**第十三章 redis**

**本节所讲：**

**13.1 redis概述**

**13.2 安装redis**

**13.3 redis基本操作**

**13.4 redis持久化**

**13.5 redis主从架构（实现读写分离）**

**13.6 使用sentinel实现主从架构高可用**

**13.1 redis概述**

**13.1.1 redis是什么：**

**redis的出现时间并不长，是NoSQL中的一种，基于键-值型的存储，与memcache类似，但是memcache中只是内存的缓存，而redis不仅是内存中的缓存，还提供持久存储，在2009年第一次发布redis**

**Redis 全称（REmote DIctionary Server）远程字典服务器，而这个字典服务器从本质上来讲，主要是提供数据结构的远程存储功能的，可以理解为redis是一个高级的K-V存储，和数据结构存储，因为redis除了能够存储K-V这种简单的数据之外，还能够存储，列表、字典、hash表、等对应的数据结构**

**在性能上redis不比memcache差，因为redis整个运行通通都是在内存中实现的，它的所有的数据集都是保存在内存中的，内存中的数据会周期性的写入到磁盘上，以实现数据的持久功能，而这种写磁盘并不是用于访问，而仅是冗余功能，所以redis所有功能都在内存中完成，因为此性能也是可想而知**

**redis与mamcache不同之处在于redis有一个周期性的将数据保存到磁盘上的机制，而且不只一种，有两种机制，这也是redis持久化的一种实现，另外与mamcache有所区别的是，redis是单线程服务器，只有一个线程来响应所有的请求**

