# 史上最全系列 | Redis 原理+知识点总结（1.5 万字，8 大知识点，17 张图）

本文作者：在 IT 中穿梭旅行

本文档来自公众号：3 分钟秒懂大数据

微信扫码关注

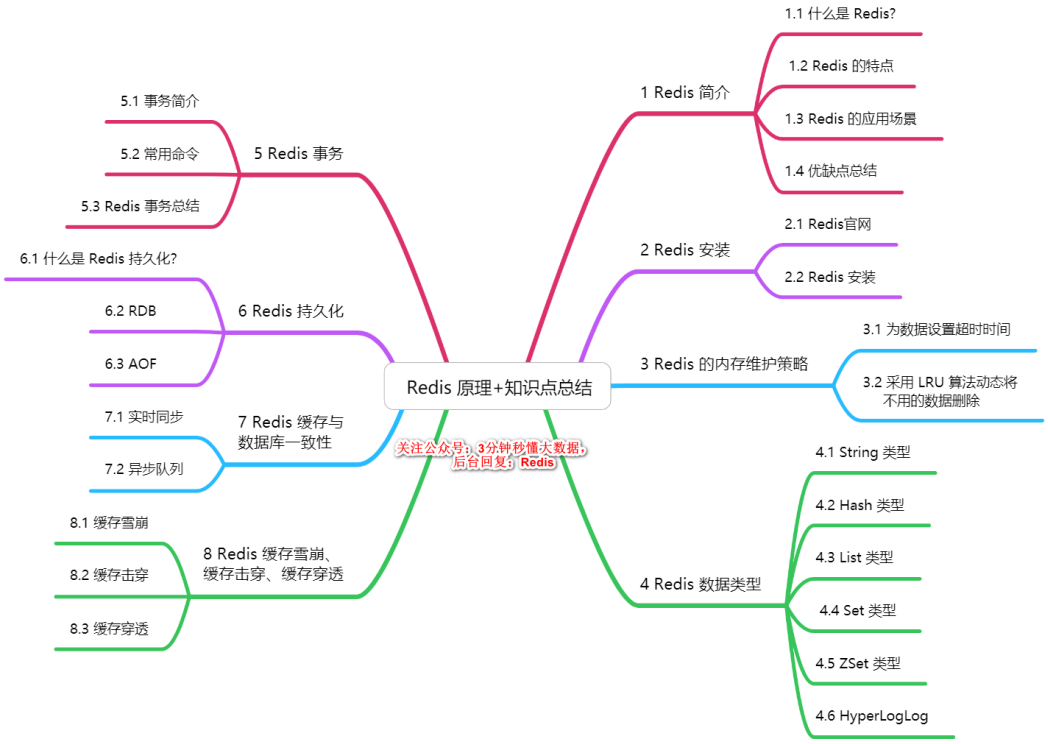
扫一扫上面的二维码图案，加我微信

大家好，我是土哥

今天为大家带来大数据组件中一个经常使用的缓存组件 Redis,本文参考网上资料及个人的理解整理而成，具体内容如下：

本文目录：






获取本文文档，直接在公众号后台回复：Redis ，扫码进群，免费领取

# 一、 Redis 简介

## 1.1 什么是 Redis?

Redis 是一个完全开源、遵守 BSD 协议、简单的、高效的、分布式的、基于内存的 k-v 数据库。

## 1.2 Redis 的特点

- **性能极高** - Redis 读速度 110000 次/s，写的速度是 81000 次/s。
- **丰富的数据类型** - Redis 支持的类型 String， Hash 、List 、Set 及 Ordered Set 数据类型操作。
- **原子性** - Redis 的所有操作都是原子性的，意思就是要么成功，要么失败。单个操作时原子性的。多个操作也支持事务，即原子性，通过 MULTI 和 EXEC 指令包起来。
- **丰富的特性** - Redis 还支持 publis/subscribe，通知，key 过期等等特性。
- **高速读写** ，redis 使用自己实现的分离器，代码量很短，没有使用 lock(MySQL),因此效率非常高。

## 1.3 Redis 的应用场景

**可以作为数据库，缓存热点数据(经常被查询，但是不经常被修改或者删除的数据)和消息中间件等大部分功能。**

Redis 常用的场景示例如下：

**1、缓存**

缓存现在几乎是所有大中型网站都在用的必杀技，合理利用缓存提升网站的访问速度，还能大大降低数据库的访问压力。Redis 提供了键过期功能，也提供了灵活的键淘汰策略，所以，现在 Redis 用在缓存的场合非常多。

**2、排行榜**

Redis 提供的有序集合数据类结构能够实现复杂的排行榜应用。

**3、计数器**

视频网站的播放量，每次浏览 +1，并发量高时如果每次都请求数据库操作无疑有很大挑战和压力。Redis 提供的 incr 命令来实现计数器功能，内存操作，性能非常好，非常适用于这些技术场景。

**4、分布式会话**

相对复杂的系统中，一般都会搭建 Redis 等内存数据库为中心的 session 服务，session 不再由容器管理，而是由 session 服务及内存数据管理。

**5、分布式锁**

在并发高的场合中，可以利用 Redis 的 setnx 功能来编写分布式的锁，如果设置返回 1，说明获取锁成功，否则获取锁失败。

**6、社交网络**

点赞、踩、关注/被关注，共同好友等是社交网站的基本功能，社交网站的访问量通常来说比较大，而且传统的关系数据库不适合这种类型的数据，Redis 提供的哈希，集合等数据结构能很方便的实现这些功能。

