1、掌握Redis常用命令和特性(数据类型、持久化、事务、发布订阅等)

2、掌握Redis Cluster特性与部署

Redis安装

wget http://download.redis.io/releases/redis-3.2.9.tar.gz

tar xvf redis-3.2.9.tar.gz

cd redis-3.2.9

make install PREFIX=/root/svr/redis-3.2.9 #安装

src/redis-server ../redis.conf& #启动

src/redis-cli #客户端连接

Ps:介绍下redis.conf文件

常用命令

Keys pattern

*表示区配所有

```
127.0.0.1:6379> keys *
1) "tl_order_id1725610"
2) "bit_order_id_1725519"
3) "tl_order_id1725521"
127.0.0.1:6379>
```

以bit开头的

```
127.0.0.1:6379> keys bit*
1) "bit_order_id_1725519"
127.0.0.1:6379>
```

查看Exists key是否存在

```
127.0.0.1:6379> exists bit
(integer) 0
127.0.0.1:6379> exists bit_order_id_1725519
(integer) 1
127.0.0.1:6379>
```

Set

设置 key 对应的值为 string 类型的 value。

setnx

设置 key 对应的值为 string 类型的 value。如果 key 已经存在，返回 0，nx 是 not exist 的意思。

删除某个key

```
127.0.0.1:6379> del bit_order_id_1725519
(integer) 1
127.0.0.1:6379> del bit_order_id_1725519
(integer) 0
127.0.0.1:6379>
```

第一次返回1 删除了 第二次返回0

Expire 设置过期时间（单位秒）

```
127.0.0.1:6379> expire bit_order_id_1725519 40
(integer) 1
127.0.0.1:6379> ttl bit_order_id_1725519
(integer) 34
127.0.0.1:6379> ttl bit_order_id_1725519
(integer) 33
127.0.0.1:6379> ttl bit_order_id_1725519
(integer) 32
127.0.0.1:6379>
```

TTL查看剩下多少时间

返回负数则key失效，key不存在了

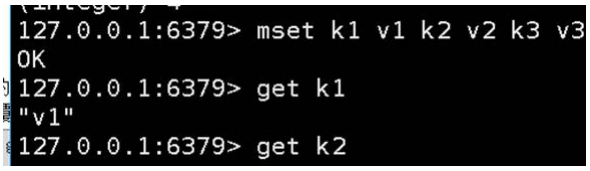
Setex

设置 key 对应的值为 string 类型的 value，并指定此键值对应的有效期。

```
27.0.0.1:6379> setex key  10 value
27.0.0.1:6379> get key
value"
27.0.0.1:6379> ttl key
integer) 6
27.0.0.1:6379> ttl key
integer) 4
```

Mset

一次设置多个 key 的值，成功返回 ok 表示所有的值都设置了，失败返回 0 表示没有任何值被设置。



Getset

设置 key 的值，并返回 key 的旧值。

Mget

一次获取多个 key 的值，如果对应 key 不存在，则对应返回 nil。

Incr

对 key 的值做加加操作,并返回新的值。注意 incr 一个不是 int 的 value 会返回错误，incr 一个不存在的 key，则设置 key 为 1

incrby

同 incr 类似，加指定值 ，key 不存在时候会设置 key，并认为原来的 value 是 0

Decr

对 key 的值做的是减减操作，decr 一个不存在 key，则设置 key 为-1

Decrby

同 decr，减指定值。

Append

给指定 key 的字符串值追加 value,返回新字符串值的长度。

Strlen

取指定 key 的 value 值的长度。

persist xxx(取消过期时间)

选择数据库（0-15库）

Select 0 //选择数据库

move age 1//把age 移动到1库

```
127.0.0.1:6379> set age 10
OK
127.0.0.1:6379> get age
"10"
127.0.0.1:6379> move age 1
(integer) 1
127.0.0.1:6379> get age
(nil)
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> get age
"10"
127.0.0.1:6379[1]>
```

Randomkey随机返回一个key

```
127.0.0.1:6379> randomkey
"tl_order_id1725610"
127.0.0.1:6379> randomkey
"tl_order_id1725521"
127.0.0.1:6379> randomkey
"tl_order_id1725610"
127.0.0.1:6379> randomkey
"tl_order_id1725610"
127.0.0.1:6379>
```

