

Redis

尚硅谷 JAVA 研究院

版本 V1.0

第 1 章 Redis 入门

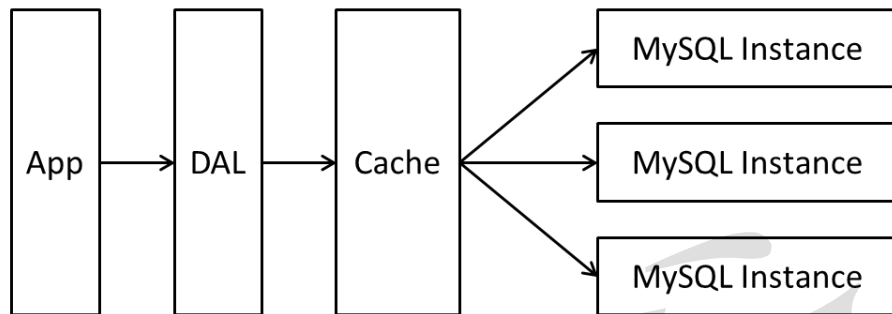
1. 互联网项目架构演变



单机版项目架构

- 项目特点
 - 90年代
 - 网站访问量低
 - 以静态网页为主，很少动态交互应用
 - 单个数据库能够满足要求
- 性能瓶颈
 - 瓶颈1：数据总量达到服务器容量极限——单表500万条记录
 - 瓶颈2：数据索引（B+树）达到服务器内存容量极限
 - 瓶颈3：读写混合的访问量达到服务器能够承受的极限
- 上述三条满足任何一个都需要升级架构

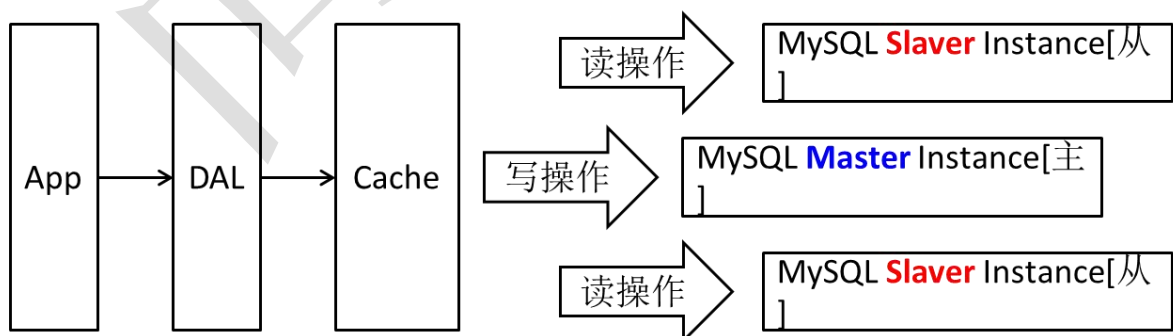
Memcache缓存+MySQL垂直拆分



随着访问量上升，大部分使用 MySQL 架构的网站在数据库上都开始出现性能问题，Web 程序不能再仅仅专注在功能上，同时也在追求性能。开始使用缓存技术缓解数据库压力，优化数据库的结构和索引。刚开始时比较流行的是通过文件缓存来缓解数据库压力，但是当访问量继续增大，文件缓存中的数据不能在多台 Web 服务器之间共享，大量的小文件 IO 也带来了比较高的 IO 压力。在这种情况下，Memcache 就成了一款非常有效的解决方案。

Memcache 作为一个独立的分布式缓存服务器，为多个 Web 服务器提供了一个共享的高性能缓存服务，在 Memcache 服务器上，又发展了根据 hash 算法来进行多台 Memcache 缓存服务的扩展，然后又出现了一致性 hash 来解决增加或减少缓存服务器导致重新 hash 带来的大量缓存失效问题。

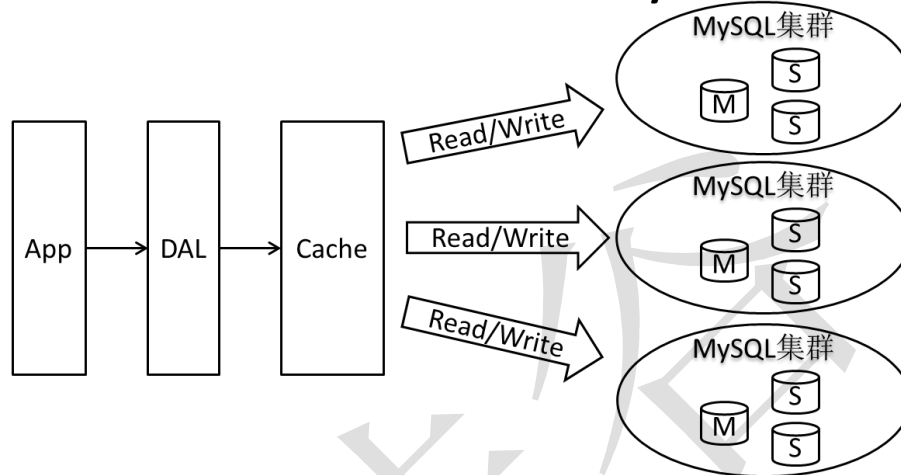
MySQL主从读写分离



由于数据库的写入压力增加，Memcached 只能缓解数据库的读取压力。读写集中在一个数据库上让数据库不堪重负，大部分网站开始使用主从复制技术来达到读写分离，以提高读写性能和读库的可扩展性。Mysql 的 master-slave

模式成为这个时候的网站标配了

分库分表+水平拆分+MySQL集群

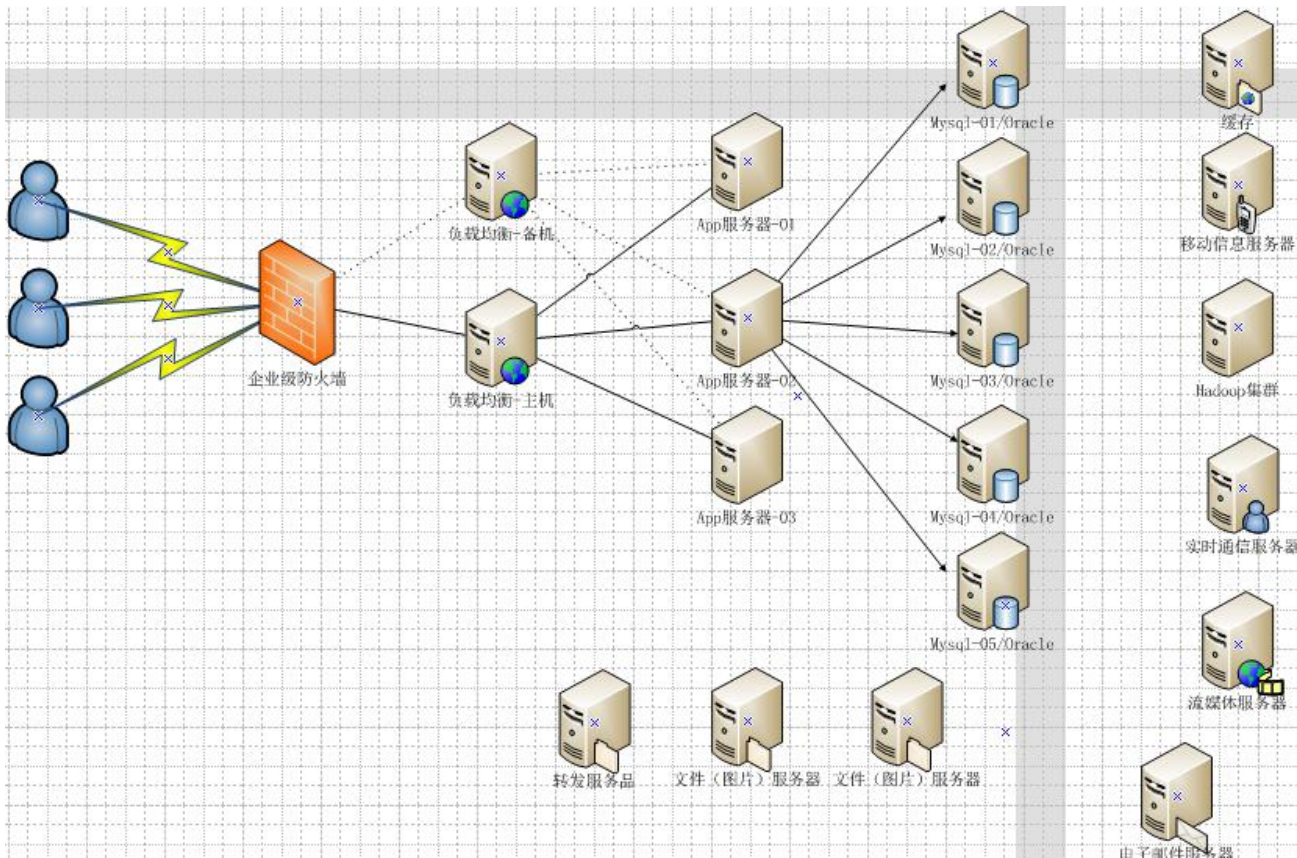


在 Memcached 的高速缓存，MySQL 的主从复制，读写分离的基础之上，这时 MySQL 主库的写压力开始出现瓶颈，而数据量的持续猛增，由于 MyISAM 使用表锁，在高并发下会出现严重的锁问题，大量的高并发 MySQL 应用开始使用 InnoDB 引擎代替 MyISAM。

同时，开始流行使用分表分库来缓解写压力和数据增长的扩展问题。这个时候，分表分库成了一个热门技术，是业界讨论的热门技术问题。也就在这个时候，MySQL 推出了还不太稳定的表分区，这也给技术实力一般的公司带来了希望。虽然 MySQL 推出了 MySQL Cluster 集群，但性能也不能很好满足互联网的要求，只是在高可靠性上提供了非常大的保证。

MySQL 数据库也经常存储一些大文本字段，导致数据库表非常的大，在做数据库恢复的时候就导致非常的慢，不容易快速恢复数据库。比如 1000 万 4KB 大小的文本就接近 40GB 的大小，如果能把这些数据从 MySQL 省去，MySQL 将变得非常的小。关系数据库很强大，但是它并不能很好的应付所有的应用场景。MySQL 的扩展性差（需要复杂的技术来实现），大数据下 IO 压力大，表结构更改困难，正是当前使用 MySQL 的开发人员面临的问题。

现在的互联网架构：



目前互联网的新要求：3V 和 3 高

大数据时代的 3V

Volume: 海量，数据量极大

Variety: 多样

数据类型：文本、图片、音频、视频.....