**redis支持主从模式，但是redis的主从模式默认就有一个sentinel工具，从而实现主从架构的高可用，也就是说，redis能够借助于sentinel工具来监控主从节点，当主节点发生故障时，会自己提升另外一个从节点成为新的主节点**

**在redis 3.0版本发布，开始支持redis集群，从而可以实现分布式，可以将用户的请求分散至多个不同节点**

**13.1.2 redis所支持的数据类型**

**支持存储的数据类型有、String（字符串，包含整数）, List（列表）, Hash（关联数组）, Sets（集合）, Sorted Sets（有序集合）, Bitmaps（位图）, HyperLoglog**

**13.1.3 redis性能评估：**

**1、100万较小的键存储字符串，大概消耗100M内存**

**2、由于redis是单线程，如果服务器主机上有多个CPU，只有一个能够使用，但并不意味着CPU会成为瓶颈，因为redis是一个比较简单的K-V数据存储，CPU通常不会成为瓶颈的**

**3、在常见的linux服务器上，500K（50万）的并发，只需要一秒钟处理，如果主机硬件较好的情况下，每秒钟可以达到上百万的并发**

**13.1.4 Redis与Memcache对比：**

**Memcache是一个分布式的内存对象缓存系统**

**而redis是可以实现持久存储**

**Memcache是一个LRU的缓存**

**redis支持更多的数据类型**

**Memcache是多线程的**

**redis是单线程的**

**二者性能几乎不相上下，实际上redis会受到硬盘持久化的影响，但是性能仍然保持在与Memcache不相上下，是非常了不起的**

**13.1.5 Redis的优势：**

**丰富的（资料形态）操作**

**String（字符串，包含整数）, List（列表）, Hash（关联数组）, Sets（集合）, Sorted Sets（有序集合）, Bitmaps（位图）, HyperLoglog**

**内建Replication和culster（自身支持复制及集群功能）**

**支持就地更新（in-place update）操作，直接可以在内存中完成更新操作**

**支持持久化（磁盘）**

**避免雪崩效应，万一出现雪崩效应，所有的数据都无法恢复，但redis由于有持久性的数据，可以实现恢复**

**13.1.6 Memcached的优势：**

**多线程**

**善用多核CPU**

**更少的阻塞操作**

**更少的内存开销**

**更少的内存分配压力**

**可能有更少的内存碎片**

**13.2 安装redis**

**13.2.1 源码包安装redis：**

**[root@xuegod63 ~]# tar xf redis-3.2.5.tar.gz -C /usr/local/**

**[root@xuegod63 ~]# cd /usr/local/redis-3.2.5**

**[root@xuegod63 redis-3.2.5]# make**

**[root@xuegod63 redis-3.2.5]# make install**

**redis执行文件位于/PATH/TO/INSTALL\_DIR/src**

**[root@xuegod63 redis-3.2.5]# vim /usr/lib/systemd/system/redis.service //手动添加服务启动文件，写上以下内容**

**[Unit]**

**Description=Redis persistent key-value database**

**After=network.target**

**[Service]**

**ExecStart=/usr/local/redis-3.2.5/src/redis-server /usr/local/redis-3.2.5/redis.conf --supervised systemd**

**ExecStop=/usr/libexec/redis-shutdown**

**Type=notify**

**User=redis**

**Group=redis**

**RuntimeDirectory=redis**

**RuntimeDirectoryMode=0755**

**[Install]**

**WantedBy=multi-user.target**

**# useradd -r redis -s /sbin/nologin //创建redis用户**

**# systemctl daemon-reload //重新加加载Unit文件**

**# systemctl start redis //启动redis**

**上传redis-shutdown脚本到服务器上**

**[root@xuegod63 redis-3.2.5]# vim /usr/libexec/redis-shutdown**

**修改：**

**REDIS\_CLI=/usr/bin/redis-cli**

**改为：**

**REDIS\_CLI=/usr/local/redis-3.2.5/src/redis-cli**

**# chmod +x /usr/libexec/redis-shutdown**

**# systemctl stop redis**

**13.2.2 使用RPM包安装redis：RHEL7**

**上传redis的PRM包到主机**

**[root@xuegod63 ~]# yum -y localinstall redis-4.0.6-1.el7.remi.x86\_64.rpm**

**注意：使用RPM包安装redis会依赖jemalloc程序包，jemalloc是内存分配工具，此包在epel源中**

**[root@xuegod63 ~]# systemctl start redis //启动redis**

**13.3 redis基本操作**

**13.3.1 redis配置文件：**

**[root@xuegod63 ~]# vim /etc/redis.conf**

**daemonize no //表示redis并不会运行成为一个守护进程，如果需要运行成为一个守护进程，则把no，改为yes即可，如果使用服务脚本启动，即使daemonize为no，也会运行为一个守护进程**

**port 6379 //监听端口：6379/tcp**

**tcp-backlog 511 //指定tcp-backlog的长度**

**说明：任何的tcp服务都有可能使用到tcp-backlog功能，backlog是一个等待队列，比如：redis的并发很高时，redis有可能运行不过来时，就连接本地缓存等队列都满了以后，就会使用额外的存储地方，把新来的请求暂存下来，而这个位置则称为backlog**

**bind 127.0.0.1 //监听的地址，默认监听在127.0.0.1地址上，可以指定为0.0.0.0地址，或某个特定的地址，或可以指定多个，使用空格分隔即可**

**# unixsocket /tmp/redis.sock //指定使用sock文件通信及sock文件位置，如果服务端和客户都在同一台主机上，建议打开此项，基于sock方式通信可以直接在内存中交换，数据不用再经过TCP/TP协议栈进行封装、拆封**

