# redis官网

https://redis.io/

# 下载地址

## linux版本

http://download.redis.io/releases/

## windows版本

https://github.com/MicrosoftArchive/redis/releases

https://github.com/zkteco-home/redis-windows/releases（使用这个地址）

# redis6.0版本新特性

## 1、模块化架构

Redis 6.0 引入了模块化架构，使用户可以通过加载插件来扩展 Redis 的功能。这允许开发者通过编写自定义模块来添加新的数据结构、命令和功能，以满足特定需求。

## 2、原生客户端执行

Redis 6.0 引入了原生客户端执行（Client-side caching）功能，通过将部分计算移动到客户端，减轻了服务器的负担并提高了性能。该特性可用于对数据进行聚合、筛选、排序等计算。

## 3、RDB持久化改进

Redis 6.0 对 RDB 持久化机制进行了优化和改进。现在，Redis 支持非阻塞的 RDB 持久化，可以在后台进行持久化操作，而不会对正常的 Redis 响应时间产生影响。此外新的 RDB 格式提供了更好的空间利用率。

## 4、ACL（Access Control List）改进

Redis 6.0 引入了 ACL 功能，允许对不同的客户端、用户或应用程序进行精细化的访问控制。通过 ACL，你可以设置不同的权限规则，限定客户端可以执行的命令和访问的数据。

