

NoSQLFan

关注NoSQL相关的新闻与技术

- [NoSQL书籍](#)
- [分类列表](#)
- [关于本站](#)
- [NoSQL讨论区](#)

解密Redis持久化

作者: [nosqlfan](#) on 星期二, 三月 27, 2012 · [4条评论](#) 【阅读: 17,365 次】

本文内容来源于[Redis](#)作者博文, Redis作者说, 他看到的所有针对Redis的讨论中, 对Redis[持久化](#)的误解是最大的, 于是他写了[一篇长文](#)来对Redis的持久化进行了系统性的论述。文章非常长, 也很值得一看, NoSQLFan将主要内容简述成本文。

什么是持久化, 简单来讲就是将数据放到断电后数据不会丢失的设备中。也就是我们通常理解的硬盘上。

写操作的流程

首先我们来看一下数据库在进行写操作时到底做了哪些事, 主要有下面五个过程。

1. 客户端向服务端发送写操作 (数据在客户端的内存中)
2. 数据库服务端接收到写请求的数据 (数据在服务端的内存中)
3. 服务端调用write(2) 这个系统调用, 将数据往磁盘上写 (数据在系统内存的缓冲区中)
4. 操作系统将缓冲区中的数据转移到磁盘控制器上 (数据在磁盘缓存中)
5. 磁盘控制器将数据写到磁盘的物理介质中 (数据真正落到磁盘上)

故障分析

写操作大致有上面5个流程, 下面我们结合上面的5个流程看一下各种级别的故障。

- 当数据库系统故障时, 这时候系统内核还是OK的, 那么此时只要我们执行完了第3步, 那么数据就是安全的, 因为后续操作系统会来完成后面几步, 保证数据最终会落到磁盘上。
- 当系统断电, 这时候上面5项中提到的所有缓存都会失效, 并且数据库和操作系统都会停止工作。所以只有当数据在完成第5步后, 机器断电才能保证数据不丢失, 在上述四步中的数据都会丢失。

通过上面5步的了解, 可能我们会希望搞清下面一些问题:

- 数据库多长时间调用一次write(2), 将数据写到内核缓冲区
- 内核多长时间会将系统缓冲区中的数据写到磁盘控制器
- 磁盘控制器又在什么时候把缓存中的数据写到物理介质上

对于第一个问题, 通常数据库层面会进行全面控制。而对第二个问题, 操作系统有其默认的策略, 但是我们也可以通过POSIX API提供的fsync系列命令强制操作系统将数据从内核区写到磁盘控制器上。对于第三个问题, 好像数据库已经无法触及, 但实际上, 大多数情况下磁盘缓存是被设置关闭的。或者是只开启为读缓存, 也就是写操作不会进行缓存, 直接写到磁盘。建议的做法是仅仅当你的磁盘设备有备用电池时才开启写缓存。

数据损坏

所谓数据损坏, 就是数据无法恢复, 上面我们讲的都是如何保证数据是确实写到磁盘上去, 但是写到磁盘上可能并不意味着数据不会损坏。比如我们可能一次写请求会进行两次不同的写操作, 当意外发生时, 可能会导致一次写操作安全完成, 但是另一次还没有进行。如果数据库的数据文件结构组织不合理, 可能会导致数据完全不能恢复的状况出现。

这里通常也有三种策略来组织数据, 以防止数据文件损坏到无法恢复的情况:

- 第一种是最粗糙的处理, 就是不通过数据的组织形式保证数据的可恢复性。而是通过配置数据同步备份的方式, 在数据文件损坏后通过数据备份来进行恢复。实际上MongoDB在不开启journaling日志, 通过配置Replica Sets时就是这种情况。
- 另一种是在上面基础上添加一个操作日志, 每次操作时记一下操作的行为, 这样我们可以通过操作日志来进行数据恢复。因为操作日志是顺序追加的方式写的, 所以不会出现操作日志也无法恢复的情况。这也类似于MongoDB开启了journaling日志的情况。
- 更保险的做法是数据库不进行老数据的修改, 只是以追加方式去完成写操作, 这样数据本身就是一份日志, 这样就永远不会出现数据无法恢复的情况了。实际上CouchDB就是此做法的优秀范例。

RDB快照

下面我们说一下Redis的第一个持久化策略, RDB快照。Redis支持将当前数据的快照存成一个数据文件的持久化机制。而一个持续写入的数据库如何生成快照呢。Redis借助了fork命令的copy on write机制。在生成快照时, 将当前进程fork出一个子进程, 然后在子进程中循环所有的数据, 将数据写成为RDB文件。

