

Redis

版本：V 1.0

Redis 简介

是一个开源的使用 **C** 语言编写，支持网络，可基于内存亦可持久化的日志 key-value 数据库 [非关系型数据库]。并支持多种语言的 **API**。注意：非关系型数据库 mysql, oracle, sqlserver, db2 关系型数据库。

支持的语言：

<ul style="list-style-type: none">• ActionScript• C• C++• C#• Clojure	<ul style="list-style-type: none">• Common Lisp• Dart• Erlang• Go• Haskell	<ul style="list-style-type: none">• Haxe• Io• Java• Node.js• Lua	<ul style="list-style-type: none">• Objective-C• Perl• PHP• Pure Data• Python	<ul style="list-style-type: none">• R• Ruby• Scala• Smalltalk• Tcl
-------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------

Redis 特性

Redis 远程的：分为客户端，服务端。可以分别部署在不同的机器上，通过自定义协议进行传输和交互的。平时说的 Redis 通常指的是 Redis 的服务端。

Redis 基于内存的：所有数据结构存在内存中。所有操作非常高速。性能优越于硬盘存储的 mysql，因为存在内存中，所有也是比较吃内存的。

Redis 非关系型数据库：本质是数据库，存储数据，区别于 Mysql。

关系型数据库在存储之前，必须要定义好所谓的数据字典，后续的存储数据按照存储字典来存储，而 Redis 就不需要了。

Redis 应用场景

1. **缓存：**当系统的接口数据比较慢的时候，可以把系统数据接口的数据缓存起来，当下次取的时候，可以直接从缓存中取就可以了。

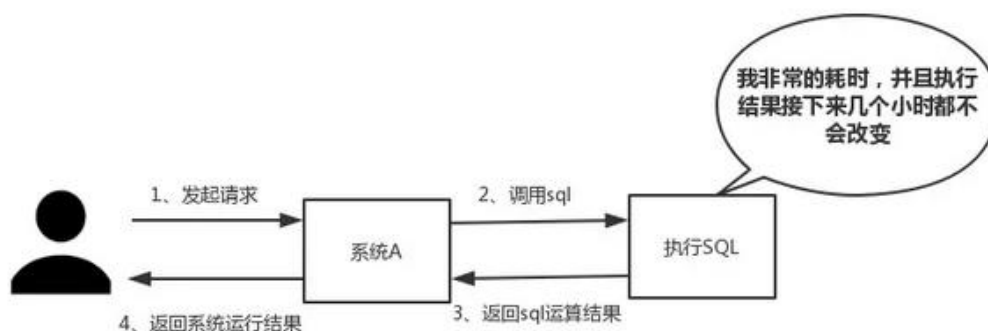
2. **数据存储：**redis 有两种非常完备的持久化机制【**AOF 和 RDB**】，可以定期将数据持久化硬盘中，保障数据的完整性，安全性。

<https://db-engines.com/en/ranking> 使用排行榜

为什么使用 redis

性能

如下图所示，我们在碰到需要执行耗时特别久，且结果不频繁变动的 SQL，就特别适合将运行结果放入缓存。这样，后面的请求就去缓存中读取，使得请求能够迅速响应。



什么叫迅速：

“在理想状态下，我们的页面跳转需要在瞬间解决，对于页内操作则需要在刹那间解决。另外，超过一弹指的耗时操作要有进度提示，并且可以随时中止或取消，这样才能给用户**最好的体验**。”

那么瞬间、刹那、一弹指具体是多少时间呢？

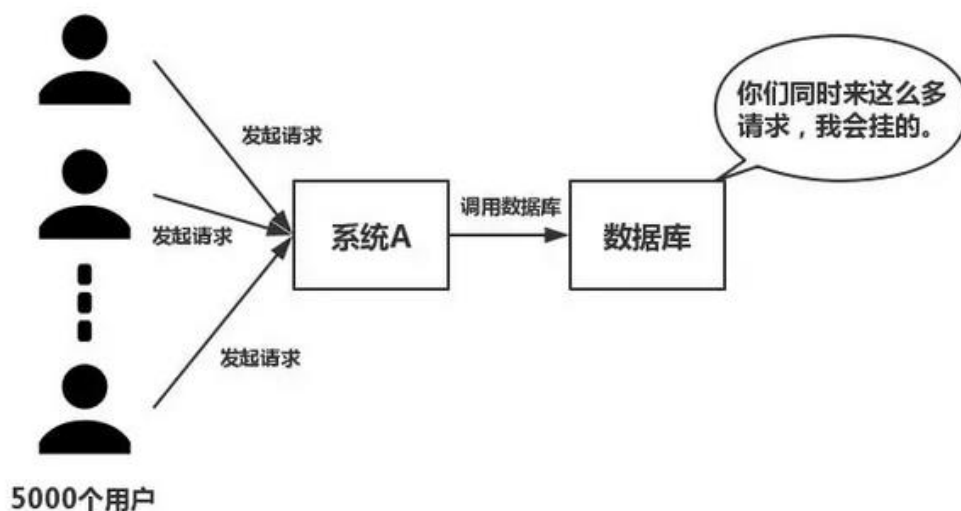
根据《摩诃僧祇律》记载

一刹那者为一念，二十念为一瞬，二十瞬为一弹指，二十弹指为一罗预，二十罗预为一须臾，一日一夜有三十须臾。

那么，经过周密的计算，一瞬间为 0.36 秒，一刹那有 0.018 秒，一弹指长达 7.2 秒。

并发

如下图所示，在大并发的情况下，所有的请求直接访问数据库，数据库会出现连接异常。这个时候，就需要使用 redis 做一个缓冲操作，让请求先访问到 redis，而不是直接访问数据库。



单线程 redis 为什么这么快？

(一)纯内存操作

(二)单线程操作，避免了频繁的上下文切换

(三)采用了非阻塞 I/O **多路复用**机制

多路复用案例：

小明在紫禁城开了一家快递店，负责同城快送服务。小明因为资金限制，雇佣了一批快递员，然后小明发现资金不够了，只够买一辆车送快递。

经营方式一

客户每送来一份快递，小明就让一个快递员盯着，然后快递员开车去送快递。

慢慢的小明就发现了这种经营方式存在下述问题

- 1、几十个快递员基本上时间都花在了抢车上了，大部分快递员都处在闲置状态，谁抢到了车，谁就能去送快递
- 2、随着快递的增多，快递员也越来越多，小明发现快递店里越来越挤，没办法雇佣新的快递员了
- 3、快递员之间的协调很花时间。

经营方式二

小明只雇佣一个快递员。然后呢，客户送来的快递，小明按送达地点标注好，然后依次放在一个地方。最后，那个快递员依次的去取快递，一次拿一个，然后开着车去送快递，送好了就回来拿下一个快递

上述两种经营方式对比，是不是明显觉得第二种，效率更高，更好呢。在上述比喻中：

每个快递员—————>每个线程

每个快递————— ->每个 socket(I/O 流)

快递的送达地点———— ->socket 的不同状态

客户送快递请求———— ->来自客户端的请求

小明的经营方式———— ->服务端运行的代码

一辆车—————>CPU 的核数

结论：

经营方式一就是传统的并发模型，每个 I/O 流(快递)都有一个新的线程(快递员)管理。

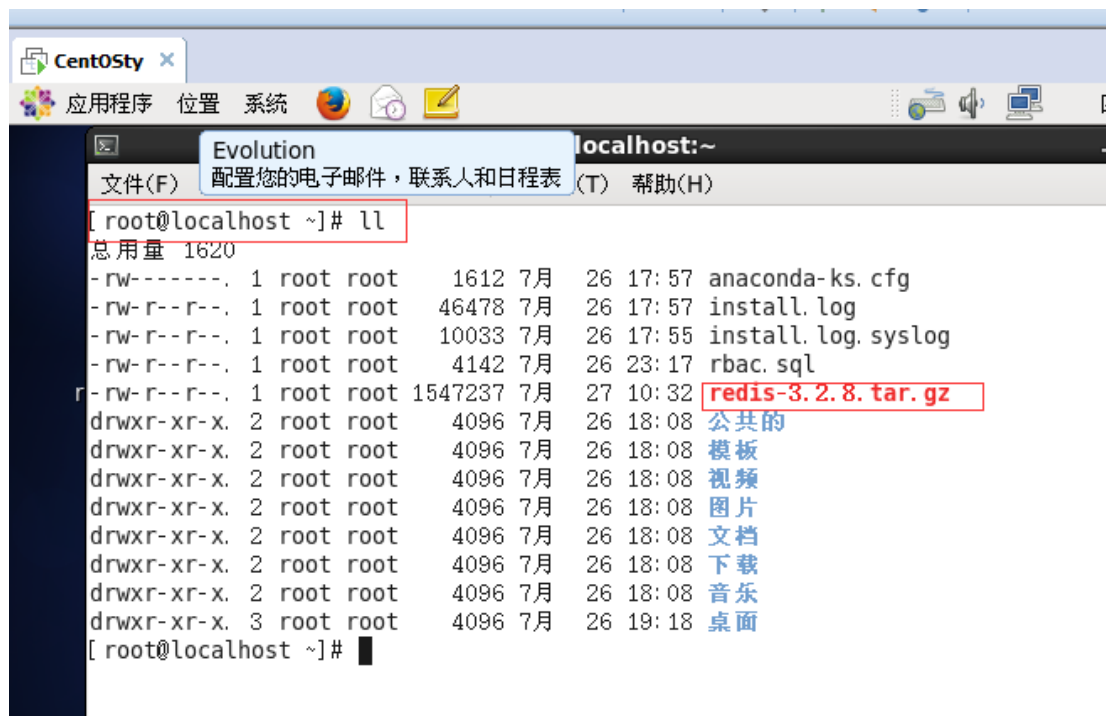
经营方式二就是 I/O 多路复用。只有单个线程(一个快递员)，通过跟踪每个 I/O 流的状态(每个快递的送达地点)，来管理多个 I/O 流。

面试的时候：

将复杂的东西简单化：生活化！

Redis 安装

导入 redis 的压缩包，root 目录即可！



```
CentOS7y x
应用程序 位置 系统
localhost:~
文件(F) Evolution 配置您的电子邮件，联系人和日程表 (T) 帮助(H)
[ root@localhost ~]# ll
总用量 1620
-rw-----. 1 root root 1612 7月 26 17:57 anaconda-ks.cfg
-rw-r--r--. 1 root root 46478 7月 26 17:57 install.log
-rw-r--r--. 1 root root 10033 7月 26 17:55 install.log.syslog
-rw-r--r--. 1 root root 4142 7月 26 23:17 rbac.sql
-rw-r--r--. 1 root root 1547237 7月 27 10:32 redis-3.2.8.tar.gz
drwxr-xr-x. 2 root root 4096 7月 26 18:08 公共的
drwxr-xr-x. 2 root root 4096 7月 26 18:08 模板
drwxr-xr-x. 2 root root 4096 7月 26 18:08 视频
drwxr-xr-x. 2 root root 4096 7月 26 18:08 图片
drwxr-xr-x. 2 root root 4096 7月 26 18:08 文档
drwxr-xr-x. 2 root root 4096 7月 26 18:08 下载
drwxr-xr-x. 3 root root 4096 7月 26 19:18 桌面
[ root@localhost ~]#
```

解压(解压到当前目录即可/root)

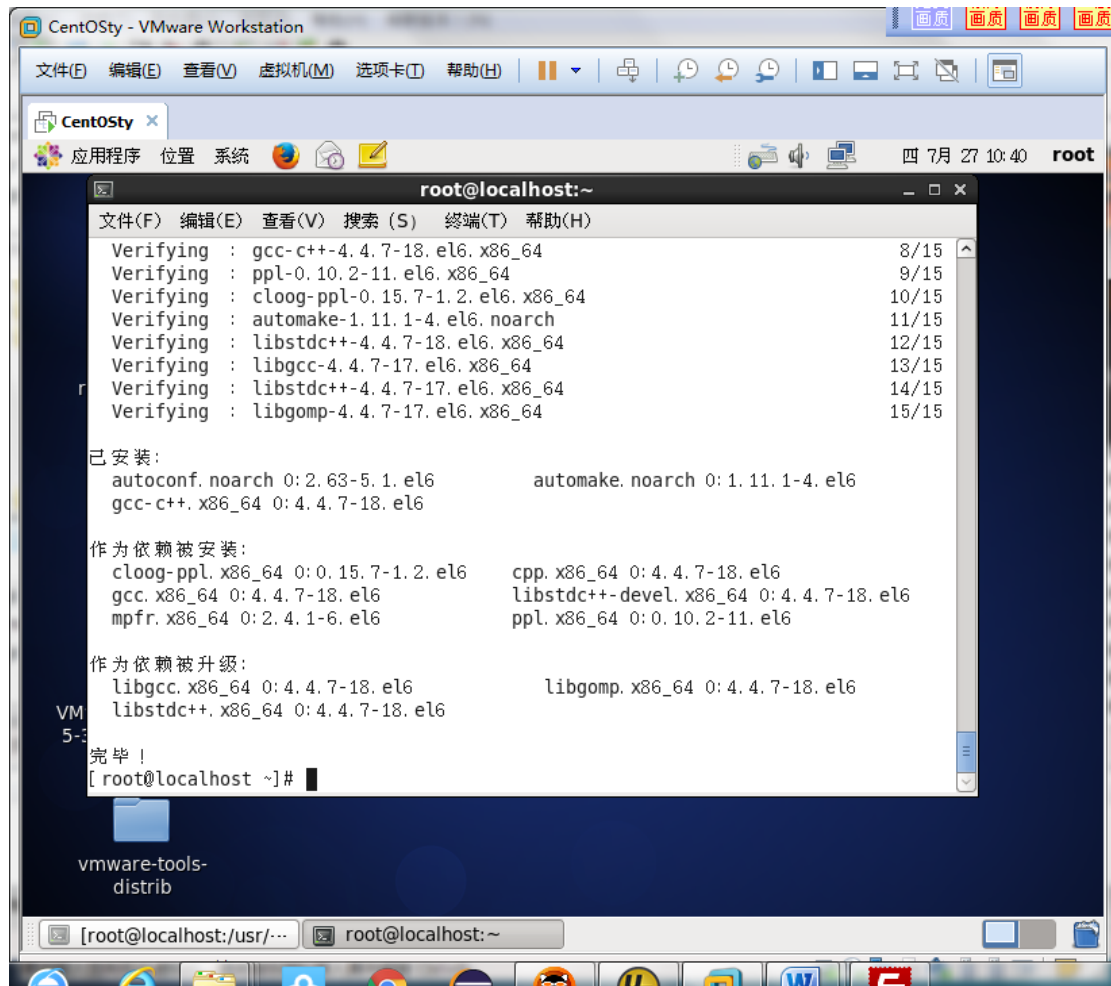
```
tar -zxvf redis-3.2.8.tar.gz
```

要添加依赖。(Redis 是使用 c 语言编写的。)在 root 目录下添加。

`yum install gcc-c++ automake autoconf`, 一般情况下只添加 `gcc-c++` 即可。`automake autoconf` 是为了确保不会出现其他问题而添加的依赖。

`yum`: 最好在有外网的情况下安装使用

如果环境不允许，使用本地 yum 源。



创建安装目录

/opt 存在临时文件

/usr/local 安装应用程序的目录

```
mkdir -p /usr/local/redis
```

进行编译：在解压后的 **redis** 文件夹中，使用 **make** 命令。

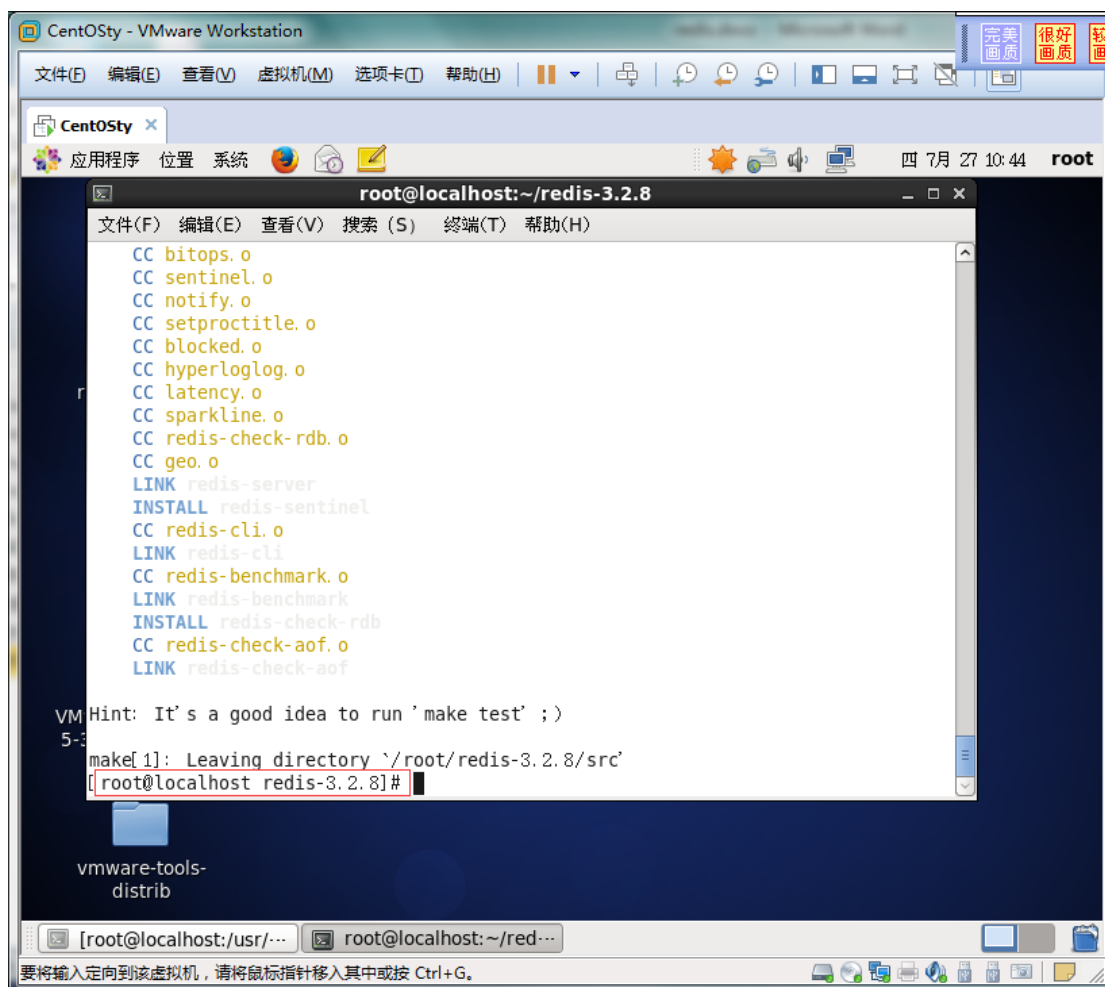
```
# cd /root/redis-3.2.8
```

```
# make
```

7

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可访问百度：尚硅谷官网

使用 make 命令进行编译：



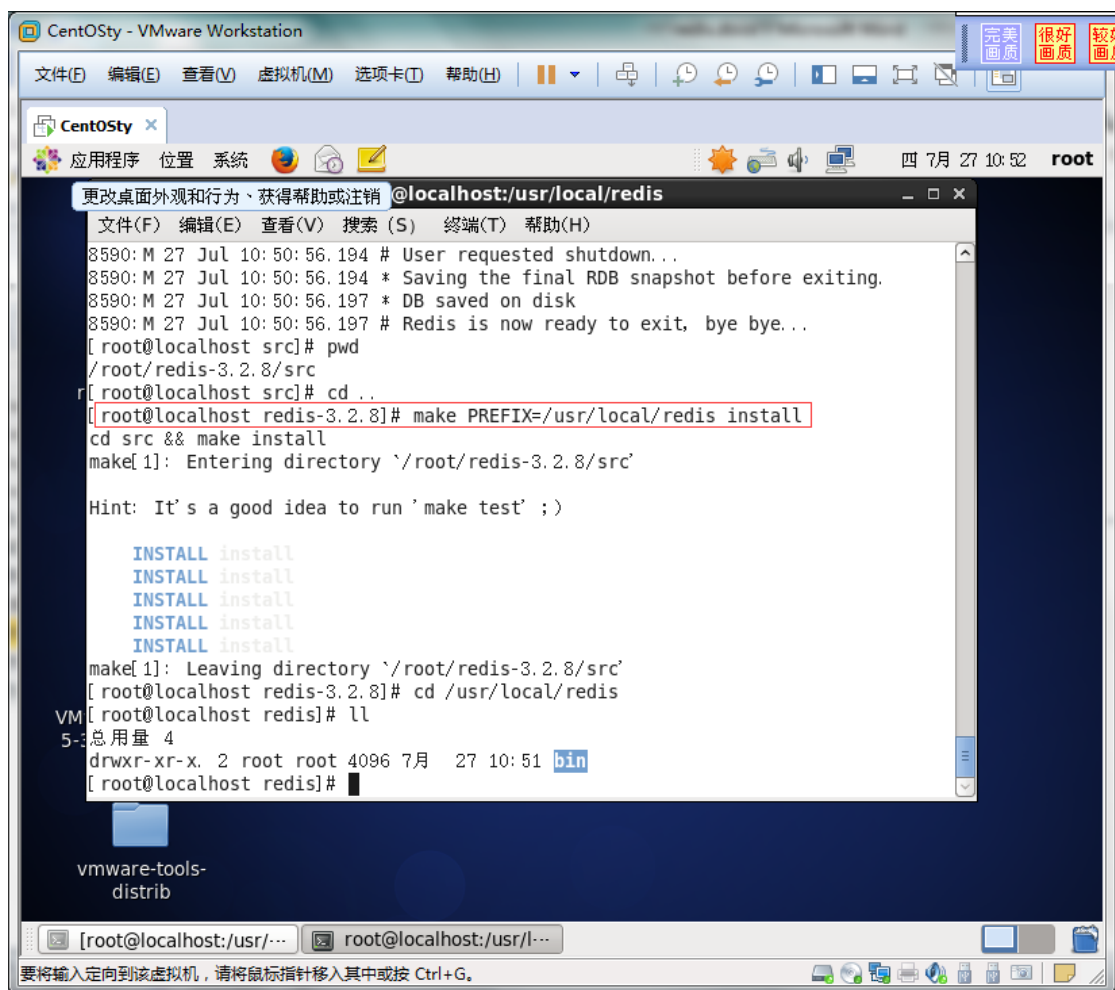
```
CentOS7y - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
CentOS7y x
应用程序 位置 系统
root@localhost:~/redis-3.2.8
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
CC bitops.o
CC sentinel.o
CC notify.o
CC setproctitle.o
CC blocked.o
CC hyperloglog.o
CC latency.o
CC sparkline.o
CC redis-check-rdb.o
CC geo.o
LINK redis-server
INSTALL redis-sentinel
CC redis-cli.o
LINK redis-cli
CC redis-benchmark.o
LINK redis-benchmark
INSTALL redis-check-rdb
CC redis-check-aof.o
LINK redis-check-aof

VM Hint: It's a good idea to run 'make test' ;)
5-3
make[1]: Leaving directory `~/redis-3.2.8/src'
root@localhost redis-3.2.8#
```

