**Redis持久化详解**

# Redis持久化

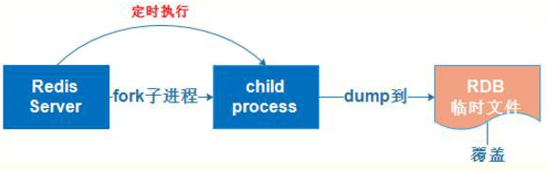
Redis是一种高级 key-value 型的NoSQL数据库。它跟 memcached类似是内存型数据库，不过Redis数据可以做持久化，即内存中的数据可以同步到磁盘进行存储。而且Redis所支持的数据类型很丰富。有字符串，链表，集合和有序集合等。Redis支持在服务器端计算集合的并，交和补集(difference)等，还支持多种排序功能。所以Redis也可以被看成是一个数据结构型服务器。

Redis的所有数据都是保存在内存中，然后不定期的通过异步方式保存到磁盘上(这称为“半持久化模式” RDB)；也可以把每一次数据变化都写入到一个 append only file(AOF)里面(这称为“全持久化模式”)。

Redis的数据都存放在内存中，如果没有配置持久化功能，Redis重启后数据就全丢失了，于是需要开启Redis的持久化功能，将数据保存到磁盘上，当Redis重启后，可以从磁盘中恢复数据。Redis提供两种方式进行持久化，一种是 RDB 持久化（原理是将 Redis在内存中的数据库记录定时 dump 到磁盘上的 RDB 持久化），另外一种是 AOF（append only file）持久化（原理是将Redis的操作日志以追加的方式写入文件）。那么这两种持久化方式有什么区别呢，该如何选择呢？

## RDB和AOF的区别

RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork 一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。



AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作。查询操作不会记录（类似于MySQL数据库的Binlog），以文本的方式记录，可以打开文件看到详细的操作记录。



## RDB和AOF的优缺点

RDB存在哪些优势呢？

1. 一旦采用该方式，那么你的整个Redis数据库将只包含一个文件，这对于文件备份而言是非常完美的。比如，你可能打算每个小时归档一次最近24小时的数据，同时还要每天归档一次最近30天的数据。通过这样的备份策略，一旦系统出现灾难性故障，我们可以非常容易的进行恢复。
2. 对于灾难恢复而言，RDB 是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介质上。
3. 性能最大化。对于Redis的服务进程而言，在开始持久化时，它唯一需要做的只是 fork出子进程，之后再由子进程完成这些持久化的工作，这样就可以极大的避免服务进程执行 IO 操作了。
4. 相比于AOF机制，如果数据集很大，RDB的启动效率会更高。

RDB又存在哪些劣势呢？

1. 如果想保证数据的高可用性，即最大限度的避免数据丢失，那么RDB将不是一个很好的选择。因为系统一旦在定时持久化之前出现宕机现象，此前没有来得及写入磁盘的数据都将丢失。
2. 由于RDB是通过 fork子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，可能会导致整个服务器停止服务几百毫秒，甚至是1秒钟。

AOF 的优势有哪些呢？

1. 该机制可以带来更高的数据安全性，即数据持久性。Redis中提供了3种同步策略，即每秒同步、每修改同步和不同步。事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。而每修改同步，我们可以将其视为同步持久化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是最低的。
2. 由于该机制对日志文件的写入操作采用的是 append（追加） 模式，因此在写入过程中即使出现宕机现象，也不会破坏日志文件中已经存在的内容。然而如果我们本次操作只是写入了一半数据就出现了系统崩溃问题，不用担心，在Redis下一次启动之前，我们可以通过 redis-check-aof工具来帮助我们解决数据一致性的问题。
3. 如果日志过大，Redis可以自动启用rewrite机制。即Redis以append模式不断的将修改数据写入到老的磁盘文件中，同时Redis还会创建一个新的文件用于记录此期间有哪些修改命令被执行。因此在进行 rewrite 切换时可以更好的保证数据安全性。
4. AOF 包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可以通过该文件完成数据的重建。

AOF 的劣势有哪些呢？

1. 对于相同数量的数据集而言，AOF文件通常要大于RDB文件。RDB在恢复大数据集时的速度比AOF的恢复速度要快。
2. 根据同步策略的不同，AOF在运行效率上往往会慢于RDB。总之，每秒同步策略的效率是比较高的，同步禁用策略的效率和RDB一样高效。

二者选择的标准，就是看系统是愿意牺牲一些性能，换取更高的缓存一致性（aof），还是愿意写操作频繁的时候，不启用备份来换取更高的性能，待手动运行save的时候，再做备份（rdb）。