**7、最新列表**

Redis 列表结构，LPUSH 可以在列表头部插入一个内容 ID 作为关键字，LTRIM 可以用来限制列表的数量，这样列表永远为 N，无需查询最新的列表，直接根据 ID 去到对应的内容也即可。

**8、消息系统**

消息对队列是网站经常用中间件，如 ActiveMQ，RabbitMQ，Kafaka 等流行的消息队列中间件，主要用于业务解耦，流量削峰及异步处理试试性低的业务。Redis 提供了发布/订阅及阻塞队列功能，能实现一个简单的消息队列系统。另外，这个不能和专业的消息中间件相比。

## 1.4 优缺点总结

**优势**

- 性能极高 - Redis 读速度 110000 次/s，写的速度是 81000 次/s。
- 丰富的数据类型 - Redis 支持的类型 String， Hash 、List 、Set 及 Ordered Set 数据类型操作。
- 原子性 - Redis 的所有操作都是原子性的，意思就是要么成功，要么失败。单个操作时原子性的。多个操作也支持事务，即原子性，通过 MULTI 和 EXEC 指令包起来。
- 丰富的特性 - Redis 还支持 publis/subscribe，通知，key 过期等等特性。
- 高速读写 ，redis 使用自己实现的分离器，代码量很短，没有使用 lock(MySQL),因此效率非常高。

**缺点**

- 持久化。 Redis 直接将数据存储到内存中，要将数据保存到磁盘上，Redis 可以使用两种方式实现持久化过程。定时快照(snapshot)：每隔一段时间将整个数据库写到磁盘上，每次均是写全部数据，代价非常高。第二种方式基于语句追加（aof）：只追踪变化的数据，但是追加的 log 可能过大，同时所有的操作均重新执行一遍，回复速度慢。

- 耗内存 、占用内存过高。

## 二、 Redis 安装部署

### 2.1 Redis 官网

官方网站： https://redis.io/

官方下载： https://redis.io/download 可以根据需要下载不同版本

### 2.2 Redis 安装

**1 安装 redis 二进制安装包**

```
[root@hlink1 lyz]# ll
total 641824
drwxr-xr-x.  7 root root        187 Apr  2  2021 apache-flume-1.9.0-bin
drwxr-xr-x.  4 root root        113 Jun 15  2021 data
drwxr-xr-x.  9 root root        155 Nov  4 22:08 elasticsearch-7.15.2
-rw-r--r--.  1 root root  340810443 Nov 26 16:26 elasticsearch-7.15.2-linux-x86_64.tar.gz
drwxr-xr-x. 10  502 games       156 Jul 23 19:40 flink-1.13.2
-rw-r--r--.  1 root root  313922934 Sep 17 11:09 flink-1.13.2-bin-scala_2.11.tgz
drwxr-xr-x.  2 root root       4096 Jun 15  2021 jar
drwxr-xr-x.  7 root root        118 Mar 31  2021 kafka_2.12-2.2.1
drwxrwxr-x.  7 root root       4096 Jul 22 02:06 redis-6.2.5
-rw-r--r--.  1 root root    2476542 Dec 19 17:11 redis-6.2.6.tar.gz
drwxr-xr-x. 15 2002  2002      4096 Mar 31  2021 zookeeper-3.4.14
[root@hlink1 lyz]#
```

**2 解压/apps 目录下**

```
#解压
[root@hlink1 lyz]# tar -zxvf redis-6.2.6.tar.gz

# 编译
[root@hlink1 redis-6.2.6]# make MALLOC=libc

# install
[root@hlink1 redis-6.2.6]# make PREFIX=/root/lyz/redis-6.2.6 install
```

**3 启动 Redis 服务端，进入到 Redis 的安装目录**

```
cd /root/lyz/redis-6.2.6/bin

# 启动 服务端
[root@hlink1 bin]# ./redis-server
```

```
[root@hlink1 bin]# ./redis-server
32565:C 19 Dec 2021 17:34:37.362 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
32565:C 19 Dec 2021 17:34:37.362 # Redis version=6.2.6, bits=64, commit=00000000, m
32565:C 19 Dec 2021 17:34:37.362 # Warning: no config file specified, using the defa
/redis.conf
32565:M 19 Dec 2021 17:34:37.363 * monotonic clock: POSIX clock_gettime
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 6.2.6 (00000000/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
      (    '      ,       .-`  | `,    )     Running in standalone mode
      |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
      |    `-._   `._    /     _.-'    |     PID: 32565
       `-._    `-._  `-./  _.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |           https://redis.io
       `-._    `-._`-.__.-'_.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
```

**4 启动客户端**

```
[root@hlink1 bin]# ./redis-cli
```

```
[root@hlink1 bin]# ./redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

## 3 Redis 的内存维护策略

redis 作为优秀的中间缓存件，时常会存储大量的数据，即使采取了集群部署来动态扩容，也应该及时的整理内存，维持系统性能。

在 redis 中有两种解决方案

### 3.1 为数据设置超时时间