终端设备：PC、移动端、嵌入式设备.....

Velocity: 实时

直播，金融证券.....

互联网时代的 3 高

高可扩展

不断优化现有的功能，不断开发新的功能；

高性能

不能让用户感觉到等待的时间；

高并发

同时处理并发请求的能力，如双十一的秒杀、抢购火车票；

提升硬件，优化系统，优化项目，将费时的操作进入异步处理；

2. NoSQL

不仅仅是 SQL—关系型数据库的强大助力

NoSQL数据库和关系型数据库对比

关系型数据库

- 高度组织化结构化数据
- 结构化查询语言（SQL）
- 数据和关系都存储在单独的表中
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

NoSQL数据库

- 没有声明性查询语言
- 没有预定义的模式
- 最终一致性，而非ACID属性
- 非结构化和不可预知的数据
- CAP定理
- 高性能，高可用性和可伸缩性

NoSQL 数据库的优势

- 易扩展
 - NoSQL 数据库种类繁多，但它们都有一个共通的特点：就是去除关系型数据库的“关系型”特点。数据之间无关系，这样就变得非常容易扩展，而相对应的来看：关系型数据库修改表结构非常困难。这就为项目架构设计提供了更大的扩展空间。
- 大数据量高性能
 - NoSQL 数据库都具有非常高的读写性能，尤其在大数据量的情况下，表现同样优秀。这得益于 NoSQL 数据库中数据之间没有“关系”，数据库结构简单。
 - 从缓存角度来看，MySQL 的 Query Cache 是表级别的粗粒度缓存，假设存储了 100 条数据，其中有一条数据修改了，整个缓存失效，效率很低。而 NoSQL 数据库的缓存是记录级的细粒度缓存，任何一条记录的修改都不影响其他记录，效率很高。

- 多样灵活的数据模型
 - NoSQL 数据库无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增减修改字段简直就是一个噩梦。

3. Redis

3.1 简介

Redis:Remote Dictionary Server(远程字典服务器)

官网:

<https://redis.io/>



redis

Commands

Clients

Documentation

Community

Download

Modules

Support

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. [Learn more →](#)

<http://www.redis.cn/>

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它支持多种类型的数据结构，如字符串（strings），散列（hashes），列表（lists），集合（sets），有序集合（sorted sets）与范围查询，bitmaps，hyperloglogs 和 地理空间（geospatial）索引半径查询。Redis 内置了复制（replication），Lua脚本（Lua scripting），LRU驱动事件（LRU eviction），事务（transactions）和不同级别的磁盘持久化（persistence），并通过 Redis哨兵（Sentinel）和自动分区（Cluster）提供高可用性（high availability）。

3.2 安装

①将 Redis 的 tar 包上传到 opt 目录

②解压缩

③安装 gcc 环境

我们需要将源码编译后再安装，因此需要安装 c 语言的编译环境！不能直接 make！

```
MAKE hiredis
cd hiredis && make static
make[3]: Entering directory `/opt/redis-3.2.5/deps/hiredis'
gcc -std=c99 -pedantic -c -O3 -fPIC -Wall -W -Wstrict-prototypes -Wwrite-strings -g -ggdb net.c
make[3]: gcc: 命令未找到
make[3]: *** [net.o] 错误 127
make[3]: Leaving directory `/opt/redis-3.2.5/deps/hiredis'
make[2]: *** [hiredis] 错误 2
make[2]: Leaving directory `/opt/redis-3.2.5/deps'
make[1]: [persist-settings] 错误 2 (忽略)
CC adlist.o
/bin/sh: cc: command not found
make[1]: *** [adlist.o] 错误 127
make[1]: Leaving directory `/opt/redis-3.2.5/src'
make: *** [all] 错误 2
```

两种方式：

第一种：可以上网，yum install -y gcc-c++

```
[root@vm1 redis-3.2.5]# yum install -y gcc-c++
```

第二种：不能上网，确保插入了安装光盘！（CentOS6）

挂载安装光盘，然后进入 Packages 中，依次执行以下命令：

```
rpm -ivh mpfr-2.4.1-6.el6.x86_64.rpm
```

```
rpm -ivh cpp-4.4.7-17.el6.x86_64.rpm
```

```
rpm -ivh ppl-0.10.2-11.el6.x86_64.rpm
```



























```
rpm -ivh cloog-ppl-0.15.7-1.2.el6.x86_64.rpm
```

```
rpm -ivh gcc-4.4.7-17.el6.x86_64.rpm
```

不能上网，确保插入了安装光盘！（CentOS7）

挂载安装光盘，然后进入 `/run/media/root/CentOS 7 x86_64/Packages` 中

把如下安装包找出，复制到独立目录 `rpmgcc`

 autogen-libopts-5.18-5.el7.x86_64.rpm	2014/7/4 0:43	RPM 文件	67 KB
 cpp-4.8.5-11.el7.x86_64.rpm	2016/11/20 17:27	RPM 文件	6,080 KB
 gcc-4.8.5-11.el7.x86_64.rpm	2016/11/20 17:46	RPM 文件	16,561 KB
 gcc-c++-4.8.5-11.el7.x86_64.rpm	2016/11/20 17:46	RPM 文件	7,344 KB
 glibc-devel-2.17-157.el7.x86_64.rpm	2016/11/20 17:54	RPM 文件	1,081 KB
 glibc-headers-2.17-157.el7.x86_64.rpm	2016/11/20 17:54	RPM 文件	669 KB
 kernel-headers-3.10.0-514.el7.x86_64.rpm	2016/11/23 2:20	RPM 文件	4,885 KB
 keyutils-libs-devel-1.5.8-3.el7.x86_64.rpm	2014/7/4 2:30	RPM 文件	38 KB
 krb5-devel-1.14.1-26.el7.x86_64.rpm	2016/11/20 18:31	RPM 文件	652 KB
 libcom_err-devel-1.42.9-9.el7.x86_64.rpm	2016/11/20 18:37	RPM 文件	31 KB
 libmpc-1.0.1-3.el7.x86_64.rpm	2014/7/4 2:59	RPM 文件	51 KB
 libselinux-devel-2.5-6.el7.x86_64.rpm	2016/11/20 19:09	RPM 文件	186 KB
 libsepol-devel-2.5-6.el7.x86_64.rpm	2016/11/20 19:10	RPM 文件	75 KB
 libstdc++-devel-4.8.5-11.el7.x86_64.rpm	2016/11/20 19:13	RPM 文件	1,538 KB
 libverto-devel-0.2.5-4.el7.x86_64.rpm	2014/7/4 3:23	RPM 文件	12 KB
 mpfr-3.1.1-4.el7.x86_64.rpm	2014/7/4 3:49	RPM 文件	204 KB
 ntp-4.2.6p5-25.el7.centos.x86_64.rpm	2016/11/20 19:40	RPM 文件	547 KB
 ntpdate-4.2.6p5-25.el7.centos.x86_64.rpm	2016/11/20 19:40	RPM 文件	86 KB
 openssl-1.0.1e-60.el7.x86_64.rpm	2016/11/20 19:48	RPM 文件	714 KB
 openssl098e-0.9.8e-29.el7.centos.3.x86_64.rpm	2016/3/9 5:34	RPM 文件	793 KB
 openssl-devel-1.0.1e-60.el7.x86_64.rpm	2016/11/20 19:49	RPM 文件	1,210 KB
 openssl-libs-1.0.1e-60.el7.x86_64.rpm	2016/11/20 19:49	RPM 文件	959 KB
 pkgconfig-0.27.1-4.el7.x86_64.rpm	2014/7/4 4:28	RPM 文件	54 KB
 tcl-8.5.13-8.el7.x86_64.rpm	2015/11/25 15:43	RPM 文件	1,935 KB
 zlib-1.2.7-17.el7.x86_64.rpm	2016/11/20 21:05	RPM 文件	90 KB
 zlib-devel-1.2.7-17.el7.x86_64.rpm	2016/11/20 21:05	RPM 文件	50 KB

进入 rpmgcc 该目录运行如下命令，按照依赖关系安装

```
rpm -Uvh *.rpm --nodeps --force
```

之后查看安装是否成功：`rpm -qa|grep gcc`

```
[root@vm1 Packages]# rpm -qa|grep gcc
libgcc-4.4.7-17.el6.x86_64
gcc-4.4.7-17.el6.x86_64
```

常见错误：在没有安装 gcc 环境下，如果执行了 make，不会成功！安装环境后，第二次 make 有可能报错：

[Jemalloc/jemalloc.h:没有那个文件](#)

```
[root@vm1 redis-3.2.5]# make
cd src && make all
make[1]: Entering directory `/opt/redis-3.2.5/src'
CC adlist.o
在包含自 adlist.c: 34 的文件中:
zmalloc.h:50:31: 错误: jemalloc/jemalloc.h: 没有那个文件或目录
zmalloc.h:55:2: 错误: #error "Newer version of jemalloc required"
make[1]: *** [adlist.o] 错误 1
make[1]: Leaving directory `/opt/redis-3.2.5/src'
make: *** [all] 错误 2
```

解决： 运行 make distclean 之后再 make

④编译，执行 make 命令！

⑤编译完成后，安装，执行 make install 命令！

```
[root@centos4 src]# make install

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

INSTALL install
INSTALL install
INSTALL install
INSTALL install
INSTALL install
```

⑥文件会被安装到 /usr/local/bin 目录

```
PREFIX?=/usr/local
INSTALL_BIN=$(PREFIX)/bin
INSTALL=install
```

```
[root@vm1 bin]# ll
总用量 26340
-rwxr-xr-x. 1 root root 5580327 3月 20 10:13 redis-benchmark
-rwxr-xr-x. 1 root root 22217 3月 20 10:13 redis-check-aof
-rwxr-xr-x. 1 root root 7826782 3月 20 10:13 redis-check-rdb
-rwxr-xr-x. 1 root root 5709036 3月 20 10:13 redis-cli
lrwxrwxrwx. 1 root root 12 3月 20 10:13 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 7826782 3月 20 10:13 redis-server
```

⑦可以将 redis 的 bin 目录，加入到环境变量中

bin 目录常用命令	
Redis-benchmark	压力测试。标准是每秒 80000 次写操作，110000 次读操作（服务启动起来后执行,类似安兔兔跑分)
Redis-check-aof	修复有问题的 AOF 文件