进行安装：

使用 make install 安装路径 /usr/local/redis

make install **PREFIX**=/usr/local/redis install 放在前后没有任何影响。



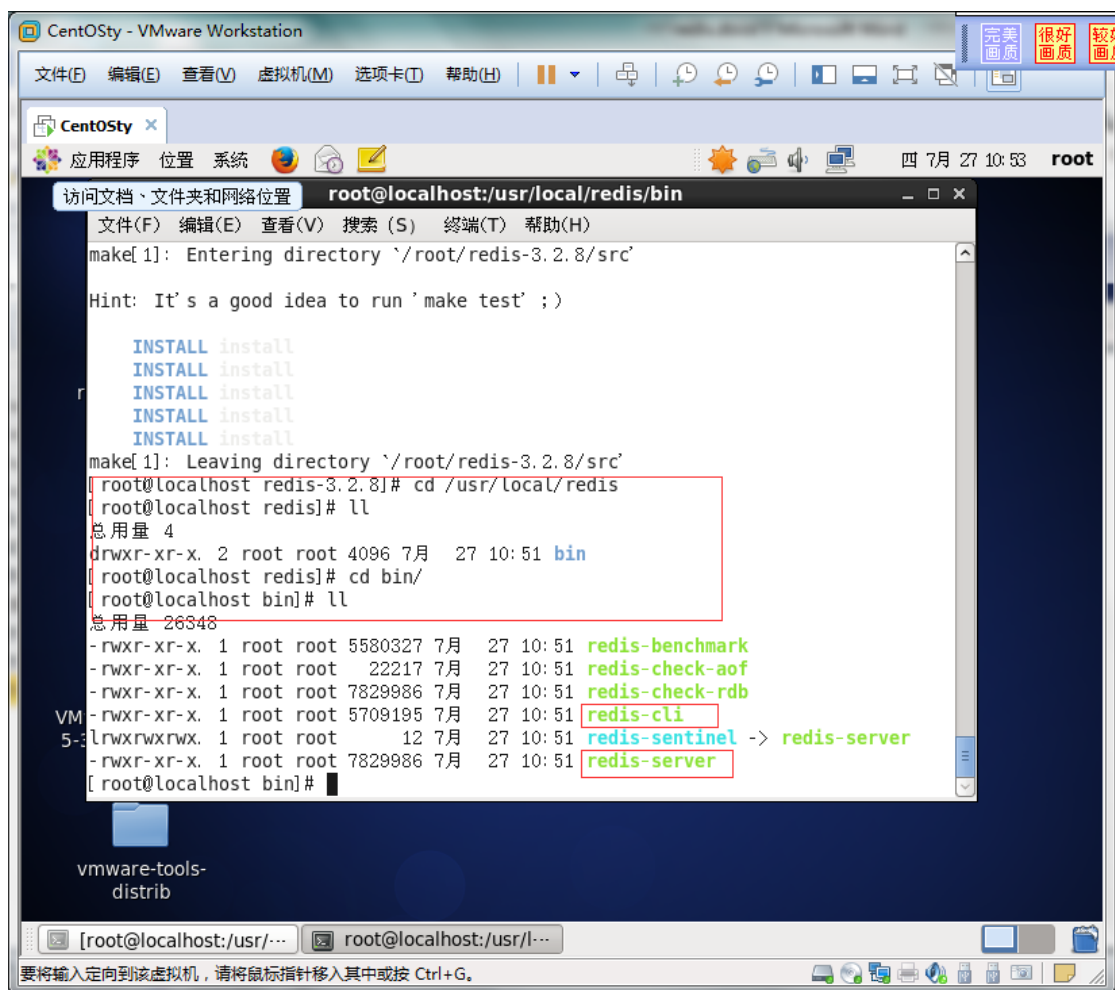
```
CentOS7y - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
CentOS7y x
应用程序 位置 系统
更改桌面外观和行为、获得帮助或注销 @localhost:/usr/local/redis
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
8590: M 27 Jul 10:50:56.194 # User requested shutdown...
8590: M 27 Jul 10:50:56.194 * Saving the final RDB snapshot before exiting.
8590: M 27 Jul 10:50:56.197 * DB saved on disk
8590: M 27 Jul 10:50:56.197 # Redis is now ready to exit, bye bye...
[root@localhost src]# pwd
/root/redis-3.2.8/src
[root@localhost src]# cd ..
[root@localhost redis-3.2.8]# make PREFIX=/usr/local/redis install
cd src && make install
make[1]: Entering directory `/root/redis-3.2.8/src'

Hint: It's a good idea to run 'make test' ;)

INSTALL install
INSTALL install
INSTALL install
INSTALL install
INSTALL install
make[1]: Leaving directory `/root/redis-3.2.8/src'
[root@localhost redis-3.2.8]# cd /usr/local/redis
VM [root@localhost redis]# ll
5-3总用量 4
drwxr-xr-x. 2 root root 4096 7月 27 10:51 bin
[root@localhost redis]#
```

进入安装目录

`cd /usr/local/redis` 目录下会有一个 `bin` 文件夹



```
CentOS7y - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
CentOS7y x
应用程序 位置 系统
访问文档、文件夹和网络位置 root@localhost:/usr/local/redis/bin
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
make[1]: Entering directory `/root/redis-3.2.8/src'
Hint: It's a good idea to run 'make test' ;)

INSTALL install
INSTALL install
INSTALL install
INSTALL install
INSTALL install
INSTALL install
make[1]: Leaving directory `/root/redis-3.2.8/src'
root@localhost redis-3.2.8]# cd /usr/local/redis
root@localhost redis]# ll
总用量 4
drwxr-xr-x. 2 root root 4096 7月 27 10:51 bin
root@localhost redis]# cd bin/
root@localhost bin]# ll
总用量 26348
-rwxr-xr-x. 1 root root 5580327 7月 27 10:51 redis-benchmark
-rwxr-xr-x. 1 root root 22217 7月 27 10:51 redis-check-aof
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-check-rdb
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rwxr-xr-x. 1 root root 12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
root@localhost bin]#
```

redis-cli :表示 redis 的客户端

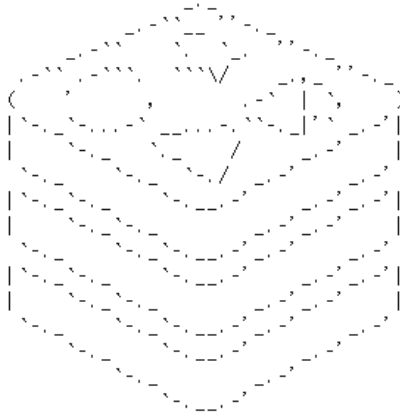
redis-server :表示 redis 的服务器

开始启动服务

./redis-server 启动服务

Port 6379 :redis 默认的端口号

```
[root@localhost bin]# ./redis-server
8639: C 27 Jul 10:54:46.491 # Warning: no config file specified, using the default
t config. In order to specify a config file use ./redis-server /path/to/redis.co
nf
8639: M 27 Jul 10:54:46.491 * Increased maximum number of open files to 10032 (it
was originally set to 1024).
```



Redis 3.2.8 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 8639

<http://redis.io>

```
8639: M 27 Jul 10:54:46.493 # WARNING: The TCP backlog setting of 511 cannot be e
nforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
8639: M 27 Jul 10:54:46.493 # Server started, Redis version 3.2.8
8639: M 27 Jul 10:54:46.493 # WARNING overcommit_memory is set to 0! Background s
ave may fail under low memory condition. To fix this issue add 'vm.overcommit_me
mory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.over
commit_memory=1' for this to take effect.
8639: M 27 Jul 10:54:46.493 # WARNING you have Transparent Huge Pages (THP) suppo
rt enabled in your kernel. This will create latency and memory usage issues with
Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transpare
nt_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retai
n the setting after a reboot. Redis must be restarted after THP is disabled.
8639: M 27 Jul 10:54:46.493 * The server is now ready to accept connections on po
rt 6379
```

启动服务后则该窗口什么也不能做了！

因为 redis 启动方式默认前台启动。为了是窗口可以用，可以设置 redis 为后台启动。

redis 配置

redis.conf 配置文件

bind 绑定那个 ip 地址

protected-mode yes 保护模式是否开启

port 6379 端口号

tcp-backlog 511 确定了 TCP 连接中已完成队列

timeout 0 设置客户端空闲超时时间，服务端不会主动断开连接，不能小于 0。

tcp-keepalive 300 每个一段时间 300 秒发送一次请求.看是否还活着

daemonize yes 是否在后台执行！

supervised no 无监督交互

pidfile /var/run/redis_6379.pid 指定存储 Redis 进程号的文件路径

loglevel notice 日志级别 notice (仅试用于生产)

logfile "" 配置 log 文件地址,默认打印在命令行终端的窗口上,也可设为/dev/null 屏蔽日志

databases 16 指定数据库个数

maxclient 最大客户端连接数

Maxmemory :

设置 Redis 可以使用的内存量。一旦到达内存使用上限,Redis 将会试图移除内部数据,移除规则 可以通过 maxmemory-policy 来指定。如果 Redis 无法根据移除规则来移除内存中的数据,或者设置了“不允许移除”,

那么 Redis 则会针对那些需要申请内存的指令返回错误信息,比如 SET、LPUSH 等。

Maxmemory-policy noeviction 默认配置

- (1) **volatile-lru**: 使用 LRU 算法移除 key, 只对设置了过期时间的键
- (2) **allkeys-lru**: 使用 LRU 算法移除 key
- (3) **volatile-random**: 在过期集合中移除随机的 key, 只对设置了过期时间的键
- (4) **allkeys-random**: 移除随机的 key
- (5) **volatile-ttl**: 移除那些 TTL 值最小的 key, 即那些最近要过期的 key
- (6) **noeviction**: 不进行移除。针对写操作, 只是返回错误信息

cluster-enabled yes 是否开启集群

设置 redis 后台启动

先使用 ctrl+c 退出前台启动

1. 回到 redis 的解压目录去找 redis.conf 配置文件

```
drwxrwxr-x. 2 root root 4096 7月 27 10:50 src
drwxrwxr-x. 10 root root 4096 2月 12 23:14 tests
drwxrwxr-x. 7 root root 4096 2月 12 23:14 utils
[root@localhost redis-3.2.8]# pwdl
bash: pwdl: command not found
[root@localhost redis-3.2.8]# pwd
/root/redis-3.2.8
[root@localhost redis-3.2.8]# ll
总用量 204
-rw-rw-r--. 1 root root 85775 2月 12 23:14 00-RELEASENOTES
-rw-rw-r--. 1 root root 53 2月 12 23:14 BUGS
-rw-rw-r--. 1 root root 1805 2月 12 23:14 CONTRIBUTING
-rw-rw-r--. 1 root root 1487 2月 12 23:14 COPYING
drwxrwxr-x. 7 root root 4096 7月 27 10:42 deps
-rw-rw-r--. 1 root root 11 2月 12 23:14 INSTALL
-rw-rw-r--. 1 root root 151 2月 12 23:14 Makefile
-rw-rw-r--. 1 root root 4223 2月 12 23:14 MANIFESTO
-rw-rw-r--. 1 root root 6834 2月 12 23:14 README.md
-rw-rw-r--. 1 root root 46695 2月 12 23:14 redis.conf
-rwxrwxr-x. 1 root root 271 2月 12 23:14 runtest
-rwxrwxr-x. 1 root root 280 2月 12 23:14 runtest-cluster
-rwxrwxr-x. 1 root root 281 2月 12 23:14 runtest-sentinel
-rw-rw-r--. 1 root root 7606 2月 12 23:14 sentinel.conf
drwxrwxr-x. 2 root root 4096 7月 27 10:50 src
drwxrwxr-x. 10 root root 4096 2月 12 23:14 tests
drwxrwxr-x. 7 root root 4096 2月 12 23:14 utils
```

2. root@localhost redis-3.2.8]# █

3. 找到之后，拷贝到安装目录

```
cp redis.conf /usr/local/redis/bin
```

```
[root@localhost redis-3.2.8]# cp redis.conf /usr/local/redis/bin/
[root@localhost redis-3.2.8]# cd /usr/local/redis/bin/
[root@localhost bin]# ll
总用量 26400
-rw-r--r--. 1 root root 76 7月 27 10:59 dump.rdb
-rwxr-xr-x. 1 root root 5580327 7月 27 10:51 redis-benchmark
-rwxr-xr-x. 1 root root 22217 7月 27 10:51 redis-check-aof
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-check-rdb
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rw-r--r--. 1 root root 46695 7月 27 11:02 redis.conf
lrwxrwxrwx. 1 root root 12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
[root@localhost bin]# a█
```

要修改 redis.conf 配置文件

daemonize no – daemonize yes 此属性表示设置 redis 的启动方式。

no 表示前台启动，yes 表示后台启动。

修改完成之后，再从新启动 redis

```
./redis-server ./redis.conf
```

```
root@localhost:usr/local/redis/bin
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
-rw-rw-r--. 1 root root 6834 2月 12 23:14 README.md
-rw-rw-r--. 1 root root 46695 2月 12 23:14 redis.conf
-rwxrwxr-x. 1 root root 271 2月 12 23:14 runtest
-rwxrwxr-x. 1 root root 280 2月 12 23:14 runtest-cluster
-rwxrwxr-x. 1 root root 281 2月 12 23:14 runtest-sentinel
-rw-rw-r--. 1 root root 7606 2月 12 23:14 sentinel.conf
drwxrwxr-x. 2 root root 4096 7月 27 10:50 src
drwxrwxr-x. 10 root root 4096 2月 12 23:14 tests
drwxrwxr-x. 7 root root 4096 2月 12 23:14 utils
[ root@localhost redis-3.2.8]# cp redis.conf /usr/local/redis/bin/
[ root@localhost redis-3.2.8]# cd /usr/local/redis/bin/
[ root@localhost bin]# ll
总用量 26400
-rw-r--r--. 1 root root 76 7月 27 10:59 dump.rdb
-rwxr-xr-x. 1 root root 5580327 7月 27 10:51 redis-benchmark
-rwxr-xr-x. 1 root root 22217 7月 27 10:51 redis-check-aof
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-check-rdb
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rw-r--r--. 1 root root 46695 7月 27 11:02 redis.conf
lrwxrwxrwx. 1 root root 12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
[ root@localhost bin]# vi redis.conf
[ root@localhost bin]# ./redis-server ./redis.conf
[ root@localhost bin]# ps -ef |grep redis
root      8702      1  0 11:05 ?        00:00:00 ./redis-server 127.0.0.1:6379
root      8706    3600  0 11:05 pts/1    00:00:00 grep redis
[ root@localhost bin]#
```

如果将 redis 的进程 kill 掉。 则再去找 redis 的服务进程。就找不到了！

```
root      8702      1  0 11:05 pts/1    00:00:00 grep redis
[ root@localhost bin]# kill -9 8702
[ root@localhost bin]# ps -ef |grep redis
root      8710    3600  0 11:06 pts/1    00:00:00 grep redis
[ root@localhost bin]#
```

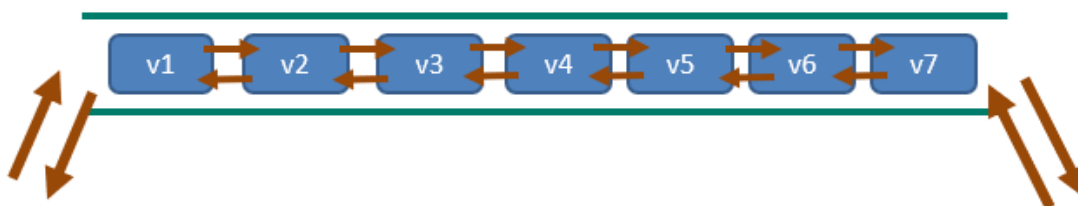
Redis 的五种数据类型

String :

- 是 Redis 最基本的类型，你可以理解成与 Memcached 一模一样的类型，一个 key 对应一个 value。 Map key-value
 - a) String str = "hello"; redis key – value String s = new String("");
- String 类型是二进制安全的。意味着 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 Redis 中字符串 value 最多可以是 512M。

List:

- 单键多值
- 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）。
- 它的底层实际是个双向链表，对两端的操作性能很高，通过索引下标的操作中间的节点性能会较差。



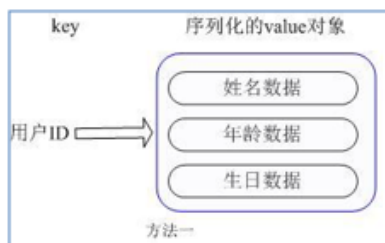
Set:

- 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 是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set 是一个很好的选择。

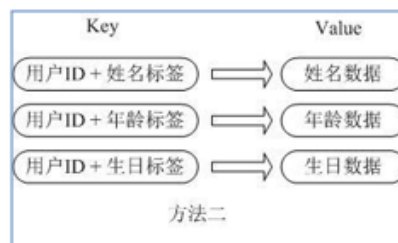
Hash:

- Redis hash 是一个键值对集合。
- 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。
- 类似 Java 里面的 Map<String,Object>

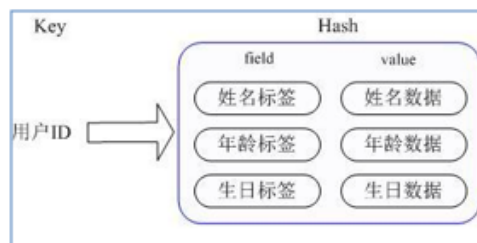
用户ID为查找的key，存储的value用户对象包含姓名，年龄，生日等信息，如果用普通的key/value结构来存储，主要有以下2种存储方式：



每次修改用户的某个属性需要，先反序列化改好后再序列化回去。开销较大。



用户ID数据冗余



通过 key(用户ID) + field(属性标签) 就可以操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题

Zset:

- Redis 有序集合 zset 与普通集合 set 非常相似，是一个没有重复元素的字符串集合。不同之处是有序集合的没有成员都关联了一个评分（score），这个评分（score）被用来按照从最低分到最高分的方式排序集合中的成员。集合的成员是唯一的，但是评分可以是重复了。
- 因为元素是有序的，所以你也可以很快的根据评分（score）或者次序（position）来获取一个范围的元素。访问有序集合的中间元素也是非常快的,因此你能够使用有序集合作为一个没有重复成员的智能列表。

使用场景：

数据类型		项目中的应用
String	字符串	比如说当一个 ip 地址访问网站超过了预定的次数，可以禁止访问，则这个预定次数就可以使用 String 来存储
List	列表	实现最新消息信息排列展示【消息队列】
Set	集合	特殊之处在于 set 是可以自动排重的。比如在微博应用中，每个

		人的好友存在一个集合（ set ）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。
Hash	散列	存储用户信息： key (用户 ID) + field (属性标签) 操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题
Zset	有序集合	以某个条件为权重，比如按顶的次数排序。 需要精准设定过期时间的应用 使用 sorted set 的设置过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据。

常用命令：

非数据类型常用命令

1. 切换库
 - a) `select 0`
2. 启动服务器，用客户端访问
 - a) `./redis-server redis.conf`
 - b) `./redis-cli`
3. 测试验证连接是否正常
 - a) `ping`
4. 查看当前库的所有键
 - a) `keys *`
5. 判断 `key` 是否存在
 - a) `exists <key>`
6. 查看键的类型

- a) type <key>
- 7. 删除某个键
 - a) del <key>
- 8. 设置 key 的过期时间单位为 秒
 - a) expire <key> <seconds>
- 9. 查看还有多少秒过期, -1 表示永不过期, -2 表示已过期
 - a) ttl <key>
- 10. 查看当前数据库的 key 的数量
 - a) dbsize
- 11. 清空当前库
 - a) flushdb
- 12. 通杀全部库
 - a) flushall

String 类型常用命令

- 1、添加键值对
 - a) set <key>
- 2、查询对应键值
 - a) get <key>
- 3、将给定的<value> 追加到原值的末尾
 - a) append <key> <value>
- 4、获得值的长度
 - a) strlen <key>
- 5、只有在 key 不存在时设置 key 的值
 - a) setnx <key> <value>
- 6、将 key 中储存的数字值增 1,只能对数字值操作, 如果为空, 新增值为 1
 - a) incr <key>

-
- 7、将 key 中储存的数字值减 1,只能对数字值操作, 如果为空, 新增值为-1
 - a) `decr <key>`
 - 8、将 key 中储存的数字值增减。自定义步长。
 - a) `incrby / decrby <key> <步长>`
 - 9、同时设置一个或多个 key-value 对
 - a) `mset <key1> <value1> <key2> <value2>`
 - 10、同时获取一个或多个 value
 - a) `mget <key1> <key2> <key3>`
 - 11、同时设置一个或多个 key-value 对, 当且仅当所有给定 key 都不存在。
 - a) `msetnx <key1> <value1> <key2> <value2>`
 - 12、获得值的范围
 - a) `getrange <key> <起始位置> <结束位置>`
 - 13、用 <value> 覆写<key> 所储存的字符串值, 从<起始位置>开始。
 - a) `setrange <key> <起始位置> <value>`
 - 14、设置键值的同时, 设置过期时间, 单位秒。
 - a) `setex <key> <过期时间> <value>`
 - 15、以新换旧, 设置了新值同时获得旧值
 - a) `getset <key> <value>`

