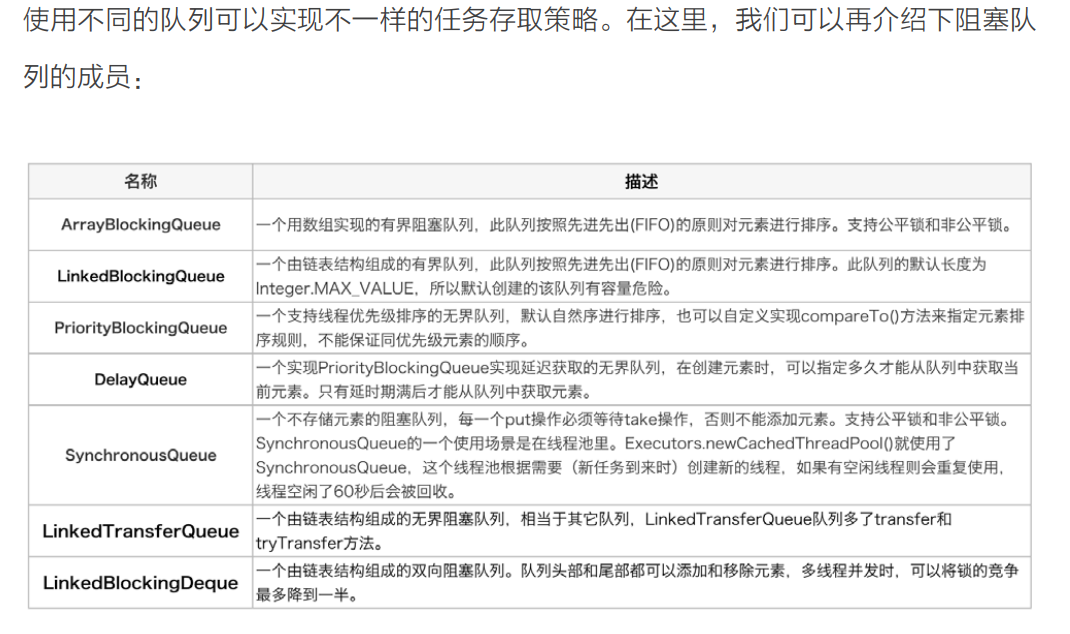
[7. MQ 19](#_Toc27541)

# 多线程

## 1.1 线程池执行流程：



线程1往阻塞队列中添加元素，线程2从阻塞队列中移除元素



# Mysql

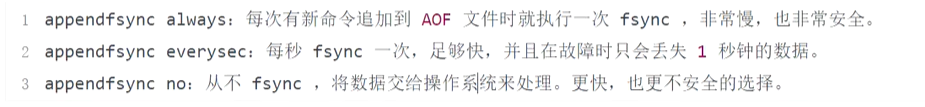
## 2.1 Mysql索引：

1. 普通索引：
   * 1. 普通索引允许字段值重复
2. 唯一索引(unique)：
   * 1. 唯一索引可以确定字段值唯一
3. 联合索引

# Redis：

**数据加载在内存中，查询单线程，多个客户端连接顺序执行。**

1. 持久化：
   * 1. AOF
        1. Redis每执行一次修改命令都追加到appendonly.aof文件中
        2. 从aof恢复数据，重启后把appendonly.aof文件中的命令按顺序执行
        3. 数据比较大（几十万条），恢复速度比较慢



