Redis 笔记

郑华

2018 年 12 月 1 日

第一章　Redis 简介

1.1　应用场景

- 缓存
- 聊天室，秒杀，任务队列
- 需要精确设定过期时间的应用
- 应用排行榜. 网站统计
- Pub/Sub 构建实时消息系统: 订阅、消息
- 数据存储 (add,del,update,select) 定期持久化到硬盘中
- 分布式集群架构中的 session 分离

1.2　安装与启动

```
// 安装
$ wget http://download.redis.io/releases/redis-2.8.17.tar.gz
$ tar xzf redis-2.8.17.tar.gz
$ cd redis-2.8.17
$ make


//启动redis服务.
$ cd src
$ ./redis-server

//使用默认配置启动redis 服务
./redis-server redis.conf

//启动redis服务进程后，就可以使用测试客户端程序redis-cli和redis服务交互了
./redis-cli
```

1.3 配置

daemonize 如果需要在后台运行,把该项改为yes
pidfile 配置多个pid的地址 默认在/var/run/redis.pid
bind 绑定ip,设置后只接受来自该ip的请求
port 监听端口,默认为6379
timeout 设置客户端连接时的超时时间,单位为秒
loglevel 分为4级,debug、verbose、notice、warning
logfile 配置log文件地址
databases 设置数据库的个数,默认使用的数据库为0
save 设置redis进行数据库镜像的频率
rdbcompression 在进行镜像备份时,是否进行压缩
Dbfilename 镜像备份文件的文件名
Dir 数据库镜像备份的文件放置路径
Slaveof 设置数据库为其他数据库的从数据库
Masterauth 主数据库连接需要的密码验证
Requirepass 设置登录时需要使用的密码
Maxclients 限制同时连接的客户数量
Maxmemory 设置redis能够使用的最大内存
Appendonly 开启append only模式

第二章　基本操作

redis 命令**不区分大小写**，所以`get var` 和`GET var` 是等价的

2.1　选库

使用 **Select** 命令用于*切换到***指定的数据库**，数据库索引号 index 用数字值指定，以 0 作为起始索引值。

flushdb # 清空数据库。

2.2　Key

表 2.1: 常用命令

命令	含义
keys [pattern]	返回相应的的 key
set key value	设定一个 key-value
get key	获取一个 key 的值
dump key	序列化给定 key, 并返回序列化的值
randomkey	返回随机的 key
exists key	判断 key 是否存在
type key	返回 key 存储的类型
del key	删除 key
expire key seconds	给 key 设置过期时间
persist key	移除 key 的过期时间，key 将持久保存

第三章　数据类型

3.1　string

二进制安全的。意思是 redis 的 string 可以包含任何数据。比如 jpg 图片或者序列化的对象
一个键最大能存储512MB

- `set key value [ex 秒数]|[px 毫秒数]  [nx]|[xx]` : 设置键值
  - `nx` 表示 key 不存在时执行操作
  - `xx` 表示 key 存在时执行操作
  - `ex` 表示设置过期时间
  - `px` 表示设置持续时间，`ex` 与`px` 不能同时设置
- `mset key1 v1 key2 v2 ...` : 一次性设置多个键值对 (multi set)
- `get key` 获取 key 的值
- `mget key1 key2 ...` : 一次性获取多个键的值
- `setrange key offset value` : 把字符串的`offset` 偏移字节改为`value`

```
set greet hello
get greet --> hello
setrange greet 2 x
get greet --> hexlo

setrange greet 2 ??
get greet --> he??o

// 当偏移量大于字符长度时，多余部分将自动以0x00 填充
setrange greet 6 !
get greet --> he??o0x00!
```

- `append key value` : 把 value 追加到 key 的原值上
- `getrange key start stop` : 获取字符串中 `start, stop` 范围的值
  - 左闭右闭区间，从 0 开始

– 负数表示倒数第多少

- `getset key newValue`：获取并返回旧值，并设置新值
- `incr|decr key`：增 1 或减 1,不存在的 key 当成 0 再 incr 返回。与之对应的有`incrby key num`
- `getbit key offset`：获取值的二进制的对应位上的值，从高位开始
- `setbit key offset value`：设置对应 2 进制位上的值，offset 最长能达到`2^32-1` 位
- `bitop op destKey key1 [key2 key3 ...]`：对 [key1 key2..] 做 op，并将结果保存到 destKey 中，op 可以为一下几种`AND OR NOT XOR`

