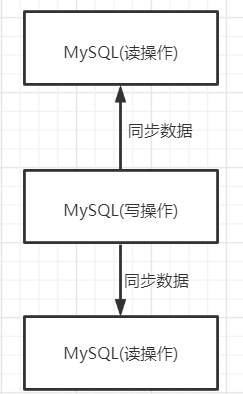
# Redis

## 一、Nosql

缓存主要是用来解决读数据的问题。

对数据库进行垂直拆分，读写分离操作。



使用分库分表来解决写的压力。对数据库进行水平拆分，MySQL集群。MySQL是关系型数据库。

用户的个人信息非常多，比如社交网络、地理位置、用户产生的数据、用户日志等呈爆发式的增长，这个时候使用MySQL就不能很好的处理这些数据，因此需要使用NoSQL数据库。

### 1.关系型数据库和非关系型数据库

常见的**关系型数据库**有：MySQL、Oracle、DB2等。

常见的**非关系型数据库**有：Redis、mongoDB等。

#### ⑴关系型数据库

**关系型数据库**：指采用了关系模型来组织数据的数据库。简单的说，关系模型指的是二维表格模型，而一个关系型数据库就是由二维表及表之间的联系所组成的一个数据组织。

**关系模型中常用概念**：

关系：可理解为一张二维表。每个关系都具有一个关系名，就是通常说的表名。

元组：可以理解为二维表中的一行，在数据库中经常被称为记录。

属性：可以理解为二维表中的一列，在数据库中经常被称为字段

域：属性的取值范围，也就是数据库中某一列的取值限制

**关系型数据库的优点**：

①易于维护：都是使用表结构，格式一致；

②使用方便：SQL语言通用；

③**支持事务控制，从而保证的数据的一致性**。

**关系型数据库的缺点**：

①对于高并发读写需求，传统关系型数据库来说，**硬盘I/O是一个很大的瓶颈**。

②表结构固定，灵活性较差。

**目前对于网站来说，关系型数据库的很多特性不再需要了**，比如：**事务的一致性方面**，现在很多web2.0系统对事物的读写一致性要求都不高；**读写实时性方面**，关系型数据库成功插入一条数据之后是可以立马读出该数据的，但是对于很多web应用来说，并不要求这么高的实时性，比如发一条消息之后，过几秒乃至十几秒之后才看到这条动态是完全可以接受的。

#### ⑵非关系型数据库

**NoSQL：Not Only SQL。泛指非关系型数据库**。

非关系型数据库严格来说不是一种数据库，而是**一种数据结构化存储方法的集合，可以是文档或者键值对等。**

**非关系型数据库的分类**：

**①key-value数据库**：数据是以key-value形式存储的。这种数据库的特点是具有极高的并发读写性能，代表有Redis。

主要应用场景：内容缓存，一些日志系统，或者用于处理大量数据的高并发操作。

**②文档型数据库**：这个主要是面向海量数据访问的文档存储，存储结构与key-value类似，但是这个value主要是以JSON或xml等格式的文档来进行存储。典型代表为MongoDB。MongoDB是一个基于分布式文件存储的数据库，使用C++编写的，是一个介于关系型数据库和非关系型数据库的中间产品，它是最像关系型数据库的。

主要应用场景：Web应用。

**③列存储数据库**：面向列的数据库是以列为单位来存储数据的，使系统具有较高的可扩展性。这类数据库可以适应数据量的增加以及数据结构的变化。代表有Hbase。

主要应用场景：分布式的文件系统中。

**④图形数据库**：存放的不是图形，而是关系。比如朋友圈社交。

**非关系型数据库的优缺点：**

**优点**：

①存储数据的格式较灵活，可以是key-value形式、文档、图片等形式。

②成本低：非关系型数据库基本上都是开源软件。

③方便扩展：数据之间没有关系，很好扩展。

④高性能。

**缺点**：

①不支持sql，造成开发人员额外的学习成本。

②不支持事务处理。

## 二、Redis入门

### 1.基本概念

Redis(Remote Dictionary Server)：远程字典服务。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型、key-value数据库。 Redis可以用作数据库、缓存、消息中间件。

**Redis的作用**：

①内存存储、持久化。内存中存放的数据是断电即丢失的，所以持久化很重要。

②效率和效率高，可用于高速缓存。读的速度是11万次/s,写的速度是8万次/s。

③发布订阅系统。

④地图信息分析。

⑤计时器、计数器(浏览量)。

Redis的特性：

①多样的数据类型。

②持久化

③集群

④支持事务控制。

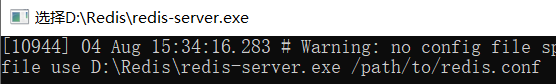
Redis官网：<https://redis.io/>。中文网：<http://www.redis.cn/>。

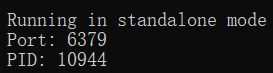
### 2.Redis安装

#### ⑴Windows安装Redis

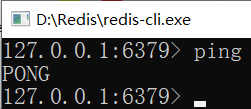
直接将下载的Redis压缩包解压即可。

双击redis-server.exe启动Redis服务端。Redis的默认端口是6379。

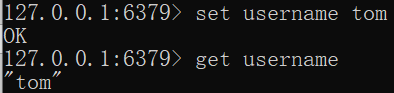




在Redis服务端不关闭的情况下，双击redis-cli.exe启动Redis的客户端，从而连接上Redis服务端。输入ping命令回车，若出现PONG则表示连接成功。



使用set key value的形式设置值，get key的方式获得值。

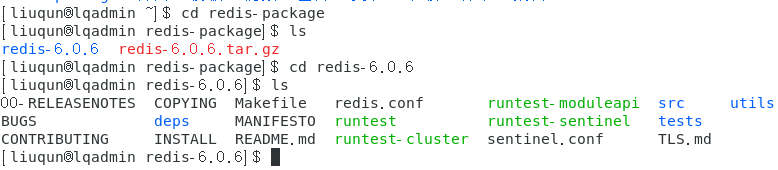


官方建议在Linux下使用Redis。

#### ⑵Linux安装Redis

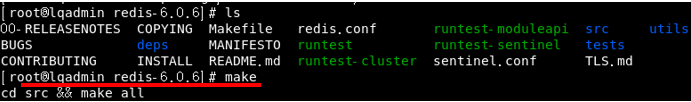
①从<https://redis.io/>下载Redis的压缩包，然后放在Linux的home目录下的文件夹中，之后解压该压缩包即可。

②进入解压后的文件可以看到redis.conf即Redis的配置文件。

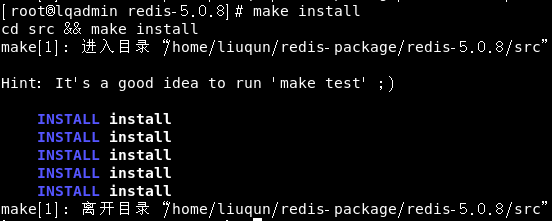


③安装gcc环境：使用yum install gcc-c++安装gcc。安装成功之后运行gcc -v出现gcc的版本信息。

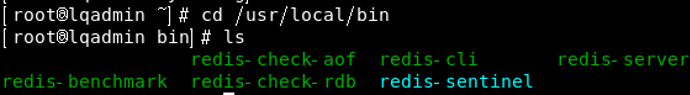
④之后进入redis的解压之后的目录下，执行make命令。注意不要使用最新的6.0以上版本的Redis，因为可能make编译时报错，6.0以上的版本需要升级C++的编译版本。使用5.0.8版本的Redis在make、make install时都能成功。



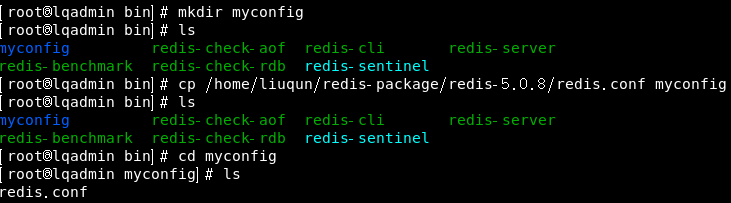
⑤执行make成功之后在执行make install。



Redis默认的安装位置是在**/usr/local/bin**目录下。

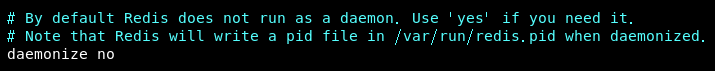


⑥将redis解压后文件夹下的redis.conf文件复制到/usr/local/bin目录下的新建myconfig文件夹下。之后就可以使用myconfig文件夹下的redis.conf配置文件进行启动。