```
//设置过期时间
expire key time(以秒为单位)--这是最常用的方式
setex(String key, int seconds, String value) --字符串独有的方式
```

1、除了字符串自己独有设置过期时间的方法外，其他方法都需要依靠 expire 方法来设置时间
2、如果没有设置时间，那缓存就是永不过期
3、如果设置了过期时间，之后又想让缓存永不过期没使用 persist key

### 3.2 采用 LRU 算法动态将不用的数据删除

内存管理的一种页面置换算法，对于在内存中但又不用的数据块(内存块)叫做 LRU，操作系统会根据哪些数据属于 LRU 而将其移除内存而腾出空间来加载另外的数据。

1.**volatile-lru**：设定超时时间的数据中，删除最不常使用的数据

2.**allkeys-lru**：查询所有的 key 对最近最不常使用的数据进行删除，这是应用最广泛的策略。

3.**volatile-random**：在已经设定了超时的数据中随机删除。

4.**allkeys-random**：查询所有的 key，之后随机删除。

5.**volatile-ttl**：查询全部设定超时时间的数据，之后排序，将马上要过期的数据进行删除操作。

6.**noeviction**：如果设置为该属性，则不会进行删除操作，如果内存溢出则报错返回。

7.**volatile-lfu**：从所有配置了过期时间的键中驱逐使用频率最少的键

8.**allkeys-lfu**：从所有键中驱逐使用频率最少的键

## 四 Redis 数据类型

Redis 支持的物种数据类型：

1. string(字符串)
2. hash(哈希)
3. list(列表)
4. set(集合)
5. zset（sorted set：有序集合）等

### 4.1 String 类型

String 类型是 Redis 最基本的数据类型，一个键最大能存储 512 MB。

String 数据结构是最简单的 key-value 类型，value 既可以是 string，也可以是数字，是包含很多种类型的特殊类型，

String 类型是二进制安全的。意思是 redis 的 string 可以包含任何数据。

比如序列化的对象进行存储，比如一张图片进行二进制存储，比如一个简单的字符串，数值等等。

## String 命令

1、复制语法：

SET KEY_NAME VALUE ：(说明：多次设置 name 会覆盖)
(Redis SET 命令用于设置给定 key 的值。如果 key 已经存储值，SET 就要写旧值，且无视类型)。

2、命令：

SETNX key1 value：(not exist) 如果 key1 不存在，则设置 并返回 1。
如果 key1 存在，则不设置并返回 0;

(解决分布式锁 方案之一，只有在 key 不存在时设置 key 的值。
setnx (SET if not exits)命令在指定的 key 不存在时，为 key 设置指定的值)。

SETEX key1 10 lx :(expired)设置 key1 的值为 lx ,过期时间为 10 秒，10 秒后 key1 清除（ key 也清除）

SETEX key1 10 lx :(expired) 设置 key1 的值为 lx，过期时间为 10 秒，10 秒后 key1 清除(key 也清除)

SETRANG STRING range value ： 替换字符串

3、取值语法：

GET KEY_NAME ： Redis GET 命令用于获取指定 key 的值。
如果 key 不存在，返回 nil。如果 key 存储的值不是字符串类型，返回一个错误。

GETRANGE key start end ： 用于获取存储在指定 key 中字符串的子字符串。
字符串的截取范围由 start 和 end 两个偏移量来决定(包括 start 和 end 在内)

GETBIT key offset ：对 key 所存储的字符串值，获取指定偏移量上的为(bit)；
GETTEST 语法 ： GETSET KEY_NAME VALUE ： GETSET 命令用于设置指定 key 的值，并返回 key 的旧值。当 key 不存在是，返回 null

STRLEN key :返回 key 所存储的字符串值的长度

4、删除语法：

DEL KEY_NAME ： 删除指定的 key，如果存在，返回数字类型。

5、批量写：MSET K1 V1 K2 V2 ... (一次性写入多个值)

6、批量读：MGET K1 K2 K3

7、GETSET NAME VALUE ： 一次性设置和读取(返回旧值，写上新值)

8、自增/自减：

INCR KEY_Name ： Incr 命令将 key 中存储的数组值增 1。
如果 key 不存在，那么 key 的值会先被初始化为 0，然后在执行 INCR 操作

自增:INCRBY KEY_Name ： 增量值 Incrby 命令将 key 中存储的数字加上指定的增量值
自减：DECR KEY_Name 或 DECYBY KEY_NAME 减值：DECR 命令将 key 中存储的数字减少 1

：(注意这些 key 对应的必须是数字类型字符串，否则会出错。)

字符串拼接：APPEND KEY_NAME VALUE
:Append 命令用于为指定的 key 追加至末尾，如果不存在，为其赋值

字符串长度 ： STRLEN key