Redis安装

[root@localhost ~]# tar xf redis-4.0.9.tar.gz -C /usr/src/

[root@localhost ~]# cd /usr/src/redis-4.0.9/

[root@localhost redis-4.0.9]# make

[root@localhost redis-4.0.9]# make install

[root@localhost redis-4.0.9]# cd utils/ #在解压路径下的utils中

[root@localhost utils]# ./install\_server.sh #执行安装脚本

Welcome to the redis service installer

This script will help you easily set up a running redis server

#一路回车即可

Please select the redis port for this instance: [6379] #设置服务端口号

Selecting default: 6379

Please select the redis config file name [/etc/redis/6379.conf] #设置主配置文件存放位置

Selected default - /etc/redis/6379.conf

Please select the redis log file name [/var/log/redis\_6379.log] #设置reids日志存放位置

Selected default - /var/log/redis\_6379.log

Please select the data directory for this instance [/var/lib/redis/6379] #设置数据目录

Selected default - /var/lib/redis/6379

Please select the redis executable path [/usr/local/bin/redis-server] #设置执行命令

Selected config:

Port : 6379 //服务端口

Config file : /etc/redis/6379.conf //配置文件

Log file : /var/log/redis\_6379.log //日志文件

Data dir : /var/lib/redis/6379 //数据目录

Executable : /usr/local/bin/redis-server //服务端命令

Cli Executable : /usr/local/bin/redis-cli //客户端命令

Is this ok? Then press ENTER to go on or Ctrl-C to abort.

Copied /tmp/6379.conf => /etc/init.d/redis\_6379

Installing service...

Successfully added to chkconfig!

Successfully added to runlevels 345!

Starting Redis server...

Installation successful! #显示此信息表示配置成功

在执行安装脚本后redis服务其实已经启动了

**[root@localhost ~]# netstat -anpt |grep 6379**

**tcp 0 0 127.0.0.1:6379 0.0.0.0:\* LISTEN 6956/redis-server**

# Redis 持久化配置

## 1、RDB持久化配置

Redis会将数据集的快照dump（将动态数据转储到静态形式如文件）到dump.rdb文件中。此外，我们也可以通过配置文件来修改Redis服务器dump快照的频率，在打开 6379.conf文件之后，我们搜索save可以看到下面的配置信息：

* save 900 1 #在900秒(15 分钟)之后，如果至少有1个key发生变化，则dump内存快照。
* save 300 10 #在300秒(5 分钟)之后，如果至少有10个key发生变化，则dump内存快照。
* save 60 10000 #在60秒(1 分钟)之后，如果至少有10000个key发生变化，则dump内存快照。

**[root@redis ~]# vim /etc/redis/6379.conf**

**219 save 900 1**

**220 save 300 10**

**221 save 60 10000**

**242 rdbcompression yes**

**254 dbfilename dump.rdb**

**264 dir /var/lib/redis/6379**

**[root@redis utils]# redis-cli -h 127.0.0.1 -p 6379**

**#默认端口（6379）及连接本地时可以省略-h和-p选项**

**127.0.0.1:6379> set name benet**

**OK**

**127.0.0.1:6379> set age 18**

**OK**

**127.0.0.1:6379> set address beijing**

**OK**

**127.0.0.1:6379>**

**127.0.0.1:6379> keys \***

**1) "address"**

**2) "age"**

**3) "name"**

**127.0.0.1:6379> save #人工保存，否则等到指定的时间才会保存**

**OK**

**127.0.0.1:6379> exit**

**[root@redis ~]# ls /var/lib/redis/6379**

**dump.rdb**

**[root@redis ~]# /etc/init.d/redis\_6379 restart #也可以用service redis\_6379 restart命令**

**Stopping ...**

**Redis stopped**

**Starting Redis server...**

**[root@redis ~]# redis-cli -h 127.0.0.1 -p 6379**

**127.0.0.1:6379> keys \***

**1) "name"**

**2) "address"**

**127.0.0.1:6379> get name**

**"benet"**

**127.0.0.1:6379> get address**

**"beijing"**

**[root@redis ~]# /etc/init.d/redis\_6379 stop**

**Stopping ...**

**Redis stopped**

**[root@redis ~]# rm -rf /var/lib/redis/6379/dump.rdb**

**[root@redis ~]# /etc/init.d/redis\_6379 start**

**Starting Redis server...**

**[root@redis ~]# ls /var/lib/redis/6379/**

**[root@redis ~]# redis-cli -h 127.0.0.1 -p 6379**

**127.0.0.1:6379> keys \***

**(empty list or set)**

## 2、AOF持久化配置

Redis的配置文件中存在三种同步方式，它们分别是：

* appendfsync always #每次有数据修改发生时都会写入AOF文件。
* appendfsync everysec #每秒钟同步一次，该策略为AOF的缺省策略。
* appendfsync no #从不同步。高效但是数据不会被持久化。

