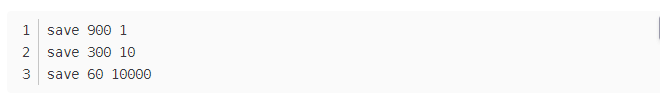
**1.Redis备份(持久化)**

在 Redis 中存在两种方式的备份: 一种是快照(snapshotting)，它是备份当前瞬间 Redis 在内存中的数据记录;另一种是只追加文件(Append-OnlyFile, AOF)， 其作用就是当 Redis 执行写命令后，在一定的条件下将执行过的写命令依次保存在 Redis 的文件中， 将来就可以依次执行那些保存的命令恢复 Redis 的数据了。对于快照备份而言， 如 果当前 Redis 的数据量大，备份可能造成 Redis 卡顿，但是恢复重启是 比较’快速的 ;对于 AOF 备份而言，它只是追加写入命令，所以备份一般不会造成Redis卡顿， 但是恢复重启要执行更多的命令，备份文件可能也很大 ， 使用者使用的时候要注意。在 Redis 中允许使用其中的 一种、同时使用两种，或者两种都不用，所以具体使用何种方式进行备份和持久化是用户可以通过配置决定的。 对于Redis而言，它的默认配置(在redis.conf文件中)为:



快照模式的配置项：



这 3 个配置项的含义分别为 :

* 当 900 秒执行 1 个 写命令时，启用快照备份。
* 当 300 秒执行 10 个写命令时，启用快照备份 。
* 当 60 秒内执行 10000 个写命令时，启用快照备份。

1. Redis 执行 save 命令的时候，将禁止写入命令 。



1. **bgsave是一个异步保存命令，也就是系统将启动另外一条进程，把 Redis 的数据保存到对应的数据文件中**。它和 save 命令最大的不同是它不会阻塞客户端的写入，也就是在执行 bgsave 的时候，允许客户端继续读/写 Redis。**在默认情况下，如果Redis 执行 bgsave 失败后，Redis 将停止接受写操作**，这样以一种强硬的方式让用户知道数据不能正确的持久化到磁盘 ， 否则就会没人注意到灾难的发生，如果后台保存进程重新启动工作了， Redis 也将自动允许写操作。然而如果安装了靠谱的监控，可能不希望 Redis 这样做，那么你可以将其修改为 no。



​这个命令意思是是否对 rbd 文件进行检验，如果是将对 rdb 文件检验。从 dbfilename的配置可以知道，rdb文件实际是 Redis 持久化的数据文件 。

1. rdb是数据文件。当采用快照模式备份(持久化)时， Redis 将使用它保存数据，将来可以使用它恢复数据 。

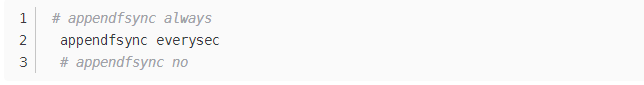


1. 如果 appendonly配置为 no，则不启用 AOF方式进行备份。如果 appendonly配置为 yes,则以 AOF 方式备份 Redis 数据，那么此时 Redis 会按照配置，在特定的时候执行追加命令，用以备份数据。



1. 这里定义追加的写入文件为 appendonly.aof， 采用 AOF 追加文件备份的时候命令都会写到这里。



1. AOF 文件和 Redis 命令是同步频率的 ， 假设配置为 always， 其含义为当 Redis 执行命令的时候，则同时同步到 AOF 文件， 这样会使得 Redis 同步刷新 AOF 文件， 造成缓慢。而采用 everysec 则代表每秒同步一次命令到 AOF 文件。采用 no 的时候 ，则 由客户端调用命令执行备份， Redis 本身不备份文件。对于采用 always 配置的时候， 每次命令都会持久化，它的好处在于安全，坏处在于每次都持久化性能较差。采用 everysec 则每秒同步 ， 安全性不如 always， 备份可能会丢失 l 秒 以内的命令 ， 但是隐患 也不大 ， 安全度 尚可 ，性能可以得到保障。采用 no， 则性能有所保障 ， 但是由于失去备份， 所以安全性比较差。一般建议使用everysec。
2. 它指定是否在后台 AOF 文件 rewrite (重写)期间调用 fsync， 默认 为 no， 表示要调用fsync (无论后台是否有子进程在刷新)。Redis在后台写 RDB文件或重写 AOF文件期间会存在大量磁盘1/0， 此时， 在某些Linux系统中，调用fsync可能会阻塞。



1. 它指定 Redis 重写 AOF 文件的条件 ， 默认为 100，表示与上次 rewrite 的 AOF 文件大小相比，当前 AOF 文件增长量超过上次 AOF 文件大小的 100%时，就会触发 background rewrite。若配置为0， 则会禁用自动 rewrite。



