# 超强、超详细Redis数据库入门教程

这篇文章主要介绍了超强、超详细Redis入门教程,本文详细介绍了Redis数据库各个方面的知识,需要的朋友可以参考下

【本教程目录】

1.redis是什么  
2.redis的作者何许人也  
3.谁在使用redis  
4.学会安装redis  
5.学会启动redis  
6.使用redis客户端  
7.redis数据结构 – 简介  
8.redis数据结构 – strings  
9.redis数据结构 – lists  
10.redis数据结构 – 集合  
11.redis数据结构 – 有序集合  
12.redis数据结构 – 哈希  
13.聊聊redis持久化 – 两种方式  
14.聊聊redis持久化 – RDB  
15.聊聊redis持久化 – AOF  
16.聊聊redis持久化 – AOF重写  
17.聊聊redis持久化 – 如何选择RDB和AOF  
18.聊聊主从 – 用法  
19.聊聊主从 – 同步原理  
20.聊聊redis的事务处理  
21.教你看懂redis配置 – 简介  
22.教你看懂redis配置 -通用  
23.教你看懂redis配置 – 快照  
24.教你看懂redis配置 – 复制  
25.教你看懂redis配置 – 安全  
26.教你看懂redis配置 -限制  
27.教你看懂redis配置 – 追加模式  
28.教你看懂redis配置 – LUA脚本  
29.教你看懂redis配置 – 慢日志  
30.教你看懂redis配置 – 事件通知  
31.教你看懂redis配置 – 高级配置

【redis是什么】

redis是一个开源的、使用C语言编写的、支持网络交互的、可基于内存也可持久化的Key-Value数据库。

redis的官网地址，非常好记，是redis.io。（特意查了一下，域名后缀io属于国家域名，是british Indian Ocean territory，即英属印度洋领地）

目前，Vmware在资助着redis项目的开发和维护。

【redis的作者何许人也】

开门见山，先看照片：



是不是出乎了你的意料，嗯，高手总会有些地方与众不同的。

这位便是redis的作者，他叫Salvatore Sanfilippo，来自意大利的西西里岛，现在居住在卡塔尼亚。目前供职于Pivotal公司。

他使用的网名是antirez，如果你有兴趣，可以去他的博客逛逛，地址是antirez.com，当然也可以去follow他的github，地址是<http://github.com/antirez>。

【谁在使用redis】

Blizzard、digg、stackoverflow、github、flickr …

【学会安装redis】

从redis.io下载最新版redis-X.Y.Z.tar.gz后解压，然后进入redis-X.Y.Z文件夹后直接make即可，安装非常简单。

make成功后会在src文件夹下产生一些二进制可执行文件，包括redis-server、redis-cli等等：

复制代码 代码如下:

$ find . -type f -executable  
./redis-benchmark //用于进行redis性能测试的工具  
./redis-check-dump //用于修复出问题的dump.rdb文件  
./redis-cli //redis的客户端  
./redis-server //redis的服务端  
./redis-check-aof //用于修复出问题的AOF文件  
./redis-sentinel //用于集群管理

【学会启动redis】

启动redis非常简单，直接./redis-server就可以启动服务端了，还可以用下面的方法指定要加载的配置文件：

复制代码 代码如下:

./redis-server ../redis.conf

默认情况下，redis-server会以非daemon的方式来运行，且默认服务端口为6379。

有关作者为什么选择6379作为默认端口，还有一段有趣的典故，英语好的同学可以看看作者这篇博文中的解释。

【使用redis客户端】

我们直接看一个例子：

复制代码 代码如下:

//这样来启动redis客户端了  
$ ./redis-cli  
//用set指令来设置key、value  
127.0.0.1:6379> set name "roc"   
OK  
//来获取name的值  
127.0.0.1:6379> get name   
"roc"  
//通过客户端来关闭redis服务端  
127.0.0.1:6379> shutdown   
127.0.0.1:6379>

【redis数据结构 – 简介】

redis是一种高级的key:value存储系统，其中value支持五种数据类型：

1.字符串（strings）  
2.字符串列表（lists）  
3.字符串集合（sets）  
4.有序字符串集合（sorted sets）  
5.哈希（hashes）

而关于key，有几个点要提醒大家：

1.key不要太长，尽量不要超过1024字节，这不仅消耗内存，而且会降低查找的效率；  
2.key也不要太短，太短的话，key的可读性会降低；  
3.在一个项目中，key最好使用统一的命名模式，例如user:10000:passwd。

【redis数据结构 – strings】

有人说，如果只使用redis中的字符串类型，且不使用redis的持久化功能，那么，redis就和memcache非常非常的像了。这说明strings类型是一个很基础的数据类型，也是任何存储系统都必备的数据类型。

我们来看一个最简单的例子：

复制代码 代码如下:

set mystr "hello world!" //设置字符串类型  
get mystr //读取字符串类型

字符串类型的用法就是这么简单，因为是二进制安全的，所以你完全可以把一个图片文件的内容作为字符串来存储。

另外，我们还可以通过字符串类型进行数值操作：

复制代码 代码如下:

127.0.0.1:6379> set mynum "2"  
OK  
127.0.0.1:6379> get mynum  
"2"  
127.0.0.1:6379> incr mynum  
(integer) 3  
127.0.0.1:6379> get mynum  
"3"

看，在遇到数值操作时，redis会将字符串类型转换成数值。

由于INCR等指令本身就具有原子操作的特性，所以我们完全可以利用redis的INCR、INCRBY、DECR、DECRBY等指令来实现原子计数的效果，假如，在某种场景下有3个客户端同时读取了mynum的值（值为2），然后对其同时进行了加1的操作，那么，最后mynum的值一定是5。不少网站都利用redis的这个特性来实现业务上的统计计数需求。

【redis数据结构 – lists】

redis的另一个重要的数据结构叫做lists，翻译成中文叫做“列表”。

首先要明确一点，redis中的lists在底层实现上并不是数组，而是链表，也就是说对于一个具有上百万个元素的lists来说，在头部和尾部插入一个新元素，其时间复杂度是常数级别的，比如用LPUSH在10个元素的lists头部插入新元素，和在上千万元素的lists头部插入新元素的速度应该是相同的。