* + - 1. AOF重写，AOF会根据最新内存重写appendonly.aof文件



AOF重写redis会fork一个子进程去做(与bgsave命令类似)，不会对redis正常命令处理有太多影响

* + 1. RDB
       1. Save 60 10000 # 配置命令
          1. Sava

同步保存，会阻塞线程

主线程操作

* + - * 1. Bgsave

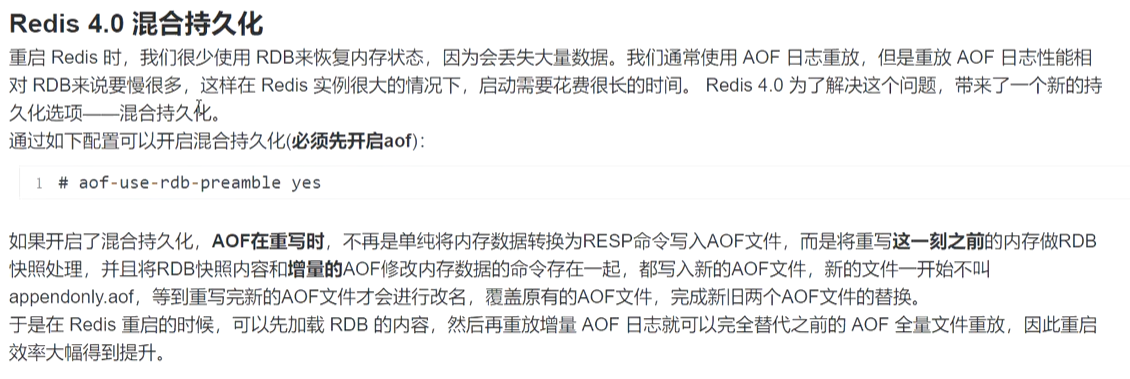
异步保存，不会阻塞线程

子线程操作

新增数据会实时同步



Redis4.0混合持久化



**Redis主从，哨兵，集群**

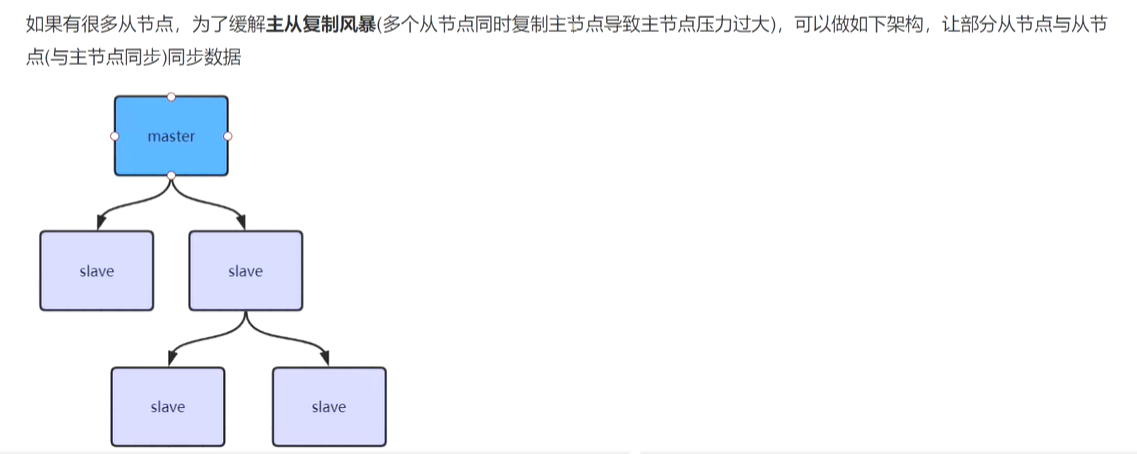
主从工作原理：

1. 从节点发起psync命令，发起命令之前会和主节点建立socket长连接
2. 主节点收到psync命令，执行bgsave生成rdb快照数据
3. 主节点send rdb到从节点
4. 从节点清空老数据并加载主节点rdb文件
5. 主节点后续有新的命令会实时send buff发送命令到从节点
6. 从节点执行buffer里的写明了到内存
7. 主节点通过socket长连接持续把写明了发送给从节点，保证主从数据一致性

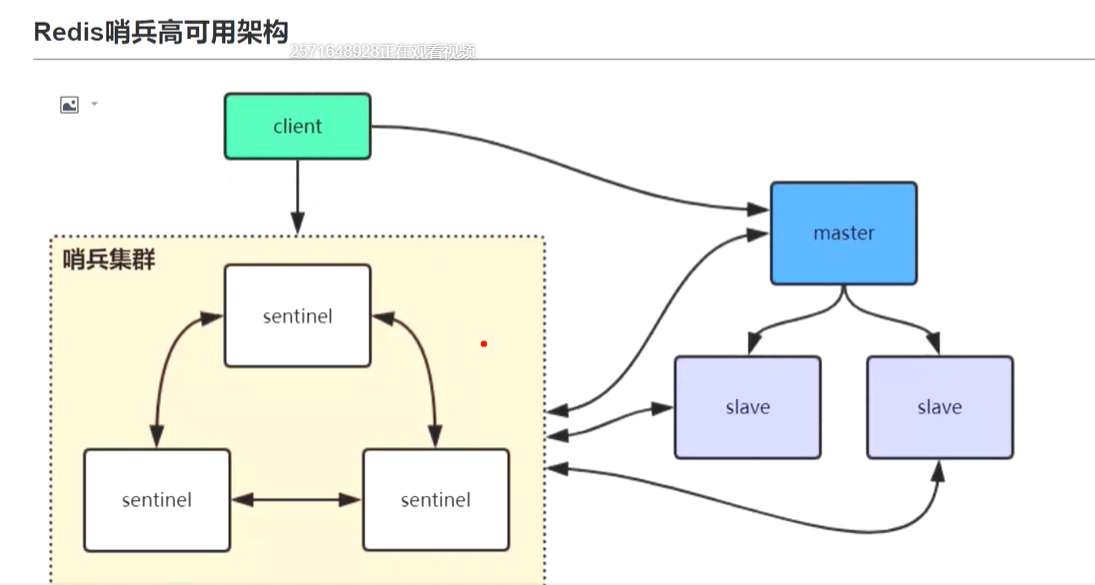
主节点会有一个缓冲区，保存最近的数据

从节点挂掉后重启记录上次同步的偏移量(offset)，主节点将缓冲区offset后的数据发送给从节点，增量同步

主从复制风暴(多个从节点同时复制主节点导致主节点压力增大)