1. 它指定触发 rewrite 的 AOF文件大小。若 AOF文件小于该值，即使当前文件的增量比例达到 auto-aof-rewrite-percentage 的配置值 ，也不会触发自动 rewrite。 即这两个配置项同时满足时，才会触发 rewrite。



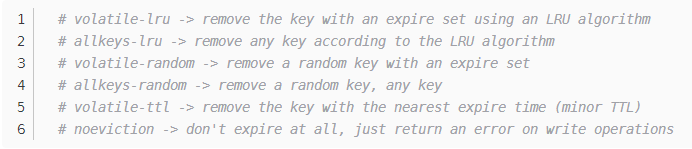
1. Redis在恢复时会忽略最后一条可能存在问题的指令， 默认为yes。 即在AOF写入时，可能存在指令写错的问题(突然断电、写了一半)， 这种情况下 yes 会 log 并继续，而no会直接恢复失败 。



**2.Redis 内存回收策略**

Redis 也会因为内存不足而产生错误，也可能因为回收过久而导致系统长期的停顿，因此掌握执行回收策略十分有必要。在 Redis 的配置文件中，当 Redis 的内存达到规定的最大值时，允许配置 6 种策略中的一种进行淘汰键值，并且将一些键值对进行回收。

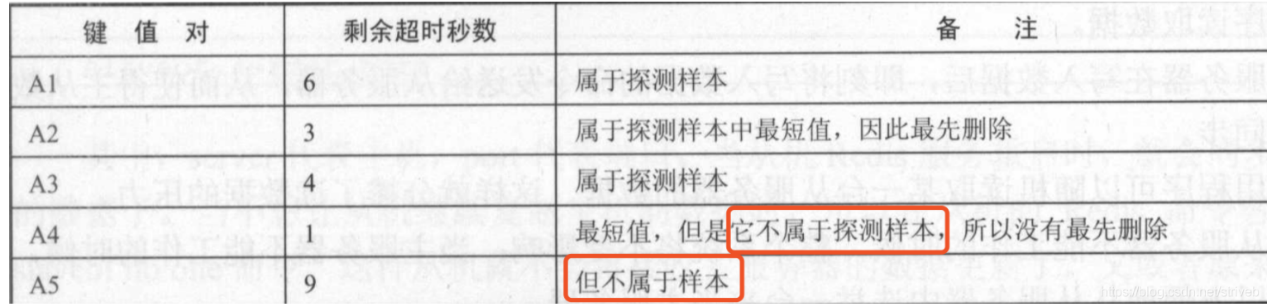
在Redis.conf中，Redis 对其中的一个配置项一maxmemory-policy：



* volatile-lru: 采用最近使用最少的淘汰策略， Redis 将回收那些超时的(仅仅是超时的)键值对 ，也就是它只淘汰那些超时的键值对。
* allkeys-lru: 采用淘汰最少使用的策略， Redis将对所有的(不仅仅是超时的)键值对采用最近使用最少的淘汰策略。
* volatile-random:采用随机淘汰策略删除超时的(仅仅是超时的)键值对 。
* allkeys-random: 采用随机、淘汰策略删除所有的(不仅仅是超时的)键值对，这个策略不常用 。
* volatile-ttl: 采用删除存活时间最短的键值对策略 。
* noeviction: 根本就不淘汰任何键值对，当内存已满时， 如果做读操作，例如get命令，它将正常工作，而做写操作，它将返回错误 。 也就是说，当 Redis 采用这个策略内存达到最大的时候，它就只能读而不能了。

Redis 在默认情况下会采用 noeviction 策略。换句话说，如果内存己满，则不再提供写入操作 ，而只提供读取操作 。显然这往往并不能满足我们的要求，因为对于互联网系统而 言 ， 常常会涉及数以百万甚至更多的用户 ， 所以往往需要设置回收策略。

这里需要指出的是 : LRU 算法或者 TTL 算法都不是很精确算法，而是一个近似的算法。 Redis不会通过对全部的键值对进行比较来确定最精确的时间值，从而确定删除哪个键值对，因为这将消耗太多的时间，导致回收垃圾执行的时间太长，造成服务停顿。而在 Redis 的默认配置文件中，存在着参数 maxmemo-samples，它的默认值为 3，假设采取了 volatile-ttl算法，让我们去了解这样的一个回收的过程，假设当前有 5 个即将超时的键值对， 如表所示 。