List 类型常用命令

- 1、从左边/右边插入一个或多个值。
 - a) `lpush/rpush <key> <value1> <value2> <value3>`
- 2、从左边/右边吐出一个值。值在键在, 值光键亡
 - a) `lpop/rpop <key>`
- 3、按照索引下标获得元素(从左到右) 元素不会丢失!
 - a) `lrange <key> <start> <stop>`

4、获得列表长度

a) `llen <key>`

5、在<value>的后面插入<newvalue> 插入值

a) `linsert <key> before <value> <pivot> <newvalue>`

Set 类型常用命令

1、将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略。

a) `sadd <key> <value1> <value2>`

2、取出该集合的所有值。

a) `smembers <key>`

3、判断集合<key>是否为含有该<value>值，有返回 1，没有返回 0

a) `sismember <key> <value>`

4、删除集合中的某个元素。

a) `srem <key> <value1> <value2>`

5、随机从该集合中吐出一个值。

a) `spop <key>`

6、随机从该集合中取出 n 个值，不会从集合中删除。

a) `srndmember <key> <n>`

7、返回两个集合的交集元素。

a) `sinter <key1> <key2>`

8、返回两个集合的并集元素。

a) `sunion <key1> <key2>`

9、返回两个集合的差集元素，返回的结果跟 key 的顺序有关系

a) `sdiff <key1> <key2>`

Hash 类型常用命令

- 1、给<key>集合中的 <field>键赋值<value>
a) `hset <key> <field> <value>`
- 2、从<key1>集合<field> 取出 value
a) `hget <key1> <field>`
- 3、批量设置 hash 的值
a) `hmset <key1> <field1> <value1> <field2> <value2>...`
- 4、批量取出 hash 的值
a) `hmget <key1> <field1> <field2>...`
- 5、查看哈希表 key 中，给定域 field 是否存在。有返回 1，没有返回 0
a) `hexists key <field>`
- 6、列出该 hash 集合的所有 field
a) `hkeys <key>`
- 7、列出该 hash 集合的所有 value
a) `hvals <key>`

Zset 类型常用命令

- 1、将一个或多个 member 元素及其 score 值加入到有序集 key 当中。
a) `zadd <key> <score1> <value1> <score2> <value2>...`
- 2、返回有序集 key 中，下标在<start><stop>之间的元素带 WITHSCORES，可以让分数一起和值返回到结果集。
a) `zrange <key> <start> <stop> [WITHSCORES]`

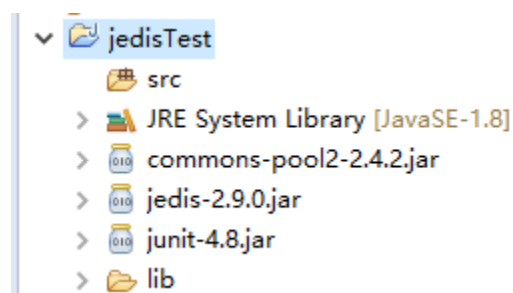
① WITHSCORES 如果在命令行上加上该选项，则将 score 和 value 一同取出，如果不加该选项，则只取 value 值！
- 3、返回有序集 key 中，所有 score 值介于 min 和 max 之间(包括等于 min 或 max)的成员。有序集成员按 score 值递增(从小到大)次序排列。

- a) `zrangebyscore key min max [withscores] [limit offset count]`
- b) `zrevrangebyscore key max min [withscores] [limit offset count]` 从大到小
- 4、为元素的 score 加上增量
 - a) `zincrby <key> <increment> <member>`
- 5、删除该集合下，指定值的元素
 - a) `zrem <key> <value>`
- 6、统计该集合，分数区间内的元素个数
 - a) `zcount <key> <min> <max>`
- 7、返回该值在集合中的排名，从 0 开始。
 - a) `zrank <key> <value>`

使用 eclipse 连接操作 redis

1. 关闭防火墙
2. 修改 bind 标签 为*，或者直接注释掉 可以使用 windows 客户端连接

创建工程



1. 导入 jar 包

commons-pool2-2.4.2.jar

jedis-2.9.0.jar

junit-4.8.jar

2. 创建测试类

- a) Jedis: 这个类是用来操作 redis 的。该类中包含了操作 redis 的所有方法。

```
public class TestJedis {
    @Test
    public void connection() {

        // 创建连接使用Jedis类

        Jedis jd = new Jedis("192.168.67.201", 6379);
        System.out.println(jd.ping());

        jd.set("gyy", "大美女");

        jd.set("ty", "清纯小美女");

        System.out.println(jd.get("gyy"));
    }
}
```

3. 使用连接池进行测试

```
JedisPool jp = null;

Jedis jd = null;

@Before

public void connection() {

    jp = new JedisPool("192.168.67.201", 6379);

    jd = jp.getResource();

    System.out.println(jd.ping());

}

// 测试 String 添加单条数据

@Test

public void tStr() {

    jd.set("name0", "陈乔恩");

    System.out.println(jd.get("name0"));

}
```

```
// 测试 String 添加多条数据 json {"name":"value"}
@Test
public void tStrs() {
    jd.mset("name1", "赵丽颖", "sex", "女");
    System.out.println(jd.mget("name1", "sex"));
}

// 测试 hash 添加单条数据
@Test
public void tHash() {
    jd.hset("h01", "name2", "姐己");
    System.out.println(jd.hget("h01", "name2"));
}

// 测试 hash 添加多条数据
@Test
public void tHashs() {
    Map map = new HashMap<>();
    map.put("name3", "大乔");
    map.put("name4", "曹操");
    jd.hmset("h02", map);
    System.out.println(jd.hmget("h02", "name3", "name4"));
}

// 测试 list
@Test
```



```
public void tlist() {  
    // jd.lpush("list01", "赵云");  
    // jd.lpush("list01", "张飞");  
    // jd.lpush("list01", "关羽");  
    // jd.lpush("list01", "刘备");  
    // 再次运行结果是:  
    jd.lpush("list01", "赵云");  
    jd.lpush("list02", "张飞");  
    jd.lpush("list03", "关羽");  
    jd.lpush("list04", "刘备");  
  
    // 从新添加一个集合  
    jd.lpush("list05", "凯");  
    jd.lpush("list06", "虞姬");  
    jd.lpush("list07", "孙悟空");  
  
    System.out.println(jd.lrange("list05", 0, 5));  
}  
  
// 测试 set  
@Test  
public void tset() {  
    jd.sadd("set01", "太乙真人");  
    jd.sadd("set01", "蔡文姬");  
    jd.sadd("set01", "扁鹊");  
    System.out.println(jd.smembers("set01"));  
}
```

```
// 测试 sorted Set 对数据进行排序

@Test

public void tsortSet() {

    Map<String, Double> map = new HashMap<>();

    map.put("100", new Double(1));

    map.put("98", new Double(3));

    map.put("99", new Double(2));

    jd.zadd("sorted01", map);

    //0,1 取得集合中的元素，主要排序是根据 Double 数值进行排序。

    System.out.println(jd.zrange("sorted01", 0, 1));

}
```

Redis

版本：V 1.0

Redis 持久化

Redis 主要是工作在内存中。内存本身就不是一个持久化设备，断电后数据会清空。所以 Redis 在工作过程中，如果发生了意外停电事故，如何尽可能减少数据丢失。

Redis 持久化

Redis 提供了不同级别的持久化方式:

- RDB持久化方式能够在指定的时间间隔对你的数据进行快照存储.
- AOF持久化方式记录每次对服务器写的操作,当服务器重启的时候会重新执行这些命令来恢复原始的数据,AOF命令以re协议追加保存每次写的操作到文件末尾.Redis还能对AOF文件进行后台重写,使得AOF文件的体积不至于过大.
- 如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式.
- 你也可以同时开启两种持久化方式,在这种情况下,当redis重启的时候会优先载入AOF文件来恢复原始的数据,因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整.
- 最重要的事情是了解RDB和AOF持久化方式的不同,让我们以RDB持久化方式开始:

1. RDB

1.1 RDB 简介

RDB: 在指定的**时间间隔内**将内存中的数据集快照**写入磁盘**,也就是行话讲的 Snapshot 快照,它恢复时是将快照文件直接读到内存里。

工作机制: 每隔一段时间,就把内存中的数据保存到硬盘上的指定文件中。

RDB 是默认开启的!

Redis 会单独创建(fork)一个子进程来进行持久化,会先将数据写入到一个临时文件中【xxx.rdb】,待持久化过程都结束了,再用这个临时文件替换上次持久化好的文件。整个过程中,主进程是不进行任何 IO 操作的,这就确保了极高的性能如果需要进行大规模数据的恢复,且对于数据恢复的完整性不是非常敏感,那 RDB 方式要比 AOF 方式更加的高效。

RDB 的缺点是最后一次持久化后的**数据可能丢失**。

1.2 RDB 保存策略

save 900 1 900 秒内如果至少有 1 个 key 的值变化,则保存

save 300 10 300 秒内如果至少有 10 个 key 的值变化,则保存

save 60 10000 60 秒内如果至少有 10000 个 key 的值变化,则保存

save "" 就是禁用 RDB 模式;

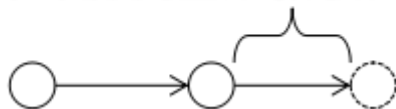
1.3 RDB 常用属性配置

属性	含义	备注
save	保存策略	
dbfilename	RDB 快照文件名	
dir	RDB 快照保存的目录	必须是一个目录，不能是文件名。最好改为固定目录。默认为./代表执行 <code>redis-server</code> 命令时的当前目录！
stop-writes-on-bgsave-error	是否在备份出错时，继续接受写操作	如果用户开启了 RDB 快照功能，那么在 <code>redis</code> 持久化数据到磁盘时如果出现失败，默认情况下， <code>redis</code> 会停止接受所有的写请求
rdbcompression	对于存储到磁盘中的快照，可以设置是否进行压缩存储。	如果是的话， <code>redis</code> 会采用 LZF 算法进行压缩。如果你不想消耗 CPU 来进行压缩的话， 可以设置为关闭此功能，但是存储在磁盘上的快照会比较大。
rdbchecksum	是否进行数据校验	在存储快照后，我们还可以让 <code>redis</code> 使用 CRC64 算法 来进行数据校验，但是这样做会增加 大约 10%的性能消耗 ， 如果希望获取到最大的性能提升，可以关闭此功能。

1.4 RDB 数据丢失的情况

两次保存的时间间隔内，服务器宕机，或者发生断电问题。

有可能发生数据丢失的时间段



1.5 RDB 的触发

设置 `save 30 10` 条的时候触发

在指定范围内向数据库添加数据

- ① 拷贝 rdb 文件 `cp dump.rdb dump_bak.rdb`
- ② 执行 `flushdb`，再执行 `shutdown` 命令，也会产生 `dump.rdb`，但里面是空的，没有意义。空的 `dump.rdb` 默认大小 76
- ③ 当执行 `shutdown` 命令时，也会主动地备份数据。
- ④ 重新将赋值的 rdb 文件数据在拷贝回去。在进行连接测试。则数据产生了。
- ⑤ **注意：使用 rdb 备份的时候，如果使用了 `flushdb`，则数据将会被取消！**

1.6 RDB 的优缺点

RDB的优点

- RDB是一个非常紧凑的文件,它保存了某个时间点得数据集,非常适用于数据集的备份,比如你可以在每个小时保存一下过去24小时内的数据,同时每天保存过去30天的数据,这样即使出了问题你可以根据需求恢复到不同版本的数据集.
- RDB是一个紧凑的单一文件,很方便传送到另一个远端数据中心或者亚马逊的S3(可能加密),非常适用于灾难恢复.
- RDB在保存RDB文件时父进程唯一需要做的就是fork出一个子进程,接下来的工作全部由子进程来做,父进程不需要再做其他IO操作,所以RDB持久化方式可以最大化redis的性能.
- 与AOF相比,在恢复大的数据集的时候,RDB方式会更快一些.

• RDB的缺点

- 如果你希望在redis意外停止工作(例如电源中断)的情况下丢失的数据最少的话,那么RDB不适合你.虽然你可以配置不同的save时间点(例如每隔5分钟并且对数据集有100个写的操作),是Redis要完整的保存整个数据集是一个比较繁重的工作,你通常会每隔5分钟或者更久做一次完整的保存,万一在Redis意外宕机,你可能会丢失几分钟的数据.
- RDB需要经常fork子进程来保存数据集到硬盘上,当数据集比较大的时候,fork的过程是非常耗时的,可能会导致Redis在一些毫秒级内不能响应客户端的请求.如果数据集巨大并且CPU性能不是很好的情况下,这种情况会持续1秒,AOF也需要fork,但是你可以调节重写日志文件的频率来提高数据集的耐久度.

2. AOF

2.1 AOF 简介

- AOF 是以日志的形式来记录每个写操作，将每一次对数据进行修改，都把新建、修改数据的命令保存到指定文件中。Redis 重新启动时读取这个文件，重新执行新建、修改数据的命令恢复数据。
- 默认不开启，需要手动开启 593 行 `appendonly yes`
- AOF 文件的保存路径，同 RDB 的路径一致。# `dir ./`
- AOF 在保存命令的时候，只会保存对数据有修改的命令，也就是写操作！
- 当 RDB 和 AOF 存的不一致的情况下，按照 AOF 来恢复。因为 AOF 是对 RDB 的补充。备份周期更短，也就更可靠。

2.2 AOF 保存策略

`appendfsync always`: 每次产生一条新的修改数据的命令都执行保存操作；效率低，但是安全！

`appendfsync everysec`: 每秒执行一次保存操作。如果在未保存当前秒内操作时发生了断电，仍然会导致一部分数据丢失（即 1 秒钟的数据）。

`appendfsync no`: 从不保存，将数据交给操作系统来处理。更快，也更不安全的选择。

推荐（并且也是默认）的措施为每秒 `fsync` 一次，这种 `fsync` 策略可以兼顾速度和安全性。

2.3 AOF 常用属性

属性	含义	备注
<code>appendonly</code>	是否开启 AOF 功能	默认是关闭的 <code>yes</code>

appendfilename	AOF 文件名称	appendonly.aof
appendfsync	AOF 保存策略	官方建议 everysec
no-appendfsync-on-rewrite	在重写时，是否执行保存策略	执行重写，可以节省 AOF 文件的体积；而且在恢复的时候效率也更高。
auto-aof-rewrite-percentage	重写的触发条件	当目前 aof 文件大小超过上一次重写的 aof 文件大小的百分之多少进行重写
auto-aof-rewrite-min-size	设置允许重写的最小 aof 文件大小	避免了达到约定百分比但尺寸仍然很小的情况还要重写

Aof：属于实时备份。文件大小，应该比 rdb 要大。

Rdb：间隔备份。

2.4 AOF 文件的修复

2.4.1 # vim redis.conf

2.4.2 开启 aof 配置 appendonly yes,则会产生对应的日志文件 appendonly.aof

2.4.3 启动空的服务，写入命令

2.4.4 写入完成命令之后，flushall，shutdown。

2.4.5 结果 error！还是空的：aof 备份的实在就是记录所有的命令！

2.4.6 分析原因：查看 vim appendonly.aof 将最后的 flushall 删除，再重新查询。

2.4.7 使用 vim 命令在 appendonly.aof 添加一些无效信息，再启动，并使用客户端连接，则会发现链接失败！

如果 AOF 文件中出现了残余命令，会导致服务器无法重启。此时需要借助 redis-check-aof 工具来修复！

命令：redis-check-aof --fix 文件

注意：只要是 aof 开启，则无论怎么样都会读取 aof 的配置文件。

2.5 AOF 的优缺点

• AOF 优点

- 使用AOF会让你的Redis更加耐久: 你可以使用不同的fsync策略: 无fsync,每秒fsync,每次写的时候fsync.使用默认的每秒fsync策略,Redis的性能依然很好(fsync是由后台线程进行处理的,主线程会尽力处理客户端请求),一旦出现故障,你最多丢失1秒的数据.
- AOF文件是一个只进行追加的日志文件,所以不需要写入seek,即使由于某些原因(磁盘空间已满,写的过程中宕机等等)未执行完整的写入命令,你也可使用redis-check-aof工具修复这些问题.
- Redis 可以在 AOF 文件体积变得过大时,自动地在后台对 AOF 进行重写: 重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。整个重写操作是绝对安全的,因为 Redis 在创建新 AOF 文件的过程中,会继续将命令追加到现有的 AOF 文件里面,即使重写过程中发生停机,现有的 AOF 文件也不会丢失。而一旦新 AOF 文件创建完毕,Redis 就会从旧 AOF 文件切换到新 AOF 文件,并开始对新 AOF 文件进行追加操作。
- AOF 文件有序地保存了对数据库执行的所有写入操作, 这些写入操作以 Redis 协议的格式保存, 因此 AOF 文件的内容非常容易被读懂, 对文件进行分析(parse)也很轻松。导出(export) AOF 文件也非常简单: 举个例子, 如果你不小心执行了 FLUSHALL 命令, 但只要 AOF 文件未被重写, 那么只要停止服务器, 移除 AOF 文件末尾的 FLUSHALL 命令, 并重启 Redis, 就可以将数据集恢复到 FLUSHALL 执行之前的状态。

优点:

- 备份机制更稳健, 丢失数据概率更低
- 可读的日志文本, 通过操作 AOF 稳健, 可以处理误操作【redis-check-aof --fix 文件】

• AOF 缺点

- 对于相同的数据集来说, AOF 文件的体积通常要大于 RDB 文件的体积。
- 根据所使用的 fsync 策略, AOF 的速度可能会慢于 RDB。在一般情况下, 每秒 fsync 的性能依然非常高, 而关闭 fsync 可以让 AOF 的速度和 RDB 一样快, 即使在高负荷之下也是如此。不过在处理巨大的写入载入时, RDB 可以提供更有保证的最大延迟时间(latency)。

缺点:

- 比起 RDB 占用更多的磁盘空间
- 恢复备份速度要慢
- 每次读写都同步的话, 有一定的性能压力
- 存在个别 Bug, 造成恢复不能。

Redis 主从复制

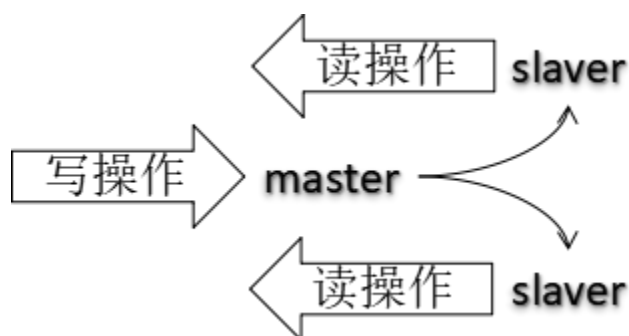
1. 主从简介

配置多台 Redis 服务器，以主机和备机的身份分开。主机数据更新后，根据配置和策略，自动同步到备机的 **master/slaver** 机制，Master 以写为主，Slave 以读为主，二者之间自动同步数据。

目的：

读写分离提高 Redis 性能；

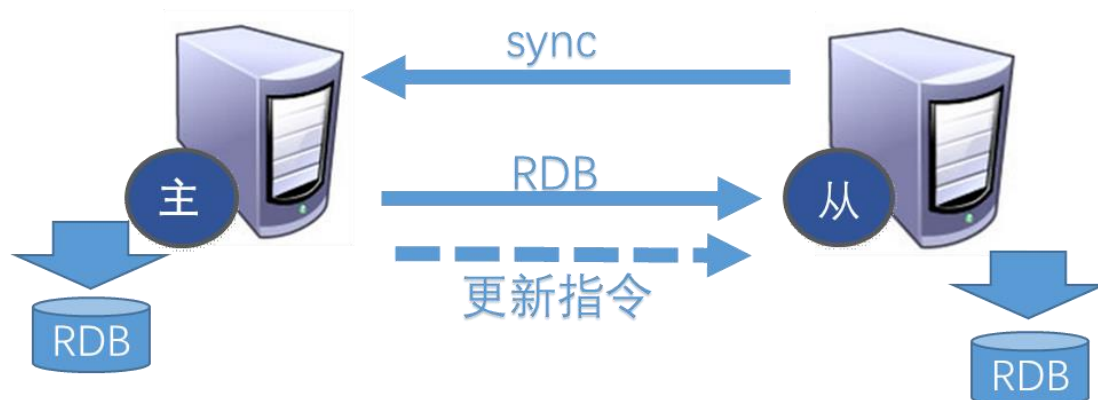
避免单点故障，容灾快速恢复



原理：

每次从机联通后，都会给主机发送 **sync** 指令，主机立刻进行存盘操作，发送 **RDB** 文件，给从机

从机收到 **RDB** 文件后，进行全盘加载。之后每次主机的写操作，都会立刻发送给从机，从机执行相同的命令



2. 主从准备

除非是不同主机配置不同的 Redis 服务，否则在一台机器上面跑多个 Redis 服务，需要配置多个 Redis 配置文件。

①准备多个 Redis 配置文件，每个配置文件，需要配置以下属性

daemonize yes: 服务在后台运行

port: 端口号

pidfile: pid 保存文件

logfile: 日志文件(如果没有指定的话，就不需要)

dump.rdb: RDB 备份文件的名称

appendonly 关掉，或者是更改 appendonly 文件的名称。 aof

样本:

```
include /usr/local/redis/bin/redis.conf  
port 6379  
pidfile /var/run/redis_6379.pid  
dbfilename dump_6379.rdb
```

②根据多个配置文件，启动多个 Redis 服务

```
[root@localhost bin]# ./redis-server 6379.conf
[root@localhost bin]# ./redis-server 6380.conf
[root@localhost bin]# ./redis-server 6381.conf
[root@localhost bin]# ps -ef |grep redis
root      3548      1  0 10:46 ?        00:00:01 ./re
dis-server *:6379
root      3665      1  0 11:08 ?        00:00:00 ./re
dis-server *:6380
root      3669      1  0 11:08 ?        00:00:00 ./re
dis-server *:6381
root      3675    3449  0 11:08 pts/0    00:00:00 grep
redis
```

