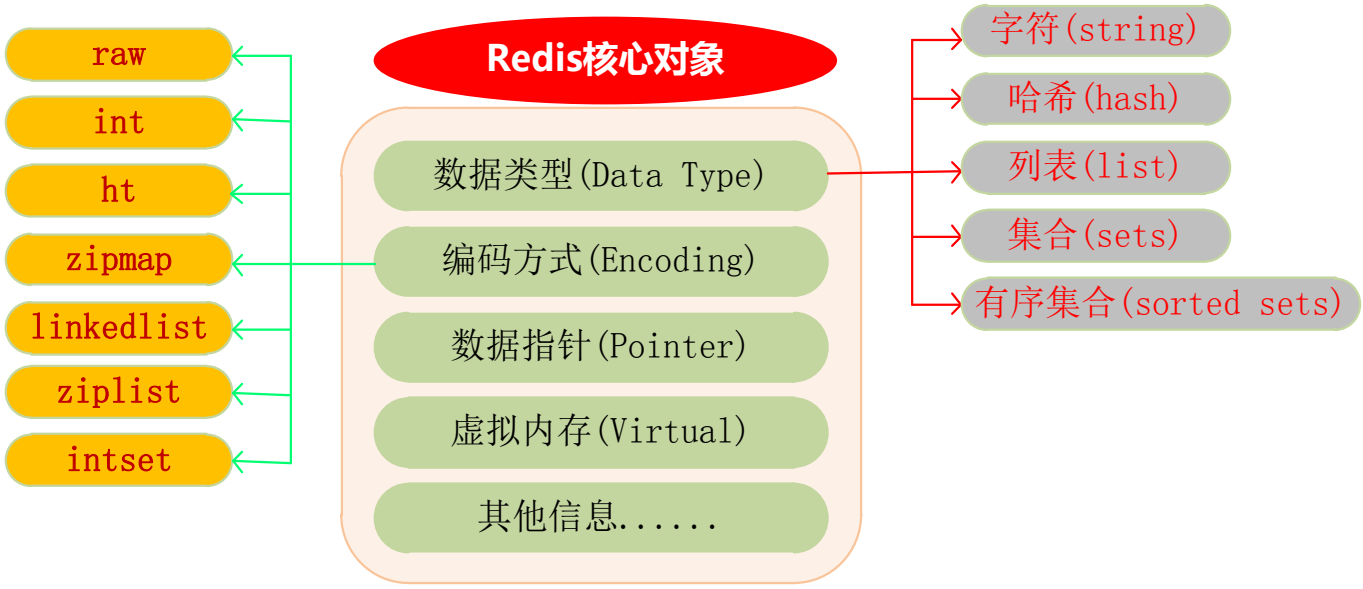
Redis 缓存服务器

一、Redis 简述

1、Redis 定义及简介：

Redis(Remote Dictionary Server)是一个 **Key/Value** 存储系统，和 Memcached 类似，它支持存储的 value 类型相对较多，包括字符串（strings）、哈希（hashes）、列表（lists）、集合（sets）和有序集合（sorted sets）。这些数据类型都支持 push/pop、add/remove 以及取交、并、差集及更丰富的操作，而且这些操作都是原子性的。

redis 支持各种不同方式的排序。它把整个数据库全加载到内存当中进行操作，与 memcached 不同的是通过异步操作定期把内存数据库中的数据保存到硬盘上，并且在此基础上实现了 master-slave 主从同步，具有非常快速的非阻塞首次同步(non-blockingfirst synchronization)、网络断开自动重连等功能。因为是纯内存操作，所以 Redis 的性能非常出色，每秒处理 10 万次以上的读写操作。当前，redis 的应用已经非常广泛了。



从上图可知：Redis 内部使用一个 redisObject 来表示所有的 key 和 value，redisObject 最主要的信息如上图所示：type 代表一个 value 对象具体是何种数据类型，encoding 是不同数据类型在 redis 内部的存储方式，比如：type=string 代表 value 存储的是一个普通字符串，那么对应的 encoding 可以是 raw 或者是 int，如果是 int 则代表实际 redis 内部是按数值类型存储和表示这个字符串的，当然前提是这个字符串本身可以用数值表示，比如:"123" "456"这样的字符串。对于 vm，只有在配置文件中启用了 redis 的虚拟内存功能，此字段才会真正的分配虚拟内存，

2、redis 的数据类型

(1)string 字符类型

string 是最常用的一种数据类型，普通的 key/value 存储都可以归为此类。String 在 redis 内部存储默认就是一个字符串，被 redisObject 所引用，当遇到 incr，decr 等操作时会转成数值型进行计算，此时 redisObject 的 encoding 字段为 int。可以把图片、css 文件、视频文件等保存在 string 中，为了提供网站的运行速度，可以用 string 类型缓存一些静态文件，如：图片、css 文件等

常用命令：set、get、decr、incr、mget。

(2)hash 哈希表

redis 的 Hash 内部存储的 Value 为一个 HashMap，并提供了直接存取这个 Map 成员的接口。。Redis Hash 对应 Value 内部实际就是一个 HashMap，实际这里会有 2 种不同实现，这个 Hash 的成员比较少时 Redis 为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的 HashMap 结构，对应的 value redisObject 的 encoding 为 zipmap,当成员数量增大时会自动转成真正的 HashMap,此时 encoding 为 ht。

常用命令：hget、hset、hgetall。

(3)list 列表

redis list 的实现为一个双向链表，即可以支持从两个方向(前后)查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis 内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

常用命令：lpush、rpush、lpop、rpop、lrange。

(4)set 集合

set 的内部实现是一个 value 永远为 null 的 HashMap，实际就是通过计算 hash 的方式来快速排重的；redis set 对外提供的功能与 list 类似是一个列表的功能，特殊之处在于 set 是可以自动排重以及剔除重复的数据(满足集合性质的互异性)。