**Redis单进程，单线程执行脚本，管道不会阻塞redis。**

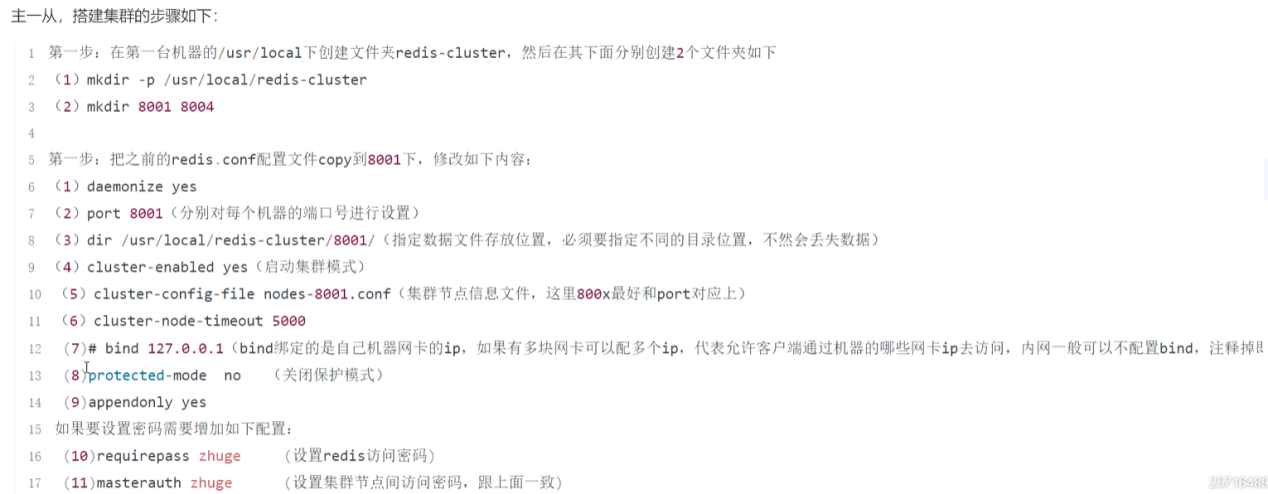


哨兵也是redis实例，不对外提供读写功能

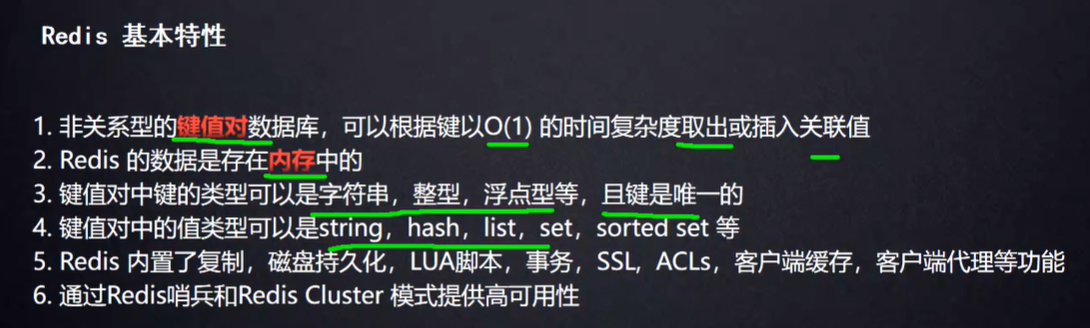
客户端连接哨兵，第一次动哨兵获取从节点信息，后面直接连接从节点，当master节点发生变动，会主动推送给客户端，客户端接收到服务端订阅事件，重新从哨兵获取从节点信息。