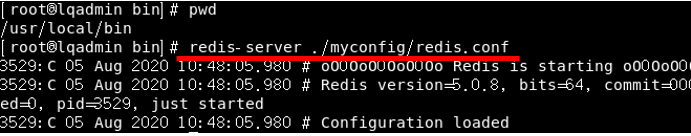


⑦Redis默认不是后台启动了，需要修改redis.congf配置文件。进入myconfig文件夹，vim redis.conf进行修改配置文件。

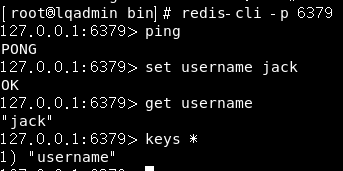
将下面的daemonize的值改为yes即可。改为yes之后按ESC之后：wq保存退出即可。



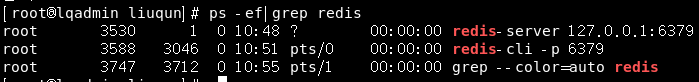
⑧启动Redis服务：在/usr/local/bin目录下，使用redis-server然后指定配置文件启动。./表示当前目录下。



⑨启动Redis客户端：同样在/usr/local/bin目录下，使用redis-cli -p 6379启动客户端。这里-p表示端口号。这里因为是在本机连接，所以省略了-h。



查看redis的进程是否开启：ps -ef|grep redis命令。



关闭redis服务shutdown，之后exit退出。关闭之后查看redis的进程会发现redis的进程也关闭了。





## 三.Redis基本使用

### 1.性能测试

redis-benchmark是一个压力测试工具，是官方自带的性能测试工具。使用方法：redis-benchmark 命令参数。常用的参数如下：

-h：指定服务器主机名，默认是本机即127.0.0.1.

-p：指定服务器端口，默认是6379。

-s：指定服务器socket。

-c：指定并发连接数，默认为50。

-n：指定请求，默认是10000。

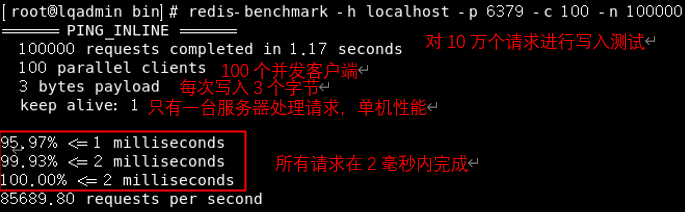
-d：以字节的形式指定请求的值的数据大小。

默认在本机上且服务器端口为6379则可以省略-h -p。

在保证启动redis服务端的情况下，执行如下命令：



对结果进行分析：



### 2.基本命令

启动Redis的服务端和客户端都是在usr/local/bin目录下。

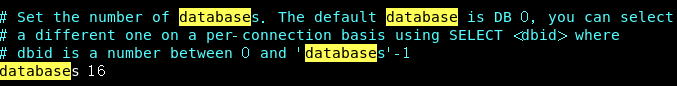
**启动Redis的服务端**：使用redis-server 指定配置文件启动。如：**redis-server ./myconfig/redis.conf**。

**启动Redis的客户端**： **redis-cli -p 6379**。可以省略默认端口号。若出现中文乱码，则启动时**redis-cli –-raw**。

**关闭Redis**：shutdown之后exit。

**查看redis的进程是否开启**：**ps -ef|grep redis**命令。

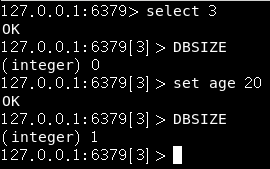
Redis默认有16个数据库。可以通过select 指定使用哪个数据库。数据库名是从0-15的，默认是0号数据库。select 3表示使用的是第四个数据库。





**查看数据库大小**：**DBSIZE**命令。

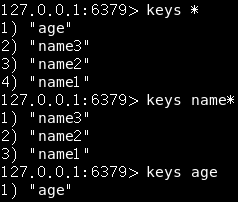
**切换数据库**：**select 数字**。



**往当前数据库中存放数据**：**set key value**形式。

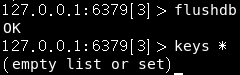
**从当前数据库中获取数据**：**get key**形式。

**查看当前数据库中的key**：**keys \***命令查看所有的key。**keys name\***，表示查看以name开头的所有key。



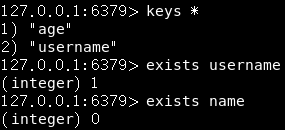
**清除当前数据库：flushdb**命令。

**清除全部数据库的内容：flushall**命令。

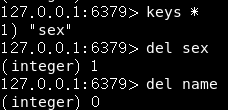




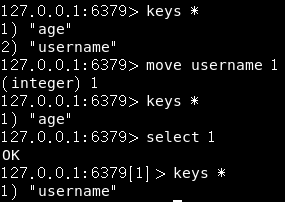
**判断当前key是否存在**：**exists key**。若key存在则返回1，否则返回0。



**删除指定的key：del key**。返回值为被删除的key的数量。

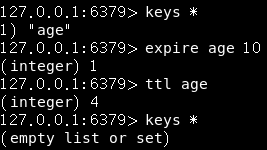


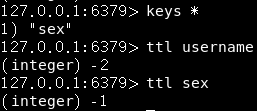
**将当前数据库中的key移动到指定的数据库中**：**move key dbnum**。



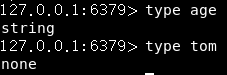
**为指定的key设置过期时间：expire key seconds**。以秒为单位。

**查看指定key的剩余生存时间：ttl key**。ttl是time to live的缩写。当key不存在时返回-2。当key 没有设置过期时间时，返回-1。否则返回key的剩余生存时间，以秒为单位。

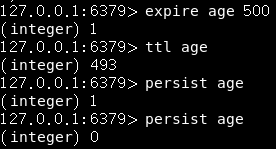




**查看指定key的数据类型**：**type key**。返回值为对应的数据类型，若返回none，则表示该key不存在。



**移除指定key的过期时间，使key永不过期**：**persist key**。移除成功则返回1，若key不存在或key没有设置过期时间，返回0。



Redis的其他指令不懂的可以在官网中搜索。



### 3.Redis的线程问题

Redis在6.0版本之前是单线程的，Redis是基于内存操作的，CPU不是Redis的性能瓶颈，Redis的瓶颈是内存和网络带宽，Redis的处理性能非常高，因此没必要使用多线程。

**多线程可能带来的问题**：可能导致程序的执行顺序不确定，可能带来并发读写问题，可能导致多个线程竞争同一资源造成死锁而导致性能损耗。

在Redis 6.0版本之后引入了多线程。Redis的瓶颈在于内存和网络。内存不够的化可以加内存。网络 IO 的读写在 Redis 整个执行期间占用了大部分的 CPU 时间，如果把网络处理这部分做成多线程处理方式，那对整个 Redis 的性能会有很大的提升。因此**Redis加入多线程的主要原因**是：充分利用cpu资源；多线程任务可以分摊Redis同步IO的读写负担。

注意：高性能的服务器不一定是多线程的。多线程也不一定比单线程的效率高。运行速度：cpu>内存>硬盘。

**Redis单线程为什么快**：

redis是将所有的数据全部放在内存中，若是多线程，则需要进行cpu上下文切换的，这个操作是耗时的，因此使用单线程去操作效率就高。对于内存系统来说，若没有上下文切换效率就是最高的。

Redis开启多线程之后，并不会存在并发安全问题，**因为Redis 的多线程部分只是用来处理网络数据的读写和协议解析，执行命令仍然是单线程顺序执行**。

## 四、五大数据类型

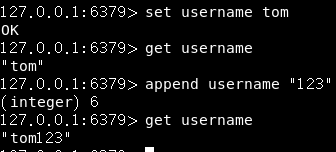
### 1.String(字符串)

默认是字符串类型。Redis中**命令**是不区分大小写的。

String类型的value可以是字符串也可以是数字。

**String类型的应用场景**：计数器、统计数量、粉丝数、对象缓存存储。

**往字符串中追加内容使用append命令**：**append key value**。若当前key不存在，则会创建该key。



**判断字符串的长度：strlen key**。

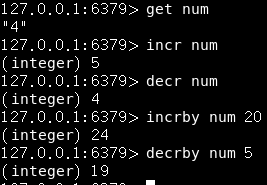


**对数据进行自增操作**：**incr key 将key自增1**。

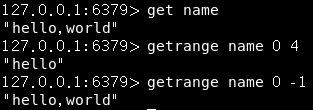
**对数据进行自减操作**：**decr key 将key自减**。

**对数据进行指定步长增加**：**incrby key num**。

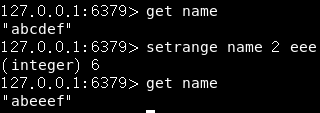
**对数据进行指定步长减少**：**decrby key num**。



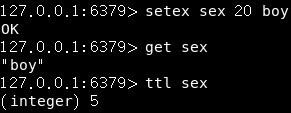
**截取字符串**：**getrange key start end**。字符串索引是从0开始的，是包括start 和end的内容。当end的值为-1表示截取到最后。