常用命令：sadd、spop、smembers、sunion。

(5)sorted set 有序集

redis sorted set 的内部使用 HashMap 和跳跃表(SkipList)来保证数据的存储和有序，HashMap 里放的是成员到 score 的映射，而跳跃表里存放的是所有的成员，排序依据是 HashMap 里存的 score，使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。redis sorted set

的使用场景与 set 类似，区别是 set 不是自动有序的，而 sorted set 可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。

常用命令：zadd、zrange、zrem、zcard。

3、redis 的持久化机制（区别于 memcached）

redis 的持久化方式与传统的数据库的方式有较多差别，redis 是一个支持内存持久化的内存数据库，即 redis 需要经常将内存中的数据同步到硬盘来保证持久化。redis 提供了如下四种数据持久化的方式：

- 定时快照方式(snapshot)
- 基于语句追加文件的方式(append-only-file)
- 虚拟内存(vm，已舍弃)
- Diskstore 方式(已舍弃)

(1)定时快照(snapshotting，内存快照)

该持久化方式是在 Redis 内部的一个定时器事件，*每隔固定时间去检查当前数据发生的改变次数与时间是否满足配置的持久化触发的条件，*如果满足则通过操作系统 fork 调用来创建出一个子进程，这个子进程默认会与父进程共享相同的地址空间，这时就可以通过子进程来遍历整个内存来进行存储操作，而主进程则仍然可以提供服务，当有写入时由操作系统按照内存页(page)为单位来进行 copy-on-write 保证父子进程之间不会互相影响。

快照是默认的持久化方式。这种方式是将 redis 保存在内存中的数据以快照的方式写入二进制文件，默认的文件名为 dump.rdb。可以通过修改配置文件，来设置自动快照。

```
[root@web ~]#vim /usr/local/redis/etc/redis.conf
save 900 1                  ## 每 900 秒，数据更改 1 次，就发起快照保存
save 300 10                 ## 每 300 秒，数据更改 10 次，则发起快照保存
save 60  10000              ## 每 60 秒，数据更改 10000，则发起快照保存
```

主要缺点是定时快照只是代表一段时间内的内存映像，所以系统重启会丢失上次快照与重启之间所有的数据。注意，*内存快照每次都把内存数据完整地写入硬盘，而不是只写入增量数据*。所以如果数据量很大，写入操作比较频繁，就会严重影响性能。

(2)基于语句的日志追加方式(append-only-file，日志追加(aof))

aof 方式实际类似 mysql 的基于语句的 binlog 方式，即对于每一条使 Redis 内存数据发生改变的命令都会追加到 appendonly.aof 文件中，也就是说这个 log 文件就是 Redis 的持久化数据。

操作系统内核的 I/O 接口可能存在缓存，所以日志追加方式不可能立即写入文件，这样就

有可能丢失部分数据。Redis 提供了解决方法，通过修改配置文件，告诉 Redis 应该在什么时候使用 fsync 函数强制操作系统把缓存中的写命令写入磁盘中的日志文件。有以下三种方法：

```
[root@web ~]#vim /usr/local/redis/etc/redis.conf
#appendonly yes              ##启用 aof 持久化方式
#appendfsync always          ##每次收到写命令就立即写入磁盘，性能最差，持久化最好
#appendfsync everysec        ##每秒钟写入磁盘一次，在性能和持久化方面做了很好的折中
#appendfsync no              ##是否写入磁盘完全依赖操作系统，性能最好，持久化没保证
```

aof 的方式的主要缺点是追加 log 文件可能导致体积过大，当系统重启恢复数据时如果是 aof 的方式则加载数据会非常慢，几十 G 的数据可能需要几小时才能加载完，当然这个耗时并不是因为磁盘文件读取速度慢，而是由于读取的所有命令都要在内存中执行一遍。为了压缩日志文件，Redis 提供了 bgrewriteaof 命令。当 Redis 收到此命令时，就使用类似于内存快照的方式将内存中的数据以命令的方式保存到临时文件中，最后替换原来的日志文件。

(3)虚拟内存(Virtual Memory)

由于内存容量的物理限制，可使用虚拟内存把那些不经常访问的数据交换到磁盘上，如此来提高数据库容量，redis 默认没有启用虚拟内存。故需在配置文件中添加一下内容：

```
[root@web ~]#vim /usr/local/redis/etc/redis.conf
vm-enabled yes                      # 开启 vm 功能(默认是没有使用虚拟内存的)
vm-swap-file /tmp/redis.swap        # 交换出来的 value 保存的文件路径
vm-max-memory 268435456             #redis 使用的最大内存，超过该上限后 Redis 开始交换 value 到磁盘文件
vm-page-size 32                     # 每个页面的大小为 32 字节
vm-pages 134217728                  # 最多使用多少个页面，即 swap 文件最多包含多少页面
vm-max-threads 4                    # 用于执行 value 对象换入换出的工作线程数量。0 表示不使用工作线程
```

redis 的虚拟内存只把 value 交换到磁盘中，而 key 依然存储在内存中，目的是让开启虚拟内存的 Redis 和完全使用内存的 Redis 性能基本保持一致。如果由于太多 key 造成的内存不足的问题，Redis 的虚拟内存并不能很好解决，一个解决的办法是将 key/value 合并为 value 进行保存，继而选择合适的 key 存取。推荐将 vm-max-threads 设置为服务器的 CPU 核心数，不是设置越大越好，因为太多的工作线程导致操作系统使用更多时间来切换线程，从而降低了效率。

4、redis 的主从复制

Redis 支持主从复制。redis 的主从复制可以让多个从服务器（slave server）拥有和主服务器（master server）相同的数据库副本。

(1) Redis 主从复制的原理：

redis 主从复制按照如下所示的五个步骤，根据配置文件，slave 自动和 master 建立连接，并

发送连接信号。无论是第一次同步建立的连接还是连接断开后重新建立的连接，master 都会启动一个后台进程，将内存数据以快照方式写入文件中，同时 master 主进程开始收集新的写命令并且缓存起来。