Rename重命名

```
127.0.0.1:6379> set age 10
OK
127.0.0.1:6379> rename age age_new
OK
127.0.0.1:6379> get age_new
"10"
127.0.0.1:6379>
```

Type 返回数据类型

```
127.0.0.1:6379> type age_new
string
127.0.0.1:6379>
```

服务器相关：

Ping 测试连接是否可以成功

```
127.0.0.1:6379> ping
PONG
```

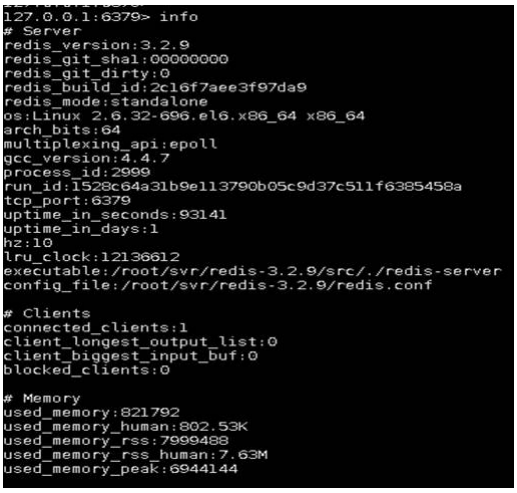
Quit退出连接

```
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> quit
[root@localhost src]#
```

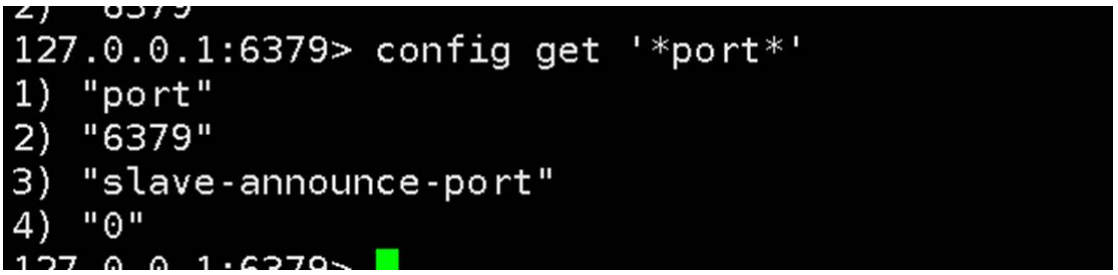
Dbsize 返回key的数量

```
[root@localhost src]# ./redis-cli -a bit
127.0.0.1:6379> dbsize
(integer) 3
127.0.0.1:6379>
```

Info 输出redis信息



Config get|set 显示与修改配置



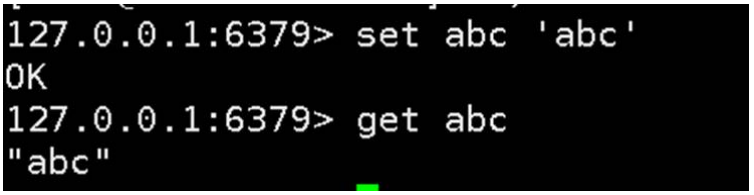
数据类型

String字符串：

格式: **set key value**

string类型是二进制安全的。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 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

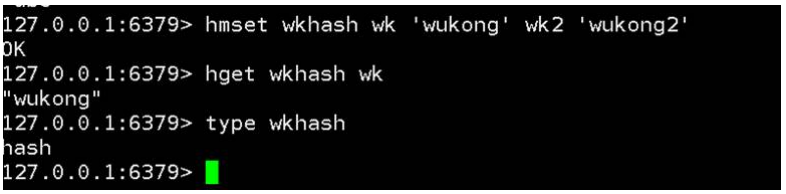


Hash（哈希）

格式: **hmset name key1 value1 key2 value2**

Redis hash 是一个键值(key=>value)对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。



