Learning Redis

目录

1. Redis 简介

Redis 是一个 Key-Value 存储系统。它支持存储的数据类型有以下几种： string（字符串）、 list（链表）、 set（无序集合）、 zset(sorted set 有序集合）和 hash，可以把 Redis 看成一个数据结构服务器。 Set 集合与 ZSet 支持 add/remove 及取交集、并集和差集运算， Redis 支持各种不同方式的排序。数据都是缓存在内存中的，它也可以周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并实现了 master-slave（主从）同步。 Redis 提供的 API 语言包括 C 、 C++ 、 C# 、 Java 、 JavaScript 、 Lua 、 Objective-C 、 Perl 、 PHP 、 Python 、 Ruby 、 Go 、 Tcl 等。在使用方式上，应用程序直接访问 Redis 数据库，使用 Redis 要考虑内存问题。

2. 安装环境和过程

1. Ubuntu 上安装 Redis

1）打开终端，解压下载后的文件，使用命令切换到解压目录，然后键入以下命令：

```
$sudo apt-get update
$sudo apt-get install redis-server
```

进行安装 Redis 。

2）启动 Redis

```
$redis-server
```

3）检查 Redis 是否在工作？

输入命令

```
$redis-cli
```

显示如下信息

```
redis 127.0.0.1:6379>
```

上面的提示 127.0.0.1 是本机的 IP 地址， 6379 为 Redis 服务器运行的默认端口。

现在输入 ping 命令： redis 127.0.0.1:6379> ping
按下回车键显示如下信息：
PONG
这说明已经成功地安装 Redis 。

1. Windows 上安装 Redis

1）下载地址： https://github.com/dmajkic/redis/downloads。解压已下载的文件到指定目录。

2）打开一个 cmd 窗口使用 cd 命令，切换目录到安装目录。例如安装根目录为 c 则就要切换到 C:\redis，运行 redis-server.exe redis.conf 。如果想方便的话，可以把 redis 的路径加到系统的环境变量里，这样就省得再输路径了，后面的那个 redis.conf 可以省略，如果省略，会启用默认的配置。

3）重新开启一个 cmd 窗口（注：原来的命令行窗口不要关闭，否则无法访问服器）。切换到 redis 目录下键入 redis-cli.exe -h 127.0.0.1 -p 6379 。设置一个键值对如 set myKey abc，然后使用 get mykey 命令取出已经设置的值，如果控制台输出得到 mykey 对应的值，表示安装已经成功。

3. 基本命令

命令原型	时间复杂度	命令描述	返回值
HSET key field value	O(1)	为指定的Key设定Field/Value对，如果Key不存在，该命令将创建新Key以参数中的Field/Value对，如果参数中的Field在该Key中已经存在，则用新值覆盖其原有值。	1表示新的Field被设置了新值，0表示Field已经存在，用新值覆盖原有值。
HGET key field	O(1)	返回指定Key中指定Field的关联值。	返回参数中Field的关联值，如果参数中的Key或Field不存，返回nil。
HEXISTS key field	O(1)	判断指定Key中的指定Field是否存在。	1表示存在，0表示参数中的Field或Key不存在。
HLEN key	O(1)	获取该Key所包含的Field的数量。	返回Key包含的Field数量，如果Key不存在，返回0。
HDEL key field [field ...]	O(N)	时间复杂度中的N表示参数中待删除的字段数量。从指定Key的Hashes Value中删除参数中指定的多个字段，如果不存在的字段将被忽略。如果Key不存在，则将其视为空Hashes，并返回0.	实际删除的Field数量。

