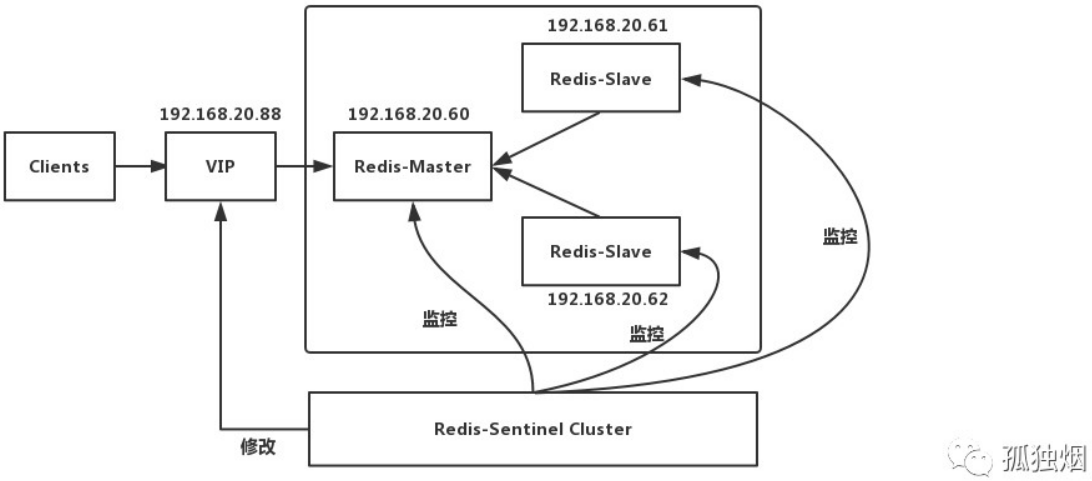
正文

老规矩，我还是以循序渐进的方式来讲，我一共经历过三套集群架构的演进！

Replication+Sentinel

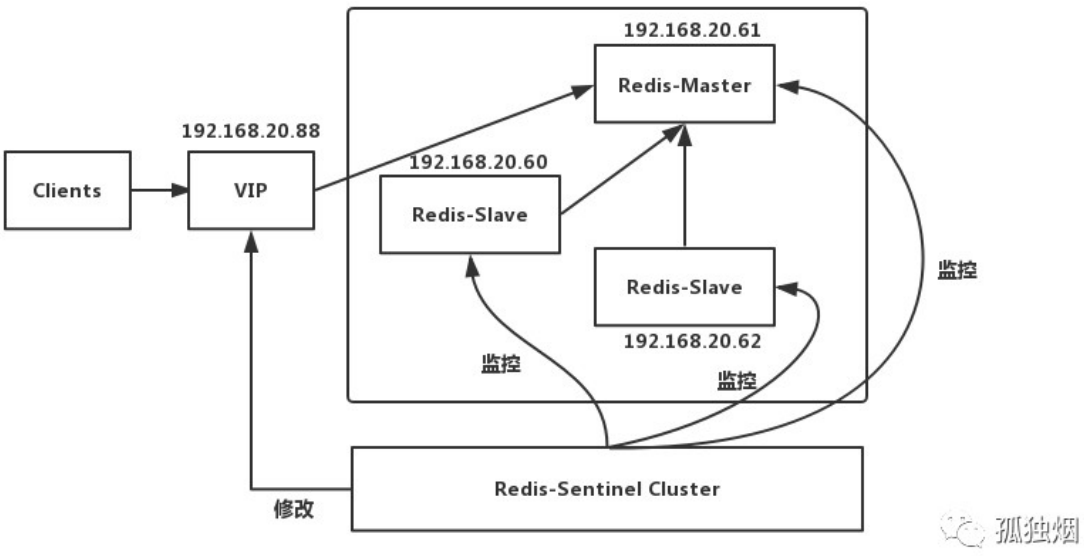
这套架构使用的是社区版本推出的原生高可用解决方案，其架构图如下！



这里Sentinel的作用有三个:

- **监控**:Sentinel 会不断的检查主服务器和从服务器是否正常运行。
- **通知**:当被监控的某个redis服务器出现问题，Sentinel通过API脚本向管理员或者其他的应用程序发送通知。
- **自动故障转移**:当主节点不能正常工作时，Sentinel会开始一次自动的故障转移操作，它会将与失效主节点是主从关系 的其中一个从节点升级为新的主节点，并且将其他的从节点指向新的主节点。

工作原理就是，当Master宕机的时候，Sentinel会选举出新的Master，并根据Sentinel中`client-reconfig-script`脚本配置的内容，去动态修改VIP(虚拟IP)，将VIP(虚拟IP)指向新的Master。我们的客户端就连向指定的VIP即可！ 故障发生后的转移情况，可以理解为下图



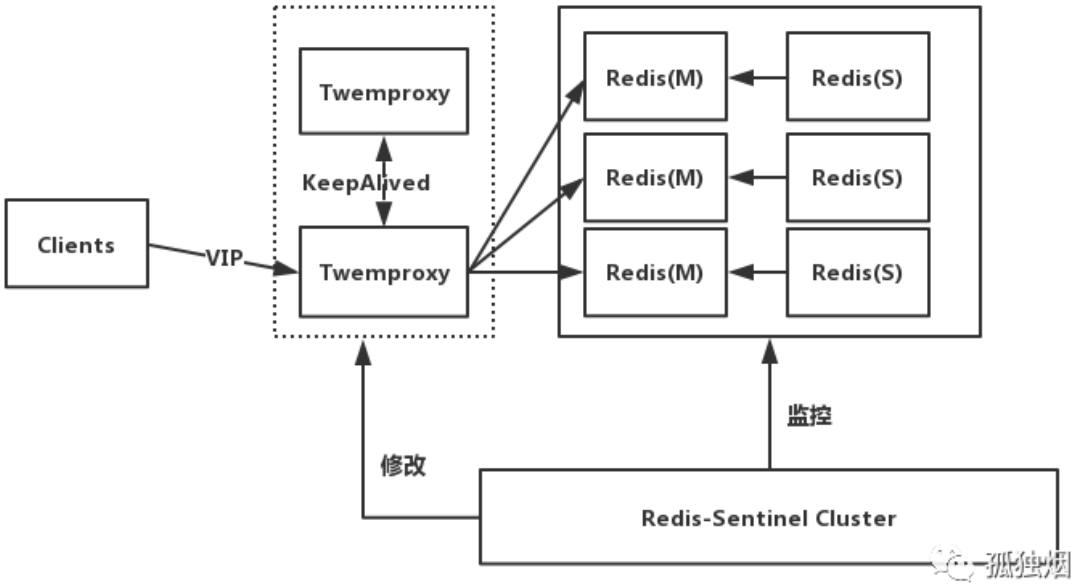
缺陷: (1)主从切换的过程中会丢数据 (2)Redis只能单点写，不能水平扩容

Proxy+Replication+Sentinel

这里的Proxy目前有两种选择:Codis和Twemproxy。我经历这套架构的时间为2015年，当时我好像咨询过我的主管为啥不用Codis和Redis官网的Redis Cluster。原因有二:

- 据说是因为Codis开源的比较晚，考虑到更换组件的成本问题。毕竟本来运行好好的东西，你再去换组件，风险是很大的。
- Redis Cluster在2015年还是试用版，不保证会遇到什么问题，因此不敢尝试。