List（列表）

Redis 列表是简单的字符串列表，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）

格式: lpush name value

在 key 对应 list 的头部添加字符串元素

格式: rpush name value

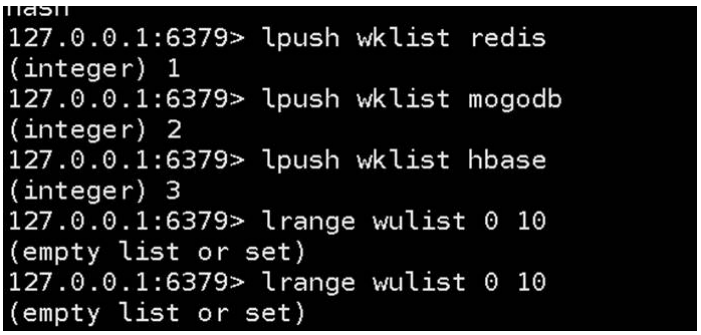
在 key 对应 list 的尾部添加字符串元素

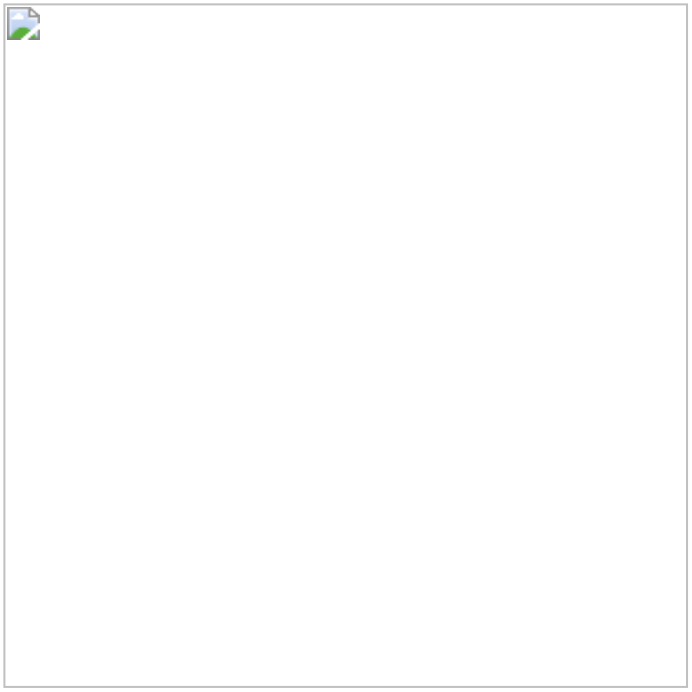
格式: lrem name index

key 对应 list 中删除 count 个和 value 相同的元素

格式: llen name

返回 key 对应 list 的长度





Set（集合）

格式: **sadd name value**

Redis的Set是string类型的无序集合。

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)。

```
127.0.0.1:6379> sadd wkset redis
(integer) 1
127.0.0.1:6379> sadd wkset redis
(integer) 0
127.0.0.1:6379> sadd wkset hbase
(integer) 1
127.0.0.1:6379> sadd wkset mogodb
(integer) 1
127.0.0.1:6379> smembers wkset
1) "redis"
2) "mogodb"
3) "hbase"
127.0.0.1:6379> type wkset
set
```

zset(sorted set：有序集合)

格式: **zadd name score value**

Redis zset 和 set 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。

zset的成员是唯一的,但分数(score)却可以重复。

```
127.0.0.1:6379> zadd wkzset 1 redis
(integer) 0
127.0.0.1:6379> zadd wkzset 3 redis
(integer) 0
127.0.0.1:6379> zadd wkzset 3 mogodb
(integer) 1
127.0.0.1:6379> zrangebyscore wkzset 0 1000
1) "mogodb"
2) "redis"
127.0.0.1:6379> type wkzset
zset
127.0.0.1:6379>
```

事务

redis 对事务的支持目前还比较简单。redis 只能保证一个 client 发起的事务中的命令可以连续的执行，而中间不会插入其他 client 的命令。 由于 redis 是单线程来处理所有 client 的请求的所以做到这点是很容易的。一般情况下 redis 在接受到一个 client 发来的命令后会立即处理并 返回处理结果，但是当一个 client 在一个连接中发出 multi 命令有，这个连接会进入一个事务上下文，该连接后续的命令并不是立即执行，而是先放到一个队列中。当从此连接