虽然lists有这样的优势，但同样有其弊端，那就是，链表型lists的元素定位会比较慢，而数组型lists的元素定位就会快得多。

lists的常用操作包括LPUSH、RPUSH、LRANGE等。我们可以用LPUSH在lists的左侧插入一个新元素，用RPUSH在lists的右侧插入一个新元素，用LRANGE命令从lists中指定一个范围来提取元素。我们来看几个例子：

复制代码 代码如下:

//新建一个list叫做mylist，并在列表头部插入元素"1"  
127.0.0.1:6379> lpush mylist "1"   
//返回当前mylist中的元素个数  
(integer) 1   
//在mylist右侧插入元素"2"  
127.0.0.1:6379> rpush mylist "2"   
(integer) 2  
//在mylist左侧插入元素"0"  
127.0.0.1:6379> lpush mylist "0"   
(integer) 3  
//列出mylist中从编号0到编号1的元素  
127.0.0.1:6379> lrange mylist 0 1   
1) "0"  
2) "1"  
//列出mylist中从编号0到倒数第一个元素  
127.0.0.1:6379> lrange mylist 0 -1   
1) "0"  
2) "1"  
3) "2"

lists的应用相当广泛，随便举几个例子：

1.我们可以利用lists来实现一个消息队列，而且可以确保先后顺序，不必像MySQL那样还需要通过ORDER BY来进行排序。  
2.利用LRANGE还可以很方便的实现分页的功能。  
3.在博客系统中，每片博文的评论也可以存入一个单独的list中。

【redis数据结构 – 集合】

redis的集合，是一种无序的集合，集合中的元素没有先后顺序。

集合相关的操作也很丰富，如添加新元素、删除已有元素、取交集、取并集、取差集等。我们来看例子：

复制代码 代码如下:

//向集合myset中加入一个新元素"one"  
127.0.0.1:6379> sadd myset "one"   
(integer) 1  
127.0.0.1:6379> sadd myset "two"  
(integer) 1  
//列出集合myset中的所有元素  
127.0.0.1:6379> smembers myset   
1) "one"  
2) "two"  
//判断元素1是否在集合myset中，返回1表示存在  
127.0.0.1:6379> sismember myset "one"   
(integer) 1  
//判断元素3是否在集合myset中，返回0表示不存在  
127.0.0.1:6379> sismember myset "three"   
(integer) 0  
//新建一个新的集合yourset  
127.0.0.1:6379> sadd yourset "1"   
(integer) 1  
127.0.0.1:6379> sadd yourset "2"  
(integer) 1  
127.0.0.1:6379> smembers yourset  
1) "1"  
2) "2"  
//对两个集合求并集  
127.0.0.1:6379> sunion myset yourset   
1) "1"  
2) "one"  
3) "2"  
4) "two"

对于集合的使用，也有一些常见的方式，比如，QQ有一个社交功能叫做“好友标签”，大家可以给你的好友贴标签，比如“大美女”、“土豪”、“欧巴”等等，这时就可以使用redis的集合来实现，把每一个用户的标签都存储在一个集合之中。

【redis数据结构 – 有序集合】

redis不但提供了无需集合（sets），还很体贴的提供了有序集合（sorted sets）。有序集合中的每个元素都关联一个序号（score），这便是排序的依据。

很多时候，我们都将redis中的有序集合叫做zsets，这是因为在redis中，有序集合相关的操作指令都是以z开头的，比如zrange、zadd、zrevrange、zrangebyscore等等

老规矩，我们来看几个生动的例子：  
//新增一个有序集合myzset，并加入一个元素baidu.com，给它赋予的序号是1：

复制代码 代码如下:

127.0.0.1:6379> zadd myzset 1 baidu.com   
(integer) 1  
//向myzset中新增一个元素360.com，赋予它的序号是3  
127.0.0.1:6379> zadd myzset 3 360.com   
(integer) 1  
//向myzset中新增一个元素google.com，赋予它的序号是2  
127.0.0.1:6379> zadd myzset 2 google.com   
(integer) 1  
//列出myzset的所有元素，同时列出其序号，可以看出myzset已经是有序的了。  
127.0.0.1:6379> zrange myzset 0 -1 with scores   
1) "baidu.com"  
2) "1"  
3) "google.com"  
4) "2"  
5) "360.com"  
6) "3"  
//只列出myzset的元素  
127.0.0.1:6379> zrange myzset 0 -1   
1) "baidu.com"  
2) "google.com"  
3) "360.com"

【redis数据结构 – 哈希】

最后要给大家介绍的是hashes，即哈希。哈希是从redis-2.0.0版本之后才有的数据结构。

hashes存的是字符串和字符串值之间的映射，比如一个用户要存储其全名、姓氏、年龄等等，就很适合使用哈希。

我们来看一个例子：

复制代码 代码如下:

//建立哈希，并赋值  
127.0.0.1:6379> HMSET user:001 username antirez password P1pp0 age 34   
OK  
//列出哈希的内容  
127.0.0.1:6379> HGETALL user:001   
1) "username"  
2) "antirez"  
3) "password"  
4) "P1pp0"  
5) "age"  
6) "34"  
//更改哈希中的某一个值  
127.0.0.1:6379> HSET user:001 password 12345   
(integer) 0  
//再次列出哈希的内容  
127.0.0.1:6379> HGETALL user:001   
1) "username"  
2) "antirez"  
3) "password"  
4) "12345"  
5) "age"  
6) "34"

有关hashes的操作，同样很丰富，需要时，大家可以从这里[查询](http://redis.io/commands" \l "hash)。

【聊聊redis持久化 – 两种方式】

redis提供了两种持久化的方式，分别是RDB（Redis DataBase）和AOF（Append Only File）。

RDB，简而言之，就是在不同的时间点，将redis存储的数据生成快照并存储到磁盘等介质上；

AOF，则是换了一个角度来实现持久化，那就是将redis执行过的所有写指令记录下来，在下次redis重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复了。