3.2 list -栈或队列

按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

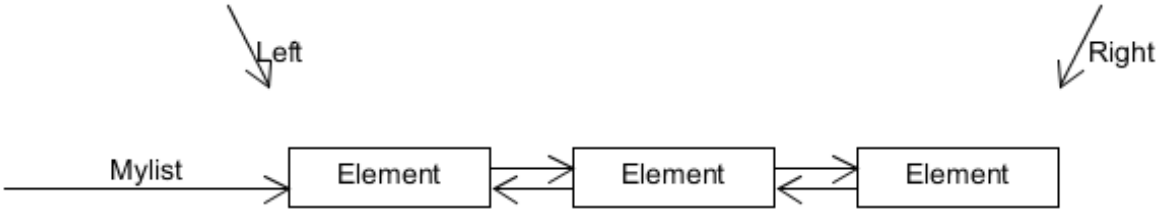

Fig 3.1: 链表结构

- **lpush** `mylist "hello"`：从头部插入字符串
- **rpush** `mylist "world"`：从尾部插入字符串
- **lrange** `mylist 0 -1`：获取从 0 到倒数第一个, 左右闭区间，如`[1)hello 2)world]`
- **linsert** `mylist before "hello" "start"`：在指定位置插入，与`before` 对应的有 `after`
- **lset** `mylist 0 "Start"`：设置指定下标的值
- **lrem** `mylist 1 "hello"`：删除 (1 个) 值为 hello 的元素，`n = 0` 全部删除，`n < 0` 从尾部开始删除
- **lpop** `mylist`：弹出开头的元素
- **rpop** `mylist`：弹出结尾的元素
- **index** `mylist 0`：取出下标为 0 的元素值
- **llen** `mylist`：返回表元素的个数
- **ltrim** `mylist 0 2`: 裁剪链表，保留从 0 号元素到 2 号元素之间的链表

3.3 hash

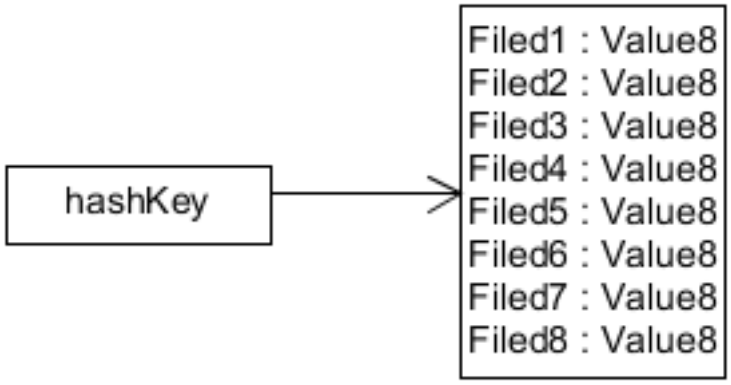

Fig 3.2: hash 结构

- `hset` user:001 name liweijie : 哈希设置 Key 为 user:001，并增加域 name，其键值为 liweijie
- `hsetnx` user:001 age 22 **：** 检测键是否存在。若不存在创建
- `hmset` user:002 name liweijie2 age 26 sex 1 : 同时设置多个键的值
- `hget` user:001 name : 哈希获取 Key user:001 的 name 域的值。
- `hmget` user:001 name age sex **：** 获取多个指定的域键的值。
- `hgetall` user:001 **：** 获取所有当前 key 域的值
- `hincrby` user:001 age -8 **：** 在指定域键上加上给定的值。
- `hexists` user:001 sex **：** 检测指定的域键是否存在。
- `hlen` user:001 : 返回指定哈希的键个数/字段个数
- `hdel` user:001 sex : 删除指定 (user:001) 哈希的指定字段或是键值。
- `hkeys` user:003: 返回哈希里所有 key。

3.4 set

Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。Redis 中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。

- **smembers** myset : 查看 myset 集合中所有元素值。
- **sadd** myset "hello" : 向 mysets 集合中添加一个值 hello