由于配置 maxmemory-samples 的值为 3，如果 Redis 是按表中的顺序探测，那么它只会取到样本 Al、 A2, A3，然后进行比较，因为 A2 过期剩余秒数最少，所以决定淘汰 A2,因此 A2是最先被删除的。注意，此时即将过期且剩余超时秒数最短的 A4却还在内存中，因为它不属于探测样本。这就是 Redis 中采用的近似算法。当设置 maxmemory-samples 越大，则 Redis 删除的就越精确，但是与此同时带来不利的是，Redis 也就需要花更多的时间去计算和l匹配更为精确的值 。

回收超时策略的缺点是必须指明超时的键值对 ，这会给程序开发带来一些设置超时的代码，无疑增加了开发者的工作量。对所有的键值对进行回收 ，有可能把正在使用的键值对删掉 ，增加了存储的不稳定性 。对于垃圾回 收的策略，还需要注意的是回收的时间，因为在 Redis 对垃圾的回收期间， 会造成系统缓慢。因 此，控制其回收时间有 一定好 处，只是这个时间不能过短或过长。过短则会造成回收次数过于频繁 ，过长则导致系统单次垃圾回收停顿时间过长 ，都不利于系统的稳定性，这些都 需要设计者在实际 的工作中进行思考。

**3.复制**

**3.1 主从同步基础概念**

联网系统一般是以主从架构为基础的，所谓主从架构设计的思路大概是:

* 在多台数据服务器中，只有一台主服务器，而主服务器只负责写入数据，不负责让外部程序读取数据。
* 存在多台从服务器，从服务器不写入数据，只负责同步主服务器的数据，并让外部程序读取数据。
* 主服务器在写入数据后，即刻将写入数据的命令发送给从服务器，从而使得主从数 据同步。
* 应用程序可以随机读取某一台从服务器的数据， 这样就分摊了读数据的压力。
* 当从服务器不能工作的时候，整个系统将不受影响；当主服务器不能工作的时候，可以方便地从从服务器中选举一 台来当主服务器 。



**3.2 Redis 主从同步配置**

对Redis进行主从同步的配置分为主机与从机，主机是一台，而从机可以是多台。

首先，明确主机。当你能确定哪台机子是主机的时候，关键的两个配置是 dir 和 dbfilename 选项， 当然必须保证这两个文件是可写的。对于 Redis 的默认配置而言， dir 的 默认值为“./”，而对于 dbfilename 的默认值为“ dump.rbd”。换句话说，默认采用 Redis 当前目录的 dump.rbd 文件进行同步。对于主机而言，只要了解这么多信息，很简单。其次，在明确了从机之后,进行进一步配置所要关注的只有 slaveof这个配置选项 ，它的配置格式是 :

