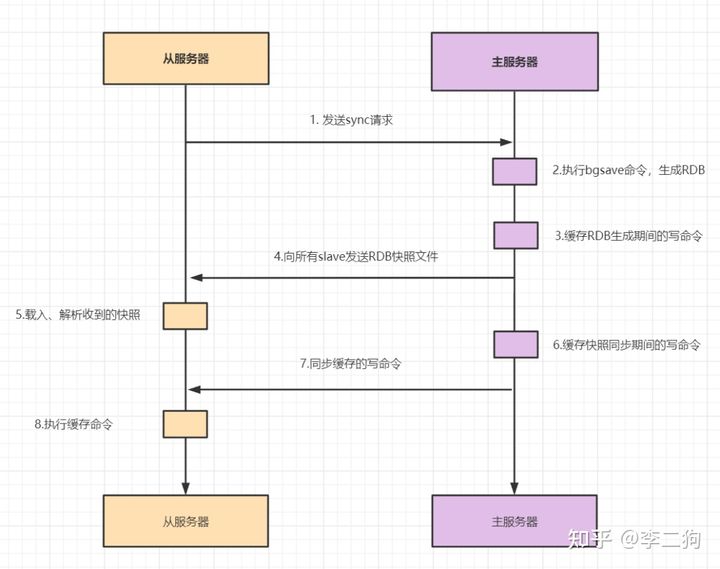
主从模式：master-slave

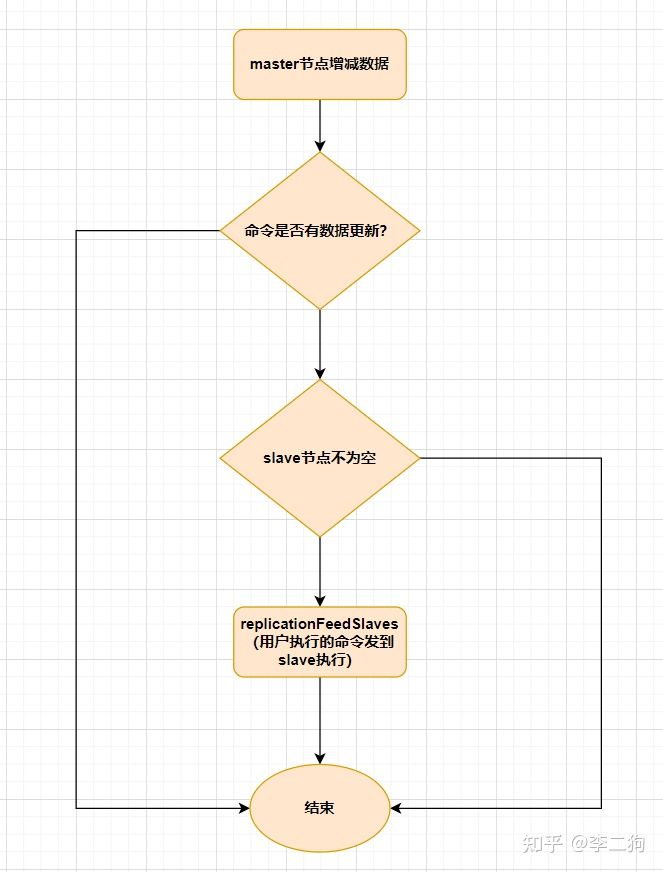
主结点负责读写，从结点只负责读。实现主从模式的需要主从复制。主从结点之间，从结点与主结点是socket的长链接。