Redis-check-dump	修复有问题的 dump.rdb 文件
Redis-sentinel	启动哨兵，集群使用
redis-server	启动服务器
redis-cli	启动客户端

3.3 启动

3.3.1 服务端启动

将配置文件，保留一份副本，进行启动。

命令：redis-server conf

```
[root@vm1 myredis]# redis-server redis.conf
6977:M 20 Mar 11:16:11.240 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.2.5 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 6977

http://redis.io

6977:M 20 Mar 11:16:11.265 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
6977:M 20 Mar 11:16:11.265 # Server started, Redis version 3.2.5
6977:M 20 Mar 11:16:11.265 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
6977:M 20 Mar 11:16:11.266 # WARNING you have Transparent Huge Pages (THP) support 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/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
6977:M 20 Mar 11:16:11.266 * The server is now ready to accept connections on port 6379
```

但是这样的话，发现命令窗口被占用了，很不方便。当然你可以再启动一个会话窗口。

解决：修改配置文件，改为守护进程，在后台运行

我们选择后台运行，怎么办？修改配置文件。

daemonize yes


```
##### GENERAL #####
# By default Redis does not run as a daemon. Use 'yes' if you need it.
# Note that Redis will write a pid file in /var/run/redis.pid when daemonized.
daemonize yes
```

后台启动后，查看服务： netstat -anp|grep 6379

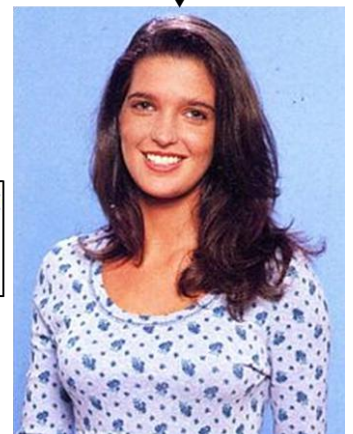
```
[root@vml myredis]# netstat -anp|grep 6379
tcp        0      0 127.0.0.1:6379      0.0.0.0:*           LISTEN      7109/redis-server 1
```

为什么是6379?



MERZ

Alessia Merz



<http://oldblog.antirez.com/post/redis-as-LRU-cache.html>

Today on Twitter I saw a tweet related to the ability to remember the Redis port number. There is a trick, the Redis port number, 6379, is **MERZ** at the phone keyboard.

Is it a coincidence that it sounds not random enough? Actually not ;) I selected 6379 because of MERZ, and not the other way around.

sentences that the showgirl was able to state in the italian TV. So we started using "MERZ" when something was... stupid. "Hey, that's merz!". And so forth.

3.3.2 客户端登录

命令	说明	举例	备注
redis-cli	启动客户端	redis-cli -p 端口号 连接指定的端口号	直接执行的话，默认端口号就是 6379;
ping	测试联通		回复 pong 代表联通
exit	退出客户端		
redis-cli shutdown	停止服务器	redis-cli -h 127.0.0.1 -p 6379	redis 是通过客户端发送停止服务器的命令

		shutdown	
		停止指定 ip 指定端口号的服务器	

第 2 章 Redis 基本操作

1. 数据库连接操作

命令	说明	举例	备注
select <dbid>	切换数据库	select 1: 切换到 1 号库	开启 redis 服务后，一共有 16（0-15）个库，默认在 0 号库
flushdb	清空当前库		
dbsize	查看数据库数据个数		
flushall	通杀全部库		

```
# Set the number of databases. The default database is DB 0, you can select
# a different one on a per-connection basis using SELECT <dbid> where
# dbid is a number between 0 and 'databases'-1
databases 16
```

2. key 的操作

Redis 中的数据以键值对（key-value）为基本存储方式，其中 key 都是字符串。

表达式	描述
KEYS pattern	查询符合指定表达式的所有 key，支持*，? 等
TYPE key	查看 key 对应值的类型
EXISTS key	指定的 key 是否存在，0 代表不存在，1 代表存在
DEL key	删除指定 key

RANDOMKEY	在现有的 KEY 中随机返回一个
EXPIRE key seconds	为键值设置过期时间，单位是秒，过期后 key 会被 redis 移除
TTL key	查看 key 还有多少秒过期，-1 表示永不过期，-2 表示已过期
RENAME key newkey	重命名一个 key，NEWKEY 不管是否是已经存在的都会执行，如果 NEWKEY 已经存在则会被覆盖
RENAMENX key newkey	只有在 NEWKEY 不存在时能够执行成功，否则失败

3. 常用五大数据类型

Redis 中的数据以键值对（key-value）为基本存储方式，其中 key 都是字符串，这里探讨数据类型都是探讨 value 的类型。

key	value	
	string	字符串
	list	可以重复的集合
	set	不可以重复的集合
	hash	类似于 Map<String,String>
	zset (sorted set)	带分数的 set

4. String 操作

String 类型是 Redis 中最基本的类型，它是 key 对应的一个单一值。

二进制安全，不必担心由于编码等问题导致二进制数据变化。所以 redis 的 string 可以包含任何数据，比如 jpg 图片或者序列化的对象。

Redis 中一个字符串值的最大容量是 512M。

SET key value	添加键值对
GET key	查询指定 key 的值

APPEND key value	将给定的 value 追加到原值的末尾
STRLEN key	获取值的长度
SETNX key value	只有在 key 不存在时设置 key 的值
INCR key	指定 key 的值自增 1，只对数字有效
DECR key	指定 key 的值自减 1，只对数字有效
INCRBY key num	自增 num
DECRBY key num	自减 num
MSET key1 value1 key2 value2...	同时设置多个 key-value 对
MGET key1 key2	同时获取一个或多个 value
MSETNX key1 value1 key2 value2	当 key 不存在时，设置多个 key-value 对
GETRANGE key 起始索引 结束索引	获取指定范围的值，都是闭区间
SETRANGE key 起始索引 value	从起始位置开始覆写指定的值
GETSET key value	以新换旧，同时获取旧值
SETEX key 过期时间 value	设置键值的同时，设置过期时间，单位秒

5. list 操作

在 Java 中 list 一般是单向链表，如常见的 ArrayList，只能从一侧插入。

在 Redis 中，list 是双向链表。可以从两侧插入。

可以简单理解为两端开口的，两端都可以进出。使用一个动画来演示。

常见操作：

遍历：遍历的时候，是从左往右取值；

删除：弹栈，POP；

添加：压栈，PUSH；



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。

LPUSE/RPUSH key value1 value2...	从左边/右边压入一个或多个值 头尾效率高，中间效率低
LPOP/RPOP key	从左边/右边弹出一个值 值在键在，值光键亡 弹出=返回+删除
LRANGE key start stop	查看指定区间的元素 正着数：0,1,2,3,... 倒着数：-1,-2,-3,...
LINDEX key index	按照索引下标获取元素（从左到右）
LLEN key	获取列表长度
LINSERT key BEFORE AFTER value newvalue	在指定 value 的前后插入 newvalue
LREM key n value	从左边删除 n 个 value
LSET key index value	把指定索引位置的元素替换为另一个值
LTRIM key start stop	仅保留指定区间的数据
RPOPLPUSH key1 key2	从 key1 右边弹出一个值，左侧压入到 key2

6. set 操作

set 是无序的，且是不可重复的。

SADD key member [member ...]	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。
------------------------------	--

SMEMBERS key	取出该集合的所有值
SISMEMBER key value	判断集合<key>是否为含有该<value>值，有返回 1，没有返回 0
SCARD key	返回集合中元素的数量
SREM key member [member ...]	从集合中删除元素
SPOP key [count]	从集合中随机弹出 count 个数量的元素，count 不指定就弹出 1 个
SRANDMEMBER key [count]	从集合中随机返回 count 个数量的元素，count 不指定就返回 1 个
SINTER key [key ...]	将指定的集合进行“交集”操作
SINTERSTORE dest key [key ...]	取交集，另存为一个 set
SUNION key [key ...]	将指定的集合执行“并集”操作
SUNIONSTORE dest key [key ...]	取并集，另存为 set
SDIFF key [key ...]	将指定的集合执行“差集”操作
SDIFFSTORE dest key [key ...]	取差集，另存为 set

7. hash 操作

Hash 数据类型的键值对中的值是“单列”的，不支持进一步的层次结构。

key	field:value
	"k01":"v01" "k02":"v02" "k03":"v03" "k04":"v04" "k05":"v05"

	"k06": "v06"
	"k07": "v07"

从前到后的数据对应关系

■ JSON

```
stu:{"stu_id":10,"stu_name":"tom","stu_age":30}
```

■ Java

```
public class Student {
    private Integer stuld;//10
    private String stuName;//"tom"
    private Integer stuAge;//30
    ...
}
```

```
Student stu=new Student()
stu.setStuld=10;
stu.setStuName="tom";
stu.setStuAge=30;
```

■ Redis hash

key	value(hash)	
stu	stu_id	10
	stu_name	tom
	stu_age	30

常用操作：

HSET key field value	为 key 中的 field 赋值 value
HMSET key field value [field value ...]	为指定 key 批量设置 field-value
HSETNX key field value	当指定 key 的 field 不存在时，设置其 value
HGETALL key	获取指定 key 的所有信息（field 和 value）
HKEYS key	获取指定 key 的所有 field

HVALS key	获取指定 key 的所有 value
HLEN key	指定 key 的 field 个数
HGET key field	从 key 中根据 field 取出 value
HMGET key field [field ...]	为指定 key 获取多个 field 的值
HEXISTS key field	指定 key 是否有 field
HINCRBY key field increment	为指定 key 的 field 加上增量 increment

8. zset 操作

zset 是一种特殊的 set (sorted set)，在保存 value 的时候，为每个 value 多保存了一个 score 信息。根据 score 信息，可以进行排序。

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

ZADD key [score member ...]	添加
ZSCORE key member	返回指定值的分数
ZRANGE key start stop [WITHSCORES]	返回指定区间的值，可选择是否一起返回 scores
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]	在分数的指定区间内返回数据，从小到大排列
ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]	在分数的指定区间内返回数据，从大到小排列
ZCARD key	返回集合中所有的元素的数量
ZCOUNT key min max	统计分数区间内的元素个数
ZREM key member	删除该集合下，指定值的元素
ZRANK key member	返回该值在集合中的排名，从 0 开始
ZINCRBY key increment value	为元素的 score 加上增量

