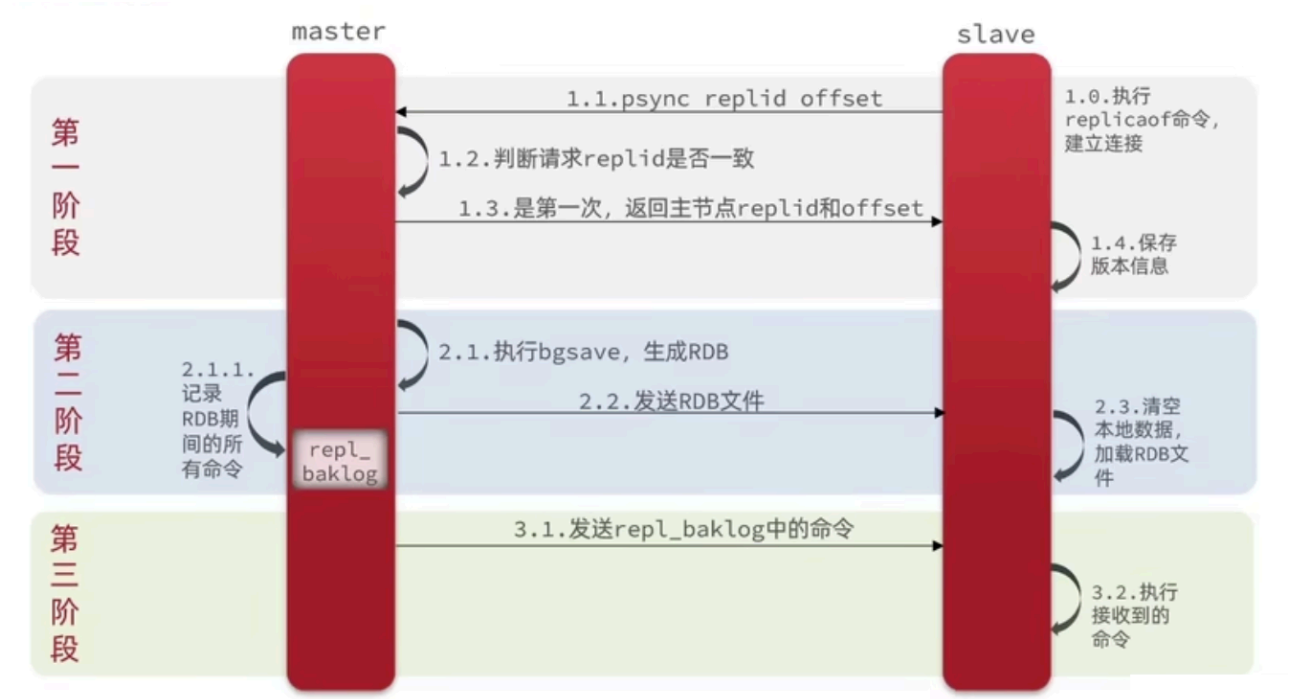
Redis运维

Redis主从数据同步

主从第一次同步是全量同步



master如何判断slave是不是第一次来同步数据：

Replication Id：简称replid，是数据集的标记(姓氏)，id一致则说明是同一数据集。每一个master都有唯一的replid，slave则会继承master节点的replid

