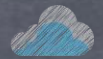







Redis专题分享

刘晓东

目录

Redis数据结构

RedisObject

```
typedef struct redisObject {
    unsigned type:4;
    unsigned encoding:4;
    unsigned lru:LRU_BITS; /* LRU time (relative to global lru_clock) or
                            * LFU data (least significant 8 bits frequency
                            * and most significant 16 bits access time). */
    int refcount;
    void *ptr;
} robj;
```

字符串

```
#set
set key value [EX s] [PX ms] [NX|XX]
setnx key value
setxx key value
mset k1 v1 k2 v2

#get
get key
mget k1 k2

#设置过期时间
expire key seconds

#自增
incr key
```

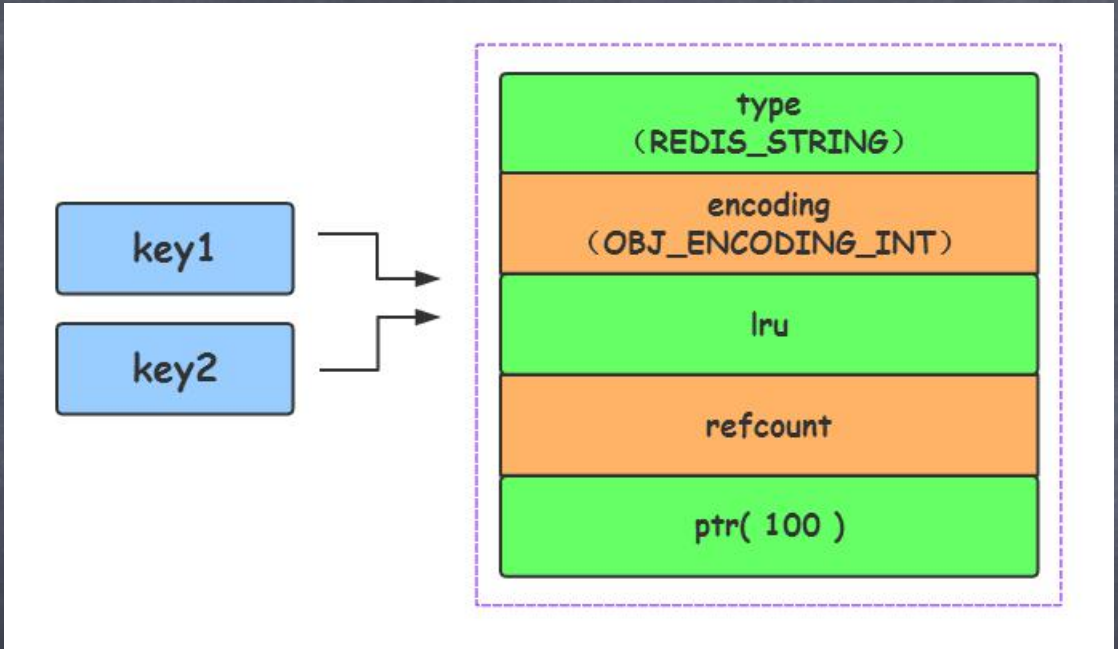
内部编码：

1、int：8个字节的长整型

2、embstr：小于等于44个字节的字符串

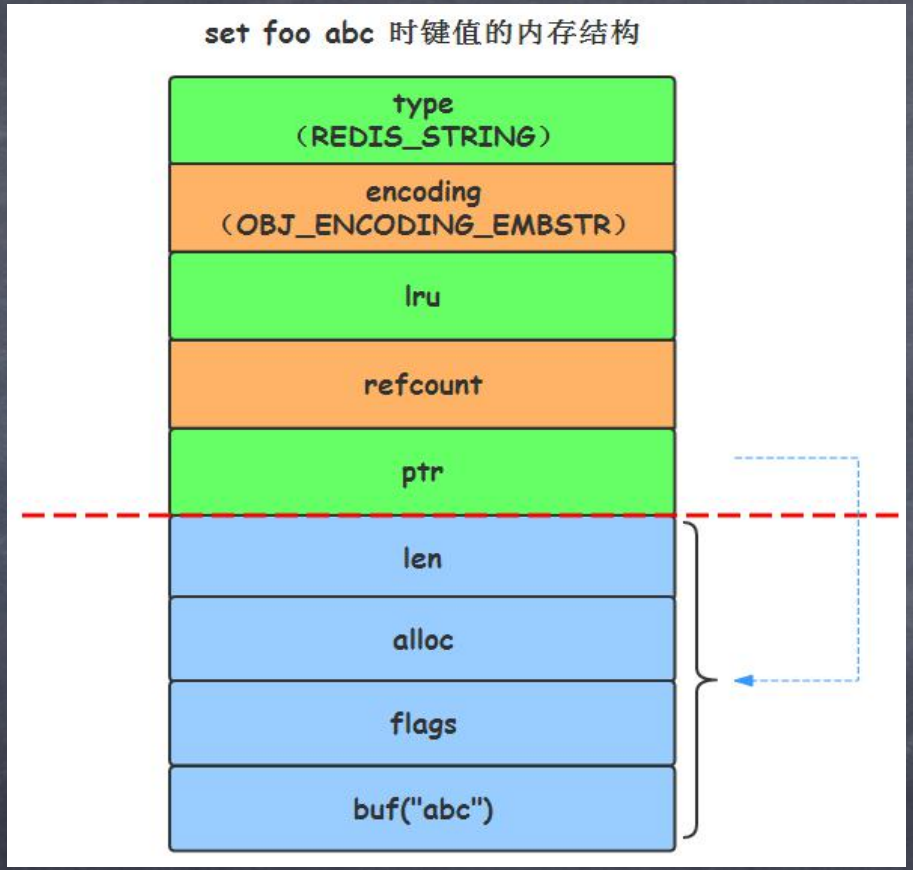
3、raw：大于44个字节的字符串

字符串


key1
key2
type
（REDIS_STRING）
encoding
（OBJ_ENCODING_INT）
lru
refcount
ptr(100)

