RedisVIP讲义-第五天

一、今日教学目标

1. Redis数据丢失场景分析
2. Redis缓存跟DB一致性问题分析
3. Redis缓存雪崩、穿透、击穿问题分析
4. Redis大Key调优、冷热数据。

三、教学过程

Redis数据丢失场景

到了今天，我们Redis相关重要的知识点都已经学得差不多了，那么这节课，更多的是总结性以及思维性的提升！

持久化丢失

采用RDB或者不持久化， 会有数据丢失，因为是手动或者配置以快照的形式来进行备份。

解决：启用AOF，以命令追加的形式进行备份，但是默认也会有1s丢失，这是在性能与数据安全性中寻求的一个最适合的方案，如果为了保证数据一致性，可以将配置更改为always，但是性能很慢，一般不用。

```
# appendfsync always
```

主从切换

因为Redis的数据是主异步同步给从的，提升了性能，但是由于是异步同步到从。所以存在数据丢失的可能。

1.master写入数据k1 ,由于 是异步同步到slave，当master没有同步给slave的时候，master挂了

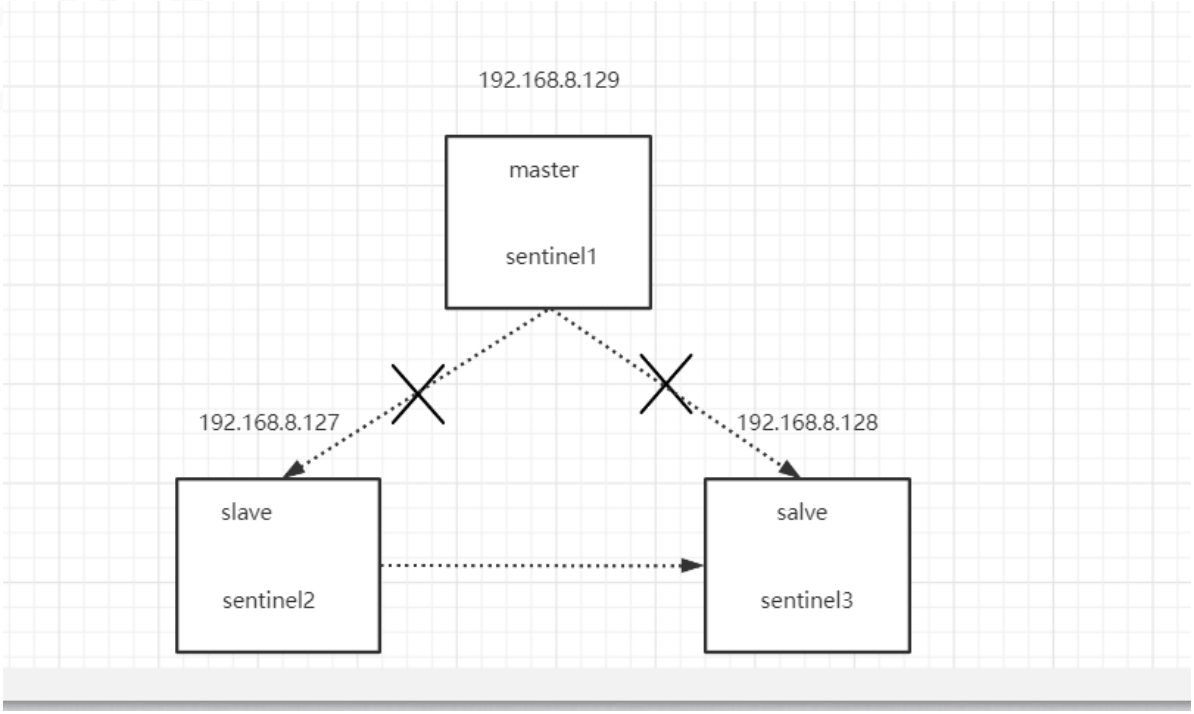
2.slave会成为新的master，并且没有同步k1，