Offset：偏移量，随着记录在repl_baklog中的数据增多而逐渐增大。slave完成同步时也会记录当前同步的offset。

如果slave的offset小于master的offset，说明slave数据落后于master，需要更新。

因此slave做数据同步，必须向master声明自己的replication id 和offset，master才可以判断到底需要同步哪些数据。

如果slave重启后同步，会进行增量同步

repl_baklog大小有上限，写满后会覆盖最早的数据。如果slave断开时间过久，导致数据被覆盖，则无法实现增量同步，只能再次全量同步。

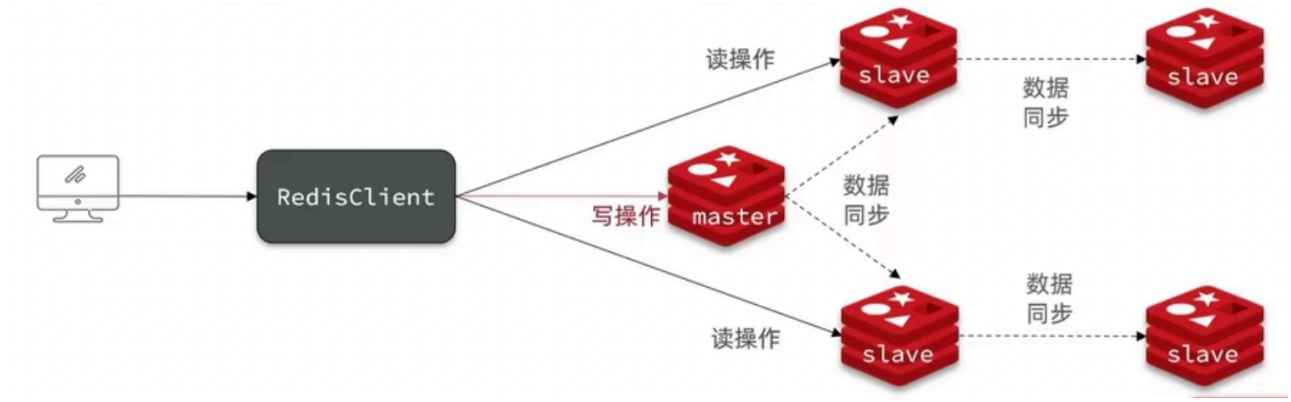
可以从以下几个方面来优化Redis主从集群：

在master中配置repl-diskless-sync yes启用无磁盘复制，避免全量同步时的磁盘IO。

Redis单节点上的内存占用不要太大，减少RDB导致的过多磁盘IO

适当提高repl_baklog的大小，发现slave宕机时尽快实现故障恢复，尽可能避免全量同步

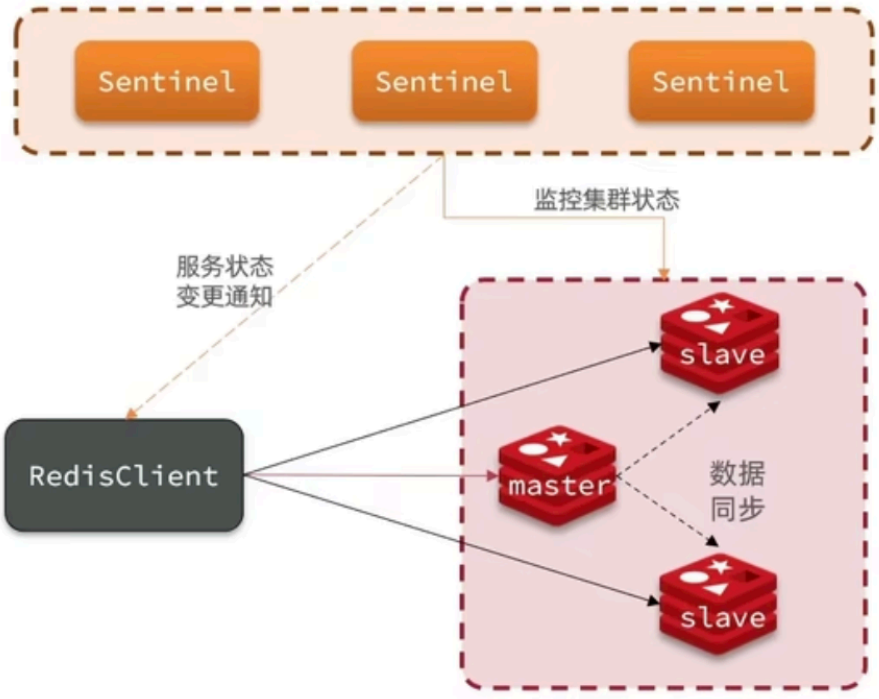
一个master上的slave节点数量，如果实在是太多slave，则可以采用主-从-从链式结构，减少master压力



slave节点宕机恢复后可以找master节点同步数据，那master节点宕机怎么办?