超过半数哨兵认为主节点挂掉才会重新选择主节点，否则出现脑裂问题

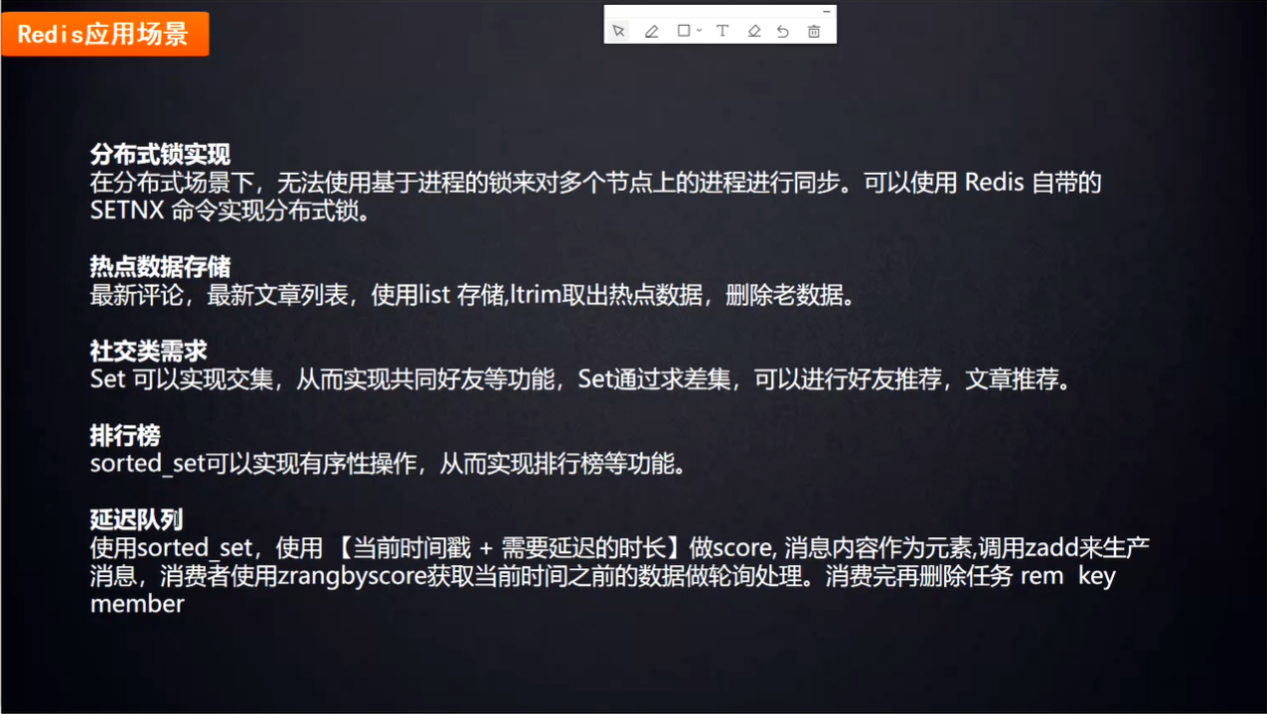
Redis集群搭建

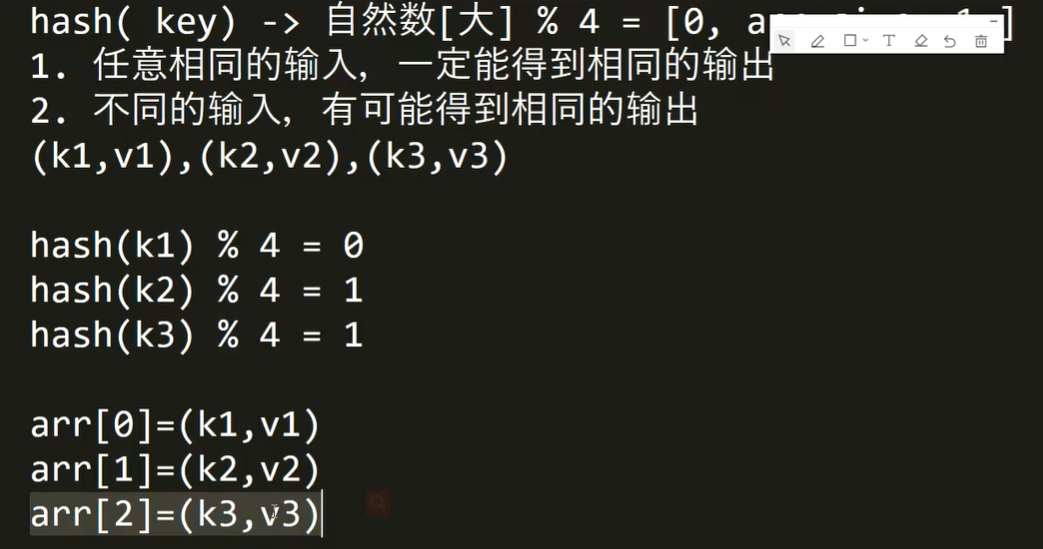


1. RedisK-V底层设计原理
2. Redis渐进式rehash及动态扩容机制
3. Redis核心编码结构
4. 亿级用户日活统计BitMap实战及源码分析









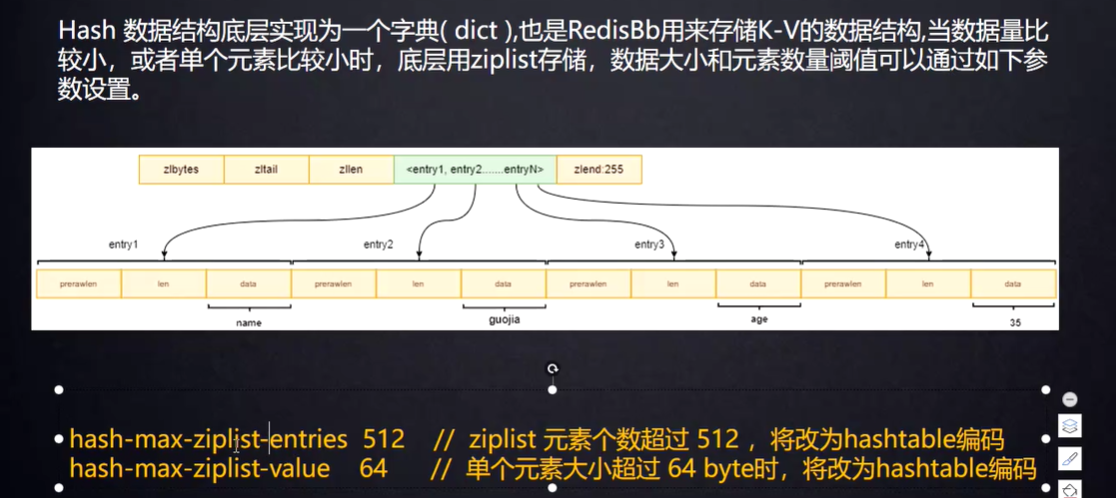
## 3.1 Redis结构：

1. List:

List是一个有序(按加入的时序排序)的数据结构，Redis采用quicklist(双端链表)和ziplist作为List的底层实现。

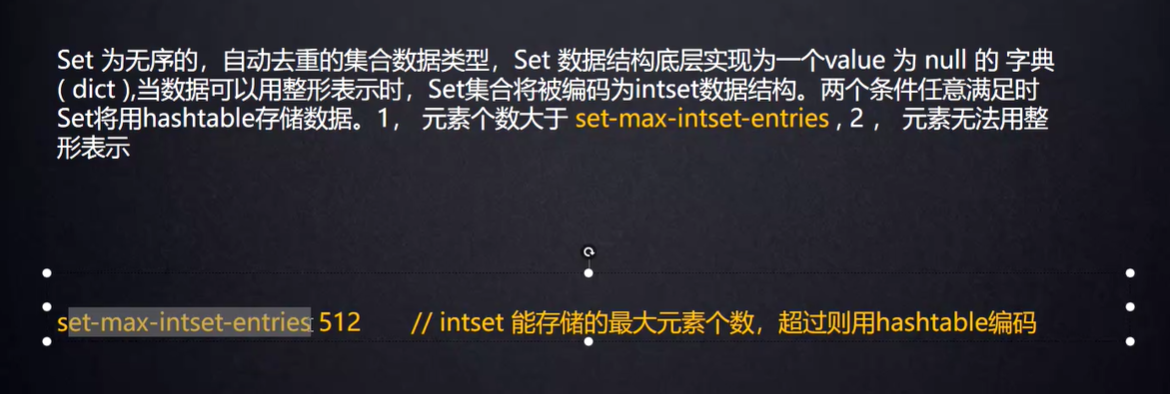
1. Hash

Hash数据结构底层实现是一个字典(dict)也是RedisDb用来存储k-v的存储结构，当数据量比较小，或者单个元素比较小时，底层用ziplist存储，数据大小和元素数量阈值可以通过如下参数设置：