其实RDB和AOF两种方式也可以同时使用，在这种情况下，如果redis重启的话，则会优先采用AOF方式来进行数据恢复，这是因为AOF方式的数据恢复完整度更高。

如果你没有数据持久化的需求，也完全可以关闭RDB和AOF方式，这样的话，redis将变成一个纯内存数据库，就像memcache一样。

【聊聊redis持久化 – RDB】

RDB方式，是将redis某一时刻的数据持久化到磁盘中，是一种快照式的持久化方法。

redis在进行数据持久化的过程中，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，才会用这个临时文件替换上次持久化好的文件。正是这种特性，让我们可以随时来进行备份，因为快照文件总是完整可用的。

对于RDB方式，redis会单独创建（fork）一个子进程来进行持久化，而主进程是不会进行任何IO操作的，这样就确保了redis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。

虽然RDB有不少优点，但它的缺点也是不容忽视的。如果你对数据的完整性非常敏感，那么RDB方式就不太适合你，因为即使你每5分钟都持久化一次，当redis故障时，仍然会有近5分钟的数据丢失。所以，redis还提供了另一种持久化方式，那就是AOF。

【聊聊redis持久化 – AOF】

AOF，英文是Append Only File，即只允许追加不允许改写的文件。

如前面介绍的，AOF方式是将执行过的写指令记录下来，在数据恢复时按照从前到后的顺序再将指令都执行一遍，就这么简单。

我们通过配置redis.conf中的appendonly yes就可以打开AOF功能。如果有写操作（如SET等），redis就会被追加到AOF文件的末尾。

默认的AOF持久化策略是每秒钟fsync一次（fsync是指把缓存中的写指令记录到磁盘中），因为在这种情况下，redis仍然可以保持很好的处理性能，即使redis故障，也只会丢失最近1秒钟的数据。

如果在追加日志时，恰好遇到磁盘空间满、inode满或断电等情况导致日志写入不完整，也没有关系，redis提供了redis-check-aof工具，可以用来进行日志修复。

因为采用了追加方式，如果不做任何处理的话，AOF文件会变得越来越大，为此，redis提供了AOF文件重写（rewrite）机制，即当AOF文件的大小超过所设定的阈值时，redis就会启动AOF文件的内容压缩，只保留可以恢复数据的最小指令集。举个例子或许更形象，假如我们调用了100次INCR指令，在AOF文件中就要存储100条指令，但这明显是很低效的，完全可以把这100条指令合并成一条SET指令，这就是重写机制的原理。

在进行AOF重写时，仍然是采用先写临时文件，全部完成后再替换的流程，所以断电、磁盘满等问题都不会影响AOF文件的可用性，这点大家可以放心。

AOF方式的另一个好处，我们通过一个“场景再现”来说明。某同学在操作redis时，不小心执行了FLUSHALL，导致redis内存中的数据全部被清空了，这是很悲剧的事情。不过这也不是世界末日，只要redis配置了AOF持久化方式，且AOF文件还没有被重写（rewrite），我们就可以用最快的速度暂停redis并编辑AOF文件，将最后一行的FLUSHALL命令删除，然后重启redis，就可以恢复redis的所有数据到FLUSHALL之前的状态了。是不是很神奇，这就是AOF持久化方式的好处之一。但是如果AOF文件已经被重写了，那就无法通过这种方法来恢复数据了。

虽然优点多多，但AOF方式也同样存在缺陷，比如在同样数据规模的情况下，AOF文件要比RDB文件的体积大。而且，AOF方式的恢复速度也要慢于RDB方式。

如果你直接执行BGREWRITEAOF命令，那么redis会生成一个全新的AOF文件，其中便包括了可以恢复现有数据的最少的命令集。

如果运气比较差，AOF文件出现了被写坏的情况，也不必过分担忧，redis并不会贸然加载这个有问题的AOF文件，而是报错退出。这时可以通过以下步骤来修复出错的文件：

1.备份被写坏的AOF文件  
2.运行redis-check-aof –fix进行修复  
3.用diff -u来看下两个文件的差异，确认问题点  
4.重启redis，加载修复后的AOF文件

【聊聊redis持久化 – AOF重写】

AOF重写的内部运行原理，我们有必要了解一下。

在重写即将开始之际，redis会创建（fork）一个“重写子进程”，这个子进程会首先读取现有的AOF文件，并将其包含的指令进行分析压缩并写入到一个临时文件中。

与此同时，主工作进程会将新接收到的写指令一边累积到内存缓冲区中，一边继续写入到原有的AOF文件中，这样做是保证原有的AOF文件的可用性，避免在重写过程中出现意外。

当“重写子进程”完成重写工作后，它会给父进程发一个信号，父进程收到信号后就会将内存中缓存的写指令追加到新AOF文件中。

当追加结束后，redis就会用新AOF文件来代替旧AOF文件，之后再有新的写指令，就都会追加到新的AOF文件中了。

【聊聊redis持久化 – 如何选择RDB和AOF】

对于我们应该选择RDB还是AOF，官方的建议是两个同时使用。这样可以提供更可靠的持久化方案。

【聊聊主从 – 用法】

像MySQL一样，redis是支持主从同步的，而且也支持一主多从以及多级从结构。

主从结构，一是为了纯粹的冗余备份，二是为了提升读性能，比如很消耗性能的SORT就可以由从服务器来承担。

redis的主从同步是异步进行的，这意味着主从同步不会影响主逻辑，也不会降低redis的处理性能。

主从架构中，可以考虑关闭主服务器的数据持久化功能，只让从服务器进行持久化，这样可以提高主服务器的处理性能。

在主从架构中，从服务器通常被设置为只读模式，这样可以避免从服务器的数据被误修改。但是从服务器仍然可以接受CONFIG等指令，所以还是不应该将从服务器直接暴露到不安全的网络环境中。如果必须如此，那可以考虑给重要指令进行重命名，来避免命令被外人误执行。

【聊聊主从 – 同步原理】

从服务器会向主服务器发出SYNC指令，当主服务器接到此命令后，就会调用BGSAVE指令来创建一个子进程专门进行数据持久化工作，也就是将主服务器的数据写入RDB文件中。在数据持久化期间，主服务器将执行的写指令都缓存在内存中。

