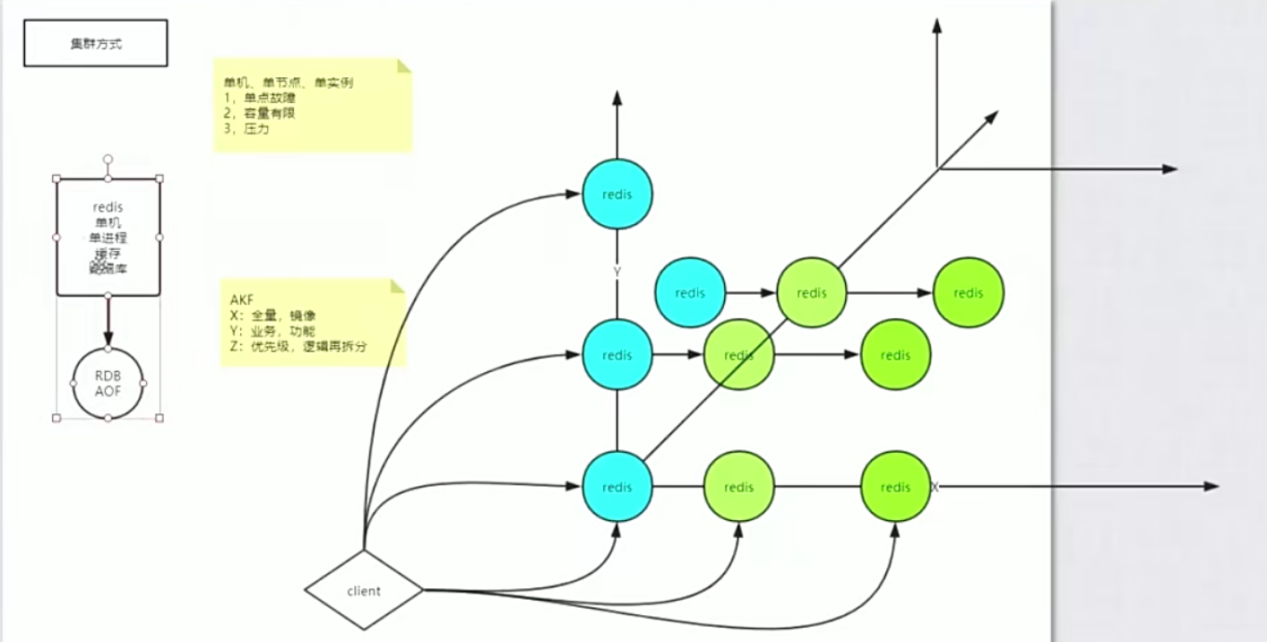
1. 单机问题

AKF

X：按照主备拆分

Y：按照业务逻辑拆分

Z：按照优先级、逻辑拆分。用户数据1个亿，将0-1w放在一个redis，1w-100w放在另外一个redis。一个业务数据放在多个redis存储



（1）解决单点故障：每个主都有多个备。（x）

（2）解决容量有限：每个点不是承载公司所有数据，它的数据量根据拆分已经变得很小了， 就能够发挥单机性能，并能够解决容量问题。（y、z）

（3）解决压力：每个点数据很小，访问压力也不是很大。（y、z）

1. CAP

一致性C：节点间数据是一致的

可用性A：系统对外体现为可用的

分区容错性：从节点允许出现故障

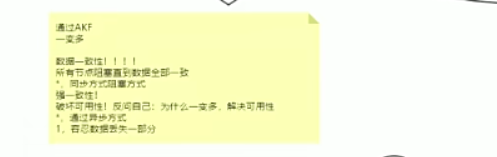
1. AKF------>单点故障，主从复制、哨兵（AP，没有强一致性）复制集群