- **srem** myset "hello" : 删除 myset 集合中名称为 hello 的元素。
- **spop** myset : 随机弹出并返回 mysets 中的一个元素。
- **sdiff** myset2 myset3 : 返回 myset2 中的与 myset3 的**差集** (以 myset2 为准)。
- **sdiffstore** myset4 myset2 myset3 : 返回 myset2 中的与 myset3 的**差集**, 并存入 myset4 中去。
- **sinter** myset2 myset3 : 返回 myset2 与 myset3 的交集。
- **sinterstore** myset5 myset2 myset3 : 返回 myset2 与 myset3 的交集, 并存入 myset5 中去。
- **sunion** myset2 myset3 : 求并集
- **sunionstore** myset6 myset2 myset3 : 求并集, 并存入 myset6 中去。
- **smove** myset2 myset3 "three" : 将 myset2 中的 three 移到 myset3 中去。
- **scard** myset2 : 返回元素个数。
- **sismember** myset2 "one" : 判断元素 one 是不是 myset2 集合的 (相当于`is_array()`)。
- **srandmember** myset2 : 随机返回 myset2 集合中的一个元素, 但不删除 (相当于`array_rand()`)。

集合的相关概念

- 并集：以属于 A 或属于 B 的元素为元素的集合成为 A 与 B 的并（集）
- 交集：以属于 A 且属于 B 的元素为元素的集合成为 A 与 B 的交（集）
- 差：以属于 A 而不属于 B 的元素为元素的集合成为 A 与 B 的差（集）

3.5 zset

有序集合和集合一样也是 string 类型元素的集合, **且不允许重复**的成员。

不同的是每个元素都会关联一个 *double* 类型的分数。redis 正是通过分数来为集合中的成员进行从小到大的排序。

有序集合的成员是唯一的, 但分数 (score) 却可以重复。

集合是通过**哈希表**实现的，所以添加，删除，查找的复杂度都是 O(1)。

- **zadd** myZSet 1 zlh ：添加分数为 1，值为 zlh 的 zset 集合，可多添加多组`[socre2 value2]`
- **zcard** myZSet : 输出 zset 的成员个数
- **zrange** mZySet 0 -1 withscores : 查看 Zset 指定范围的成员，withscores 为输出结果带分数,0 为开始，-1(倒数第一) 为结束

- **zrank** mZySet Jim ： 获取 zset 成员的下标位置，如果值不存在返回 null
- **zcount** mySet 1 3 ：输出 分数>=1 and 分数 <=3 的成员个数, 因为分数是可以重复的，所以这个命令是有道理的
- **zrem** myZSet zlh : 删除指定的一个成员或多个成员,[]
- **zscore** myZset zlh : 获取指定值的分数
- **zincrby** myZset 4 tom : 给指定元素的分数进行增减操作，负值为减，正值为加
- **zrangebyscore** myZset : 根据指定分数的范围获取值
  - zrangebyscore myZset 1 5 : 输出分数 >=1 and <=5 的成员值
  - zrangebyscore myZset (1 5 : 输出分数 >1 and <=5 的成员值
  - zrangebyscore myZset 2 5 limit 1 2 : 检索分数为 2 到 5 之间的数据，然后从下标为 1 的数据开始往后输出 2 个数据，包含下标为 1 的数据
  - zrangebyscore myZset -inf +inf limit 2 3 :+inf 表示最后一个成员，-inf 表示第一个成员，意思是：检索所有数据，然后从下标为 2 的数据开始再往后输出 2 个数据
- **zrevrange**，**zrevrangebyscore** : 倒序，从高到底排序输出指定范围的数据
- **zremrangebyscore**，**zremrangebyrank** : 根据坐标，分数范围删除数据
- **zunionzstore** heZset 2 myZset youZset ： 将 myzset 和 youzset 的**并集**添加到 hezset 中。
- **zinterstore** sheZset 2 myZset youZset ： 将 myzset 和 youZset 的交集添加到 sheZset 中。

3.6 事务与锁

类似于 mysql 的 start transaction，可以保证原子性，可以使用 rollback 取消操作。
redis 使用 multi 命令实现
使用 watch 锁，解决多用户竞争
Redis 事务可以一次执行多个命令，并且带有以下两个重要的保证：

- 批量操作在发送 EXEC 命令前被放入队列缓存。
- 收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。
- 在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。

一个事务从开始到执行会经历以下三个阶段：

1. 开始事务。
2. 命令入队。如果中间存在错误，事务执行取消

3. 执行事务。

表 3.1: 事务与锁操作说明

命令	含义
MULTI	标记一个事务块的开始。
EXEC	执行所有事务块内的命令。
DISCARD	取消事务，放弃执行事务块内的所有命令。
UNWATCH	取消 WATCH 命令对所有 key 的监视。
WATCH key [key ...]	监视一个 (或多个) key ，如果在事务执行之前这个 (或这些) key 被其他命令所改动，那么事务将被打断。