在BGSAVE指令执行完成后，主服务器会将持久化好的RDB文件发送给从服务器，从服务器接到此文件后会将其存储到磁盘上，然后再将其读取到内存中。这个动作完成后，主服务器会将这段时间缓存的写指令再以redis协议的格式发送给从服务器。

另外，要说的一点是，即使有多个从服务器同时发来SYNC指令，主服务器也只会执行一次BGSAVE，然后把持久化好的RDB文件发给多个下游。在redis2.8版本之前，如果从服务器与主服务器因某些原因断开连接的话，都会进行一次主从之间的全量的数据同步；而在2.8版本之后，redis支持了效率更高的增量同步策略，这大大降低了连接断开的恢复成本。

主服务器会在内存中维护一个缓冲区，缓冲区中存储着将要发给从服务器的内容。从服务器在与主服务器出现网络瞬断之后，从服务器会尝试再次与主服务器连接，一旦连接成功，从服务器就会把“希望同步的主服务器ID”和“希望请求的数据的偏移位置（replication offset）”发送出去。主服务器接收到这样的同步请求后，首先会验证主服务器ID是否和自己的ID匹配，其次会检查“请求的偏移位置”是否存在于自己的缓冲区中，如果两者都满足的话，主服务器就会向从服务器发送增量内容。

增量同步功能，需要服务器端支持全新的PSYNC指令。这个指令，只有在redis-2.8之后才具有。

【聊聊redis的事务处理】

众所周知，事务是指“一个完整的动作，要么全部执行，要么什么也没有做”。

在聊redis事务处理之前，要先和大家介绍四个redis指令，即MULTI、EXEC、DISCARD、WATCH。这四个指令构成了redis事务处理的基础。

1.MULTI用来组装一个事务；  
2.EXEC用来执行一个事务；  
3.DISCARD用来取消一个事务；  
4.WATCH用来监视一些key，一旦这些key在事务执行之前被改变，则取消事务的执行。

纸上得来终觉浅，我们来看一个MULTI和EXEC的例子：

复制代码 代码如下:

redis> MULTI //标记事务开始  
OK  
redis> INCR user\_id //多条命令按顺序入队  
QUEUED  
redis> INCR user\_id  
QUEUED  
redis> INCR user\_id  
QUEUED  
redis> PING  
QUEUED  
redis> EXEC //执行  
1) (integer) 1  
2) (integer) 2  
3) (integer) 3  
4) PONG

在上面的例子中，我们看到了QUEUED的字样，这表示我们在用MULTI组装事务时，每一个命令都会进入到内存队列中缓存起来，如果出现QUEUED则表示我们这个命令成功插入了缓存队列，在将来执行EXEC时，这些被QUEUED的命令都会被组装成一个事务来执行。

对于事务的执行来说，如果redis开启了AOF持久化的话，那么一旦事务被成功执行，事务中的命令就会通过write命令一次性写到磁盘中去，如果在向磁盘中写的过程中恰好出现断电、硬件故障等问题，那么就可能出现只有部分命令进行了AOF持久化，这时AOF文件就会出现不完整的情况，这时，我们可以使用redis-check-aof工具来修复这一问题，这个工具会将AOF文件中不完整的信息移除，确保AOF文件完整可用。

有关事务，大家经常会遇到的是两类错误：

1.调用EXEC之前的错误  
2.调用EXEC之后的错误

“调用EXEC之前的错误”，有可能是由于语法有误导致的，也可能时由于内存不足导致的。只要出现某个命令无法成功写入缓冲队列的情况，redis都会进行记录，在客户端调用EXEC时，redis会拒绝执行这一事务。（这时2.6.5版本之后的策略。在2.6.5之前的版本中，redis会忽略那些入队失败的命令，只执行那些入队成功的命令）。我们来看一个这样的例子：

复制代码 代码如下:

127.0.0.1:6379> multi  
OK  
127.0.0.1:6379> haha //一个明显错误的指令  
(error) ERR unknown command 'haha'  
127.0.0.1:6379> ping  
QUEUED  
127.0.0.1:6379> exec  
//redis无情的拒绝了事务的执行，原因是“之前出现了错误”  
(error) EXECABORT Transaction discarded because of previous errors.

而对于“调用EXEC之后的错误”，redis则采取了完全不同的策略，即redis不会理睬这些错误，而是继续向下执行事务中的其他命令。这是因为，对于应用层面的错误，并不是redis自身需要考虑和处理的问题，所以一个事务中如果某一条命令执行失败，并不会影响接下来的其他命令的执行。我们也来看一个例子：

复制代码 代码如下:

127.0.0.1:6379> multi  
OK  
127.0.0.1:6379> set age 23  
QUEUED  
//age不是集合，所以如下是一条明显错误的指令  
127.0.0.1:6379> sadd age 15   
QUEUED  
127.0.0.1:6379> set age 29  
QUEUED  
127.0.0.1:6379> exec //执行事务时，redis不会理睬第2条指令执行错误  
1) OK  
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value  
3) OK  
127.0.0.1:6379> get age  
"29" //可以看出第3条指令被成功执行了

好了，我们来说说最后一个指令“WATCH”，这是一个很好用的指令，它可以帮我们实现类似于“乐观锁”的效果，即CAS（check and set）。

WATCH本身的作用是“监视key是否被改动过”，而且支持同时监视多个key，只要还没真正触发事务，WATCH都会尽职尽责的监视，一旦发现某个key被修改了，在执行EXEC时就会返回nil，表示事务无法触发。

复制代码 代码如下:

127.0.0.1:6379> set age 23  
OK  
127.0.0.1:6379> watch age //开始监视age  
OK  
127.0.0.1:6379> set age 24 //在EXEC之前，age的值被修改了  
OK  
127.0.0.1:6379> multi  
OK  
127.0.0.1:6379> set age 25  
QUEUED  
127.0.0.1:6379> get age  
QUEUED  
127.0.0.1:6379> exec //触发EXEC  
(nil) //事务无法被执行