哨兵

Redis提供了哨兵(Sentinel）机制来实现主从集群的自动故障恢复。哨兵的结构和作用如下：



- 监控：Sentinel 会不断检查master和slave是否按预期工作
- 自动故障恢复：如果master故障，Sentinel会将一个slave提升为master。当故障实例恢复后也以新的master为主
- 通知：Sentinel充当Redis客户端的服务发现来源，当集群发生故障转移时，会将最新信息推送给Redis的客户端

Sentinel基于心跳机制监测服务状态，每隔1秒向集群的每个实例发送ping命令：

- 主观下线：如果某sentinel节点发现某实例未在规定时间响应，则认为该实例主观下线。

- 客观下线：若超过指定数量（quorum）的sentinel都认为该实例主观下线，则该实例客观下线。quorum值最好超过Sentinel实例数量的一半。

一旦发现master故障，sentinel需要在salve中选择一个作为新的master，选择依据是这样的：

- 首先会判断slave节点与master节点断开时间长短，如果超过指定值（down-after-milliseconds * 10）则会排除该slave节点
- 然后判断slave节点的slave-priority值，越小优先级越高，如果是0则永不参与选举
- 如果slave-prority一样，则判断slave节点的offset值，越大说明数据越新，优先级越高
- 最后判断slave节点的运行id大小，越小优先级越高

当选中了其中一个slave为新的master后（例如slave1），故障的转移的步骤如下：

- sentinel给备选的slave1节点发送slaveof no one命令，让该节点成为master
- sentinel给所有其它slave发送slaveof ip port命令，让这些slave成为新master的从节点，开始从新的master上同步数据
- Sentinel将故障节点标记为slave（修改配置），当故障节点恢复后会自动成为新的master的slave节点

```
@Bean
public LettuceClientConfigurationBuilderCustomizer clientConfigurationBuilderCustomizer(){
    return clientConfigurationBuilder -> clientConfigurationBuilder.readFrom(ReadFrom.REPLICA_PREFERRED);
}
```