主从复制分为全量复制和增量复制：

全量复制：slave第一次启动连接master，或者认为(例如网络中断过长)是第一次连接。图中的sync应该是psync。

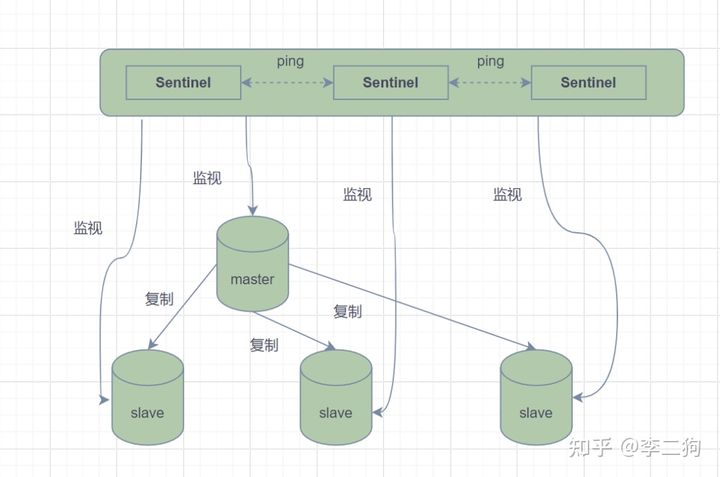


增量复制：全量同步后，如果master上的数据再次发生变化，就会进行增量复制。



增量复制的核心思想就是把涉及到数据变化的命令都发送到所有的slave结点上，让它们执行。

哨兵模式：主从模式为了保证高可用，当master挂掉时，需要从slave结点推一台出来升级为master，这就涉及到如何判断master和slave状态是否正常，就用到了哨兵结点(Sentinel)。通常情况下，不会使用一个Sentinel(否则就又会出现单点问题)，而是多个哨兵。哨兵之间也要相互监视。



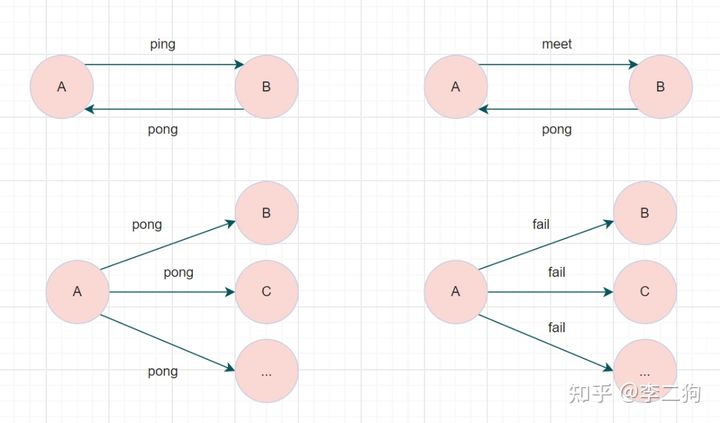
哨兵的作用：

1. 发送命令给master和slave，返回它们的运行状态
2. 监控到主结点宕机，就会把从结点升级为主结点，并通过发布/订阅模式通知其它从结点修改配置文件，切换主机
3. 哨兵之间还会相互监控

哨兵判断master是否宕机，是通过哨兵之间的投票进行判断的。投票的结果由一个哨兵发起，进行升级操作。

集群模式：

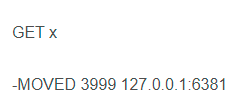
Redis集群中的结点是通过Gossip协议通信，常用的Gossip消息为：ping，pong，meet，fail。每个节点是通过集群总线(cluster bus)与其他的节点进行通信的。通讯时，使用特殊的端口号，即对外服务端口号加10000。



参考Redis的学习-一致性哈希与哈希槽.docx，虽然数据是分开存储在不同节点上的，但是对客户端来说，整个集群Cluster，被看做一个整体。客户端连接任意一个node，看起来跟操作单实例的Redis一样。当客户端操作的key没有被分配到正确的node节点时，Redis会返回转向指令(请求重定向)，最后指向正确的node，这就有点像浏览器页面的302重定向跳转。

请求重定向：MOVED和ASK

客户端连接的redis结点，没有包含传送过来的key时，结点就会返回moved错误，客户端需要根据返回的正确的指令来重新访问：



返回的MOVED指令包含key所对应的槽，以及IP和端口。

针对重定向，较好的处理方式是使用smart client，在客户端缓存槽信息，如果碰到了MOVED错误，就更新缓存的槽信息，这样就能节省大量时间。

槽的迁移：

如果把结点A上的槽333迁移到结点B上，则需要把槽333上对应的key-value都迁移过去，在迁移过程中，部分key在结点A，部分key在结点B，在迁移完成前，其它结点仍然会认为是结点A在负责槽333。当客户端访问时，请求会统一转到结点A，如果结点A上已经没有要访问的数据了，就会返回ASK重定向。ASK和MOVED一样会返回槽值，结点B的地址和端口。

MOVED和ASK重定向的区别在于：当客户端收到MOVED重定向时会选择更新本地的缓存，但是接收到ASK重定向并不会更新本地缓存。当节点A和节点B完成了桶333的迁移之后，节点B会向整个集群广播自己目前负责的桶的信息，之后整个集群的其他节点就会认为桶333由B负责，此时如果客户端再向节点A访问桶333的数据，那么节点A就会返回MOVED重定向。

Redis还支持客户端的分区，Java中支持客户端以类似一致性哈希算法或者代理中间件，将特定的key存储到特定的结点上，redis的master结点彼此之间没有联系，也不会相互影响，这样方便增删redis结点，坏处是需要自己处理故障转移，高可用等问题。