【教你看懂redis配置 – 简介】

我们可以在启动redis-server时指定应该加载的配置文件，方法如下：

复制代码 代码如下:

$ ./redis-server /path/to/redis.conf

接下来，我们就来讲解下redis配置文件的各个配置项的含义，注意，本文是基于redis-2.8.4版本进行讲解的。

redis官方提供的redis.conf文件，足有700+行，其中100多行为有效配置行，另外的600多行为注释说明。

在配置文件的开头部分，首先明确了一些度量单位：

复制代码 代码如下:

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

可以看出，redis配置中对单位的大小写不敏感，1GB、1Gb和1gB都是相同的。由此也说明，redis只支持bytes，不支持bit单位。

redis支持“主配置文件中引入外部配置文件”，很像C/C++中的include指令，比如：

复制代码 代码如下:

include /path/to/other.conf

如果你看过redis的配置文件，会发现还是很有条理的。redis配置文件被分成了几大块区域，它们分别是：

1.通用（general）  
2.快照（snapshotting）  
3.复制（replication）  
4.安全（security）  
5.限制（limits)  
6.追加模式（append only mode)  
7.LUA脚本（lua scripting)  
8.慢日志（slow log)  
9.事件通知（event notification）

下面我们就来逐一讲解。

【教你看懂redis配置 -通用】

默认情况下，redis并不是以daemon形式来运行的。通过daemonize配置项可以控制redis的运行形式，如果改为yes，那么redis就会以daemon形式运行：

复制代码 代码如下:

daemonize no

当以daemon形式运行时，redis会生成一个pid文件，默认会生成在/var/run/redis.pid。当然，你可以通过pidfile来指定pid文件生成的位置，比如：

复制代码 代码如下:

pidfile /path/to/redis.pid

默认情况下，redis会响应本机所有可用网卡的连接请求。当然，redis允许你通过bind配置项来指定要绑定的IP，比如：

复制代码 代码如下:

bind 192.168.1.2 10.8.4.2

redis的默认服务端口是6379，你可以通过port配置项来修改。如果端口设置为0的话，redis便不会监听端口了。

复制代码 代码如下:

port 6379

有些同学会问“如果redis不监听端口，还怎么与外界通信呢”，其实redis还支持通过unix socket方式来接收请求。可以通过unixsocket配置项来指定unix socket文件的路径，并通过unixsocketperm来指定文件的权限。

复制代码 代码如下:

unixsocket /tmp/redis.sock  
unixsocketperm 755

当一个redis-client一直没有请求发向server端，那么server端有权主动关闭这个连接，可以通过timeout来设置“空闲超时时限”，0表示永不关闭。

复制代码 代码如下:

timeout 0

TCP连接保活策略，可以通过tcp-keepalive配置项来进行设置，单位为秒，假如设置为60秒，则server端会每60秒向连接空闲的客户端发起一次ACK请求，以检查客户端是否已经挂掉，对于无响应的客户端则会关闭其连接。所以关闭一个连接最长需要120秒的时间。如果设置为0，则不会进行保活检测。

复制代码 代码如下:

tcp-keepalive 0

redis支持通过loglevel配置项设置日志等级，共分四级，即debug、verbose、notice、warning。

复制代码 代码如下:

loglevel notice

redis也支持通过logfile配置项来设置日志文件的生成位置。如果设置为空字符串，则redis会将日志输出到标准输出。假如你在daemon情况下将日志设置为输出到标准输出，则日志会被写到/dev/null中。

复制代码 代码如下:

logfile ""

如果希望日志打印到syslog中，也很容易，通过syslog-enabled来控制。另外，syslog-ident还可以让你指定syslog里的日志标志，比如：

复制代码 代码如下:

syslog-ident redis

而且还支持指定syslog设备，值可以是USER或LOCAL0-LOCAL7。具体可以参考syslog服务本身的用法。

复制代码 代码如下:

syslog-facility local0

对于redis来说，可以设置其数据库的总数量，假如你希望一个redis包含16个数据库，那么设置如下：

复制代码 代码如下:

databases 16

这16个数据库的编号将是0到15。默认的数据库是编号为0的数据库。用户可以使用select <DBid>来选择相应的数据库。

【教你看懂redis配置 – 快照】

快照，主要涉及的是redis的RDB持久化相关的配置，我们来一起看一看。

我们可以用如下的指令来让数据保存到磁盘上，即控制RDB快照功能：

复制代码 代码如下:

save <seconds> <changes>

举例来说：

复制代码 代码如下:

save 900 1 //表示每15分钟且至少有1个key改变，就触发一次持久化

save 300 10 //表示每5分钟且至少有10个key改变，就触发一次持久化

save 60 10000 //表示每60秒至少有10000个key改变，就触发一次持久化

如果你想禁用RDB持久化的策略，只要不设置任何save指令就可以，或者给save传入一个空字符串参数也可以达到相同效果，就像这样：

复制代码 代码如下:

save ""

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。这样做的好处在于可以让用户很明确的知道内存中的数据和磁盘上的数据已经存在不一致了。如果redis不顾这种不一致，一意孤行的继续接收写请求，就可能会引起一些灾难性的后果。

如果下一次RDB持久化成功，redis会自动恢复接受写请求。

当然，如果你不在乎这种数据不一致或者有其他的手段发现和控制这种不一致的话，你完全可以关闭这个功能，以便在快照写入失败时，也能确保redis继续接受新的写请求。配置项如下：

复制代码 代码如下:

stop-writes-on-bgsave-error yes

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。

复制代码 代码如下:

rdbcompression yes

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，如果你希望获取到最大的性能提升，可以关闭此功能。

复制代码 代码如下:

rdbchecksum yes

我们还可以设置快照文件的名称，默认是这样配置的：

复制代码 代码如下:

dbfilename dump.rdb

最后，你还可以设置这个快照文件存放的路径。比如默认设置就是当前文件夹：

复制代码 代码如下:

dir ./

【教你看懂redis配置 – 复制】

redis提供了主从同步功能。

通过slaveof配置项可以控制某一个redis作为另一个redis的从服务器，通过指定IP和端口来定位到主redis的位置。一般情况下，我们会建议用户为从redis设置一个不同频率的快照持久化的周期，或者为从redis配置一个不同的服务端口等等。