**# unixsocketperm 700 //定义sock文件的访问权限**

**timeout 0 //表示当客户端连接成功后，空闲（非活跃、或没有任何数据交互）多长时间则连接超时，0表示不启用此功能**

**tcp-keepalive 0 //定义是否启用tcp-keepalive功能**

**loglevel notice //定义日志级别**

**logfile /var/log/redis/redis.log //定义日志文件**

**databases 16 //定义redis默认有多少个databases，但是在分布式中，只能使用一个**

**#### SNAPSHOTTING #### //定义RDB的持久化相关**

**save <seconds> <changes> //使用save指令，并指定每隔多少秒，如果发生多大变化，进行存储**

**示例：**

**save 900 1 //表示在900秒（15分钟内），如果至少有1个键发生改变，则做一次快照（持久化）**

**save 300 10 //表示在300秒（5分钟内），如果至少有10个键发生改变，则做一次快照（持久化）**

**save 60 10000 //表示在60秒（1分钟内），如果至少有10000个键发生改变，则做一次快照（持久化）**

**save "" //如果redis中的数据不需做持久化，只是作为缓存，则可以使用此方式关闭持久化功能**

**######## REPLICATION ####### //配置主从相关**

**# slaveof <masterip> <masterport> //此项不启用时，则为主，如果启动则为从，但是需要指明主服务器的IP，端口**

**# masterauth <master-password> //如果主服务设置了密码认证，那么从的则需要启用此项并指明主的认证密码**

**slave-read-only yes //定义从服务对主服务是否为只读（仅复制）**

**##### LIMITS ##### //定义与连接和资源限制相关的配置**

**# maxclients 10000 //定义最大连接限制（并发数）**

**# maxmemory <bytes> //定义使用主机上的最大内存，默认此项关闭，表示最大将使用主机上的最大可用内存**

**###### APPEND ONLY MODE ####### //定义AOF的持久化功能相关配置，一旦有某一个键发生变化，将修改键的命令附加到命令列表的文件中，类似于MySQL二进制日志**

**appendonly no //定义是否开启此功能，no表示关闭，yes表示开启**

**说明：RDB和AOF两种持久功能可以同时启用，两者不影响**

**13.3.2 登录redis：**

**[root@xuegod63 ~]# redis-cli -h**

**选项：**

**-h <hostname> 指定主机IP**

**-p <port> 指定端口socket文件进行通信**

**-s <socket> 指定socket文件，如果客户端和服务端都在同一台主机，可以指定socket文件进行通信**

**-a <password> 指定认证密码**

**-r <repeat> 连接成功后指定运行的命令N次**

**-i <interval> 连接成功后每个命令执行完成等待时间，使用-i选项指定**

**-n <db>**

**[root@xuegod63 ~]# redis-cli -h 192.168.1.63 //连接redis，默认不启用密码认证**

**或：**

**[root@xuegod63 ~]# redis-cli //使用redis-cli直接连接，默认连接是127.0.0.1 IP**

**127.0.0.1:6379> exit //退出连接**

**13.3.3 redis获取帮助：**

**127.0.0.1:6379> help //获取使用帮助**

**说明：redis的help命令非常强大，因为redis支持众多的数据结构，每一种数据结构当中都支持N种操作，因此需要使用 help @group方式来获取某一种数据结构所支持的操作**

**例：获取字符串组所支持有那些操作**

**127.0.0.1:6379> help @string**

**127.0.0.1:6379> help APPEND //获取单个命令的使用方法**

**APPEND key value //命令方法**

**summary: Append a value to a key**

**since: 2.0.0 //说明此命令在哪个版本中引入的**

**group: string //该命令所属哪一个组**

**查看都有哪些组：**

**127.0.0.1:6379> help TAB键，每敲一次轮换一个，带有@则为一个组，不带@则为命令使用**

**切换库（名称空间）：**

**127.0.0.1:6379> select 1 //表示切换到1号库中，默认为0号库，共16个，0-15**

**OK**

**127.0.0.1:6379[1]>**

**13.3.4 键的遵循：**

**可以使用ASCII字符**

**键的长度不要过长，键的长度越长则消耗的空间越多**

**在同一个库中（名称空间），键的名称不得重复，如果复制键的名称，实际上是修改键中的值**

**在不同的库中（名称空间），键的同一个名称可以重复**

**键可以实现自动过期**

**13.3.5 Strings的操作：**

**127.0.0.1:6379> help set**

**SET key value [EX seconds] [PX milliseconds] [NX|XX] //命令 键 值 [EX 过期时间，单位秒]**

**summary: Set the string value of a key**

**since: 1.0.0**

**group: string**

**NX：如果一个键不存在，才创建并设定值，否则不允许设定**

**XX：如果一个键存在则设置建的值，如果不存在则不创建并不设置其值**

**例：**

**127.0.0.1:6379> set cjk lzll**

**OK**

**127.0.0.1:6379> set cjk aaa NX**

**(nil) //反回提示一个没能执行的操作**

**127.0.0.1:6379> get cjk**

**"lzll"**

**127.0.0.1:6379> set foo abc XX**

**(nil)**

**定义一个键并设置过期时间为60秒**

**127.0.0.1:6379> set fda abc EX 60**

**OK**

**获取键中的值：**

**127.0.0.1:6379> help get**

**GET key**

**summary: Get the value of a key**

**since: 1.0.0**

**group: string**

**例：**

**127.0.0.1:6379> get cjk**

**"lzll"**

**添加键中的值（在原有键中附加值的内容）：**