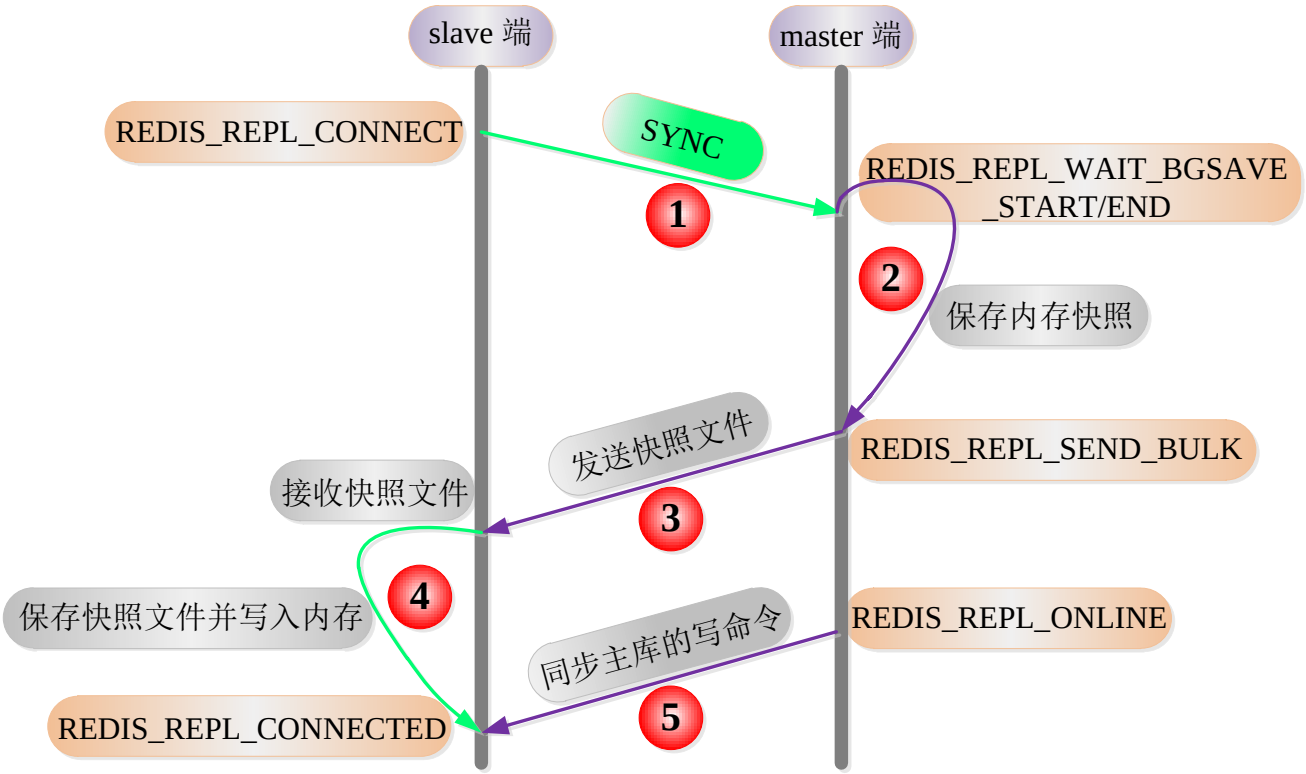
1)Slave 端在配置文件中添加了 slave of 指令，于是 Slave 启动时读取配置文件，主动准备连接 master，初始状态为 REDIS_REPL_CONNECT；

2)Slave 端在定时任务 serverCron(Redis 内部的定时器触发事件)中连接 Master，发送 sync 命令，然后等待 master 发送内存快照文件(rdb)；

3)Master 端收到 sync 命令简单判断是否有正在进行的内存快照子进程，没有则立即开始内存快照，有则等待其结束，当快照完成后会将该 rdb 文件发送给 Slave 端；

4)Slave 端接收 Master 发来的内存快照文件，保存到本地，待接收完成后，清空内存表，重新读取 Master 发来的内存快照文件，重新建立整个内存表数据结构，并将最终状态置位修改为 REDIS_REPL_CONNECTED 状态，Slave 状态机流转完成。

5)Master 端在发送快照文件过程中，接收的任何会改变数据集的命令都会暂时先保存在 Slave 网络连接的发送缓存队列里（list 数据结构），待快照完成后，依次发给 Slave，之后收到的命令相同处理，并将状态置修改为 REDIS_REPL_ONLINE。



当 master 与 slave 断开连接，slave 自动重新建立连接。如果 master 同时收到多个 slave 发来

的同步请求，其只启动一个进程写数据库镜像，然后发送给所有的 slave。如上，redis 完成了整个主从复制的过程。

(2)Redis 主从复制的特点：

- 一个 master 可以拥有多个 slave；
- 多个 slave 除了可以连接同一个 master 外，还可以连接其他的 slave，形成从主；
- 不会阻塞 master，在 slave 同步数据时，master 可以继续处理客户端的请求；
- 提高了系统的伸缩性，比如多个 slave 专门用于客户端的读操作；
- 可在 master 服务器上禁止数据持久化，而只在 slave 服务器上进行数据持久化操作。

(3)配置文件的设置：

redis 主从复制配置文件设置很简单，只需要在 slave 服务器上添加如下的信息：

```
[root@web ~]#vim /usr/local/redis/etc/redis.conf
#slaveof 192.168.1.115 6379          #指定 master（主服务器）的 ip 和端口
#masterauth 密码                     #如果主服务器设置了安全密码，还要进行授权
```