复制代码 代码如下:

slaveof <masterip> <masterport>

如果主redis设置了验证密码的话（使用requirepass来设置），则在从redis的配置中要使用masterauth来设置校验密码，否则的话，主redis会拒绝从redis的访问请求。

复制代码 代码如下:

masterauth <master-password>

当从redis失去了与主redis的连接，或者主从同步正在进行中时，redis该如何处理外部发来的访问请求呢？这里，从redis可以有两种选择：

第一种选择：如果slave-serve-stale-data设置为yes（默认），则从redis仍会继续响应客户端的读写请求。

第二种选择：如果slave-serve-stale-data设置为no，则从redis会对客户端的请求返回“SYNC with master in progress”，当然也有例外，当客户端发来INFO请求和SLAVEOF请求，从redis还是会进行处理。

你可以控制一个从redis是否可以接受写请求。将数据直接写入从redis，一般只适用于那些生命周期非常短的数据，因为在主从同步时，这些临时数据就会被清理掉。自从redis2.6版本之后，默认从redis为只读。

复制代码 代码如下:

slave-read-only yes

只读的从redis并不适合直接暴露给不可信的客户端。为了尽量降低风险，可以使用rename-command指令来将一些可能有破坏力的命令重命名，避免外部直接调用。比如：

复制代码 代码如下:

rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

从redis会周期性的向主redis发出PING包。你可以通过repl\_ping\_slave\_period指令来控制其周期。默认是10秒。

复制代码 代码如下:

repl-ping-slave-period 10

在主从同步时，可能在这些情况下会有超时发生：

1.以从redis的角度来看，当有大规模IO传输时。  
2.以从redis的角度来看，当数据传输或PING时，主redis超时  
3.以主redis的角度来看，在回复从redis的PING时，从redis超时

用户可以设置上述超时的时限，不过要确保这个时限比repl-ping-slave-period的值要大，否则每次主redis都会认为从redis超时。

复制代码 代码如下:

repl-timeout 60

我们可以控制在主从同步时是否禁用TCP\_NODELAY。如果开启TCP\_NODELAY，那么主redis会使用更少的TCP包和更少的带宽来向从redis传输数据。但是这可能会增加一些同步的延迟，大概会达到40毫秒左右。如果你关闭了TCP\_NODELAY，那么数据同步的延迟时间会降低，但是会消耗更多的带宽。（如果你不了解TCP\_NODELAY，可以到这里来科普一下）。

复制代码 代码如下:

repl-disable-tcp-nodelay no

我们还可以设置同步队列长度。队列长度（backlog)是主redis中的一个缓冲区，在与从redis断开连接期间，主redis会用这个缓冲区来缓存应该发给从redis的数据。这样的话，当从redis重新连接上之后，就不必重新全量同步数据，只需要同步这部分增量数据即可。

复制代码 代码如下:

repl-backlog-size 1mb

如果主redis等了一段时间之后，还是无法连接到从redis，那么缓冲队列中的数据将被清理掉。我们可以设置主redis要等待的时间长度。如果设置为0，则表示永远不清理。默认是1个小时。

复制代码 代码如下:

repl-backlog-ttl 3600

我们可以给众多的从redis设置优先级，在主redis持续工作不正常的情况，优先级高的从redis将会升级为主redis。而编号越小，优先级越高。比如一个主redis有三个从redis，优先级编号分别为10、100、25，那么编号为10的从redis将会被首先选中升级为主redis。当优先级被设置为0时，这个从redis将永远也不会被选中。默认的优先级为100。

复制代码 代码如下:

slave-priority 100

假如主redis发现有超过M个从redis的连接延时大于N秒，那么主redis就停止接受外来的写请求。这是因为从redis一般会每秒钟都向主redis发出PING，而主redis会记录每一个从redis最近一次发来PING的时间点，所以主redis能够了解每一个从redis的运行情况。

复制代码 代码如下:

min-slaves-to-write 3  
min-slaves-max-lag 10

上面这个例子表示，假如有大于等于3个从redis的连接延迟大于10秒，那么主redis就不再接受外部的写请求。上述两个配置中有一个被置为0，则这个特性将被关闭。默认情况下min-slaves-to-write为0，而min-slaves-max-lag为10。

【教你看懂redis配置 – 安全】

我们可以要求redis客户端在向redis-server发送请求之前，先进行密码验证。当你的redis-server处于一个不太可信的网络环境中时，相信你会用上这个功能。由于redis性能非常高，所以每秒钟可以完成多达15万次的密码尝试，所以你最好设置一个足够复杂的密码，否则很容易被黑客破解。

复制代码 代码如下:

requirepass zhimakaimen

这里我们通过requirepass将密码设置成“芝麻开门”。

redis允许我们对redis指令进行更名，比如将一些比较危险的命令改个名字，避免被误执行。比如可以把CONFIG命令改成一个很复杂的名字，这样可以避免外部的调用，同时还可以满足内部调用的需要：

复制代码 代码如下:

rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c89

我们甚至可以禁用掉CONFIG命令，那就是把CONFIG的名字改成一个空字符串：

复制代码 代码如下:

rename-command CONFIG ""

但需要注意的是，如果你使用AOF方式进行数据持久化，或者需要与从redis进行通信，那么更改指令的名字可能会引起一些问题。

【教你看懂redis配置 -限制】

我们可以设置redis同时可以与多少个客户端进行连接。默认情况下为10000个客户端。当你无法设置进程文件句柄限制时，redis会设置为当前的文件句柄限制值减去32，因为redis会为自身内部处理逻辑留一些句柄出来。

如果达到了此限制，redis则会拒绝新的连接请求，并且向这些连接请求方发出“max number of clients reached”以作回应。

复制代码 代码如下:

maxclients 10000

我们甚至可以设置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，或者我们设置了“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。但是对于无内存申请的指令，仍然会正常响应，比如GET等。

复制代码 代码如下:

maxmemory <bytes>

需要注意的一点是，如果你的redis是主redis（说明你的redis有从redis），那么在设置内存使用上限时，需要在系统中留出一些内存空间给同步队列缓存，只有在你设置的是“不移除”的情况下，才不用考虑这个因素。