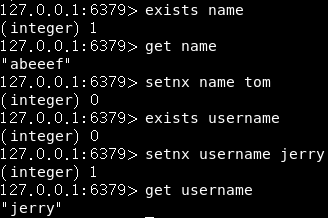
**替换指定位置开始的字符串**：**setrange key offset value**。是从offset开始。



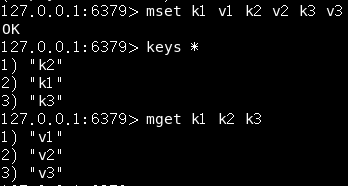
**setex(set with expire)命令**：指定的key设置值及其过期时间，如果key已经存在会替换旧的值。**setex key seconds value**。



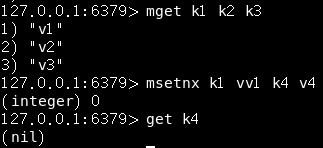
**setnx(set if not exist)命令**：在指定的key不存在时，为key设置值。 设置成功则返回1，设置失败则返回0。**setnx key value**。在分布式锁中会常常使用。



**批量设置值和批量获取值**：**mset k1 v1 k2 v2……；mget k1 k2……**。

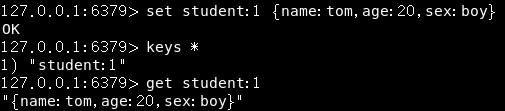


**msetnx则是给批量不存在的key设置值**。msetnx是一个原子操作，要么都成功，要么都失败。



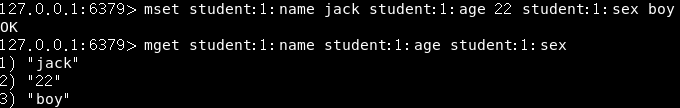
**在redis中设置对象**：

**法一**：如**set student:1 {name:tom,age:20,sex:boy}**形式。student:1可以作为key，值为json格式的字符串。

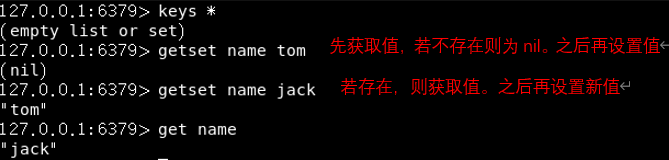


**法二**：也可以**使用mset、mget的方式设置和获取对象的值**。

student:1:name相当于key1,student:1:age相当于key2。

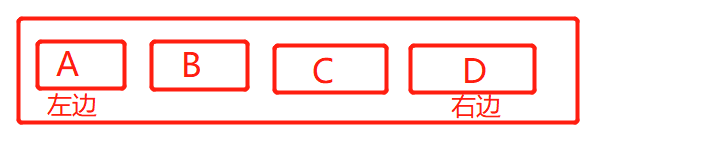


**getset命令**：先get获取值，然后再set设置值。



### 2.List(列表)

**所有的关于List操作的命令都是以l或r开头**。redis中的List的结构对应的是栈，先进后出。

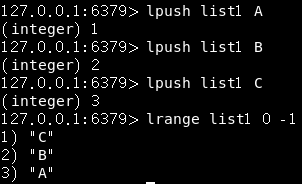


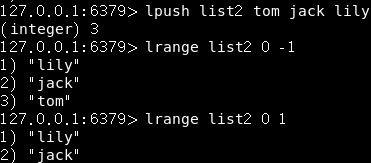
**List的应用场景**：消息排队。比如消息队列(先进先出lpush rpop)、栈(先进后出 lpush lpop)。

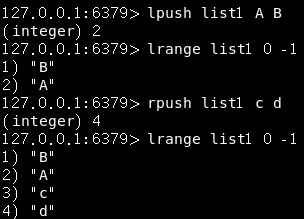
**往List头部添加元素(最左边)**：**lpush key value1 value2 value3……**。这是将一个或多个值依次插入到列的顶部，这部分元素是最先被遍历的。

**往List尾部添加元素(最右边)**：**rpush key value1 value2……**。这是将一个或多个值依次插入列的底部，这部分元素是最后被遍历的。

**遍历List中的元素**：**lrange key start end**。

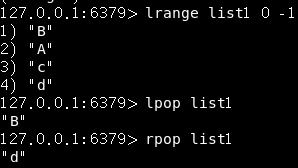




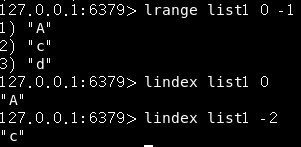


**移除List中左边第一个的元素**：**lpop key**。并返回移除的元素。

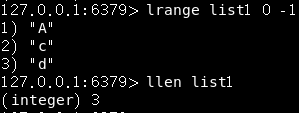
**移除List中右边的第一个元素**：**rpop key**。并返回移除的元素。



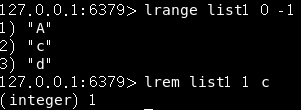
**通过索引获取List中的元素**：**lindex key indexnum**。索引下标也可以从负数开始，-1表示最后一个元素，-2表示倒数第二个元素。



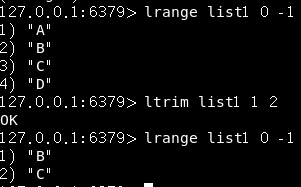
**返回List的长度：llen key**。



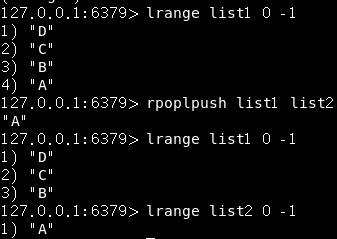
**移除List中指定的值**：**lrem key count value**。count为移除的数量。value为移除的值。当count=0时表示移除表中所有与value相同的值。当count>0表示从头部向尾部搜索，移除与value相同的值，移除的个数为count。当count<0表示从尾部向头部搜索，移除与value相同的值，移除个数为count的绝对值。返回值为被移除元素的数量。



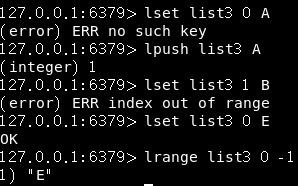
**ltrim命令**：**截取List中的元素**，让List只保留指定区间的元素，不在指定区间的元素将会被删除。**ltrim key start end**。



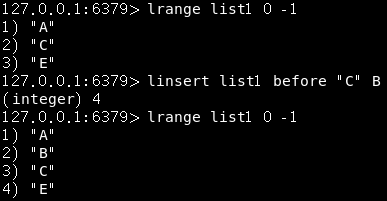
**rpoplpush命令**：移除列表的最后一个元素，并将该元素添加到另一个列表。返回值为被移除的元素。**rpoplpush src dest**。



**lset命令**：通过索引来修改List中的值。**lset key indexNum value**。若不存在该key，则会报错。lset命令不能向List中添加元素，只能根据索引修改List中元素的值。



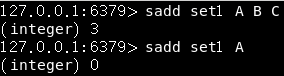
**linsert命令**：用于在List中的元素前或者元素后插入新元素。当指定元素不存在于列表中时，不执行任何操作。**linsert key before|after oldVal newVal**。



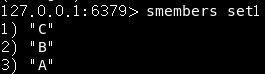
### 3.set(集合)

set中的值是不能重复的。set是无序不可重复的集合。

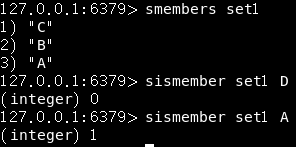
**往集合中添加元素**：**sadd key value1 value2……**。返回值为添加进集合中的元素个数。重复的元素不会被添加进set中。



**查看集合中的所有元素**：**smembers key**。



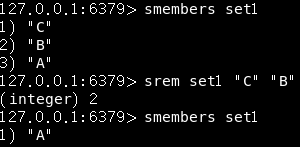
**判断某个元素是否存在于set中**：**sismember key value**。若存在，则返回1。若不存在或者key不存在则返回0。



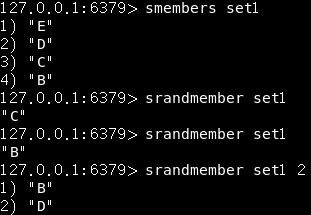
**获取集合中元素的个数：scard key**。



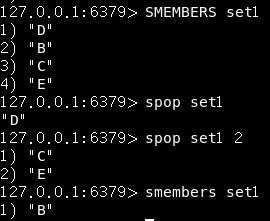
**移除集合中一个或多个元素：srem key value1 value2……**。返回值为被移除的元素个数。



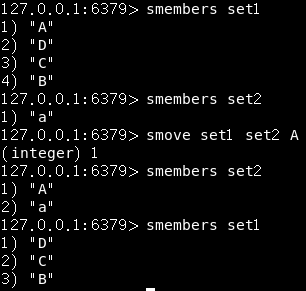
**随机获取集合中的元素**：srandmember key [count]。count为获取的元素个数，可选，不指定的化默认是一个。