第 3 章 Redis 配置文件

1. 单位说明

```
1k => 1000 bytes
1kb => 1024 bytes
1m => 1000000 bytes
1mb => 1024*1024 bytes
1g => 1000000000 bytes
1gb => 1024*1024*1024 bytes

units are case insensitive so 1GB 1Gb 1gB are all the same.
```

1k 和 1kb 是不同的；单位的大小写不敏感！

2. include

```
include /path/to/local.conf
include /path/to/other.conf
```

可以将公共的配置放入到一个公共的配置文件中，然后通过子配置文件引入父配置文件中的内容！

将配置按照模块分开！

3. network

属性	含义	备注
bind	限定访问的主机地址	如果没有 bind，就是任意 ip 地址都可以访问。生产环境下，需要写自己应用服务器的 ip 地址。
protected-mode	安全防护模式	如果没有指定 bind 指令，也没有配置密码，那么保护模式就开启，

		只允许本机访问。
port	端口号	默认是 6379
tcp-backlog	网络连接过程中, 某种状态的队列的长度	edis 是单线程的, 指定高并发时访问时排队的长度。超过后, 就呈现阻塞状态。可以理解是一个请求到达后至到接受进程处理前的队列长度。(一般情况下是运维根据集群性能调控) 高并发情况下, 此值可以适当调高。
timeout	超时时间	默认永超时
tcp-keepalive	对客户端的心跳检测间隔时间	

4. general

属性	含义	备注
daemonize	是否为守护进程模式运行	守护进程模式可以在后台运行
pidfile	进程 id 文件保存的路径	配置 PID 文件路径, 当 redis 作为守护进程运行的时候, 它会默认写到 /var/redis/run/redis_6379.pid 文件里面
loglevel	定义日志级别	debug (记录大量日志信息, 适用于开发、测试阶段) verbose (较多日志信息) notice (适量日志信息, 使用于生产环境) warning (仅有部分重要、关键信息才会被记录)
logfile	日志文件的位置	当指定为空字符串时, 为标准输出, 如果 redis 以守护进程模式运行, 那么日志将会输出到/dev/null
syslog-enabled	是否记录到系统日志	要想把日志记录到系统日志服务中, 就把它改成 yes
syslog-ident	设置系统日志的 ID	
syslog-facility	指定系统日志设置	必须是 USER 或者是 LOCAL0-LOCAL7 之间的值
databases	设置数据库数量	

5. 其他

属性	含义	备注
requirepass	设置密码	
maxclients	最大连接数	
maxmemory	最大占用多少内存	一旦占用内存超限，就开始根据缓存清理策略移除数据。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	缓存清理策略	<ul style="list-style-type: none">(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
maxmemory-samples	样本数	样本数越小，准确率越低，但是性能越好。LRU 算法和最小 TTL 算法都并非是精确的算法，而是估算值，所以你可以设置样本的大小。一般设置 3 到 7 的数字。

第 4 章 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能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redis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，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；

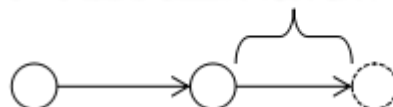
1.3 RDB 常用属性配置

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

- ①基于自动保存的策略
- ②执行 `save`，或者 `bgsave` 命令！执行时，是阻塞状态。
- ③执行 `flushdb` 命令，也会产生 `dump.rdb`，但里面是空的，没有意义。
- ④当执行 `shutdown` 命令时，也会主动地备份数据。

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

- AOF 文件的保存路径，同 RDB 的路径一致。
- 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
appendfilename	AOF 文件名称	
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-load-truncated	截断设置	如果选择的是 yes，当截断的 aof 文件被导入的时候，会自动发布一个 log 给客户端然后 load
--------------------	------	--

2.4 AOF 文件的修复

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

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

• 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

3. 备份建议

3.1 如何看待数据“绝对”安全

Redis 作为内存数据库从本质上来说，如果不想牺牲性能，就不可能做到数据的“绝对”安全。

RDB 和 AOF 都只是尽可能在兼顾性能的前提下降低数据丢失的风险，如果真的发生数据丢失问题，尽可能减少损失。

在整个项目的架构体系中，Redis 大部分情况是扮演“二级缓存”角色。

二级缓存适合保存的数据

- 经常要查询，很少被修改的数据。
- 不是非常重要，允许出现偶尔的并发问题。
- 不会被其他应用程序修改。

如果 Redis 是作为缓存服务器，那么说明数据在 MySQL 这样的传统关系型数据库中是有正式版本的。数据最终以 MySQL 中的为准。

3.2 官方建议

• 如何选择使用哪种持久化方式？

一般来说，如果想达到足以媲美 PostgreSQL 的数据安全性，你应该同时使用两种持久化功能。

如果你非常关心你的数据，但仍然可以承受数分钟以内的数据丢失，那么你可以只使用 RDB 持久化。

有很多用户都只使用 AOF 持久化，但我们并不推荐这种方式：因为定时生成 RDB 快照（snapshot）非常便于进行数据库备份，并且 RDB 恢复数据集的速度也要比 AOF 恢复的速度要快，除此之外，使用 RDB 还可以避免之前提到的 AOF 程序的 bug。

Note: 因为以上提到的种种原因，未来我们可能会将 AOF 和 RDB 整合成单个持久化模型。（这是一个长期计划。）接下来的几个小节将介绍 RDB 和 AOF 的更多细节。

官方推荐两个都用；如果对数据不敏感，可以选单独用 RDB；不建议单独用 AOF，因为可能出现 Bug；如果只是做纯内存缓存，可以都不用

第 5 章 事务

1. 事务简介

- Redis 中事务，不同于传统的关系型数据库中的事务。
- Redis 中的事务指的是一个单独的隔离操作。
- Redis 的事务中的所有命令都会序列化、按顺序地执行且不会被其他客户端发送来的命令请求所打断。
- Redis 事务的主要作用是串联多个命令防止别的命令插队

2. 事务常用命令

MULTI	标记一个事务块的开始
EXEC	执行事务中所有在排队等待的指令并将链接状态恢复到正常 当使用 WATCH 时，只有当被监视的键没有被修改，且允许检查设定机制时，EXEC 会被执行

DISCARD	刷新一个事务中所有在排队等待的指令，并且将连接状态恢复到正常。 如果已使用 WATCH，DISCARD 将释放所有被 WATCH 的 key。
WATCH	标记所有指定的 key 被监视起来，在事务中有条件的执行（乐观锁）

3. 事务的常见演示

3.1 简单组队

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set k1 v1
QUEUED
127.0.0.1:6379> get k1
QUEUED
127.0.0.1:6379> set k1 100
QUEUED
127.0.0.1:6379> get k1
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) "v1"
3) OK
4) "100"
```

MULTI 开启组队，EXEC 依次执行队列中的命令。

DISCARD 中途取消组队

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set k2 v1
QUEUED
127.0.0.1:6379> set k2 1
QUEUED
127.0.0.1:6379> DISCARD
OK
```

3.2 组队失败

3.2.1 自作自受

```
127.0.0.1:6379[1]> MULTI
OK
127.0.0.1:6379[1]> set k2 v2
QUEUED
127.0.0.1:6379[1]> set k3 v3
QUEUED
127.0.0.1:6379[1]> INCR k1
QUEUED
127.0.0.1:6379[1]> set k4 v4
QUEUED
127.0.0.1:6379[1]> EXEC
1) OK
2) OK
3) (error) ERR value is not an integer or out of range
4) OK
```

此种情况，语法符合规范，Redis 只有在执行中，才可以发现错误。而在 Redis 中，并没有回滚机制，因此错误的命令，无法执行，正确的命令会全部执行！

3.2.2 殃及池鱼

```
127.0.0.1:6379[1]> MULTI
OK
127.0.0.1:6379[1]> set k1 11
QUEUED
127.0.0.1:6379[1]> set k2 22
QUEUED
127.0.0.1:6379[1]> sett k3 33
(error) ERR unknown command 'sett'
127.0.0.1:6379[1]> set k4 44
QUEUED
127.0.0.1:6379[1]> EXEC
(error) EXECABORT Transaction discarded because of previous errors.
127.0.0.1:6379[1]> get k1
"v1"
```

在编译的过程中，Redis 检测出来了错误的语法命令，因此它认为这条组队，一定会发生错误，因此全体取消；

3.3.3 官方说明

为什么 Redis 不支持回滚 (roll back)

如果你有使用关系式数据库的经验，那么“Redis 在事务失败时不进行回滚，而是继续执行余下的命令”这种做法可能会让你觉得有点奇怪。

以下是这种做法的优点：

- Redis 命令只会因为错误的语法而失败（并且这些问题不能在入队时发现），或是命令用在了错误类型的键上面：这也就是说，从实用性的角度来说，失败的命令是由编程错误造成的，而这些错误应该在开发的过程中被发现，而不应该出现在生产环境中。
- 因为不需要对回滚进行支持，所以 Redis 的内部可以保持简单且快速。

有种观点认为 Redis 处理事务的做法会产生 bug，然而需要注意的是，在通常情况下，回滚并不能解决编程错误带来的问题。举个例子，如果你本来想通过 `INCR` 命令将键的值加上 1，却不小心加上了 2，又或者对错误类型的键执行了 `INCR`，回滚是没有办法处理这些情况的。

4. 锁



4.1 悲观锁

执行操作前假设当前的操作肯定（或有很大几率）会被打断（悲观）。基于这个假设，我们在做操作前就会把相关资源锁定，不允许自己执行期间有其他操作干扰。

Redis 不支持悲观锁。 Redis 作为缓存服务器使用时，以读操作为主，很少写操作，相应的操作被打断的几率较少。不采用悲观锁是为了防止降低性能。

4.2 乐观锁

执行操作前假设当前操作不会被打断（乐观）。基于这个假设，我们在做操作前不会锁定资源，万一发生

了其他操作的干扰，那么本次操作将被放弃。

5. Redis 中的锁策略

- Redis 采用了乐观锁策略（通过 `watch` 操作）。乐观锁支持读操作，适用于多读少写的情况！
- 在事务中，可以通过 `watch` 命令来加锁；使用 `UNWATCH` 可以取消加锁；
- 如果在事务之前，执行了 `WATCH`（加锁），那么执行 `EXEC` 命令或 `DISCARD` 命令后，锁对自动释放，即不需要再执行 `UNWATCH` 了