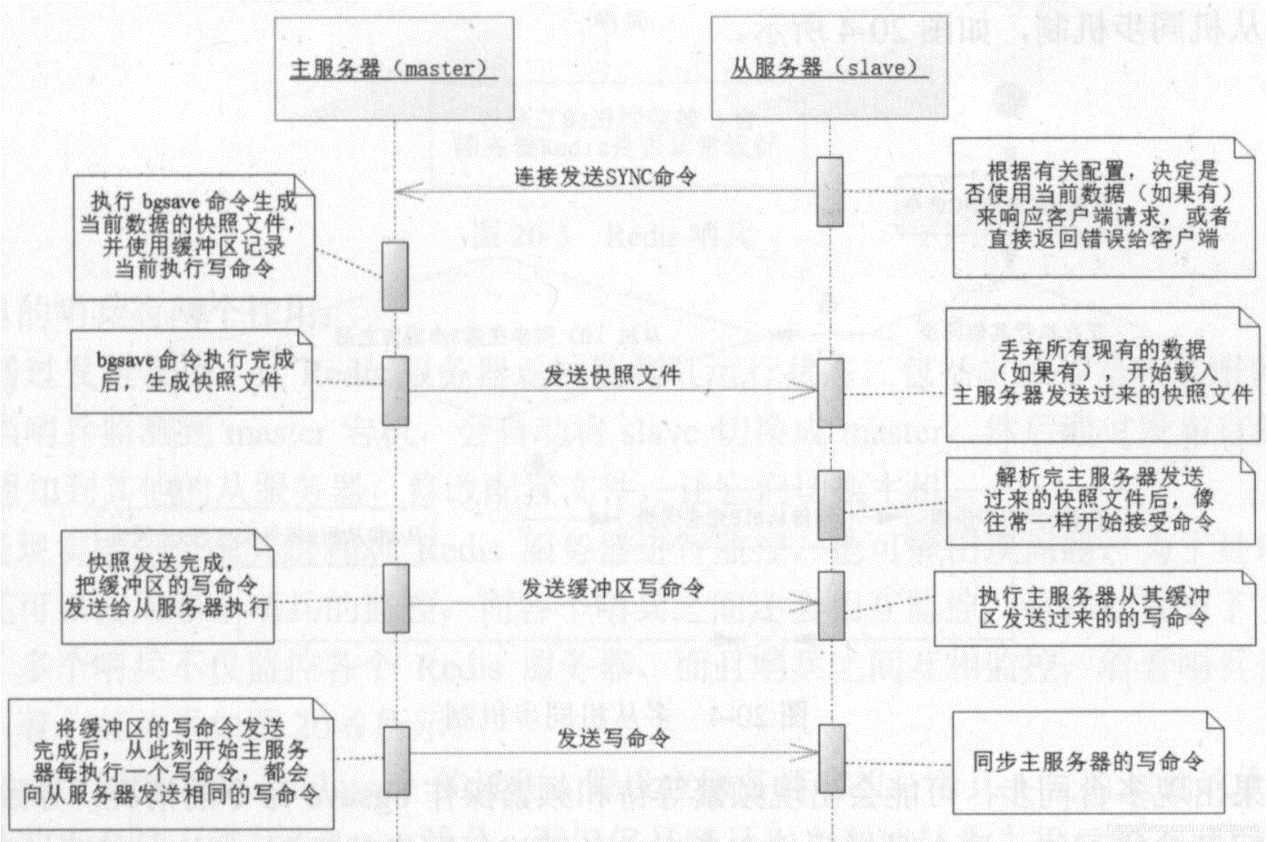


server代表主机， port 此代表端口。当从机Redis服务重启时，就会同步对应主机的数据了。当不想让从机继续复制主机的数据时，可以在从机的 Redis 命令客户端发送 slaveof no one命令，这样从机就不会再接收主服务器的数据更新了。又或者原来主服务器已经无法工作了，而你可能需要去复制新的主机，这个时候执行 slaveof sever port就能让从机复制另外一台主机的数据了。

redis.conf中还有一个 bind 的配置 ，默认为 127.0.0.1, 也就是只允许本机访 问 ，把它修改为 bind 0.0.0.0，其他的服务器就能够访问了。

**3.3 Redis 主从同步的过程**

Redis主从同步过程图如下：



(1) 无论如何要先保证开启主服务器，开启主服务器后，从服务器通过命令或者重启配置项可以同步到主服务器。

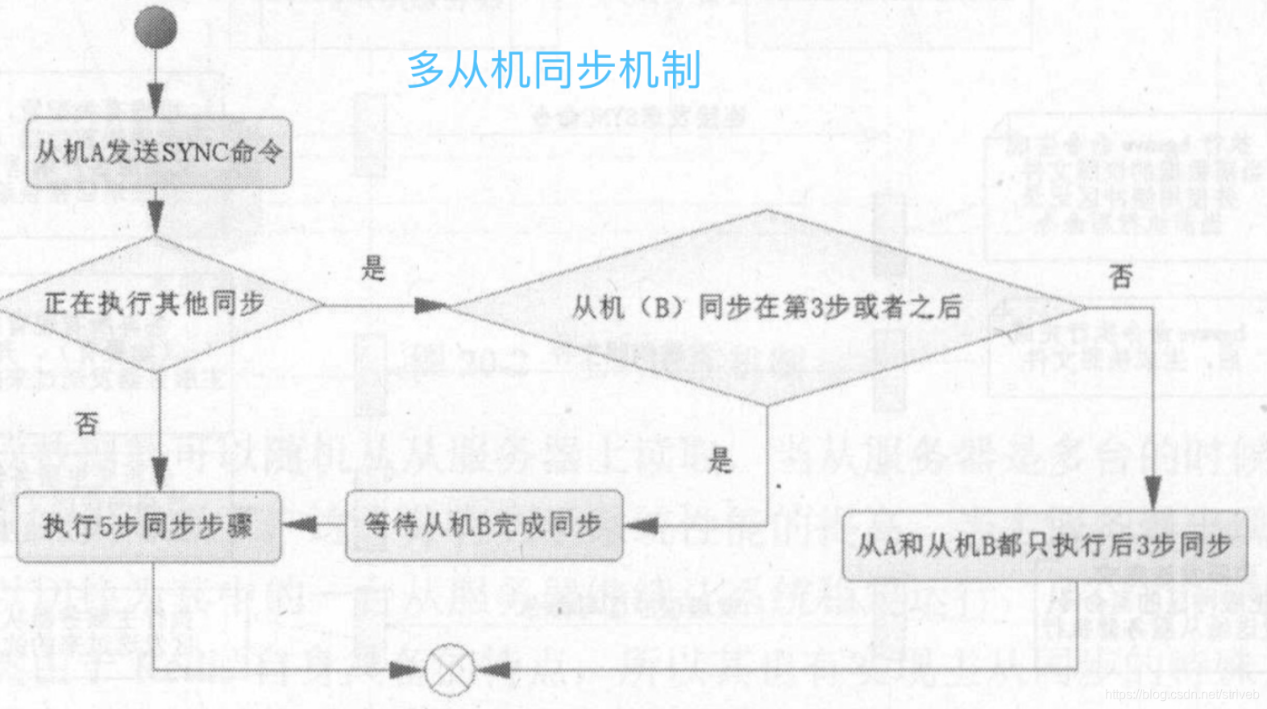
(2)当从服务器启动时，读取同步的配置，根据配置决定是否使用当前数据响应客户端，然后发送 SYNC 命令。当主服务器接收到同步命令的时候，就会执行bgsave命令备份数据，但是主服务器并不会拒绝客户端的读/写，而是将来自客户端的写命令写入缓冲区。从服务器未收到主服务器备份的快照文件的时候，会根据其配置决定使用现有数据响应客户端或者拒绝。