**随机移除集合中一个或多个元素**：**spop key [count]**。count表示移除的元素个数，当不指定count时表示随机移除一个元素，count可选。返回值为移除的元素。



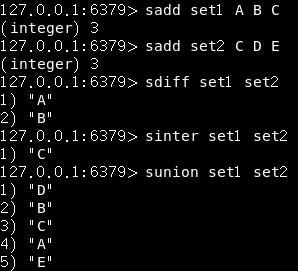
**将指定集合中的元素移动到另一个集合中**：**smove src dest val**。返回值为1表示元素移除成功，返回值为0表示移除失败。



**获取第一个集合中与其他集合不同的元素(差集)**：**sdiff key1 key2……**。

**获取所有集合的交集**：**sinter key1 key2……**。

**获取所有集合的并集**：**sunion key1 key2……**。

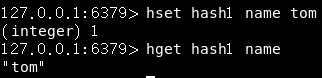


### 4.Hash(哈希)

Hash中的value是一个Map形式的，即key-value形式的。hash一般可以用来存储对象、经常变动的信息。String更适合字符串的存储。

**给Hash表的字段赋值**：**hset key field value**。若Hash表不存在则创建新的，若已存在则覆盖。

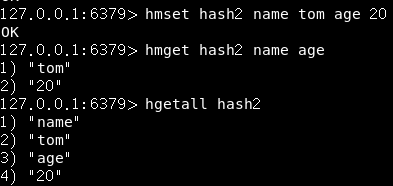
**获取Hash表中的字段的值**：**hget key field**。



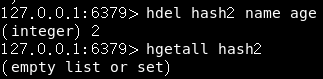
**同时给hash表中设置多个field-value值**：**hmset key f1 v1 f2 v2……**。

**同时获得hash表中多个field的值**：**hmget key f1 f2……**。

**返回hash表中所有的字段名和值**：**hgetall key**。

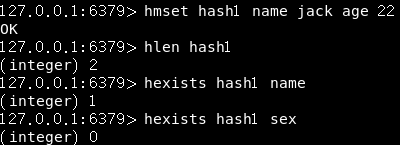


**删除hash表中一个或多个字段的值**：**hdel key field1 field2…**。返回值为被删除的字段数量。



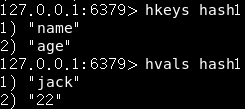
**判断hash表中字段的数量**：**hlen key**。返回值为hash表字段数量。当hash表不存在则返回0。

**判断hash表中是否存在某个字段**：**hexists key field**。若存在则返回1，若不存在或者hash表不存在则返回0。

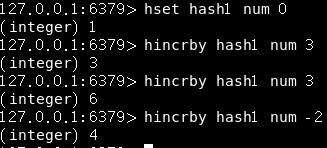


**获取hash表中所有的字段**：**hkeys key**。

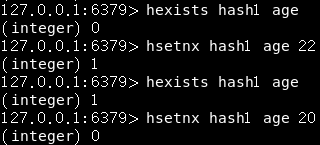
**获取hash表中所有的值**：**hvals key**。



**给hash表中指定的字段设置增量**：**hincrby key field increNum**。当increNum为正数则是增加，为负数则是减少。没有hdecrby的方法。



**给hash表中不存在的字段赋值**：**hsetnx key field value**。返回值为1表示设置成功，设置失败则返回0。

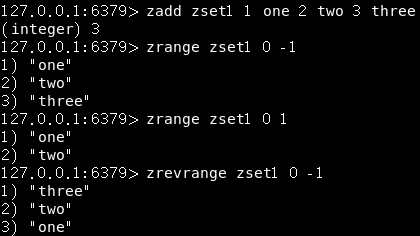


### 5.Zset(有序集合)

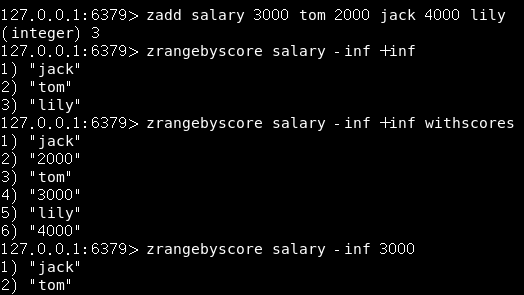
Zset是有序集合，在set集合的基础之上添加了一个score，这个score就是用来排序用的。**应用：工资、成绩排序等**。

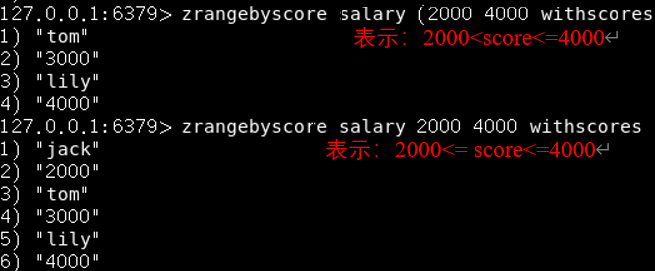
**往有序集合中添加一个或多个元素**：**zadd key score1 val1 score2 val2……**。

**获取有序集中指定区间内的元素**：**zrange key start end**，其中元素是按分数**从小到大来排序的**。若需要**从大到小排序**，则使用**zrevrange key start end**。

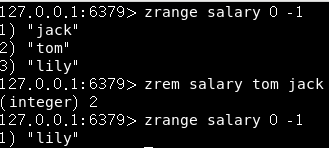


**返回有序列表中指定分数区间的元素**：**zrangebyscore key min max [withscores]**。**-inf表示负无穷大，+inf表示正无穷大**。**zrangebyscore是按从 小到大排序**。默认情况下，区间的取值使用闭区间，即小于等于或大于等于，若要是使用开区间可以通过在参数前增加左半括号表示大于或小于。**若要从大到小排序，则使用zrevrangebyscore命令**。

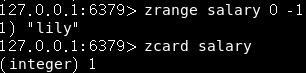




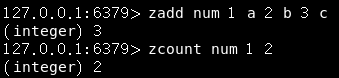
**移除有序集合中指定的元素**：**zrem key value1 value2……**。返回值为成功被移除的元素个数。



**获取有序集合中元素的个数**：**zcard key**。返回值为元素的个数。



**获取有序集合中指定分数区间的元素个数**：**zcount key min max**。返回值为区间内的元素个数。

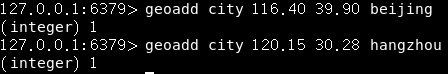


## 五、三种特殊数据类型

### 1.geospatial(地理空间)

**geo底层的实现原理是Zset，因此可以使用zset的命令来操作geo。**

**将指定的地理空间一个或多个添加到指定的key中**：**geoadd key longitude latitude value [longitude latitude vale……]**。longitude是表示经度，latitude表示纬度。有效的经度是-180度到+180度之间，有效的纬度是-85.05度到+85.05度之间。



**获取key中指定地理位置的经度和纬度**：**geopos key val1 val2……**。



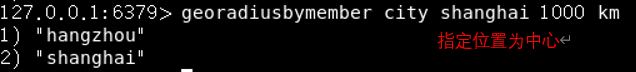
**返回指定key中两个指定地理位置的距离**：**geodist key val1 va2 [unit]**。unit表示单位，若不指定则是m。若其中一个地理位置不存在则返回空值。m(米)、km(千米)、mi(英里)、ft(英尺)。



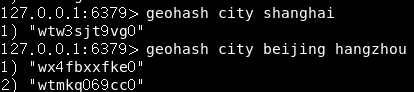
**以给定经纬度为中心，查找半径为指定范围内的所有地址位置信息**：**georadius key longitude latitude radius unit [withcoord] [withdist] [count]**。注意这个地理位置必须是实现存放在指定key中才能查找出来。withdist表示同时显示出到中心位置的距离，withcoord表示同时显示查找的地址的位置经纬度。count表示显示几条数据。

**以指定位置为中心，找出指定半径返回内的地理位置**：**georadiusbymember key member radius unit [withdist……]**。这个命令和georadius命令基本一样，只不过这命令是以指定位置为中心，而georadius则是以指定经纬度为中心。

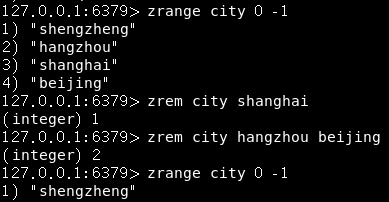




geohash命令：将返回11个字符的geohash字符串。将二维的经纬度转换成字符串。很少使用。



因为geo的底层原理是zset，因此要删除地理空间信息，可以直接使用zset的方法。



### 2.Hyperloglogs

Redis HyperLogLog 是用来做基数统计的算法。HyperLogLog 的优点是，计算基数所需的内存总是固定的、并且是很小的，比如计算264个不同元素的基数所占的内存只需12KB。