第 6 章 redis 消息订阅

消息订阅是进程间的一种消息通信方式，即发送者（`pub`）发送消息，订阅者（`sub`）接收消息。Redis 支持消息订阅机制。

命令	描述	举例
<code>SUBSCRIBE [频道]</code>	订阅频道	
<code>PUBLISH [频道][消息]</code>	向指定频道发布消息	

示例：

订阅者订阅消息后，此时控制台会处于阻塞状态，用于接收发布者发布的消息。

```
127.0.0.1:6379[1]> SUBSCRIBE CCTV1 CCTV2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "CCTV1"
3) (integer) 1
1) "subscribe"
2) "CCTV2"
3) (integer) 2
```

发布者发布消息：

```
127.0.0.1:6379[1]> PUBLISH CCTV1 news
(integer) 1
127.0.0.1:6379[1]> PUBLISH CCTV2 weather
(integer) 1
```

```
1) "message"
2) "CCTV1"
3) "news"
1) "message"
2) "CCTV2"
3) "weather"
```

第 7 章 主从复制

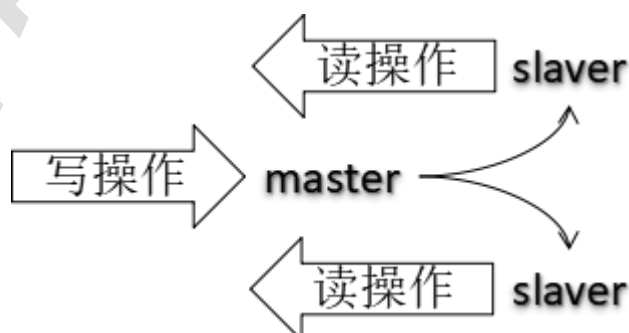
1. 主从简介

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：

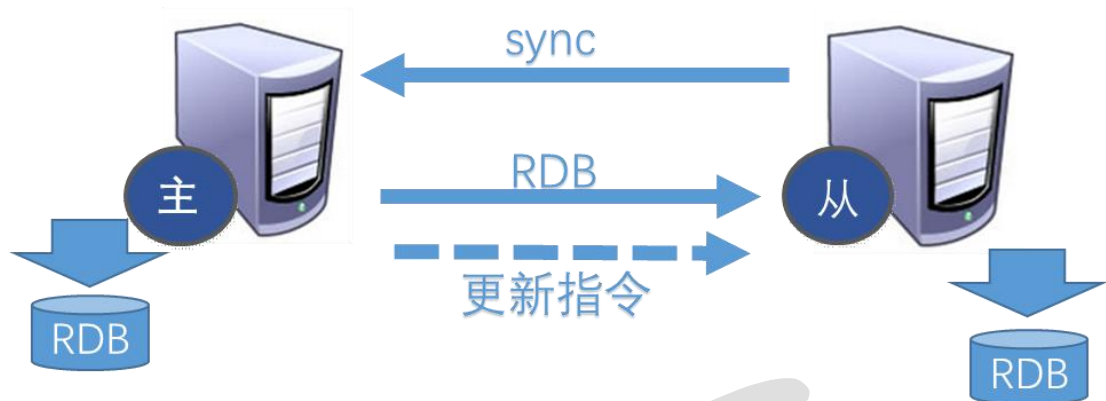
读写分离提高 Redis 性能；

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。

样本：

```
include /root/redis_replication/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

原则是配从不配主。

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
```

3.2 永久建立

在从机的配置文件中，编写 `slaveof` 属性配置！

3.3 恢复身份

执行命令 `slaveof no noe` 恢复自由身！

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。

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

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

⑤从机宕机后，重启，宕机期间主机的新增记录，从集是否会顺利复制？

答：可以

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`

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！

5. 哨兵模式

5.1 简介

作用：

①Master 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

下线:

①主观下线: Subjectively Down, 简称 SDOWN, 指的是当前 Sentinel 实例对某个 redis 服务器做出的下线判断。

②客观下线: Objectively Down, 简称 ODOWN, 指的是多个 Sentinel 实例在对 Master Server 做出 SDOWN 判断, 并且通过 SENTINEL is-master-down-by-addr 命令互相交流之后, 得出的 Master Server 下线判断, 然后开启 failover.

工作原理:

- ①每个 Sentinel 以每秒钟一次的频率向它所知的 Master, Slave 以及其他 Sentinel 实例发送一个 PING 命令;
- ②如果一个实例(instance)距离最后一次有效回复 PING 命令的时间超过 down-after-milliseconds 选项所指定的值, 则这个实例会被 Sentinel 标记为主观下线;
- ③如果一个 Master 被标记为主观下线, 则正在监视这个 Master 的所有 Sentinel 要以每秒一次的频率确认 Master 的确进入了主观下线状态;
- ④当有足够数量的 Sentinel (大于等于配置文件指定的值) 在指定的时间范围内确认 Master 的确进入了主观下线状态, 则 Master 会被标记为客观下线;
- ⑤在一般情况下, 每个 Sentinel 会以每 10 秒一次的频率向它已知的所有 Master, Slave 发送 INFO 命令
- ⑥当 Master 被 Sentinel 标记为客观下线时, Sentinel 向下线的 Master 的所有 Slave 发送 INFO 命令的频率会从 10 秒一次改为每秒一次;
- ⑦若没有足够数量的 Sentinel 同意 Master 已经下线, Master 的客观下线状态就会被移除;
若 Master 重新向 Sentinel 的 PING 命令返回有效回复, Master 的主观下线状态就会被移除;

5.2 配置

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/s
ys/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

+sdown master mymaster 127.0.0.1 6379 【主观下线】

+odown master mymaster 127.0.0.1 6379 #quorum 1/1 【客观下线】

.....

+vote-for-leader 17818eb9240c8a625d2c8a13ae9d99ae3a70f9d2 1 【选举 leader】

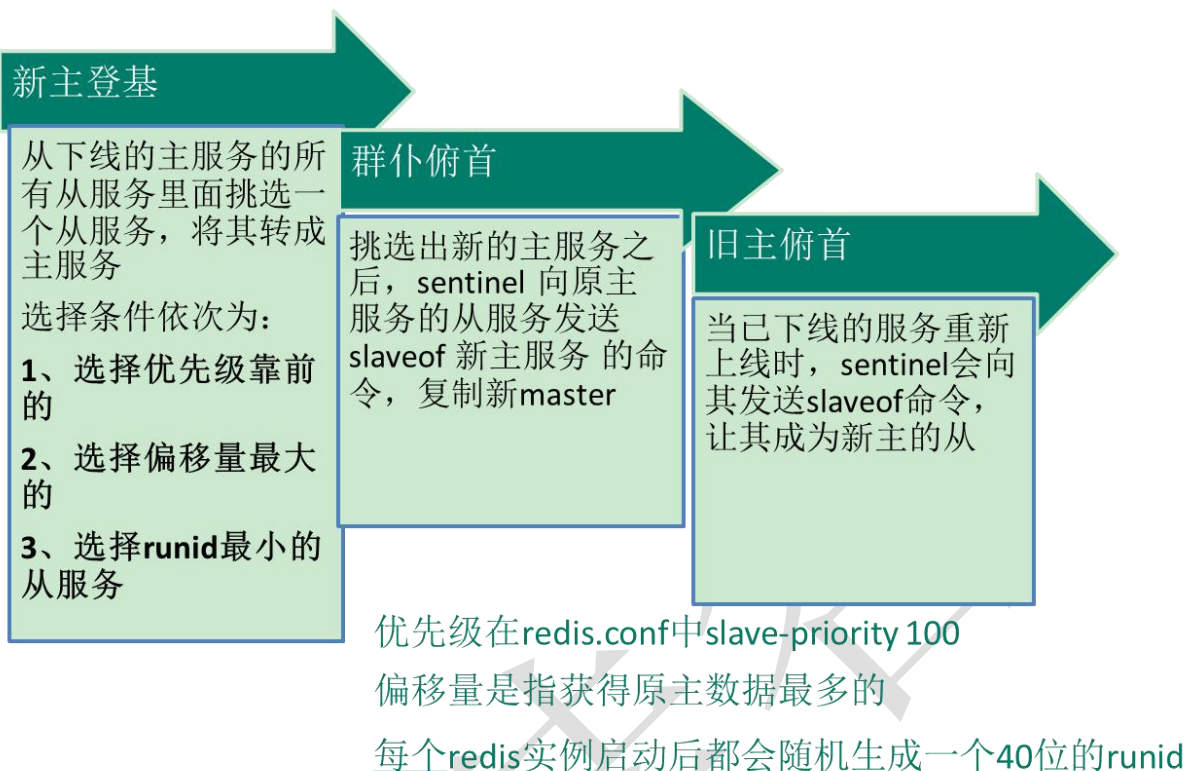
.....

+failover-state-send-slaveof-noone slave 127.0.0.1:6381 127.0.0.1 6381 @ mymaster 127.0.0.1 6379 【把一个从机设置为主机】

-----挂掉的主机又重新启动-----

-sdown slave 127.0.0.1:6379 127.0.0.1 6379 @ mymaster 127.0.0.1 6381 【离开主观下线状态】

+convert-to-slave slave 127.0.0.1:6379 127.0.0.1 6379 @ mymaster 127.0.0.1 6381 【转换为从机】



第 8 章 Jedis


1. 简介

Jedis 指通过 Java 连接 Redis 客户端。需要导入 jar 包：

MySQL	Redis
Connection 对象	Jedis 对象
连接池/数据源	连接池
Connection 对象每次用完需要关闭	Jedis 对象每次用完需要关闭

2. 配置

①导包：

 commons-pool2-2.4.2.jar
 jedis-2.9.0.jar

②在所连接的 Redis 客户端的配置文件中：

- 注释掉 bind 127.0.0.1
- 关闭保护模式，将 `protected-mode` 设置为 `no`
- 禁用 Linux 防火墙，执行 `service iptables stop`

③ 测试连接

```
@Test

public void testConnect() {

    //连接指定的redis，需要ip地址和端口号

    Jedis jedis=new Jedis("192.168.162.128", 6379);

    String ping = jedis.ping();

    System.out.println(ping);

}
```

3. 常用 API

3.1 测试 key

```
@Test

public void testKeys() {

    //连接指定的redis，需要ip地址和端口号

    Jedis jedis=new Jedis("192.168.162.128", 6379);

    //获取所有key

    Set<String> keys = jedis.keys("*");

}
```

```
for (String key : keys) {  
    System.out.println(key);  
}  
  
//判断是否存在某个key  
System.out.println("是否存在k2:"+jedis.exists("k2"));  
  
//测试某个key的过期时间  
System.out.println("k1的存活时间:"+jedis.ttl("k2"));  
  
jedis.close();  
}
```