要添加当前会话，点击左侧的箭头按钮。

```
[root@localhost bin]# ./redis-cli -p 6379
127.0.0.1:6379>
^ [root@localhost bin]# ./redis-cli -p 6380
127.0.0.1:6380>
^ [root@localhost bin]# ./redis-cli -p 6381
127.0.0.1:6381>
```

要添加当前会话，点击左侧的箭头按钮。

```
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> set k2 v2
OK
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6379>
^ [root@localhost bin]# ./redis-cli -p 6380
127.0.0.1:6380> get k2
(nil)
127.0.0.1:6380> get k1
(nil)
127.0.0.1:6380>
^ [root@localhost bin]# ./redis-cli -p 6381
127.0.0.1:6381> get k2
(nil)
127.0.0.1:6381> get k1
(nil)
127.0.0.1:6381>
```

`./redis-cli -p 6379`

`-p` :表示要连接到哪个端口上的服务器！

3. 主从建立

3.1 临时建立

原则：配从不配主。

配置：在从服务器上执行 `SLAVEOF ip:port` 命令；

查看：执行 `info replication` 命令；

```
127.0.0.1:6380> slaveof 127.0.0.1 6379
OK
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:2
master_sync_in_progress:0
slave_repl_offset:1
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

```
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=71,lag=0
slave1:ip=127.0.0.1,port=6381,state=online,offset=71,lag=0
master_repl_offset:71
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:70
127.0.0.1:6379>

127.0.0.1:6380> slaveof 127.0.0.1 6379
OK
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:10
master_sync_in_progress:0
slave_repl_offset:113
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6380>

127.0.0.1:6381> slaveof 127.0.0.1 6379
OK
127.0.0.1:6381> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:7
master_sync_in_progress:0
slave_repl_offset:127
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6381>
```

确立关系之后，set key value 查看效果。关键是以前的 k1, k2 从机同样能够得到

```
ssh://root***@192.168.67.201:22
要添加当前会话，点击左侧的箭头按钮。

127.0.0.1:6379> set k3 v3
OK
127.0.0.1:6379>

127.0.0.1:6380> get k3
"v3"
127.0.0.1:6380> get k1
"v1"
127.0.0.1:6380>

127.0.0.1:6381> get k3
"v3"
127.0.0.1:6381> get k2
"v2"
127.0.0.1:6381>
```

注意：从机只会读取，不能写入

```
127.0.0.1:6380> set k10 v10
(error) READONLY You can't write against a read only slave.
127.0.0.1:6380>
```

3.2 永久建立

在从机的配置文件中，编写 slaveof 属性配置！

```
# slaveof 127.0.0.1 6379
```

3.3 恢复身份

执行命令 `slaveof no one` 恢复自由身！【反客为主】

4. 主从常见问题

①从机是从头开始复制主机的信息，还是只复制切入以后的信息？

答：从头开始复制，即完全复制。实际上是读取主机的 rdb

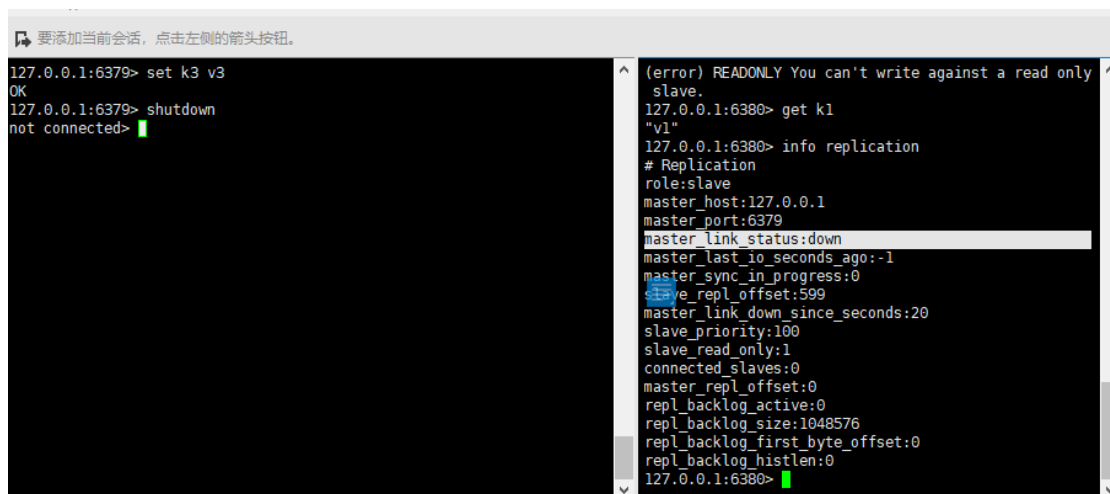
②从机是否可以写？

答：不能

```
127.0.0.1:6381> set k2 v1
(error) READONLY You can't write against a read only slave
```

③主机 shutdown 后，从机是上位还是原地待命？

答：原地待命



```
要添加当前会话，点击左侧的箭头按钮。
127.0.0.1:6379> set k3 v3
OK
127.0.0.1:6379> shutdown
not connected>
127.0.0.1:6380> (error) READONLY You can't write against a read only slave.
127.0.0.1:6380> get k1
"v1"
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:down
master_last_io_seconds_ago:-1
master_sync_in_progress:0
slave_repl_offset:599
master_link_down_since_seconds:20
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6380>
```

④主机又回来了后，主机新增记录，从机还能否顺利复制？

答：可以

```
要添加当前会话，点击左侧的箭头按钮。
127.0.0.1:6379> set k3 v3
OK
127.0.0.1:6379> shutdown
not connected>
[root@localhost bin]# ./redis-server 6379.conf
[root@localhost bin]# ./redis-cli
127.0.0.1:6379> set k11 v11
OK
127.0.0.1:6379>
127.0.0.1:6380> get k11
"v11"
127.0.0.1:6380>
```

⑤从机宕机后，重启，宕机期间主机的新增记录，从机是否会顺利复制？

答：可以

```
not connected>
[root@localhost bin]# ./redis-server 6379.conf
[root@localhost bin]# ./redis-cli
127.0.0.1:6379> set k11 v11
OK
127.0.0.1:6379> set k12 v12
OK
127.0.0.1:6379>
(error) ERR unknown command 'shutdown'
127.0.0.1:6380> shutdown
not connected>
[root@localhost bin]# ./redis-server 6380.conf
[root@localhost bin]# ./redis-cli
127.0.0.1:6379> get k12
"v12"
127.0.0.1:6379>
"v2"
127.0.0.1:6381> get k11
"v11"
127.0.0.1:6381> get k12
"v12"
127.0.0.1:6381>
```

⑥其中一台从机 down 后重启，能否重认旧主？

答：不一定，看配置文件中是否配置了 slaveof

slaveof 127.0.0.1 6379

```
[root@localhost bin]# ./redis-server 6380.conf
[root@localhost bin]# ./redis-cli -p 6380
127.0.0.1:6380> set k100 v100
(error) READONLY You can't write against a read only slave.
127.0.0.1:6380>
```

⑦如果两台从机都从主机同步数据，此时主机的 IO 压力会增大，如何解决？

答：按照主---从（主）---从模式配置！[薪火相传]

slaveof 127.0.0.1 6380

5. 哨兵模式

5.1 简介

- 反客为主的自动版，能够后台监控主机是否故障，如果故障了根据投票数自动将从库转换为主库。



作用：

- ① Master 状态检测
- ② 如果 Master 异常，则会进行 Master-Slave 切换，将其中一个 Slave 作为 Master，将之前的 Master 作为 Slave

5.2 配置

自定义的/usr/local/redis/bin 目录下新建 sentinel.conf 文件

```
# vim sentinel.conf
```

哨兵模式需要配置哨兵的配置文件！

```
sentinel monitor mymaster 127.0.0.1 6379 1
```

启动哨兵: `./redis-sentinel sentinel.conf`

```
[root@centos4 redis_replication]# redis-sentinel sentinel.conf
2918:X 11 Apr 00:49:43.127 * Increased maximum number of open files to 10032 (it was originally set to 1024).

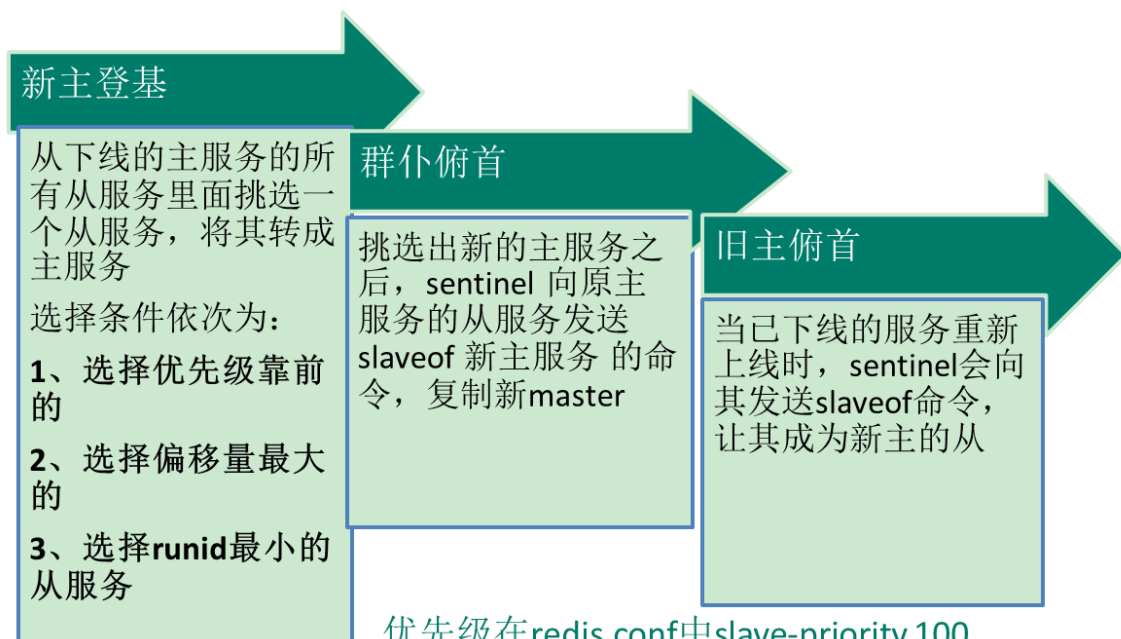
Redis 3.2.5 (00000000/0) 64 bit

Running in sentinel mode
Port: 26379
PID: 2918

http://redis.io

2918:X 11 Apr 00:49:43.129 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
2918:X 11 Apr 00:49:43.141 # Sentinel ID is fc2f5019106049a4b444437b0f4147883d20bffa
2918:X 11 Apr 00:49:43.141 # +monitor master mymaster 127.0.0.1 6379 quorum 1
2918:X 11 Apr 00:49:43.143 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster 127.0.0.1 6379
```

5.3 主机宕机后



优先级在redis.conf中slave-priority 100

偏移量是指获得原主数据最多的

每个redis实例启动后都会随机生成一个40位的runid

slave-priority 100

主推:

redis 集群

创建目录

配置文件:

conf 是用来放配置文件

`mkdir -p opt/redis/conf`

log 是用来放日志

`mkdir -p /opt/redis/log`

data 是用来放集群之后，产生的数据文件。

`mkdir -p /opt/redis/data`

```
[root@localhost ~]# mkdir -p /opt/redis/log
[root@localhost ~]# mkdir -p /opt/redis/data
[root@localhost ~]# mkdir -p /opt/redis/conf
[root@localhost ~]# █
```

不行知道用谁来管理集群? ruby!

上传 ruby 脚本：集群工具用来管理集群使用。

```
[ root@localhost ~]# ll
总用量 4776
-rw-----. 1 root root    1612 7月 26 17:57 anaconda-ks.cfg
-rw-r--r--. 1 root root   46478 7月 26 17:57 install.log
-rw-r--r--. 1 root root   10033 7月 26 17:55 install.log.syslog
-rw-r--r--. 1 root root    4142 7月 26 23:17 rbac.sql
drwxrwxr-x. 6 root root    4096 2月 12 23:14 redis-3.2.8
-rw-r--r--. 1 root root 1547237 7月 27 10:32 redis-3.2.8.tar.gz
-rw-r--r--. 1 root root 3226139 7月 27 15:07 ruby-gems.tar.gz
drwxr-xr-x. 2 root root    4096 7月 26 18:08 公共的
drwxr-xr-x. 2 root root    4096 7月 26 18:08 模板
drwxr-xr-x. 2 root root    4096 7月 26 18:08 视频
drwxr-xr-x. 2 root root    4096 7月 26 18:08 图片
drwxr-xr-x. 2 root root    4096 7月 26 18:08 文档
drwxr-xr-x. 2 root root    4096 7月 26 18:08 下载
drwxr-xr-x. 2 root root    4096 7月 26 18:08 音乐
drwxr-xr-x. 3 root root    4096 7月 26 19:18 桌面
[ root@localhost ~]#
```

用来管理集群

解压 ruby 脚本

```
[ root@localhost ~]# tar -zxvf ruby-gems.tar.gz
ruby-gems/
ruby-gems/ruby-rdoc-1.8.7.374-4.el6_6.x86_64.rpm
ruby-gems/ruby-irb-1.8.7.374-4.el6_6.x86_64.rpm
ruby-gems/redis-3.0.6.gem
ruby-gems/compat-readline5-5.2-17.1.el6.x86_64.rpm
ruby-gems/rubygems-1.3.7-5.el6.noarch.rpm
ruby-gems/install.sh
ruby-gems/ruby-1.8.7.374-4.el6_6.x86_64.rpm
ruby-gems/ruby-libs-1.8.7.374-4.el6_6.x86_64.rpm
[ root@localhost ~]# ll
总用量 4780
-rw-----. 1 root root    1612 7月 26 17:57 anaconda-ks.cfg
-rw-r--r--. 1 root root   46478 7月 26 17:57 install.log
-rw-r--r--. 1 root root   10033 7月 26 17:55 install.log.syslog
-rw-r--r--. 1 root root    4142 7月 26 23:17 rbac.sql
drwxrwxr-x. 6 root root    4096 2月 12 23:14 redis-3.2.8
-rw-r--r--. 1 root root 1547237 7月 27 10:32 redis-3.2.8.tar.gz
drwxr-xr-x. 2 root root    4096 8月 23 2016 ruby-gems
-rw-r--r--. 1 root root 3226139 7月 27 15:07 ruby-gems.tar.gz
drwxr-xr-x. 2 root root    4096 7月 26 18:08 公共的
drwxr-xr-x. 2 root root    4096 7月 26 18:08 模板
drwxr-xr-x. 2 root root    4096 7月 26 18:08 视频
drwxr-xr-x. 2 root root    4096 7月 26 18:08 图片
drwxr-xr-x. 2 root root    4096 7月 26 18:08 文档
drwxr-xr-x. 2 root root    4096 7月 26 18:08 下载
```

进入 ruby-gems 文件中，进行安装

```
[root@localhost ~]# cd ruby-gems
[root@localhost ruby-gems]# ll
总用量 3352
-rw-r--r--. 1 root root 132636 8月 23 2016 compat-readline5-5.2-17.1.el6.x86_64.rpm
-rwxr-xr-x. 1 root root 318 8月 23 2016 install.sh
-rw-r--r--. 1 root root 66048 8月 23 2016 redis-3.0.6.gem
-rw-r--r--. 1 root root 551232 3月 2 2015 ruby-1.8.7.374-4.el6_6.x86_64.rpm
-rw-r--r--. 1 root root 211764 11月 25 2013 rubygems-1.3.7-5.el6.noarch.rpm
-rw-r--r--. 1 root root 324992 8月 23 2016 ruby-irb-1.8.7.374-4.el6_6.x86_64.rpm
-rw-r--r--. 1 root root 1732652 3月 2 2015 ruby-libs-1.8.7.374-4.el6_6.x86_64.rpm
-rw-r--r--. 1 root root 389836 8月 23 2016 ruby-rdoc-1.8.7.374-4.el6_6.x86_64.rpm
[root@localhost ruby-gems]# ./install.sh
Preparing... [100%]
 1: compat-readline5 [100%]
Preparing... [100%]
 1: ruby-libs [100%]
Preparing... [100%]
 1: ruby [100%]
Preparing... [100%]
 1: ruby-irb [100%]
Preparing... [100%]
 1: ruby-rdoc [100%]
Preparing... [100%]
 1: rubygems [100%]
Successfully installed redis-3.0.6
1 gem installed
Installing ri documentation for redis-3.0.6...
Installing RDoc documentation for redis-3.0.6...
^[[A[root@localhost ruby-gems]#
```

整理配置集群的配置文件。

Redis 在运行的时候，是通过处理 `redis.conf` 配置文件来运行的。而 `redis.conf` 配置文件中的每个属性都有其特殊的含义。要想配置集群，则应该会有多个配置文件。集群的每个配置文件中应该都有特殊的设置。比如说 `port` 应该不同。应该共同之处。在配置集群的时候，只要将特殊部分取出来。再有一个公共的配置文件来处理公共的属性。

1-6.配置文件中每个端口号应该不一致！除了端口号之外，可能其他的共同属性，把共同属性放在一个配置文件中即可。7 配置文件。

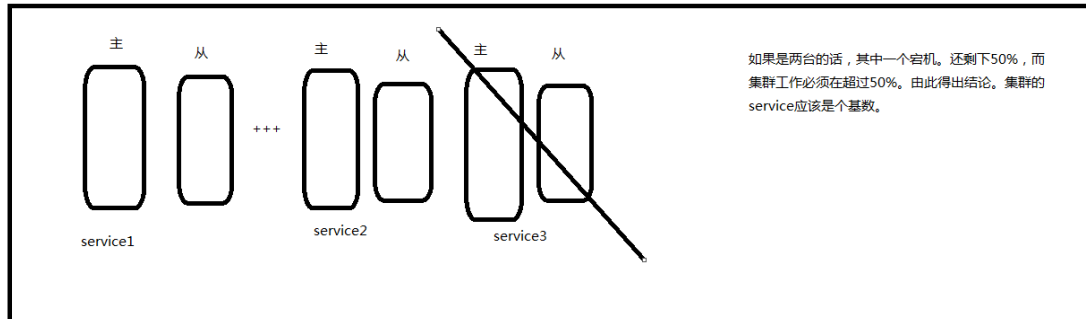
为什么是 6 个配置文件

redis 集群，相对于项目来说，是保证项目能够正常运行！

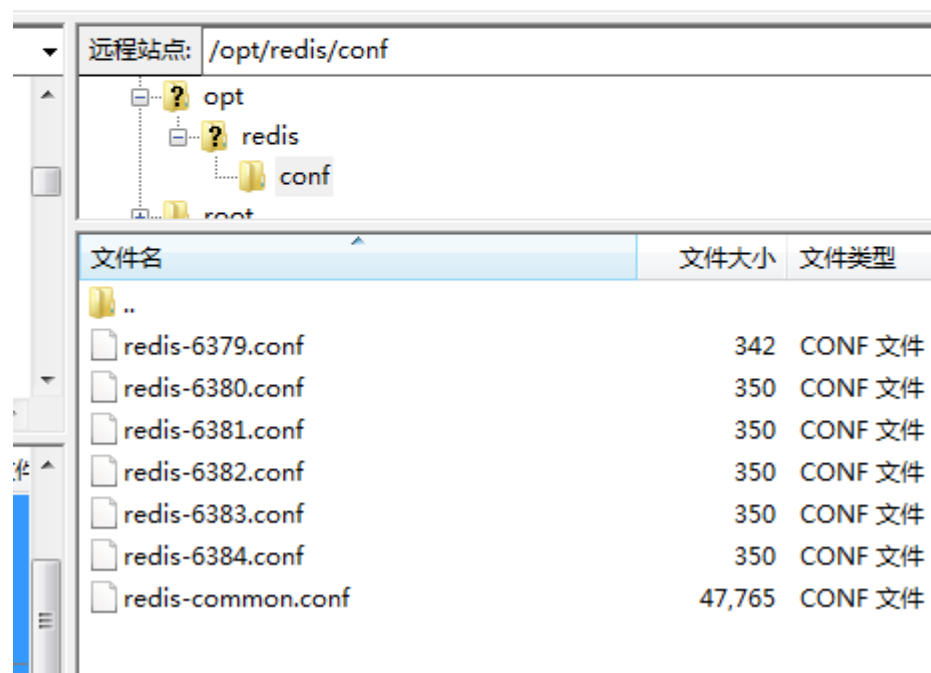
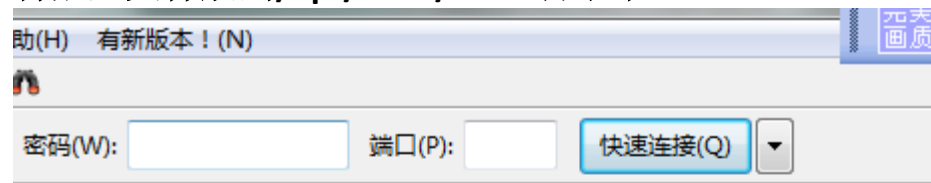
每个集群都有一主一从：也就是说要有两个服务参与。

集群是采用选举方式来的！也就是说每次选举的结果应该大于 50% 的时候，集群才能够正常运行工作！否则集群就会失败！

比如：如果有只有两个？集群就不能正常工作了。



将配置文件放到/opt/redis/conf 目录中



启动配置好的集群配置文件