**[root@redis ~]# /etc/init.d/redis\_6379 stop**

**[root@redis ~]# rm -rf /var/lib/redis/6379/dump.rdb**

**[root@redis ~]# /etc/init.d/redis\_6379 start**

**[root@redis ~]# vim /etc/redis/6379.conf**

**673 appendonly yes #开启半持久化**

**677 appendfilename "appendonly.aof" #存储的文件名**

**703 appendfsync everysec #同步方式**

**744 auto-aof-rewrite-percentage 100**

**745 auto-aof-rewrite-min-size 64mb #重写之后的体积大了至少一倍(100%)#并且AOF体积大于64MB时，**移除AOF文件中的冗余命令，使得AOF文件的体积变得尽可能的小

**[root@redis ~]# service redis\_6379 restart**

**[root@redis ~]# redis-cli**

**127.0.0.1:6379> set name benet**

**OK**

**127.0.0.1:6379> set age 18**

**OK**

**127.0.0.1:6379> set address beijing**

**OK**

**127.0.0.1:6379> exit #无需人工保存，默认aof每秒保存一次**

**[root@redis ~]# ls /var/lib/redis/6379/**

**appendonly.aof dump.rdb**

**[root@redis ~]# rm -rf /var/lib/redis/6379/dump.rdb**

**[root@redis ~]# cat /var/lib/redis/6379/appendonly.aof**

**\*2**

**$6**

**SELECT**

**$1**

**0**

**\*3**

**$3**

**set**

**$4**

**name**

**$10**

**benet**

**\*3**

**$3**

**set**

**$3**

**age**

**$2**

**18**

**\*3**

**$3**

**set**

**$7**

**address**

**$7**

**beijing**

**[root@redis ~]# service redis\_6379 restart**

**Stopping ...**

**Redis stopped**

**Starting Redis server...**

**[root@redis ~]# redis-cli**

**127.0.0.1:6379> keys \***

**1) "age"**

**2) "address"**

**3) "name"**

**127.0.0.1:6379> exit**

## 3、AOF 重写功能

AOF持久化存在以下缺点：

* Redis会不断地将被执行的命令记录到AOF文件里面，所以随着Redis不断运行，AOF文件的体积也会不断增长。在极端情况下，体积不断增大的AOF文件甚至可能会用完硬盘的所有可用空间。
* Redis在重启之后需要通过重新执行 AOF 文件记录的所有写命令来还原数据集，所以如果AOF文件的体积非常大，那么还原操作执行的时间就可能会非常长。

为了解决AOF文件体积不断增大的问题，用户可以向Redis发送BGREWRITEAOF命令，这个命令会通过移除AOF文件中的冗余命令来重写（rewrite）AOF文件，使AOF文件的体积变得尽可能地小。BGREWRITEAOF的工作原理和BGSAVE创建快照的工作原理非常相似： Redis会创建一个子进程，然后由子进程负责对AOF文件进行重写。因为AOF文件重写也需要用到子进程，所以快照持久化因为创建子进程而导致的性能问题和内存占用问题，在AOF持久化中也同样存在。

跟快照持久化可以通过设置save选项来自动执行BGSAVE一样，AOF持久化也可以通过设置auto-aof-rewrite-percentage选项和 auto-aof-rewrite-min-size选项来自动执行BGREWRITEAOF。

举个例子，假设用户对Redis设置了配置选项auto-aof-rewrite-percentage 100和 auto-aof-rewrite-min-size 64mb，并且启动了AOF持久化，那么当AOF文件的体积大于64MB，并且AOF文件的体积比上一次重写之后的体积大了至少一倍（100%）的时候，Redis将执行 BGREWRITEAOF 命令。如果AOF重写执行得过于频繁的话，用户可以考虑将 auto-aof-rewrite-percentage 选项的值设置为100以上，这种做法可以让Redis在AOF文件的体积变得更大之后才执行重写操作，不过也会让Redis在启动时还原数据集所需的时间变得更长。

验证BGREWRITEAOF（重写）功能

[root@localhost 6379]# redis-cli

127.0.0.1:6379> set k1 a

OK

127.0.0.1:6379> set k1 b

OK

127.0.0.1:6379> set k1 c

OK

127.0.0.1:6379> exit

[root@localhost 6379]# cat appendonly.aof

\*2

$6

SELECT

$1

0

\*3

$3

set

$2

k1

$1

a

\*3

$3

set

$2

k1

$1

b

\*3

$3

set

$2

k1

$1

c

#没有重写时aof文件中保存了k1分别设置值为a,b,c的三个命令

[root@localhost 6379]# redis-cli

127.0.0.1:6379> keys \*

1) "k1"

127.0.0.1:6379> BGREWRITEAOF #手工执行重写,不区分大小写

Background append only file rewriting started

127.0.0.1:6379> exit

[root@localhost 6379]# cat /var/lib/redis/6379/appendonly.aof

\*2

$6

SELECT

$1

0

\*3

$3

SET

$2

k1

；

#可以看到执行BGREWRITEAOF后k1的前两个值a和b的写入命令已经删除了，只保留了最后的c这个值的命令

**同时开启两种持久化方式**：

默认只开启了rdb，没有开启AOF。

①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。

②RDB的数据不实时，同时使用两者时服务器重启也只会找AOF文件，那要不要使用AOF呢？建议不要，因为RDB更适合用于备份数据库(AOF在不断变化不好备份)，快速重启，而且不会有AOF可能潜在的bug，留着作为一个万一的手段。