3.2 测试 string

```
@Test  
  
public void testString() {  
    //连接指定的redis, 需要ip地址和端口号  
    Jedis jedis=new Jedis("192.168.162.128", 6379);  
  
    System.out.println("获取K1的值: "+jedis.get("k1"));  
  
    jedis.msetnx("k11", "v12", "k22", "v22", "k33", "v33");  
  
    System.out.println(jedis.mget("k11", "k22", "k33"));  
  
    //关闭连接  
    jedis.close();  
}
```

3.3 测试 list

```
@Test
```

```
public void testList() {  
  
    //连接指定的redis, 需要ip地址和端口号  
  
    Jedis jedis=new Jedis("192.168.162.128", 6379);  
  
    jedis.lpush("mylist", "1","2","3","4");  
  
    List<String> list = jedis.lrange("mylist", 0, -1);  
  
    for (String element : list) {  
  
        System.out.println(element);  
  
    }  
  
    //关闭连接  
  
    jedis.close();  
  
}
```

3.5 测试 set

```
@Test  
  
public void testSet() {  
  
    //连接指定的redis, 需要ip地址和端口号  
  
    Jedis jedis=new Jedis("192.168.162.128", 6379);  
  
    //添加元素  
  
    jedis.sadd("mySet", "Jack","Marry","Tom","Tony");  
  
    //删除指定元素  
  
    jedis.srem("mySet", "Tony");  
  
    //获取指定key的元素  
  
    Set<String> smembers = jedis.smembers("mySet");  
  
    for (String member : smembers) {  
  
        System.out.println(member);  
  
    }  
  
}
```

```
//关闭连接

jedis.close();

}
```

3.5 测试 hash

```
@Test

public void testHash() {

    //连接指定的redis, 需要ip地址和端口号

    Jedis jedis=new Jedis("192.168.162.128", 6379);

    jedis.hset("myHash", "username", "Jack");

    jedis.hset("myHash", "password", "123123");

    jedis.hset("myHash", "age", "11");

    //将多个数据封装为一个map

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

    map.put("gender", "male");

    map.put("department", "研发部");

    //批量设置多个数据

    jedis.hmset("myHash", map);

    List<String> values = jedis.hmget("myHash", "username", "password");

    for (String val : values) {

        System.out.println(val);

    }

    //关闭连接

    jedis.close();

}
```

3.6 测试 zset

```
@Test

    public void testZset() {

        //连接指定的redis, 需要ip地址和端口号

        Jedis jedis=new Jedis("192.168.162.128", 6379);

        jedis.zadd("myZset", 100, "math");

        //将多个数据封装为一个map

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

        subject.put("chinese", 88d);

        subject.put("english", 86d);

        //批量添加数据

        jedis.zadd("myZset", subject);

        Set<String> zset = jedis.zrange("myZset", 0, -1);

        for (String val : zset) {

            System.out.println(val);

        }

        //关闭连接

        jedis.close();

    }
```

4. 使用连接池

连接池的好处：节省每次连接 redis 服务带来的消耗，将创建好的连接实例反复利用；

4.1 连接池常用参数

参数	含义
MaxTotal	控制一个 pool 可分配多少个 jedis 实例，通过 pool.getResource()来获取；如果赋值为 -1，则表示不限制；如果 pool 已经分配了 MaxTotal 个 jedis 实例，则此时 pool 的状态为 exhausted。
maxIdle	控制一个 pool 最多有多少个状态为 idle(空闲)的 jedis 实例。
MaxWaitMillis	表示当 borrow 一个 jedis 实例时，最大的等待毫秒数，如果超过等待时间，则直接抛 JedisConnectionException。
testOnBorrow	获得一个 jedis 实例的时候是否检查连接可用性 (ping())；如果为 true，则得到的 jedis 实例均是可用的。

4.2 示例

```
@Test
public void testPool() {
    //默认的连接池配置
    GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
    System.out.println(poolConfig);
    JedisPool jedisPool=new JedisPool(poolConfig, "192.168.4.128", 6379,60000);
    Jedis jedis = jedisPool.getResource();
    String ping = jedis.ping();
    System.out.println(ping);
    //如果是从连接池中获取的，那么执行close方法只是将连接放回到池中
    jedis.close();
    jedisPool.close();}
```


第 9 章 案例

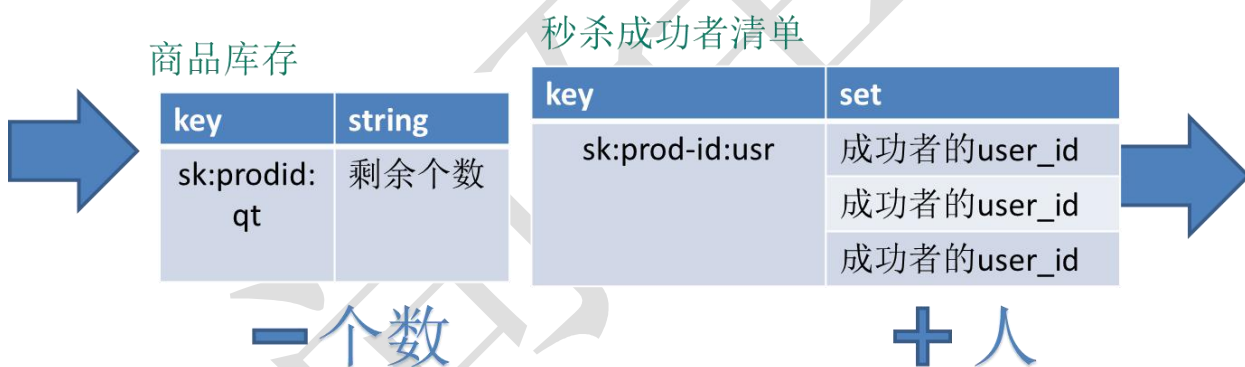
1. 手机验证码

需求:

- ①输入手机号，点击发送后随机生成 6 位数字码，2 分钟有效
- ②输入验证码，点击验证，返回成功或失败
- ③每个手机号每天只能输入 3 次

2. 秒杀

2.1 实现思路



2.2 数据库连接池:

```
public class JedisPoolUtil {  
  
    private static volatile JedisPool jedisPool = null;  
  
    private JedisPoolUtil() {  
  
    }  
}
```

```
public static JedisPool getJedisPoolInstance() {  
  
    if (null == jedisPool) {  
  
        synchronized (JedisPoolUtil.class) {  
  
            if (null == jedisPool) {  
  
                JedisPoolConfig poolConfig = new JedisPoolConfig();  
  
                poolConfig.setMaxTotal(200); //最大连接数  
  
                poolConfig.setMaxIdle(32); //最大空闲连接数  
  
                //获取连接时的最大等待毫秒数,如果超时就抛异常,默认-1  
  
                poolConfig.setMaxWaitMillis(100*1000);  
  
                //连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true  
  
                poolConfig.setBlockWhenExhausted(true);  
  
                //在获取连接的时候检查有效性, 默认false  
  
                poolConfig.setTestOnBorrow(true);  
  
                jedisPool = new JedisPool(poolConfig, "192.168.162.128", 6379, 60000 );  
  
            }  
  
        }  
  
    }  
  
    return jedisPool;  
  
}
```

2.3. 使用 ab 进行压力测试

```
ab -c 10 -n 100 -p /root/postarg -T application/x-www-form-urlencoded http://192.168.107.1:8080/seckill/doseckill
```

参数含义:

- c:模拟多少个客户端
- n:测试几次
- T:内容类型
- P:参数文件（注意参数文件的参数要以&符号结尾，注意携带绝对路径）

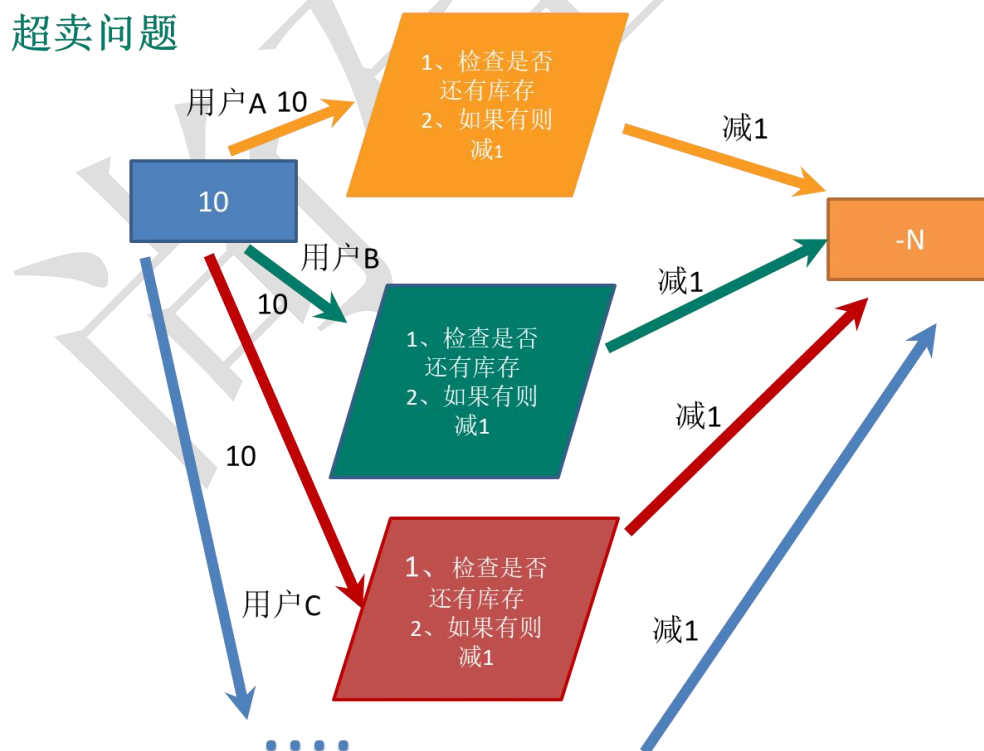
postarg 文件:

```
prodid=1001&
```

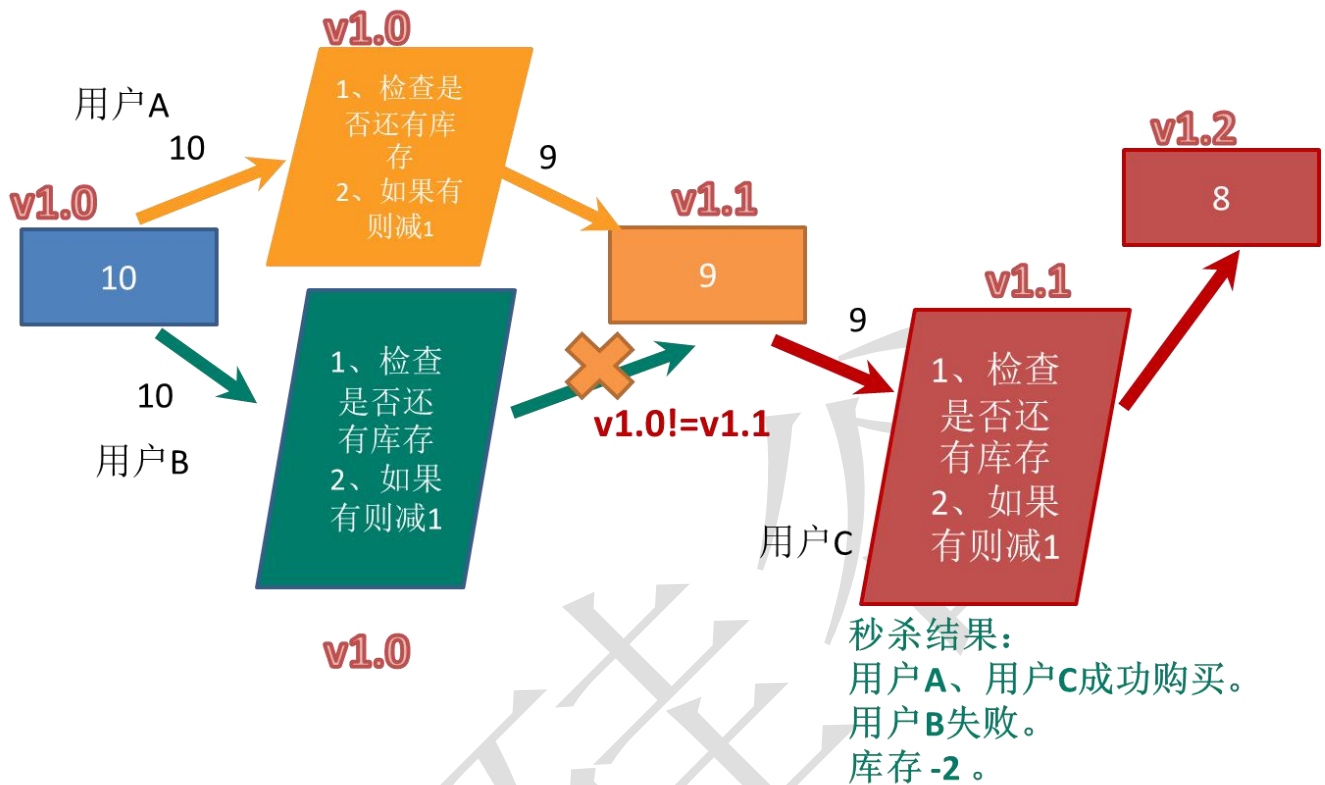
2.4 超卖问题

```
127.0.0.1:6379> get sk:1001:product
"-83"
```

超卖问题



解决：使用乐观锁解决超卖！



2.5 秒杀冗余问题

2.5.1 压测脚本

```
ab -c 200 -n 2000 -p /root/postarg -T application/x-www-form-urlencoded http://192.168.107.1:8080/seckill/doseckill
```

2.5.2 如何解决

使用 lua 脚本解决！

2.5.3 Lua

Lua 是一个小巧的脚本语言，Lua 脚本可以很容易的被 C/C++ 代码调用，也可以反过来调用 C/C++ 的函数，Lua 并没有提供强大的库，一个完整的 Lua 解释器不过 200k，所以 Lua 不适合作为开发独立应用程序的语言，而是作为嵌入式脚本语言



很多应用程序、游戏使用 LUA 作为自己的嵌入式脚本语言，以此来实现可配置性、可扩展性。这其中包括魔兽争霸地图、魔兽世界、博德之门、愤怒的小鸟等众多游戏插件或外挂。

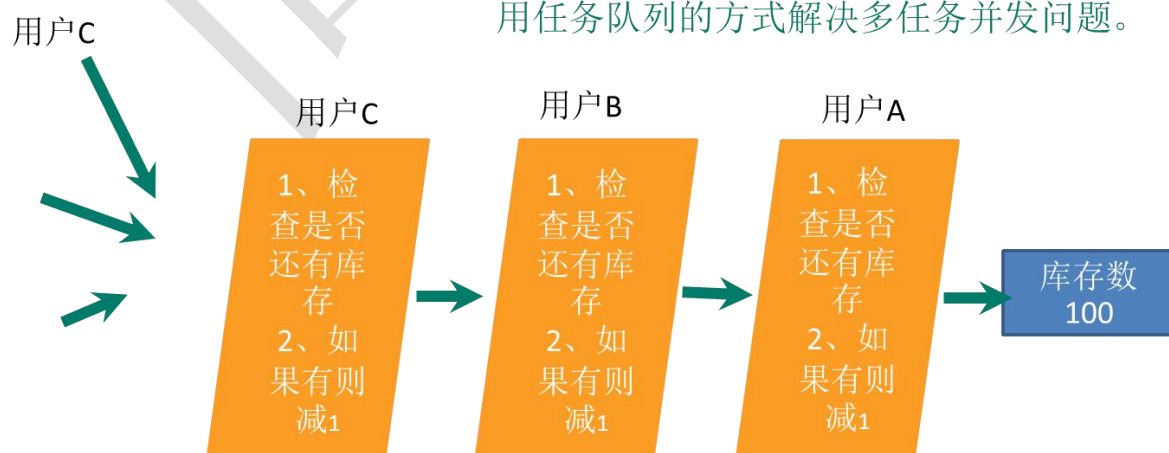
LUA 脚本在 Redis 中的优势：

将复杂的或者多步的 redis 操作，写为一个脚本，一次提交给 redis 执行，减少反复连接 redis 的次数。提升性能。

LUA 脚本是类似 redis 事务，有一定的原子性，不会被其他命令插队，可以完成一些 redis 事务性的操作。

但是注意 redis 的 lua 脚本功能，只有在 2.6 以上的版本才可以使用。

redis 2.6 版本以后，通过 lua 脚本解决争抢问题，实际上是 redis 利用其单线程的特性，用任务队列的方式解决多任务并发问题。



第 10 章 Redis Cluster

1. 引入集群

问题：

- 容量不够，redis 如何进行扩容？
- 并发写操作，redis 如何分摊？

什么是集群：

Redis 集群实现了对 Redis 的水平扩容，即启动 N 个 redis 节点，将整个数据库分布存储在这 N 个节点中，每个节点存储总数据的 $1/N$ 。

Redis 集群通过分区（partition）来提供一定程度的可用性（availability）：即使集群中有一部分节点失效或者无法进行通讯，集群也可以继续处理命令请求。

2. 创建集群

2.1 安装 ruby 环境：

本身 redis 集群的安装是很麻烦了，通过 ruby 工具，可以非常方便的将一系列命令打包为一个脚本！

（1）无网环境（CentOS6）

依次执行在安装光盘下的 Package 目录(/media/CentOS_6.8_Final/Packages)下的 rpm 包：

```
rpm -ivh compat-readline5-5.2-17.1.el6.x86_64.rpm
```

```
rpm -ivh ruby-libs-1.8.7.374-4.el6_6.x86_64.rpm
```

```
rpm -ivh ruby-1.8.7.374-4.el6_6.x86_64.rpm
```

```
rpm -ivh ruby-irb-1.8.7.374-4.el6_6.x86_64.rpm
```

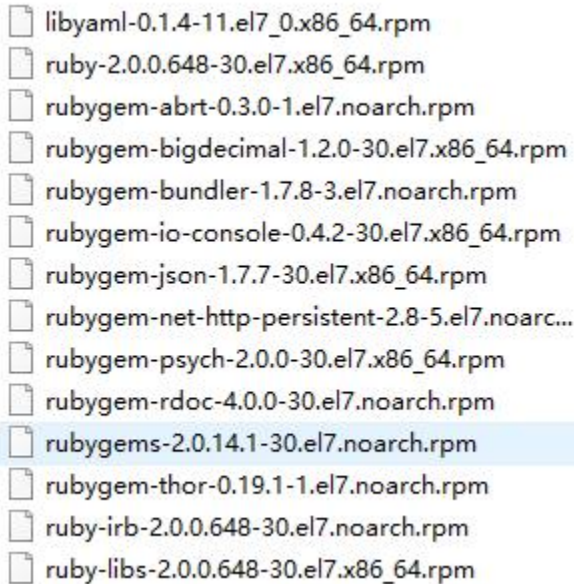
```
rpm -ivh ruby-rdoc-1.8.7.374-4.el6_6.x86_64.rpm
```



```
rpm -ivh rubygems-1.3.7-5.el6.noarch.rpm
```

(2) 无网环境 (CentOS7)

进入 Package 目录(/run/media/root/CentOS 7 x86_64/Packages), 拷贝如下安装包到独立目录 rpmruby



- libyaml-0.1.4-11.el7_0.x86_64.rpm
- ruby-2.0.0.648-30.el7.x86_64.rpm
- rubygem-abrt-0.3.0-1.el7.noarch.rpm
- rubygem-bigdecimal-1.2.0-30.el7.x86_64.rpm
- rubygem-bundler-1.7.8-3.el7.noarch.rpm
- rubygem-io-console-0.4.2-30.el7.x86_64.rpm
- rubygem-json-1.7.7-30.el7.x86_64.rpm
- rubygem-net-http-persistent-2.8-5.el7.noarch...
- rubygem-psych-2.0.0-30.el7.x86_64.rpm
- rubygem-rdoc-4.0.0-30.el7.noarch.rpm
- rubygems-2.0.14.1-30.el7.noarch.rpm**
- rubygem-thor-0.19.1-1.el7.noarch.rpm
- ruby-irb-2.0.0.648-30.el7.noarch.rpm
- ruby-libs-2.0.0.648-30.el7.x86_64.rpm

进入此目录运行如下命令, 按照依赖安装

```
rpm -Uvh *.rpm --nodeps --force
```

(3) 联网环境

在联网状态下, 执行 yum 安装, 执行 `yum -y install ruby`;

之后安装 rubygem, rubygem 是 ruby 的包管理框架。 `yum -y install rubygems`

2.2 安装 redis gem

redis-3.2.0.gem 是一个通过 ruby 操作 redis 的插件!