如果运行容错，则可以使用HypeLogLog，若不允许容错则使用set等其他类型。

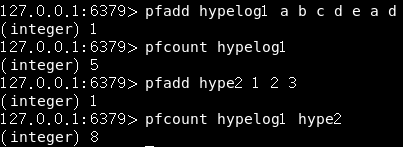
**基数**：比如数据集{1，2，5，8，5}，那么这个数据集的基数集就是{1，2，5，8}，基数(不重复的个数)则为4个。

传统的方式计算基数，set集合中保存了用户的id，可以统计set中元素的数量作为判断标准。这种方式若保存了大量用户的id，就会比较麻烦。我们的目的是计数而不是保存用户id。

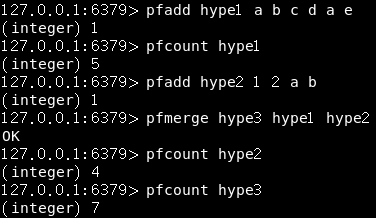
**基本命令**：**pfadd**、**pfcount**、**pfmerge**。

**将数据添加到HyperLoglog数据结构中**：**pfadd key val1 val2……**。若至少有一个元素被添加，则返回1，否则返回0。

**获得指定HyperLoglog的基数估算值**：**pfcount key1 key2……**。返回给定 HyperLogLog 的基数值，若多个 HyperLogLog 则返回基数估值之和。



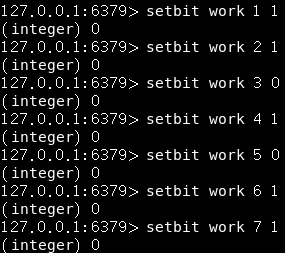
**将多个HypeLogLog合并为一个指定的HypeLoglog**：**pfmerge destkey srckey1 srckey2……**。



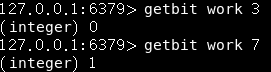
### 3.Bitmaps

Bitmaps位图，一种数据结构。都是操作二进制位来进行记录，就只有0和1两个状态。

**往Bitmaps数据结果中存数据：setbit key offset value**。使用例子：统计周一到周日上班情况，offset 1-7表示周一到周日，value值0表示休息，值1表示上班。



**获得Bitmaps中存放的数据的值**：**getbit key offset**。返回对应的值。



**统计Bitmaps中存放的数据值为1的数量**：**bitcount key [start end]**。start和end表示起始和结束的字节数。



## 六、事务

**Redis单条命令执行是具有原子性的，但是Redis中的事务是不保证原子性的，Redis中的事务也是没有隔离级别的概念**，因此就是没有幻读、脏读、不可重复读的问题。

**Redis中事务的本质**：一组命令的集合。**Redis中，一个事务的所有命令都会被按照顺序执行，若中间某条命令操作失败不会导致前面执行成功的命令回滚，也不会造成后续的命令不做**。

所有的命令在事务中，并没有直接被执行。只有发起执行命令的时候才会被执行。

**Redis的事务**：

**①开启事务**(**multi**命令)

**②命令入队**(……)

**③执行事务**(**exec**命令)

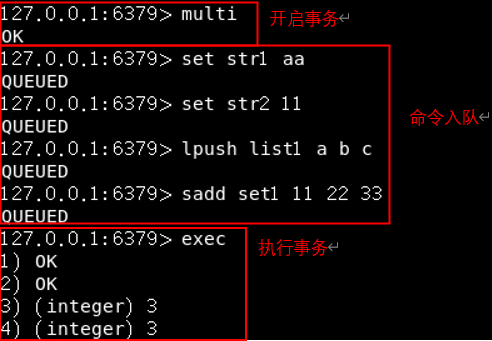
**放弃事务**：**discard**命令。放弃事务之后，事务中的所有命令都不会被执行。

每一个事务执行完之后就结束了，下一个事务需要再次开启事务。

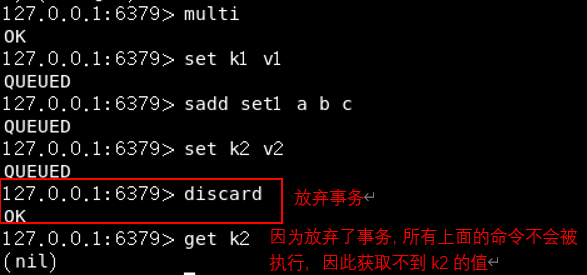
**Redis中的编译报错(代码有问题，命令写错了)**，则事务中所有的命令都不会被执行。