**127.0.0.1:6379> append cjk fda**

**(integer) 7**

**127.0.0.1:6379> get cjk**

**"lzllfda"**

**获取指定键中的值的字符串的长度：**

**127.0.0.1:6379> strlen cjk**

**(integer) 7**

**定义整数值：**

**127.0.0.1:6379> set fda 0 //整数值为0**

**OK**

**增加键中的整数值：**

**127.0.0.1:6379> incr fda**

**(integer) 1**

**127.0.0.1:6379> incr fda**

**(integer) 2**

**127.0.0.1:6379> incr fda**

**(integer) 3**

**127.0.0.1:6379> incr fda**

**(integer) 4**

**127.0.0.1:6379> get fda**

**"4"**

**注：incr命令只能对整数使用**

**删除键：**

**127.0.0.1:6379> del fda**

**(integer) 1**

**127.0.0.1:6379> get fda**

**(nil)**

**13.3.6 列表的操作：**

**键指向一个列表，而列表可以理解为是一个字符串的容器，列表是有众多元素组成的集合，可以在键所指向的列表中附加一个值**

**LPUSH //在键所指向的列表前面插入一个值（左边加入）**

**RPUSH //在键所指向的列表后面附加一个值（右边加入）**

**LPOP //在键所指向的列表前面弹出一个值（左边弹出）**

**RPOP //在键所指向的列表后面弹出一个值（右边弹出）**

**LINDEX //根据索引获取值，指明索引位置进行获取对应的值**

**LSET //用于修改指定索引的值为指定的值**

**例：**

**127.0.0.1:6379> help @list**

**LSET key index value**

**summary: Set the value of an element in a list by its index**

**since: 1.0.0**

**指定一个新的列表，在帮助中并没产明哪个命令用于创建一个新的列表，实际上创建一个新的列表使用LPUSH或RPUSH都可以**

**例：**

**127.0.0.1:6379> lpush ll cjk //ll为列表名称，cjk为值（索引）**

**(integer) 1**

**获取列表中的值：需要指明索引位置进行获取对应的值**

**127.0.0.1:6379> lindex ll 0 //第一个索引则为0**

**"cjk"**

**在原有的列表中的左侧加入一个值：**

**127.0.0.1:6379> lpush ll fda**

**(integer) 2**

**127.0.0.1:6379> lindex ll 0**

**"fda"**

**127.0.0.1:6379> lindex ll 1**

**"cjk"**

**在原有的列表中的右侧加入一个值**

**127.0.0.1:6379> rpush ll lzll**

**(integer) 3**

**127.0.0.1:6379> lindex ll 2**

**"lzll"**

**127.0.0.1:6379> lindex ll 1**

**"cjk"**

**127.0.0.1:6379> lindex ll 0**

**"fda"**

**修改一个已有的列表中的值：**

**127.0.0.1:6379> lset ll 0 abc**

**OK**

**127.0.0.1:6379> lindex ll 0**

**"abc"**

**查看列表中的值的数量**

**127.0.0.1:6379> llen ll**

**(integer) 3**

**在已有的列表中右侧弹出（删除）一个值**

**127.0.0.1:6379> rpop ll**

**"lzll"**

**在已有的列表中左侧弹出（删除）一个值**

**127.0.0.1:6379> lpop ll**

**"abc"**

**127.0.0.1:6379> lpop ll**

**"cjk"**

**127.0.0.1:6379> lpop ll**

**(nil)**

**13.3.7 认证实现方法：**

**(1) redis.conf**

**# requirepass foobared //启用此项，并指定密码即可**

**requirepass PASSWORD**

**例：**

**# vim /etc/redis.conf**

**requirepass kill**

**(1) redis.conf**

**# requirepass foobared //启用此项，并指定密码即可**

**requirepass PASSWORD**

**例：**

**# vim /etc/redis.conf**

**requirepass kill**

**# redis-cli**

**127.0.0.1:6379> select 1**

**(error) NOAUTH Authentication required.**

**127.0.0.1:6379> auth kill**

**OK**

**127.0.0.1:6379> select 1**

**OK**

**127.0.0.1:6379[1]>**

**13.3.8 清空数据库：**

**FLUSHDB：删除当前选择的数据库所有key**

**FLUSHALL：清空所有库**

**127.0.0.1:6379> flushdb**

**OK**

**13.4 redis持久化**

**13.4.1 持久化概述：**

**默认情况下，redis工作时所有数据集都是存储于内存中的，不论是否有磁盘上的持久化数据，都是工作于内存当中，redis本身就是一个内存的数据库，把所有数据库相关的存储都存储在内存中，如果redis崩溃或断电导致所有数据丢失，所以redis提供了持久化功能来保证数据的可靠性，redis持久化有两种实现，RDB和AOF**

**13.4.2 RDB: 存储为二进制格式的数据文件，默认启动的持久化机制；按事先定制的策略，周期性地将数据保存至磁盘，使用save命令即可设定周期和策略即可；数据文件默认为dump.rdb，客户端连接服务器以后可以用去使用save命令进行保存数据至磁盘**

**保存快照有两种方式：**

**1、客户端也可显式使用SAVE或BGSAVE命令启动快照保存机制；**

1. **借助于配置文件所定义的save和策略进行保存：**

**SAVE: 是同步保存，在客户端使用save保存快照时，是在redis主线程中保存快照；因为redis的主线程是用于处理请求的，所以此时会阻塞所有客户端请求，每次的保存快照都是把内存中的数据完整的保存一份，并非是增量的，如果内存中的数据比较大，而还有大量的写操作请求时，此方式会引起大量的I/O，会导致redis性能下降**