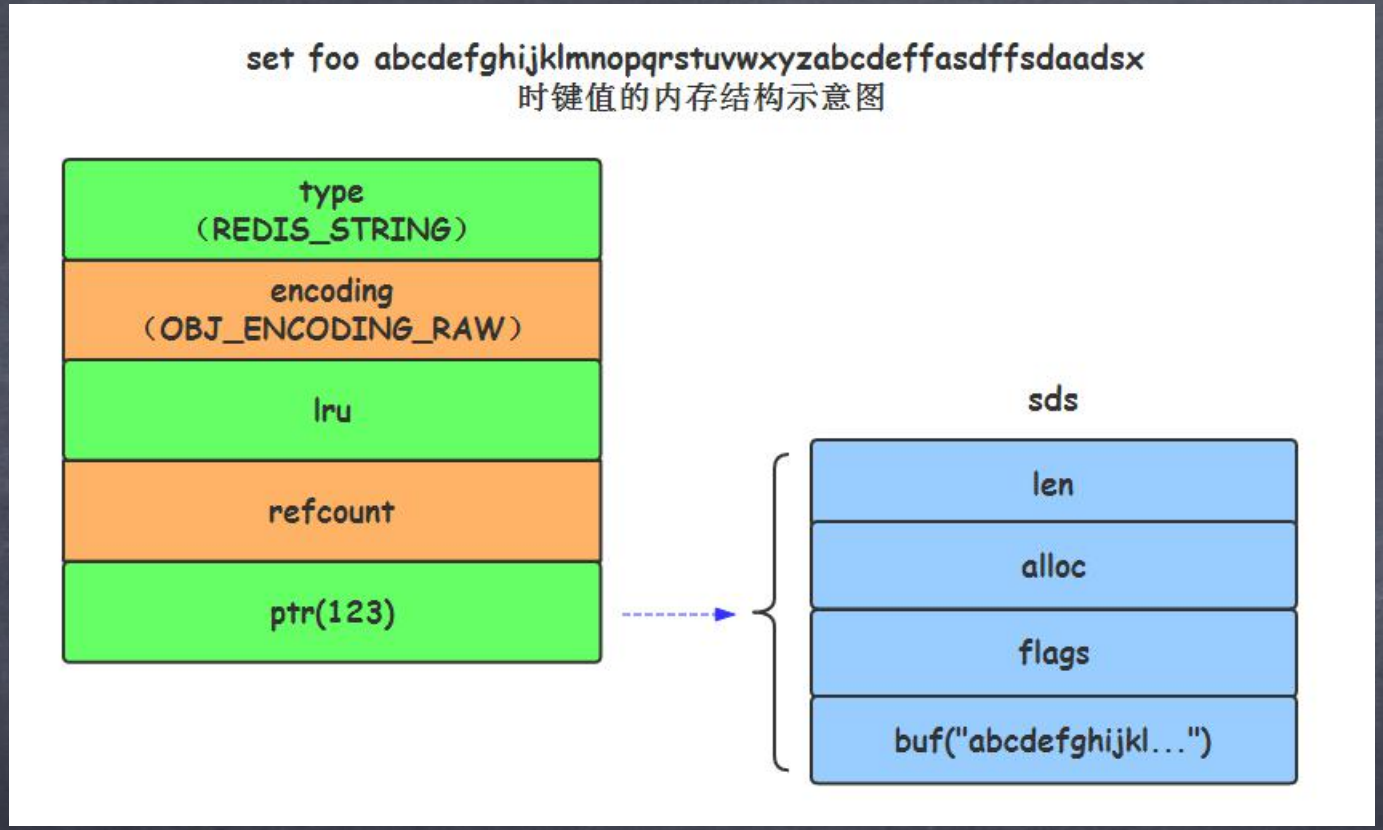
int型存储

字符串



embstr数据结构

字符串


set foo abcdefghijklmnopqrstuvwxyzabcdeffasdffsdaadsx
时键值的内存结构示意图
type
（REDIS_STRING）
encoding
（OBJ_ENCODING_RAW）
lru
refcount
ptr(123)
sds
