讲师介绍

Hash　　QQ：805921455

从事Java软件研发十年。

前新浪支付核心成员、

咪咕视讯(中国移动)项目经理、

对分布式架构、高性能编程有深入的研究。

明天，你一定会感谢今天奋力拼搏的你

掌握Redis哨兵机制，从容应对生产服务器的突发故障

分布式高并发—缓存技术

目录

课程安排

01

哨兵使用架构

基于哨兵的Redis高可用架构

02

哨兵如何监控redis

03

哨兵如何高可用

04

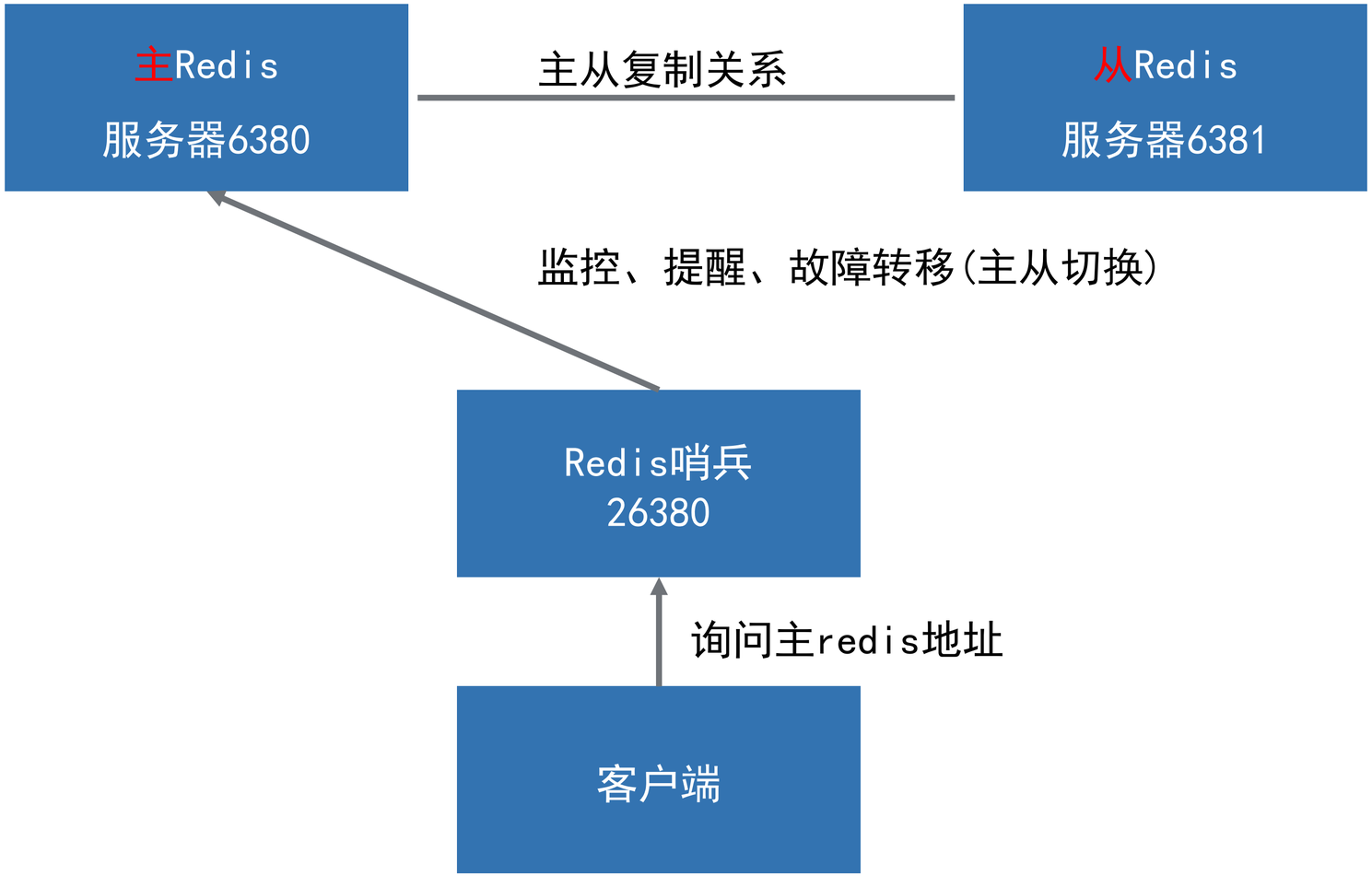
哨兵内部通讯原理

通讯原理剖析

目录

哨兵使用架构

哨兵(Sentinel)机制核心作用



核心运作流程

服务发现和健康检查流程

搭建redis主从集群

⇩

启动哨兵(客户端通过哨兵发现Redis实例信息)