二、Redis 安装配置

1、redis 的安装

官网：http://download.redis.io/releases/

```
[root@web ~]#wget http://download.redis.io/releases/redis-2.8.23.tar.gz
[root@web ~]#tar zxvf redis-2.8.23.tar.gz
[root@web ~]#cd redis-2.8.23
##redis 自带有 MakeFile 文件，指定安装路径(可选)
[root@web ~]#make && make install(或 make PREFIX=/usr/local/redis/ install)
##复制配置文件至安装目录下：
[root@web ~]#mkdir /usr/local/redis/{etc,bin}
[root@web ~]#cp redis-2.8.23/redis.conf /usr/local/redis/etc/
[root@web ~]#cp redis-2.8.23/sentinel.conf /usr/local/redis/etc
[root@web ~]#cd /root/redis-2.8.23/src/
[root@web ~]#mv mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-dump redis-cli
redis-server redis-sentinel /usr/local/redis/bin
##配置 redis 的环境变量：
[root@web ~]#echo "export PATH=$PATH:/usr/local/redis/bin" >/etc/profile.d/redis.sh
```

Redis 无法编译安装报错处理，一般是因为内存空间不足导致引用程序报错，可对参数值进行调整，需要设定内核参数 overcommit_memory，指定内核针对内存分配的策略，其值可以是 0、1、2，分别代表如意含义：

0—>表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。

1—>表示内核允许分配所有的物理内存，而不管当前的内存状态如何。

2—>表示内核允许分配超过所有物理内存和交换空间总和的内存当 Redis 在备份数据的时候，会 fork 出一个子进程，理论上 child 进程所占用的内存和 parent 是一样的，比如 parent 占用的内存为 8G，这个时候也要同样分配 8G 的内存给 child, 如果内存无法负担，往往会造成 redis 服务器的宕机或者 IO 负载过高，效率下降。

这里比较优化的内存分配策略应该设置为 1(表示内核允许分配所有的物理内存，而不管当前的内存状态如何)。

```
##设置方式有两种，需确定当前用户的权限活使用 root 用户修改：
[root@web ~]#echo 1 > /proc/sys/vm/overcommit_memory(默认为 0)        ##或
[root@web ~]#echo "vm.overcommit_memory=1" >> /etc/sysctl.conf  && sysctl -p
```

2、redis 的程序和文件

(1)redis 的二进制程序和各种数据目録

```
redis-sentinel.conf       #redis 的高可用组件 sentinel 的默认配置文件
redis-server      #redis 的主程序，主要的格式 redis [option] <config-file> 。
redis.conf                #redis 的默认配置文件。
redis-cli         #redis 的命令行工具，支持远程连接到 redis 进行基于 command 的操作。
##   Usage: redis-cli [-h <hostname> -p <port> -a <password>]
redis-sentinel
#sentinel 的主程序，实际上可以认为是 redis [option] <config-file> -sentinel 的 alias。
/var/lib/redis                        #redis 的默认库目录。
/var/log/redis.log                    #redis 的默认日志目录。
/var/run/redis.pid                    #redis 运行中产生 pidfile 存放的默认目录。
```

(2)redis 的主配置文件——/etc/redis.conf 内容详解：

```
#### GENERAL ####
daemonize yes                         #以守护进程的方式运行
pidfile "/var/run/redis/redis.pid"#pidfile 位置
port 6379                             #port
tcp-backlog 511                       #tcp 的 backlog 队列长度，backlog 的长度是未建立的 tcp 连接和已经建立的 tcp 连接之和，等待进程从队列中调用建立的连接。tcp 三次握手等待确认 ack 最大的队列数
bind 192.168.88.66 10.0.0.1           #监听的地址，可以指定多个，0.0.0.0 代表本机所有所有地址。
timeout 5000                          #客户端连接的超时时间，单位是毫秒。
loglevel notice                       #日志的记录等级，分为四个等级：debug、verbose、notice、warning
```