我们可以通过Redis的save指令来配置RDB快照生成的时机, 比如你可以配置当10分钟以内有100次写入就生成快照, 也可以配置当1小时内有1000次写入就生成快照, 也可以多个规则一起实施。这些规则的定义就在Redis的配置文件中, 你也可以通过Redis的CONFIG SET命令在Redis运行时设置规则, 不需要重启Redis。

Redis的RDB文件不会坏掉，因为其写操作是在一个新进程中进行的，当生成一个新的RDB文件时，Redis生成的子进程会先将数据写到一个临时文件中，然后通过原子性rename系统调用将临时文件重命名为RDB文件，这样在任何时候出现故障，Redis的RDB文件都总是可用的。

同时，Redis的RDB文件也是Redis主从同步内部实现中的一环。

但是，我们可以很明显的看到，RDB有他的不足，就是一旦数据库出现问题，那么我们的RDB文件中保存的数据并不是全新的，从上次RDB文件生成到Redis停机这段时间的数据全部丢掉了。在某些业务下，这是可以忍受的，我们也推荐这些业务使用RDB的方式进行持久化，因为开启RDB的代价并不高。但是对于另外一些对数据安全性要求极高的应用，无法容忍数据丢失的应用，RDB就无能为力了，所以Redis引入了另一个重要的持久化机制：[AOF](#)日志。

AOF日志

aof日志的全称是append only file，从名字上我们就能看出来，它是一个追加写入的日志文件。与一般数据库的binlog不同的是，AOF文件是可识别的纯文本，它的内容就是一个一个的Redis标准命令。比如我们进行如下实验，使用Redis2.6版本，在启动命令参数中设置开启aof功能：

```
./redis-server --appendonly yes
```

然后我们执行如下的命令：

```
redis 127.0.0.1:6379> set key1 Hello
OK
redis 127.0.0.1:6379> append key1 " World!"
(integer) 12
redis 127.0.0.1:6379> del key1
(integer) 1
redis 127.0.0.1:6379> del non_existing_key
(integer) 0
```

这时我们查看AOF日志文件，就会得到如下内容：

```
$ cat appendonly.aof
*2
$6
SELECT
$1
0
*3
$3
set
$4
key1
$5
Hello
*3
$6
append
$4
key1
$7
World!
*2
$3
del
$4
key1
```

可以看到，写操作都生成了一条相应的命令作为日志。其中值得注意的是最后一个del命令，它并没有被记录在AOF日志中，这是因为Redis判断出这个命令不会对当前数据集做出修改。所以不需要记录这个无用的写命令。另外AOF日志也不是完全按客户端的请求来生成日志的，比如命令INCRBYFLOAT在记AOF日志时就被记成一条SET记录，因为浮点数操作可能在不同的系统上会不同，所以为了避免同一份日志在不同的系统上生成不同的数据集，所以这里只将操作后的结果通过SET来记录。

AOF重写

你可以会想，每一条写命令都生成一条日志，那么AOF文件是不是会很大？答案是肯定的，AOF文件会越来越大，所以Redis又提供了一个功能，叫做AOF rewrite。其功能就是重新生成一份AOF文件，新的AOF文件中一条记录的操作只会有一次，而不像一份老文件那样，可能记录了对同一个值的多次操作。其生成过程和RDB类似，也是fork一个进程，直接遍历数据，写入新的AOF临时文件。在写入新文件的过程中，所有的写操作日志还是会写到原来老的AOF文件中，同时还会记录在内存缓冲区中。当重写操作完成后，会将所有缓冲区中的日志一次性写入到临时文件中。然后调用原子性的rename命令用新的AOF文件取代老的AOF文件。

从上面的流程我们能够看到，RDB和AOF操作都是顺序IO操作，性能都很高。而同时在通过RDB文件或者AOF日志进行数据库恢复的时候，也是顺序的读取数据加载到内存中。所以也不会造成磁盘的随机读。

AOF可靠性设置

AOF是一个写文件操作，其目的是将操作日志写到磁盘上，所以它也同样会遇到我们上面说的写操作的5个流程。那么写AOF的操作安全性又有多高呢。实际上这是可以设置的，在Redis中对AOF调用write(2)写入后，何时再调用fsync将其写到磁盘上，通过[appendfsync](#)选项来控制，下面appendfsync的三个设置项，安全强度逐渐变强。

appendfsync no

当设置appendfsync为no的时候，Redis不会主动调用fsync去将AOF日志内容同步到磁盘，所以这一切就完全依赖于操作系统的调试了。对大多数Linux操作系统，是每30秒进行一次fsync，将缓冲区中的数据写到磁盘上。

appendfsync everysec

当设置appendfsync为everysec的时候，Redis会默认每隔一秒进行一次fsync调用，将缓冲区中的数据写到磁盘。但是当这一次的fsync调用时长超过1秒时。Redis会采取延迟fsync的策略，再等一秒钟。也就是在两秒后再进行fsync，这一次的fsync就不管会执行多长时间都会进行。这时候由于在fsync时文件描述符会被阻塞，所以当前的写操作就会阻塞。