主从：客户端既可以访问主，也可以访问从

主备：客户端只能访问主，备只是用来替代主的

可用性：集群系统对外表现为可用的，即时系统里面有故障

1. 主从复制：一致性问题

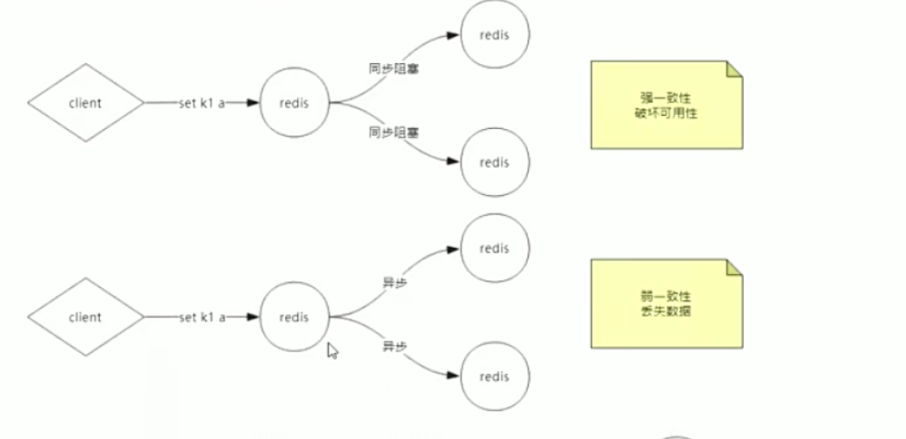


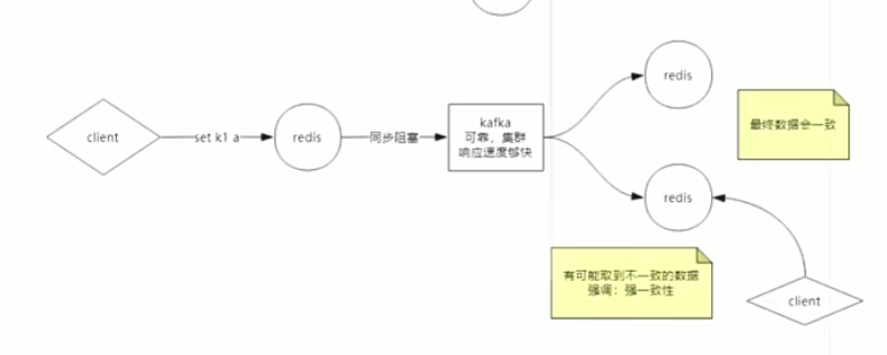
强一致性：处理客户端数据，阻塞所有节点做一致性工作

弱一致性：收到客户端数据，以异步方式将数据发给从

最终一致性：收到客户端数据，同步数据到中间件

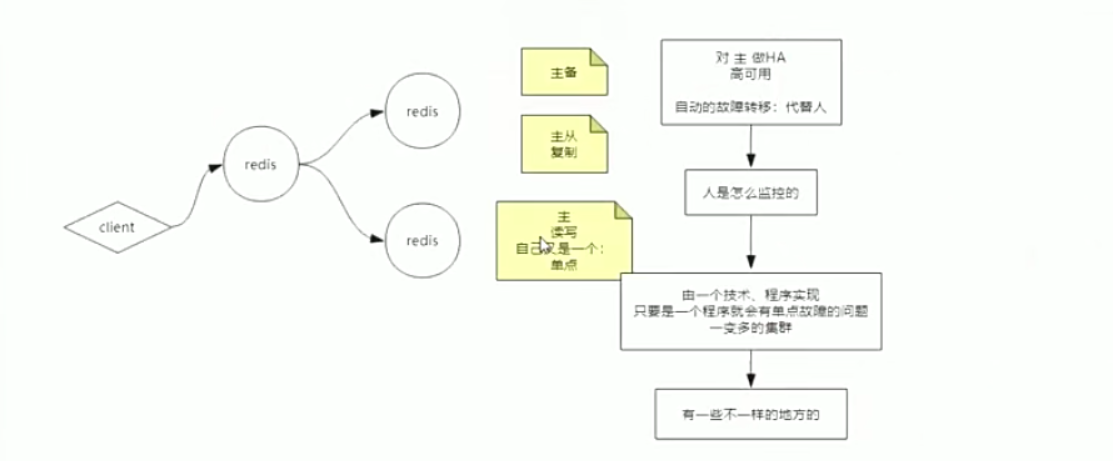
Redis采用弱一致性，具有低延迟、高性能，但是数据可能丢失，主出故障之后，可能数据还没来得及写到从