len
alloc
flags
buf("abcdefghijkl...")

raw数据结构

哈希

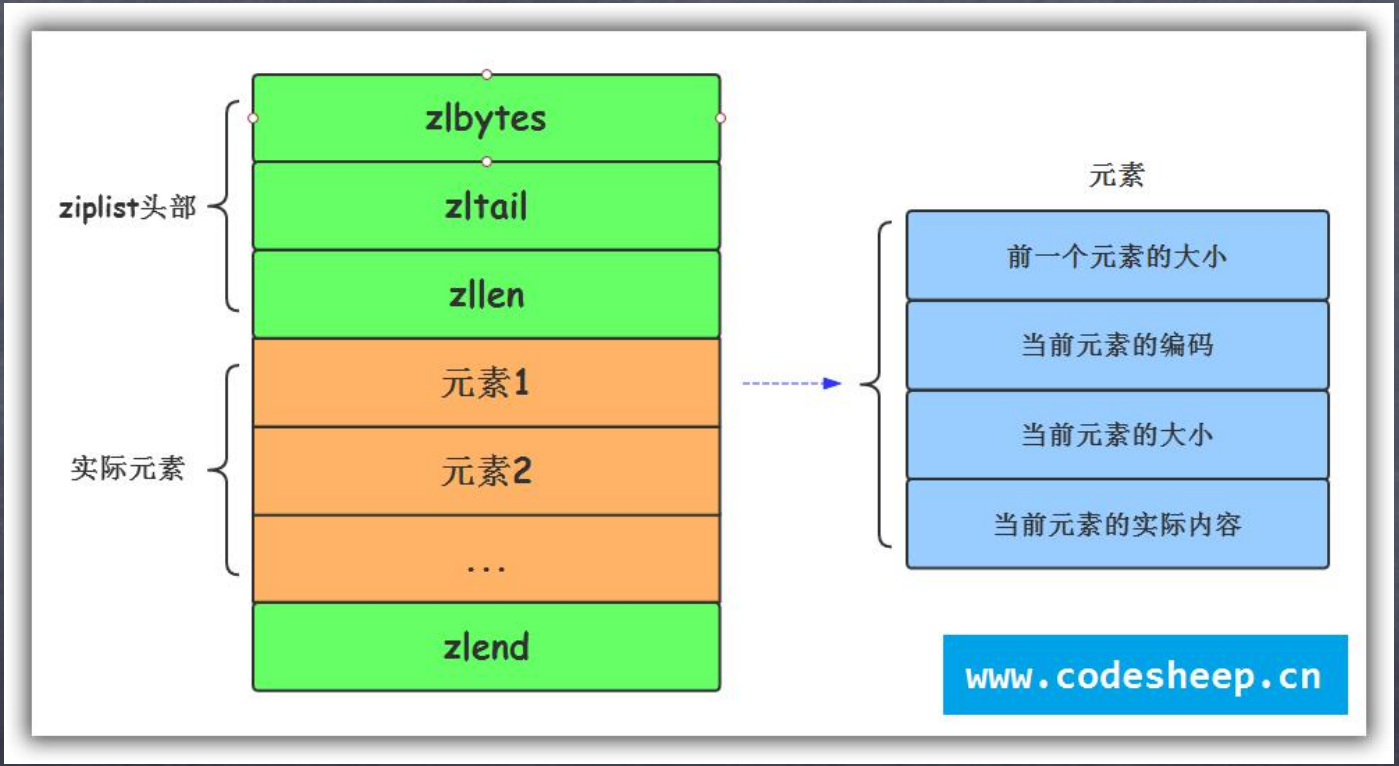
```
hset key field value
hget key field
hdel key field
hgetall key
hscan key cursor [MATCH pattern] [COUNT count]
```