第四章　订阅与发布

Observer 观察者模式

4.1　publish 主题消息

声明、发布主题，并提供消息。

4.2　subscribe 主题

收听、监听、订阅某主题

4.3　psubscribe 主题模式

收听某种正则规则的主题

第五章　持久化

5.1　rdb 持久化

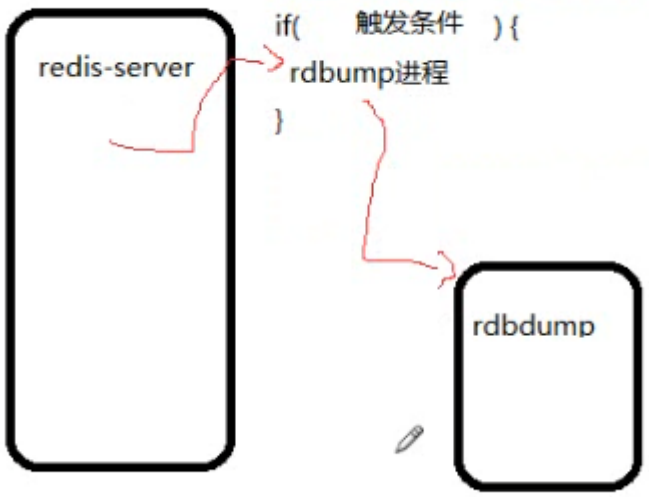

Fig 5.1: 镜像备份原理

SAVE|BGSAVE　用于创建当前数据库的备份。

相关参数

- `save 900 1`：刷新快照到硬盘中，必须满足 900 秒后至少有一个关键字发生变化
- `save 300 10`：必须是 300 秒以后至少 10 个关键字发生变化
- `save 60 10000`：必须是 60 秒以后至少 10000 个关键字发生变化
- `stop-writes-on-bgsave-error yes`：后台存储进程出错时，前端缓存不允许写操作，既改变关键字
- `rdbcompression yes`：使用 LZF 压缩 rdb 文件
- `dbfilename dump.rdb`：设置 rdb 文件名
- `dir ./`：设置备份保存路径