受到 exec 命令后，redis 会顺序的执行队列中的所有命令。并将所有命令的运行结果打包到一起返回给 client.然后此连接就 结束事务上下文。

开启事物：exec 取消事物：discard 结束事物：exec

```
127.0.0.1:6379> set age 20
OK
127.0.0.1:6379> get name
"wukong"
127.0.0.1:6379> get age
"20"
127.0.0.1:6379> multi
OK
127.0.0.1:6379> incr age
QUEUED
127.0.0.1:6379> incr name
QUEUED
127.0.0.1:6379> exec
1) (integer) 21
2) (error) ERR value is not an integer or out of range
127.0.0.1:6379> get age
"21"
127.0.0.1:6379> get name
"wukong"
127.0.0.1:6379>
```

发布与订阅：

布订阅(pub/sub)是一种消息通信模式，主要的目的是解耦消息发布者和消息订阅者之间的耦合，这点和设计模式中的观察者模式比较相似。pub/sub 不仅仅解决发布者和订阅者直接代码级别耦合也解决两者在物理部署上的耦合。

在redis实现是SUBSCRIBE （订阅主题）、 UNSUBSCRIBE(取消主题) 和 PUBLISH（推送）

订阅：

```
27.0.0.1:6379> subscribe wk
eading messages... (press Ctrl-C to quit)
) "subscribe"
) "wk"
) (integer) 1
) "message"
) "wk"
) "abc"
) "message"
) "wk"
) "ccc"
```

生产：

```
127.0.0.1:6379>  publish wk abc
(integer) 1
```

PSUBSCRIBE支持*匹配的方式

持久化：

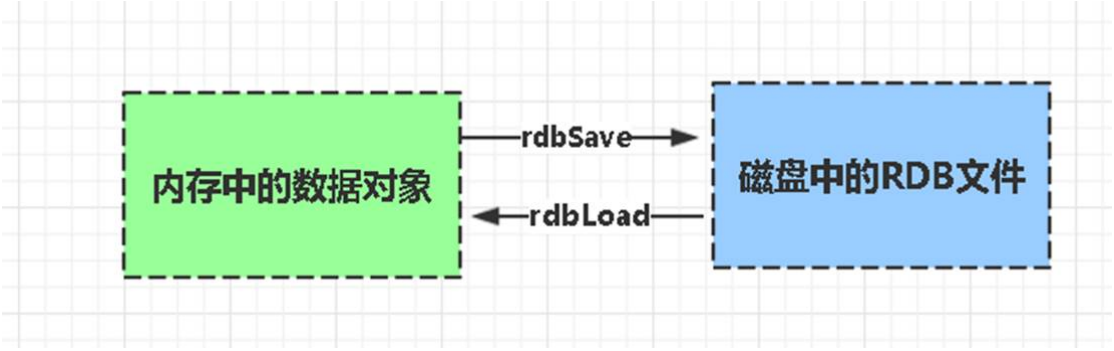
持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis 提供了两种持久化方式:RDB（默认） 和AOF

RDB：

rdb是Redis DataBase缩写

RDB功能核心函数rdbSave(生成RDB文件)和rdbLoad（从文件加载内存）两个函数



rdbSave函数: 将内存中的数据库数据以 RDB 格式保存到磁盘（文件）中，文件存在,那么新的 RDB 文件将替换已有的 RDB 文件。

在保存 RDB 文件期间， 主进程会被阻塞， 直到保存完成为止。

SAVE 和 BGSAVE 两个命令是操作 rdbSave函数的区别：

SAVE 直接调用 rdbSave ，阻塞 Redis 主进程，直到保存完成为止。在主进程阻塞期间，服务器不能处理客户端的任何请求。

BGSAVE 则 fork 出一个子进程，子进程负责调用 rdbSave ，并在保存完成之后向主进程发送信号，通知保存已完成。因为 rdbSave 在子进程被调用，所以 Redis 服务器在 BGSAVE 执行期间仍然可以继续处理客户端的请求。