3.master重启，会成为新master的slave，同步数据会清空自己的数据，从新的master加载

4.k1丢失

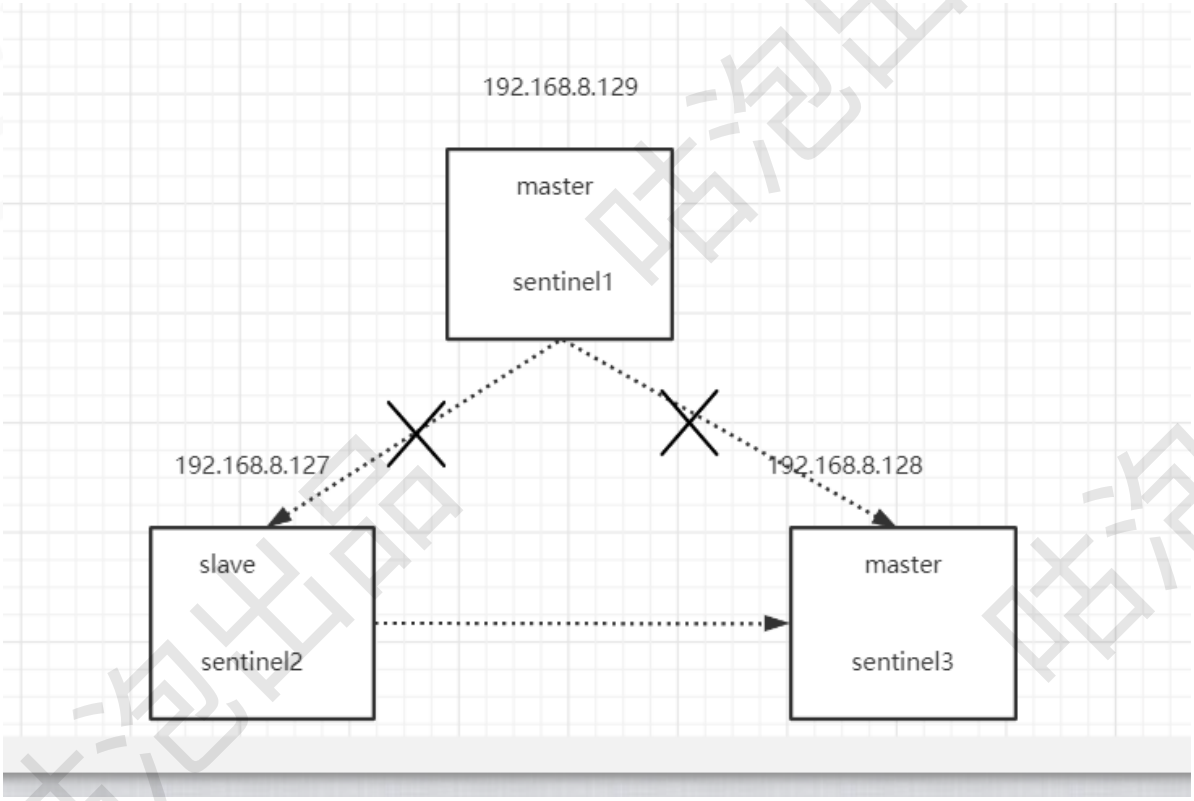
sentinel脑裂

1.假如有个sentinel集群与Redis集群，如图



2.当我的master机器（129）跟另外2台发生分区容错，网络断开

3.sentinel2跟sentinel3大于三分之二，满足故障转移条件（sentinel故障转移需要过半），这个时候会从2个slave中选出一个master。



4.客户端假如连接到sentinel1，我们数据会写入129，连接到sentinel2或者sentinel3，数据就会写入128，同时有2个master写入数据

假如129 写了k1

5.当网络恢复后，129会变成128的从，129的数据全部从128同步

6.k1丢失。

尽量减少主从切换跟sentinel数据丢失的解决办法：

```
min-replicas-to-write 1          至少有1个从节点同步到我主节点的数据，但是由于是异步同步，所以是最终一致性  不会确保有数据写入
min-replicas-max-lag 10          判断上面1个的延迟时间必须小于等于10s
```