```
############################
 setex   (set with expire) #设置过期时间
 setnx   (set if not exist) #不存在设置 在分布式锁中会常常使用!
```

**string 应用场景**

- **1、String 通常用于保存单个字符串或 JSON 字符串数据**
- **2、因 String 是二进制安全的，所以你完全可以把一个图片文件的内容作为字符串来存储**
- **3、计数器(常规 key-value 缓存应用。常规计数：微博数，粉丝数)**

### 4.2 Hash 类型

Hash 类型是 String 类型的 field 和 value 的映射表，或者说是一个 String 集合。hash 特别适合用于存储对象，相比较而言，将一个对象类型存储在 Hash 类型比存储在 String 类型里占用更少的内存空间，并对整个对象的存取。可以看成具有 KEY 和 VALUE 的 MAP 容器，该类型非常适合于存储值对象的信息。

如：uname，upass，age 等。该类型的数据仅占用很少的磁盘空间(相比于 JSON ).

Redis 中每一个 hash 可以存储 2 的 32 次方 -1 键值对(40 多亿)

**Hash 命令**

常用命令

1、赋值语法：

1、 HSET KEY FIELD VALUE ： 为指定的 KEY,设定 FILD/VALUE

2、 HMSET KEY FIELD VALUE [FIELD1，VALUE]... : 同时将多个 field-value(域-值)对设置到哈希表 key 中。

2、取值语法：

HGET KEY FIELD :获取存储在 HASH 中的值，根据 FIELD 得到 VALUE
HMGET KEY FIELD [FIELD1] :获取 key 所有给定字段的值
HGETALL KEY :返回 HASH 表中所有的字段和值

HKEYS KEY : 获取所有哈希表中的字段
HLEN KEY : 获取哈希表中字段的数量

3、删除语法：

HDEL KEY FIELD[FIELD2] :删除一个或多个 HASH 表字段

4、其它语法：

HSETNX KEY FIELD VALUE : 只有在字段 field 不存在时，设置哈希表字段的值

HINCRBY KEY FIELD INCREMENT :为哈希 key 中的指定字段的整数值加上增量 increment。

HINCRBYFLOAT KEY FIELD INCREMENT :为哈希表 key 中的指定字段的浮点数值加上增量 increment

HEXISTS KEY FIELD : 查看哈希表中 key 中，指定的字段是否存在

**Hash 的应用场景：(存储一个用户信息对象数据)**

- **常用于存储一个对象**
- **为什么不用 string 存储一个对象**

  hash 值最接近关系数据库结构的数据类型，可以将数据库一条记录或程序中一个对象转换成 hashmap 存放在 redis 中。

  用户 ID 为查找的 key，存储的 value 用户对象包含姓名，年龄，生日等信息，如果用普通的 key/value 结构来存储，主要有以下 2 种方式：

第一种方式将用户 ID 作为查找 key，把其他信息封装成为一个对象以序列化的方式存储，这种方式增加了序列化/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，用用户 ID+ 对应属性的名称作为唯一标识来取的对应属性的值，虽然省去了序列化开销和并发问题，但是用户 ID 重复存储，如果存在大量这样的数据，内存浪费还是非常可观的。

Redis 提供的 Hash 很好的解决了这个问题，Redis 的 Hash 实际内部存储的 Value 为一个 HashMap。

### 4.3 List 类型

List 类型是一个链表结构的集合，其主要功能有 push、pop、获取元素等。更详细的说，List 类型是一个**双端链表**的结构，我们可以通过相关的操作进行集合的头部或者尾部添加和删除元素，List 的设计是非常简单精巧，既可以为栈，又可以为队列，满足绝大多数的需求。

**常用命令**

1、赋值语法：

```
    LPUSH KEY VALUE1 [VALUE2] :将一个或多个值插入到列表头部（从左侧添加）
RPUSH KEY VALUE1 [VALUE2] ： 在列表中添加一个或多个值（从有侧添加）
    LPUSHX KEY VAKUE :将一个值插入到已存在的列表头部。如果列表不在，操作无效
    RPUSHX KEY VALUE :一个值插入已经在的列表尾部（最右边）。如果列表不在，操作无效
```

2、取值语法：

```
    LLEN KEY   :获取列表长度
LINDEX KEY INDEX    :通过索引获取列表中的元素
    LRANGE KEY START STOP   :获取列表指定范围内的元素
```

描述：返回列表中指定区间的元素，区间以偏移量 START 和 END 指定。

其中 0 表示列表的第一个元素，1 表示列表的第二个元素，以此类推。。。

也可以使用负数下标，以 -1 表示列表的最后一个元素，-2 表示列表的倒数第二个元素，依次类推

start：页大小（页数-1）

stop：（页大小页数）-1

3、删除语法：

LPOP KEY 移除并获取列表的第一个元素（从左侧删除）

RPOP KEY 移除列表的最后一个元素，返回值为移除的元素（从右侧删除）

BLPOP key1 [key2]timeout 移除并获取列表的第一个元素，如果列表没有元素会阻塞列表知道等待超时或发现可弹出元素为止。

4、修改语法：

LSET KEY INDEX VALUE :通过索引设置列表元素的值

LINSERT KEY BEFORE|AFTER WORIL VALUE ：在列表的元素前或者后 插入元素

描述：将值 value 插入到列表 key 当中，位于值 world 之前或之后。

## 高级命令

高级语法：

RPOPLPUSH source destiation ： 移除列表的最后一个元素，并将该元素添加到另外一个列表并返回

示例描述：

RPOPLPUSH a1 a2 ： a1 的最后元素移到 a2 的左侧

RPOPLPUSH a1 a1 ： 循环列表，将最后元素移到最左侧

BRPOPLPUSH source destination timeout :从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它；

如果列表没有元素会阻塞列表知道等待超时或发现可弹出的元素为止。

## List 的应用场景

## 项目应用于：1、对数据量大的集合数据删除；2、任务队列

## 1、对数据量大的集合数据删减

列表数据显示，关注列表，粉丝列表，留言评论等.....分页，热点新闻等

利用 LRANG 还可以很方便的实现分页的功能，在博客系统中，每篇博文的评论也可以存入一个单独的 list 中。

## 2、任务队列

(list 通常用来实现一个消息队列，而且可以确认表先后顺序，不必像 MySQL 那样还需要通过 ORDER BY 来进行排序)

任务队列介绍(生产者和消费者模式：)

在处理 web 客户端发送的命令请求时，某些操作的执行时间可能会比我们预期的更长一些，通过将待执行任

务的相关信息放入队列里面，并在之后队列进行处理，用户可以推迟执行那些需要一段时间才能完成的操作，

这种将工作交个任务处理器来执行的做法被称为任务队列（task queue）。