拷贝 redis-3.2.0.gem 到/opt 目录下, 在 opt 目录下执行 `gem install --local redis-3.2.0.gem`;

2.3 制作 6 个 redis 配置文件

端口号分别是：6379,6380,6381,6389,6390,6391

注意：每个配置文件中需要指定

`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。

`cluster-enabled yes` 打开集群模式

`cluster-config-file nodes-6379.conf` 设定节点配置文件名

`cluster-node-timeout 15000` 设定节点失联时间, 超过该时间 (毫秒), 集群自动进行主从切换。

样例:

```
include /usr/local/myredis/rediscluster/redis_base.conf

pidfile "/var/run/redis_6379.pid"

port 6379

dbfilename "dump_6379.rdb"

cluster-enabled yes

cluster-config-file nodes-6379.conf

cluster-node-timeout 15000
```

注意在创建集群的时候, 初始化的时候, 把所有节点的 `dump` 文件全部删掉。

2.4 开启集群

①首先依次启动 6 个节点，启动后，会在当前文件夹生成 nodes-xxxx.conf 文件

```
-rw-r--r--. 1 root root 757 4月 6 13:33 nodes-6379.conf
-rw-r--r--. 1 root root 757 4月 6 13:33 nodes-6380.conf
-rw-r--r--. 1 root root 757 4月 6 13:33 nodes-6381.conf
-rw-r--r--. 1 root root 757 4月 6 13:33 nodes-6389.conf
-rw-r--r--. 1 root root 757 4月 6 13:33 nodes-6390.conf
-rw-r--r--. 1 root root 757 4月 6 13:33 nodes-6391.conf
```

②配置集群

在/opt/redis-3.2.5/src 目录下，执行命令：

```
./redis-trib.rb create --replicas 1 192.168.31.211:6379 192.168.31.211:6380 192.168.31.211:6381
```

```
192.168.31.211:6389 192.168.31.211:6390 192.168.31.211:6391
```

注意，此处不要用 127.0.0.1，请用真实 IP 地址！

```
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

```
[root@vm1 src]# ps -ef|grep redis
root      3713      1  0 22:52 ?        00:00:01 redis-server *:6379 [cluster]
root      3717      1  0 22:52 ?        00:00:01 redis-server *:6380 [cluster]
root      3721      1  0 22:52 ?        00:00:01 redis-server *:6381 [cluster]
root      3725      1  0 22:52 ?        00:00:00 redis-server *:6389 [cluster]
root      3730      1  0 22:52 ?        00:00:00 redis-server *:6390 [cluster]
root      3735      1  0 22:52 ?        00:00:00 redis-server *:6391 [cluster]
```

③之后登录到客户端，通过 cluster nodes 命令查看集群信息

```
127.0.0.1:6379> cluster nodes
9d46b569c347a6848329a7f75fa7f6ed9743fbc1 192.168.4.128:6379 myself,master - 0 0 1 connected 0-5460
77e100f34644b6eea861a0ff634e3a8cb78bb06b 192.168.4.128:6380 master - 0 1523372352250 2 connected 5461-10922
f0ecd3483827ad2236f5f3889e11a38dd70fa2d1 192.168.4.128:6389 slave 9d46b569c347a6848329a7f75fa7f6ed9743fbc1 0 1523372351243 4 connected
700458389820e7788c08b62f2d174f8f8797b0a7 192.168.4.128:6391 slave 8674e072e79c6fdb29532ab00f2952501a5fda18 0 1523372349731 6 connected
91b714f8fea29eae34c92e920f2a4da2b96e44f0 192.168.4.128:6390 slave 77e100f34644b6eea861a0ff634e3a8cb78bb06b 0 1523372349227 5 connected
8674e072e79c6fdb29532ab00f2952501a5fda18 192.168.4.128:6381 master - 0 1523372348209 3 connected 10923-16383
```

④6 个节点，为什么是三主三从？

配置机器，至少需要 6 个节点，否则会报错：

```
>>> Creating cluster
*** ERROR: Invalid configuration for cluster creation.
*** Redis Cluster requires at least 3 master nodes.
*** This is not possible with 2 nodes and 1 replicas per node.
*** At least 6 nodes are required.
```

命令 `create`, 代表创建一个集群。参数 `--replicas 1` 表示我们希望为集群中的每个主节点创建一个从节点。一个集群至少要有三个主节点, 分配原则尽量保证每个主数据库运行在不同的 IP 地址, 每个从库和主库不在一个 IP 地址上。

2.5 slot

进入集群后, 如果我们, 直接写入数据, 可能会看到报错信息:

```
127.0.0.1:6379> set k1 v1
(error) MOVED 12706 192.168.4.128:6381
127.0.0.1:6379> set k2 v2
OK
127.0.0.1:6379> set k3 v3
OK
```

这是因为, 集群中多了 slot(插槽)的设计。一个 Redis 集群包含 16384 个插槽 (hash slot), 数据库中的每个键都属于这 16384 个插槽的其中一个, 集群使用公式 $CRC16(key) \% 16384$ 来计算键 key 属于哪个槽, 其中 $CRC16(key)$ 语句用于计算键 key 的 CRC16 校验和。

集群中的每个节点负责处理一部分插槽。举个例子, 如果一个集群可以有主节点, 其中:

节点 A 负责处理 0 号至 5500 号插槽。

节点 B 负责处理 5501 号至 11000 号插槽。

节点 C 负责处理 11001 号至 16383 号插槽。

2.6 集群中写入数据

2.6.1 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。

②`redis-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。

```
[root@vml src]# redis-cli -c -p 6379
127.0.0.1:6379> set k4 v4
-> Redirected to slot [8455] located at 192.168.4.128:6380
OK
```

③每个 slot 可以存储一批键值对。

2.6.2 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。

```
192.168.4.128:6380> mset k11 v11 k12 v12 k13 v13 k14 v14 k15 v15
(error) CROSSSLOT Keys in request don't hash to the same slot
```

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。

```
192.168.4.128:6380> mset {key}k11 v11 {key}k12 v12 {key}k13 v13 {key}k14 v14 {key}k15 v15
-> Redirected to slot [12539] located at 192.168.4.128:6381
OK
```

2.7 集群中读取数据

➤ CLUSTER KEYSLOT <key> 计算键 key 应该被放置在哪个槽上

```
192.168.4.128:6381> CLUSTER KEYSLOT k1
(integer) 12706
```

➤ CLUSTER COUNTKEYSINSLOT <slot> 返回槽 slot 目前包含的键值对数量

```
192.168.4.128:6381> CLUSTER COUNTKEYSINSLOT 12539
(integer) 5
```

➤ CLUSTER KEYSLOT <key>:计算 key 应该放在哪个槽

```
192.168.4.128:6381> CLUSTER KEYSLOT jack
(integer) 7830
```

➤ CLUSTER GETKEYSINSLOT <slot> <count> 返回 count 个 slot 槽中的键。

```
192.168.4.128:6381> CLUSTER GETKEYSINSLOT 12539 3
1) "{key}k11"
2) "{key}k12"
3) "{key}k13"
```

2.8 集群中故障恢复

问题 1：如果主节点下线？从节点能否自动升为主节点？

答：主节点下线，从节点自动升为主节点。


```
[root@vml ~]# redis-cli -h 192.168.4.128 -p 6380 shutdown
```

```
192.168.4.128:6381> cluster nodes
700458389820e7788c08b62fd174f8f8797b0a7 192.168.4.128:6391 slave 8674e072e79c6fdb29532ab00f2952501a5fda18 0 1523376194991 6 connected
f0ecd3483827ad2236f5f3889e11a38dd70fa2d1 192.168.4.128:6389 slave 9d46b569c347a6848329a7f75fa7f6ed9743fbc1 0 1523376196003 4 connected
9d46b569c347a6848329a7f75fa7f6ed9743fbc1 192.168.4.128:6379 master - 0 1523376192972 1 connected 0-5460
77e100f34644b6eea861a0ff634e3a8cb78bb06b 192.168.4.128:6380 master,fail - 1523376075223 1523376072296 2 disconnected
91b714f8fea29eae34c92e920f2a4da2b96e44f0 192.168.4.128:6390 master - 0 1523376197015 7 connected 5461-10922
8674e072e79c6fdb29532ab00f2952501a5fda18 192.168.4.128:6381 myself,master - 0 0 3 connected 10923-16383
```

问题 2：主节点恢复后，主从关系会如何？

主节点恢复后，主节点变为从节点！

```
192.168.4.128:6381> cluster nodes
700458389820e7788c08b62fd174f8f8797b0a7 192.168.4.128:6391 slave 8674e072e79c6fdb29532ab00f2952501a5fda18 0 1523376351051 6 connected
f0ecd3483827ad2236f5f3889e11a38dd70fa2d1 192.168.4.128:6389 slave 9d46b569c347a6848329a7f75fa7f6ed9743fbc1 0 1523376352061 4 connected
9d46b569c347a6848329a7f75fa7f6ed9743fbc1 192.168.4.128:6379 master - 0 1523376353071 1 connected 0-5460
77e100f34644b6eea861a0ff634e3a8cb78bb06b 192.168.4.128:6380 slave 91b714f8fea29eae34c92e920f2a4da2b96e44f0 0 1523376348991 7 connected
91b714f8fea29eae34c92e920f2a4da2b96e44f0 192.168.4.128:6390 master - 0 1523376350043 7 connected 5461-10922
8674e072e79c6fdb29532ab00f2952501a5fda18 192.168.4.128:6381 myself,master - 0 0 3 connected 10923-16383
```

问题 3：如果所有某一段插槽的主从节点都宕掉，redis 服务是否还能继续？

答：服务是否继续，可以通过 redis.conf 中的 cluster-require-full-coverage 参数进行控制。

主从都宕掉，意味着有一片数据，会变成真空，没法再访问了！

如果无法访问的数据，是连续的业务数据，我们需要停止集群，避免缺少此部分数据，造成整个业务的异常。此时可以通过配置 cluster-require-full-coverage 为 yes。

如果无法访问的数据，是相对独立的，对于其他业务的访问，并不影响，那么可以继续开启集群提供服务。此时，可以配置 cluster-require-full-coverage 为 no。

2.9 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");
}
```


2.10 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。