HSETNX key field value	O(1)	只有当参数中的Key或Field不存在的情况下，为指定的Key设定Field/Value对，否则该命令不会进行任何操作。	1表示新的Field被设置了新值，0表示Key或Field已经存在，该命令没有进行任何操作。
HINCRBY key field increment	O(1)	增加指定Key中指定Field关联的Value的值。如果Key或Field不存在，该命令将会创建一个新Key或新Field，并将其关联的Value初始化为0，之后再指定数字增加的操作。该命令支持的数字是64位有符号整型，即increment可以负数。	返回运算后的值。
HGETALL key	O(N)	时间复杂度中的N表示Key包含的Field数量。获取该键包含的所有Field/Value。其返回格式为一个Field、一个Value，并以此类推。	Field/Value的列表。
HKEYS key	O(N)	时间复杂度中的N表示Key包含的Field数量。返回指定Key的所有Fields名。	Field的列表。
HVALS key	O(N)	时间复杂度中的N表示Key包含的Field数量。返回指定Key的所有Values名。	Value的列表。
HMGET key field [field ...]	O(N)	时间复杂度中的N表示请求的Field数量。获取和参数中指定Fields关联的一组Values。如果请求的Field不存在，其值返回nil。如果Key不存在，该命令将其视为空Hash，因此返回一组nil。	返回和请求Fields关联的一组Values，其返回顺序等同于Fields的请求顺序。
HMSET key field value [field value ...]	O(N)	时间复杂度中的N表示被设置的Field数量。逐对依次设置参数中给出的Field/Value对。如果其中某个Field已经存在，则用新值覆盖原有值。如果Key不存在，则创建新Key，同时设定参数中的Field/Value。	

命令行使用用例

```
# 在 Shell 命令行启动 Redis 客户端程序
    /> redis-cli
    # 给键值为 myhash 的键设置字段为 field1，值为 stephen。
        redis 127.0.0.1:6379> hset myhash field1 "stephen"
    # 获取键值为 myhash，字段为 field1 的值。
        redis 127.0.0.1:6379> hget myhash field1
    #myhash 键中不存在 field2 字段，因此返回 nil。
        redis 127.0.0.1:6379> hget myhash field2
    # 给 myhash 关联的 Hashes 值添加一个新的字段 field2，其值为 liu。
        redis 127.0.0.1:6379> hset myhash field2 "liu"
    # 获取 myhash 键的字段数量。
        redis 127.0.0.1:6379> hlen myhash
  # 判断 myhash 键中是否存在字段名为 field1 的字段，由于存在，返回值为 1
    redis 127.0.0.1:6379> hexists myhash field1
```

删除 myhash 键中字段名为 field1 的字段，删除成功返回 1 。
redis 127.0.0.1:6379> hdel myhash field1
再次删除 myhash 键中字段名为 field1 的字段，由于上一条命令已经将其删除，因为没有删除，返回 0 。
redis 127.0.0.1:6379> hdel myhash field1
判断 myhash 键中是否存在 field1 字段，由于上一条命令已经将其删除，因为返回 0 。
redis 127.0.0.1:6379> hexists myhash field1
通过 hsetnx 命令给 myhash 添加新字段 field1 ，其值为 stephen ，因为该字段已经被删除，所以该命令添加成功并返回 1 。
redis 127.0.0.1:6379> hsetnx myhash field1 stephen
由于 myhash 的 field1 字段已经通过上一条命令添加成功，因为本条命令不做任何操作后返回 0 。
redis 127.0.0.1:6379> hsetnx myhash field1 stephen

4. 数据类型

1)String －字符串型

Redis 的字符串是字节序列。在 Redis 中字符串是二进制安全的，这意味着他们有一个已知的长度，是没有任何特殊字符终止决定的，所以可以存储任何东西，最大长度可达 *512M*。常用操作命令包括 SET 、GET 、 APPEND 、 BITCOUNT 、 BITOP 、 DECR 、 DECRBY 等。

应用场景

String 是最常用的一种数据类型，普通的 key/value 存储都可以归为此类， value 其实不仅是 String ，也可以是数字：比如想知道什么时候封锁一个 IP 地址（访问超过几次）。 INCRBY 命令让这些变得很容易，通过原子递增保持计数。

实现方式

decrby,decr 等操作时会转成数值型进行计算，此时 redisObject 的 encoding 字段为 int 。

2)Hash - 哈希值

Redis的哈希键值对的集合。 Redis的哈希值是字符串字段和字符串值之间的映射，所以它们被用来表示对象。常用命令包括hset，hget,hmset,hmget,hgetall等。

应用场景

例：我们要存储一个用户信息对象数据，包含以下信息：用户 ID 为查找的 key，存储的 value 用户对象包含姓名 name，年龄 age，生日 birthday。如果用普通的 key/value 结构来存储，主要有以下 2 种存储方式：

第一种方式将用户 ID 作为查找 key，把其他信息封装成一个对象以序列化的方式存储。如： set u001 ″李三,18,20010101″。这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，引入 CAS 等复杂问题。