功能：

1、支持基于用户名的权限控制，可以创建多个用户，每个用户有不同的权限。

2、支持命令权限管理，能够配置哪些命令对特定用户可用，哪些命令不可用。

3、可以限制用户访问某些特定的数据库或键。

4、提供了 ACL SETUSER 和 ACL GETUSER 等命令，用于管理和查询用户权限。

示例：



## 5、Stream 数据结构改进

## 6、布隆过滤器（Bloom Filter）

## 7、Redis 协议的改进

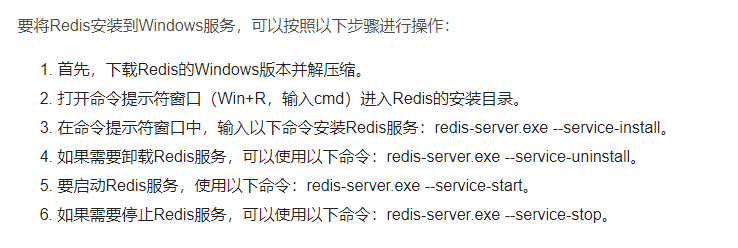
# 在windows上安装redis

## 安装包方式安装

## zip文件安装方式

https://blog.csdn.net/weixin\_44893902/article/details/123087435

## redis安装为windows服务



redis-server.exe --service-install

## 怎么给redis设置密码

使用命令的方式

config set requirepass 密码 #设置redis密码

# 在Linux上安装redis

## 1、下载reids

我们访问下面的地址：

http://download.redis.io/releases/

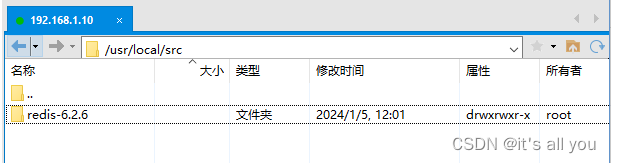
下载你想要的版本

## 2、安装redis依赖

yum install -y gcc tcl

## 3、将redis上传到服务器的指定目录中

这里我们上传到了/usr/local/src目录



## 4、解压缩

tar -xzf redis-6.2.6.tar.gz

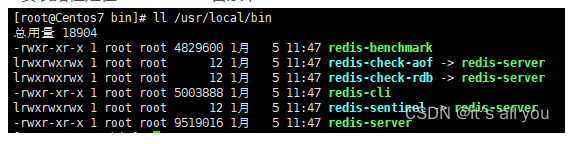
## 5、进入到redis目录中

cd redis-6.2.6

## 6、安装redis

make && make install

安装路径在/usr/local/bin目录下

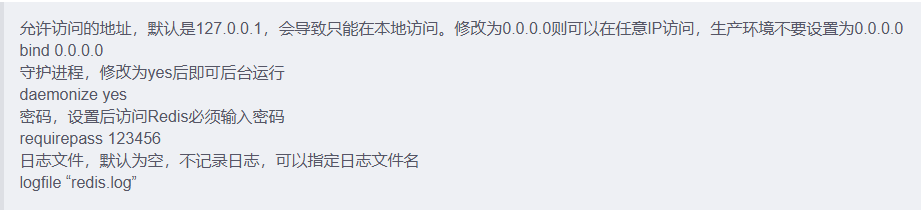


## 7、以指定配置方式启动

如果要让Redis以后台方式启动，则必须修改Redis配置文件，就在我们之前解压的redis安装包下（/usr/local/src/redis-6.2.6），名字叫redis.conf。先将这个配置文件备份一份：

cp redis.conf redis.conf.bck

然后修改redis.conf文件中的一些配置



bind 0.0.0.0

daemonize yes

requirepass AXB37nVDNQrvWK@jPpfJERxG8

logfile ""

## 8、启动redis

### 1、进入redis安装目录

cd /usr/local/src/redis-6.2.6

### 2、启动

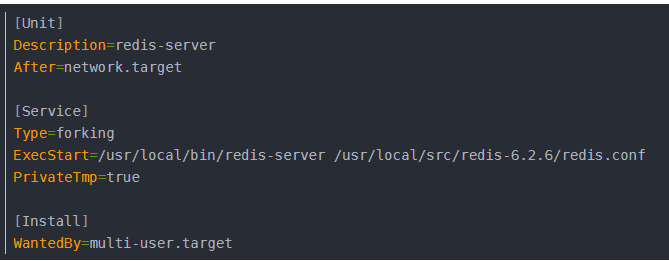
redis-server redis.conf

## 9、设置开机自启动

### （1）新建一个系统服务文件

vi /etc/systemd/system/redis.service

### （2）设置内容



### （3）重载系统服务

systemctl daemon-reload

### （4）使用命令



#### 启动

systemctl start redis

#### 停止

systemctl stop redis

#### 重启

systemctl stop redis

#### 查看状态

systemctl status redis

### （5）设置redis开机自启

systemctl enable redis

## 查找redis的安装路径

### ****使用which命令****

which redis-server

这会输出 redis-server 可执行文件的路径，通常情况下就是 Redis 的安装路径。

# 基本概念

Redis的全称是: Remote Dictionary.Server，本质上是一个 Key-Value 类型的内存数据库，很像 Memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存，因为是纯内存操作,Redis 的性能非常出色，每秒可以处理超过 10 万次读写提作，是已知性能最快的 Key-Value DB。Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构，因此 Redis 可以用来实现很多有用的功能。比方说用他的 List 来做 FIFO 双向表,实现一个轻量级的高性能消息队列服务，用他的 Set 可以做高性能的 tag 系统等等。另外 Redis也可以对存入的 Key-Value 设置expire 时间,因此也可以被当作一功能加强版的 Memcached 来用。

Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

# 优点

## 1、性能极高

Redis能读的速度是110000次/s,写的速度是81000次/s。

## 2、丰富的数据类型

Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

## 3、原子性

Redis的所有操作都是原子性的，同时Redis还支持对几个操作合并后的原子性执行。（事务）

## 4、丰富的特性

Redis还支持 publish/subscribe, 通知, key 过期等特性。

# redis配置文件

## protected-mode

protected-mode 是 Redis 配置文件中的一个属性，它是用来控制 Redis 服务器是否启用保护模式（protected mode）的。保护模式是 Redis 为了增强安全性而设置的一种机制，当开启保护模式时，Redis 只能在本地环境中（即 127.0.0.1 或者 localhost）运行，并且拒绝来自外部网络的连接请求，这样可以避免 Redis 服务器受到未经授权的访问或者攻击。