⇩

哨兵通过连接master发现主从集群内的所有实例信息

⇩

哨兵监控redis实例的健康状况

故障切换流程

哨兵一旦发现master不能正常提供服务，则通知给其他哨兵

⇩

当一定数量的哨兵都认为master挂了

⇩

选举一个哨兵作为故障转移的执行者

⇩

执行者在slave中选取一个作为新的master

⇩

将其他slave重新设定为新master的从属

目录

哨兵如何监控redis

7大核心概念

1. 哨兵如何知道Redis主从信息（自动发现机制）
2. 什么是master主观下线
3. 什么是客观下线
4. 哨兵之间如何通信（哨兵之间的自动发现）
5. 哪个哨兵负责故障转移？（哨兵领导选举机制）
6. slave选举机制
7. 最终主从切换的过程

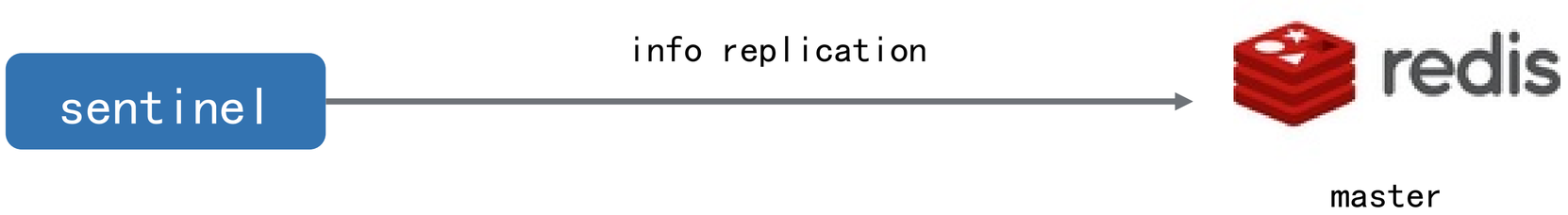
哨兵启动和配置

启动命令： redis-server /path/to/sentinel.conf --sentinel

配置文件启动时指定，运行过程中会自动变更，记录哨兵的监测结果

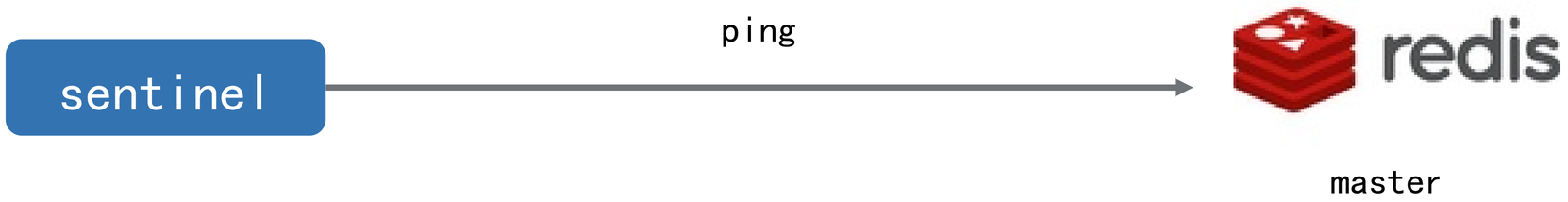
```
# 配置文件在sentinel运行期间是会被动态修改的
# sentinel如果重启时，就可以根据这个配置来恢复其之前所监控的redis集群的状态
# 绑定IP
bind 0.0.0.0
# 默认yes，没指定密码或者指定IP的情况下，外网无法访问
protected-mode no
# 哨兵的端口，客户端通过这个端口来发现redis
port 26380
# 哨兵自己的IP，手动设定也可自动发现，用于与其他哨兵通信
sentinel announce-ip
# 临时文件夹
dir /tmp
# sentinel监控的master的名字叫做mymaster,地址为 60.205.209.106 6380,两个及以上哨兵认定为死亡，才认为是真的死亡
sentinel monitor mymaster 60.205.209.106 6380 2
# 发送心跳PING来确认master是否存活
# 如果master在“一定时间范围”内不回应ping 或者是回复了一个错误消息，那么这个sentinel会主观地(单方面地)认为这个master已经不可用了
sentinel down-after-milliseconds mymaster 1000
# 如果在该时间（ms）内未能完成failover操作，则认为该failover失败
sentinel failover-timeout mymaster 3000
# 指定了在执行故障转移时，最多可以有多少个从Redis实例在同步新的主实例，在从Redis实例较多的情况下这个数字越小，同步的时间越长，完成故障转移所需的时间就越长
sentinel parallel-syncs mymaster 1
```