**Redis中的运行报错**：如果事务队列中存在语法性，那么执行事务时，其他命令可以正常执行，错误命令会报异常。

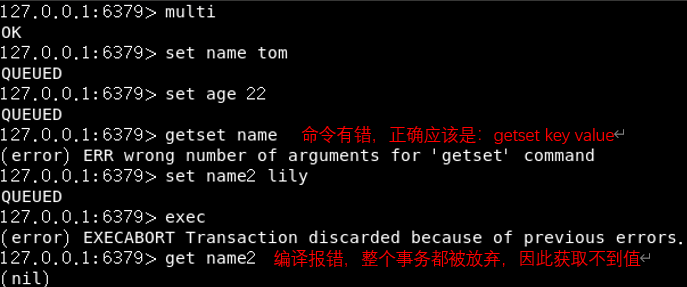
**完整事务例子**：



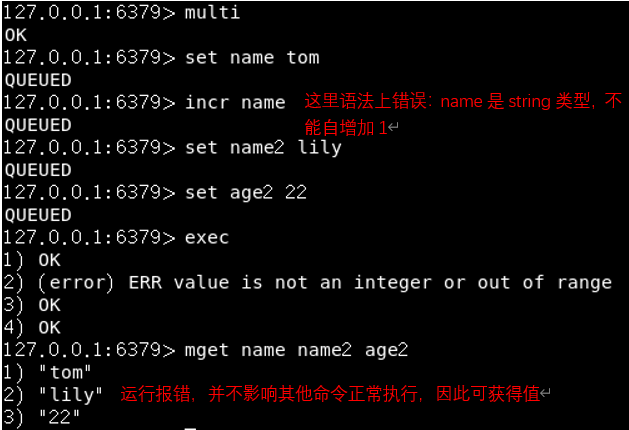
**放弃事务例子**：



**编译报错，执行事务的例子**：



**运行报错，执行事务的例子**：



## 七、Redis实现乐观锁

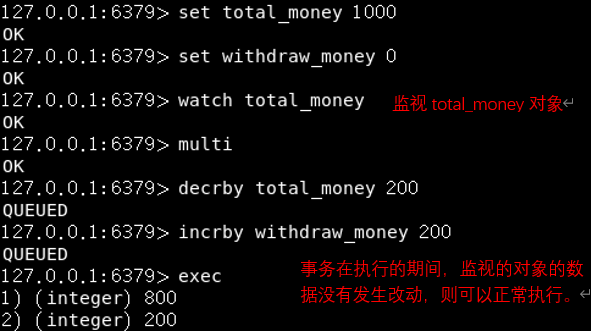
悲观锁：认为什么时候都会出现问题，所以无论做什么都会加锁。

乐观锁：认为什么时候都不会出现问题，所以不会上锁。更新数据的时候判断一下，在此期间是否有人修改过这个数据。

**实现乐观锁**：先获取监视的对象，更新的时候比较这个监视的对象的值是否修改。

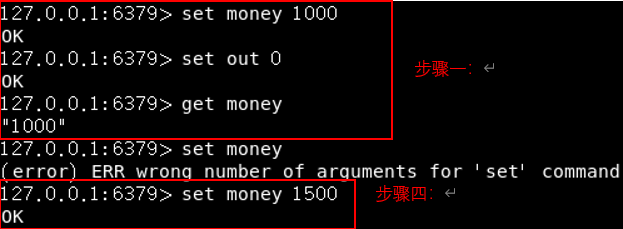
Redis中实现乐观锁：使用**watch**监视对象。

**正常执行的过程**：



测试多线程修改值，测试：按照如下步骤执行，会发现操作失败。

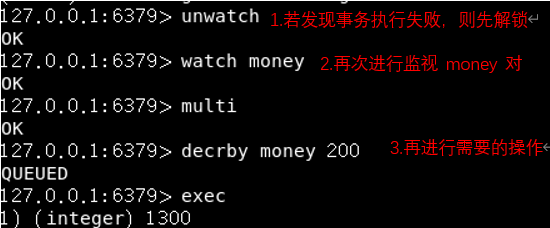
**线程一**：



**线程二**：



**若修改数据失败，解决办法**：



## 八、Jedis

Jedis是Redis官方推荐的java连接开发工具，是使用java操作Redis的中间件。

Jedis：采用的是直连的方式，多个线程操作的化，是不安全的。若要想避免不安全，则使用Jedis pool连接池。更像BIO模式。

Lettuce：采用netty，实例可以在多个线程中进行共享，不存在线程不安全的情况。更像NIO模式。

**使用Jedis的步骤**：

**⑴导入依赖**：导入Jedis的依赖。下面的例子中使用到了fastjson，因此也导入了对应的依赖。



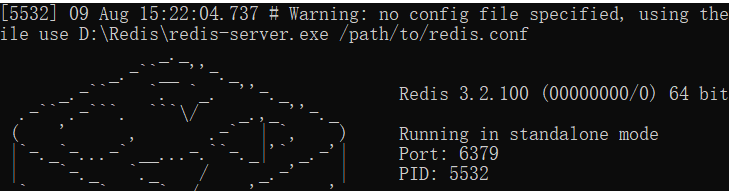
**⑵进行测试**：

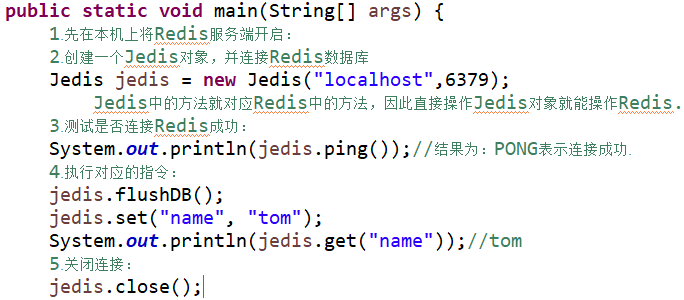
①启动Redis服务端；

②创建Jedis对象并连接Redis服务端；

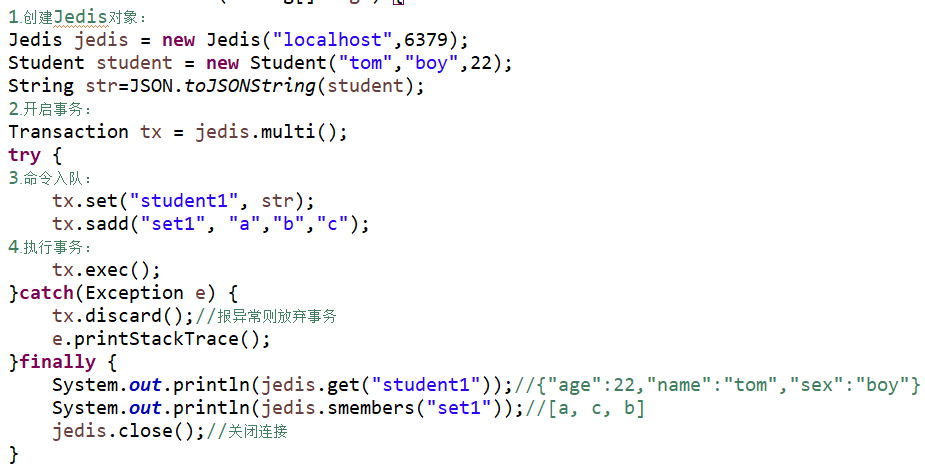
③根据Jedis对象操作对应的方法；

④断开连接。





**Jedis中操作Redis事务的例子**：



## 九、SpringBoot整合Redis

学完SpringBoot在整理这个。

## 十、redis.conf配置文件

在Linux中启动Redis服务端时，需要通过这个配置文件来启动。

### 1.单位：

配置文件中unit单位对大小写不敏感。



### 2.包含其他配置文件：

可以引用其他配置文件。



### 3.网络的配置：

**绑定ip：**127.0.0.1的ip表示该Redis只能在本机上访问。若要在远程访问，需要指定自己的ip地址或者使用通配符\*。



**保护模式**：这个一般要是开启状态。



**端口设置**：

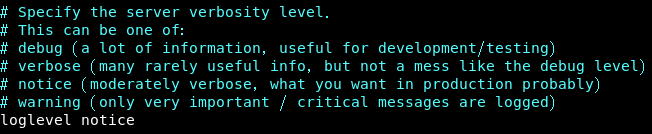


### 4.通用设置：

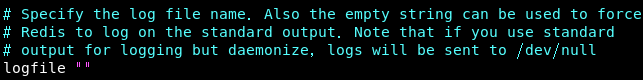
是否以守护进程的方式运行，默认是no，需要自己手动开启为yes。否则的话一退出，那么这个进程就结束。



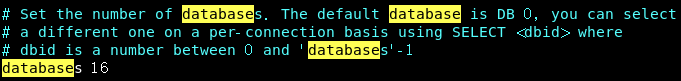
**日志级别**：



logfile用来指定日志的文件位置名。若为空则表示直接输出。



**数据库的数量**：默认是16个数据库。



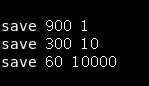
### 5.快照(rdb配置)

这里一般进行rdb的配置。

持久化，在规定的时间内，执行了多少次操作，则会持久化到.rdb或.aof文件中。

**Redis是内存数据库，若没有持久化，则数据就会断电即失**。

**900表示**：若900秒内至少有一个key进行了修改，则进行持久化操作。**300表示**：300秒内至少有10个key进行了修改，则进行持久化操作。**60表示**：60秒内至少有10000个key进行了修改，则进行持久化操作。



持久化之后若出错是否继续工作，默认是开启的。



是否压缩rdb文件，需要消耗cpu资源，默认开启的。



保存rdb文件的时候，进行错误的检查校验。



rdb文件保存的目录，默认是当前目录下。



rdb文件的保存名称。



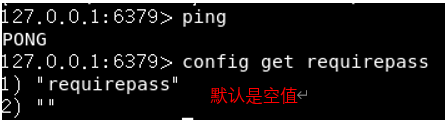
### 6.replication

主从复制的配置。



### 7.security

可设置Redis的密码，redis默认没有密码。



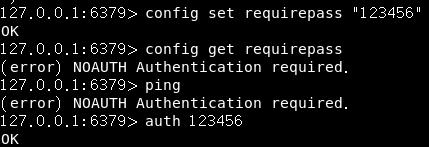
**设置密码有两种方式**：

**法一**：直接在配置文件中设置。



**法二**：使用**config set requirepass password**命令设置密码。

设置完密码之后，使用**auth password**命令登录。



### 8.client设置

设置能连接上redis的最大客户端的数量，默认是10000。



redis配置最大的内存容量。



内存到达上限之后的处理策略。有空网上再了解。



### 9.append only(aof)配置

这里一般进行aof的配置。

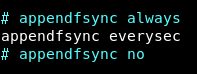
默认是不开启aof模式的，默认使用rdb方式持久化的。在大部分情况下，rdb完全够用。



持久化的文件名字。



**always表示**：每次修改了数据都会同步，消耗性能。**everysec表示**：每秒执行一次同步，可能会丢失这条数据。**no表示**：不执行同步，这个时候操作系统自己同步数据，速度最快。



## 十一、Redis持久化

Redis是内存数据库，若不将内存中的数据库状态保存到磁盘中，那么一旦服务器进程退出，服务器中的数据库状态也会消失。故Redis提供了持久化功能。

### 1.RDB(Redis DataBase)

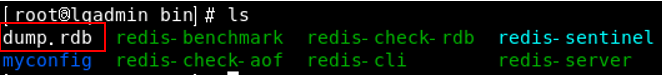
**在指定的时间间隔内将内存中的数据集快照写入磁盘中，也就是snapshot快照，它恢复时是将快照文件直接读到内存中的**。

在主从复制中，rdb就是备用的。

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。若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方式更加高效。

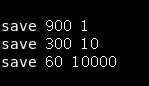
**RDB的缺点是最后一次持久化后的数据可能丢失**。默认使用RDB，一般情况下不需要修改这个配置。

**rdb保存的文件是dump.rdb**。**有时候会对这个dump.rdb文件进行备份**。



**产生rdb文件的情况**：

**①**save的规则满足的情况下，会触发rdb规则，即产生rdb文件。

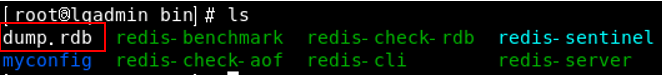


**②**执行flushall命令，也会产生rdb文件。

**③**退出redis时，也会产生rdb文件。

**恢复rdb文件**：只需要将rdb文件放在redis的启动目录下，redis启动时就会自动检查dump.rdb恢复其中的数据，即放在和redis-server同级的目录中。





rdb的优点：适合大规模的数据恢复；对数据的完整性要求不高。

rdb的缺点：需要一定的时间间隔进行操作。若reids意外宕机，则最后一次修改数据就没有了。fork进程的时候，会占用一定的内容空间。

### 2.AOF(Append Only File)

**将我们的所有的写命令都记录在日志文件中，恢复的时候就把这个文件的内容全部执行一遍**。

以日志的形式记录每个写操作，将Redis执行过程中的所有指令记录下来(读操作不记录)，只许追加文件但不可改写文件，redis启动之初会读取该文件重新构建数据。简单的说，Redis重启的话就根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作。