```
root@localhost bin# ps -ef | grep redis
root      3334      1  0 12:22 ?        00:00:19 ./redis-server *:6379
root      4150    3290  0 15:33 pts/0    00:00:00 grep redis
root@localhost bin# kill -9 3334
root@localhost bin# ps -ef | grep redis
root      4152    3290  0 15:34 pts/0    00:00:00 grep redis
root@localhost bin# ll
总用量 26400
-rw-r--r--. 1 root root    670 7月 27 14:45 dump.rdb
-rwxr-xr-x. 1 root root 5580327 7月 27 10:51 redis-benchmark
-rwxr-xr-x. 1 root root  22217 7月 27 10:51 redis-check-aof
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-check-rdb
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rw-r--r--. 1 root root  46688 7月 27 11:44 redis.conf
lrwxrwxrwx. 1 root root    12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
root@localhost bin# ./redis-server /opt/redis/conf/redis-6379.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6380.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6381.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6382.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6383.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6384.conf
root@localhost bin#
```

启动集群配置文件

vim startup.sh

./redis-server /opt/redis/conf/redis-6379.conf

./redis-server /opt/redis/conf/redis-6380.conf

./redis-server /opt/redis/conf/redis-6381.conf

./redis-server /opt/redis/conf/redis-6382.conf

./redis-server /opt/redis/conf/redis-6383.conf

./redis-server /opt/redis/conf/redis-6384.conf

chmod +x startup.sh

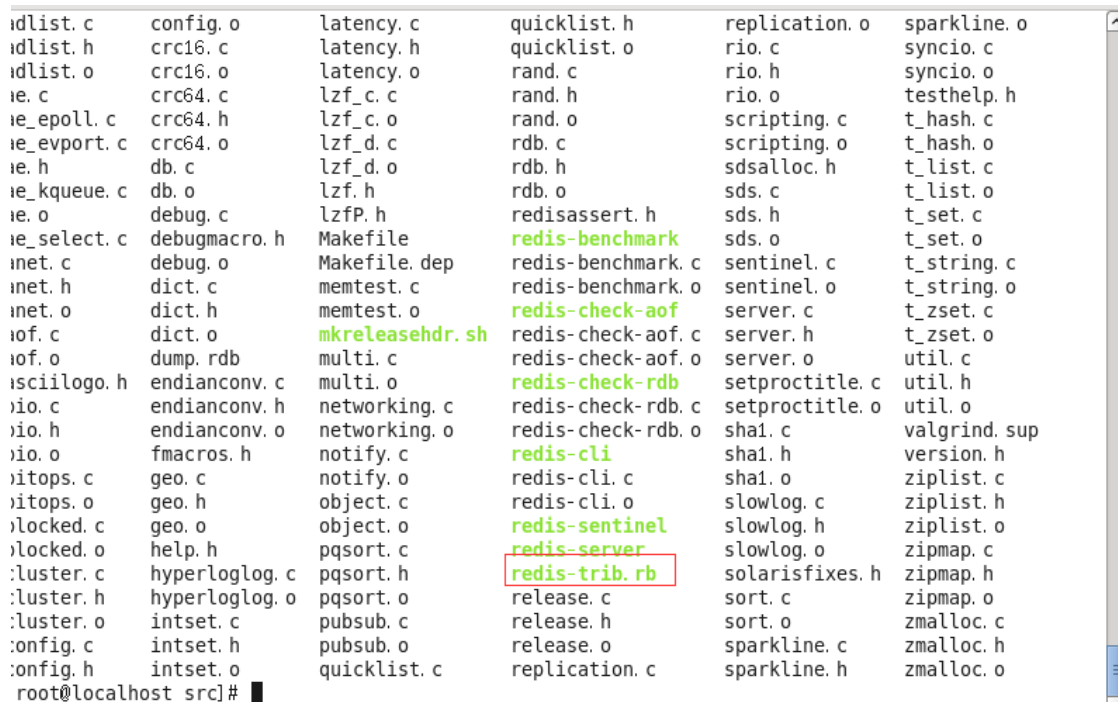
查看一下是否启动成功,相当于 6 个服务已经启动成功

ps -ef | grep redis

```
[root@localhost bin]# ps -ef | grep redis
root      4157      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6379 [cluster]
root      4161      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6380 [cluster]
root      4165      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6381 [cluster]
root      4169      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6382 [cluster]
root      4173      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6383 [cluster]
root      4177      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6384 [cluster]
root      4185    3290  0 15:37 pts/0    00:00:00 grep redis
[root@localhost bin]#
```

将 6 个 redis 服务通过 ruby 脚本来管理，实现集群。

```
cd /root/redis-3.2.8/src
```



```
idlist.c      config.o      latency.c     quicklist.h   replication.o  sparkline.o
idlist.h      crc16.c       latency.h     quicklist.o    rio.c          syncio.c
idlist.o      crc16.o       latency.o     rand.c         rio.h          syncio.o
ie.c          crc64.c       lzf_c.c      rand.h         rio.o          testhelp.h
ie_epoll.c    crc64.h       lzf_c.o      rand.o         scripting.c    t_hash.c
ie_evport.c   crc64.o       lzf_d.c      rdb.c          scripting.o    t_hash.o
ie.h          db.c          lzf_d.o      rdb.h          sdsalloc.h    t_list.c
ie_queue.c    db.o          lzf.h        rdb.o          sds.c         t_list.o
ie.o          debug.c       lzfp.h       redisassert.h  sds.h         t_set.c
ie_select.c   debugmacro.h Makefile      redis-benchmark sds.o         t_set.o
inet.c        debug.o       Makefile.dep redis-benchmark.c sentinel.c     t_string.c
inet.h        dict.c        memtest.c    redis-benchmark.o sentinel.o     t_string.o
inet.o        dict.h        memtest.o    redis-check-aof redis-check-aof.c server.c       t_zset.c
iof.c         dict.o        mkreleasehdr.sh redis-check-aof.o server.h       t_zset.o
iof.o         dump.rdb      multi.c      redis-check-rdb redis-check-rdb.o server.o       util.c
iscilogo.h    endianconv.c multi.o       redis-check-rdb.c setproctitle.c util.h
io.c          endianconv.h networking.c  redis-check-rdb.o sha1.c        setproctitle.o util.o
io.h          endianconv.o networking.o notify.c      sha1.h        sha1.o        valgrind.sup
io.o          fmacros.h     notify.o     sha1.o         slowlog.c     version.h
itops.c       geo.c         notify.o     sha1.o         slowlog.h     ziplist.c
itops.o       geo.h         object.c     sha1.o         slowlog.o     ziplist.h
llocked.c     geo.o         object.o     slowlog.c      solarisfixes.h zipmap.c
llocked.o     help.h        pqsort.c     sort.c         sort.o        zipmap.h
luster.c      hyperloglog.c pqsort.h     sort.o         sparkline.c   zmalloc.c
luster.h      hyperloglog.o pqsort.o     sparkline.h    sparkline.c   zmalloc.h
luster.o      intset.c      pubsub.c     sparkline.o    sparkline.h   zmalloc.o
onfig.c       intset.h      pubsub.o     replication.c
onfig.h       intset.o      quicklist.c
root@localhost src#
```

将 ruby 脚本拷贝到 redis 安装目录

```
cp redis-trib.rb /usr/local/redis/bin/
```

开始创建集群

```
./redis-trib.rb create --replicas 1 192.168.26.30:6379 192.168.26.30:6380 192.168.26.30:6381
192.168.26.30:6382 192.168.26.30:6383 192.168.26.30:6384
```

问一句：192.168.26.30 = 127.0.0.1？

必须写虚拟机的真实 ip 地址！否则创建失败！

```
应用程序 位置 系统 四 月 27 15:44 root
root@localhost:/usr/local/redis/bin
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rw-r--r--. 1 root root 46688 7月 27 11:44 redis.conf
lrwxrwxrwx. 1 root root 12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
-rwxr-xr-x. 1 root root 60852 7月 27 15:41 redis-trib.rb
[root@localhost bin]# ./redis-trib.rb create --replicas 1 192.168.26.30:6379 192.168.26.30:6380 192.168.26.30:6381 192.168.26.30:6382 192.168.26.30:6383 192.168.26.30:6384
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes... 创建集群命令
Using 3 masters:
192.168.26.30:6379
192.168.26.30:6380
192.168.26.30:6381
Adding replica 192.168.26.30:6382 to 192.168.26.30:6379 M: 代表集群的主
Adding replica 192.168.26.30:6383 to 192.168.26.30:6380 S:表示集群从
Adding replica 192.168.26.30:6384 to 192.168.26.30:6381
M: 74f8cb0e161b2267b67ea048b21258b78f515600 192.168.26.30:6379
slots:0-5460 (5461 slots) master
M: 8f7e65ff0aec38770c26aa2d3c461dd9b899d036 192.168.26.30:6380
slots:5461-10922 (5462 slots) master
M: 3066add8fd6f747cefebd5e6d6ca098f32f0a78 192.168.26.30:6381
slots:10923-16383 (5461 slots) master
S: fa8b368f7f7ca5c32603fb6b3a0709318e8d2441 192.168.26.30:6382
replicates 74f8cb0e161b2267b67ea048b21258b78f515600
S: 7d426aa3c7d4e0764e49c24cda3347b2d959f246 192.168.26.30:6383
replicates 8f7e65ff0aec38770c26aa2d3c461dd9b899d036
S: 1e5dd8b9bd18119167dd7b717707d114c864af1c 192.168.26.30:6384
replicates 3066add8fd6f747cefebd5e6d6ca098f32f0a78
Can I set the above configuration? (type 'yes' to accept):
Waiting for the cluster to join....
>>> Performing Cluster Check (using node 192.168.26.30:6379)
M: 74f8cb0e161b2267b67ea048b21258b78f515600 192.168.26.30:6379
slots:0-5460 (5461 slots) master
1 additional replica(s)
S: 7d426aa3c7d4e0764e49c24cda3347b2d959f246 192.168.26.30:6383
slots: (0 slots) slave
replicates 8f7e65ff0aec38770c26aa2d3c461dd9b899d036
S: 1e5dd8b9bd18119167dd7b717707d114c864af1c 192.168.26.30:6384
slots: (0 slots) slave
replicates 3066add8fd6f747cefebd5e6d6ca098f32f0a78
M: 3066add8fd6f747cefebd5e6d6ca098f32f0a78 192.168.26.30:6381
slots:10923-16383 (5461 slots) master
1 additional replica(s)
M: 8f7e65ff0aec38770c26aa2d3c461dd9b899d036 192.168.26.30:6380
slots:5461-10922 (5462 slots) master
1 additional replica(s)
S: fa8b368f7f7ca5c32603fb6b3a0709318e8d2441 192.168.26.30:6382
slots: (0 slots) slave
replicates 74f8cb0e161b2267b67ea048b21258b78f515600
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@localhost bin]#
```


测试集群

```
./redis-cli -c -p 6379
```

-c：表示集群

-p：端口号

面试官：

Redis 服务是通过什么，实现在每个端口之间互相交互的？

集群原理：

Slots:(槽) redis 总共有 16384 个槽 每个服务器所占的槽的区间不一样。

Service1-6379 : 0-5460

Service2-6380 : 5461-10922

Service3-6381 : 10923-16383

set name admin , crc16 算法。返回一个槽的位置【5798】，判断你返回这个槽在那个区间？

$\text{crc16}(\text{admin}) = \text{值} / 16384 = 5798.$

对应的区间，就是对应的服务，无论是存，还是取都会遵循该算法！

关掉集群：实际就是关掉 redis 服务。 Kill 掉 redis 服务即可

```
killall redis-server
```

集群中写入数据

客户端重定向

①在 redis-cli 每次录入、查询键值，redis 都会计算出该 key 应该送往的插槽，如果不是该客户端对应服务器插槽，redis 会报错，并告知应前往的 redis 实例地址和端口。

②redis-cli 客户端提供了 -c 参数实现自动重定向。如 redis-cli -c -p 6379 登入后，再录入、查询键值对可以自动重定向。

```
# redis-cli -c -p 6379
```

③每个 slot 可以存储一批键值对。

```
127.0.0.1:6379> set name admin
-> Redirected to slot [5798] located at 192.168.67.201:6383
OK
192.168.67.201:6383> █
```

如何多键操作

采用哈希算法后,会自动地分配 slot,而 不在一个 slot 下的键值,是不能使用 mget,mset 等多键操作。

如果有需求,需要将一批业务数据一起插入呢?

解决:可以通过`{}`来定义组的概念,从而使 key 中`{}`内相同内容的键值对放到一个 slot 中去。

```
192.168.67.201:6383> mset k11 v11 k12 v12 k13 v13
(error) CROSSSLOT Keys in request don't hash to the same slot
192.168.67.201:6383> mset {key}k11 v11 {key}k12 v12 {key}k13 v13
-> Redirected to slot [12539] located at 192.168.67.201:6381
OK
192.168.67.201:6381> mget{key}
(error) ERR unknown command 'mget{key}'
192.168.67.201:6381> mget {key}k11 {key}k12
1) "v11"
2) "v12"
192.168.67.201:6381> █
```

集群中故障恢复

1、主机挂掉,从机会上位

```
[root@localhost bin]# ./redis-cli -h 192.168.67.201 -p 6380 shutdown
[root@localhost bin]# █
```

`./redis-cli -h:` 表示 host !

```
9b4d2bf35b060e550a412b970f8cad8960df3482 192.168.67.201:6381 master - 0 1527741697260 3 connected 10923-16383
7dfe2ad9a8668b24a768b0194c3efb52f3f33f7e 192.168.67.201:6380 master, fail - 1527741650334 1527741644793 2 disconnected
f543cdb108f76377a50c04e133ec8650141ac952 192.168.67.201:6383 master - 0 1527741692210 7 connected 5461-10922
ac72adc1b3ce1d8b6448d00a4cbda2a5681133fa 192.168.67.201:6379 myself, master - 0 0 1 connected 0-5460
67e57e54eb315c2ac850f7e0aafa81f1ef0671a8 192.168.67.201:6382 slave ac72adc1b3ce1d8b6448d00a4cbda2a5681133fa 0 1527741696250 4 connect
8fe2b771b4390de010464953b9f1ffa27a8a0167 192.168.67.201:6384 slave 9b4d2bf35b060e550a412b970f8cad8960df3482 0 1527741698270 6 connect
127.0.0.1:6379> █
```

2、主节点恢复后如何

```
[root@localhost bin]# ./redis-cli -h 192.168.67.201 -p 6380 shutdown
[root@localhost bin]# ./redis-server /opt/redis/conf/redis-6380.conf
```

```
7dfe2ad9a8668b24a768b0194c3efb52f3f33f7e 192.168.67.201:6380 slave f543cdb108f76377a50c04e133ec8650141ac952 0 1527741983907 7 connected
f543cdb108f76377a50c04e133ec8650141ac952 192.168.67.201:6383 master - 0 1527741988941 7 connected 5461-10922
```

50

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载, 可访问百度: [尚硅谷官网](#)

当主节点：挂掉之后，从节点上位！当主节点恢复之后，不会成为主，应该改成从节点！

== 集群的从节点与主从复制的从节点有所有区别！

集群的从节点作用：当主节点挂掉，则会立即上位。

主从复制的从节点：只负责读取数据

集群的 Jedis 开发

```
@Test
    public void testCluster() {
        Set<HostAndPort> jedisClusterNodes = new
HashSet<HostAndPort>();
        //Jedis Cluster will attempt to discover cluster nodes
        automatically
        jedisClusterNodes.add(new HostAndPort("192.168.4.128", 6379));
        JedisCluster jc = new JedisCluster(jedisClusterNodes);
        jc.set("foo", "bar");
        String value = jc.get("foo");
    }
```

集群的优缺点

优点：

- 实现扩容
- 分摊压力
- 无中心配置相对简单

缺点：

- 多键操作是不被支持的
- 多键的 Redis 事务是不被支持的。lua 脚本不被支持。
- 由于集群方案出现较晚，很多公司已经采用了其他的集群方案，而代理或者客户端分片的方案想要迁移至 redis cluster，需要整体迁移而不是逐步过渡，复杂度较大。

使用 eclipse 连接集群，进行数据操作

假如说，明天在使用集群的话。还需要创建么？NO

只需要启动集群即可 `/usr/local/redis/bin ./start.sh`

1. 启动集群

启动集群：也就是启动多个 redis 服务，而 redis 服务之前我们已经配置好了。`/opt/redis/conf`

该目录下：`redis-6379.conf`---`redis-6384.conf`

```
lrwxrwxrwx. 1 root root      12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
-rwxr-xr-x. 1 root root  60852 7月 27 15:41 redis-trib.rb
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6379.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6380.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6381.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6382.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6383.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6384.conf
root@localhost bin]# ps -ef | grep redis
root      3376      1  0 09:33 ?        00:00:00 ./redis-server 0.0.0.0:6379 [c
luster]
root      3380      1  0 09:33 ?        00:00:00 ./redis-server 0.0.0.0:6380 [c
luster]
root      3384      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6381 [c
luster]
root      3388      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6382 [c
luster]
root      3394      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6383 [c
luster]
root      3401      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6384 [c
luster]
root      3407    3360  0 09:34 pts/0    00:00:00 grep redis
root@localhost bin]#
```

2. 使用 eclipse 连接集群。

第一步： 应该要关闭防火墙， service iptables stop

第二步：

```
public static void main(String[] args) {  
    // 创建多个redis服务的host和port  
    Set<HostAndPort> set = new HashSet<>();  
    set.add(new HostAndPort("192.168.26.30", 6379));  
    set.add(new HostAndPort("192.168.26.30", 6380));  
    set.add(new HostAndPort("192.168.26.30", 6381));  
    set.add(new HostAndPort("192.168.26.30", 6382));  
    set.add(new HostAndPort("192.168.26.30", 6383));  
    set.add(new HostAndPort("192.168.26.30", 6384));  
    // 创建一个JedisCluster 对象 并将set集合中的host和port给  
    redis集群对象  
    JedisCluster jc = new JedisCluster(set);  
    jc.set("one", "小骨");  
    System.out.println(jc.get("one"));  
}
```

注意：导入 **redis.jar,commons-pool.jar**

使用 spring 整合 redis

1. 导入 jar 包：spring 和 redis

```
commons-logging-1.1.3.jar  
commons-pool2-2.4.2.jar  
jedis-2.9.0.jar  
junit-4.8.jar  
spring-aop-4.1.6.RELEASE.jar
```

```
spring-aspects-4.1.6.RELEASE.jar
spring-beans-4.1.6.RELEASE.jar
spring-context-4.1.6.RELEASE.jar
spring-context-support-4.1.6.RELEASE.jar
spring-core-4.1.6.RELEASE.jar
spring-expression-4.1.6.RELEASE.jar
spring-instrument-4.1.6.RELEASE.jar
spring-instrument-tomcat-4.1.6.RELEASE.jar
spring-jdbc-4.1.6.RELEASE.jar
spring-jms-4.1.6.RELEASE.jar
spring-messaging-4.1.6.RELEASE.jar
spring-orm-4.1.6.RELEASE.jar
spring-oxm-4.1.6.RELEASE.jar
spring-test-4.1.6.RELEASE.jar
spring-tx-4.1.6.RELEASE.jar
spring-web-4.1.6.RELEASE.jar
spring-webmvc-4.1.6.RELEASE.jar
spring-webmvc-portlet-4.1.6.RELEASE.jar
spring-websocket-4.1.6.RELEASE.jar
```

2. 配置 spring 的核心配置文件

Spring 核心配置文件的主要工作就是将 JedisPool 类交给 spring 容器来管理。

redis.clients.jedis.JedisCluster 也要交给 spring 容器来管理。

具体配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
```

```
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
4.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">
<!-- 配置jedis连接池 -->
<bean id="jedisPoolConfig"
class="redis.clients.jedis.JedisPoolConfig">
    <!-- 最大连接数 -->
    <property name="maxTotal" value="30" />
    <!-- 最大空闲连接数 -->
    <property name="maxIdle" value="10" />
    <!-- 每次释放连接的最大数目 -->
    <property name="numTestsPerEvictionRun" value="1024" />
    <!-- 释放连接的扫描间隔（毫秒） -->
    <property name="timeBetweenEvictionRunsMillis"
value="30000" />
    <!-- 连接最小空闲时间 -->
```

```
<property name="minEvictableIdleTimeMillis" value="1800000"
/>

<!-- 连接空闲多久后释放，当空闲时间>该值 且 空闲连接>最大空闲连接
数 时直接释放 -->

<property name="softMinEvictableIdleTimeMillis"
value="10000" />

<!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -
->

<property name="maxWaitMillis" value="1500" />

<!-- 在获取连接的时候检查有效性, 默认false -->

<property name="testOnBorrow" value="true" />

<!-- 在空闲时检查有效性, 默认false -->

<property name="testWhileIdle" value="true" />

<!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true
-->

<property name="blockWhenExhausted" value="false" />

</bean>

<!-- jedis整合spring单机版 -->

<!-- <bean id="jedisClient"
class="redis.clients.jedis.JedisPool">

    <constructor-arg name="host" value="192.168.26.128"/>

    <constructor-arg name="port" value="6379"/>

    <constructor-arg name="poolConfig" ref="jedisPoolConfig"/>

</bean> -->

<!-- jedisCluster JedisCluster jedisClients = new JedisCluster
```



```
-->
    <bean id="jedisClients"
class="redis.clients.jedis.JedisCluster">
    <!-- ref引用了上面的配置 -->
        <constructor-arg name="poolConfig" ref="jedisPoolConfig"/>
        <constructor-arg name="nodes">
            <set>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6379"/>
                </bean>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6380"/>
                </bean>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6381"/>
                </bean>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6382"/>
                </bean>
            </set>
        </constructor-arg>
    </bean>
```

```

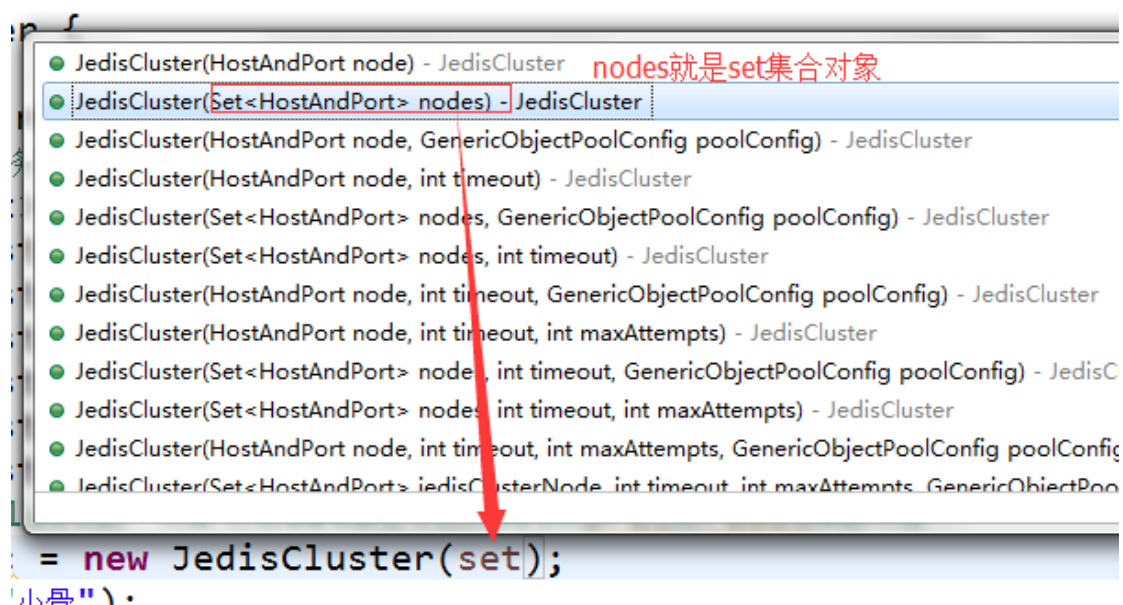
        <bean class="redis.clients.jedis.HostAndPort">
            <constructor-arg name="host"
value="192.168.26.30"/>
            <constructor-arg name="port" value="6383"/>
        </bean>
        <bean class="redis.clients.jedis.HostAndPort">
            <constructor-arg name="host"
value="192.168.26.30"/>
            <constructor-arg name="port" value="6384"/>
        </bean>
    </set>
</constructor-arg>
</bean>
</beans>