对于内存移除规则来说，redis提供了多达6种的移除规则。他们是：

1.volatile-lru：使用LRU算法移除过期集合中的key  
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  
4.allkeys-random：移除随机的key  
5.volatile-ttl：移除那些TTL值最小的key，即那些最近才过期的key。  
6.noeviction：不进行移除。针对写操作，只是返回错误信息。

无论使用上述哪一种移除规则，如果没有合适的key可以移除的话，redis都会针对写请求返回错误信息。

复制代码 代码如下:

maxmemory-policy volatile-lru

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。假如redis默认会检查三个key并选择其中LRU的那个，那么你可以改变这个key样本的数量。

复制代码 代码如下:

maxmemory-samples 3

最后，我们补充一个信息，那就是到目前版本（2.8.4）为止，redis支持的写指令包括了如下这些：

复制代码 代码如下:

set setnx setex append  
incr decr rpush lpush rpushx lpushx linsert lset rpoplpush sadd  
sinter sinterstore sunion sunionstore sdiff sdiffstore zadd zincrby  
zunionstore zinterstore hset hsetnx hmset hincrby incrby decrby  
getset mset msetnx exec sort

【教你看懂redis配置 – 追加模式】

默认情况下，redis会异步的将数据持久化到磁盘。这种模式在大部分应用程序中已被验证是很有效的，但是在一些问题发生时，比如断电，则这种机制可能会导致数分钟的写请求丢失。

如博文上半部分中介绍的，追加文件（Append Only File）是一种更好的保持数据一致性的方式。即使当服务器断电时，也仅会有1秒钟的写请求丢失，当redis进程出现问题且操作系统运行正常时，甚至只会丢失一条写请求。

我们建议大家，AOF机制和RDB机制可以同时使用，不会有任何冲突。对于如何保持数据一致性的讨论，请参见[本文](http://redis.io/topics/persistence)。

复制代码 代码如下:

appendonly no

我们还可以设置aof文件的名称：

复制代码 代码如下:

appendfilename "appendonly.aof"

fsync()调用，用来告诉操作系统立即将缓存的指令写入磁盘。一些操作系统会“立即”进行，而另外一些操作系统则会“尽快”进行。

redis支持三种不同的模式：

1.no：不调用fsync()。而是让操作系统自行决定sync的时间。这种模式下，redis的性能会最快。  
2.always：在每次写请求后都调用fsync()。这种模式下，redis会相对较慢，但数据最安全。  
3.everysec：每秒钟调用一次fsync()。这是性能和安全的折衷。

默认情况下为everysec。有关数据一致性的揭秘，可以参考[本文](http://antirez.com/post/redis-persistence-demystified.html)。

复制代码 代码如下:

appendfsync everysec

当fsync方式设置为always或everysec时，如果后台持久化进程需要执行一个很大的磁盘IO操作，那么redis可能会在fsync()调用时卡住。目前尚未修复这个问题，这是因为即使我们在另一个新的线程中去执行fsync()，也会阻塞住同步写调用。

为了缓解这个问题，我们可以使用下面的配置项，这样的话，当BGSAVE或BGWRITEAOF运行时，fsync()在主进程中的调用会被阻止。这意味着当另一路进程正在对AOF文件进行重构时，redis的持久化功能就失效了，就好像我们设置了“appendsync none”一样。如果你的redis有时延问题，那么请将下面的选项设置为yes。否则请保持no，因为这是保证数据完整性的最安全的选择。

复制代码 代码如下:

no-appendfsync-on-rewrite no

我们允许redis自动重写aof。当aof增长到一定规模时，redis会隐式调用BGREWRITEAOF来重写log文件，以缩减文件体积。

redis是这样工作的：redis会记录上次重写时的aof大小。假如redis自启动至今还没有进行过重写，那么启动时aof文件的大小会被作为基准值。这个基准值会和当前的aof大小进行比较。如果当前aof大小超出所设置的增长比例，则会触发重写。另外，你还需要设置一个最小大小，是为了防止在aof很小时就触发重写。

复制代码 代码如下:

auto-aof-rewrite-percentage 100  
auto-aof-rewrite-min-size 64mb

如果设置auto-aof-rewrite-percentage为0，则会关闭此重写功能。

【教你看懂redis配置 – LUA脚本】

lua脚本的最大运行时间是需要被严格限制的，要注意单位是毫秒：

复制代码 代码如下:

lua-time-limit 5000

如果此值设置为0或负数，则既不会有报错也不会有时间限制。

【教你看懂redis配置 – 慢日志】

redis慢日志是指一个系统进行日志查询超过了指定的时长。这个时长不包括IO操作，比如与客户端的交互、发送响应内容等，而仅包括实际执行查询命令的时间。

针对慢日志，你可以设置两个参数，一个是执行时长，单位是微秒，另一个是慢日志的长度。当一个新的命令被写入日志时，最老的一条会从命令日志队列中被移除。

单位是微秒，即1000000表示一秒。负数则会禁用慢日志功能，而0则表示强制记录每一个命令。

复制代码 代码如下:

slowlog-log-slower-than 10000

慢日志最大长度，可以随便填写数值，没有上限，但要注意它会消耗内存。你可以使用SLOWLOG RESET来重设这个值。

复制代码 代码如下:

slowlog-max-len 128

【教你看懂redis配置 – 事件通知】

redis可以向客户端通知某些事件的发生。这个特性的具体解释可以参见[本文](http://redis.io/topics/keyspace-events)。

【教你看懂redis配置 – 高级配置】

有关哈希数据结构的一些配置项：

复制代码 代码如下:

hash-max-ziplist-entries 512  
hash-max-ziplist-value 64

有关列表数据结构的一些配置项：

复制代码 代码如下:

list-max-ziplist-entries 512  
list-max-ziplist-value 64

有关集合数据结构的配置项：

复制代码 代码如下:

set-max-intset-entries 512

有关有序集合数据结构的配置项：

复制代码 代码如下:

zset-max-ziplist-entries 128  
zset-max-ziplist-value 64

关于是否需要再哈希的配置项：

复制代码 代码如下:

activerehashing yes

关于客户端输出缓冲的控制项：

复制代码 代码如下:

client-output-buffer-limit normal 0 0 0  
client-output-buffer-limit slave 256mb 64mb 60  
client-output-buffer-limit pubsub 32mb 8mb 60