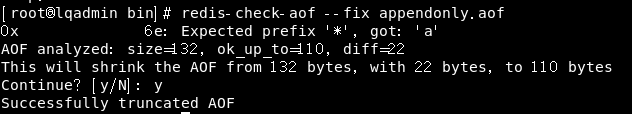
AOF保存的是**appendonly.aof**文件。默认是不开启的，需要手动进行配置。只需要将appendonly改为yes就可以开启aof了。



开启之后，重启redis就可以查看到aof文件。



如果aof文件有错误，则redis客户端是启动不起来的，需要修复这个aof文件。使用**redis-check-aof –-fix** appendonly.aof这个命令。



**aof的优点**：若每一次修改都同步，则文件的完整性会更好；若每秒同步一次，可能会丢失一秒的数据；从不同步，效率最高。

**aof的缺点**：相对于数据文件来说，aof远远大于rdb，修复的速度也比rdb慢；aof运行效率也比rdb慢。

### 3.扩展

rdb持久化方式能够在执行的时间间隔对数据进行快照存储。

aof持久化方式记录每次对服务器的写操作，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。

当只做缓存时，若只希望数据在服务器运行的时候存在，可以不使用任何持久化。

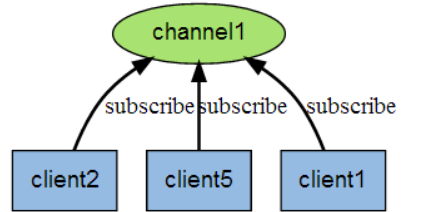
**在同时开启两种持久化方式时**，redis重启时会优先载入aof文件来恢复数据，因为通常情况下aof文件保存的数据要比rdb保存的数据更完整。

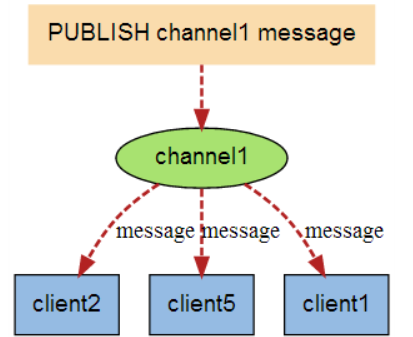
## 十二、发布订阅

Redis发布订阅(pub/sub)是**一种消息通信模式**，发送者(pub)发送消息，接收者(sub)接受消息。Redis客户端可以订阅任意数量的频道。

**使用场景**：实时消息系统；订阅、关注系统；实时聊天等。**对于稍微复杂的场景，会使用消息中间件MQ**。

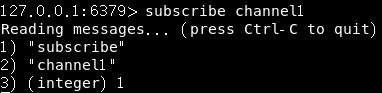
下面是三个客户端订阅了频道channel1。当有新消息通过publish命令发送给频道channel1时，这个消息同时也会被发送给订阅它的三个客户端。





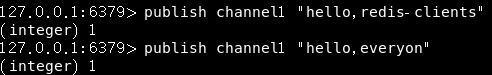
常用命令：

**订阅指定的一个或多个频道信息**：**subscribe channel1 [channel2……]**。

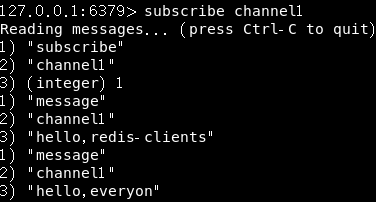


**将信息发送到指定的频道中**：**publish channel message**。之后只要订阅端没取消订阅都能接收到消息。

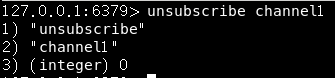
发送端：



接收端：



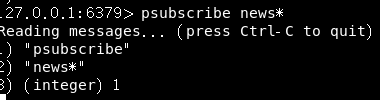
**取消订阅指定的一个或多个频道的信息**：**unsubscribe channel[channel2……]**。



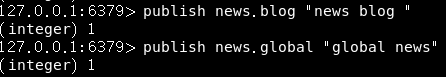
pubsub命令用于查看订阅与发布系统状态。比如**查看发布或订阅的频道列表**：**pubsub channels**。



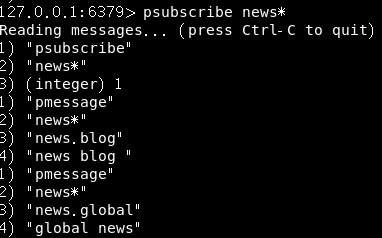
**订阅一个或多个指定模式的频道**：**psubscribe pattern[pattern……]**。每个模式以\*作为通配符，比如news\*表示匹配所有以news开头的频道，如news.blog,newsglobal等。



发送者：



接收者：



**退订所有指定模式的频道**：**punsubscribe pattern [pattern……]**。

## 十三、主从复制

### 1.概念

**主从复制**：是指将一台Redis服务器的数据复制到其他的Redis服务器中。前者称为主节点(master/leader)，后者称为从节点(slave/follower)。数据的复制是单向的，只能从主节点到从节点。

**主从复制主要是用来处理读写分离，主节点用来处理写请求，从节点用来处理读请求**。

默认情况下，每台Redis服务器都是主节点，且一个主节点可以有多个从节点(或没有从节点)。一个从节点只能有一个主节点。

**主从复制的作用**：

**①数据冗余**：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。

**②故障恢复**：当主节点出现问题，可以由从节点提供服务，从而实现快速的故障恢复。

**③负载均衡**：在主从复制的基础上，配合读写分离，由主节点提供写服务，从节点提供读服务，从而分担服务器的负载，从而提高Redis服务器的并发量。

一般要将Redis运用于工程项目中，只使用一条Redis是不够的，原因如下：

①从结构上，单个Redis服务器可能会发生单点故障，并且一台服务器需要处理所有的请求负载，压力较大。

②从容量上，单个Redis服务器内存容量有限。一般单台Redis最大使用内存不应该超过20G。

### 2.集群配置

只配置从库，不用配置主库，因为默认是主库。**info replication**命令是查看当前库的信息。



**步骤一**：复制三个redis.conf文件，修改对应的信息：端口、pid名字、log文件名、dump.rdb名称。



比如：

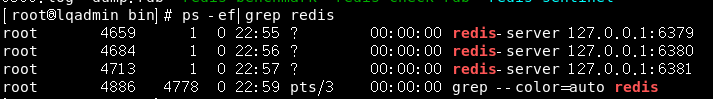






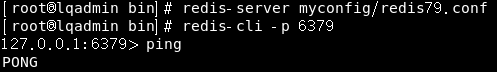


**步骤二**：修改完之后，根据相应的redis配置文件启动三个redis服务器。启动成功之后可以通过进行信息查看。

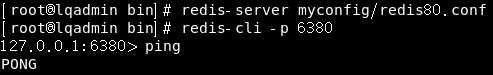


**步骤三**：一主二从，默认都是主节点。选两个作为从节点进行配置。使用**slaveof host port**指定从节点。host表示哪个主机，port表示作为主节点的Redis使用的端口号。

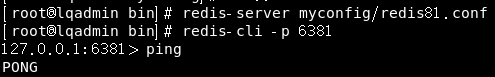
启动界面1：



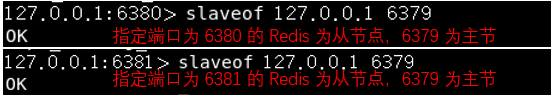
启动界面2：



启动界面3：



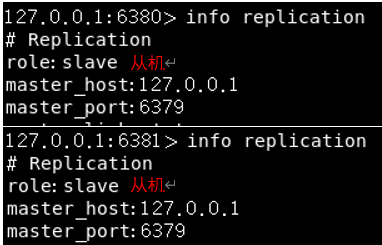
设置从节点：



之后查看主节点的信息：

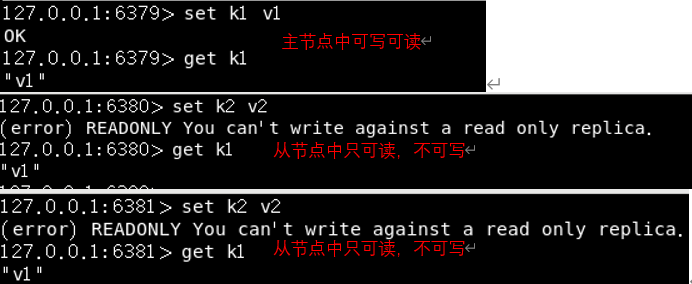


查看从节点的信息：



真实的主从配置应该是在redis.conf配置文件中的replication中进行配置，这样的话才是永久的。上面的例子使用的是slaveof 命令方式，是暂时的。