1. Set

Set是无序的，自动去重的集合数据类型，Set数据结构底层实现为一个value为null的字典(dict)，当数据可以用整形表示时，Set集合将被编码为intset的数据结构，两个条件任意满足时，Set将用hashtable存储数据，元素个数大于set-max-intset-entries，2，元素无法用整形表示



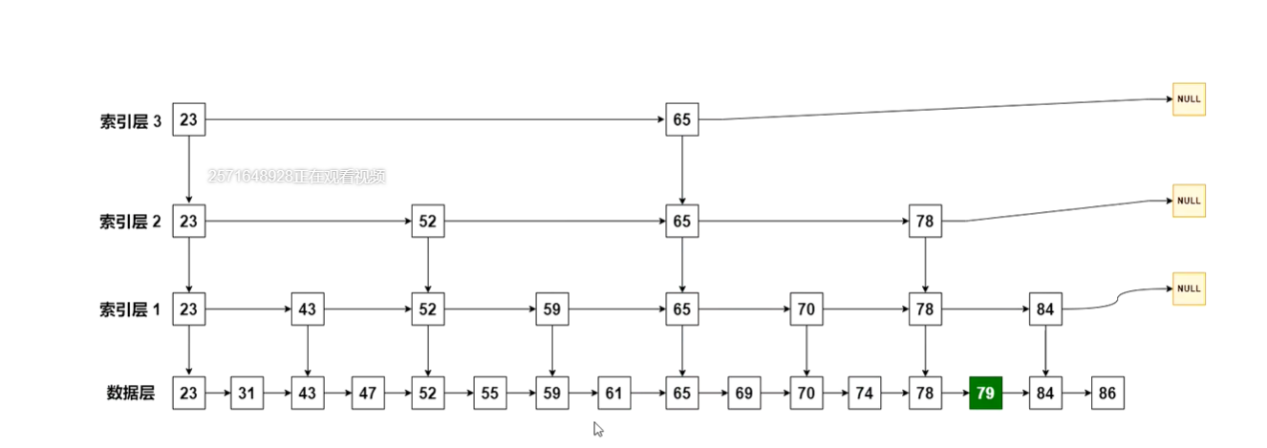
1. ZSet

ZSet为有序的，自动去重的集合数据类型，ZSet数据结构底层实现为字典(dict) + 调表(skiplist)当数据比较少时，用ziplist编码数据结构存储。



1. SkipList

SkipList底层是链表结构实现



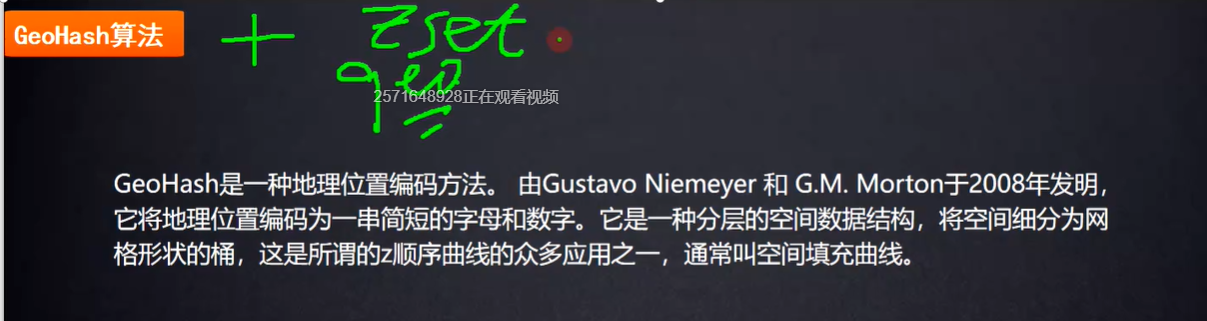
创建索引：每次二分直到索引数为2



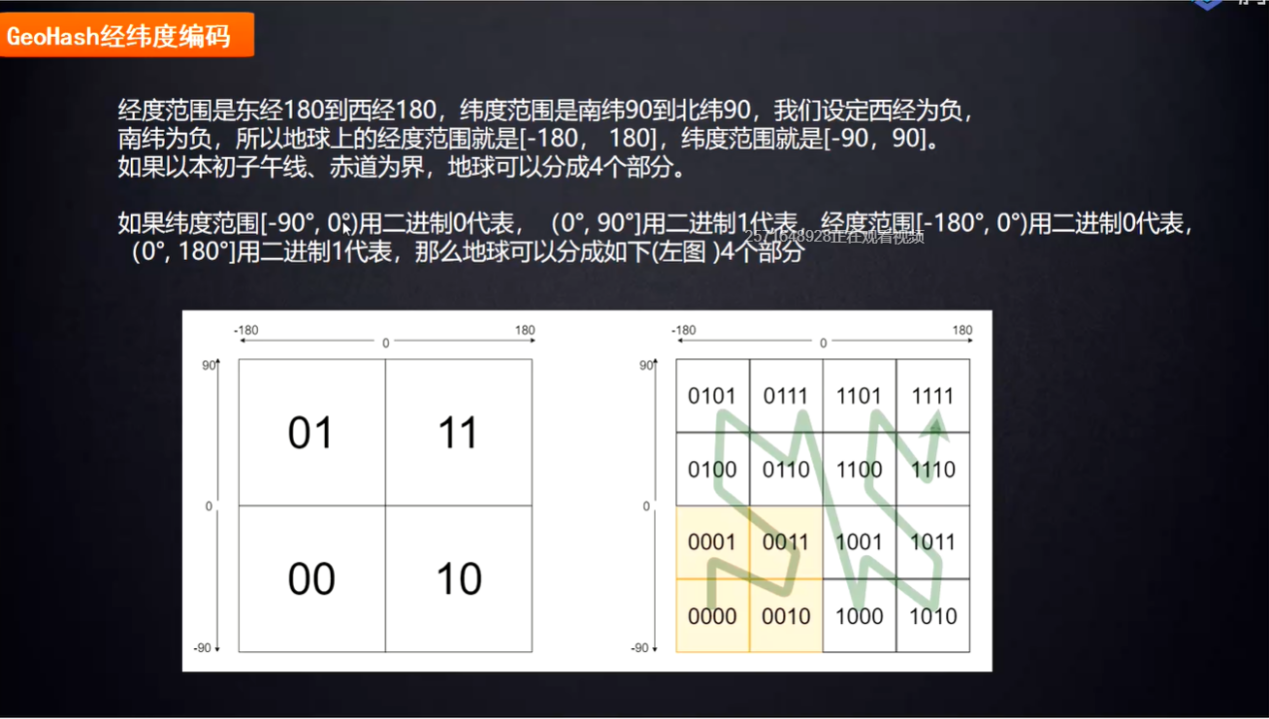
1. GeoHash算法(geo+zset)