恢复数据　如果需要恢复数据，只需将备份文件 (dump.rdb) **移动到 `redis` 安装目录**并启动服务即可。

获取 redis 目录可以使用 CONFIG 命令:`CONFIG GET dir`

特点　恢复数据快，镜像恢复，但是可能存在数据丢失 (当断电时没有触发备份条件)。

5.2　aof 持久化

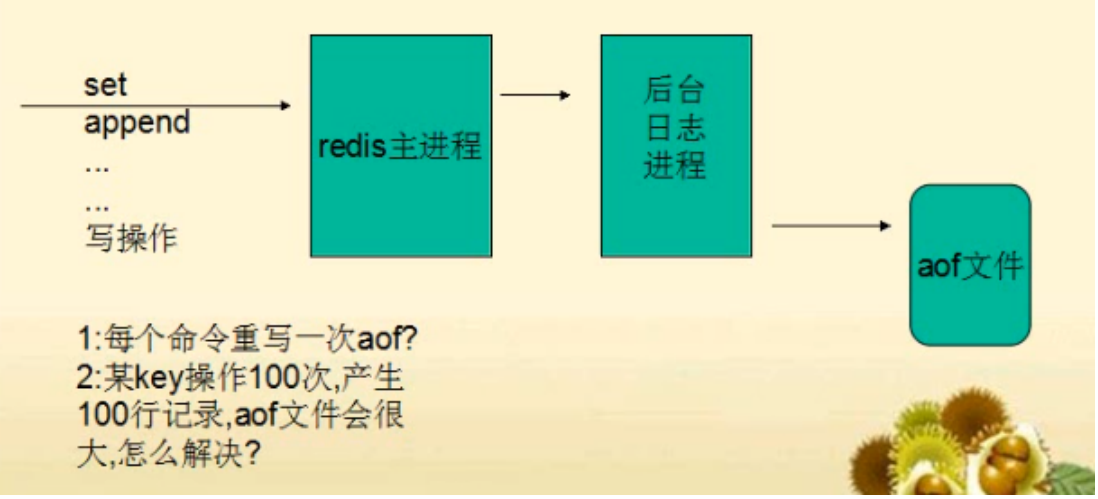

Fig 5.2: aof 备份原理

Aof 配置

- `appendonly on` : 是否打开 aof 日志功能
- `appendfsync always` : 每 1 个命令，都立即同步到 aof，安全，速度慢
- `appendfsync every sec` : 折衷方案，每秒写一次
- `appendfsync no` : 写入工作交给操作系统，由操作系统判断缓冲区大小统一写入到 aof, 同步频率低，速度快
- `no-appendfsync-on-rewrite  yes` : 正在导出 rdb 快照的过程中，要不要停止同步 aof
- `auto-aof-rewrite-percentage 100` :aof 文件大小比上次重写时的大小，增长率 100% 时，重写
- `auto-aof-rewrite-min-size-64mb` :aof 文件至少超过 64MB 时重写

Aof 重写 BGREWRITE　aof 重写是指把内存中的数据，逆化成命令，简化命令个数，写入到 aof 日志里，以解决 aof 日志过大的问题。

当 rdb 和 aof 同时存在时

- 当有 aof 文件时，使用 aof 恢复数据，否则使用 rdb
- 在恢复速度上，rdb 因为是直接的内存映射，而 aof 是逐条运行命令，因此在速度上来说 rdb 比 aof 快很多。

第六章　分布式

6.1　主从复制

- 主从备份，防止主机宕机
- 读写分离，分担 master 的任务
- 任务分离，从服分别担任不同的工作 (备份、计算)