哨兵如何知道Redis主从信息



哨兵配置文件中，保存着主从集群中master的信息，可以通过info命令，进行主从信息自动发现。

什么是主观下线(sdown)

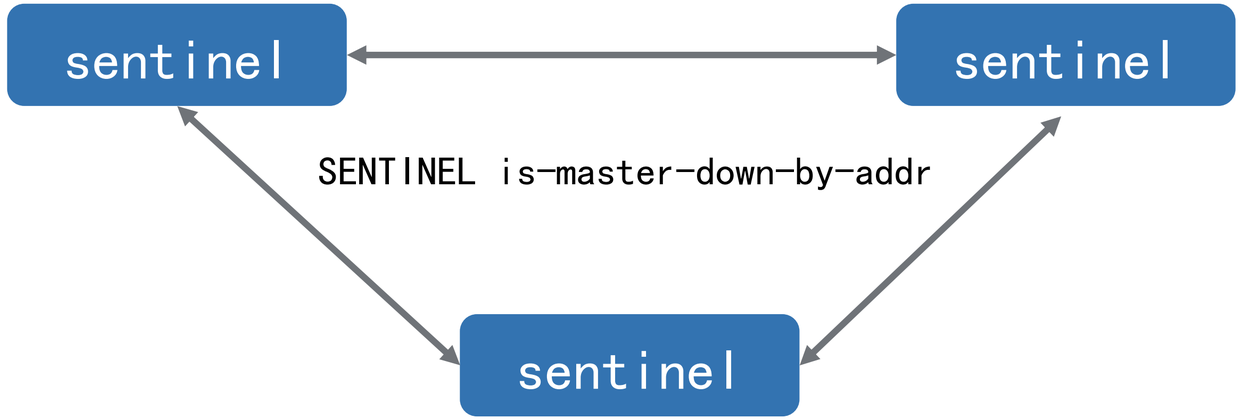


主观下线：单个哨兵自身认为redis实例已经不能提供服务

检测机制：哨兵向redis发送ping请求，+PONG、-LOADING、 -MASTERDOWN这三种情况视为正常，其他回复均视为无效。

对应配置文件的配置项：sentinel down-after-milliseconds mymaster 1000

什么是客观下线(odown)



客观下线：一定数量值的哨兵认为master已经下线。

检测机制：当哨兵主观认为master下线后，则会通过 SENTINEL is-master-down-by-addr命令 询问其他哨兵是否认为master已经下线，如果达成共识（达到quorum个数），就会认为master节点客观下线，开始故障转移流程。

对应配置文件的配置项：sentinel monitor mymaster 60.205.209.106 6380 2

slave选举方案

从服务器实例中，按右侧顺序依次筛选

slave 节点状态，非 S_DOWN,O_DOWN，DISCONNECTED

判断规则：(down-after-milliseconds * 10) + milliseconds_since_master_is_in_SDOWN_state