所以我没接触过Codis，之前一直用的是Twemproxy作为Proxy。 这里以Twemproxy为例说明，如下图所示



工作原理如下

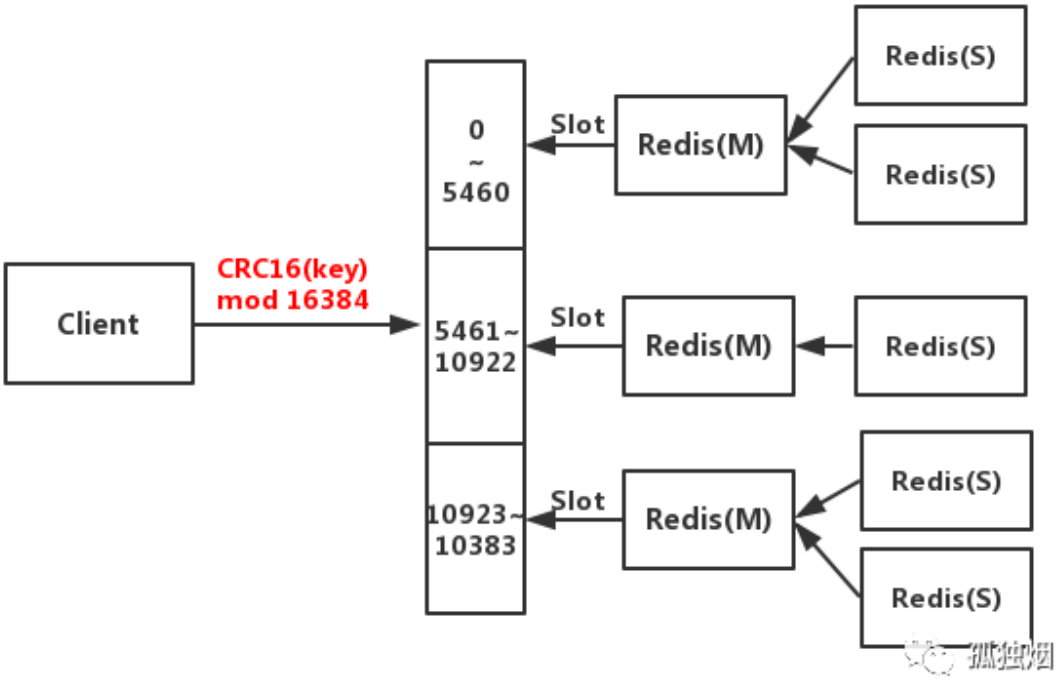
- 前端使用Twemproxy+KeepAlived做代理，将其后端的多台Redis实例分片进行统一管理与分配
- 每一个分片节点的Slave都是Master的副本且只读
- Sentinel持续不断的监控每个分片节点的Master，当Master出现故障且不可用状态时，Sentinel会通知/启动自动故障转移等动作
- Sentinel 可以在发生故障转移动作后触发相应脚本（通过 client-reconfig-script 参数配置 ），脚本获取到最新的Master来修改Twemproxy配置

缺陷: (1)部署结构超级复杂 (2)可扩展性差，进行扩缩容需要手动干预 (3)运维不方便

Redis Cluster

我经历这套架构的时间为2017年，在这个时间Redis Cluster已经很成熟了！你们在网上能查到的大部分缺点，在我接触到的时候基本已经解决! 比如没有完善的运维工具？可以参照一下搜狐出的`CacheCloud`。 比如没有公司在生产用过？我接触到的时候，百度贴吧，美团等大厂都用过了。 比如没有Release版？我接触到的时候距离Redis Cluster发布Release版已经很久。 而且毕竟是官网出的，肯定会一直维护、更新下去，未来必定会更加成熟、稳定。换句话说，Redis不倒，Redis Cluster就不

会放弃维护。所以，我推荐还是这套架构! 如下图所示



工作原理如下

- 客户端与Redis节点直连,不需要中间Proxy层，直接连接任意一个Master节点
- 根据公式`HASH_SLOT=CRC16(key) mod 16384`，计算出映射到哪个分片上，然后Redis会去相应的节点进行操作

具有如下优点: (1)无需Sentinel哨兵监控，如果Master挂了，Redis Cluster内部自动将Slave切换Master (2)可以进行水平扩容 (3)支持自动化迁移，当出现某个Slave宕机了，那么就只有Master了，这时候的高可用性就无法很好的保证了，万一master也宕机了，咋办呢？ 针对这种情况，如果说其他Master有多余的Slave ，集群自动把多余的Slave迁移到没有Slave的Master 中。

缺点: (1)批量操作是个坑 (2)资源隔离性较差，容易出现相互影响的情况。

烟哥彩蛋

在面试中如果碰到下列问题，如何应用上本篇的知识呢？先明确一点，我推荐的是Redis Cluster。
OK，开始举例说明

问题1:懂Redis事务么?