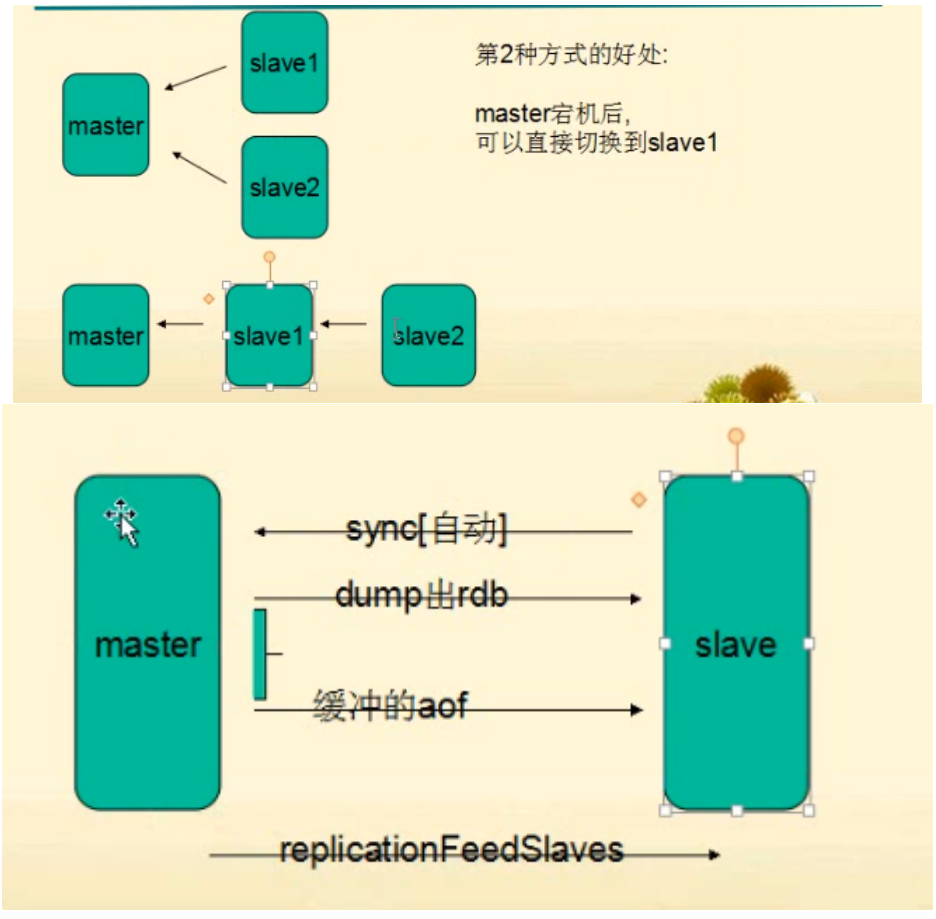

Fig 6.1: 主从 2 种结构、通信方式

6.2 分布式集群

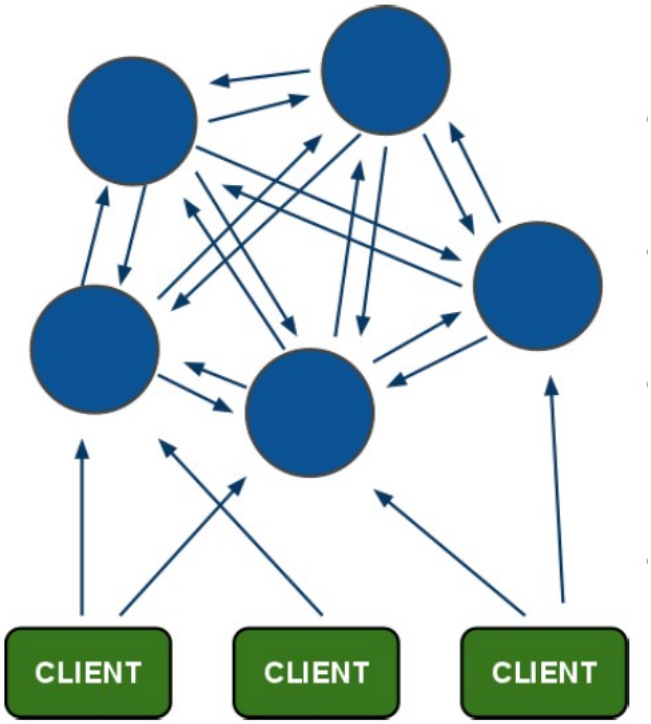

Fig 6.2: Redis 集群示意

在这个图中，每一个蓝色的圈都代表着一个 redis 的服务器节点。它们任何两个节点之间都是相互连通的。客户端可以与任何一个节点相连接，然后就可以访问集群中的任何一个节点。对其进行存取和其他操作。

还有就是因为如果集群的话，是有好多个 redis 一起工作的，那么，就需要这个集群不是那么容易挂掉，所以呢，理论上就应该给集群中的每个节点至少一个备用的 redis 服务。这个备用的 redis 称为从节点（slave）。那么这个集群是如何判断是否有某个节点挂掉了呢？

首先要说的是，每一个节点都存有这个集群所有主节点以及从节点的信息。

它们之间通过互相的 ping-pong 判断是否节点可以连接上。如果有一半以上的节点去 ping 一个节点的时候没有回应，集群就认为这个节点宕机了，然后去连接它的备用节点。如果某个节点和所有从节点全部挂掉，我们集群就进入 faill 状态。还有就是如果有一半以上的主节点宕机，那么我们集群同样进入发力了状态。这就是我们的 redis 的投票机制，具体原理如下图所示：

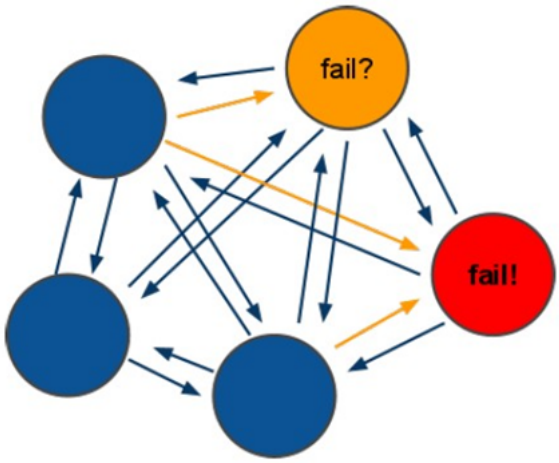

Fig 6.3: 宕机演示

1. 投票过程是集群中所有master 参与, 如果**半数以上 master 节点**与master节点通信超时(cluster-node-timeout), 认为当前 master 节点**挂掉**.
2. 什么时候整个集群不可用 (cluster_state:fail)?
   - 如果集群任意 master 挂掉, 且当前 master 没有 slave. **集群进入 fail 状态**, 也可以理解成集群的 slot 映射 [0-16383] 不完整时进入 fail 状态.
   - 如果集群超过半数以上 master 挂掉，无论是否有 slave，**集群进入 fail 状态**.