当 protected-mode 的值为 yes 时，表示 Redis 启用了保护模式；当值为 no 时，表示 Redis 关闭了保护模式。默认情况下，Redis 是开启了保护模式的。除了 yes 和 no 之外，protected-mode 还可以设置为 auth-pass 或者 auth-ip-address，代表使用密码验证或者 IP 地址验证的方式来开启保护模式。这两种方式需要在 Redis 配置文件中分别设置 requirepass 和 bind 属性。需要注意的是，即使开启了保护模式，也不能完全保证 Redis 服务器是安全的。因此在实际生产环境中，还需要采取其他措施来保障 Redis 服务器的安全。例如，设置复杂的密码、限制 Redis 的命令和操作、限制访问地址等等。

# redis单线程的理解

目前所说的Redis单线程，指的是"其网络IO和键值对读写是由一个线程完成的"，也就是说Redis中只有网络请求模块和数据操作模块是单线程的，而其他的如持久化存储模块、集群支撑模块等是多线程的，所以说Redis中并不是没有多线程模型的，早在Redis 4.0的时候就已经针对部分命令做了多线程化。

# UNIX时间戳

## 基本概念

UNIX时间戳是指自1970年1月1日00:00:00以来经过的秒数，用于表示时间和日期的一种方式。它是计算机系统中常用的一种时间表示方法，特别在UNIX/Linux操作系统和许多编程语言中被广泛采用。UNIX时间戳的基准点是协调世界时（UTC）的1970年1月1日零点。每过一秒，UNIX时间戳就会增加一单位。因此，UNIX时间戳可以简单地表示某个时间点与基准点之间的相对时间差。

## 特点

### 统一性

不受时区的影响，全球范围内的UNIX时间戳都是一致的。

### 递增性

每秒递增一单位，不考虑闰秒等因素。

### 方便计算

UNIX时间戳是一个整数，便于计算和比较。

### 可读性差

UNIX时间戳通常以十进制或十六进制表示，对人类来说不太直观。

根据UNIX时间戳，可以将时间戳转换为对应的日期和时间。各种编程语言和工具提供了相应的函数或库来进行时间戳和日期之间的转换。通过将时间戳转换为日期时间格式，我们可以理解该时间戳所代表的具体时刻。

# TTL

## 基本概念

TTL（Time To Live）是用于设置键（Key）的生存时间的概念。每个键都可以关联一个 TTL 值，用于指定键在多长时间后自动过期并被删除。使用 TTL 功能可以控制 Redis 数据中的键在一段时间后自动过期，这对于缓存、限时数据和临时状态非常有用。当 TTL 值为正数时，表示在指定的秒数后键会自动过期。当键过期时，它将会从 Redis 中被删除，无法再被访问到。