```
RPOPLPUSH source destination
```

移除列表的最后一个元素，并将该元素添加到另一个列表并返回

### 4.4 Set 类型

Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。Redis 中集合是通过哈希表实现的，set 是通过 hashtable 实现的

集合中最大的成员数为 2^32 -1,类似于 JAVA 中的 Hashtable 集合。

**命令**

1、复制语法：

```
SADD KEY member1 [member2]:向集合添加一个或多个成员
```

2、取值语法：

```
SCARD KEY :获取集合的成员数
SMEMBERS KEY : 返回集合中的所有成员
SISMEMBER KEY MEMBER :判断 member 元素是否是集合 key 的成员(开发中：验证是否存在判断)
SRANDMEMBER KEY [COUNT] :返回集合中一个或对个随机数
```

3、删除语法：

```
SREM key member1 [member2] : 移除集合中一个或多个成员
SPOP key [count] : 移除并返回集合中的一个随机元素
SMOVE source destination member :将 member 元素从 Source 集合移动到 destination 集合中
```

4、差集语言：

```
SDIFF key1 [key2] :返回给定所有集合的差集
SDIFFSTORE destination key1 [key2] :返回给定所有集合的茶几并存储在 destination 中
```

5、交集语言：

```
SUNION key1 [key2] : 返回所有给定集合的并集
SUNIONSTORE destination key1 [key2] :所有给定集合的并集存储在 destinatiion 集合中
```

### 4.5 ZSet 类型

有序集合(sorted set)

**简介**

1、Redis 有序集合和集合一样也是 string 类型元素的集合，且不允许重复的成员。

2、不同的是每个元素都会关联一个 double 类型的分数。redis 正是通过分数来为集合中的成员进行从小到大的排序。

3、有序集合的成员是唯一的，但分数（score）却可以重复。

4、集合是通过哈希表实现的。集合中最大的成员数为 2^32 -1。Redis 的 ZSet 是有序，且不重复。

（很多时候，我们都将 redis 中的有序结合叫做 zsets，这是因为在 redis 中，有序集合相关的操作指令都是以 z 开头的）

## 命令

1、复制语法：

ZADD KEY score1 member1 【score2 member2】 ：向有序集合添加一个或多个成员，或者更新已经存在成员的分数

2、取值语法：

ZCARD key ：获取有序结合的成员数

ZCOUNT key min max :计算在有序结合中指定区间分数的成员数

```
######################################################################
127.0.0.1:6379> ZADD kim 1 tian
(integer) 0
127.0.0.1:6379> zadd kim 2 yuan 3 xing
(integer) 2
127.0.0.1:6379> zcount kim 1 2
(integer) 2
127.0.0.1:6379>
######################################################################
```

ZRANK key member :返回有序集合中指定成员的所有

ZRANGE KEY START STOP [WITHSCORES]:通过索引区间返回有序集合成指定区间内的成员(低到高)

ZRANGEBYSCORE KEY MIN MAX [WITHSCORES] [LIMIT] :通过分数返回有序集合指定区间内的成员

ZREVRANGE KEY START STOP [WITHSCORES] :返回有序集中是定区间内的成员，通过索引，分数从高到底

ZREVERANGEBYSCORE KEY MAX MIN [WITHSCORES] :返回有序集中指定分数区间的成员，分数从高到低排序

删除语法：

DEL KEY ： 移除集合

ZREM key member [member...] 移除有序集合中的一个或多个成员

ZREMRANGEBYSCORE KEY MIN MAX :移除有序集合中给定的分数区间的所有成员。

ZREMRANGEBYSCORE KEY MIN MAX :移除有序集合中给定的分数区间的所有成员。

ZINCRBY KEY INCREMENT MEMBER :增加 member 元素的分数 increment，返回值是更改后的分数

**HyperLogLog**

**常用命令**

PFADD key element [element ...] : 添加指定元素到 HyperLoglog 中

PFCOUNT KEY [key ...] :返回给定 HyperLogLog 的基数估算值

PFMERGE destkey sourcekey [sourcekey ...] :将过个 HyperLogLog 合并为一个 HyperLoglog

**应用场景**

基数不大，数据量不大就用不上，会有点大材小用浪费空间

有局限性，就是指能统计基数数量，而没办法去知道具体的内容是什么

统计注册 IP 数
统计每日访问 IP 数
统计页面实时 UV 数
统计在线用户数
统计用户每天搜索不同词条的个数
统计真实文章阅读数

# 五 Redis 事务

Redis 事务可以一次执行多个命令，（按顺序地串行化执行，执行中不会被其他命令插入，不许加塞）

## 5.1 事务简介

Redis 事务可以一次指定多个命令（允许在一个单独的步骤中执行一组命令），并且带有以下两个重要的保证：

批量操作在发送 EXEC 命令前被放入队列缓存。

收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余命令依然被执行。

在事务执行过程中，其他客户端提交的命令请求不会插入到事务执行命令列中。

1. **Redis 会将一个事务中的所有命令序列化，然后按顺序执行**
2. **执行中不会被其它命令插入，不许出现加赛行为**

## 5.2 常用命令

DISCARD:

取消事务，放弃执行事务块内的所有命令。

EXEC :

执行所有事务块内的命令。

MULTI:

标记一个事务块的开始。

UNWATCH:

取消 watch 命令对所有 key 的监视。

WATCH KEY [KEY ...]

:监视一个(或多个)key，如果在事务执行之前这个(或这些)key 被其他命令所改动，那么事务将被打断。

一个事务从开始到执行会经历以下三个阶段：

1、开始事务。

2、命令入队。

3、执行事务。

**示例 1 MULTI EXEC**

转账功能，A 向 B 转账 50 元

一个事务的例子，它先以 MULTI 开始一个事务，然后将多个命令入队到事务中，最后由 EXEC 命令触发事务