6.2.1 参考

- https://www.cnblogs.com/yuanermen/p/5717885.html
- https://www.cnblogs.com/cjsblog/p/9048545.html
- Redis Cluster 原理:https://www.cnblogs.com/liyasong/p/redis_jiqun.html?utm_source=itdadao&utm_medium=referral
- Redis Cluster:http://www.cnblogs.com/foxmailed/p/3630875.html
- 集群实践方案:http://itindex.net/detail/51037-redis-%E5%AE%9E%E8%B7%B5-%E9%9B%86%E7%BE%A4

6.3 Redis 分区

Redis Cluster 是 **Redis 的集群实现**，内置数据**自动分片**机制，集群内部**将所有的 key 映射到 16384 个 Slot 中**，集群中的**每个 Redis Instance 负责其中的一部分的 Slot 的读写**。集群客户端连接集群中任一 Redis Instance 即可发送命令，当 *Redis Instance 收到自己不负责的 Slot*

的请求时，会将负责请求 *Key* 所在 *Slot* 的 *Redis Instance* 地址返回给客户端，客户端收到后自动将原请求重新发往这个地址，对外部透明。**一个 Key 到底属于哪个 Slot 由 crc16(key) % 16384 决定**。

关于负载均衡，集群的 Redis Instance 之间可以迁移数据，以 Slot 为单位，但不是自动的，**需要外部命令触发**。

关于集群成员管理，集群的节点 (Redis Instance) 和节点之间两两定期交换集群内节点信息并且更新，从发送节点的角度看，这些信息包括：集群内有哪些节点，IP 和 PORT 是什么，节点名字是什么，节点的状态 (比如 OK，PFAIL，FAIL，后面详述) 是什么，包括节点角色 (master 或者 slave) 等。

关于可用性，集群由 N 组主从 Redis Instance 组成。主可以没有从，但是没有从意味着主宕机后主负责的 Slot 读写服务不可用。**一个主可以有多个从，主宕机时，某个从会被提升为主，具体哪个从被提升为主，协议类似于 Raft**，参见这里。**如何检测主宕机？Redis Cluster 采用 quorum+ 心跳的机制**。从节点的角度看，节点会定期给其他所有的节点发送 Ping，cluster-node-timeout(可配置，秒级) 时间内没有收到对方的回复，则单方面认为对端节点宕机，将该节点标为 PFAIL 状态。通过节点之间交换信息收集到 quorum 个节点都认为这个节点为 PFAIL，则将该节点标记为 FAIL，并且将其发送给其他所有节点，其他所有节点收到后立即认为该节点宕机。从这里可以看出，主宕机后，至少 cluster-node-timeout 时间内该主所负责的 Slot 的读写服务不可用。

Redis 集群的目的是实现数据的横向伸缩，把一块数据分片保存到多个机器，可以横向扩展数据库大小，扩展带宽，计算能力等。

分区是分割数据到多个 Redis 实例的处理过程，因此每个实例只保存 key 的一个子集。

分区的优势

- 通过利用多台计算机内存的和值，允许我们构造更大的数据库。
- 通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。

redis 集群大多数支持在**运行时**增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做`presharding` 的技术对此是有帮助的。

分区类型

- 范围分区：0-1000,1001-2000，..
- 哈希分区：

第七章　应用

7.1　位图法统计活跃用户

- 1 亿个用户

- 如何记录用户的登录信息

- 如何查询活跃用户，一周连续登录

从信息的角度看，一个用户的登录只需要一个位就可以表示。0-1

在数据库中，数据一般都有编号

简化如：

```
7个用户

周一 01011100
周二 01010010
周三 01111000

setbit mon 100000000 0 // 初始化为0
setbit mon 3 1 // 第3号用户登录，标记为1
setbit mon 9 1


setbit tuseday 10000000 0
set bit tuesday 4 1


bitop and mon tuseday ...
```

7.2 频道发布与订阅

7.3 微博之用户注册与微博发布

7.4 微博之粉丝关系与推送微博

7.5 哈希数据存储微博

第八章　PHP 与 Redis

8.1　关联

```php
$redis = new redis();
$result = $redis->connect('127.0.0.1', 6379);
var_dump($result); //结果: bool(true)

$result = $redis->set('test',"11111111111");
var_dump($result); //结果: bool(true)

$result = $redis->get('test');
var_dump($result); //结果: string(11) "11111111111"

$redis->delete('test');
var_dump($redis->get('test')); //结果: bool(false)
```