使用 TTL 命令可以获取键的剩余生存时间。返回值有以下几种情况：

（1）当键不存在或已过期时，返回 -2。

（2）当键存在且没有设置 TTL 时，返回 -1（永远不会过期）。

（3）当键存在且有设置 TTL 时，返回键的剩余生存时间（以秒为单位）。

## 默认时间单位

秒

## 命令格式

TTL key

除了直接使用 TTL 命令，还可以使用其他的命令来设置键的生存时间

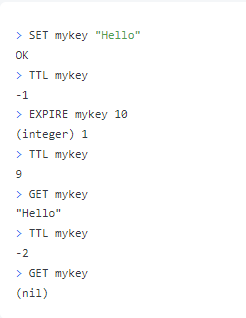
EXPIRE key seconds：设置键的过期时间，以秒为单位。

PEXPIRE key milliseconds：设置键的过期时间，以毫秒为单位。

EXPIREAT key timestamp：设置键的过期时间，使用 UNIX 时间戳表示过期时间。

PEXPIREAT key timestamp：设置键的过期时间，使用毫秒级的 UNIX 时间戳表示过期时间。

## 示例



需要注意的是，Redis 的 TTL 是以秒为单位的，因此在设置 TTL 时需要注意单位的转换。

# Redis数据类型

## 基本说明

Redis 中 **键（Key）本身始终是字符串类型**（即 String 类型），而**值（Value）** 可以支持多种数据类型。

## String（字符串）

### 基本概念

string是redis 最基本的类型，你可以理解成与 Memcached 一模一样的类型，一个 key 对应一个 value。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jpg图片或者序列化的对象。string 类型是 Redis 最基本的数据类型，string 类型的值最大能存储 512MB。

### 底层数据结构

简单动态字符串（SDS）

### Redis为什么构建自己的简单动态字符串 SDS

Redis 通过自构建SDS，而不使用 C 字符串，不仅解决了 C 字符串存在的缓冲区溢出问题，同时还通过减少因修改字符串导致的频繁分配内存空间和获取长度导致的性能消耗。再者由于 SDS 不再使用空字符 '\0' 标志字符串的结尾，使得Redis 不但可以存储文本数据，还可以保存任意格式二进制数据。

1、获取字符串长度的复杂度

2、杜绝缓冲区溢出

3、减少修改字符串时带来的内存分配次数

4、二进制安全

### 应用场景

常规key-value缓存应用

常规计数: 微博数, 粉丝数。

分布式锁

实现方式：String在redis内部存储默认就是一个字符串，被redisObject所引用，当遇到incr,decr等操作时会转成数值型进行计算，此时redisObject的encoding字段为int。

## Hash（哈希）

### 基本概念

Redis hash 是一个键值(key=>value)对集合。Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象。每个 hash 可以存储 232 -1 键值对（40多亿）。

### 底层数据结构

ZipList（压缩列表）、Hashtable

### 应用场景

存放结构化的数据，比如用户信息。

## List（列表）

### 基本概念

Redis 列表是简单的字符串列表，按照插入顺序排序，列表中的元素可以重复，你可以添加一个元素到列表的头部（左边）或者尾部（右边）。列表最多可存储 232 - 1 元素 (4294967295, 每个列表可存储40多亿)。有序可重复的。

### 底层数据结构

ziplist、linkedlist

### 应用场景

1、各种列表，比如twitter的关注列表、粉丝列表等，最新消息排行、每篇文章的评论等也可以用Redis的list结构来实现。

2、消息队列

3、利用LRANGE可以很方便的实现list内容分页的功能。

4、取最新N个数据的操作

## Set（集合）

Redis 的 Set 是 string 类型的无序集合。集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。集合中最大的成员数为 232 - 1(4294967295, 每个集合可存储40多亿个成员)。无序不能重复。

### 底层数据结构

Hashtable、IntSet（整数集）

### 应用场景

1、某些需要去重的列表，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。

2、可以存储一些集合性的数据

3、想要知道某些特定的注册用户或IP地址，他们到底有多少访问了某个页面，可以这样实现：SADD page:day1:<page\_id> <user\_id> 。想知道特定用户的数量，使用SCARD page:day1:<page\_id>。 需要测试某个特定用户是否访问了这个页面？SISMEMBER page:day1:<page\_id>。

## Zset（有序集合）

### 基本概念

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。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。zset的成员是唯一的,但分数(score)却可以重复。有序不能重复。

### 底层数据结构

zipList、zskipList（跳表）

### 应用场景

1、存放一个有序的并且不重复的集合列表

2、可以做带权重的队列

3、排行榜相关：ZADD leaderboard <score> <username> 。 得到前100名高分用户很简单：ZREVRANGE leaderboard 0 99。用户的全球排名也相似，只需要执行：ZRANK leaderboard <username>。