(3)当bgsave命令被主服务器执行完后，开始向从服务器发送备份文件，这个时候从服务器就会丢弃所有现有的数据，开始载入发送的快照文件。

(4)当主服务器发送完备份文件后，从服务器就会执行这些写入命令。此时就会把bgsave 执行之后的缓存区内的写命令也发送给从服务器，从服务完成备份文件解析，就开始像往常一样，接收命令，等待命令写入。

(5)缓冲区的命令发送完成后，当主服务器执行一条写命令后，就同时往从服务器发送同步写入命令，从服务器就和主服务器保持一致了。而此时当从服务器完成主服务器发送的缓冲区命令后，就开始等待主服务器的命令了。

只是在主服务器同步到从服务器的过程中，需要备份文件，所以在配置的时候一般需要预留 一些内存空间给主服务器，用以腾出空间执行备份命令。 一般来说主服务器使用 50%~65%的内存 空间 ，为主从复制留下可用的内存空间。



如果出现多台同步，可能会出现频繁等待和频繁操作 bgsave 命令的情况，导致主机在较长时间里性能不佳，这个时候我们会考虑主从链进行同步的机制，以减少这种可能 。

**4.哨兵( Sentinel )模式**

Redis可以存在多台服务器，并且实现了主从复制的功能。哨兵模式是一种特殊的模式，首先 Redis提供了哨兵的命令，哨兵是一个独立的进程，作为进程，它会独立运行。其原理是哨兵通过发送命令， 等待Redis服务器响应，从而监控运行的多个Redis实例。



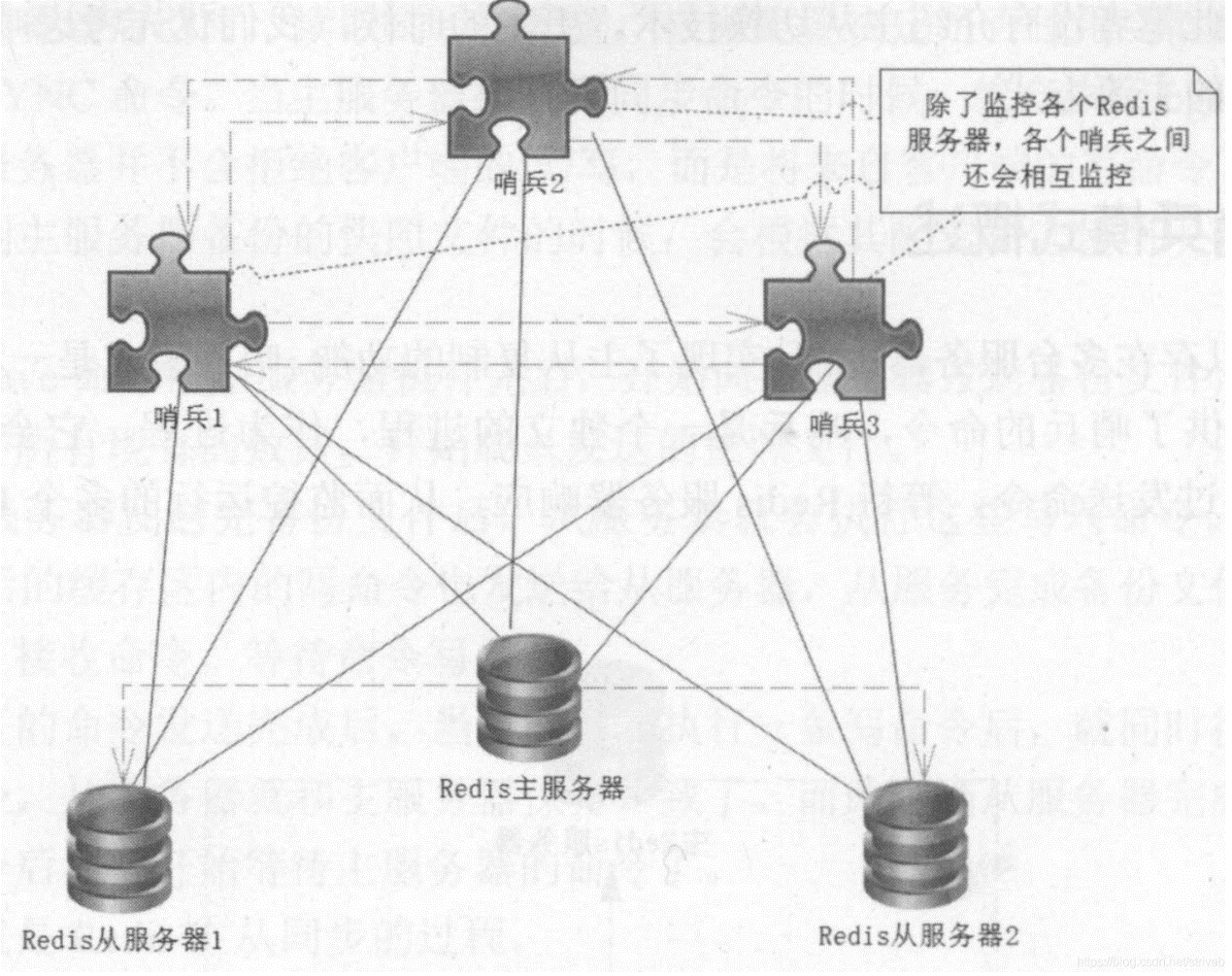
这里的哨兵有两个作用 :