内部编码：

1、ziplist：hash元素个数小于512，所有值长度小于64个字节（可以修改）

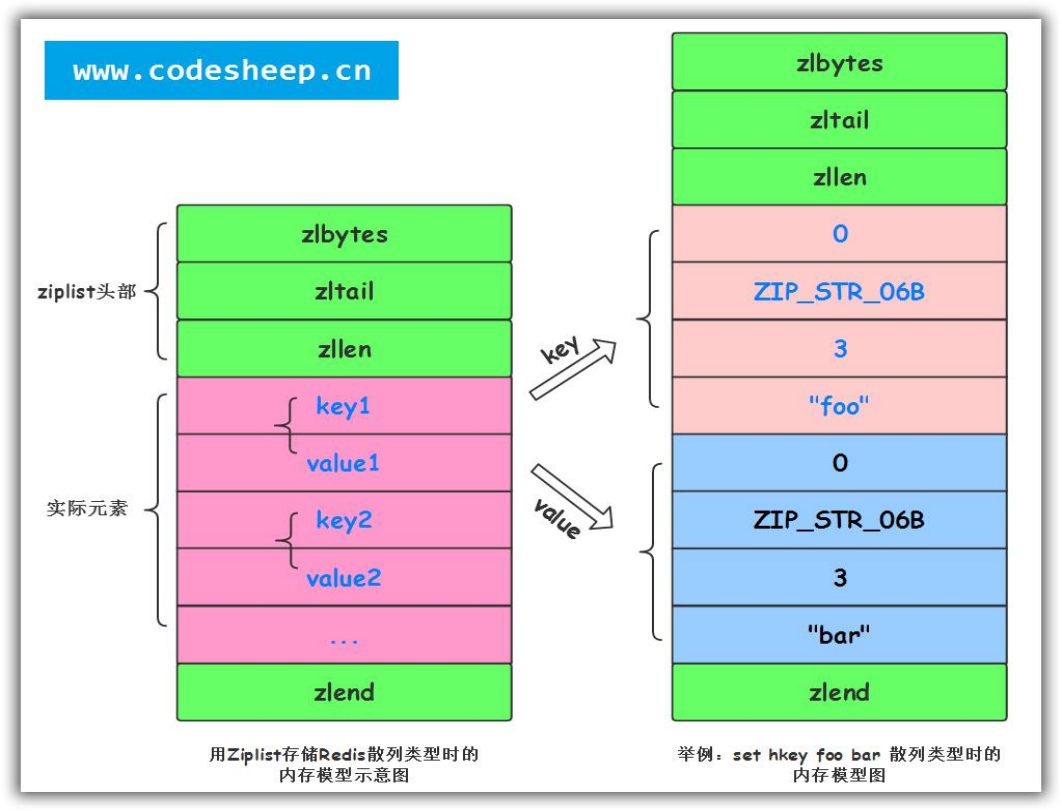
2、hashtable：当数据无法满足ziplist的要求时，采用hashtable

哈希



ziplist数据结构

哈希



Redis ziplist数据结构

列表

```
#插入
lpush key value
rpush key value

#查找
lrange key start end

#删除
lpop
rpop

#阻塞操作
blpop brpop
```

内部编码：

1、ziplist：元素个数小于512，所有值长度小于64个字节（可以修改）

2、linkedlist：当数据无法满足ziplist的要求时，采用linkedlist

集合

```
#添加元素
sadd key member
#删除元素
srem key member
#获取集合大小
scard key
#判断是否在集合中
sismember key member
#获取集合的所有元素
smembers keys
```

内部编码：

1、intset：元素个数小于512，所有值都是整数

2、hashtable：当数据无法满足ziplist的要求时，采用hashtable

有序集合

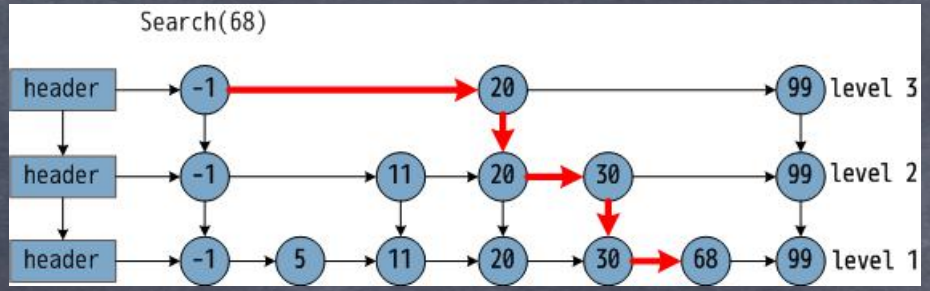
```
#添加元素
zadd key score member
#删除元素
zrem key member
#获取集合大小
zcard key
#获取元素
zrange key start end [withscores]
zrevrange key start end [withscores]
#获取集合的所有元素
zrange 0 -1
```