4、新闻按照用户投票和时间排序

## bitmap

### 基本概念

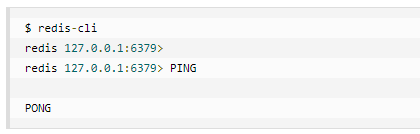
### 应用场景

# redis命令

## 本地连接命令

redis-cli

启动 redis 服务器，打开终端并输入命令 redis-cli，该命令会连接本地的 redis 服务。

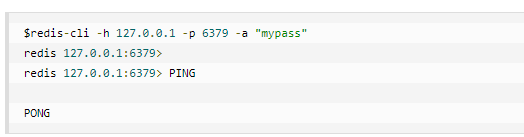


在以上实例中我们连接到本地的 redis 服务并执行 PING 命令，该命令用于检测 redis 服务是否启动。

## 远程服务器上连接redis的命令

redis-cli -h host -p port -a password

以下实例演示了如何连接到主机为 127.0.0.1，端口为 6379 ，密码为 mypass 的 redis 服务上。



## 启动redis服务的命令

redis-server --service-start

redis-server.exe redis.windows.conf（已指定的配置文件启动）

说明：在windows系统下面可以使用

## 停止redis服务的命令

redis-server --service-stop

redis-cli shutdown

说明：在windows系统下面可以使用

## 查看redis的版本

info

文本

描述已自动生成

## redis中命令返回1和0的含义

返回值1表示操作成功，命令执行完成并成功地进行了相应的操作。

返回值0表示操作失败，命令执行未能完成或没有进行相应的操作。

## nil

nil 是一个特殊的值，表示 **空值** 或 **不存在**。它类似于其他编程语言中的 null 或 None，用于表示某个键不存在、某个操作未返回有效结果或某个字段为空。

## 键有关的命令

### del key

该命令用于在 key 存在时删除 key。

### dump key

序列化给定key，并返回被序列化的值。

### exists key

检查给定的key是否存在，如果指定的key不存在，则返回0。

### expire key seconds

为给定的key设置过期时间，以秒计算。返回值是1表示成功，0表示失败。

### expireat key timestamp

EXPIREAT 的作用和 EXPIRE 类似，都用于为 key 设置过期时间。不同在于 EXPIREAT 命令接受的时间参数是 UNIX 时间戳(unix timestamp)。