有关频率的配置项：

复制代码 代码如下:

hz 10

有关重写aof的配置项

复制代码 代码如下:

aof-rewrite-incremental-fsync yes

至此，redis的入门内容就结束了，内容实在不少，但相对来说都很基础，本文没有涉及redis集群、redis工作原理、redis源码、redis相关LIB库等内容，后续会陆续奉献，大家敬请期待：）

谢谢！

#### 您可能感兴趣的文章:

* [Redis中的动态字符串学习教程](http://www.jb51.net/article/70933.htm)
* [Redis有序集合类型的常用命令小结](http://www.jb51.net/article/93531.htm)
* [redis 队列操作的例子(php)](http://www.jb51.net/article/30093.htm)
* [redis常用命令、常见错误、配置技巧等分享](http://www.jb51.net/article/60627.htm)
* [Redis操作命令总结](http://www.jb51.net/article/61793.htm)
* [Redis中5种数据结构的使用场景介绍](http://www.jb51.net/article/54774.htm)
* [redis中使用redis-dump导出、导入、还原数据实例](http://www.jb51.net/article/57066.htm)
* [redis密码设置、访问权限控制等安全设置](http://www.jb51.net/article/50488.htm)
* [redis常用命令小结](http://www.jb51.net/article/51625.htm)
* [Redis字符串类型的常用命令小结](http://www.jb51.net/article/93569.htm)

Tags：[Redis](http://img.jb51.net/tag/Redis/1.htm) [入门教程](http://img.jb51.net/tag/%C8%EB%C3Ž̳%CC/1.htm)

## 相关文章

* 2016-05-05[在redhat6.4安装redis集群【教程】](http://www.jb51.net/article/83809.htm)
* 2015-06-06[Redis的Python客户端redis-py安装使用说明文档](http://www.jb51.net/article/68264.htm)
* 2016-11-11[在CentOS 7环境下安装Redis数据库详解](http://www.jb51.net/article/97350.htm)
* 2015-04-04[Redis教程(九)：主从复制配置实例](http://www.jb51.net/article/65262.htm)
* 2015-06-06[将MongoDB作为Redis式的内存数据库的使用方法](http://www.jb51.net/article/68600.htm)
* 2015-06-06[Redis的LRU机制介绍](http://www.jb51.net/article/68260.htm)
* 2014-09-09[64位Windows下安装Redis教程](http://www.jb51.net/article/54747.htm)
* 2015-07-07[mac下设置redis开机启动方法步骤](http://www.jb51.net/article/68749.htm)
* 2015-06-06[Redis中主键失效的原理及实现机制剖析](http://www.jb51.net/article/68254.htm)
* 2015-03-03[Redis操作命令总结](http://www.jb51.net/article/61793.htm)

## 最新评论

评论(21人参与*，*1条评论)





来说两句吧...

* [微博登录](javascript:;)
* [QQ登录](javascript:;)
* [手机登录](javascript:void(0))
* 最新评论



2015年8月24日 20:41 [菜蚜](javascript:void(0)) [*浙江省杭州市*网友]

沙发

[8回复分享](javascript:void(0))

Powered by 畅言

#### 大家感兴趣的内容

* 1[超强、超详细Redis数据库入门教程](http://www.jb51.net/article/56448.htm)
* 2[64位Windows下安装Redis教程](http://www.jb51.net/article/54747.htm)
* 3[redis常用命令、常见错误、配置技](http://www.jb51.net/article/60627.htm)
* 4[Redis操作命令总结](http://www.jb51.net/article/61793.htm)
* 5[Redis中5种数据结构的使用场景介](http://www.jb51.net/article/54774.htm)
* 6[redis中使用redis-dump导出、导入](http://www.jb51.net/article/57066.htm)
* 7[Redis中统计各种数据大小的方法](http://www.jb51.net/article/62877.htm)
* 8[redis常用命令小结](http://www.jb51.net/article/51625.htm)
* 9[让Redis在你的系统中发挥更大作用](http://www.jb51.net/article/51624.htm)
* 10[CentOS 6.6下Redis安装配置记录](http://www.jb51.net/article/62507.htm)

#### 最近更新的内容

* + [简单粗暴的Redis数据备份和恢复方法](http://www.jb51.net/article/87249.htm)
  + [使用Redis实现用户积分排行榜的教程](http://www.jb51.net/article/64726.htm)
  + [Redis sort 排序命令详解](http://www.jb51.net/article/69131.htm)
  + [将MongoDB作为Redis式的内存数据库的使用](http://www.jb51.net/article/68600.htm)
  + [redis的hGetAll函数的性能问题(记Redis那](http://www.jb51.net/article/80176.htm)
  + [在Redis数据库中实现分布式速率限制的方法](http://www.jb51.net/article/68515.htm)
  + [Redis02 使用Redis数据库(String类型)全面](http://www.jb51.net/article/88364.htm)
  + [Redis实现信息已读未读状态提示](http://www.jb51.net/article/82438.htm)
  + [Redis整合Spring结合使用缓存实例](http://www.jb51.net/article/77096.htm)
  + [Redis总结笔记（一）：安装和常用命令](http://www.jb51.net/article/59429.htm)

#### 常用在线小工具

[关于我们](http://www.jb51.net/about.htm) - [广告合作](http://www.jb51.net/support.htm) - [联系我们](http://www.jb51.net/linkus.htm) - [免责声明](http://www.jb51.net/sm.htm) - [网站地图](http://www.jb51.net/sitemap.htm) - [投诉建议](tencent://message/?uin=461478385&Site=http://www.jb51.net) - [在线投稿](http://www.jb51.net/up.htm)

©CopyRight 2006-2016 JB51.Net Inc All Rights Reserved. 脚本之家 版权所有