DOI:10.16667/j.issn.2095-1302.2015.03.032

Redis数据库特性分析

马豫星
(华北计算机系统工程研究所，北京 100083)

摘 要：Redis是一款开源的、网络化的、基于内存的、可进行数据持久化的Key-Value存储系统。详细介绍了redis数据库底层数据结构、数据库的持久化方式、数据库事务特性以及隐藏在设计之中的一些考量。阐明了Redis高效性的原因在于其精简高效的底层数据结构设计以及对具有高消耗的功能进行分散处理。

关键词：数据库；Redis；NoSQL；分散处理
中图分类号：TP316 文献标识码：A 文章编号：2095-1302 (2015) 03-00105-02

0 引 言

随着互联网的发展以及 Web 2.0 的兴起，超大规模以及高并发的纯动态型网站日渐成为主流，由于 SNS 类网站在数据存取过程中有着实时性等刚性需求的原因，致使关系型数据库越来越不足以胜任，这使得目前 NoSQL 数据库慢慢成了人们所关注的焦点，并大有成为取代关系型数据库而成为未来主流数据存储模式的趋势。当前 NoSQL 数据库很多，大部分都是开源的，其中比较知名的有：MemcacheDB、Redis、Tokyo Cabinet、Flare、MongoDB、CouchDB、Cassandra、Voldemort 等。本文主要介绍 Redis，这是一款足以满足海量读写需求基于 Key-Value 数据存储方式的高性能 NoSQL 数据库。

1 Redis 简介

Redis 是一款开源的、网络化的、基于内存的、可进行数据持久化的 Key-Value 存储系统。它的数据模型建立在外层，类似于其它结构化存储系统，是通过 Key 映射 Value 的方式来建立字典以保存数据，有别于其它结构化存储系统的是，它支持多类型存储，包括 String、List、Set、Sort set 和 Hash 等，你可以在这些数据类型上做很多原子性操作。

在操作方面，Redis 基于 TCP 协议的特性使得它可以通过管道的方式进行数据操作，Redis 本身提供了一个可连接 Server 的客户端，通过客户端，可方便地进行数据存取操作。

2 Redis 底层数据结构中的两种：字符串和字典

在 Redis 的内部，数据结构类型值由高效的数据结构和算法进行支持，并且在 Redis 自身的构建当中，也大量用到了这些数据结构。

2.1 字符串

SDS（Simple Dynamic String，简单动态字符串）是 Redis 底层所使用的字符串表示，几乎所有的 Redis 模块中都用了 SDS。用 SDS 取代 C 默认的 char* 类型。

因为 char* 类型的功能单一，抽象层次低，并且不能高效地支持一些 Redis 常用的操作，所以在 Redis 程序内部，绝大部分情况下都会使用 SDS 而不是 char* 来表示字符串。

在 C 语言中，字符串可以用一个 \0 结尾的 char 数组来表示。但是，它并不能高效地支持长度计算和追加这两种操作：

(1) 计算字符串长度的复杂度为 $\theta(N)$。

(2) 对字符串进行 N 次追加，必定需要对字符串进行 N 次内存重分配。

在 Redis 内部，字符串的追加和长度计算很常见，这两个简单的操作不应该成为性能的瓶颈。

另外，Redis 除了处理 C 字符串之外，还需要处理单纯的字节数组，以及服务器协议等内容，所以为了方便起见，Redis 的字符串表示还应该是二进制安全的：程序不应对字符串里面保存的数据做任何假设，数据可以是以 \0 结尾的 C 字符串，也可以是单纯的字节数组，或者其他格式的数据。

考虑到这两个原因，Redis 使用 SDS 类型替换了 C 语言的默认字符串表示：SDS 既可高效地实现追加和长度计算，同时是二进制安全的。

值得一提的是，在 Redis 最初的设计中就加入了统计信息：

在设计 SDS 的时候，在内部使用了 zmalloc 与 zfree 来动态使用内存，并记录占有内存大小，方便计算 Redis 的性能。

2.2 字典

实现字典的方法有很多种：为了兼顾高效和简单性，Redis 使用了哈希表。在实现哈希表时，有一个问题就是采用何种策略来解决碰撞问题。对于使用链地址法来解决碰撞问题的哈希表来说，哈希表的性能取决于哈希表大小与保存节点数量之间的比率：

(1) 哈希表的大小与节点数量，比率在 1：1 时，哈希表

收稿日期：2015-04-28

(C)1994-2020 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

的性能最好 ；

（2）如果节点数量比哈希表的大小要大很多的话，那么哈希表就会退化成多个链表，哈希表本身的性能优势便不复存在 ；

Redis 保证当上述比率达到一定值时，会执行 rehash 操作，即对哈希表进行扩容或缩减。当扩容时，是以空间换取时间，当缩减时是以时间换空间。由此可以看出 Redis 对时间和空间的高效利用率。当然，rehash 操作一般是渐进方式执行的。因为其中涉及到对整个哈希表的迁移，如果数据量很大，那么势必会影响系统的性能。