Redis缓存跟DB一致性问题

什么是缓存数据一致性问题

什么是数据一致性，在并发环境下，可能会导致redis跟DB的数据产生不一致。

怎么产生

查询缓存逻辑

1.我们查询DB之前，先去查询Redis，如果Redis存在，直接返回；如果Redis不存在，从DB查询。

2.从DB查询后，回写到Redis

缓存依赖

1.当修改DB的数据，DB数据修改成功后，删除Redis数据，做好缓存依赖。

丢失场景：

1.线程A请求缓存，没有缓存，从DB拿到1。

2.线程B将1更新为2，并且删除缓存，DB的值为2

3.线程A更新缓存，redis为1

我们发现，redis数据为1 ，db数据为2，出现了数据一致性问题。

所以，数据一致性产生的根本问题，是查询DB 跟操作Redis不是原子性的，所以并发会导致数据一致性问题。

怎么解决

不可取的强一致性方案

延时双删

所谓延时双删，就是在更新DB后，等待一段时间，再进行Redis删除！来等待其他的线程拿到的都是最新数据！

也会产生很多问题。

1.延时多久？不知道其他线程要多久。

2.不够优雅，代码中写入延时代码。

采用锁机制，不让有并发

在更新的时候，采取锁的机制，不让其他线程进行删除操作！

但是会拖慢整个性能，违背了Redis的初衷

所以，我们只能采用最终一致性，不应该去保证强一致性

最终一致性方案

每个缓存设置过期时间

设置过期时间，就算数据不一致，也只是在有效时间内的不一致。

Mysql canal等数据同步工具

捕捉到DB的更改，同步到相关Redis，相对比较复杂，要知道每个数据对应的缓存。

Redis缓存雪崩、穿透、击穿问题分析

缓存雪崩

缓存雪崩就是Redis的大量热点数据同时过期（失效），因为设置了相同的过期时间，刚好这个时候Redis请求的并发量又很大，就会导致所有的请求落到数据库。

1.保证Redis的高可用，防止由于Redis不可用导致全部打到DB

2. 加互斥锁或者使用队列，针对同一个key只允许一个线程到数据库查询

3.缓存定时预先更新，避免同时失效

4.通过加随机数，使key在不同的时间过期

缓存穿透

缓存穿透是指缓存和数据库中都没有的数据，但是用户一直请求不存在的数据！这时的用户很可能就是攻击者，恶意搞你们公司的，攻击会导致数据库压力过大。最好的办法：封IP

还有，我们的Redis跟Mysql有自己的一些高可用的方案，没有那么脆弱，比如Redis的集群、Mysql的集群。在你封IP之前Redis肯定不会挂

那么假如，面试官硬要你给一个解决的方法：那就是我们的布隆过滤器

布隆过滤器的思想：既然是因为你Redis跟DB都没有，然后用户恶意一直访问不存在的key，然后全部打到Redis跟DB。

那么我们能不能单独把DB的数据单独存到另外一个地方，但是这个地方只存一个简单的标记，标记这个key存不存在，如果存在，就去访问Redis跟DB,否则直接返回。并且这个内存最好很小

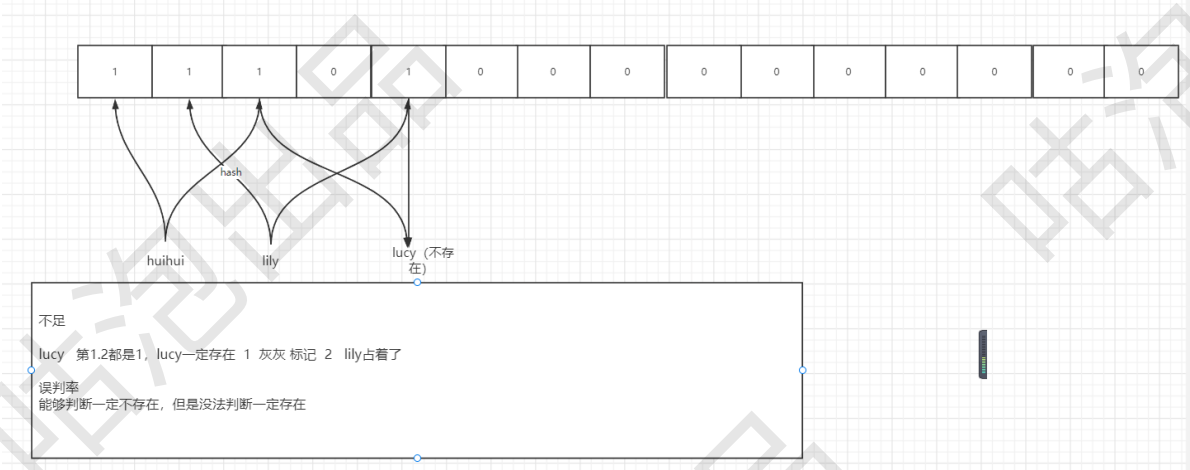
比如，你去按摩，你有指定的技师，那么你先看下这个技师会不会值班，如果值班，你就去，否则你就不去。并且这个值班表必须提前安排好！可以用一个在线表格，在就打钩，不能说你把这个人站在这里，看到这个人就说明在。