所以，结论就是，在绝大多数情况下，Redis会每隔一秒进行一次fsync。在最坏的情况下，两秒钟会进行一次fsync操作。

这一操作在大多数数据库系统中被称为group commit，就是组合多次写操作的数据，一次性将日志写到磁盘。

appendfsync always

当设置appendfsync为always时，每一次写操作都会调用一次fsync，这时数据是最安全的，当然，由于每次都会执行fsync，所以其性能也会受到影响。

对于pipelining有什么不同

对于pipelining的操作，其具体过程是客户端一次性发送N个命令，然后等待这N个命令的返回结果被一起返回。通过采用pipelining就意味着放弃了对每一个命令的返回值确认。由于在这种情况下，N个命令是在同一个执行过程中执行的。所以当设置appendfsync为everysec时，可能会有一些偏差，因为这N个命令可能执行时间超过1秒甚至2秒。但是可以保证的是，最长时间不会超过这N个命令的执行时间和。

与postgresql和MySQL的比较

这一块就不多说了，由于上面操作系统层面的数据安全已经讲了很多，所以其实不同的数据库在实现上都大同小异。总之最后的结论就是，在Redis开启AOF的情况下，其单机数据安全性并不比这些成熟的SQL数据库弱。

数据导入

这些持久化的数据有什么用，当然是用于重启后的数据恢复。Redis是一个内存数据库，无论是RDB还是AOF，都只是其保证数据恢复的措施。所以Redis在利用RDB和AOF进行恢复的时候，都会读取RDB或AOF文件，重新加载到内存中。相对于MySQL等数据库的启动时间来说，会长很多，因为MySQL本来是不需要将数据加载到内存中的。

但是相对来说，MySQL启动后提供服务时，其被访问的热数据也会慢慢加载到内存中，通常我们称之为预热，而在预热完成前，其性能都不会太高。而Redis的好处是一次性将数据加载到内存中，一次性预热。这样只要Redis启动完成，那么其提供服务的速度都是非常快的。

而在利用RDB和利用AOF启动上，其启动时间有一些差别。RDB的启动时间会更短，原因有两个，一是RDB文件中每一条数据只有一条记录，不会像AOF日志那样可能有一条数据的多次操作记录。所以每条数据只需要写一次就行了。另一个原因是RDB文件的存储格式和Redis数据在内存中的编码格式是一致的，不需要再进行数据编码工作。在CPU消耗上要远小于AOF日志的加载。

好了，大概内容就说到这里。更详细完整的版本请看Redis作者的博文：[Redis persistence demystified](#)。本文如有描述不周之处，就大家指正。

赠人玫瑰，手有余香，分享知识，德艺双馨！

anyShare



相关文章：

- 非同一般的Redis介绍
- 深入剖析Redis RDB持久化机制
- Redis2.6新功能预告：aof性能提升
- Redis短期发展规划
- Redis RDB文件格式全解析

[无关联推荐](#)

分类 [Redis](#) · tag [AOF](#), [appendfsync](#), [rdb](#), [Redis](#), [持久化](#)

5 条评论

107 条新浪微博

13 条腾讯微博

最新 最早 最热



jianfengye110

好详细的讲解

2012年3月28日

回复

顶

转发

whenlove



有些以前了解，有些是刚知道。
不过这个整体更详细呀。底层的一些知识也提到了。很好。

2012年3月28日 回复 顶 转发



Maddemon
如何不持久化呢。。。纯缓存用，丢就丢了

2013年2月4日 回复 顶 转发



Allen
memcache

2013年2月16日 回复 顶 转发



paul
nice

2013年6月13日 回复 顶 转发

社交帐号登录: 微博 QQ 人人 豆瓣 [更多»](#)



说点什么吧...