这里的ReadFrom是配置Redis的读取策略，是一个枚举，包括下面选择：

MASTER： 从主节点读取

MASTER_PREFERRED：优先从master节点读取，master不可用才读取replica

REPLICA: 从slave (replica) 节点读取

REPLICA_PREFERRED：优先从slave (replica) 节点读取，所有的slave都不可用才读取master

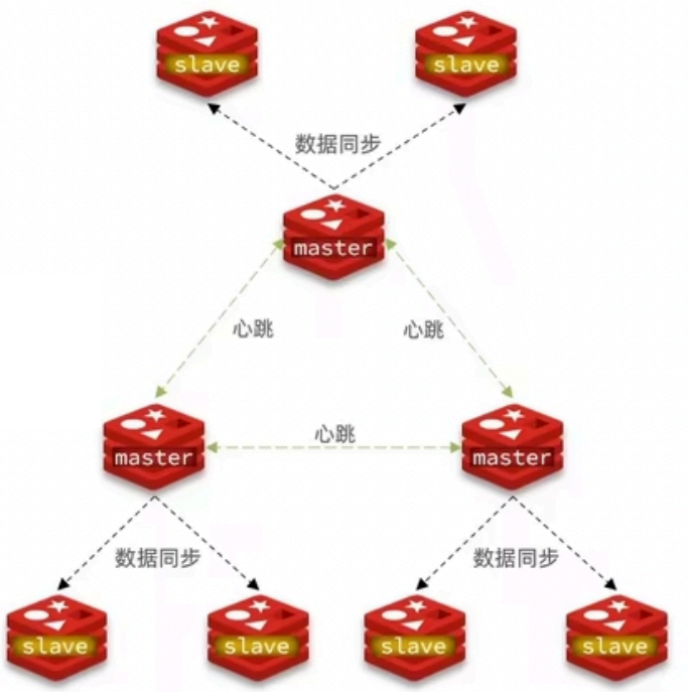
分片集群

主从和哨兵可以解决高可用、高并发读的问题。但是依然有两个问题没有解决：

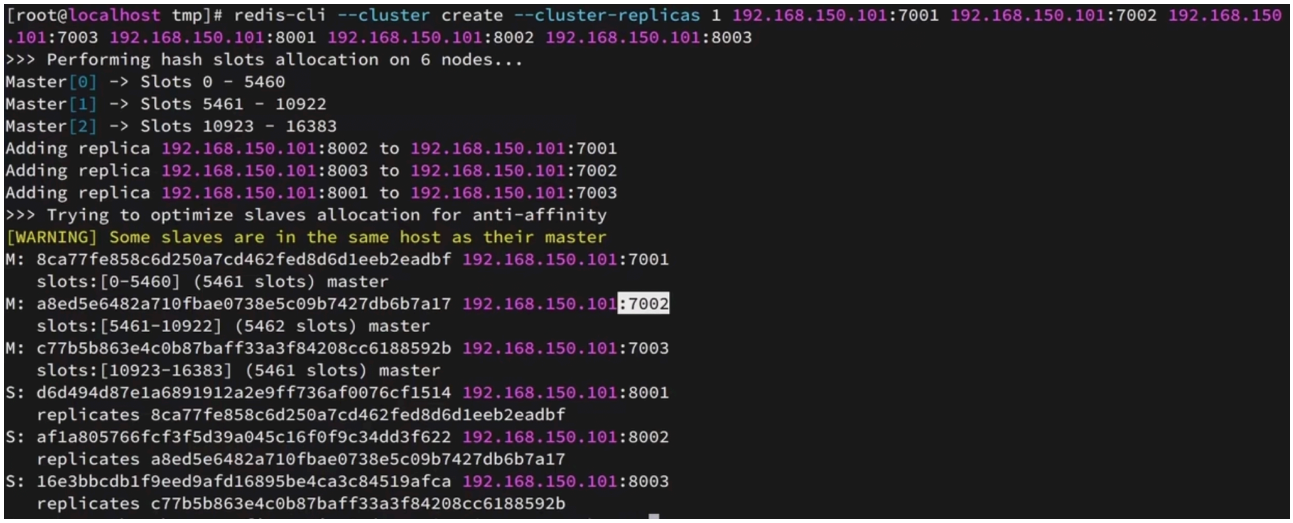
- 海量数据存储问题
- 高并发写的问题

使用分片集群可以解决上述问题，分片集群特征：

- 集群中有多个master，每个master保存不同数据
- 每个master都可以有多个slave节点
- master之间通过ping监测彼此健康状态
- 客户端请求可以访问集群任意节点，最终都会被转发到正确节点



Redis会把每一个master节点映射到0~16383共16384个插槽（hash slot）上，查看集群信息时就能看到：

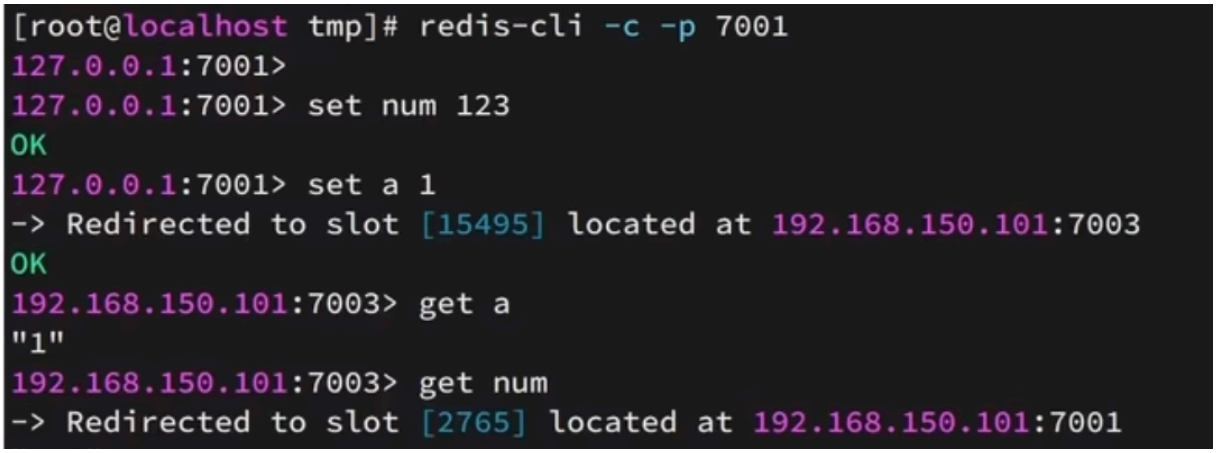


数据key不是与节点绑定，而是与插槽绑定。redis会根据key的有效部分计算插槽值，分两种情况：

Key中包含{}，且{}中至少包含1个字符，{}中的部分是有效部分

key中不包含{}，整个key都是有效部分

例如：key是num，那么就根据num计算，如果是{a}num，则根据a计算。计算方式是利用CRC16算法得到一个hash值，然后对16384取余，得到的结果就是slot值。



cluster failover命令可以手动让集群中的某个master宕机，切换到执行cluster failover命令的slave节点，实现无感知的数据迁移。手动的failover支持三种不同模式：

- 缺省：默认的流程
- force：省略了对offset的一致性校验
- takeover：直接执行第5步，忽略数据一致性、忽略master状态和其它master的意见