SENTINEL slaves mymaster

优先级

redis.conf中的一个配置项： slave-priority 值越小，优先级越高

数据同步情况

Replication offset processed

最小的run id

run id 比较方案： 字典顺序， ASCII码

最终主从切换的过程

- 针对即将成为master的slave节点，将其撤出主从集群

自动执行：slaveof NO ONE

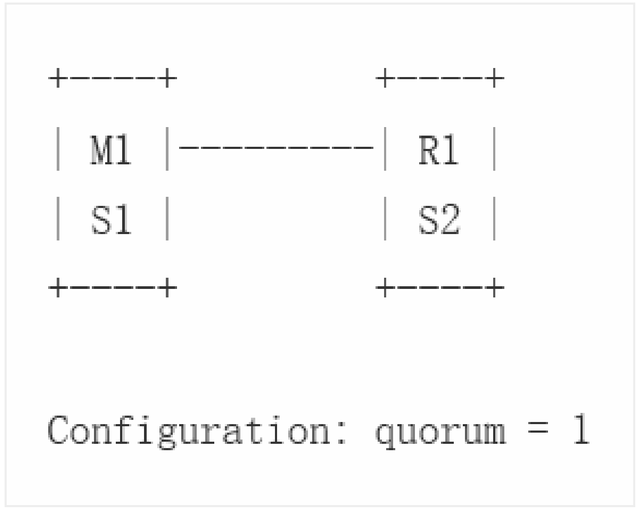
- 针对其他slave节点，使它们成为新master的从属

自动执行：slaveof new_master_host new_master_port

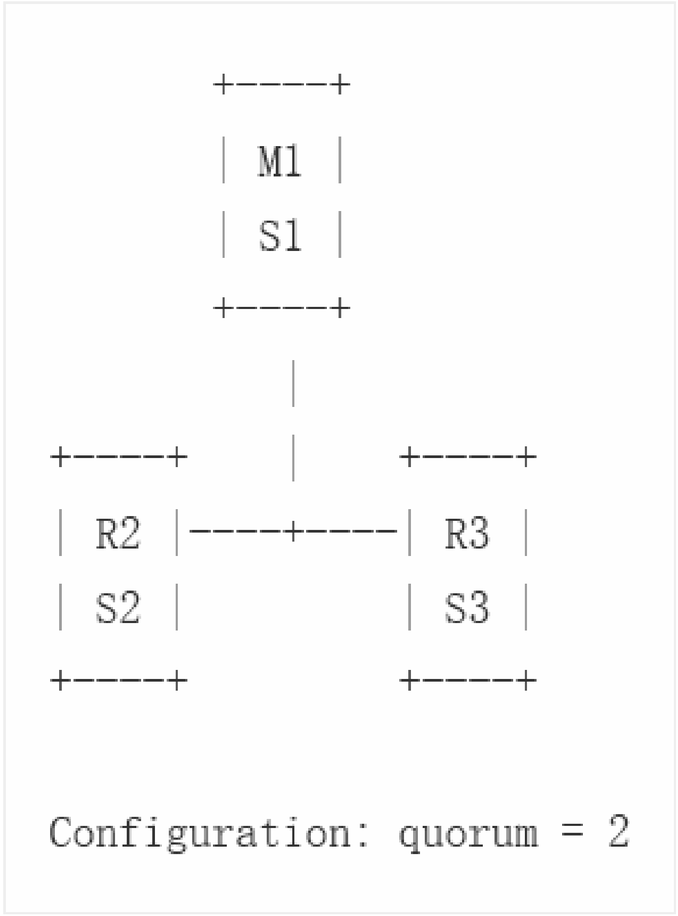
目录

哨兵如何高可用

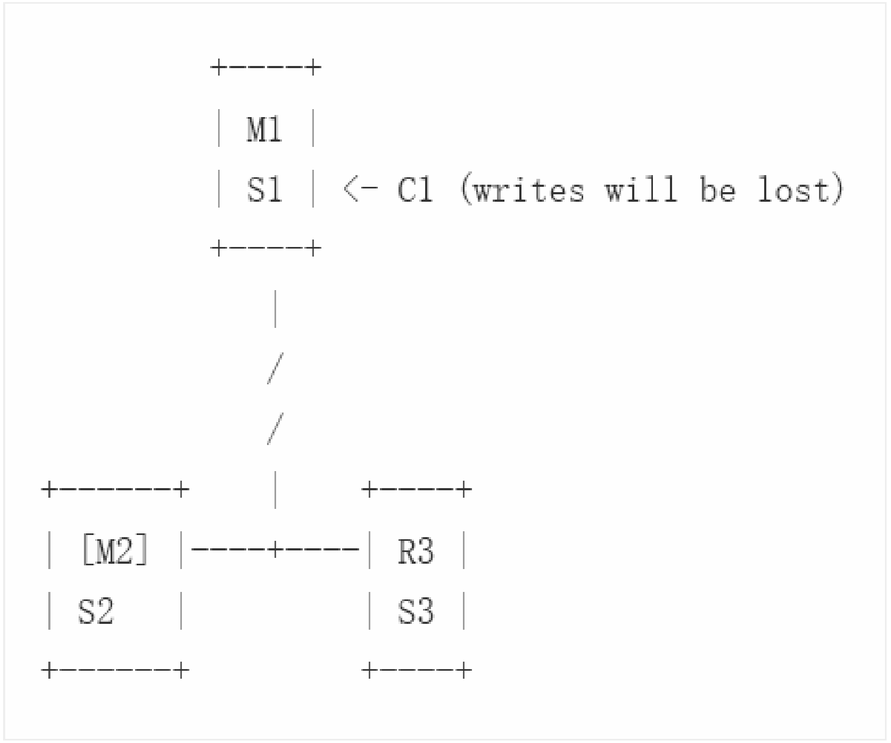
哨兵服务部署方案



两个哨兵，不建议



1主2从，三个哨兵



1主2从，网络分区下

可能出现数据不一致或丢失。