### keys pattern

查找所有符合给定模式( pattern)的key。

### move key db

将当前数据库的 key 移动到给定的数据库db当中。

### persist key

移除key的过期时间，key将持久保持。

### ttl key

查看某个key的过期时间

## 字符串命令

### set key value

设置指定key的值

### get key

获取指定 key 的值

### getrange key start end

返回 key 中字符串值的子字符

### getset key value

将给定 key 的值设为 value ，并返回 key 的旧值(old value)。

### mget key1[key2…]

获取所有(一个或多个)给定 key 的值

### setex key seconds value

将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。

### setnx key value（重点）

只有在 key 不存在时设置 key 的值。

#### setnx与set的区别

1、SET 命令会覆盖已存在键的值。

2、SETNX 命令只在键不存在时设置键值对，在键已经存在时不进行任何操作。

3、SET 命令还支持设置键的过期时间和覆盖已存在的键值对，而 SETNX 命令则没有这些功能。

### strlen key

返回 key 所储存的字符串值的长度。

### mset key value[key value..]

同时设置一个或多个 key-value 对。

### msetnx key value[key value…]

同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。

### incr key

将 key 中储存的数字值增一，并返回值。

### incrby key increment

将 key 所储存的值加上给定的增量值（increment）。

### incrbyfloat key increment

将 key 所储存的值加上给定的浮点增量值（increment）

### decr key

将 key 中储存的数字值减一，并返回值。

### decrby key decrement

key 所储存的值减去给定的减量值（decrement）

### append key value

如果 key 已经存在并且是一个字符串， APPEND 命令将指定的 value 追加到该 key 原来值（value）的末尾。

## 哈希命令

### hset key field value

将哈希表 key 中的字段 field 的值设为 value 。

### hdel key field1 [field2]

删除一个或多个哈希表字段

### HEXISTS key field

查看哈希表 key 中，指定的字段是否存在。

### hget key field

获取在哈希表中指定字段的值。

### hgetall key

获取在哈希表中指定key的所有字段和值

### hkeys key

获取所有哈希表中的字段。

## List命令

### 插入元素

#### lpush key element [element ...]

向列表 **头部** 插入一个或多个元素（左进）

返回值：插入后的列表长度

#### RPUSH key element [element ...]

向列表 **尾部** 插入一个或多个元素（右进）

返回值：插入后的列表长度

#### LPUSHX key element

仅当列表存在时，向头部插入元素

返回值：插入后的列表长度（或 0）

#### RPUSHX key element

仅当列表存在时，向尾部插入元素

返回值：插入后的列表长度（或 0）

### 删除元素

#### LPOP key [count]

从列表 **头部** 移除并返回一个或多个元素

返回值：被移除的元素（或 nil）

#### RPOP key [count]

从列表 **尾部** 移除并返回一个或多个元素

返回值：被移除的元素（或 nil）

#### LTRIM key start stop

保留列表中指定范围的元素，其余删除

返回值：状态码 OK

#### LREM key count element

删除列表中 **前/后** count 个匹配的元素

返回值：实际删除的元素数量

### 查询元素

#### LRANGE key start stop

返回列表中指定索引范围的元素（包含两端）

返回值：元素列表

#### LINDEX key index

返回列表中指定索引位置的元素

返回值：元素值（或 nil）

#### LLEN key

返回列表的长度

返回值：列表长度（或 0）

### ****阻塞式弹出元素****

#### BLPOP key [key ...] timeout

#### BRPOP key [key ...] timeout

### 元素操作

#### LSET key index element

设置列表中指定索引位置的元素值

返回值：状态码 OK

#### LINSERT key BEFORE|AFTER pivot element

在列表中某个元素 **前/后** 插入新元素

返回值：插入后的列表长度（或 -1）

### 列表间操作

#### RPOPLPUSH source destination

从源列表尾部弹出元素，并插入目标列表头部

返回值：被移动的元素（或 nil）

#### BRPOPLPUSH source destination timeout

阻塞版本的 RPOPLPUSH，直到超时或元素可用

返回值：被移动的元素（或 nil）

### 索引规则

正数索引：从头部（左）开始，0 表示第一个元素，1 表示第二个，依此类推。

负数索引：从尾部（右）开始，-1 表示最后一个元素，-2 表示倒数第二个，依此类推。

### 时间复杂度

#### 高效操作

LPUSH/RPUSH/LPOP/RPOP：O(1)（单个元素操作）。

LRANGE：O(S+N)，其中 S 是起始偏移，N 是返回元素数量。

#### 谨慎操作

LINSERT、LREM、LSET：O(N)，遍历列表可能导致性能问题。

### 应用场景

#### 消息队列

生产者：LPUSH 消息到队列。

消费者：BRPOP 阻塞获取消息。

#### 最新动态列表

LPUSH 插入最新数据，LTRIM 0 99 保留最近 100 条

#### 栈结构

入栈：LPUSH，出栈：LPOP。

### 注意事项

1、大列表性能：避免对长列表频繁执行 LINSERT、LREM 等 O(N) 操作。

2、阻塞命令超时：BLPOP/BRPOP 的 timeout 单位为秒，设为 0 表示无限等待。

3、空列表处理：当列表为空时，Redis 会自动删除该键。

## bitmap命令

### SETBIT key offset value

1、针对key存储的字符串值，设置或清除指定偏移量offset上的位(bit)