```
Warning: Using a password with '-a' or '-u' option on the command line interf
127.0.0.1:6379> set account:a 100
OK
127.0.0.1:6379> set account:b 50
OK
127.0.0.1:6379> multi
OK
127.0.0.1:6379> get account:a
QUEUED
127.0.0.1:6379> get account:b
QUEUED
127.0.0.1:6379>
127.0.0.1:6379> incrby account:a 20
QUEUED
127.0.0.1:6379> decrby account:b 20
QUEUED
127.0.0.1:6379> exec
1) "100"
2) "50"
3) (integer) 120
4) (integer) 30
127.0.0.1:6379>
```

1. **输入 Multi 命令开始，输入的命令都会一次进入命令队列中，但不会执行**

2. **知道输入 Exce 后，Redis 会将之前的命令队列中的命令一次执行。**

**示例 2 DISCARD 放弃队列运行**


1. **输入 MULTI 命令，输入的命令都会依次进入命令队列中，但不会执行。**
2. **直到输入 Exec 后，Redis 会将之前的命令队列中的命令依次执行。**
3. **命令队列的过程中可以使用命令 DISCARD 来放弃队列运行。**

**示例 3 事务的错误处理**

事务的错误处理：

队列中的某个命令出现了 报告错误，执行是整个的所有队列都会被取消。

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set a kkk
QUEUED
127.0.0.1:6379> fsfsf
(error) ERR unknown command `fsfsf`, with args beginning with:
127.0.0.1:6379> incr a
QUEUED
127.0.0.1:6379>
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
127.0.0.1:6379>
```

**由于之前的错误，事务执行失败**

## 5.3 Redis 事务总结

Redis 事务本质：一组命令的集合！一个事务中的所有命令都会被序列化，在事务执行过程中，会按照顺序执行！一次性，顺序性，排他性！执行一些列的命令！

**Redis 事务没有隔离级别的概念！**

所有的命令在事务中，并没有直接被执行！只有发起执行命令的时候才会执行！Exec

**Redis 单条命令保存原子性，但是事务不保证原子性！**

Redis 事务其实是支持原子性的！即使 Redis 不支持事务回滚机制，但是它会检查每一个事务中的命令是否错误。

# 六 Redis 持久化

## 6.1 什么是 Redis 持久化？

持久化就是把内存的数据写到磁盘中去，防止服务宕机内存数据丢失。

Redis 提供了两种持久化方式：**RDB(默认)和 AOF**

**简介**

数据存放于：

**内存：高效，断电（关机）内存数据会丢失**

**硬盘：读写速度慢于内存，断电数据不会丢失**

**Redis 持久化存储支持两种方式：RDB 和 AOF。RDB 一定时间取存储文件，AOF 默认每秒去存储历史命令，**

**Redis 是支持持久化的内存数据库，也就是说 redis 需要经常将内存中的数据同步到硬盘来保证持久化。**

## 6.2 RDB

RDB 是 Redis DataBase 缩写

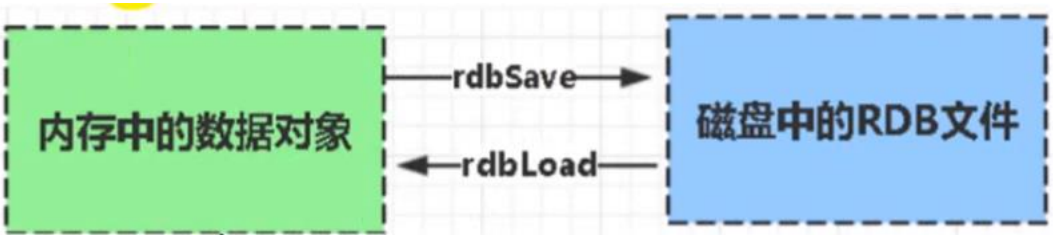
Redis 是内存数据库，如果不将内存中的数据库状态保存到磁盘中，那么一旦服务器进程退出，服务器中的数据库的状态也会消失。造成数据的丢失，所以 redis 提供了持久化的功能。

在指定的时间间隔内将内存中的数据集快照写入磁盘，也就是所说的 snapshot 快照，它恢复是将磁盘中的数据直接读到内存里。

Redis 会单独创建（fock）一个子进程来进行持久化，会先将数据写入到一个临时文件中，待持久化过程都结束了，再用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何 IO 操作的。这就确保了极高的性能。如果需要进行大规模的数据的恢复，且对于数据恢复的完整性不是非常敏感，那 RDB 方式要比 AOF 方式更加的高效。RDB 的缺点是最后一次持久化的数据可能丢失。

