第四十六章：Redis数据库持久化

**一、数据库概述；**

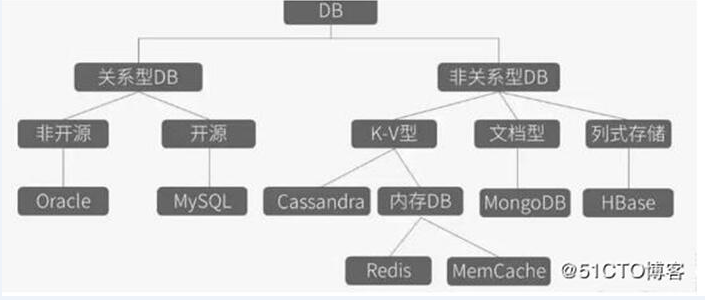
**二、持久化的作用以及实现方式；**

**三、RDB持久化方式配置；**

**四、AOF持久化方式配置；**

**一、数据库概述：**

Redis是一种高级key-value数据库。它跟memcached类似，不过数据可以持久化，而且支持的数据类型很丰富。有字符串，链表，集 合和有序集合。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服务器。



**二、持久化的作用以及实现方式：**

**作用：**

Redis的所有数据都是保存在内存中，如果没有配置持久化，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。那么不定期的通过异步方式保存到磁盘上（半持久化模式）；也可以把每一次数据变化都写入到一个append only file里面（全持久化模式）。 如若在服务器中开启了两种持久化的方式，默认执行AOF持久化方式；

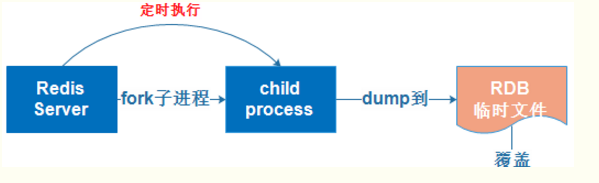
**实现方式：**

RDB持久化：将Reids在内存中的数据库记录定时dump到磁盘上，类似于快照功能。

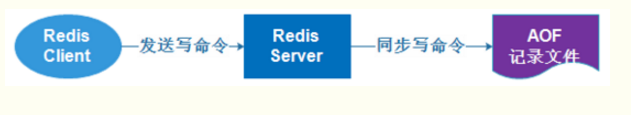
AOF持久化：append only file--原理是将Reids的操作日志以追加的方式写入文件，近似实时性。

**二者的区别：**

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持久化是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，以文本的方式记录，可以打开文件看到详细的操作记录。



**RDB的优势：**

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主进程执行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的劣势：**

1). 如果你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周期之内出现宕机现象，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：**

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中同步策略，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：**

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，如果在数据量比较大的情况下，AOF在运行效率上往往会慢于RDB。

**应用场景：**

1). 愿意牺牲一些性能（选择AOF）；

2). 换取更高的缓存一致性，保证数据库最高可能的完整性（选择AOF）；

3). 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）；

**三、RDB持久化方式配置；**

**环境准备：**根据“Redis高性能数据库”安装Redis服务；

[root@redis ~]# vi /usr/local/redis/redis.conf

218 save 900 1 ##开启RDB快照功能，在900秒内完成一个key值得变动触发快照功能，如若将所有save配置项都注释掉，那么则将关闭RDB快照功能

219 save 300 10

220 save 60 10000

235 stop-writes-on-bgsave-error yes ##当RDB快照后台进程失败，不影响用户得写操作

241 rdbcompression yes ##是否将RDB快照文件压缩，关闭后会增加性能

250 rdbchecksum no ##关闭RDB快照文件的检查校验，增加性能

253 dbfilename dump.rdb ##快照文件的名称

263 dir /usr/local/redis/ ##快照文件的存储路径

[root@redis redis]# /etc/init.d/redis restart

[root@redis redis]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> set haha hehe

OK

192.168.100.101:6379> set name 111

OK

192.168.100.101:6379> keys \*

1) "haha"

2) "name"

192.168.100.101:6379> exit

[root@redis redis]# reboot ##重启后验证key值是否还是存在

[root@redis redis]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> keys \*

1) "haha"

2) "name"

192.168.100.101:6379> exit

**四、AOF持久化方式配置；**

[root@redis ~]# vi /usr/local/redis/redis.conf

218 #save 900 1

219 #save 300 10

220 #save 60 10000

672 appendonly yes ##开启AOF持久化方式

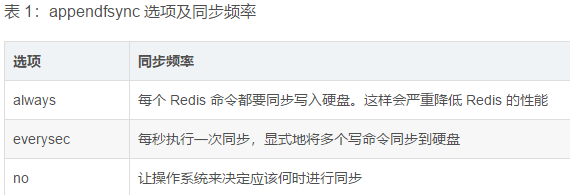
676 appendfilename "appendonly.aof" ##指定持久化的文件

702 appendfsync everysec ##频率，具体参数见下方表一

724 no-appendfsync-on-rewrite no ##如若设置为yes，则redis执行的命令会存放到缓冲区，待系统自动同步到硬盘

743 auto-aof-rewrite-percentage 100 ##若当前写入的AOF文件达到了上次rewrite文件大小的100%，则触发rewrite操作

744 auto-aof-rewrite-min-size 64mb ##设置AOF持久化重写文件的最小值，当达到60M并且符合100%的条件时，则触发rewrite操作



AOF 文件的生成过程具体包括命令追加，文件写入，文件同步三个步骤。   
Redis 打开 AOF 持久化功能后，Redis 在执行完一个写命令后，都会将执行的写命令追回到 Redis 内部的缓冲区的末尾。这个过程是命令的追加过程。 接下来，缓冲区的写命令会被写入到 AOF 文件，这一过程是文件写入过程。对于操作系统来说，调用write函数并不会立刻将数据写入到硬盘，为了将数据真正写入硬盘，还需要调用fsync函数，调用fsync函数即是文件同步的过程。只有经过文件同步过程，AOF 文件才在硬盘中真正保存了 Redis 的写命令。appendfsync 配置选项正是用来配置将写命令同步到文件的频率的，各个选项的值的含义如表 1 所示。而值为no的话表示为写入AOF文件，但是不进行磁盘同步，根据linux系统默认进行磁盘同步，默认30s，效率是最高的；

[root@redis ~]# /etc/init.d/redis restart

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> keys \*

1) "haha"

2) "name"

192.168.100.101:6379> set benet cloud

OK

192.168.100.101:6379> keys \*

1) "benet"

2) "haha"

3) "name"

192.168.100.101:6379> get benet

"cloud"

192.168.100.101:6379> exit

[root@redis ~]# cat /usr/local/redis/appendonly.aof

\*2

$6

SELECT

$1

0

\*3

$3

set

$4

haha

$1

1

\*3

$3

set

$4

name

$1

2

\*3

$3

set

$5

benet

$4

accp

[root@redis ~]# reboot ##重启后验证key是否存在

[root@redis ~]# /etc/init.d/redis restart

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> keys \*

1) "name"

2) "haha"

3) "benet"

192.168.100.101:6379> exit