* 通过发送命令，让Redis服务器返回监测其运行状态，包括主服务器和从服务器。
* 当哨兵监测到 master 宕机， 会自动将 slave 切换成 master，然后通过发布订阅模式通知到其他的从服务器，修改配置文件，让它们切换主机 。

但其实在使用的过程中会通过多个哨兵进行监控，即哨兵不但监控Redis服务器，哨兵之间也在相互监控着，判断哨兵是否“存活”。

故障切换(failover)处理过程：

假设主服务器岩机，哨兵1先监测到这个结果，当时系统并不会马上进行 failover操作 ，而仅仅是哨兵 l 主观地认为 主机己经不可用，这个现象被称为主观下线。当后面的哨兵监测也监测到了主服务器不可用 ， 并且有了 一 定数量的哨兵认为主服务器不可用，那么哨兵之间就会形成一次投票，投票的结果由一个哨兵发起，进行 failover操作，在 failover操作的过程中切换成功后，就会通过发布订阅方式，让各个哨兵把自己监控的服务器实现切换主机 ，这个过程被称为客观下线。这样对于客户端而言， 一切都是透明的，即不可见的。

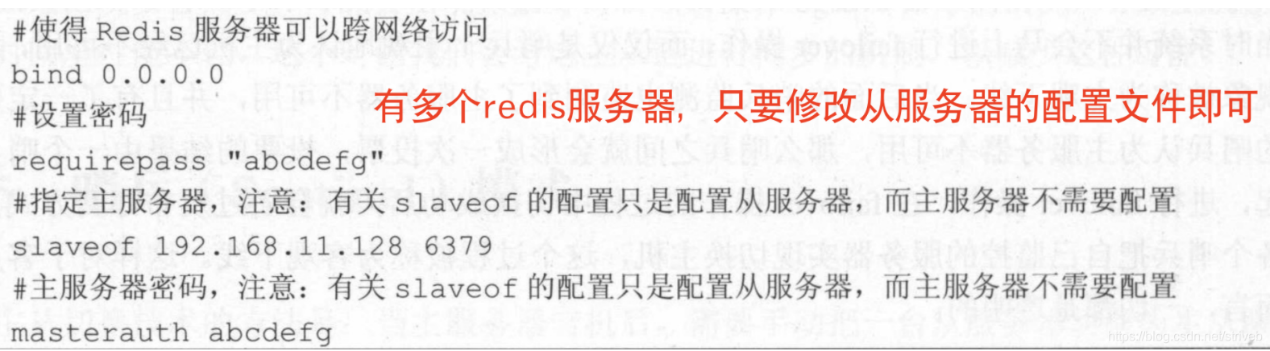


**4.1 搭建哨兵模式**

配置 3 个哨兵和 l 主 2 从的 Redis 服务器来演示这个过程。



结构就是如上图所示，下面是修改Redis配置文件：



上述内容主要是配置 Redis服务器，从服务器比主服务器多一个 slaveof的配置和密码， 这里配置的 bind使得 Redis服务器可以跨网段访问 。 而对于外部的访问还需要提供密码， 因此还提供了requirepass的配置，用以配置密码 ，这样就配置完了 3 台服务器 。