第二种方法是这个用户信息对象有多少成员就存成多少个 key-value 对儿，用用户 ID+ 对应属性的名称作为唯一标识来取得对应属性的值。如： mset user:001:name ″李三″， user:001:age18，

user:001:birthday"20010101"。虽然省去了序列化开销和并发问题，但是用户 ID 为重复存储，如果存在大量这样的数据，内存仍然存在严重浪费情况。

Redis 提供的 Hash 很好地解决了这个问题， Redis 的 Hash 实际是内部存储的 Value 为一个 HashMap，并提供了直接存取这个 Map 成员的接口，如： hmset user:001 name "李三" age 18 birthday "20010101"。也就是说， Key 仍然是用户 ID, value 是一个 Map，这个 Map 的 key 是成员的属性名， value 是属性值。这样对数据的修改和存取都可以直接通过其内部 Map 的 Key(Redis 里称内部 Map 的 key 为 field)， 也就是通过 key(用户 ID) + field(属性标签) 操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题。

实现方式

Redis Hash 对应 Value 内部实际就是一个 HashMap，实际这里会有 2 种不同实现，这个 Hash 的成员比较少时 Redis 为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的 HashMap 结构，对应的 value redisObject 的 encoding 为 zipmap, 当成员数量增大时会自动转成真正的 HashMap, 此时 encoding 为 ht 。

3)List - 列表

Redis 的列表是简单的字符串列表，排序插入顺序。可以添加元素到 Redis 列表的头部或尾部。常用命令 lpush, rpush, lpop, rpop, lrange, rpush, BLPOP（阻塞版）等。

应用场景

Redis list 的应用场景非常多，也是 Redis 最重要的数据结构之一。我们可以轻松地实现最新消息排行等功能。Lists 的另一个应用就是消息队列，可以利用 Lists 的 PUSH 操作，将任务存在 Lists 中，然后工作线程再用 POP 操作将任务取出进行执行。

实现方式

Redis list 的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis 内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

4)Set-集合

Redis set 集合是字符串的无序集合。在 Redis 中可以添加，删除和测试文件是否存的成员。常用命令包括 Sadd, srem, spop, sdiff ,smembers,sunion 等。

应用场景

Redis set 对外提供的功能与 list 类似是一个列表的功能，特殊之处在于 set 是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时， set 是一个很好的选择，并且 set 提供了判断某个成员是否在一个 set 集合内的重要接口，这个也是 list 所不能提供的。

比如在微博应用中，每个人的好友存在一个集合（ set ）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。

Redis 还为集合提供了求交集、并集、差集等操作，可以非常方便的实现。

实现方式

set 的内部实现是一个 value 永远为 null 的 HashMap ，实际就是通过计算 hash 的方式来快速排重的，这也是 set 能提供判断一个成员是否在集合内的原因。

5)ZSet-有序集合

将一个或多个 member 元素及其 score 值加入到有序集 key 当中。如果某个 member 已经是有序集的成员，那么更新这个 member 的 score 值，并通过重新插入这个 member 元素，来保证该 member 在正确的位置上。 score 值可以是整数值或双精度浮点数。如果 key 不存在，则创建一个空的有序集并执行 ZADD 操作。当 key 存在但不是有序集类型时，返回一个错误。常用命令包括 zadd,zrange,zrem,zcard 等。

使用场景

以某个条件为权重，比如按顶的次数排序.

ZREVRANGE 命令可以用来按照得分来获取前 100 名用户， ZRANK 可以用来获取用户排名，非常直接而且操作容易。 Redis ZSset 的使用场景与 set 类似，区别是 set 不是自动有序的，而 sorted set 可以通过用户额外提供一个优先级（score）的参数来为成员排序，并且是插入有序的，即自动排序。

比如：全班同学成绩的 SortedSets， value 可以是同学的学号，而 score 就可以是其考试得分，这样数据插入集合的，就已经进行了天然的排序。

另外还可以用 Sorted Sets 来做带权重的队列，比如普通消息的 score 为 1，重要消息的 score 为 2，然后工作线程可以选择按 score 的倒序来获取工作任务。让重要的任务优先执行。

需要精准设定过期时间的应用

比如你可以把上面说到的 sorted set 的 score 值设置成过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据了，不仅是清除 Redis 中的过期数据，你完全可以把 Redis 里这个过期时间当成是对数据库中数据的索引，用 Redis 来找出哪些数据需要过期删除，然后再精准地从数据库中删除相应的记录。