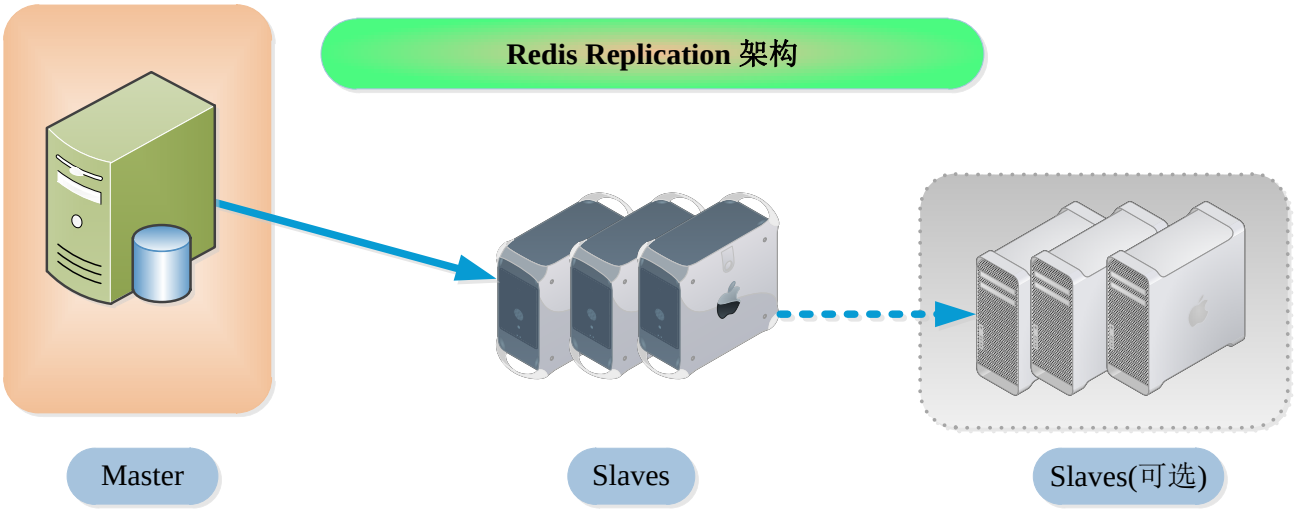
```
logfile "/var/log/redis/redis.log"#日志的存放位置。
databases 16                        #redis 连接的数据库数量，可以在 cli 中通过 select #来切换。

## SNAPSHOTTING ## #在 RDB(内存快照)方式下，数据被定期存储到磁盘的策略。
save 900 1                          #在 900 秒内，如果 1 个 key 发生改变就写一次磁盘；
save 300 10                         #在 300 秒内，如果 10 个 key 发生改变就写一次磁盘；
save 60 10000                       #在 60 秒内如果 10000 个 key 发生改变就写一次磁盘。
stop-writes-on-bgsave-error yes     #在 bgsave 时，发生写入操作时会报告错误，在 bgsave 时不允许写入数据。
rdbcompression yes                  #镜像备份时，是否采用压缩。
rdbchecksum yes                     #rdb 文件开启校验。
dbfilename "dump.rdb"               #db 的名称为 dump.rdb
dir "/redis/data1"                  #RDB 的数据目录，在指定非默认目录时，需要修改目录的属主属组为你当前
redis 进程的属主属组，一般为 redis。

# APPEND ONLY MODE ###基于语句追加日志的方式(aof)
appendonly yes                      #开启 AOF 持久化方式。
appendfilename "appendonly.aof"     #AOF 文件名。
appendfsync everysec                #AOF 的持久化策略，有三个值，always 表示只要发生数据操作就执行保存
或者重写 AOF；everysec 表示一秒一次；no 表示关闭；
no-appendfsync-on-rewrite yes       #在重写时将当前 AOF 的日志存储在内存中，防止 AOF 附加操作与重写产生
数据写入堵塞问题，提高了性能却增加了数据的风险性。
auto-aof-rewrite-percentage 100     #每当 AOF 日志是上次重写的一倍时就自动触发重写操作。
auto-aof-rewrite-min-size 64mb
#自动触发重写的最低 AOF 日志大小为 64MB，为了防止在 AOF 数据量较小的情况话频繁发生重写操作。
aof-load-truncated yes
#当 redis 发生奔溃，通过 AOF 恢复时，不执行最后最后那条因为中断而发生问题的语句。

#### Virtual Memory ####
vm-enabled                  #是否开启虚拟内存支持
vm-swap-file                #设置虚拟内存的交换文件路径
vm-max-memory               #设置 redis 能够使用的最大虚拟内存
vm-page-size                #设置虚拟内存的页大小
vm-pages                    #设置交换文件的总的 page 数量
vm-max-threads              #设置 VMIO 同时使用的线程数量

#### LIMITS ####
maxclients 10000            #设置最大的客户端连接数为 10000 条。
maxmemory <bytes>           #限制最大的内存使用量。
### SECURITY ###
```

```
requirepass <password>
#设定认证的密码，如果设定了，在远程 cli 登录，主从配置和 sentinel 时都需要指定对应参数为此 passwd。
#####----------------Master-----------------Slave----------------########
# slaveof  <masterip>  <masterport>     ###如果是 slave 需要写明 master
# masterauth <master-password>      ###master 密码验证(因为 redis 基于内存,作用不大)
#slave-serve-stale-data no          ###如果 master 宕机,slave 是否还正常对外服务 no->停止服务
#slave-read-only yes                ###salve 是否只读(默认只读)
#slave-priority 100   ###slave 优先级别,数字越小优先级别越高,当 master 宕机时优先选取 slave
```

三、Redis 的分布式架构

1、redis 主从复制的架构实现：

前文已经介绍了 redis 主从复制的原理，如下是简单的 redis 主从复制架构图：



(1)redis 主从复制的配置过程

master 的配置和正常安装 redis 无异，主要针对 slave 的配置：

```
##在 slave 上修改配置文件的”##REPLICATION##”字段
[root@web ~]#cat /etc/redis.conf
slaveof 192.168.1.29 6379                  #主节点地址,<host>  <port>。
#masterauth <master-password>              #如果设置了访问认证就需要设定此项。
slave-server-stale-data yes
#当 slave 与 master 连接断开或者 slave 正处于同步状态时，如果 slave 收到请求允许响应，no 表示返回错误。
slave-read-only yes                        #slave 节点是否为只读。
slave-priority 100                         #设定此节点的优先级，是否优先被同步。
```

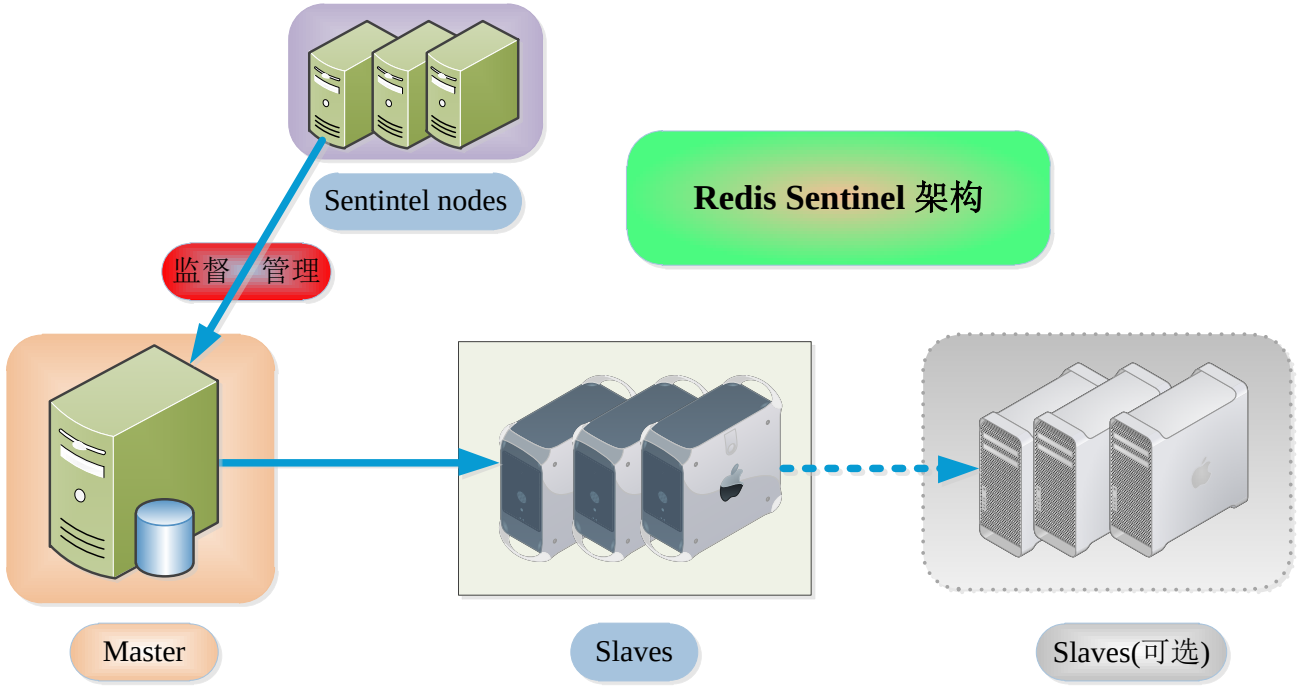
或者在 master 端的命令行(redis-cli)中使用如下命令，连接 slave：

```
[root@web ~]#redis-server /etc/redis.conf
[root@web ~]#redis-cli -h 192.168.88.66 -p 6379 ##连接本机的 IP 和端口
192.168.88.66:6379>slaveof 192.168.88.11 6379   ##slave 的 IP 和端口，显示”OK”则表示主从成功。
```

```
192.168.88.66:6379>info                ##查看详细的连接信息
```