GeoHash是一种地理位置编码方式，由Gustavo Niemeyer和G.M Morton于2008年发明，它将地理位置编码为一串简短的字母和数字，它是一种分层的空间数据结构，将空间细分为网格形状的桶，这是所谓的z顺序曲线的众多应用之一，通常叫空间填充曲线。

查询地理位置，地点距离，附近的人

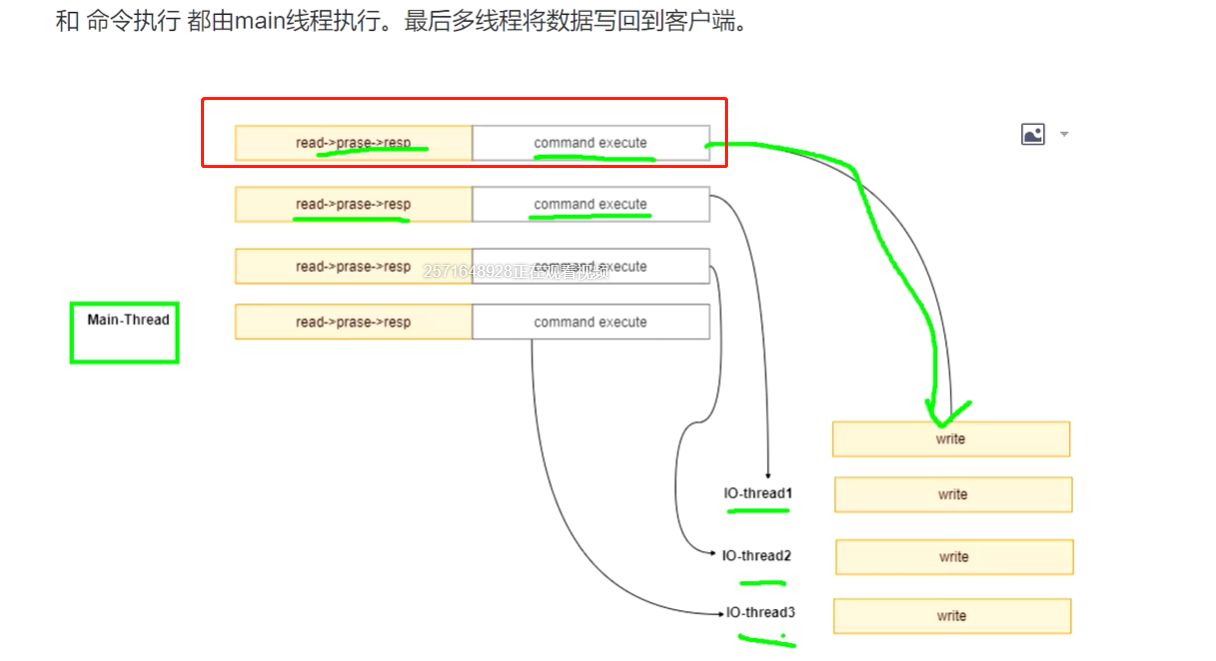


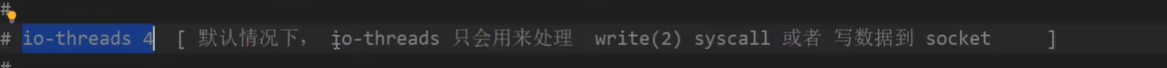
1. GeoHash经纬度编码



## 3.2 Redis6.0 新特性：

6.0以前单线程

6.0 新特性



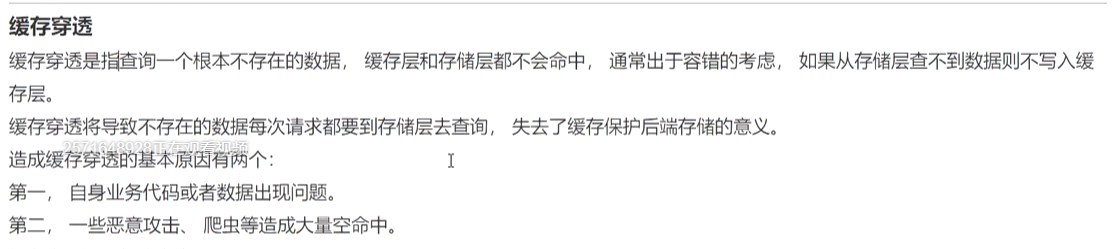


1. 支持多线程(默认关闭)
2. 客户端缓存

单机环境下支持，集群不支持

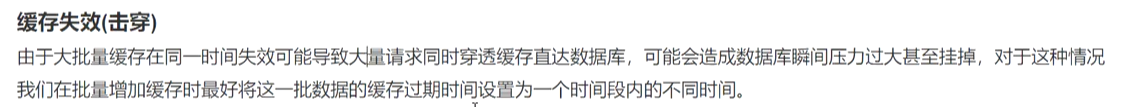