```

<bean id="jedisClients" class="redis.clients.jedis.JedisCluster"> 相当于:

JedisCluster jedisClients = new JedisCluster();

<constructor-arg name="nodes"> 相当于:



```
<set>

    <bean class="redis.clients.jedis.HostAndPort">
        <constructor-arg name="host"
value="192.168.26.30"/>
        <constructor-arg name="port" value="6379"/>
    </bean>

</set>
```

相当于：给 set 集合中赋予一个 HostAndPort 对象

```
<constructor-arg name="host" value="192.168.26.30"/>
    <constructor-arg name="port" value="6379"/>
```

通过构造器方式对 HostAndPort 进行初始化数据

```
// 创建多个redis服务的host和port
Set<HostAndPort> set = new HashSet<>();
set.add(new HostAndPort("192.168.26.30", 6379));
set.add(new HostAndPort("192.168.26.30", 6380));
set.add(new HostAndPort("192.168.26.30", 6381));
set.add(new HostAndPort("192.168.26.30", 6382));
set.add(new HostAndPort("192.168.26.30", 6383));
set.add(new HostAndPort("192.168.26.30", 6384));
```

3. 编写测试类

```
public static void main(String[] args) {

    // 读取 spring 的配置文件

    ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext-jedis.xml");

    JedisCluster jc = (JedisCluster) ac.getBean("jedisClients");

    jc.set("cluster", "集群");

    System.out.println(jc.get("cluster"));

}
```

Redis

版本：V 1.0

Redis 持久化

Redis 主要是工作在内存中。内存本身就不是一个持久化设备，断电后数据会清空。所以 Redis 在工作过程中，如果发生了意外停电事故，如何尽可能减少数据丢失。

Redis 持久化

Redis 提供了不同级别的持久化方式：

- RDB持久化方式能够在指定的时间间隔对你的数据进行快照存储。
- AOF持久化方式记录每次对服务器写的操作,当服务器重启的时候会重新执行这些命令来恢复原始的数据,AOF命令以re协议追加保存每次写的操作到文件末尾.Redis还能对AOF文件进行后台重写,使得AOF文件的体积不至于过大。
- 如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式。
- 你也可以同时开启两种持久化方式,在这种情况下,当redis重启的时候会优先载入AOF文件来恢复原始的数据,因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整。
- 最重要的事情是了解RDB和AOF持久化方式的不同,让我们以RDB持久化方式开始:

1. RDB

1.1 RDB 简介

RDB: 在指定的**时间间隔内**将内存中的数据集快照**写入磁盘**,也就是行话讲的 Snapshot 快照,它恢复时是将快照文件直接读到内存里。

工作机制: 每隔一段时间,就把内存中的数据保存到硬盘上的指定文件中。

RDB 是默认开启的!

Redis 会单独创建(fork)一个子进程来进行持久化,会先将数据写入到一个临时文件中【xxx.rdb】,待持久化过程都结束了,再用这个临时文件替换上次持久化好的文件。整个过程中,主进程是不进行任何 IO 操作的,这就确保了极高的性能如果需要进行大规模数据的恢复,且对于数据恢复的完整性不是非常敏感,那 RDB 方式要比 AOF 方式更加的高效。

RDB 的缺点是最后一次持久化后的数据可能丢失。

1.2 RDB 保存策略

save 900 1 900 秒内如果至少有 1 个 key 的值变化，则保存

save 300 10 300 秒内如果至少有 10 个 key 的值变化，则保存

save 60 10000 60 秒内如果至少有 10000 个 key 的值变化，则保存

save "" 就是禁用 RDB 模式；

1.3 RDB 常用属性配置

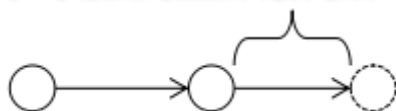
属性	含义	备注
save	保存策略	
dbfilename	RDB 快照文件名	
dir	RDB 快照保存的目录	必须是一个目录，不能是文件名。最好改为固定目录。默认为./代表执行 redis-server 命令时的当前目录！
stop-writes-on-bgsave-error	是否在备份出错时，继续接受写操作	如果用户开启了 RDB 快照功能，那么在 redis 持久化数据到磁盘时如果出现失败，默认情况下，redis 会停止接受所有的写请求
rdbcompression	对于存储到磁盘中的快照，可以设置是否进行压缩存储。	如果是的话，redis 会采用 LZF 算法进行压缩。如果你不想消耗 CPU 来进行压缩的话， 可以设置为关闭此功能，但是存储在磁盘上的快照会比较大。
rdbchecksum	是否进行数据校验	在存储快照后，我们还可以让 redis 使用 CRC64 算法来进行数据校验，但是

		<p>这样做会增加大约 10%的性能消耗，</p> <p>如果希望获取到最大的性能提升，</p> <p>可以关闭此功能。</p>
--	--	-------------------------------------------------------------------------

1.4 RDB 数据丢失的情况

两次保存的时间间隔内，服务器宕机，或者发生断电问题。

有可能发生数据丢失的时间段



1.5 RDB 的触发

设置 `save 30 10` 条的时候触发

在指定范围内向数据库添加数据

- ⑥ 拷贝 rdb 文件 `cp dump.rdb dump_bak.rdb`
- ⑦ 执行 `flushdb`，再执行 `shutdown` 命令，也会产生 `dump.rdb`，但里面是空的，没有意义。空的 `dump.rdb` 默认大小 76
- ⑧ 当执行 `shutdown` 命令时，也会主动地备份数据。
- ⑨ 重新将赋值的 rdb 文件数据在拷贝回去。在进行连接测试。则数据产生了。
- ⑩ **注意：使用 rdb 备份的时候，如果使用了 `flushdb`，则数据将会被取消！**

1.6 RDB 的优缺点

RDB的优点

- RDB是一个非常紧凑的文件,它保存了某个时间点得数据集,非常适用于数据集的备份,比如你可以在每个小时报保存一下过去24小时内的数据,同时每天保存过去30天的数据,这样即使出了问题你也可以根据需求恢复到不同版本的数据集.
- RDB是一个紧凑的单一文件,很方便传送到另一个远端数据中心或者亚马逊的S3(可能加密),非常适用于灾难恢复.
- RDB在保存RDB文件时父进程唯一需要做的就是fork出一个子进程,接下来的工作全部由子进程来做,父进程不需要再做其他IO操作,所以RDB持久化方式可以最大化redis的性能.
- 与AOF相比,在恢复大的数据集的时候,RDB方式会更快一些.

RDB的缺点

- 如果你希望在redis意外停止工作(例如电源中断)的情况下丢失的数据最少的话,那么RDB不适合你.虽然你可以配置不同的save时间点(例如每隔5分钟并且对数据集有100个写的操作),是Redis要完整的保存整个数据集是一个比较繁重的工作,你通常会每隔5分钟或者更久做一次完整的保存,万一在Redis意外宕机,你可能会丢失几分钟的数据.
- RDB需要经常fork子进程来保存数据集到硬盘上,当数据集比较大的时候,fork的过程是非常耗时的,可能会导致Redis在一些毫秒级内不能响应客户端的请求.如果数据集巨大并且CPU性能不是很好的情况下,这种情况会持续1秒,AOF也需要fork,但是你可以调节重写日志文件的频率来提高数据集的耐久度.

2. AOF

2.1 AOF 简介

- AOF 是以日志的形式来记录每个写操作,将每一次对数据进行修改,都把新建、修改数据的命令保存到指定文件中.Redis 重新启动时读取这个文件,重新执行新建、修改数据的命令恢复数据。
- 默认不开启,需要手动开启 593 行 appendonly yes
- AOF 文件的保存路径,同 RDB 的路径一致。# dir ./
- AOF 在保存命令的时候,只会保存对数据有修改的命令,也就是写操作!
- 当 RDB 和 AOF 存的不一致的情况下,按照 AOF 来恢复。因为 AOF 是对 RDB 的补充。备份周期更短,也就更可靠。

2.2 AOF 保存策略

appendfsync always: 每次产生一条新的修改数据的命令都执行保存操作;效率低,但是安全!

appendfsync everysec: 每秒执行一次保存操作。如果在未保存当前秒内操作时发生了断电,仍然会导致一部分数据丢失(即 1 秒钟的数据)。

appendfsync no: 从不保存,将数据交给操作系统来处理。更快,也更不安全的选择。

推荐(并且也是默认)的措施为每秒 **fsync** 一次,这种 **fsync** 策略可以兼顾速度和安全性。

2.3 AOF 常用属性

属性	含义	备注
appendonly	是否开启 AOF 功能	默认是关闭的 yes
appendfilename	AOF 文件名称	appendonly.aof
appendfsync	AOF 保存策略	官方建议 everysec
no-appendfsync-on-rewrite	在重写时,是否执行保存策略	执行重写,可以节省 AOF 文件的体积;而且在恢复的时候效率也更高。
auto-aof-rewrite-percentage	重写的触发条件	当目前 aof 文件大小超过上一次重写的 aof 文件大小的百分之多少进行重写
auto-aof-rewrite-min-size	设置允许重写的最小 aof 文件大小	避免了达到约定百分比但尺寸仍然很小的情况还要重写

Aof : 属于实时备份。文件大小,应该比 rdb 要大。

Rdb : 间隔备份。

2.5 AOF 文件的修复

2.5.1 # vim redis.conf

- 2.5.2 开启 aof 配置 `appendonly yes`, 则会产生对应的日志文件 `appendonly.aof`
- 2.5.3 启动空的服务, 写入命令
- 2.5.4 写入完成命令之后, `flushall`, `shutdown`。
- 2.5.5 结果 `error!` 还是空的: `aof` 备份的实在就是记录所有的命令!
- 2.5.6 分析原因: 查看 `vim appendonly.aof` 将最后的 `flushall` 删除, 再重新查询。
- 2.5.7 使用 `vim` 命令在 `appendonly.aof` 添加一些无效信息, 再启动, 并使用客户端连接, 则会发现链接失败!

如果 AOF 文件中出现了残余命令, 会导致服务器无法重启。此时需要借助 `redis-check-aof` 工具来修复!

命令: `redis-check-aof --fix 文件`

注意: 只要是 `aof` 开启, 则无论怎么样都会读取 `aof` 的配置文件。

2.5 AOF 的优缺点

• AOF 优点

- 使用 AOF 会让你的 Redis 更加耐久: 你可以使用不同的 `fsync` 策略: 无 `fsync`, 每秒 `fsync`, 每次写的时候 `fsync`。使用默认的每秒 `fsync` 策略, Redis 的性能依然很好 (`fsync` 是由后台线程进行处理的, 主线程会尽力处理客户端请求), 一旦出现故障, 你最多丢失 1 秒的数据。
- AOF 文件是一个只进行追加的日志文件, 所以不需要写入 `seek`, 即使由于某些原因 (磁盘空间已满, 写的过程中宕机等等) 未执行完整的写入命令, 你也可使用 `redis-check-aof` 工具修复这些问题。
- Redis 可以在 AOF 文件体积变得过大时, 自动地在后台对 AOF 进行重写: 重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。整个重写操作是绝对安全的, 因为 Redis 在创建新 AOF 文件的过程中, 会继续将命令追加到现有的 AOF 文件里面, 即使重写过程中发生停机, 现有的 AOF 文件也不会丢失。而一旦新 AOF 文件创建完毕, Redis 就会从旧 AOF 文件切换到新 AOF 文件, 并开始对新 AOF 文件进行追加操作。
- AOF 文件有序地保存了对数据库执行的所有写入操作, 这些写入操作以 Redis 协议的格式保存, 因此 AOF 文件的内容非常容易被读懂, 对文件进行分析 (`parse`) 也很轻松。导出 (`export`) AOF 文件也非常简单: 举个例子, 如果你不小心执行了 `FLUSHALL` 命令, 但只要 AOF 文件未被重写, 那么只要停止服务器, 移除 AOF 文件末尾的 `FLUSHALL` 命令, 并重启 Redis, 就可以将数据集恢复到 `FLUSHALL` 执行之前的状态。

优点:

- 备份机制更稳健, 丢失数据概率更低
- 可读的日志文本, 通过操作 AOF 稳健, 可以处理误操作【`redis-check-aof --fix 文件`】

• AOF 缺点

- 对于相同的数据集来说，AOF 文件的体积通常要大于 RDB 文件的体积。
- 根据所使用的 fsync 策略，AOF 的速度可能会慢于 RDB。在一般情况下，每秒 fsync 的性能依然非常高，而关闭 fsync 可以让 AOF 的速度和 RDB 一样快，即使在高负荷之下也是如此。不过在处理巨大的写入载入时，RDB 可以提供更有保证的最大延迟时间（latency）。

缺点：

- 比起 RDB 占用更多的磁盘空间
- 恢复备份速度要慢
- 每次读写都同步的话，有一定的性能压力
- 存在个别 Bug，造成恢复不能。

Redis 主从复制

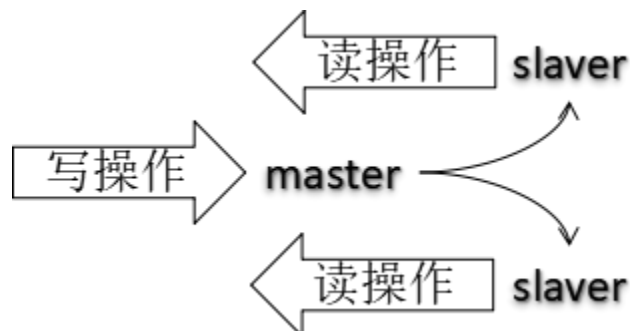
2. 主从简介

配置多台 Redis 服务器，以主机和备机的身份分开。主机数据更新后，根据配置和策略，自动同步到备机的 **master/slaver** 机制，Master 以写为主，Slave 以读为主，二者之间自动同步数据。

目的：

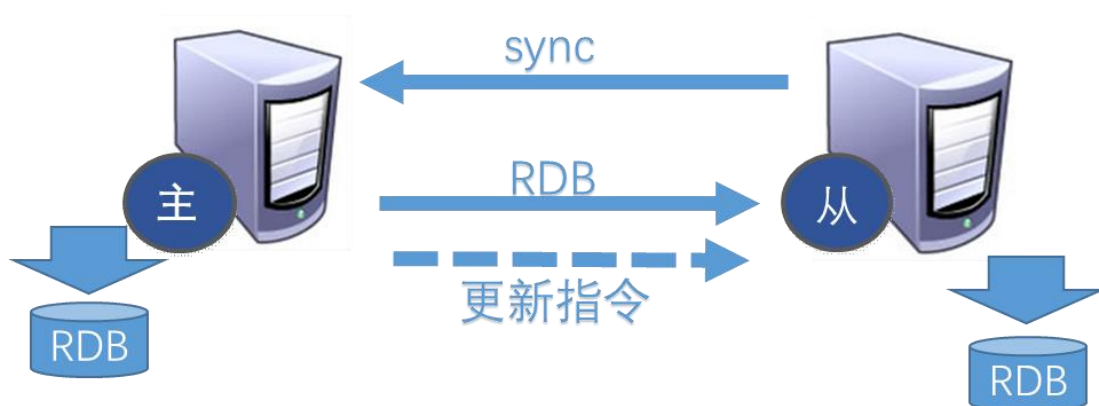
读写分离提高 Redis 性能；

避免单点故障，容灾快速恢复



原理:

每次从机联通后,都会给主机发送 `sync` 指令,主机立刻进行存盘操作,发送 RDB 文件,给从机
从机收到 RDB 文件后,进行全盘加载。之后每次主机的写操作,都会立刻发送给从机,从机执行相同的命令



2. 主从准备

除非是不同的主机配置不同的 Redis 服务,否则在一台机器上面跑多个 Redis 服务,需要配置多个 Redis 配置文件。

①准备多个 Redis 配置文件,每个配置文件,需要配置以下属性

`daemonize yes`: 服务在后台运行

`port`: 端口号

`pidfile:pid` 保存文件

`logfile`: 日志文件(如果没有指定的话,就不需要)

`dump.rdb`: RDB 备份文件的名称

`appendonly` 关掉,或者是更改 `appendonly` 文件的名称。 `aof`

样本:

```
include /usr/local/redis/bin/redis.conf
```

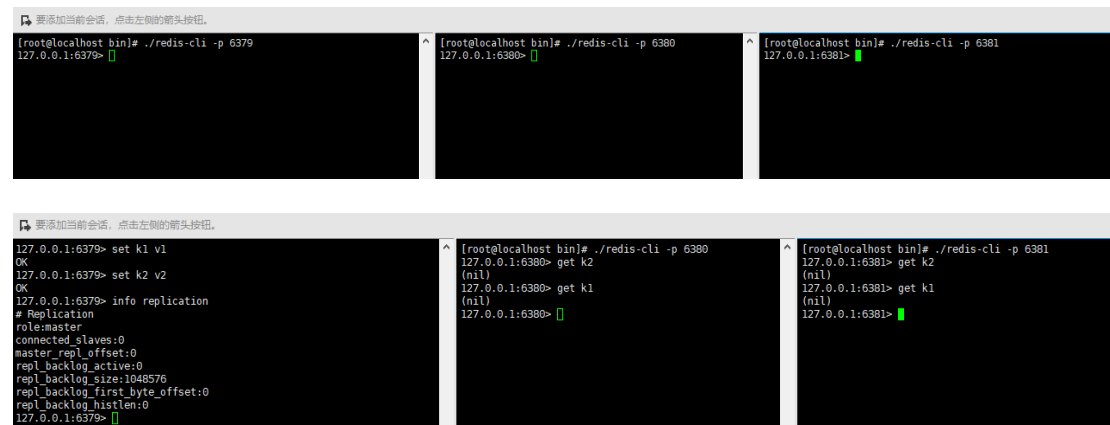
```
port 6379
```

```
pidfile /var/run/redis_6379.pid
```

```
dbfilename dump_6379.rdb
```

②根据多个配置文件，启动多个 Redis 服务

```
[root@localhost bin]# ./redis-server 6379.conf
[root@localhost bin]# ./redis-server 6380.conf
[root@localhost bin]# ./redis-server 6381.conf
[root@localhost bin]# ps -ef |grep redis
root      3548      1   0 10:46 ?        00:00:01 ./re
dis-server *:6379
root      3665      1   0 11:08 ?        00:00:00 ./re
dis-server *:6380
root      3669      1   0 11:08 ?        00:00:00 ./re
dis-server *:6381
root      3675    3449   0 11:08 pts/0    00:00:00 grep
redis
```



The image shows three terminal windows side-by-side, each running a Redis CLI client on a different port. The first window (port 6379) shows the output of 'set k1 v1', 'set k2 v2', and 'info replication'. The second window (port 6380) shows the output of 'get k2' and 'get k1'. The third window (port 6381) shows the output of 'get k2' and 'get k1'.

```
./redis-cli -p 6379
```

-p :表示要连接到哪个端口上的服务器！

3. 主从建立

3.1 临时建立

原则：配从不配主。

配置：在从服务器上执行 **SLAVEOF ip:port** 命令；

查看：执行 **info replication** 命令；

```
127.0.0.1:6380> slaveof 127.0.0.1 6379
OK
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:2
master_sync_in_progress:0
slave_repl_offset:1
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

```
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=71,lag=0
slave1:ip=127.0.0.1,port=6381,state=online,offset=71,lag=0
master_repl_offset:71
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:70
127.0.0.1:6379>

127.0.0.1:6380> slaveof 127.0.0.1 6379
OK
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:10
master_sync_in_progress:0
slave_repl_offset:113
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6380>

127.0.0.1:6381> slaveof 127.0.0.1 6379
OK
127.0.0.1:6381> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:7
master_sync_in_progress:0
slave_repl_offset:127
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6381>
```

确立关系之后，set key value 查看效果。关键是以前的 k1, k2 从机同样能够得到

```
ssh://root***@192.168.67.201:22
要添加当前会话，点击左侧的箭头按钮。

127.0.0.1:6379> set k3 v3
OK
127.0.0.1:6379>

127.0.0.1:6380> get k3
"v3"
127.0.0.1:6380> get k1
"v1"
127.0.0.1:6380>

127.0.0.1:6381> get k3
"v3"
127.0.0.1:6381> get k2
"v2"
127.0.0.1:6381>
```

注意：从机只会读取，不能写入

```
127.0.0.1:6380> set k10 v10
(error) READONLY You can't write against a read only slave.
127.0.0.1:6380>
```

3.2 永久建立

在从机的配置文件中，编写 slaveof 属性配置！