功能核心函数 rdbSave（生成 RDB 文件）和 rdbLoad（从文件加载内存）两个函数

- **rdbSave：生成 RDB 文件**
- **rdbLoad：从文件夹杂内存**




RDB : 是 redis 默认的持久化机制

快照是默认的持久化方式。这种方式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**。

**优点：**

- 快照保存数据极快，还原数据极快
- 适用于灾难备份

**缺点：**

- 小内存机器不适合使用，RDB 机制符合要求就会照快照

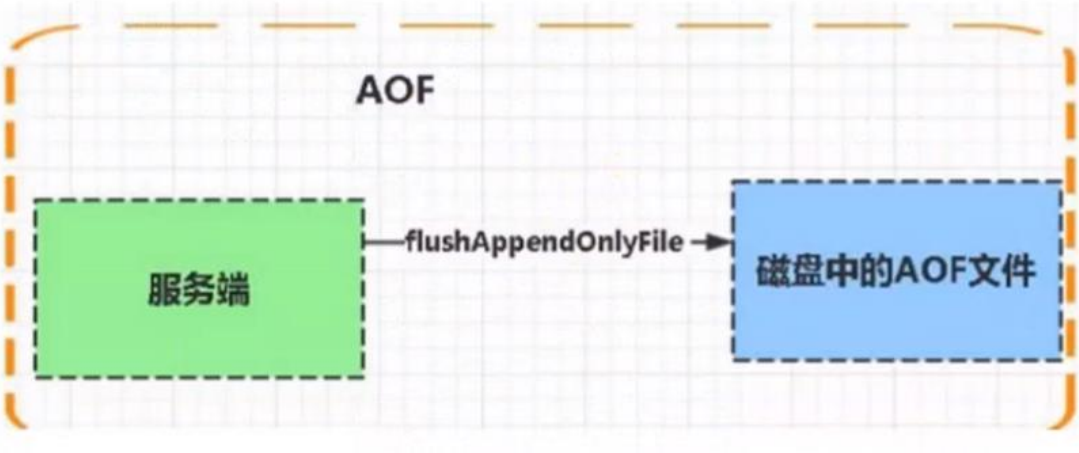
**快照条件：**

```
1、服务器正常关闭：./bin/redis-cli shutdown
2、key 满足一定条件，会进行快照
 vim redis.config 搜索 save
 /save
save   900  1      //每秒 900 秒（15 分钟）至少 1 个 key 发生变化，产生快照
save   300  10     //每 300 秒  （5 分钟）至少 10 个 key 发生变化，产生快照
save   60   10000  //每 60 秒（1 分钟）至少 10000 个 key 发生变化，产生快照
```

### 6.3 AOF

由于快照方式是在一定间隔时间做一次的，所以如果 redis 意外 down 掉的话，就会丢失最后一个快照后的所有修改。如果应用要求不能丢失任何修改的话，可以采用 aof 持久化方式。

Append-only file：aof 比 rdb 有更好的持久化性，是由于在使用 aof 持久化方式时，redis 会将每一个收到的命令都通过 write 函数追加到文件中（默认是 appendonly.aof)。当 redis 重启是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。




每当执行服务器（定时）任务或者函数时， flushAppendOnlyFile 函数都会被调用，这个函数执行以下两个工作 aof 写入保存：

**WRITE：根据条件，将 aof_buf 中的缓存写入到 AOF 文件。**

**SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。**

**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 //收到写命令就立即写入磁盘，最慢，但是保证完全的持久化
- appendfysnceverysec //每秒钟写入磁盘一次，在性能和持久化方面做了很好的折中
- # appendfysnc no //完全依赖 os，性能孔，持久化没保证

**产生的问题：**

aof 的方式也同时带来了另一个问题。持久化文件会变的越来越大。例如我们调用 incr test 命令 100 次，文件中必须保存全部的 1000 条命令，其实有 99 条都是多余的。

# 七 Redis 缓存与数据库一致性

## 7.1 实时同步

对缓存要求比较高的，应采用实时同步方案，即查询缓存查询不到再从 DB 查询，保存到缓存；更新缓存时，先更新数据库，再将缓存的设置过期（建议不要去更新缓存内容，直接设置缓存过期）。

@Cacheable：查询时使用，注意 Long 类型需要转换为 String 类型，否则会抛异常

@CachePut：更新时使用，使用此注解，一定会从 DB 上查询数据

@CacheEvict：删除时使用；

@Caching ：组合用法

## 7.2 异步队列

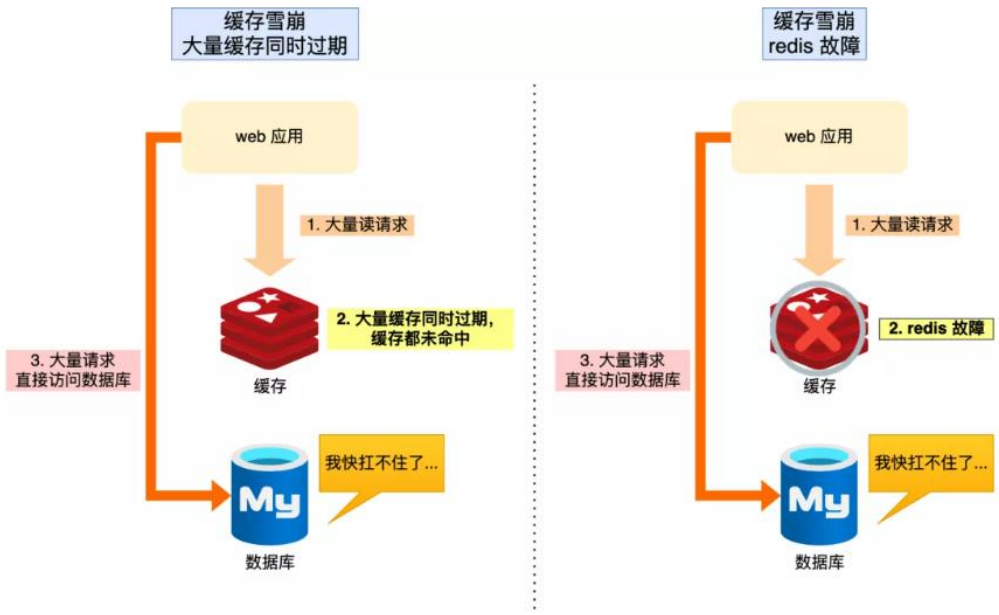
对于并发程度高的，可采用异步队列的方式同步，可采用 kafka 等消息中间件处理消息生产和消费。