2、redis 的高可用方案 - sentinel

主从复制架构中，如果主节点产生故障，那么整个 redis 系统都会瘫痪，为了解决这个单点，对主节点的高可用就是非常有必要的，redis 自带的 sentinel，就能很好的解决 redis 的高可用问题，通过 ping 机制检测节点状态，并且帮助主节点转移；sentinel 是一个第三方节点来监理而解决高可用的方案，第三方的一台或多台节点共同判定投票主节点的状态并且是否转移。



Sentinel 节点群负责对 Redis 的复制集群做 ping 监控和管理，并且多台 sentinel 节点间进行投票和协调来防止监控错误，当投票确定 master 节点下线时，就会触发转移操作，自动根据算法和优先级(slave-priority 100)来确定新的主节点来提供相应的服务；sentinel nodes 应该为单数个，防止由于仲裁发生集群分裂。

(1)Redis Sentinel 架构的配置过程(更多配置信息参见附件内容)

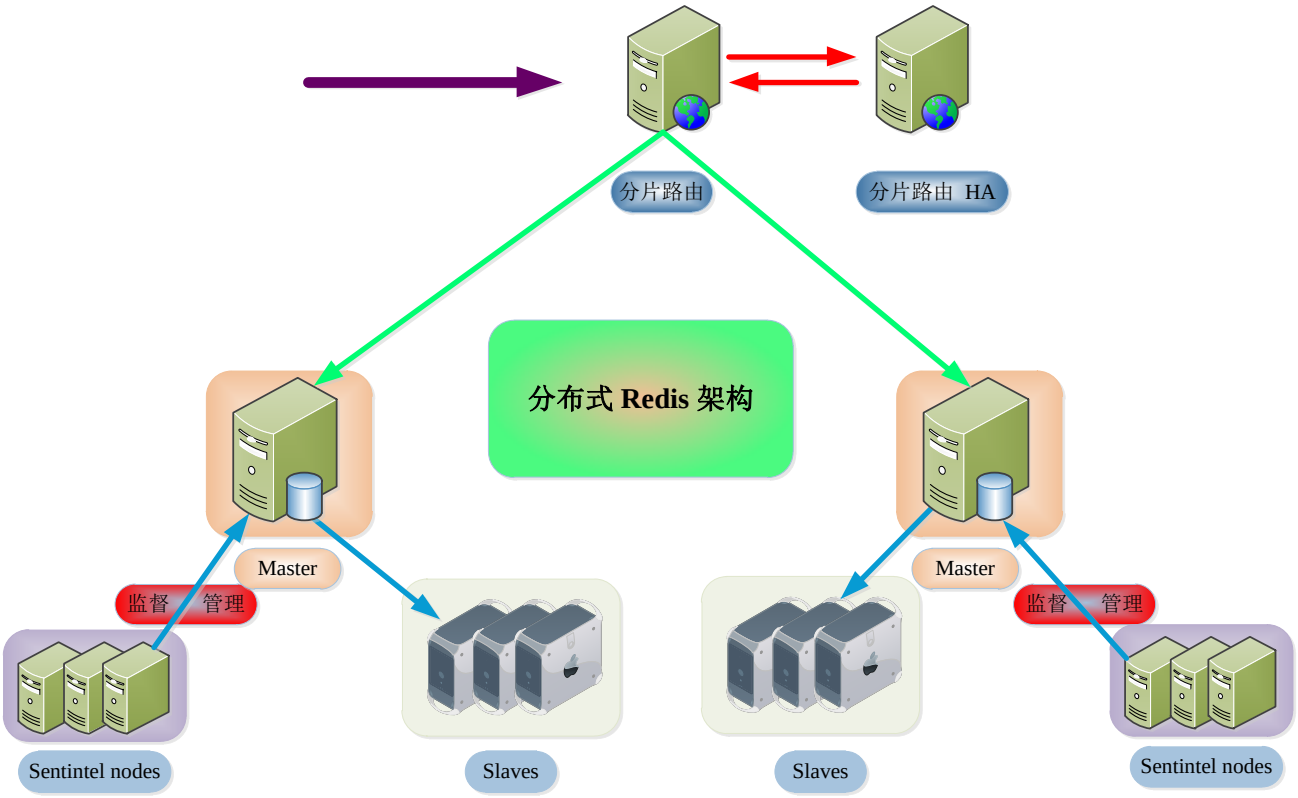
```
[root@web ~]#cat /etc/redis-sentinel.conf
port 26379                              #设定监听端口
#sentinel announce-ip 1.2.3.4           #默认监听在 0.0.0.0 所以此处可以注释。
dir "/tmp"                              #设定数据存储目录
## Usage:  sentinel  moitor  <master-name>  <ip>  <redis-port>  <法定人数 quorum>
sentinel  monitor  mymaster  192.168.1.29 6379  1  #法定人数表示多少台 sentinel 节点认同才可以上线。
sentinel down-after-milliseconds mymaster 5000     #如果联系不到节点 5000 毫秒，我们就认为此节点下线。
sentinel failover-timeout mymaster 60000           #设定转移主节点的目标节点的超时时长。
sentinel auth-pass  <master-name> <password>   #如果 redis 节点启用了 auth，此处也要设置 password。
```