rdbLoad函数:是redis服务重启或者启动的时候回加载保存到磁盘的RDB文件加载到内存中会被阻塞。

存储结构：

```
+-------+------------+----------+----------------+-----+----------+
| REDIS | RDB-VERSION | SELECT-DB | KEY-VALUE-PAIRS | EOF | CHECK-SUM |
+-------+------------+----------+----------------+-----+----------+
```

http://redisbook.readthedocs.io/en/latest/internal/rdb.html#id4

保存策略：

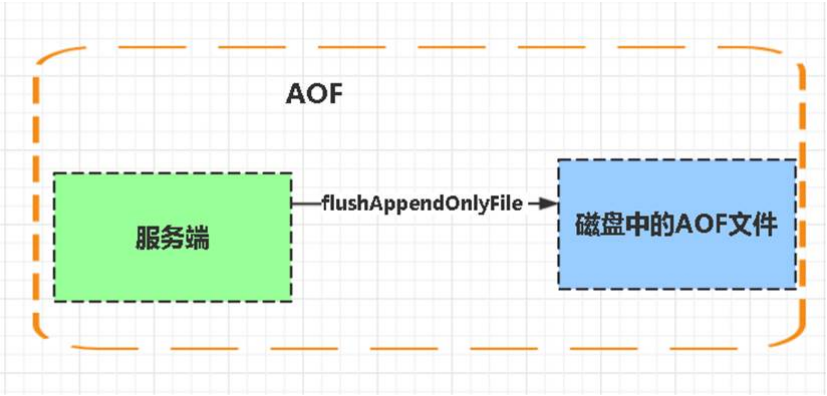
save 900 1

save 300 10 #300 秒内容如超过 10 个 key 被修改，则发起快照保存

save 60 10000

AOF:

Aof是Append-only file缩写



每当执行服务器(定时)任务或者函数时flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作

aof写入保存：

WRITE：根据条件，将 aof_buf 中的缓存写入到 AOF 文件

SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。

两个步骤都需要根据一定的条件来执行，Redis提供了三种条件。

保存策略：

模式	WRITE 是否阻塞？	SAVE 是否阻塞？	停机时丢失的数据量
AOF_FSYNC_NO 不保存	阻塞	阻塞	操作系统最后一次对 AOF 文件触发 SAVE 操作之后的数据。
AOF_FSYNC_EV	阻塞	不阻塞	一般情况下不超过 2 秒

ERYSEC 每一秒钟保存一次			钟的数据。
AOF_FSYNC_ALWAYS 每执行一个命令保存一次	阻塞	阻塞	最多只丢失一个命令的数据。

存储结构:

内容是redis通讯协议(RESP)格式的命令文本存储。

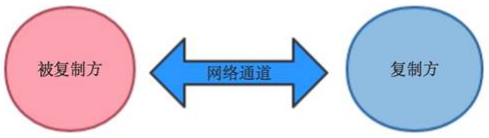
总结：RDB:数据 .AOF:数据+命令

AOF更新频率RDB高 优先加载aof。

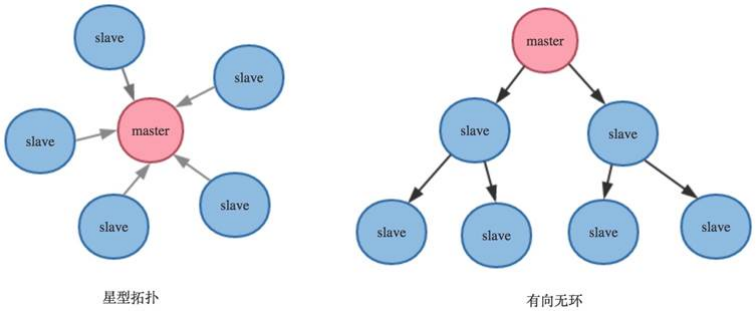
加载的时候没有RDB(数据文件要小)快吧。

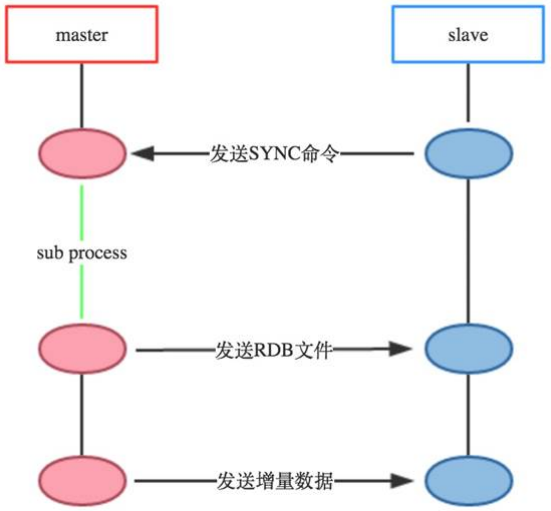
复制:

通常为被复制方（master）主动将数据发送到复制方（slave），复制方接收到数据存储在当前实例，最终目的是为了保证双方的数据一致,同时也是降低了master的压力。



Redis的复制方式有两种，一种是主（master）-从（slave）模式，一种是从（slave）-从（slave）模式.





复制流程图

1、slave向master发送sync命令。

2、master开启子进程执行bgsave写入rdb文件，同时将子进程接收到的写命令缓存起来。

3、子进程写完，父进程得知，开始将RDB文件发送给slave。

4、master发送完RDB文件，将缓存的命令也发给slave。

5、master增量的把写命令发给slave。

主从复制：

第1步：cp reids.conf redis2.conf

第2步：Vim redis2.conf（slave）

第3步：slaveof 192.168.0.12 6379（master的地址）

第4步：Vim redis.conf （master）

第5步：bind 0.0.0.0 #无ip 都可以访问

第6步：./redis-server ../redis.conf #master

第7步./redis-server ../redis.2conf #slave

Ps:是否成功set get请求来判断或执行info命令role

判断是否成功：

1、master客户端set值，slave客户端能不能获取到

2、config get 'slaveof*'

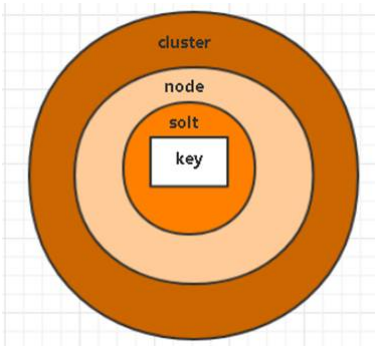
集群：

是一个提供**多个Redis**（分布式）**节点间共享数据**的程序集。

集群部署

Redis 集群的键空间被分割为 16384 hash个槽（slot）， 集群的最大节点数量也是 16384 个

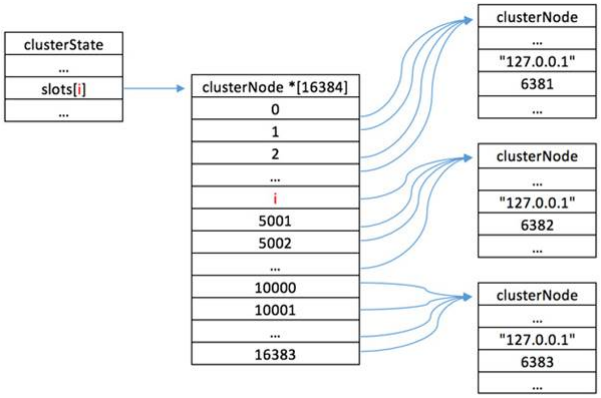
关系:cluster>node>slot>key



分片:

Redis Cluster在设计中没有使用一致性哈希（Consistency Hashing），而是使用数据分片引入哈希槽（hash slot）来实现；一个 Redis Cluster包含16384（0~16383）个哈希槽，存储在Redis Cluster中的所有键都会被映射到这些slot中，集群中的每个键都属于这16384个哈希槽中的一个，集群使用公式slot=CRC16（key）/16384来计算key属于哪个槽，其中CRC16(key)语句用于计算key的CRC16 校验和。

按照槽来进行分片，通过为每个节点指派不同数量的槽，可以控制不同节点负责的数据量和请求数.



当前集群有3个节点,槽默认是平均分的:

节点 A （6381）包含 0 到 5499号哈希槽.

节点 B （6382）包含5500 到 10999 号哈希槽.