高调版: 我们在生产上采用的是Redis Cluster集群架构，不同的key是有可能分配在不同的Redis节点上的，在这种情况下Redis的事务机制是不生效的。其次，Redis事务不支持回滚操作，简直是鸡肋！所以基本不用!

问题2:Redis的多数据库机制，了解多少？ 正常版： Redis支持多个数据库，并且每个数据库的数据是隔离的不能共享，单机下的redis可以支持16个数据库（db0 ~ db15） 高调版: 在Redis Cluster集群架构下只有一个数据库空间，即db0。因此，我们没有使用Redis的多数据库功能!

问题3:Redis集群机制中，你觉得有什么不足的地方吗？ 高调版: 假设我有一个key，对应的value是Hash类型的。如果Hash对象非常大，是不支持映射到不同节点的！只能映射到集群中的一个节点上！还有就是做批量操作比较麻烦!

问题4:懂Redis的批量操作么？ 高调版: 我们在生产上采用的是Redis Cluster集群架构，不同的key会划分到不同的slot中，因此直接使用mset或者mget等操作是行不通的。

问题5:那在Redis集群模式下，如何进行批量操作？ 高调版:这个问题其实可以写一篇文章了，改天写。这里说一种有一个很简单的答法，足够面试用。即: 如果执行的key数量比较少，就不用mget了，就用串行get操作。如果真的需要执行的key很多，就使用Hashtag保证这些key映射到同一台redis节点上。简单来说语法如下

> **对于key为{foo}.student1、{foo}.student2，{foo}student3，这类key一定是在同一个redis节点上。因为key中"{}"之间的字符串就是当前key的hash tags， 只有key中{ }中的部分才被用来做hash，因此计算出来的redis节点一定是同一个!**

ps:如果你用的是Proxy分片集群架构，例如Codis这种，会将mget/mset的多个key拆分成多个命令发往不同得redis实例，这里不多说。我推荐答的还是redis cluster。

问题6:你们有对Redis做读写分离么？ 高调版:不做读写分离。我们用的是Redis Cluster的架构，是属于分片集群的架构。而redis本身在内存上操作，不会涉及IO吞吐，即使读写分离也不会提升太多性能，Redis在生产上的主要问题是考虑容量，单机最多10-20G，key太多降低redis性能.因此采用分片集群结构，已经能保证了我们的性能。其次，用上了读写分离后，还要考虑主从一致性，主从延迟等问题，徒增业务复杂度。

redis集群（redis cluster）基础知识详解

1. redis3以后，节点之间提供了完整的sharding（分片）、replication（主备感知能力）、failover（故障转移）的特性
2. 配置一致性：

   每个节点（Node）内部都保存了集群的配置信息，存储在clusterState中，通过引入自增的epoch变量来使得集 群配置在各个节点间保持一致
3. sharding数据分片

- - 将所有数据划分为16384个分片（slot），每个节点会对应一部分slot，每个key都会根据分布算法映射到16384个slot中的一个，分布算法为slotId=crc16(key)%16384
  - 当一个client访问的key不在对应节点的slots中，redis会返回给client一个moved命令，告知其正确的路由信息从而重新发起请求。client会根据每次请求来缓存本地的路由缓存信息，以便下次请求直接能够路由到正确的节点
  - 分片迁移：分片迁移的触发和过程控制由外部系统完成，redis只提供迁移过程中需要的原语支持。主要包含两种：一种是节点迁移状态设置，即迁移前标记源、目标节点；另一种是key迁移的原子化命令

1. failover故障转移

- - 故障发现：节点间两两通过TCP保持连接，周期性进行PING、PONG交互，若对方的PONG相应超时未收到，则将其置为PFAIL状态，并传播给其他节点
  - 故障确认：当集群中有一半以上的节点对某一个PFAIL状态进行了确认，则将起改为FAIL状态，确认其故障
  - slave选举：当有一个master挂掉了，则其slave重新竞选出一个新的master。
  - 主要根据各个slave最后一次同步master信息的时间，越新表示slave的数据越新，竞选的优先级越高，就更有可能选中。竞选成功之后将消息传播给其他节点。

1. 集群不可用的情况：

- - 集群中任意master挂掉，且当前master没有slave。
  - 集群中超过半数以上master挂掉。
  - 有A，B，C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用。