**BGSAVE：异步方式，将立即返回结果，但自动在后台保持操作，所以BGSAVE命令启动以后，前台不会被占用，客户端的请求是不会被阻塞（主进程不会被阻塞）**

**如果是在配置文件中定义的save，那么redis在持久化的时候，则会开启另外的进程去处理，不会阻塞redis的主进程**

**redis的RDB持久化不足之处则是，一旦数据出现问题，由于RDB的数据不是最新的，所以基于RDB恢复过来的数据一定会有一部分数据丢失，也就是RDB保存之后的修改的数据会丢失**

**13.4.3 AOF：Append Only File，有着更好的持久化能力的解决方案，AOF类似于MySQL的二进制日志，记录每一次redis的写操作命令，以顺序IO方式附加在指定文件的尾部，是使用追加方式实现的，这也叫做一种附加日志类型的持久化机制，由于每一次的操作都记录，则会随着时间长而增大文件的容量，并且有些记录的命令是多余的，AOF不像RDB，RDB是保存数据集的本身**

**但是redis进程能够自动的去扫描这个对应的AOF文件，把其中一些冗余的操作给合并一个，以实现将来一次性把数据恢复，也就是说redis能够合并重写AOF的持久化文件，由BGREWRITEAOF命令来实现，BGREWRITEAOF命令是工作于后台的重写AOF文件的命令，重写后redis将会以快照的方式将内存中的数据以命令的方式保存在临时文件中，最后替换原来的文件，重写AOF文件方式，并没有读取旧AOF文件，而是直接将当前内存中的所有数据直接生成一个类似于MySQL二进日志命令一样的操作，例：set cjk 0 ，incr cjk ... 1000 ，则会替换为set cjk 1000 些命令放到重写文件中，如果此过程完成，那么原有的AOF将被删除**

**BGREWRITEAOF：AOF文件重写；**

**不会读取正在使用AOF文件，而通过将内存中的数据，为内存中的所有数据生成一个命令集，以命令的方式保存到临时文件中，完成之后替换原来的AOF文件；所以AOF文件是通过重写将其变小**

**13.4.4 配置文件中的与RDB相关的参数：**

**stop-writes-on-bgsave-error yes //在进行快照备份时，一旦发生错误的话是否停止写操作**

**rdbcompression yes //RDB文件是否使用压缩，压缩会消耗CPU**

**rdbchecksum yes //是否对RDB文件做校验码检测，此项定义在redis启动时加载RDB文件是否对文件检查校验码，在redis生成RDB文件是会生成校验信息，在redis再次启动或装载RDB文件时，是否检测校验信息，如果检测的情况下会消耗时间，会导致redis启动时慢，但是能够判断RDB文件是否产生错误**