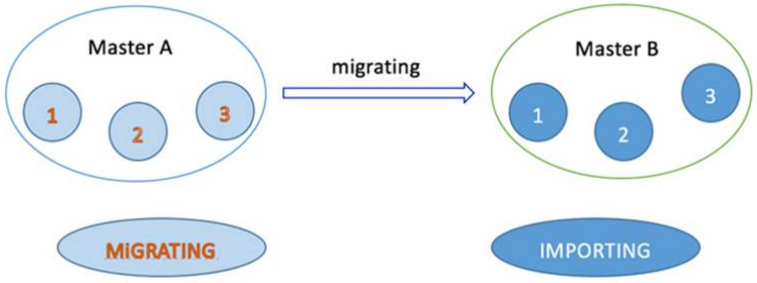
节点 C （6383）包含11000 到 16383号哈希槽.

这种结构很容易添加或者删除节点. 比如如果我想新添加个节点D, 我需要从节点 A, B, C中得部分槽到D上. 如果我像移除节点A,需要将A中得槽移到B和C节点上,然后将没有任何槽的A节点从集群中移除即可. 由于从一个节点将哈希槽移动到另一个节点并不会停止服务,所以无论添加删除或者改变某个节点的哈希槽的数量都不会造成集群不可用的状态.

数据迁移

数据迁移可以理解为slot(槽)和key的迁移，这个功能很重要，极大地方便了集群做线性扩展，以及实现平滑的扩容或缩容。

现在要将Master A节点中编号为1、2、3的slot迁移到Master B节点中，在slot迁移的中间状态下，slot 1、2、3在Master A节点的状态表现为MIGRATING（迁移）,在Master B节点的状态表现为IMPORTING（入口）。



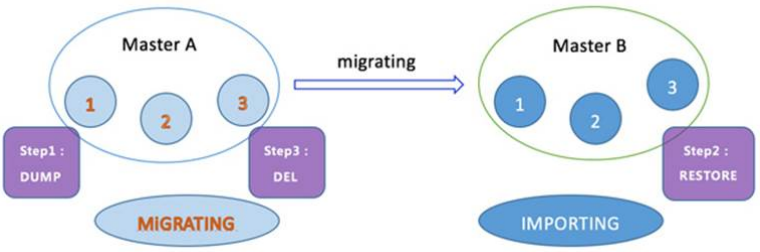
此时并不刷新node的映射关系

IMPORTING状态

被迁移slot 在目标Master B节点中出现的一种状态，准备迁移slot从Mater A到Master B的时候，被迁移slot的状态首先变为IMPORTING状态。

键空间迁移

键空间迁移是指当满足了slot迁移前提的情况下，通过相关命令将slot 1、2、3中的键空间从Master A节点转移到Master B节点。此时刷新node的映射关系。



复制&高可用:

集群的节点内置了复制和高可用特性。

特点：1、节点自动发现

2、slave->master 选举,集群容错

3、Hot resharding:在线分片

4、基于配置(nodes-port.conf)的集群管理

5、客户端与redis节点直连、不需要中间proxy层.

6、所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.

redis-trib.rb 详解：

redis-trib.rb是redis作者用ruby完成的是对redis cluster管理工具，
集成在redis的源码src目录下。

名称	作用
call	在集群全部节点上执行命令
set-timeout	设置集群节点间心跳连接的超时时间
del-node	从集群中删除节点
reshard	在线迁移slot
check	检查集群
import	将外部redis数据导入集群
add-node	将新节点加入集群
create	创建集群
info	查看集群信息
fix	修复集群
rebalance	平衡集群节点slot数量

增加节点：./redis-trib.rb add-node ip:port ip:port

第一个参数是新节点的地址，第二个参数是任意一个已经存在的节点的IP和端口.

移除节点：redis-trib del-node ip:port `<node-id>`

第一个参数是任意一个节点的地址,第二个节点是你想要移除的节点地址

改变一个从节点的主节点cluster replicate master-node-id

压测：

Redis有多快，Redis 自带了一个叫 redis-benchmark 的工具来模拟 N 个客户端同时发出 M 个请求

redis-benchmark -n 100000 -c 60

向redis服务器发送100000个请求，每个请求附带60个并发客户端