## 3.3 Redis缓存设计与性能优化

1. 多级缓存架构详解
2. 缓存穿透&缓存击穿&缓存雪崩详解
   1. 缓存穿透：查询不存在的数据



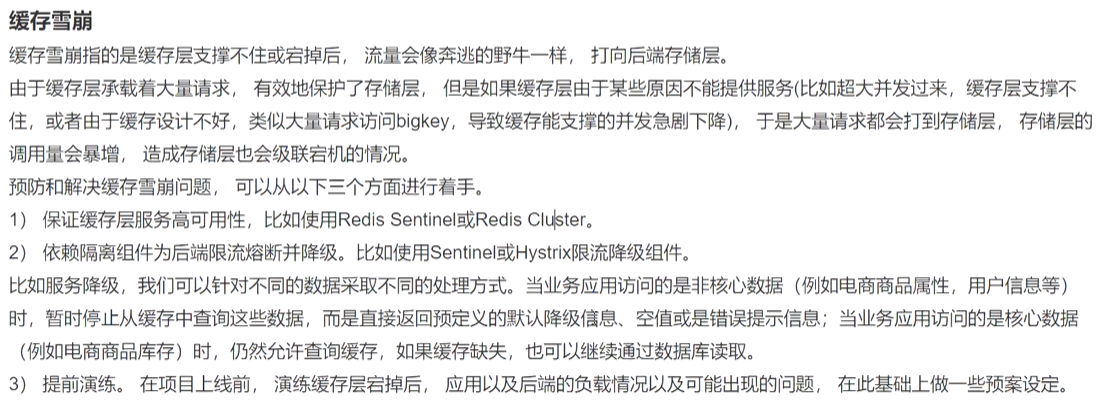
解决方法：

1. 缓存空对象(设置过期时间)
2. 布隆过滤器
   1. 缓存击穿：



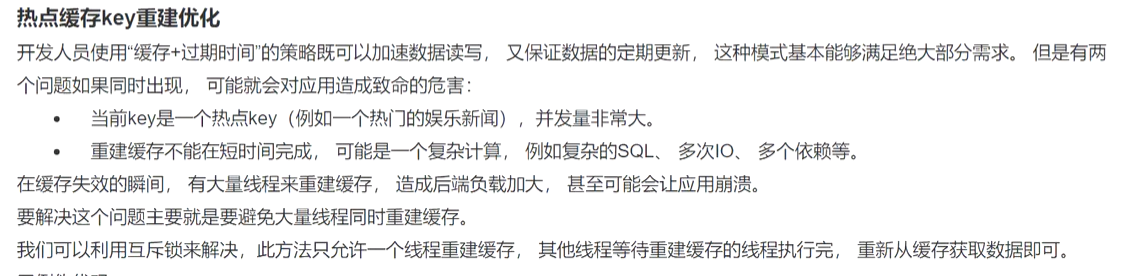
解决方法：

1. 给每个缓存设置随机过期时间(避免大量缓存在同一时间失效)
   1. 缓存雪崩：



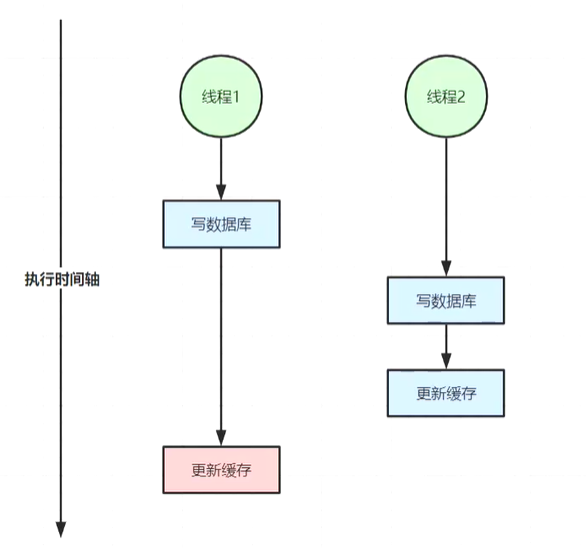
解决方法：

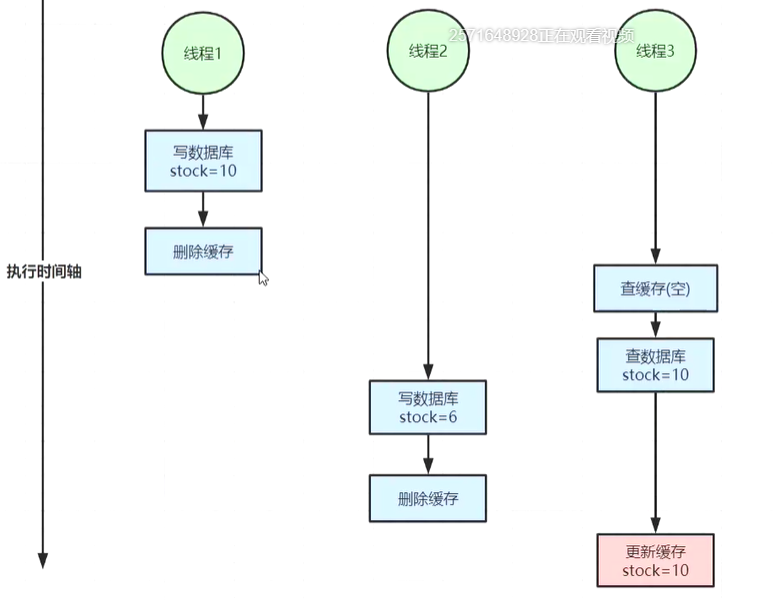
1. 保证缓存服务高可用(哨兵，集群)
2. 服务降级(限流)，返回空值
3. 提前演练
4. 热点缓存key重建优化