那么这个就是我们的布隆过滤器的思想。

布隆过滤器

实现原理

怎么存储这个value是不是存在的标记？我们在第一天讲过一个bitMap的数据类型。他能存储某个key是否存在，并且非常节省内存。



布隆过滤器，位图里面只有0跟1，然后将访问的值通过多个不同的hash算法得到不同的值，放置在不同的位图标记为1

有误判，没法判断数据一定存在，但是能判断一定不存在！

怎么减少误判？

其实就是减少hash冲突：

1.多次hash。

2.增大位图大小。

弊端：必须初始化数据到布隆过滤器，并且不能删除（因为一个位可能有其他的元素占有，如果把位图设为0，那么可能db有的数据你就查不到）

实现方式：

1Redission封装　基于Redis的bitMap实现 分布式

2.google.guava　　本地内存

3.Redis　bloom模板 https://redis.io/docs/stack/bloom/

缓存击穿

单个key过期的时候有大量并发，使用互斥锁，回写redis，并且采用双重检查锁来提升性能！减少对DB的访问。

以上，我们Mysql现在有集群 等方案，所以也没那么脆弱，如果真的到了瓶颈我们也可以进行DB横向扩容。

慢查询分析

许多存储系统都会有慢日志查询，提供给开发跟运维来找到哪些指令是比较耗时的。比如Mysql。那么Redis中也会有慢日志查询。

但是Redis的慢查时间只会去统计执行指令的时间，不会统计网络消耗时间。所以没有慢查不代表没有超时。

多慢才是慢查询?

```
# The following time is expressed in microseconds, so
1000000 is equivalent    微秒表示
# to one second. Note that a negative number disables the
slow log, while   负数禁用慢日志记录
# a value of zero forces the logging of every command.   为
0记录每个命令
slowlog-log-slower-than 10000      //默认10ms   建议1ms
```

最多存储多少慢查?

```
slowlog-max-len 128   //最多存储128条数据
```

查看慢查

```
127.0.0.1:6379> slowlog get  10          10为要查询的数量
1) 1) (integer) 2   //慢查询标志id
   2) (integer) 1659806161  //发生的时间戳
   3) (integer) 11025   //耗时 11.025ms
   4) 1) "EVAL"    //执行指令和参数
      2) "local size = redis.call('hget', KEYS[1],
'size');local hashIterations = redis.call('hget', KEYS[1],
'hashIterations');assert(siz... (83 more bytes)"
      3) "1"
      4) "{product:bloom}:config"
      5) "1459688"
      6) "5"
   5) "192.168.8.23:59236"
   6) ""
2) 1) (integer) 1
   2) (integer) 1659806161
   3) (integer) 11019
   4) 1) "hget"
      2) "{product:bloom}:config"
      3) "hashIterations"
   5) "?:0"
   6) ""
3) 1) (integer) 0
   2) (integer) 1659806161
   3) (integer) 10977
   4) 1) "hget"
      2) "{product:bloom}:config"
      3) "size"
   5) "?:0"
   6) ""
```

也可以对慢查进行清理

```
127.0.0.1:6379> slowlog reset
OK
```

如果发现慢查：

1.尽量不要使用hgetall keys 等指令

2.调整大对象，变成多个子对象（一般超过10K就算比较大的key，但是根据业务来）

```
[root@localhost src]# ./redis-cli --bigkeys   //可以找到大对象
```

阻塞分析

1.业务记录好相关日志，以及降级报警等系统，知道有阻塞

2.原因主要分为几个点

外部原因： 网络阻塞 CPU竞争等

内部原因：数据结构不合理导致大key等单条指令耗时过大、for子进程阻塞、AOF刷盘阻塞