Redis 使用了两种渐进式的 rehash 方式 ：

（1）每次执行一次添加、查找、删除操作，rehash 都会被执行一次 ；

（2）当 Redis 的服务器常规任务执行时，rehash 会被执行。在规定的时间内，尽可能地对数据库字典中那些需要 rehash 的字典进行 rehash，从而加速数据库字典的 rehash 进程。

3 Redis 的持久化方式 ：RDB 与 AOF

在运行情况下，Redis 以数据结构的形式将数据维持在内存中，为了让这些数据在 Redis 重启之后仍然可用，Redis 分别提供了 RDB 和 AOF 两种持久化模式。

RDB 将数据库的快照以二进制的方式保存到磁盘中。在 Redis 运行时，RDB 程序将当前内存中的数据库快照保存到磁盘文件中，在 Redis 重启动时，RDB 程序可以通过载入 RDB 文件来还原数据库的状态。

AOF 则以协议文本的方式，将所有对数据库进行过写入的命令（及其参数）记录到 AOF 文件，以此达到记录数据库状态的目的。AOF 更像是历史记录，记录所有运行过的命令。但是 AOF 文件就会随着时间持续增长，进而占据整个磁盘。为此，Redis 设计了 AOF 重写机制，通过开启新线程，扫描数据库数据，将其转化为 Redis 命令，存入临时的 AOF 文件。当扫描完后，用临时文件代替 AOF 文件。这样一来，AOF 文件中记录的命令就是最简洁的，因而不会占据很多空间。

4 Redis 事务

4.1 一致性

Redis 的一致性问题可以分为两部分来讨论 ：入队错误、执行错误。

在命令入队的过程中，如果客户端向服务器发送了错误的命令，Redis 会拒绝执行事务，并返回失败信息。如果命令在事务执行的过程中发生错误，那么 Redis 只会将错误包含在事务的结果中，这不会引起事务中断或整个失败，不会影响已执行事务命令的结果，也不会影响后面要执行的事务命令，所以它对事务的一致性也没有影响。

4.2 隔离性

Redis 是单进程程序，并且它保证在执行事务时，不会对事务进行中断，事务可以运行直到执行完所有事务队列中的命令为止。因此，Redis 的事务是总是带有隔离性的。

4.3 原子性

在上述一致性的介绍中，可以看出在事务队列中，即使有命令执行错误，该事务也会执行完，符合原子性的要求。

4.4 持久性

因为事务不过是用队列包裹起了一组 Redis 命令，并没有提供任何额外的持久性功能，所以事务的持久性由 Redis 所使用的持久化模式决定 ：

在单纯的内存模式下，事务肯定是不持久的 ；

在 RDB 模式下，服务器可能在事务执行之后、RDB 文件更新之前的这段时间失败，所以 RDB 模式下的 Redis 事务也是不持久的 ；

在 AOF 的“ 总是 SYNC ”模式下，事务的每条命令在执行成功之后，都会立即调用 fsync 或 fdatasync 将事务数据写入到 AOF 文件。但是，这种保存是由后台线程进行的，主线程不会阻塞直到保存成功，所以从命令执行成功到数据保存到硬盘之间，还是有一段非常小的间隔，所以这种模式下的事务也是不持久的 ；

其他 AOF 模式也和“ 总是 SYNC ”模式类似，所以它们都是不持久的 ；

综上所述，Redis 事务满足原子性、一致性、隔离性，不满足持久性。

5 结 语

本文详细介绍了 Redis 数据库数据结构、事务、持久化等特性，为读者深入理解 Redis 提供了帮助。

参 考 文 献

[1] 李子骅 .Redis 入门指南 [M]. 北京 ：人民邮电出版社，2013.
[2] 黄健宏 .Redis 设计与实现 [M]. 北京 ：机械工业出版社，2014.
[3] 皮雄军 .NoSQL 数据库技术实战 [M]. 北京 ：清华大学出版社，2015.
[4] Shashank Tiwari. 深入 NoSQL[M]. 北京 ：人民邮电出版社，2012.
[5] 徐小威 . 非关系型数据库数据恢复技术研究 [D]. 杭州，杭州电子科技大学，2014.

作者简介 :马豫星 (1991—) , 男, 山西省河津市人, 硕士研究生。主要研究方向是工业智能与仿真。

(C)1994-2020 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net