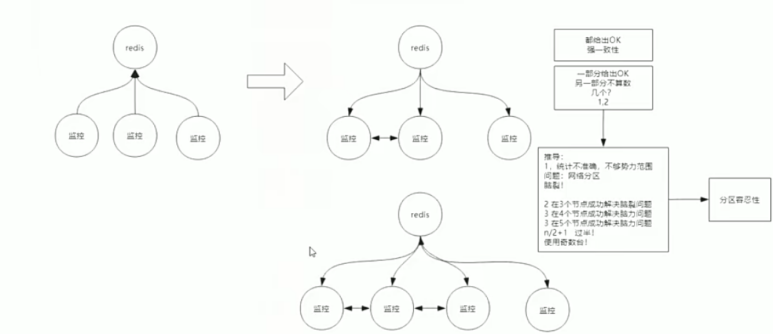




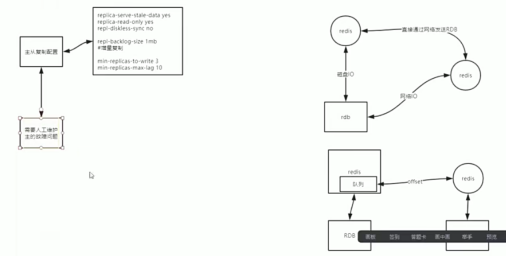
1. 哨兵（HA）：主故障

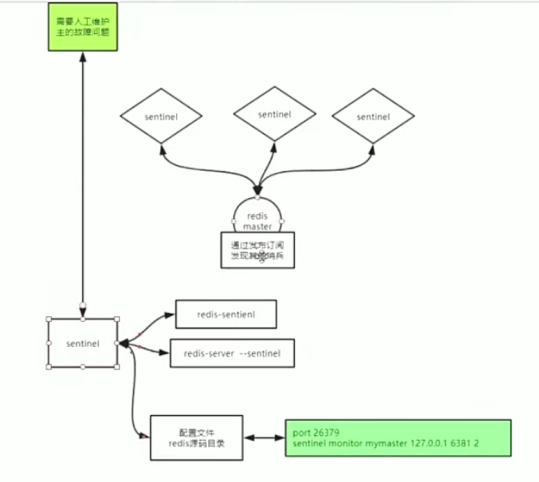
哨兵监控，使用奇数个哨兵，因为奇数个和偶数个容错性一样（允许故障的哨兵数），但是偶数的风险更大。





1. 配置



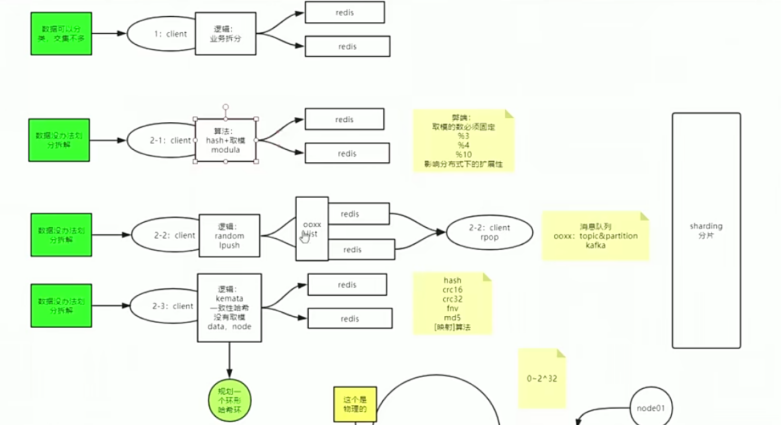


1. AFK-容量、压力 分片集群

解决容量、压力：部署多台redis服务器。

如何找到对应的redis服务器：通过hash算法定位到某台redis服务器，若在计算得到的redis服务器上找不到数据，就直接访问数据库。

（1）选取服务器的算法



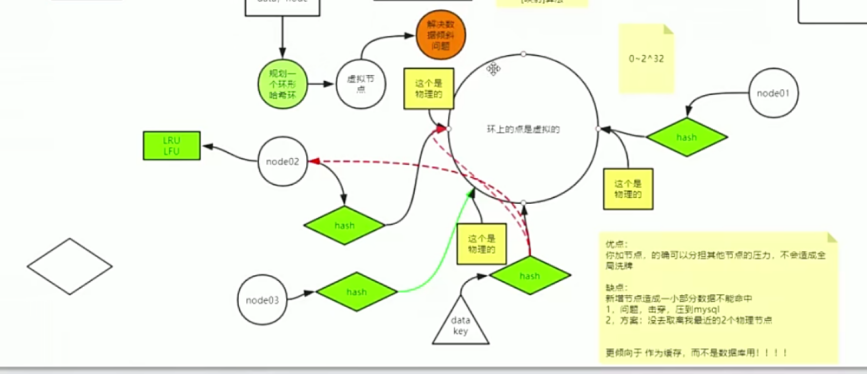
a）取模算法------>几乎不用

优点：算法简单，分散效果好

缺点：取模算法很明显，结果很容易受N的影响，当服务器数量N增加或者减少的时候，原先的缓存数据定位几乎失效，缓存数据定位失效意味着要到数据库重新查询（击穿），这对于高并发的系统来说是致命的。（节点变化，数据几乎全部失效）