发布

Powered by 多说

•

• 关注与订阅

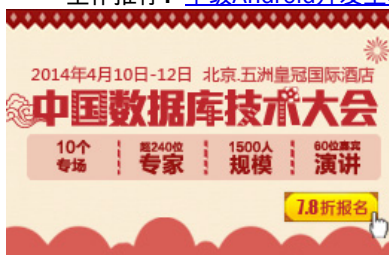


• 《Redis设计与实现》转发送书活动



• 专题

- [Redis资料汇总专题 new](#)
- [MongoDB资料汇总专题 new](#)
- 工作推荐: [高级Android开发工程师](#)
- 工作推荐: [中级Android开发工程师](#)



• NoSQLFan图书频道



• 热门文章

- [Redis资料汇总专题](#)
- [十五分钟介绍 Redis数据结构](#)
- [MongoDB资料汇总专题](#)
- [强烈推荐！非同一般的Redis介绍](#)
- [Redis系统性介绍](#)
- [一个很棒的Redis介绍PPT](#)
- [redis 适用场景与实现](#)
- [2013年中国数据库大会PPT](#)
- [MongoDB管理工具](#)
- [Redis资料汇总](#)

• 标签云

[node.js](#)
[源码](#)
[Redis](#)
[Neo4j](#)
[Python](#)
[Cassandra](#)
[hadoop](#)
[MongoDB](#)
[CouchDB](#)
[hbase](#)
[Riak](#)
[性能](#)
[nosql](#)
[Replica Sets](#)
[LevelDB](#)
[介绍](#)
[监控](#)
[mapreduce](#)
[MySQL](#)
[教程](#)

• 分类

- [Cassandra](#) (41)
- [CouchDB](#) (55)
- [Hadoop&HBase](#) (57)
- [Hypertable](#) (4)
- [Kyoto Cabinet](#) (2)
- [MongoDB](#) (203)
- [Neo4j](#) (8)
- [NoSQL杂谈](#) (114)
- [Redis](#) (160)
- [未分类](#) (74)
- [理论原地](#) (46)
- [站内事务](#) (3)

• 最近文章

- [我对Lamport Logical Clock的理解](#)
- [深入理解Redis主键失效原理及实现机制](#)
- [MongoDB 2.5版本将提供新的查询引擎](#)
- [2013年中国数据库大会PPT](#)
- [Redis核心解读系列](#)
- [MongoDB 2.4版本发布](#)
- [Redis核心解读-集群管理工具\(Redis-sentinel\)](#)
- [MapReduce的模式、算法和用例](#)
- [Redis监控技巧](#)
- [NoSQL反模式 - 文档数据库篇](#)
- [Redis如何处理客户端连接](#)
- [MongoDB查询谜题 \(1\)](#)
- [Twemproxy - Twitter 开源的 Redis proxy](#)
- [从MySQL到Redis, 提升数据迁移的效率](#)
- [NoSQL数据库的分布式算法](#)

• 热门文章

- [Redis资料汇总专题](#)
- [十五分钟介绍 Redis数据结构](#)
- [MongoDB资料汇总专题](#)
- [强烈推荐！非同一般的Redis介绍](#)
- [Redis系统性介绍](#)
- [一个很棒的Redis介绍PPT](#)
- [redis 适用场景与实现](#)
- [2013年中国数据库大会PPT](#)
- [MongoDB管理工具](#)
- [Redis资料汇总](#)
- [Redis作者谈Redis应用场景](#)
- [新浪微博开放平台Redis实践](#)
- [Memcached真的过时了吗？](#)
- [\[译\] NoSQL生态系统](#)
- [Redis之七种武器](#)

• 文章索引模板

- [2013年九月](#)
- [2013年七月](#)
- [2013年五月](#)
- [2013年四月](#)
- [2013年三月](#)
- [2013年二月](#)
- [2013年一月](#)
- [2012年十一月](#)
- [2012年九月](#)
- [2012年八月](#)
- [2012年七月](#)
- [2012年六月](#)
- [2012年五月](#)
- [2012年四月](#)
- [2012年三月](#)
- [2012年二月](#)
- [2012年一月](#)
- [2011年十二月](#)
- [2011年十一月](#)
- [2011年十月](#)
- [2011年九月](#)
- [2011年八月](#)
- [2011年七月](#)
- [2011年六月](#)
- [2011年五月](#)
- [2011年四月](#)
- [2011年三月](#)
- [2011年二月](#)
- [2011年一月](#)
- [2010年十二月](#)
- [2010年十一月](#)
- [2010年十月](#)
- [2010年九月](#)
- [2010年八月](#)
- [2010年七月](#)
- [2010年六月](#)
- [2010年五月](#)
- [2010年四月](#)

• 链接表

- [10GT](#)
- [3w招聘-互联网招聘](#)
- [iammutex的个人博客](#)
- [Mongo IC - 盛大MongoDB云服务](#)
- [python-cn 新闻 招聘 分享 酷站](#)
- [TechTarget数据库](#)
- [VPSee](#)
- [开源中国社区](#)
- [我自然-颜开的博客](#)
- [结构之法 算法之道](#)
- [豆瓣NoSQL小组](#)
- [运维和开发](#)