实现方式

Redis sorted set 的内部使用 HashMap 和跳跃表（SkipList）来保证数据的存储和有序，HashMap 里放的是成员到 score 的映射，而跳跃表里存放的是所有的成员，排序依据是 HashMap 里存的 score，使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

5. 应用场景

1)High performance - 对数据高并发读写
2)Huge storage - 对海量数据的高效率存储和访问
3)High scalability && HighAvailability - 对数据的高可扩展性和高可用性

6. 持久化方式

1）快照
快照是默认的持久化方式。这种方式是就是将内存中数据以快照的方式写入到二进制文件中，默认的文件名为 dump.rdb 。
可以通过配置设置自动做快照持久化的方式。我们可以配置 redis 在 n 秒内如果超过 m 个 key 被修改就自动做快照。
下面是默认的快照保存配置：
save 900 1 #900 秒内如果超过 1 个 key 被修改，则发起快照保存
save 300 10 #300 秒内容如超过 10 个 key 被修改，则发起快照保存
save 60 10000

保存过程
1. redis 调用 fork，现在有了子进程和父进程。
2. 父进程继续处理 client 请求，子进程负责将内存内容写入到临时文件。
由于 os 的写时复制机制（copy on write）父子进程会共享相同的物理页面，当父进程处理写请求时 os 会为父进程要修改的页面创建副

本，而不是写共享的页面。所以子进程的地址空间内的数据是 fork 时刻整个数据库的一个快照。

3. 当子进程将快照写入临时文件完毕后，用临时文件替换原来的快照文件，然后子进程退出（ fork 一个进程入内在也被复制了，即内存会是原来的两倍）。

2)Append-only file

aof 比快照方式有更好的持久化性，是由于在使用 aof 持久化方式时， redis 会将每一个收到的写命令都通过 write 函数追加到文件中（默认是 appendonly.aof）。

当 redis 重启时会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。当然由于 os 会在内核中缓存 write 做的修改，所以可能不是立即写到磁盘上。这样 aof 方式的持久化也还是有可能会丢失部分修改。

不过我们可以通过配置文件告诉 redis 我们想要通过 fsync 函数强制 os 写入到磁盘的时机。有三种方式如下（默认是：每秒 fsync 一次）：

```
appendonly yes                  # 启用 aof 持久化方式
# appendfsync always    # 每次收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用
```

appendfsync everysec　　#每秒钟强制写入磁盘一次，在性能和持久化方面做了很好的折中，推荐
appendfsync no　　#完全依赖os，性能最好，持久化没保证

3）虚拟内存

由于虚拟内存方式不推荐使用．主要有以下原因。

a. slow restart 重启太慢
b. slow saving 保存数据太慢
c. slow replication 上面两条导致 replication 太慢
d. complex code 代码过于复杂

7. Redis Java 连接操作

Java 程序连接 Redis 步骤

1) 确保 Redis 服务开启

2) 向项目中添加 jar 包： jedis.jar

3) 创建连接如下代码

```
public static void main(String[] args) {
  Jedis jedis = new Jedis("localhost");
  jedis.lpush("tutorial-list","Redis");
  jedis.lpush("tutorial-list","Mongodb");
  jedis.lpush("tutorial-list","Mysql");
 List<String> list = jedis.lrange("tutorial-list",0,5);
 for(int i=0; i<list.size(); i++) {
   System.out.println(list.get(i));
 }
}
```

8. Redis 安全

Redis 数据库可以设置安全，所以做出相关的任何客户端都需要在执行命令之前进行身份验证。为了确保 Redis 需要设置在配置文件中的密码验证一致。

下面给出的例子显示的步骤，以确保 Redis 实例。

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
```

默认情况下，此属性为空，表示没有设置密码，此实例。您可以通过执行以下命令来更改这个属性

```
127.0.0.1:6379> CONFIG set requirepass "yiibai"
OK
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) "yiibai"
```

设置密码，如果任何客户端运行命令没有验证后，再（错误）NOAUTH 需要验证。错误将再回到这点。因此，客户端需要使用 AUTH 命令进行认证。

```
    127.0.0.1:6379> AUTH password
```

语法

```
127.0.0.1:6379> AUTH "yiibai"
OK
127.0.0.1:6379> SET mykey "Test value"
OK
127.0.0.1:6379> GET mykey
"Test value"
```

Thank you !