b）Random----->消息队列

c）一致性算法



虚拟节点：一台服务器做多次hash，这样一个物理机就会有多个hash点，防止都落在一个服务器上（数据倾斜）

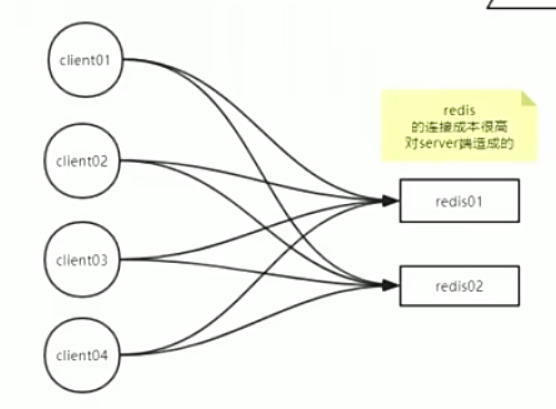
优点：最大限度的减少了server节点变化带来的影响，当节点变化时，只影响一个server节点的部分数据，且hash算法能够保证需要重新分布的缓存数据能映射到新的server节点中。（节点变化，只会影响一个节点数据）

1. 架构一

缺点：架构一无法用做数据库，因为数据很可能找不到

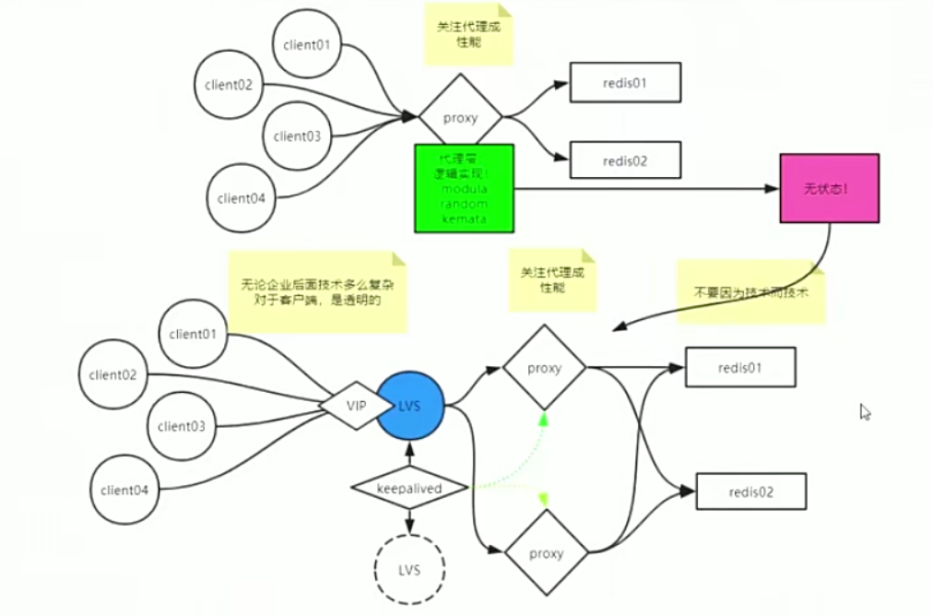
1. 客户端通过算法选择服务器

缺点：对server，redis连接成本很高，因为每个客户端连接都会进行三次握手、四次分手



1. 让代理选择服务器

优点：让客户端变得简单轻盈



1. 架构二

解决数据找不到问题

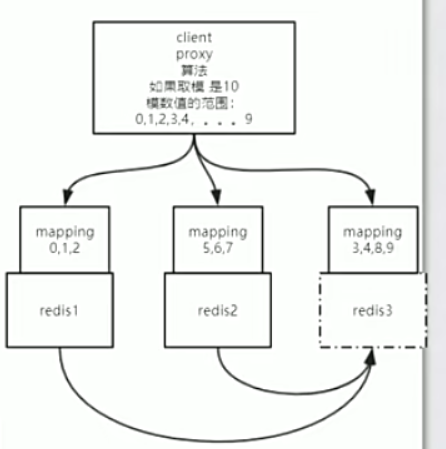
弊端：很难做到聚合，即一个事务，发生在多个服务器；此时就需要程序员把同一类数据尽量放在同一个服务器

1. 预分区

Hash模数预先大于服务器数，这样服务器数量增长，也不会对原先数据照成影响。

然后每个服务器存储mapping，即有多个槽位。

数据通过取模n算出槽位，再通过mapping定位到服务器上。



1. redis集群

每个redis服务器维护自己以及其他服务器mapping，找到直接返回数据，找不到返回需要重定向到其他服务器

set：计算槽位，在当前redis则存储，不在当前redis则重定向到正确的redis进行set

get：计算槽位，在当前redis则返回数据，不在当前redis则重定向到正确的redis进行get