**dbfilename dump.rdb //定义RDB文件的名称**

**dir /var/lib/redis //定义RDB文件存放的目录路径**

**127.0.0.1:6379> config get dir**

**1) "dir"**

**2) "/var/lib/redis"**

**13.4.5 配置文件中的与AOF相关的参数：**

**appendonly no //定义是否开启AOF功能，默认为关闭**

**appendfilename "appendonly.aof" //定义AOF文件**

**appendfsync always //表示每次收到写命令时，立即写到磁盘上的AOF文件，虽然是最好的持久化功能，但是每次有写命令时都会有磁盘的I/O操作，容易影响redis的性能**

**appendfsync everysec //表示每秒钟写一次，不管每秒钟收到多少个写请求都往磁盘中的AOF文件中写一次**

**appendfsync no //表示append功能不会触发写操作，所有的写操作都是提交给OS，由OS自行决定是如何写的**

**no-appendfsync-on-rewrite no //当此项为yes时，表示在重写时，对于新的写操作不做同步，而暂存在内存中**

**auto-aof-rewrite-percentage 100 //表示当前AOF文件的大小是上次重写AOF文件的二倍时，则自动日志重写过程**

**auto-aof-rewrite-min-size 64mb //定义AOF文件重写过程的条件，最少为定义大小则触发重写过程**

**注意：持久本身不能取代备份；还应该制定备份策略，对redis数据库定期进行备份；**

**13.4.6 RDB与AOF同时启用：**

**(1) BGSAVE和BGREWRITEAOF不会同时执行，为了避免对磁盘的I/O影响过大，在某一时刻只允许一者执行；**

**如果BGSAVE在执行当中，而用户手动执行BGREWRITEAOF时，redis会立即返回OK，但是redis不会同时执行，会等BGSAVE执行完成，再执行BGREWRITEAOF**

**(2) 在Redis服务器启动用于恢复数据时，会优先使用AOF**

**13.5 redis主从架构（实现读写分离）**

**13.5.1 复制的工作过程：**

**主库会基于pingcheck方式检查从库是否在线，如果在线则直接同步数据文件至从服务端，从服务端也可以主动发送同步请求到主服务端，主库如果是启动了持久化功能时，会不断的同步数据到磁盘上，主库一旦收到从库的同步请求时，主库会将内存中的数据做快照，然后把数据文件同步给从库，从库得到以后是保存在本地文件中（磁盘），而后则把该文件装载到内存中完成数据重建，链式复制也同步如此，因为主是不区分是真正的主，还是另外一个的从**

**1、启动一slave**

**2、slave会向master发送同步命令，请求主库上的数据，不论从是第一次连接，还是非第一次连接，master此时都会启动一个后台的子进程将数据快照保存在数据文件中，然后把数据文件发送给slave**

**3、slave收到数据文件 以后会保存到本地，而后把文件重载装入内存**

**13.5.2 特点：**

**1、一个Master可以有多个Slave；**

**2、支持链式复制(一个slave也可以是其他的slave的slave)；**

**3、Master以非阻塞方式同步数据至slave(master可以同时处理多个slave的读写请求，salve端在同步数据时也可以使用非阻塞方式)；**

**13.5.3 启动复制功能：**

**1、使用用户端启用：**

**在slave上:**

**> SLAVAOF MASTER\_IP MASTER\_PORT**

**例：**

**127.0.0.1:6379> slaveof 192.168.1.64 6379 //成为从库**

**OK**

**2、使用配置配置（在从库上操作）：**

**# vim /etc/redis.conf**

**# slaveof <masterip> <masterport> //修改此项如下**

**slaveof 192.168.1.63 6379**

**从库上查看：**

**127.0.0.1:6379> info**

**# Replication**

**role:slave**

**master\_host:192.168.1.63**

**master\_port:6379**

**master\_link\_status:up**

**13.5.4 主从相关配置：**

**slave-serve-stale-data yes //表示当主服务器不可以用时，则无法判定数据是否过期，此时从服务器仍然接收到读请求时，yes表示仍然响应（继续使用过期数据）**

**slave-read-only yes //启用slave时，该服务器是否为只读**

**repl-diskless-sync no //是否基于diskless机制进行sync操作，一般情况下如果disk比较慢，网络带宽比较大时，在做复制时，此项可以改为Yes**

**repl-diskless-sync-delay 5 //指定在slave下同步数据到磁盘的延迟时间，默认为5秒，0表示不延迟**

**slave-priority 100 //指定slave优先级，如果有多个slave时，那一个slave将优先被同步**

**# min-slaves-to-write 3 //此项表示在主从复制模式当中，如果给主服务器配置了多个从服务器时，如果在从服务器少于3个时，那么主服务器将拒绝接收写请求，从服务器不能少于该项的指定值，主服务器才能正常接收用户的写请求**

**# min-slaves-max-lag 10 //表示从服务器与主服务器的时差不能够相差于10秒钟以上，否则写操作将拒绝进行**

**注意：如果master使用requirepass开启了认证功能，从服务器要使用masterauth <PASSWORD>来连入服务请求使用此密码进行认证；**

**主从复制的问题：**

**例：有一主三从，如果主服务器离线，那么所有写操作操作则无法执行，为了避免此情况发生，redis引入了sentinel（哨兵）机制**

**13.6 使用sentinel实现主从架构高可用**

**13.6.1 sentinel的工作过程：**

**sentinel安装在另外的主机上，sentinel主机既能监控又能提供配置功能，向sentinel指明主redis服务器即可（仅监控主服务器），sentinel可以从主服务中获取主从架信息，并分辨从节点，sentinel可以监控当前整个主从服务器架构的工作状态，一旦发现master离线的情况，sentinel会从多个从服务器中选择并提升一个从节点成为主节点，当主节点被从节点取代以后，那么IP地址则发生了，客户所连接之前的主节点IP则不无法连接，此时可以向sentinel发起查询请求，sentinel会告知客户端新的主节点的IP，所以sentinel是redis在主从架构中实现高可用的解决方，sentinel为了误判和单点故障，sentinel也应该组织为集群，sentinel多个节点同时监控redis主从架构，一旦有一个sentinel节点发现redis的主节点不在线时，sentinel会与其他的sentinel节点协商其他的sentinel节点是否也为同样发现redis的主节点不在线的情况，如果sentinel的多个点节点都发现redis的主节点都为离线的情况，那么则判定redis主节点为离线状态，以此方式避免误判，同样也避免了单点故障**