内部编码：

1、ziplist：元素个数小于128，所有值都是长度均小于64个字节（可配置）

2、skiplist：当数据无法满足ziplist的要求时，采用skiplist

有序集合


Search(68)
header
-1
20
99
level 3
header
-1
11
20
30
99
level 2
header
-1
5
11
20
30
68
99
level 1

跳跃表

Redis过期策略

Redis过期策略

1、惰性删除

当客户端读取带有超时属性的键时，如果已经超过键设置的过期时间则执行删除操作并返回空

2、定时任务删除

Redis内部维护有一个定时任务，每10秒随机选择检查20个键，如果超过25%的过期比例，则循环执行删除直到超时（慢模式25ms）。如果超时了，则进入快模式（超时时间1ms，2s内运行1次）

Redis驱逐策略

驱逐策略

- noevicition：直接返回错误
- allkeys-lru：所有key通用，优先删除最少使用的key
- volatile-lru：针对设置了expire属性的key
- allkeys-random：随机删除一部分key
- volatile-random：针对设置了expire的key
- volatile-ttl：针对设置了expire的key，优先删除一部分剩余时间短的key

Redis持久化

RDB持久化

RDB持久化是把当前进程数据生成快照以二进制文件保存在磁盘中。

1、save命令

阻塞进程，直到RDB文件创建完毕，中间不会接受其它命令，线上不建议使用

2、bgsave命令

fork子进程执行，不会阻塞主进程，仍然可以处理请求。

通过设置save m n 进行m秒内执行n次操作则执行bgsave

RDB优缺点

优点：

1、全量备份，适用于容灾

2、Redis加载RDB文件的速度远高于AOF

缺点：

1、每次都需要fork新的进程，代价比较高昂

2、没法做到实时持久化

AOF基本介绍

以追加日志的方式记录命令，每次重启的时候再执行一遍命令即可。主要用于解决数据持久化的实时性问题。文件格式为文本文件。

AOF过程

1、命令追加到内存缓存区

2、内存缓存区根据策略进行刷盘操作

3、定期重写AOF文件，减小体积

4、重启的时候进行reload加载到内存中

AOF刷盘策略

always：服务器每写入一个命令，就调用一次fdatasync，将缓冲区里面的命令写入到硬盘。

everysec（默认）：服务器每一秒调用一次fdatasync，将缓冲区里面的命令写入到硬盘。

no：服务器不主动调用fdatasync，由操作系统决定何时将缓冲区里面的命令写入到硬盘。

Redis主从复制

为什么要复制

1、故障恢复、容灾
2、读写分离

复制的拓扑

1、一主一从

一个主节点下面挂一个从节点

2、一主多从