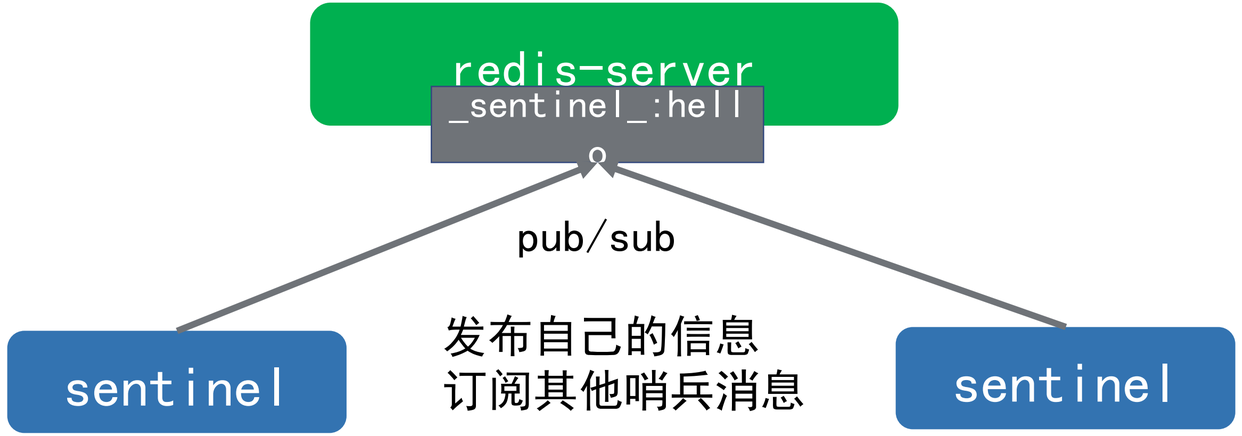
Redis集群非强一致

目录

哨兵内部通讯原理

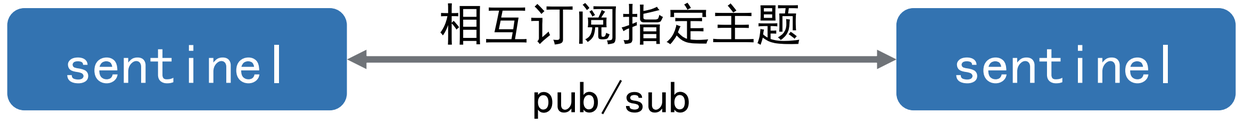
哨兵之间如何通信

1、 哨兵之间的自动发现



2、 哨兵之间通过命令进行通信

3、 哨兵之间通过订阅发布进行通信



哨兵领导选举机制

基于Raft算法实现的选举机制，流程简述如下：

1. 拉票阶段：每个哨兵节点希望自己成为领导者；

2. sentinel节点收到拉票命令后，如果没有收到或同意过其他sentinel节点的请求，就同意该sentinel节点的请求（每个sentinel只持有一个同意票数）；

3. 如果sentinel节点发现自己的票数已经超过一半的数值，那么它将成为领导者，去执行故障转移；

4. 投票结束后，如果超过failover-timeout的时间内，没进行实际的故障转移操作，则重新拉票选举。

注： 以了解raft协议为主。https://raft.github.io/、http://thesecretlivesofdata.com/raft/

谢谢观看