```
# slaveof 127.0.0.1 6379
```

3.3 恢复身份

执行命令 `slaveof no one` 恢复自由身！【反客为主】

4. 主从常见问题

①从机是从头开始复制主机的信息，还是只复制切入以后的信息？

答：从头开始复制，即完全复制。实际上是读取主机的 rdb

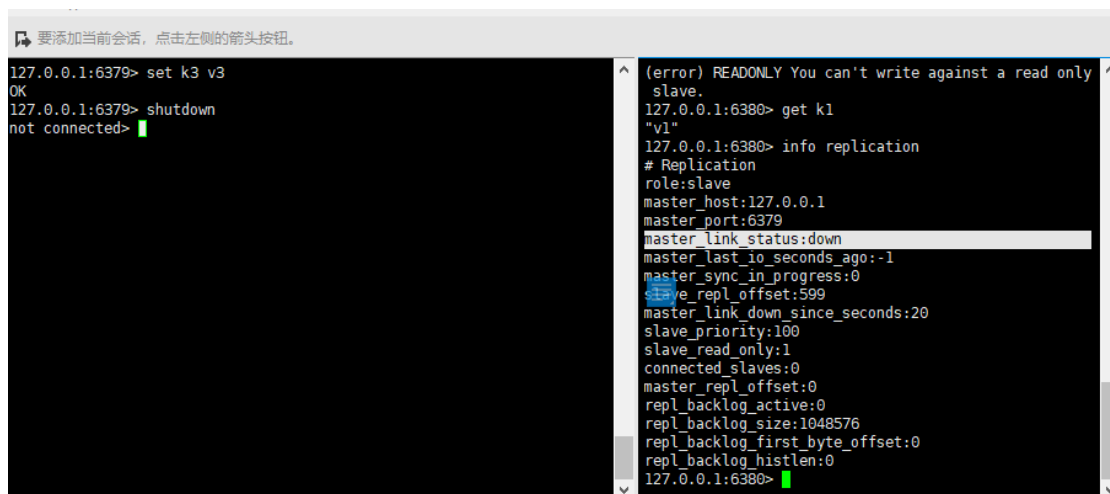
②从机是否可以写？

答：不能

```
127.0.0.1:6381> set k2 v1
(error) READONLY You can't write against a read only slave
```

③主机 shutdown 后，从机是上位还是原地待命？

答：原地待命



```
要添加当前会话，点击左侧的箭头按钮。
127.0.0.1:6379> set k3 v3
OK
127.0.0.1:6379> shutdown
not connected>
127.0.0.1:6380> get k1
"v1"
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:down
master_last_io_seconds_ago:-1
master_sync_in_progress:0
slave_repl_offset:599
master_link_down_since_seconds:20
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6380>
```

④主机又回来了后，主机新增记录，从机还能否顺利复制？

答：可以

```
127.0.0.1:6379> set k3 v3
OK
127.0.0.1:6379> shutdown
not connected>
[root@localhost bin]# ./redis-server 6379.conf
[root@localhost bin]# ./redis-cli
127.0.0.1:6379> set k11 v11
OK
127.0.0.1:6379>

127.0.0.1:6380> get k11
"v11"
127.0.0.1:6380>
```

⑤从机宕机后，重启，宕机期间主机的新增记录，从机是否会顺利复制？

答：可以

```
not connected>
[root@localhost bin]# ./redis-server 6379.conf
[root@localhost bin]# ./redis-cli
127.0.0.1:6379> set k11 v11
OK
127.0.0.1:6379> set k12 v12
OK
127.0.0.1:6379>

(error) ERR unknown command 'shutdown'
127.0.0.1:6380> shutdown
not connected>
[root@localhost bin]# ./redis-server 6380.conf
[root@localhost bin]# ./redis-cli
127.0.0.1:6379> get k12
"v12"
127.0.0.1:6379>

"v2"
127.0.0.1:6381> get k11
"v11"
127.0.0.1:6381> get k12
"v12"
127.0.0.1:6381>
```

⑥其中一台从机 down 后重启，能否重认旧主？

答：不一定，看配置文件中是否配置了 slaveof

slaveof 127.0.0.1 6379

```
[root@localhost bin]# ./redis-server 6380.conf
[root@localhost bin]# ./redis-cli -p 6380
127.0.0.1:6380> set k100 v100
(error) READONLY You can't write against a read only slave.
127.0.0.1:6380>
```

⑦如果两台从机都从主机同步数据，此时主机的 IO 压力会增大，如何解决？

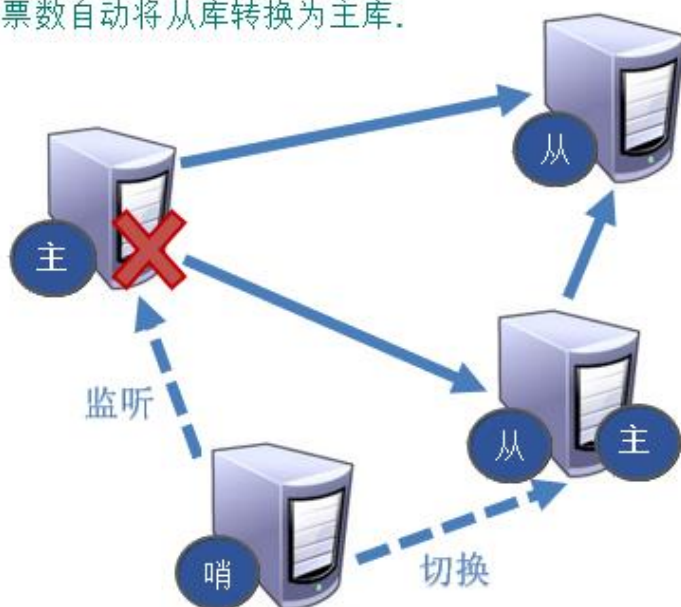
答：按照主---从（主）---从模式配置！[薪火相传]

slaveof 127.0.0.1 6380

5. 哨兵模式

5.1 简介

- 反客为主的自动版，能够后台监控主机是否故障，如果故障了根据投票数自动将从库转换为主库。



作用：

- ① Master 状态检测
- ② 如果 Master 异常，则会进行 Master-Slave 切换，将其中一个 Slave 作为 Master，将之前的 Master 作为 Slave

5.2 配置

自定义的/usr/local/redis/bin 目录下新建 sentinel.conf 文件

```
# vim sentinel.conf
```

哨兵模式需要配置哨兵的配置文件！

```
sentinel monitor mymaster 127.0.0.1 6379 1
```


启动哨兵: `./redis-sentinel sentinel.conf`

```
[root@centos4 redis_replication]# redis-sentinel sentinel.conf
2918:X 11 Apr 00:49:43.127 * Increased maximum number of open files to 10032 (it was originally set to 1024).

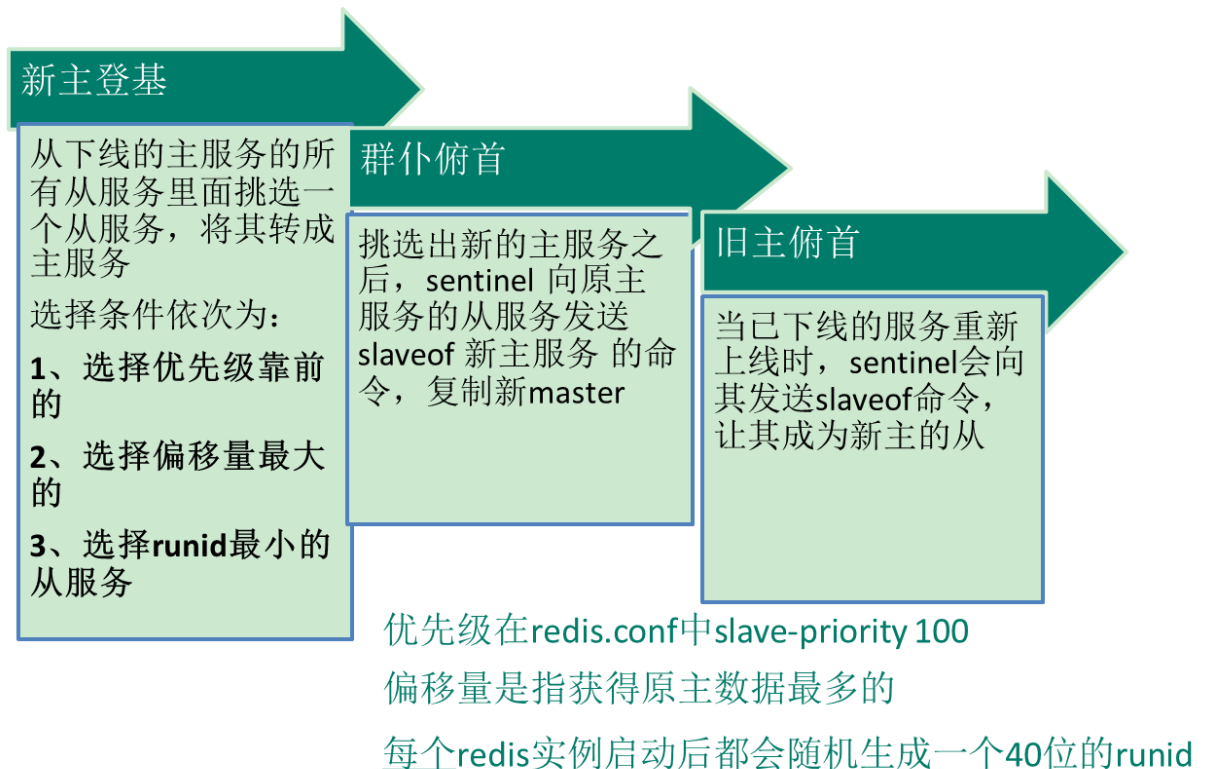
Redis 3.2.5 (00000000/0) 64 bit

Running in sentinel mode
Port: 26379
PID: 2918

http://redis.io

2918:X 11 Apr 00:49:43.129 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
2918:X 11 Apr 00:49:43.141 # Sentinel ID is fc2f5019106049a4b444437b0f4147883d20bffa
2918:X 11 Apr 00:49:43.141 # +monitor master mymaster 127.0.0.1 6379 quorum 1
2918:X 11 Apr 00:49:43.143 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster 127.0.0.1 6379
```

5.3 主机宕机后



```
# slave-priority 100
```

主推:

redis 集群

创建目录

配置文件:

conf 是用来放配置文件

```
mkdir -p opt/redis/conf
```

log 是用来放日志

```
mkdir -p /opt/redis/log
```

data 是用来放集群之后，产生的数据文件。

```
mkdir -p /opt/redis/data
```

```
[root@localhost ~]# mkdir -p /opt/redis/log
[root@localhost ~]# mkdir -p /opt/redis/data
[root@localhost ~]# mkdir -p /opt/redis/conf
[root@localhost ~]# █
```

不行知道用谁来管理集群? ruby!

上传 ruby 脚本：集群工具用来管理集群使用。

```
[ root@localhost ~]# ll
总用量 4776
-rw-----. 1 root root    1612 7月  26 17:57 anaconda-ks.cfg
-rw-r--r--. 1 root root   46478 7月  26 17:57 install.log
-rw-r--r--. 1 root root   10033 7月  26 17:55 install.log.syslog
-rw-r--r--. 1 root root    4142 7月  26 23:17 rbac.sql
drwxrwxr-x. 6 root root    4096 2月  12 23:14 redis-3.2.8
-rw-r--r--. 1 root root 1547237 7月  27 10:32 redis-3.2.8.tar.gz
-rw-r--r--. 1 root root 3226139 7月  27 15:07 ruby-gems.tar.gz
drwxr-xr-x. 2 root root    4096 7月  26 18:08 公共的
drwxr-xr-x. 2 root root    4096 7月  26 18:08 模板
drwxr-xr-x. 2 root root    4096 7月  26 18:08 视频
drwxr-xr-x. 2 root root    4096 7月  26 18:08 图片
drwxr-xr-x. 2 root root    4096 7月  26 18:08 文档
drwxr-xr-x. 2 root root    4096 7月  26 18:08 下载
drwxr-xr-x. 2 root root    4096 7月  26 18:08 音乐
drwxr-xr-x. 3 root root    4096 7月  26 19:18 桌面
[ root@localhost ~]#
```

用来管理集群

解压 ruby 脚本

```
[ root@localhost ~]# tar -zxvf ruby-gems.tar.gz
ruby-gems/
ruby-gems/ruby-rdoc-1.8.7.374-4.el6_6.x86_64.rpm
ruby-gems/ruby-irb-1.8.7.374-4.el6_6.x86_64.rpm
ruby-gems/redis-3.0.6.gem
ruby-gems/compat-readline5-5.2-17.1.el6.x86_64.rpm
ruby-gems/rubygems-1.3.7-5.el6.noarch.rpm
ruby-gems/install.sh
ruby-gems/ruby-1.8.7.374-4.el6_6.x86_64.rpm
ruby-gems/ruby-libs-1.8.7.374-4.el6_6.x86_64.rpm
[ root@localhost ~]# ll
总用量 4780
-rw-----. 1 root root    1612 7月  26 17:57 anaconda-ks.cfg
-rw-r--r--. 1 root root   46478 7月  26 17:57 install.log
-rw-r--r--. 1 root root   10033 7月  26 17:55 install.log.syslog
-rw-r--r--. 1 root root    4142 7月  26 23:17 rbac.sql
drwxrwxr-x. 6 root root    4096 2月  12 23:14 redis-3.2.8
-rw-r--r--. 1 root root 1547237 7月  27 10:32 redis-3.2.8.tar.gz
drwxr-xr-x. 2 root root    4096 8月  23 20:16 ruby-gems
-rw-r--r--. 1 root root 3226139 7月  27 15:07 ruby-gems.tar.gz
drwxr-xr-x. 2 root root    4096 7月  26 18:08 公共的
drwxr-xr-x. 2 root root    4096 7月  26 18:08 模板
drwxr-xr-x. 2 root root    4096 7月  26 18:08 视频
drwxr-xr-x. 2 root root    4096 7月  26 18:08 图片
drwxr-xr-x. 2 root root    4096 7月  26 18:08 文档
drwxr-xr-x. 2 root root    4096 7月  26 18:08 下载
```

进入 ruby-gems 文件中，进行安装

```
[root@localhost ~]# cd ruby-gems
[root@localhost ruby-gems]# ll
总用量 3352
-rw-r--r--. 1 root root 132636 8月 23 2016 compat-readline5-5.2-17.1.el6.x86_64.rpm
-rwxr-xr-x. 1 root root 318 8月 23 2016 install.sh
-rw-r--r--. 1 root root 66048 8月 23 2016 redis-3.0.6.gem
-rw-r--r--. 1 root root 551232 3月 2 2015 ruby-1.8.7.374-4.el6_6.x86_64.rpm
-rw-r--r--. 1 root root 211764 11月 25 2013 rubygems-1.3.7-5.el6.noarch.rpm
-rw-r--r--. 1 root root 324992 8月 23 2016 ruby-irb-1.8.7.374-4.el6_6.x86_64.rpm
-rw-r--r--. 1 root root 1732652 3月 2 2015 ruby-libs-1.8.7.374-4.el6_6.x86_64.rpm
-rw-r--r--. 1 root root 389836 8月 23 2016 ruby-rdoc-1.8.7.374-4.el6_6.x86_64.rpm
[root@localhost ruby-gems]# ./install.sh
Preparing... [100%]
1: compat-readline5 [100%]
Preparing... [100%]
1: ruby-libs [100%]
Preparing... [100%]
1: ruby [100%]
Preparing... [100%]
1: ruby-irb [100%]
Preparing... [100%]
1: ruby-rdoc [100%]
Preparing... [100%]
1: rubygems [100%]
Successfully installed redis-3.0.6
1 gem installed
Installing ri documentation for redis-3.0.6...
Installing RDoc documentation for redis-3.0.6...
^[[A[root@localhost ruby-gems]#
```

整理配置集群的配置文件。

Redis 在运行的时候，是通过处理 `redis.conf` 配置文件来运行的。而 `redis.conf` 配置文件中的每个属性都有其特殊的含义。要想配置集群，则应该会有多个配置文件。集群的每个配置文件中应该都有特殊的设置。比如说 `port` 应该不同。应该共同之处。在配置集群的时候，只要将特殊部分取出来。再有一个公共的配置文件来处理公共的属性。

1-6.配置文件中每个端口号应该不一致！除了端口号之外，可能其他的共同属性，把共同属性放在一个配置文件中即可。7 配置文件。

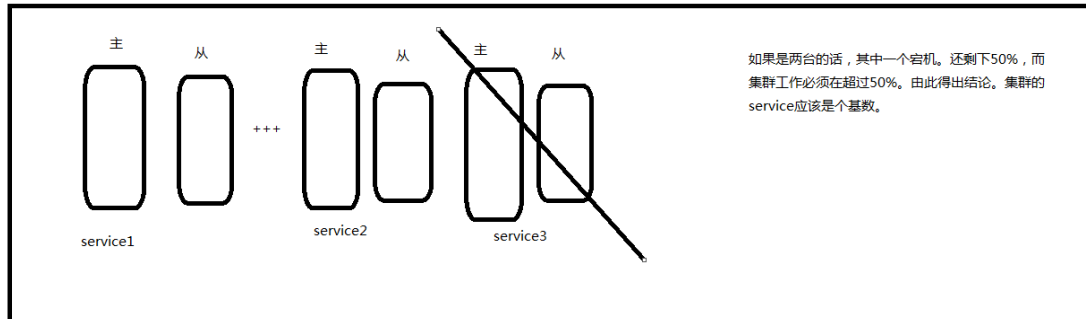
为什么是 6 个配置文件

redis 集群，相对于项目来说，是保证项目能够正常运行！

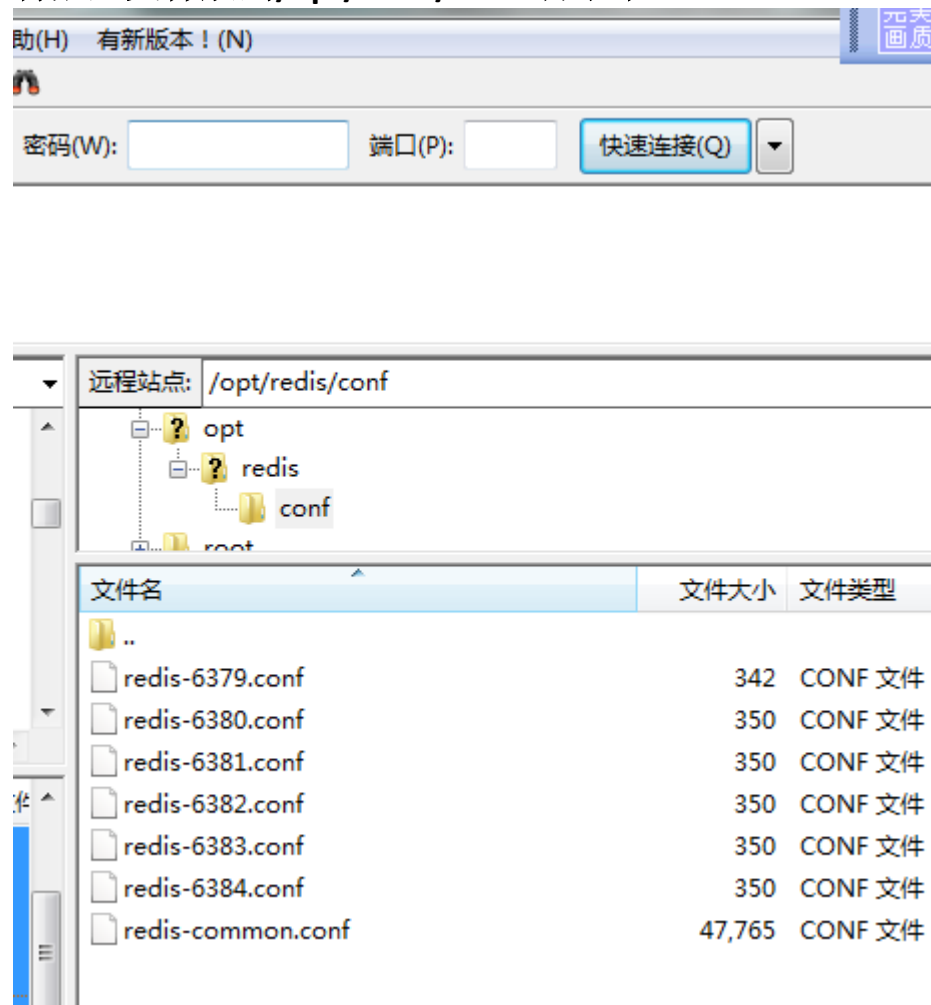
每个集群都有一主一从：也就是说要有两个服务参与。

集群是采用选举方式来的！也就是说每次选举的结果应该大于 50% 的时候，集群才能够正常运行工作！否则集群就会失败！

比如：如果有只有两个？集群就不能正常工作了。



将配置文件放到/opt/redis/conf 目录中



启动配置好的集群配置文件

```
root@localhost bin# ps -ef | grep redis
root      3334      1  0 12:22 ?        00:00:19 ./redis-server *:6379
root      4150    3290  0 15:33 pts/0    00:00:00 grep redis
root@localhost bin# kill -9 3334
root@localhost bin# ps -ef | grep redis
root      4152    3290  0 15:34 pts/0    00:00:00 grep redis
root@localhost bin# ll
总用量 26400
-rw-r--r--. 1 root root    670 7月 27 14:45 dump.rdb
-rwxr-xr-x. 1 root root 5580327 7月 27 10:51 redis-benchmark
-rwxr-xr-x. 1 root root  22217 7月 27 10:51 redis-check-aof
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-check-rdb
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rw-r--r--. 1 root root   46688 7月 27 11:44 redis.conf
lrwxrwxrwx. 1 root root    12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
root@localhost bin# ./redis-server /opt/redis/conf/redis-6379.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6380.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6381.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6382.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6383.conf
root@localhost bin# ./redis-server /opt/redis/conf/redis-6384.conf
root@localhost bin#
```

启动集群配置文件

vim startup.sh

./redis-server /opt/redis/conf/redis-6379.conf

./redis-server /opt/redis/conf/redis-6380.conf

./redis-server /opt/redis/conf/redis-6381.conf

./redis-server /opt/redis/conf/redis-6382.conf

./redis-server /opt/redis/conf/redis-6383.conf

./redis-server /opt/redis/conf/redis-6384.conf

chmod +x startup.sh

查看一下是否启动成功,相当于 6 个服务已经启动成功

ps -ef | grep redis