配置3个哨兵 ，每一个哨兵的配置都是一样的 ，在 Redis 安装目录下可以找到 sentinel.conf文件，然后对其进行修改。下面对 3 个哨兵的这个文件作出修改，同样也是在原有的基础上进行修改 ，如下所示。



启动哨兵模式：



注意服务器启动的顺序 ， 首先是主机 (192.168.11.128) 的 Redis 服务进程 ， 然后启动从机的服务进程，最后启动 3 个哨 兵的服务进程。

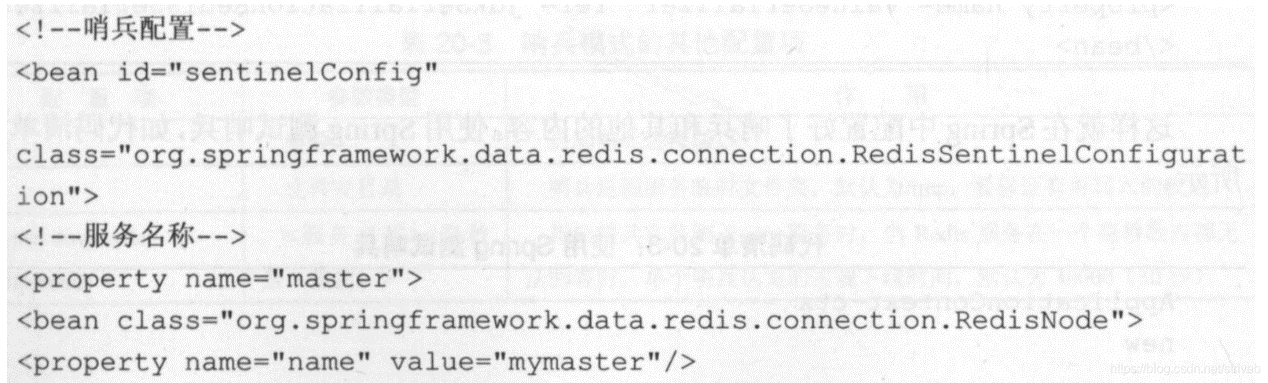
**4.2 在Java中使用哨兵模式**

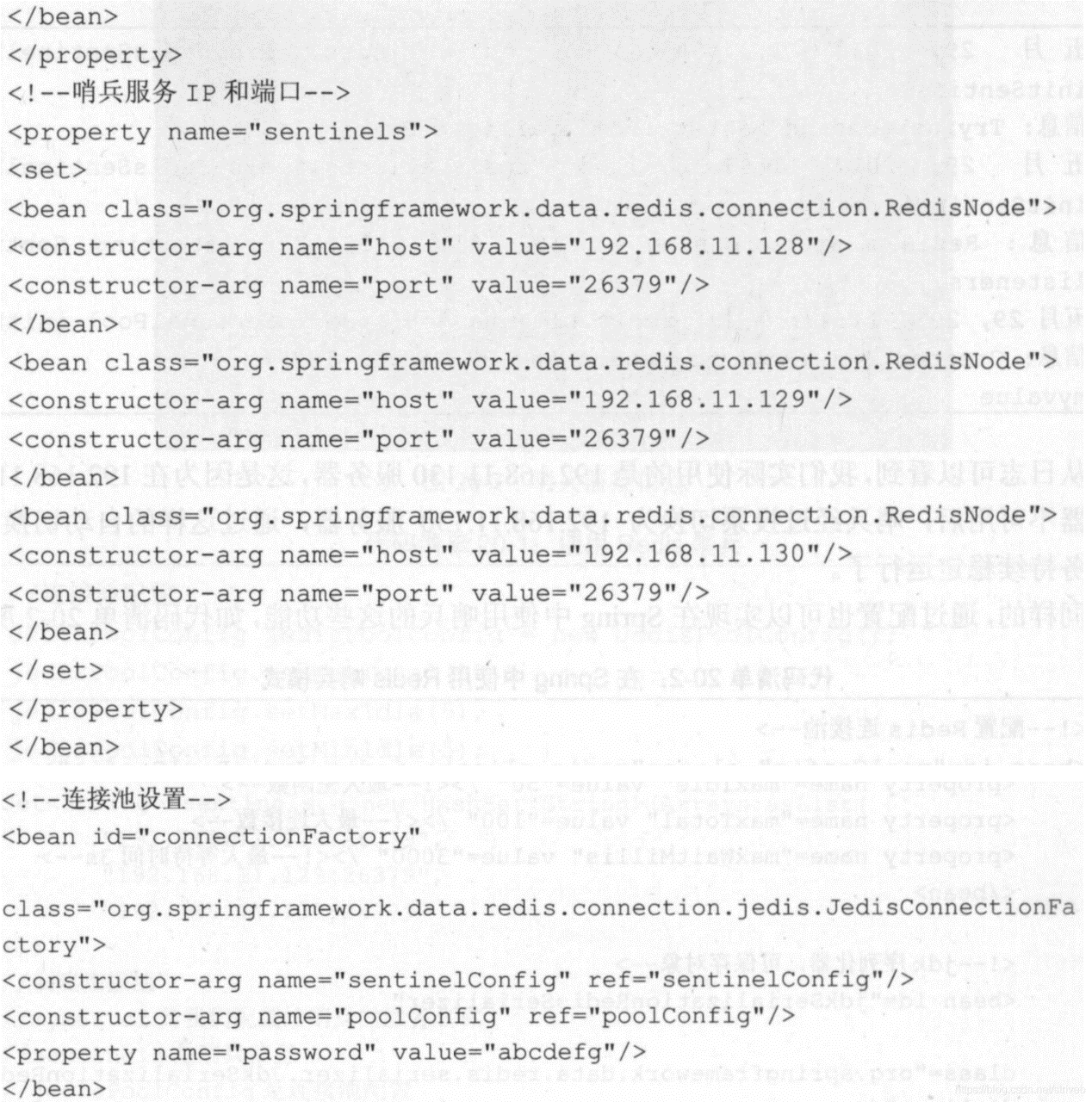
代码如下：