(2) redis-cli 中的 sentinel 组命令

```
[root@web ~]#redis-cli -h 192.168.88.66 -p 6379
192.168.88.66:6379>sentinel master <master-name>          #查看此复制集群的主节点信息。
192.168.88.66:6379>sentinel slaves   <master-name>        #查看此复制集群的从节点信息。
192.168.88.66:6379>sentinel failover  <node-name>         #切换指定的节点为节点为主节点。
更多参数选项参考链接：http://www.178linux.com/2471
```

3、redis 的分布式架构与简述(数据分片)

redis 与 mariadb、mysql 一样，在海量数据的情况下，复制已经无法解决性能压力，这时候我们就需要通过分片机制分割数据，构建分布式系统；如下图是分布式 redis 的架构图



与 mariadb 和 mysql 类似，请求到分片路由节点，之后由路由节点智能的分配到相对的数据节点，并且要考虑到结构的可靠性，同样的需要符合分布式系统的 CAP【*一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）*】和 BASE【*Basically Available(基本可用)、Soft state(软状态)和 Eventually consistent(最终一致性)*】标准；并且还要考虑到数据怎么分片的问题。

4、分布式和集中式 CAP、BASE(补充内容)

4.1 分布式的定义：

分布式系统是一个硬件或软件组件分布在不同的网络计算机上，彼此之间仅仅通过消息传递进行通信和协调的系统。一个标准的分布式系统在没有任何业务逻辑约束的情况下，都会有以下特征：分布性、对等性、并发性、缺乏全局时钟故障总是会发生

4.1.1 分布式环境的各种问题

(1)通信异常：由于网络本身的不可靠性，导致各网络节点之间进行网络通信时，会伴随着不可预知的风险，网络光纤、路由器或是 DNS 等硬件设备或是系统不可用都会导致最终分布式系统无法顺利完成一次网络通信。另外，集群的延时通常会远大于单机操作。

(2)网络分区：网络的异常会导致分布式系统中，只有部分节点能够正常通信，而部分节点间可以互相通信，从而形成了网络分区。

(3)三态：相比于单机的失败或者成功，集群会出现“三态”的概念，即成功、失败与超时。有两种丢消息的情况：由于网络原因，消息没有成功发送到接收方，而是在发送过程就发生了丢失现象。接收方处理后，响应给发送方的过程中，发生消息丢失现象。节点故障：分布式服务器节点宕机或“僵死”现象。

4.2、从 ACID 到 CAP/BASE

4.2.1、ACID(事务的原子性、一致性、隔离性、持久性)

事务(Transaction)是由一系列对系统中数据进行访问和更新的操作锁组成的一个程序执行逻辑单元(Unit)，狭义上的事务特指数据库事务。一方面，当多个应用程序并发访问数据库时，事务可以在这些应用程序之间提供一个隔离方法，以防止彼此的操作互相干扰。

事务(Transaction)具有四个特征，分别是*原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持久性(Durability)，简称事务的 ACID 特性。*

（1）原子性：指事务必须是一个院子的操作序列单元。事务中包含的各项操作在一次执行过程中，只允许出现以下两种状态之一：全部成功执行或全部不执行。任何一项操作失败都将导致整个事务失败，同时其他已经被执行的操作豆浆杯撤销并回滚，只有所有的操作全部成功，整个事务才算是成功完成。

（2）一致性：事务的一致性是指事务的执行不能破坏数据库数据的完整性和一致性，一个事务在执行之前和执行之后，数据库都必须处于一致性状态。

（3）隔离性：是指在并发环境中，并发的事务是相互隔离的，一个事务的执行不能被其他事务干扰。在标准的 SQL 规范中，定义了 4 个事务隔离级别，不同的隔离级别对事务的处理不同，如未授权读取(Read Uncommitted)、授权读取(Read Committed)、可重复读(Repeatable Read,MYSQL 默认采用)和串行化(Serializable)。

（4）持久性：是指事务一旦提交，他对数据库中对应数据的状态变更就应该是永久性的。

4.2.2、分布式事务

分布式事务是指事务的参与者、支持事务的众多服务器、资源服务器以及事务管理器*分别位于分布式系统的不同节点之上*。通常一个分布式事务中会涉及到对多个数据源或业务系统的操作和管理。

4.2.3、CAP 和 BASE 理论的提出

对于一个高访问量、高并发的互联网分布式系统来说，如果我们期望实现一套严格满足 ACID 特性的分布式事务，很可能出现的情况就是在系统的可用性和严格一致性之间出现冲突。因为当我们要求分布式系统具有严格一致性时，很可能就需要牺牲掉系统的可用性。但是，可用性优势一个不允许我们讨价还价的系统属性，对于一致性，则更加是所有消费者对于一个软件系统的刚需。因此，在可用性和一致性之间永远无法存在一个两全其美的方案，于是出现了诸如 CAP 和 BASE 这样的分布式系统经典理论。

4.3、CAP(Consistency、Availability、Partition tolerance)定理

*一个分布式系统不可能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三个基本需求，*最多只能同时满足其中的两项。