一个主节点下面挂多个从节点

3、树状主从

主节点下面挂从节点，从节点下面可以继续挂从节点

复制过程

1、从节点执行slave of ip port

2、主从建立socket连接

3、发送ping命令

4、权限校验

5、同步数据集

6、持续复制

全量复制

用于首次建立主从连接或者主从之间数据差异过大。主节点生成RDB文件，并同步给从节点，从节点清除旧的数据并加载RDB

部分复制

用于首次建立主从连接后的命令复制

1、主节点维护自身复制偏移量和从节点偏移量。其中从节点偏移量是从节点每秒上报的

2、主节点也维护有一个复制缓冲区，每次命令发给从节点的同时，也会写入到复制缓冲区

3、从节点接收到主节点的命令后会更新自身的偏移量

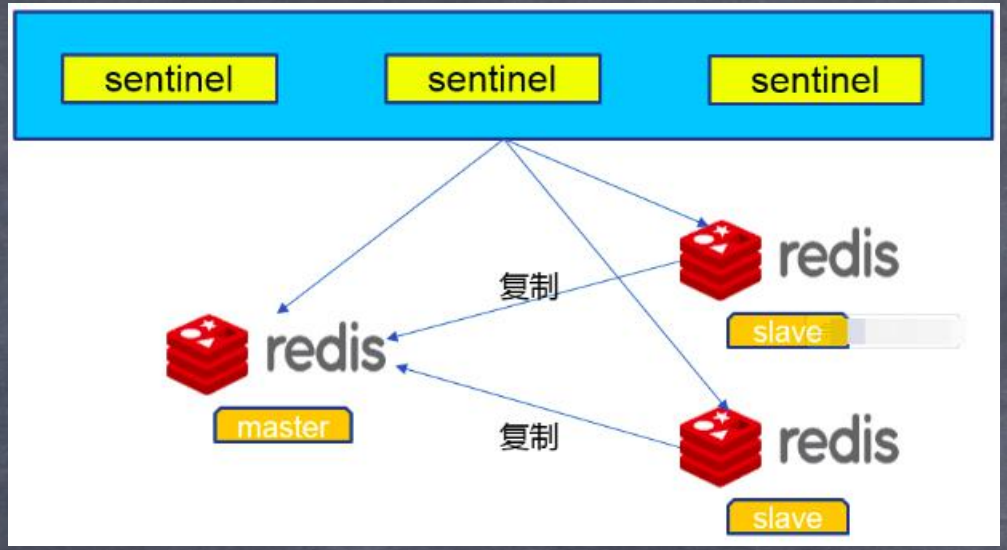
4、如果因为网络原因或者其他原因导致主从复制数据丢失，主节点可以根据主从复制量偏移从复制缓冲区把命令传给从节点。如果差异过大，则会进行全量复制

Redis Sentinel

背景

主从复制架构下，如果主节点发生故障，无法自动切换主从，需要人工干预。同时应用方也需要更新相应的IP地址，代价比较高昂。

原理



独立的哨兵节点定时检测主节点的状态，当主节点发生异常时，进行切主操作，并通知客户端更新主节点信息

搭建Sentinel集群

1、redis-server启动主节点

2、启动从节点，指定slaveof ip port

3、启动sentinel节点，和数据节点一样，只是需要调整一下配置文件

sentinel monitor mastername ip port n

sentinel down-after-milliseconds mymaster 5000

sentinel failover-timeout mymaster 60000

实现原理

1、每隔10s，每个sentinel节点向主节点和从节点发送info命令，通过解析返回的结果就可以得到主从节点的拓扑关系。

2、每隔2s，每个sentinel节点都会向_sentinel_:hello频道发送该节点对主节点的判断以及当前sentinel节点的信息。同时每个节点也会订阅该频道来更新自身的状态。

3、每隔1s，每个sentinel节点会向主节点、从节点以及其他sentinel节点发送ping命令来做心跳检查。

故障恢复

1、下线认定：当一个sentinel节点认为主节点故障时，会询问其他sentinel节点，当认为主节点故障的sentinel节点个数超过给定值时，会认为此主节点确实存在问题

2、领导者Sentinel节点选举：采用Raft算法选出领导者Sentinel

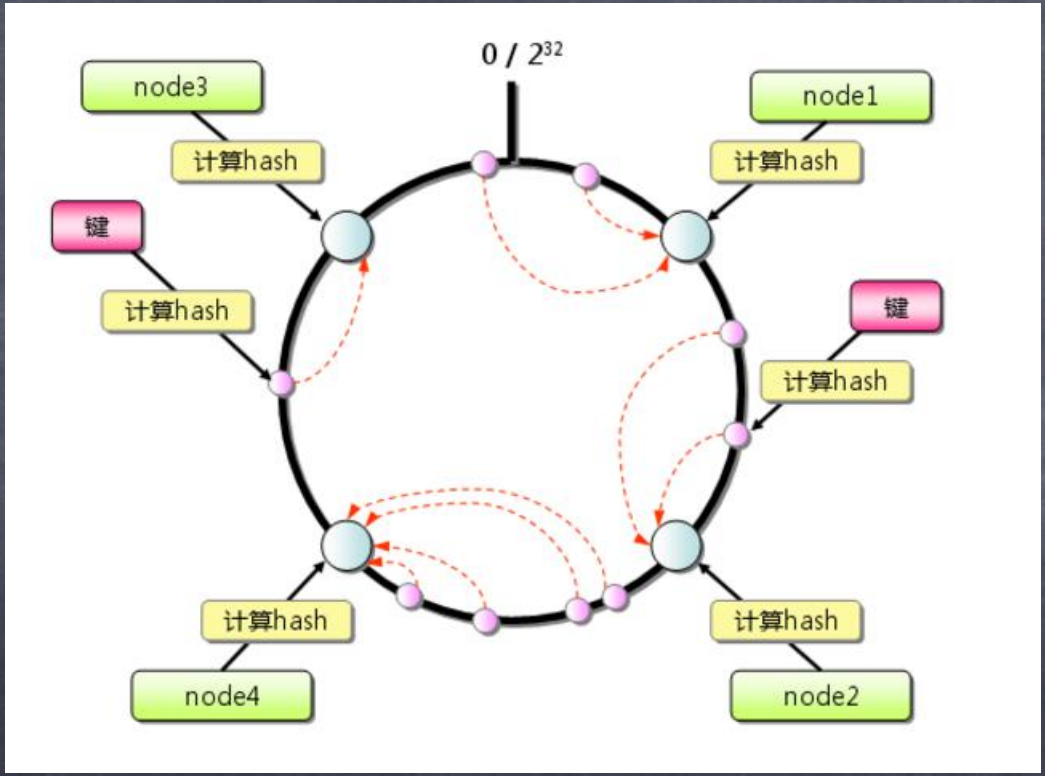
3、故障转移：根据一定规则选出一个从节点作为主节点，调整其它节点，让它们成为新的主节点的从节点