**13.6.2 sentinel：**

**用于管理多个redis服务实现HA；**

**监控多个redis服务节点**

**自动故障转移**

**sentinel也是一个分布式系统，可以在一个架构中运行多个sentinel进程，多个进程之间使用“流言协议”接收redis主节点是否离线，并使用“投票协议”是否实现故障转移，选择哪一个redis的从服务器成为主服务器**

**启用sentinel：**

**redis-sentinel可以理解为运行有着特殊代码的redis，redis自身也可以运行为sentinel，sentinel也依赖配置文件，用于保存sentinel不断收集的状态信息**

**程序：**

**redis-sentinel /path/to/file.conf**

**redis-server /path/to/file.conf --sentinel**

**运行sentinel的步骤：**

**(1) 服务器自身初始化（运行redis-server中专用于sentinel功能的代码）；**

**(2) 初始化sentinel状态，根据给定的配置文件，初始化监控的master服务器列表；**

**(3) 创建连向master的连接；**

**13.6.3 专用配置文件：/etc/redis-sentinel.conf**

**(1) # sentinel monitor <master-name> <ip> <redis-port> <quorum> //此项可以出现多次，可以监控多组redis主从架构，此项用于监控主节点 <master-name> 自定义的主节点名称，<ip> 主节点的IP地址，<redis-port>主节点的端口号，<quorum>主节点对应的quorum法定数量，用于定义sentinel的数量，是一个大于值尽量使用奇数，如果sentinel有3个，则指定为2即可，如果有4个，不能够指定为2，避免导致集群分裂，注意，<master-name>为集群名称，可以自定义，如果同时监控有多组redis集群时，<master-name>不能同样**

**(2) sentinel down-after-milliseconds <master-name> <milliseconds> //sentinel连接其他节点超时时间，单位为毫秒（默认为30秒）**

1. **sentinel parallel-syncs <master-name> <numslaves> //提升主服务器时，允许多少个从服务向新的主服务器发起同步请求**
2. **sentinel failover-timeout <master-name> <milliseconds> //故障转移超时时间，在指定时间没能完成则判定为失败，单位为毫秒（默认为180秒）**