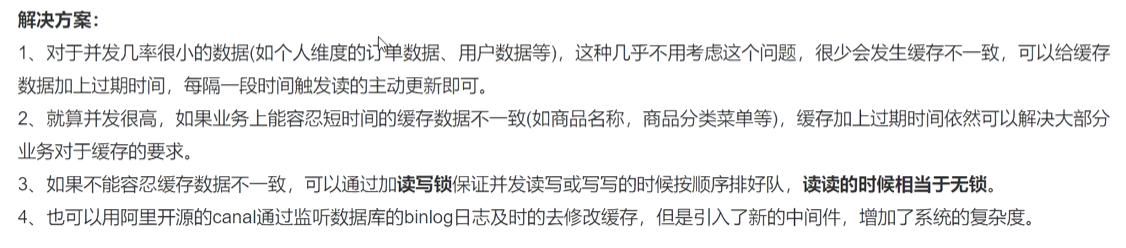


解决方法：

1. 使用互斥锁重建缓存热点key
2. 缓存与数据库双写不一致终极解决







1. 延迟双删
2. 加锁
3. Redis开发规范与性能优化

**Hash冲突：**

1. **开放定址法**
2. **链地址法**

# Zookeeper

## 核心概念：

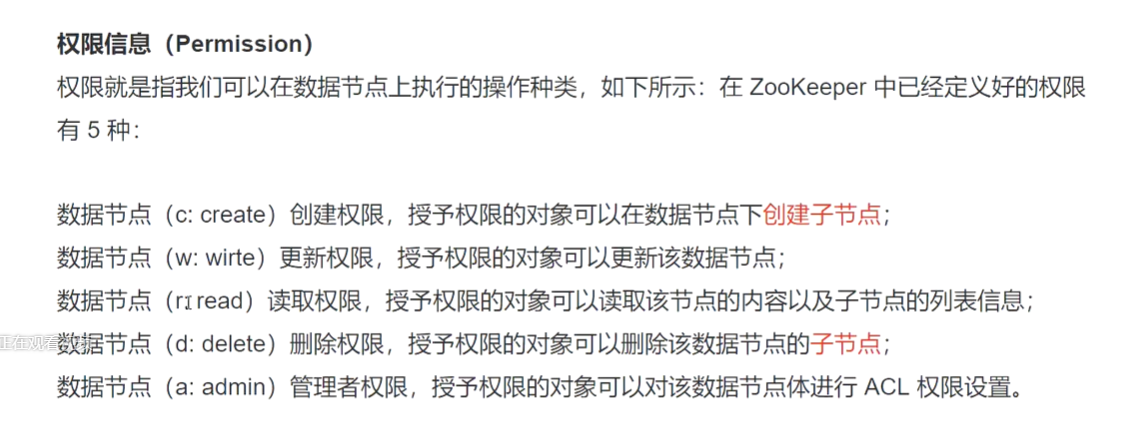
节点：

1. Master
2. Fllower
3. Observer

事件监听：

Watcher

## ACL权限控制：



## 内存数据和持久化：

3.5.0后支持不停机扩容

## Zookeeper客户端使用与经典应用场景

1. Zookeeper Java客户端ZClient
2. Apache Curator客户端
3. Zookeeper集群&不停机动态扩容/缩容
   * 1. 三种部署方式：
        1. 单机
        2. 集群
        3. 单机集群
     2. 3.5版本前不支持不停机扩容，3.5版本后支持不停机扩容
4. Zookeeper经典应用场景
   * 1. 注册中心，配置中心
     2. 添加watcher监听，配置改变通知给客户端
5. 分布式锁

加锁，释放锁

Zookeeper是一个典型的分布式数据一致性解决方案，分布式应用程序可以基于Zookeeper实现诸如数据发布/订阅，负载均衡，命名服务，分布式协调/通知，集群管理，Master选举，分布式锁和分布式队列等功能

Zookeeper 一个最常见的使用场景就是用于担任服务生产者和服务消费者的注册中心，服务生产者将自己提供的服务注册到Zookeeper注册中心，然后消费者从Zookeeper注册中心获取生产者提供的接口。

Zookeeper特点：

1. 顺序一致性：从同一客户端发起的事务请求，最终将会严格地按照顺序被应用到Zookeeper中
2. 原子性：所有事务请求的处理结果在整个集群中所有机器上的应用情况一致，也就是说，要么整个集群中所有的机器都成功应用了某一个事务，要么都没有应用
3. 可靠性：一旦一次更改请求被应用，更改的结果就会被持久化，知道被下一次更改覆盖

Zookeeper设计目标  
 1. 简单的数据模型：Zookeeper允许分布式进程通过共享的层次结构命名空间进行相互协调，文件结构为Znode Tree，数据保存在内存中，读多于写。

2. 可构建集群

# Dubbo

# 分布式

# MQ

# Spring

生命周期