# 八 Redis 缓存雪崩、缓存击穿、缓存穿透

## 8.1 缓存雪崩

通常我们为了保证缓存中的数据与数据库中的数据一致性，会给 Redis 里的数据设置过期时间，当缓存数据过期后，用户访问的数据如果不在缓存里，业务系统需要重新生成缓存，因此就会访问数据库，并将数据更新到 Redis 里，这样后续请求都可以直接命中缓存。

那么，当**大量缓存数据**在同一时间过期（失效）或者 Redis 故障宕机时，如果此时有大量的用户请求，都无法在 Redis 中处理，于是全部请求都直接访问数据库，从而导致数据库的压力骤增，严重的会造成数据库宕机，从而形成一系列连锁反应，造成整个系统崩溃，这就是缓存雪崩的问题。




**解决办法：**

用锁/分布式锁或者队列串行访问

缓存失效时间均匀分布

> 如果缓存集中在一端时间内失效，发生大量的缓存穿透，所有的查询都落在数据库上，造成了缓存雪崩。
>
> 这个没有完美解决办法，但是可以分析用户的行为，尽量让失效时间点均匀分布。大所属系统设计者考虑用加锁或者队列的方式保证缓存的单线程写，从而避免失效时大量的并发请求落到底层存储系统上。

**1. 加锁排队。限流---限流算法**

(1) 在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个 key 只允许一个线程查询数据和写缓存，其他线程等待。

简单地来说，就是在缓存失效的时候（判断拿出来的值为空），不是立即去 load db，而是先使用缓存工具的某些带成功操作返回值的操作（比如 Redisde SETNX 或者 Memcache 的 ADD）去 set 一个 mutex key，当操作返回成功是，在进行 koad db 的操作应设缓存；否则，就重试整个 get 缓存的方法。

(2) SETNX ,是【SET IF NOT EXISTS]的缩写，也就是只有不存在的时候才设置，可以利用它来实现锁的效果。

**2. 数据预热**

可以通过缓存 reload 机制，预选去更新缓存，再即将发生大并发访问前手动触发加载缓存不同的 key，设置不同的过期时间，让缓存失效的时间点尽量均匀。

## 8.2 缓存击穿

我们的业务通常会有几个数据会被频繁地访问，比如秒杀活动，这类被频地访问的数据被称为热点数据。

如果缓存中的某个热点数据过期了，此时大量的请求访问了该热点数据，就无法从缓存中读取，直接访问数据库，数据库很容易就被高并发的请求冲垮，这就是缓存击穿的问题。

缓存击穿跟缓存雪崩很相似，你可以认为缓存击穿是缓存雪崩的一个子集。

**解决方案：**

（1）**互斥锁方案**，保证同一时间只有一个业务线程更新缓存，未能获取互斥锁的请求，要么等待锁释放后重新读取缓存，要么就返回空值或者默认值。

（2）**不给热点数据设置过期时间**，由后台异步更新缓存，或者在热点数据准备要过期前，提前通知后台线程更新缓存以及重新设置过期时间；

## 8.3 缓存穿透

当发生缓存雪崩或击穿时，数据库中还是保存了应用要访问的数据，一旦缓存恢复相对应的数据，就可以减轻数据库的压力，而缓存穿透就不一样了。

当用户访问的数据，既不在缓存中，也不在数据库中，导致请求在访问缓存时，发现缓存缺失，再去访问数据库时，发现数据库中也没有要访问的数据，没办法构建缓存数据，来服务后续的请求。那么当有大量这样的请求到来时，数据库的压力骤增，这就是**缓存穿透**的问题。

**解决方案**

**（1）非法请求的限制**

当有大量恶意请求访问不存在的数据的时候，也会发生缓存穿透，因此在 API 入口处我们要判断求请求参数是否合理，请求参数是否含有非法值、请求字段是否存在，如果判断出是恶意请求就直接返回错误，避免进一步访问缓存和数据库。

**（2）缓存空值或者默认值**

当我们线上业务发现缓存穿透的现象时，可以针对查询的数据，在缓存中设置一个空值或者默认值，这样后续请求就可以从缓存中读取到空值或者默认值，返回给应用，而不会继续查询数据库。

### （3）使用布隆过滤器快速判断数据是否存在，避免通过查询数据库来判断数据是否存在

我们可以在写入数据库数据时，使用布隆过滤器做个标记，然后在用户请求到来时，业务线程确认缓存失效后，可以通过查询布隆过滤器快速判断数据是否存在，如果不存在，就不用通过查询数据库来判断数据是否存在。

即使发生了缓存穿透，大量请求只会查询 Redis 和布隆过滤器，而不会查询数据库，保证了数据库能正常运行，Redis 自身也是支持布隆过滤器的。

布隆过滤器是一种数据结构，对所有可能查询的参数以 hash 形式存储，在控制层先进行校验，不符合则丢弃，从而避免了对底层存储系统的查询压力；