**13.6.4 专用命令：**

**SENTINEL masters //列出所有监控的主服务器**

**SENTINEL slaves <master name> //获取指定redis集群的从节点**

**SENTINEL get-master-addr-by-name <master name> //根据指定master的名称获取其IP**

**SENTINEL reset //用于重置，包括名称，清除服务器所有运行状态，故障转移、等等**

**SENTINEL failover <master name> //手动向某一组redis集群发起执行故障转移**

**13.6.5 配置（使用单台主机进行配置）：**

**# mkdir /etc/redis -pv**

**# cp /etc/redis.conf /etc/redis**

**# cd /etc/redis**

**# cp redis.conf redis.conf2**

**# mv redis.conf redis.conf3**

**# mkdir -pv /redis/db{2,3}**

**# chown -R redis. /redis/db\***

**13.6.7 配置redis实例1:**

**# vim /etc/redis.conf**

**修改：**

**# bind 127.0.0.1**

**改为：**

**bind 0.0.0.0**

**修改：**

**daemonize no**

**改为：**

**daemonize yes**

**13.6.8 配置redis实例2：**

**# vim /etc/redis/redis.conf2**

**修改：**

**# bind 127.0.0.1**

**改为：**

**bind 0.0.0.0**

**修改：**

**port 6379**

**改为：**

**port 6380**

**修改：**

**daemonize no**

**改为：**

**daemonize ye**

**修改：**

**pidfile /var/run/redis\_6379.pid**

**改为：**

**pidfile /var/run/redis\_6380.pid**

**修改：**

**logfile /var/log/redis/redis.log**

**改为：**

**logfile /var/log/redis/redis\_6380.log**

**修改：**

**dir /var/lib/redis**

**改为：**

**dir /redis/db2**

**修改：**

**# slaveof <masterip> <masterport>**

**改为：**

**slaveof 192.168.1.63 6379**

**13.6.9 配置redis实例3：**

**# vim /etc/redis/redis.conf3**

**修改：**

**# bind 127.0.0.1**

**改为：**

**bind 0.0.0.0**

**修改：**

**port 6379**

**为：**

**port 6381**

**修改：**

**daemonize no**

**改为：**

**daemonize ye**

**修改：**

**pidfile /var/run/redis\_6379.pid**

**改为：**

**pidfile /var/run/redis\_6381.pid**

**修改：**

**logfile /var/log/redis/redis.log**

**改为：**

**logfile /var/log/redis/redis\_6381.log**

**修改：**

**dir /var/lib/redis**

**改为：**

**dir /redis/db3**

**修改：**

**# slaveof <masterip> <masterport>**

**改为：**

**slaveof 192.168.1.63 6379**

**启动实例1：**

**# systemctl start redis**

**启动实例2：**

**# redis-server /etc/redis/redis.conf2**

**启动实例3：**

**# redis-server /etc/redis/redis.conf3**

**# redis-cli //查看到以下内容则为成功**

**127.0.0.1:6379> info replication**

**# Replication**

**role:master**

**connected\_slaves:2**

**slave0:ip=192.168.1.63,port=6380,state=online,offset=28,lag=1**

**slave1:ip=192.168.1.63,port=6381,state=online,offset=28,lag=1**

**13.6.10 启用sentinel：**

**# vim /etc/redis-sentinel.conf**

**修改：**

**# bind 127.0.0.1 192.168.1.1**

**改为：**

**bind 0.0.0.0**

**修改：**

**sentinel monitor mymaster 127.0.0.1 6379 2**

**改为：**

**sentinel monitor mymaster 192.168.1.63 6379 1**

**# systemctl start redis-sentinel.service**

**登录redis-sentinel：**

**# redis-cli -h 192.168.1.63 -p 26379**

**192.168.1.63:26379> info sentinel**

**# Sentinel**

**sentinel\_masters:1**

**sentinel\_tilt:0**

**sentinel\_running\_scripts:0**

**sentinel\_scripts\_queue\_length:0**

**sentinel\_simulate\_failure\_flags:0**

**master0:name=mymaster,status=ok,address=192.168.1.63:6379,slaves=2,sentinels=1**

**192.168.1.63:26379> sentinel masters //获取主节点及从节点的信息**

**1) 1) "name"**

**2) "mymaster"**

**3) "ip"**

**4) "192.168.1.63"**

**5) "port"**

**6) "6379"**

**7) "runid"**

**8) "824e49626453b61077bf2cfa36c9bb464bbcd595"**

**9) "flags"**

**10) "master"**

**192.168.1.63:26379> sentinel slaves mymaster //获取mymaster集群的从节点信息**

**13.6.11 测试：**

**# systemctl stop redis //关闭主服务**

**192.168.1.63:26379> info sentinel**

**# Sentinel**

**sentinel\_masters:1**

**sentinel\_tilt:0**

**sentinel\_running\_scripts:0**

**sentinel\_scripts\_queue\_length:0**

**sentinel\_simulate\_failure\_flags:0**

**master0:name=mymaster,status=ok,address=192.168.1.63:6381,slaves=2,sentinels=1 //由此可见6381的成为了主**

**启动主服务：**

**# systemctl start redis**

**192.168.1.63:26379> info sentinel**

**# Sentinel**

**sentinel\_masters:1**

**sentinel\_tilt:0**

**sentinel\_running\_scripts:0**

**sentinel\_scripts\_queue\_length:0**

**sentinel\_simulate\_failure\_flags:0**

**master0:name=mymaster,status=ok,address=192.168.1.63:6381,slaves=2,sentinels=1**

**# redis-cli -h 192.168.1.63 -p 6381**

**192.168.1.63:6381> info replication**

**# Replication**

**role:master**

**connected\_slaves:2**

**slave0:ip=192.168.1.63,port=6380,state=online,offset=224567,lag=0**

**slave1:ip=192.168.1.63,port=6379,state=online,offset=224706,lag=0**

**由以上测试可见，主的故障离线后，sentinel重新选了其 一个从的成为了新的主节点，在原来的主节点重新上线后，仍然不会恢复为主节点**

**注意：**

**将来客户端应连接sentinel，向sentinel发请求去寻址，并根据sentinel的反馈，进行连接新的redis主节点，这一点需要使用redis专用客户端来实现。redis客户端会根据sentinel返回的新节点IP进行连接**

**总结：**

**13.1 redis概述**

**13.2 安装redis**

**13.3 redis基本操作**

**13.4 redis持久化**

**13.5 redis主从架构（实现读写分离）**

**13.6 使用sentinel实现主从架构高可用**