```
[root@localhost bin]# ps -ef | grep redis
root      4157      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6379 [cluster]
root      4161      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6380 [cluster]
root      4165      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6381 [cluster]
root      4169      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6382 [cluster]
root      4173      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6383 [cluster]
root      4177      1  0 15:35 ?        00:00:00 ./redis-server 0.0.0.0:6384 [cluster]
root      4185    3290  0 15:37 pts/0    00:00:00 grep redis
[root@localhost bin]#
```

将 6 个 redis 服务通过 ruby 脚本来管理，实现集群。

```
cd /root/redis-3.2.8/src
```

```
idlist.c      config.o      latency.c      quicklist.h    replication.o   sparkline.o
idlist.h      crc16.c       latency.h      quicklist.o     rio.c          syncio.c
idlist.o      crc16.o       latency.o      rand.c          rio.h          syncio.o
ie.c          crc64.c       lzf_c.c       rand.h          rio.o          testhelp.h
ie_epoll.c    crc64.h       lzf_c.o       rand.o          scripting.c    t_hash.c
ie_evport.c   crc64.o       lzf_d.c       rdb.c           scripting.o    t_hash.o
ie.h          db.c          lzf_d.o       rdb.h           sdsalloc.h    t_list.c
ie_queue.c    db.o          lzf.h         rdb.o           sds.c         t_list.o
ie.o          debug.c       lzfp.h        redisassert.h   sds.h         t_set.c
ie_select.c   debugmacro.h Makefile       redis-benchmark sds.o         t_set.o
inet.c        debug.o       Makefile.dep  redis-benchmark.c sentinel.c     t_string.c
inet.h        dict.c        memtest.c     redis-benchmark.o sentinel.o     t_string.o
inet.o        dict.h        memtest.o     redis-check-aof redis-check-aof.c server.c       t_zset.c
iof.c         dict.o        mkreleashdr.sh redis-check-aof.c server.h       t_zset.o
iof.o         dump.rdb      multi.c       redis-check-aof.o server.o       util.c
isciiologo.h endianconv.c  multi.o       redis-check-rdb redis-check-rdb.o server.o       util.c
io.c          endianconv.h networking.c   redis-check-rdb.c setproctitle.c util.h
io.h          endianconv.o networking.o  redis-check-rdb.o sha1.c         setproctitle.o util.o
io.o          fmacros.h    notify.c      redis-cli       sha1.h         valgrind.sup
itops.c       geo.c        notify.o      redis-cli.c     sha1.o         version.h
itops.o       geo.h        object.c      redis-cli.o     sha1.o         ziplist.c
llocked.c     geo.o        object.o      redis-sentinel  slowlog.c      ziplist.h
llocked.o     help.h       pqsort.c     redis-server    slowlog.h      ziplist.o
luster.c      hyperloglog.c pqsort.h      redis-trib.rb   slowlog.o      zipmap.c
luster.h      hyperloglog.o pqsort.o      release.c       solarisfixes.h zipmap.h
luster.o      intset.c     pqsub.c      release.h       sort.c          zipmap.o
onfig.c       intset.h     pubsub.c     release.o       sort.o          zmalloc.c
onfig.h       intset.o     quicklist.c  replication.c   sparkline.c    zmalloc.h
root@localhost src#
```

将 ruby 脚本拷贝到 redis 安装目录

```
cp redis-trib.rb /usr/local/redis/bin/
```

开始创建集群

```
./redis-trib.rb create --replicas 1 192.168.26.30:6379 192.168.26.30:6380 192.168.26.30:6381
```

```
192.168.26.30:6382 192.168.26.30:6383 192.168.26.30:6384
```

问一句：192.168.26.30 = 127.0.0.1?

必须写虚拟机的真实 ip 地址！否则创建失败！


```

应用程序 位置 系统 四 月 27 15:44 root
root@localhost:/usr/local/redis/bin
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
-rwxr-xr-x. 1 root root 5709195 7月 27 10:51 redis-cli
-rw-r--r--. 1 root root 46688 7月 27 11:44 redis.conf
lrwxrwxrwx. 1 root root 12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
-rwxr-xr-x. 1 root root 60852 7月 27 15:41 redis-trib.rb
[root@localhost bin]# ./redis-trib.rb create --replicas 1 192.168.26.30:6379 192.168.26.30:6380 192.168.26.30:6381 192.168.26.30:6382 192.168.26.30:6383 192.168.26.30:6384
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes... 创建集群命令
Using 3 masters:
192.168.26.30:6379
192.168.26.30:6380
192.168.26.30:6381
Adding replica 192.168.26.30:6382 to 192.168.26.30:6379 M: 代表集群的主
Adding replica 192.168.26.30:6383 to 192.168.26.30:6380 S:表示集群从
Adding replica 192.168.26.30:6384 to 192.168.26.30:6381
M: 74f8cb0e161b2267b67ea048b21258b78f515600 192.168.26.30:6379
slots:0-5460 (5461 slots) master
M: 8f7e65ff0aec38770c26aa2d3c461dd9b899d036 192.168.26.30:6380
slots:5461-10922 (5462 slots) master
M: 3066add8fd6f747cefebd5e6d6ca098f32f0a78 192.168.26.30:6381
slots:10923-16383 (5461 slots) master
S: fa8b368f7f7ca5c32603fb6b3a0709318e8d2441 192.168.26.30:6382
replicates 74f8cb0e161b2267b67ea048b21258b78f515600
S: 7d426aa3c7d4e0764e49c24cda3347b2d959f246 192.168.26.30:6383
replicates 8f7e65ff0aec38770c26aa2d3c461dd9b899d036
S: 1e5dd8b9bd18119167dd7b717707d114c864af1c 192.168.26.30:6384
replicates 3066add8fd6f747cefebd5e6d6ca098f32f0a78
Can I set the above configuration? (type 'yes' to accept):
Waiting for the cluster to join....
>>> Performing Cluster Check (using node 192.168.26.30:6379)
M: 74f8cb0e161b2267b67ea048b21258b78f515600 192.168.26.30:6379
slots:0-5460 (5461 slots) master
1 additional replica(s)
S: 7d426aa3c7d4e0764e49c24cda3347b2d959f246 192.168.26.30:6383
slots: (0 slots) slave
replicates 8f7e65ff0aec38770c26aa2d3c461dd9b899d036
S: 1e5dd8b9bd18119167dd7b717707d114c864af1c 192.168.26.30:6384
slots: (0 slots) slave
replicates 3066add8fd6f747cefebd5e6d6ca098f32f0a78
M: 3066add8fd6f747cefebd5e6d6ca098f32f0a78 192.168.26.30:6381
slots:10923-16383 (5461 slots) master
1 additional replica(s)
M: 8f7e65ff0aec38770c26aa2d3c461dd9b899d036 192.168.26.30:6380
slots:5461-10922 (5462 slots) master
1 additional replica(s)
S: fa8b368f7f7ca5c32603fb6b3a0709318e8d2441 192.168.26.30:6382
slots: (0 slots) slave
replicates 74f8cb0e161b2267b67ea048b21258b78f515600
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@localhost bin]#

```

测试集群

```
./redis-cli -c -p 6379
```

-c：表示集群

-p：端口号

面试官：

Redis 服务是通过什么，实现在每个端口之间互相交互的？

集群原理：

Slots:(槽) redis 总共有 16384 个槽 每个服务器所占的槽的区间不一样。

Service1-6379 : 0-5460

Service2-6380 : 5461-10922

Service3-6381 : 10923-16383

set name admin , crc16 算法。返回一个槽的位置【5798】，判断你返回这个槽在那个区间？

$\text{crc16}(\text{admin}) = \text{值} / 16384 = 5798$ 。

对应的区间，就是对应的服务，无论是存，还是取都会遵循该算法！

关掉集群：实际就是关掉 redis 服务。 Kill 掉 redis 服务即可

```
killall redis-server
```

集群中写入数据

客户端重定向

①在 redis-cli 每次录入、查询键值，redis 都会计算出该 key 应该送往的插槽，如果不是该客户端对应服务器插槽，redis 会报错，并告知应前往的 redis 实例地址和端口。

②redis-cli 客户端提供了 -c 参数实现自动重定向。如 redis-cli -c -p 6379 登入后，再录入、查询键值对可以自动重定向。

```
# redis-cli -c -p 6379
```

③每个 slot 可以存储一批键值对。

```
127.0.0.1:6379> set name admin
-> Redirected to slot [5798] located at 192.168.67.201:6383
OK
192.168.67.201:6383> █
```

如何多键操作

采用哈希算法后,会自动地分配 slot,而 不在一个 slot 下的键值,是不能使用 mget,mset 等多键操作。

如果有需求,需要将一批业务数据一起插入呢?

解决:可以通过`{}`来定义组的概念,从而使 key 中`{}`内相同内容的键值对放到一个 slot 中去。

```
192.168.67.201:6383> mset k11 v11 k12 v12 k13 v13
(error) CROSSSLOT Keys in request don't hash to the same slot
192.168.67.201:6383> mset {key}k11 v11 {key}k12 v12 {key}k13 v13
-> Redirected to slot [12539] located at 192.168.67.201:6381
OK
192.168.67.201:6381> mget{key}
(error) ERR unknown command 'mget{key}'
192.168.67.201:6381> mget {key}k11 {key}k12
1) "v11"
2) "v12"
192.168.67.201:6381> █
```

集群中故障恢复

1、主机挂掉,从机会上位

```
[root@localhost bin]# ./redis-cli -h 192.168.67.201 -p 6380 shutdown
[root@localhost bin]# █
```

`./redis-cli -h:` 表示 host !

```
9b4d2bf35b060e550a412b970f8cad8960df3482 192.168.67.201:6381 master - 0 1527741697260 3 connected 10923-16383
7dfe2ad9a8668b24a768b0194c3efb52f3f33f7e 192.168.67.201:6380 master, fail - 1527741650334 1527741644793 2 disconnected
f543cdb108f76377a50c04e133ec8650141ac952 192.168.67.201:6383 master - 0 1527741692210 7 connected 5461-10922
ac72adc1b3ce1d8b6448d00a4cbda2a5681133fa 192.168.67.201:6379 myself, master - 0 0 1 connected 0-5460
67e57e54eb315c2ac850f7e0aafa81f1ef0671a8 192.168.67.201:6382 slave ac72adc1b3ce1d8b6448d00a4cbda2a5681133fa 0 1527741696250 4 connect
8fe2b771b4390de010464953b9f1ffa27a8a0167 192.168.67.201:6384 slave 9b4d2bf35b060e550a412b970f8cad8960df3482 0 1527741698270 6 connect
127.0.0.1:6379> █
```

2、主节点恢复后如何

```
[root@localhost bin]# ./redis-cli -h 192.168.67.201 -p 6380 shutdown
[root@localhost bin]# ./redis-server /opt/redis/conf/redis-6380.conf
```

```
7dfe2ad9a8668b24a768b0194c3efb52f3f33f7e 192.168.67.201:6380 slave f543cdb108f76377a50c04e133ec8650141ac952 0 1527741983907 7 connected
f543cdb108f76377a50c04e133ec8650141ac952 192.168.67.201:6383 master - 0 1527741988941 7 connected 5461-10922
```

当主节点：挂掉之后，从节点上位！当主节点恢复之后，不会成为主，应该改成从节点！

== 集群的从节点与主从复制的从节点有所有区别！

集群的从节点作用：当主节点挂掉，则会立即上位。

主从复制的从节点：只负责读取数据

集群的 Jedis 开发

```
@Test
    public void testCluster() {
        Set<HostAndPort> jedisClusterNodes = new
HashSet<HostAndPort>();
        //Jedis Cluster will attempt to discover cluster nodes
        automatically
        jedisClusterNodes.add(new HostAndPort("192.168.4.128", 6379));
        JedisCluster jc = new JedisCluster(jedisClusterNodes);
        jc.set("foo", "bar");
        String value = jc.get("foo");
    }
```

集群的优缺点

优点：

- 实现扩容
- 分摊压力
- 无中心配置相对简单

缺点：

- 多键操作是不被支持的
- 多键的 Redis 事务是不被支持的。lua 脚本不被支持。
- 由于集群方案出现较晚，很多公司已经采用了其他的集群方案，而代理或者客户端分片的方案想要迁移至 redis cluster，需要整体迁移而不是逐步过渡，复杂度较大。

使用 eclipse 连接集群，进行数据操作

假如说，明天在使用集群的话。还需要创建么？NO

只需要启动集群即可 `/usr/local/redis/bin ./start.sh`

3. 启动集群

启动集群：也就是启动多个 redis 服务，而 redis 服务之前我们已经配置好了。`/opt/redis/conf`

该目录下：`redis-6379.conf`---`redis-6384.conf`

```
lrwxrwxrwx. 1 root root      12 7月 27 10:51 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7829986 7月 27 10:51 redis-server
-rwxr-xr-x. 1 root root  60852 7月 27 15:41 redis-trib.rb
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6379.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6380.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6381.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6382.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6383.conf
root@localhost bin]# ./redis-server /opt/redis/conf/redis-6384.conf
root@localhost bin]# ps -ef | grep redis
root      3376      1  0 09:33 ?        00:00:00 ./redis-server 0.0.0.0:6379 [c
luster]
root      3380      1  0 09:33 ?        00:00:00 ./redis-server 0.0.0.0:6380 [c
luster]
root      3384      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6381 [c
luster]
root      3388      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6382 [c
luster]
root      3394      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6383 [c
luster]
root      3401      1  0 09:34 ?        00:00:00 ./redis-server 0.0.0.0:6384 [c
luster]
root      3407    3360  0 09:34 pts/0    00:00:00 grep redis
root@localhost bin]#
```

4. 使用 eclipse 连接集群。

第一步： 应该要关闭防火墙，service iptables stop

第二步：

```
public static void main(String[] args) {  
    // 创建多个redis服务的host和port  
    Set<HostAndPort> set = new HashSet<>();  
    set.add(new HostAndPort("192.168.26.30", 6379));  
    set.add(new HostAndPort("192.168.26.30", 6380));  
    set.add(new HostAndPort("192.168.26.30", 6381));  
    set.add(new HostAndPort("192.168.26.30", 6382));  
    set.add(new HostAndPort("192.168.26.30", 6383));  
    set.add(new HostAndPort("192.168.26.30", 6384));  
    // 创建一个JedisCluster 对象 并将set集合中的host和port给  
    redis集群对象  
    JedisCluster jc = new JedisCluster(set);  
    jc.set("one", "小骨");  
    System.out.println(jc.get("one"));  
}
```

注意：导入 **redis.jar,commons-pool.jar**

使用 spring 整合 redis

4. 导入 jar 包：spring 和 redis

```
commons-logging-1.1.3.jar  
commons-pool2-2.4.2.jar  
jedis-2.9.0.jar  
junit-4.8.jar  
spring-aop-4.1.6.RELEASE.jar
```

```
spring-aspects-4.1.6.RELEASE.jar
spring-beans-4.1.6.RELEASE.jar
spring-context-4.1.6.RELEASE.jar
spring-context-support-4.1.6.RELEASE.jar
spring-core-4.1.6.RELEASE.jar
spring-expression-4.1.6.RELEASE.jar
spring-instrument-4.1.6.RELEASE.jar
spring-instrument-tomcat-4.1.6.RELEASE.jar
spring-jdbc-4.1.6.RELEASE.jar
spring-jms-4.1.6.RELEASE.jar
spring-messaging-4.1.6.RELEASE.jar
spring-orm-4.1.6.RELEASE.jar
spring-oxm-4.1.6.RELEASE.jar
spring-test-4.1.6.RELEASE.jar
spring-tx-4.1.6.RELEASE.jar
spring-web-4.1.6.RELEASE.jar
spring-webmvc-4.1.6.RELEASE.jar
spring-webmvc-portlet-4.1.6.RELEASE.jar
spring-websocket-4.1.6.RELEASE.jar
```

5. 配置 spring 的核心配置文件

Spring 核心配置文件的主要工作就是将 JedisPool 类交给 spring 容器来管理。

redis.clients.jedis.JedisCluster 也要交给 spring 容器来管理。

具体配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
```

```
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
4.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">
<!-- 配置jedis连接池 -->
<bean id="jedisPoolConfig"
class="redis.clients.jedis.JedisPoolConfig">
    <!-- 最大连接数 -->
    <property name="maxTotal" value="30" />
    <!-- 最大空闲连接数 -->
    <property name="maxIdle" value="10" />
    <!-- 每次释放连接的最大数目 -->
    <property name="numTestsPerEvictionRun" value="1024" />
    <!-- 释放连接的扫描间隔（毫秒） -->
    <property name="timeBetweenEvictionRunsMillis"
value="30000" />
    <!-- 连接最小空闲时间 -->
```



```
<property name="minEvictableIdleTimeMillis" value="1800000"
/>

<!-- 连接空闲多久后释放，当空闲时间>该值 且 空闲连接>最大空闲连接
数 时直接释放 -->

<property name="softMinEvictableIdleTimeMillis"
value="10000" />

<!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -
->

<property name="maxWaitMillis" value="1500" />

<!-- 在获取连接的时候检查有效性, 默认false -->

<property name="testOnBorrow" value="true" />

<!-- 在空闲时检查有效性, 默认false -->

<property name="testWhileIdle" value="true" />

<!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true
-->

<property name="blockWhenExhausted" value="false" />

</bean>

<!-- jedis整合spring单机版 -->

<!-- <bean id="jedisClient"
class="redis.clients.jedis.JedisPool">

    <constructor-arg name="host" value="192.168.26.128"/>

    <constructor-arg name="port" value="6379"/>

    <constructor-arg name="poolConfig" ref="jedisPoolConfig"/>

</bean> -->

<!-- jedisCluster JedisCluster jedisClients = new JedisCluster
```

```
-->
    <bean id="jedisClients"
class="redis.clients.jedis.JedisCluster">
    <!-- ref引用了上面的配置 -->
        <constructor-arg name="poolConfig" ref="jedisPoolConfig"/>
        <constructor-arg name="nodes">
            <set>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6379"/>
                </bean>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6380"/>
                </bean>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6381"/>
                </bean>
                <bean class="redis.clients.jedis.HostAndPort">
                    <constructor-arg name="host"
value="192.168.26.30"/>
                    <constructor-arg name="port" value="6382"/>
                </bean>
            </set>
        </constructor-arg>
    </bean>
```

```

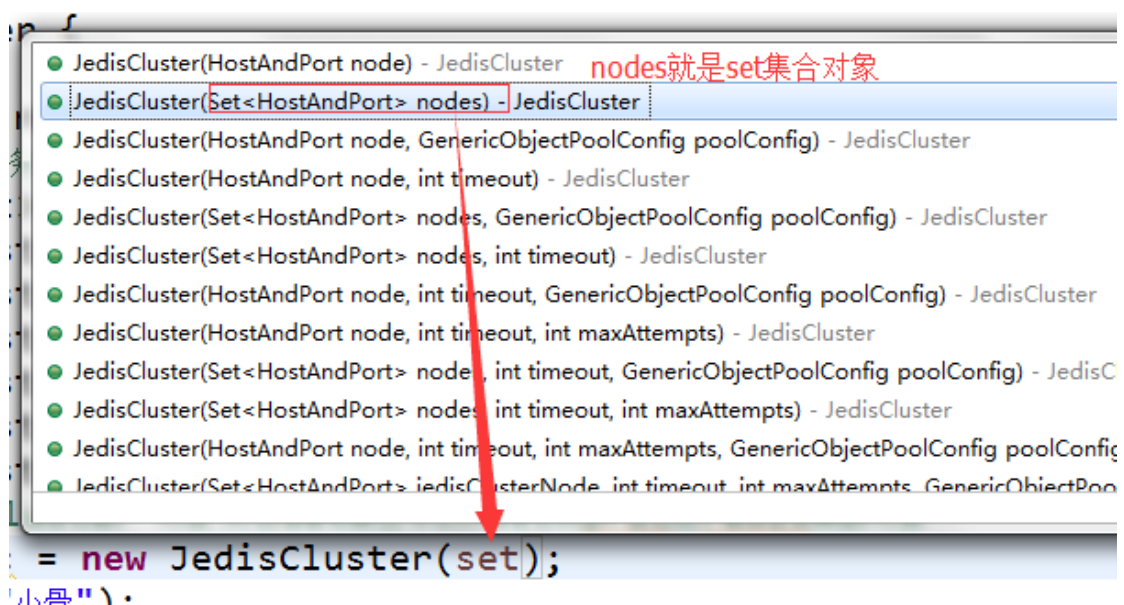
        <bean class="redis.clients.jedis.HostAndPort">
            <constructor-arg name="host"
value="192.168.26.30"/>
            <constructor-arg name="port" value="6383"/>
        </bean>
        <bean class="redis.clients.jedis.HostAndPort">
            <constructor-arg name="host"
value="192.168.26.30"/>
            <constructor-arg name="port" value="6384"/>
        </bean>
    </set>
</constructor-arg>
</bean>
</beans>

```

<bean id="jedisClients" class="redis.clients.jedis.JedisCluster"> 相当于:

JedisCluster jedisClients = new JedisCluster();

<constructor-arg name="nodes"> 相当于:



```
<set>

    <bean class="redis.clients.jedis.HostAndPort">
        <constructor-arg name="host"
value="192.168.26.30"/>
        <constructor-arg name="port" value="6379"/>
    </bean>

</set>
```

相当于：给 set 集合中赋予一个 HostAndPort 对象

```
<constructor-arg name="host" value="192.168.26.30"/>
    <constructor-arg name="port" value="6379"/>
```

通过构造器方式对 HostAndPort 进行初始化数据

```
// 创建多个redis服务的host和port
Set<HostAndPort> set = new HashSet<>();
set.add(new HostAndPort("192.168.26.30", 6379));
set.add(new HostAndPort("192.168.26.30", 6380));
set.add(new HostAndPort("192.168.26.30", 6381));
set.add(new HostAndPort("192.168.26.30", 6382));
set.add(new HostAndPort("192.168.26.30", 6383));
set.add(new HostAndPort("192.168.26.30", 6384));
```

6. 编写测试类

```
public static void main(String[] args) {

    // 读取 spring 的配置文件

    ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext-jedis.xml");

    JedisCluster jc = (JedisCluster) ac.getBean("jedisClients");

    jc.set("cluster", "集群");

    System.out.println(jc.get("cluster"));

}
```