Redis Cluster

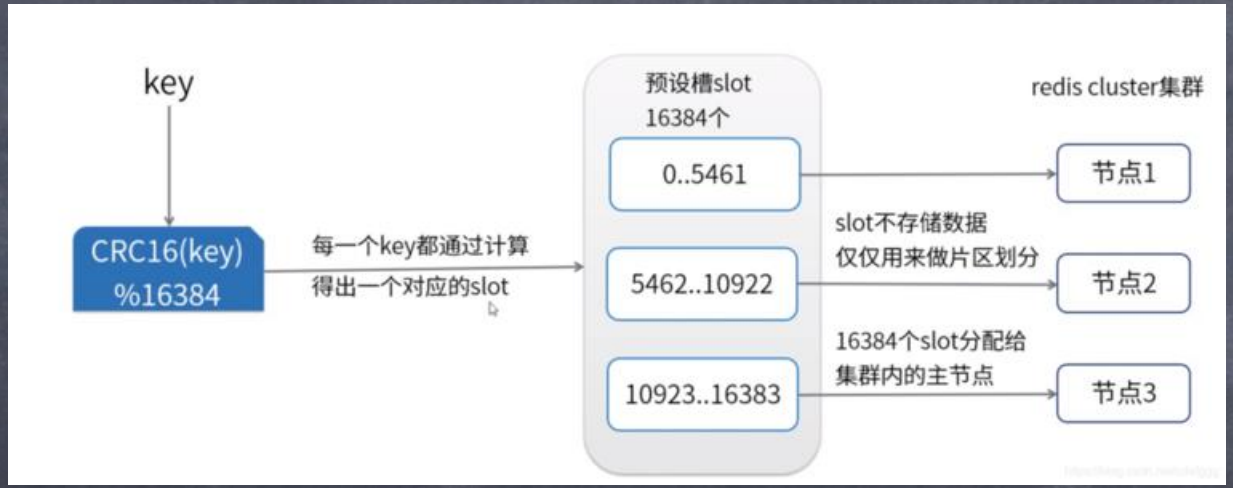
背景

RedisSentinel虽然解决了主从切换问题，保证了Redis的高可用，但无法解决单机在内存、流量等方面的瓶颈问题。Redis官方提供的RedisCluster方案很好的满足了应用分布式方面的需求。

一致性Hash


0 / 2^32
node3
计算hash
键
计算hash
node1
计算hash
键
计算hash
计算hash
node4
计算hash
node2

虚拟槽分区


key
CRC16(key)
%16384
每一个key都通过计算
得出一个对应的slot
预设槽slot
16384个
0..5461
5462..10922
10923..16383
redis cluster集群
节点1
节点2
节点3
slot不存储数据
仅仅用来做片区划分
16384个slot分配给
集群内的主节点

搭建集群-启动服务

```
# redis-7000.conf
port 7000
cluster-enabled yes
cluster-config-file nodes_7000.conf
cluster-node-timeout 5000
daemonize yes

#开启7个服务
redis-server redis-7000.conf
redis-server redis-7001.conf
redis-server redis-7002.conf
redis-server redis-7003.conf
redis-server redis-7004.conf
redis-server redis-7005.conf
```

搭建集群-节点握手

```
cluster meet 127.0.0.1 7001
cluster meet 127.0.0.1 7002
cluster meet 127.0.0.1 7003
cluster meet 127.0.0.1 7004
cluster meet 127.0.0.1 7005
```

```
127.0.0.1:7000> cluster nodes
9293b75943bbaf14f8b17e04b77d8ed2f83a9421 127.0.0.1:7001@17001 master - 0 1566636454808 1 connected
b2d4ffdef12a4c9308b51a3736d58e5d588a7590 127.0.0.1:7000@17000 myself,master - 0 1566636455000 3 connected
19c74a5edcb458f9b42f562cf7e4e78dfb28d9fc 127.0.0.1:7005@17005 master - 0 1566636455815 5 connected
7c4f96782a4848aef5730d3f105570f6275f4d63 127.0.0.1:7004@17004 master - 0 1566636454000 4 connected
2bbf070dd2214d47db5d9f33bcd7c4f907bcbd5d 127.0.0.1:7002@17002 master - 0 1566636455000 2 connected
bb3cd801cedd4560f086a75c6b11f6d0be6c7e60 127.0.0.1:7003@17003 master - 0 1566636456823 0 connected
```