**主从复制中，主节点可写可读，而从节点只能读不能写。主节点中保存的数据，都会自动被从节点保存**。



主节点断开连接之后，从节点依旧连接到主节点，但是没有写操作。若主节点又启动了，则从节点依旧可以直接获取到主机写的信息。

若是使用命令行slaveof配置的从节点，当从节点关闭之后又重新启动，则会恢复到主节点状态，因为使用slaveof配置的从节点是暂时的。

如果主节点断开了连接，可以使用slaveof no one让从节点变为主节点。其他的节点就可以使用slaveof host port手动连接到这个最新的主节点。

slave启动成功连接到master后会发送一个sync同步命令。master接到命令，启动后台的存盘进程，同时收集所有接收到的用于修改数据集命令，在后台进程执行完毕之后，master将传送整个数据文件到slave，并完成一次完全同步。

**全量复制**：slave服务在接收到数据库文件数据后，将其存盘并加载到内存中。

**增量复制**：master继续将新的所有收集到的修改命令依次传给slave，完成同步。

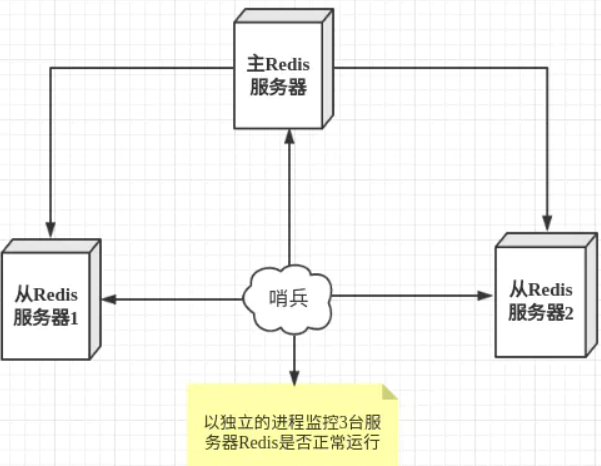
**但只要是从新连接master，一次完全同步(全量复制)将被自动执行**。

### 3.哨兵模式

**主从切换技术的方法**是：当主服务器宕机后，需要手动把一台从服务器切换为主服务器(使用slave no one命令)，这需要人工干预，费时费力，还会造成一段时间内服务不可用。不推荐使用这种方式，推荐使用哨兵模式。

在哨兵模式中，它能够后台监控主机是否故障，若故障了则会自动将从节点转换为主节点。

哨兵模式是一种特殊的模式，首先Redis提供了哨兵的命令，哨兵是一个独立的进程，作为进程它可独立运行。**原理**是**哨兵通过发送命令，等待Redis服务器响应，从而监控运行的多个Redis实例**。

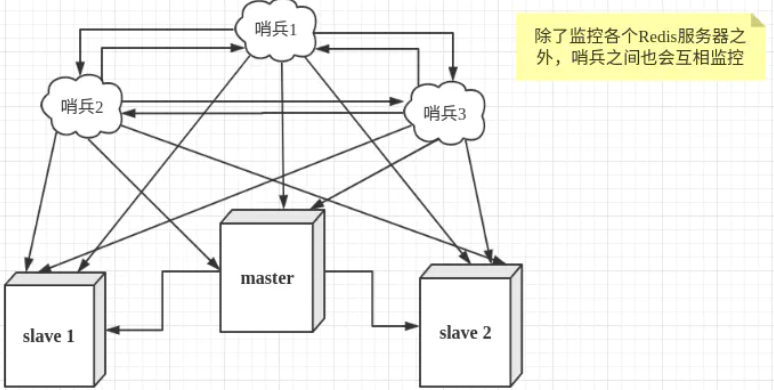


**哨兵的作用**：

通过发送命令，判断Redis服务器(主服务器和从服务器)的运行状态。

当哨兵监测到master宕机，会自动将slave切换成master，然后通过发布订阅模式通知其他的从服务器，修改配置文件，让它们切换主机。

一个哨兵进程对Redis服务器进行监控，可能会出现问题，因此可使用多个哨兵进行监控。各个哨兵之间还会进行监控，这样就形成了多哨兵模式。



**描述一下故障切换（failover）的过程**：假设主服务器宕机，哨兵1先检测到这个结果，系统并不会马上进行failover过程，仅仅是哨兵1主观的认为主服务器不可用，这个现象成为主观下线。当后面的哨兵也检测到主服务器不可用，并且数量达到一定值时，那么哨兵之间就会进行一次投票，投票的结果由一个哨兵发起，进行failover操作。切换成功后，就会通过发布订阅模式，让各个哨兵把自己监控的从服务器实现切换主机，这个过程称为客观下线。这样对于客户端而言，一切都是透明的。

**哨兵模式完整操作步骤**：

**步骤一**：**主从服务器配置**。也可以用slaveof host port命令。

修改主服务器的redis.conf配置文件。



修改从服务器redis.conf的配置文件。

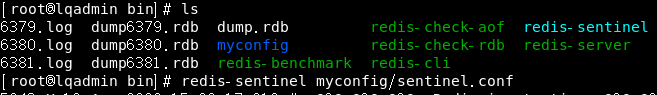


**步骤二**：**配置哨兵**。在Redis的安装目录下有一个sentinel.conf，将这个文件复制到/usr/local/bin目录下的myconfig目录下。对这个sentinel.conf文件进行修改。有多少个哨兵就要有多少个sentinel.conf配置文件。



**步骤三**：**依次启动主服务器、从服务器、哨兵的服务进程**。

启动哨兵的命令：使用redis-sentinel命令指定哨兵配置文件启动哨兵。如redis-sentinel myconfig/sentinel.conf。



之后，只要将主服务器宕机，那么过一会哨兵会将其中的一个从服务器设置为主服务器，然后其他的从服务器就会自动连接上这个主服务器。

**哨兵模式的优点**：

基于主从复制，所有的主从配置的优点它都有。

主从可以切换，故障可以转移，系统的可用性更好。

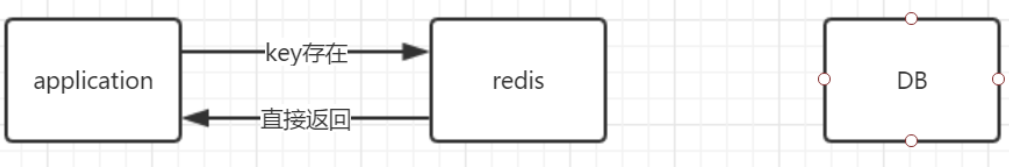
哨兵模式是主从复制的升级，主服务器宕机，从服务器自动转换为主服务器。

## 十四、Redis缓存穿透和雪崩

### 1.缓存穿透

**当使用Redis做缓存时，数据访问过程如下**：

用户先访问Redis，若Redis中存放了该数据，则直接将数据返回给用户。



若Redis中没有存放该数据，则用户直接访问数据库，然后数据库将查询的数据直接返回给用户，同时将该数据存入Redis中。



当用户查询的数据不在Redis中，也不在持久层数据库中，用户发送大量查询该数据的请求，则会对持久层数据库造成很大的压力，这样的话Redis缓存的作用就失效了，这就相当于出现了**缓存穿透**。

**解决缓存穿透的办法**

**法一**：**使用布隆过滤器**。对所有可能查询的参数以hash形式存储，在控制层先进行校验，不符合则丢弃，从而避免了对底层存储系统的查询压力。

**法二**：**缓存空对象**。当持久层数据库中没有查询要求的数据，则返回一个空对象，并存放到Redis中。之后再访问这个数据就会从缓存中获取，从而保护了持久层数据库。

**缓存空对象有弊端**：

缓存太多的空值，需要占用更多的空间。优化：给空值设置过期时间。

即使给空值设置了过期时间，当存储层的数据更新了，缓存层和存储层的数据会有一段时间内是不一致的，这对于需要保持一致性的业务会有影响。

### 2.缓存击穿

对于一些设置了过期时间的key，若某个key是热点，在某段时间内被超高并发访问，当这个key在失效的瞬间，持续的高并发就穿破缓存，直接访问数据库，数据库要进行查询数据操作，同时还需要将数据写入缓存中，这个瞬间会给数据库造成巨大压力，这就是**缓存击穿**。

**解决办法**：设置热点数据永不过期；使用分布式锁，保证每个key同时只有一个线程去查询后端服务。

### 3.缓存雪崩

**缓存雪崩**是指在缓存时对大量的key采用了相同的过期时间，导致这些key在某一时刻同时失效，导致大量并发的查询操作落在了数据库上。或者Redis服务器宕机了。

**解决办法**：使用多态Redis，给Redis搭建集群；在缓存失败后，通过加锁或者队列来控制数据库写缓存的线程数量；在即将发生大并发访问前手动触发加载不同的key，设置不同的过期时间。

简要理解：

**缓存穿透就是在缓存中查询不到数据，而直接查询数据库。**

**缓存击穿就是查询数据的量太大，同时缓存中某个key过期。**

**缓存雪崩就是缓存中大量的key同时过期，大量的查询操作直接落在数据库上**。