需要明确的一点是，对于一个分布式系统而言，分区容错性可以说是一个最基本的要求。因为既然是一个分布式系统，那么分布式系统中的组件必然需要被部署到不同的节点，否则也就无所谓分布式系统了，因此必然出现子网络。而对于分布式系统而言，网络问题又是一个必定会出现的异常情况，因此分区容错性也就成为了一个分布式系统必然要面对和解决的问题。因此系统架构设计师往往需要把经历花在如何根据业务特点在 C 和 A 之间寻求平衡。

4.3、BASE 理论

BASE 是 Basically Available(基本可用)、Soft state(软状态)和 Eventually consistent(最终一致性)三个短语的简写。

总的来说，BASE 理论面向的是大型高可用可扩展的分布式系统，和传统事务的 ACID 特性是相反的，完全不同于 ACID 的强一致性模型，而是提出通过牺牲强一致性来获得可用性，并允许数据在一段时间内是不一致的，但最终达到一致状态。但同时，在实际的分布式场景中，不同业务单元和组件对数据一致性的要求是不同的，因此在具体的分布式系统架构设计中，ACID 特性和 BASE 理论往往会结合在一起使用。

参考链接：http://www.178linux.com/37667(Redis 简介及其分布式架构)

5、PHP 安装 redis 的扩展库：

redis 的各种客户端可在官网下载得到: https://redis.io/clients，其中有基于 php、java、matlab 等语言的各种客户端。

```
[root@web ~]#unzip phpredis-devlop.zip
[root@web ~]#cd phpredis-devlop
[root@web ~]#/usr/local/php/bin/phpize
[root@web ~]#./configure --with-php-config=/usr/local/php/bin/php-config
[root@web ~]#make ## make install
[root@web ~]#echo 'extension="redis.so"' >> /etc/php.d/redis.ini
[root@web ~]#service php-fpm restart
```

附件：

redis 的 startup 文件

```
#!/bin/sh
#
# redis       init file for starting up the redis daemon
#
# chkconfig:   - 20 80
# description: Starts and stops the redis daemon.
# Source function library.
. /etc/rc.d/init.d/functions
name="redis-server"
exec="/usr/local/bin/$name"
pidfile="/var/run/redis/redis.pid"
REDIS_CONFIG="/etc/redis/redis.conf"
[ -e /etc/sysconfig/redis ] && . /etc/sysconfig/redis
lockfile=/var/lock/subsys/redis
start() {
    [ -f $REDIS_CONFIG ] || exit 6
    [ -x $exec ] || exit 5
    echo -n $"Starting $name: "
    daemon --user ${REDIS_USER-redis} "$exec $REDIS_CONFIG"
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}
stop() {
    echo -n $"Stopping $name: "
    killproc -p $pidfile $name
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}
restart() {
    stop
    start
}
reload() {
    false
```

```
}
rh_status() {
    status -p $pidfile $name
}
rh_status_q() {
    rh_status >/dev/null 2>&1
}
case "$1" in
    start)
        rh_status_q && exit 0
        $1;;
    stop)
        rh_status_q || exit 0
        $1;;
    restart)
        $1;;
    reload)
        rh_status_q || exit 7
        $1;;
    force-reload)
        force_reload;;
    status)
        rh_status;;
    condrestart|try-restart)
        rh_status_q || exit 0
        restart;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|try-restart}"
        exit 2;;
esac
exit $?
```

redis 之/etc/redis-sentinel.conf 配置项解释：

```
1.     ##sentinel 实例之间的通讯端口
2.     ##redis-0
3.     port 26379
4.     ##sentinel 需要监控的 master 信息：<mastername> <masterIP> <masterPort> <quorum>
5.     ##<quorum>应该小于集群中 slave 的个数,只有当至少<quorum>个 sentinel 实例提交"master 失效"
6.     ##才会认为 master 为 O_DWON("客观"失效)
7.     sentinel monitor def_master 127.0.0.1 6379 2
8.
9.     sentinel auth-pass def_master 012_345^678-90
10.
11.   ##master 被当前 sentinel 实例认定为"失效"的间隔时间
12.   ##如果当前 sentinel 与 master 直接的通讯中，在指定时间内没有响应或者响应错误代码，那么
13.   ##当前 sentinel 就认为 master 失效(SDOWN，"主观"失效)
14.   ##<mastername> <millseconds>
15.   ##默认为 30 秒
16.   sentinel down-after-milliseconds def_master 30000
17.
18.   ##当前 sentinel 实例是否允许实施"failover"(故障转移)
19.   ##no 表示当前 sentinel 为"观察者"(只参与"投票".不参与实施 failover)，
20.   ##全局中至少有一个为 yes
21.   sentinel can-failover def_master yes
22.
23.   ##当新 master 产生时，同时进行"slaveof"到新 master 并进行"SYNC"的 slave 个数。
24.   ##默认为 1,建议保持默认值
```

```
25.   ##在 salve 执行 salveof 与同步时，将会终止客户端请求。
26.   ##此值较大，意味着"集群"终止客户端请求的时间总和和较大。
27.   ##此值较小,意味着"集群"在故障转移期间，多个 salve 向客户端提供服务时仍然使用旧数据。
28.   sentinel parallel-syncs def_master 1
29.
30.   ##failover 过期时间，当 failover 开始后，在此时间内仍然没有触发任何 failover 操作，
31.   ##当前 sentinel 将会认为此次 failoer 失败。
32.   sentinel failover-timeout def_master 900000
33.
34.   ##当 failover 时，可以指定一个"通知"脚本用来告知系统管理员，当前集群的情况。
35.   ##脚本被允许执行的最大时间为 60 秒，如果超时，脚本将会被终止(KILL)
36.   ##脚本执行的结果：
37.   ## 1    -> 稍后重试，最大重试次数为 10；
38.   ## 2    -> 执行结束，无需重试
39.   ##sentinel notification-script mymaster /var/redis/notify.sh
40.   ##failover 之后重配置客户端，执行脚本时会传递大量参数，请参考相关文档
41.   # sentinel client-reconfig-script <master-name> <script-path>
```