搭建集群-分配槽

```
cluster replicate b2d4ffdef12a4c9308b51a3736d58e5d588a7590
cluster replicate 9293b75943bbaf14f8b17e04b77d8ed2f83a9421
cluster replicate 2bbf070dd2214d47db5d9f33bcd7c4f907bcbd5d
```

```
127.0.0.1:7005> cluster nodes
b2d4ffdef12a4c9308b51a3736d58e5d588a7590 127.0.0.1:7000@17000 master - 0 1566636647262 3 connected 0-5461
19c74a5edcb458f9b42f562cf7e4e78dfb28d9fc 127.0.0.1:7005@17005 myself,slave 2bbf070dd2214d47db5d9f33bcd7c4f907bcbd5d 0 1566636647000 5 connected
7c4f96782a4848aef5730d3f105570f6275f4d63 127.0.0.1:7004@17004 slave 9293b75943bbaf14f8b17e04b77d8ed2f83a9421 0 1566636646116 4 connected
9293b75943bbaf14f8b17e04b77d8ed2f83a9421 127.0.0.1:7001@17001 master - 0 1566636646000 1 connected 5462-10922
bb3cd801cedd4560f086a75c6b11f6d0be6c7e60 127.0.0.1:7003@17003 slave b2d4ffdef12a4c9308b51a3736d58e5d588a7590 0 1566636644000 3 connected
2bbf070dd2214d47db5d9f33bcd7c4f907bcbd5d 127.0.0.1:7002@17002 master - 0 1566636648270 2 connected 10923-16383
```

搭建集群

```
./redis-trib.rb  create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003
127.0.0.1:7004 127.0.0.1:7005
./redis-trib.rb check 127.0.0.1:7000
```

```
app@lxd-PC:/usr/local/redis-4.0.2/src$ ./redis-trib.rb check 127.0.0.1:7000
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: b2d4ffdef12a4c9308b51a3736d58e5d588a7590 127.0.0.1:7000
   slots:0-5461 (5462 slots) master
   1 additional replica(s)
M: 9293b75943bbaf14f8b17e04b77d8ed2f83a9421 127.0.0.1:7001
   slots:5462-10922 (5461 slots) master
   1 additional replica(s)
S: 19c74a5edcb458f9b42f562cf7e4e78dfb28d9fc 127.0.0.1:7005
   slots: (0 slots) slave
   replicates 2bbf070dd2214d47db5d9f33bcd7c4f907bcbd5d
S: 7c4f96782a4848aef5730d3f105570f6275f4d63 127.0.0.1:7004
   slots: (0 slots) slave
   replicates 9293b75943bbaf14f8b17e04b77d8ed2f83a9421
M: 2bbf070dd2214d47db5d9f33bcd7c4f907bcbd5d 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
S: bb3cd801cedd4560f086a75c6b11f6d0be6c7e60 127.0.0.1:7003
   slots: (0 slots) slave
   replicates b2d4ffdef12a4c9308b51a3736d58e5d588a7590
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

节点通信-Gossip协议

1、每个节点均开辟单独的TCP通道用于节点间通信
2、每个节点固定周期内（1秒10次）基于特定规则（最久没有通信）发送ping消息
3、接受到ping消息的节点发送pong消息作为响应

集群扩容

1、准备新节点，加入集群中
cluster meet ip port

2、迁移槽和数据

故障转移

1、故障节点认证

2、从节点上升为主节点

Redis高级命令

高级命令

1、info：获取Redis信息，包含CPU、内存、持久化以及集群等信息

2、slowlog get :获取慢请求执行日志

3、redis-cli --bigkeys: 查找大key

4、monitor：集群环境下慎用，会导致很大的内存缓冲区占用

缓存设计

缓存设计

1、Redis数据结构的选择和Key的设计问题

2、缓存穿透问题

3、缓存雪崩问题

4、Redis慢请求排查

5、读写分离

性能问题排查

性能排查方向

1、同一节点其它实例的影响

2、持久化的影响

3、Key设计的设计问题

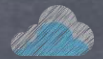
4、使用方式不合理

5、内存使用情况

6、命令执行情况








感谢您的观看