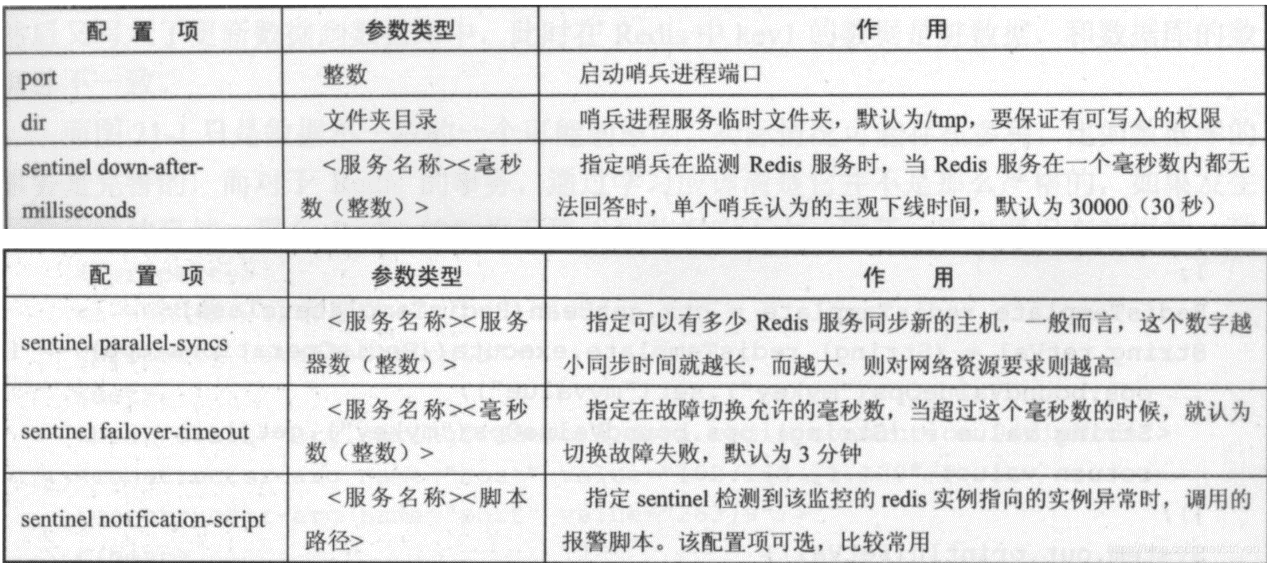
Redis 哨兵默认超时 3 分钟后才会进行投票切换主机。

在spring中使用哨兵模式：





**4.3 哨兵模式的其他配置项**



sentinel down-after-milliseconds 配置项只是一个哨兵在超过其指定的毫秒数依旧没有得到回答消息后，会自己认为主机不可用。对于其他哨兵而言，并不会认为主机不可用。哨兵会记录这个消息， 当拥有认为主观下线的哨兵到达 sentinel monitor所配置的数量的时候， 就会发起一次新的投票， 然后切换主机，此时哨兵会重写 Redis 的哨兵配置文件， 以适应新场景 的需要。