2、位的设置或清除取决于value值，即1或0

3、当key不存在时，会创建一个新的字符串。而且这个字符串的长度会伸展，直到可以满足指定的偏移量offset（0 ≤offset< 2^32），在伸展过程中，新增的位的值被设置为0。

返回值：整数：偏移量offset位置的原始值

### GETBIT key offset

1、返回key对应的字符串，offset位置的位（bit）

2、当offset大于值的长度时，返回0

3、当key不存在时，可以认为value为空字符串，此时offset肯定大于空字符串长度，参考上一条，也返回0

返回值：整数：偏移量offset位置的bit值

### BITCOUNT key [start end]

1、统计给定字符串中，比特值为1的数量

2、默认会统计整个字符串，同时也可以通过指定 start 和 end 来限定范围

3、start 和 end 也可以是负数，-1表示最后一个字节，-2表示倒数第二个字节。注意这里是字节，1字节=8比特

4、如果key不存在，返回0。

返回值：整数：bit值为1的数量

使用场景：

### BITPOS key bit [start [end]]

1、返回字符串中，从左到右，第一个比特值为bit（0或1）的偏移量

2、默认情况下会检查整个字符串，但是也可以通过指定start和end变量来指定字节范围，与BITCOUNT中的范围描述一致

3、SETBIT和GETBIT指定的都是比特偏移量，BITCOUNT和BITPOS指定的是字节范围

4、不论是否指定查询范围，该命令返回的偏移量都是基于0开始的

5、如果key不存在，认为是空字符串

返回值：

整数：第一个比特值为指定bit（0或1）的偏移量

### BITOP operation destkey key [key ...]

1、对多个字符串进行位操作，并将结果保存到destkey中

2、operation 可以是 AND、OR、XOR 或者 NOT

3、BITOP AND destkey srckey1 srckey2 srckey3 ... srckeyN，对多个key求逻辑与，并将结果保存到destkey中

4、BITOP OR destkey srckey1 srckey2 srckey3 ... srckeyN，对多个key求逻辑或，并将结果保存到destkey中

5、BITOP XOR destkey srckey1 srckey2 srckey3 ... srckeyN，对多个key求异或，并将结果保存到destkey中

6、BITOP NOT destkey srckey，对key求逻辑非，并将结果保存到destkey中。

7、除了 NOT 操作之外，其他操作都可以接受一个或多个 key 作为输入

返回值：整数：保存到 destkey 的字符串的长度（和参数给定的key中最长的字符串长度相等）

### BITFIELD key [GET type offset] [SET type offset value] [INCRBY type offset increment] [OVERFLOW WRAP|SAT|FAIL]

将Bitmap看作由多个整数组成的，对其中的整数操作

## 同步的命令

### save

SAVE 命令用于在 Redis 中执行同步保存（synchronous save）操作，即创建当前数据库的一个时间点的备份。与 BGSAVE 命令不同，SAVE 命令会阻塞 Redis 服务器的主进程，直到备份过程完成为止。

1、同步保存操作

（1）当执行 SAVE 命令时，Redis 服务器会立即开始创建一个当前数据库的快照文件（RDB 文件）。

（2）在保存操作完成之前，Redis 服务器会一直处于阻塞状态，无法处理其他请求，直到保存操作完成。

2、创建RDB文件

与 BGSAVE 命令类似，SAVE 命令也会生成一个包含当前数据库状态的 RDB 文件。这个文件可以用来还原数据库状态，也可以作为备份用途。

3、适用场景

由于 SAVE 命令会阻塞 Redis 服务器的主进程，在执行期间无法处理其他请求，因此它通常不适合用于生产环境或者对服务响应要求较高的情况。一般来说，更推荐使用 BGSAVE 命令进行后台保存操作。

### bgsave

BGSAVE 命令用于在 Redis 中执行后台保存（background save）操作，即创建当前数据库的一个时间点的备份。这个备份会以 RDB 文件的形式存储，可以用来在需要时恢复数据或者进行持久化备份。

1、后台保存操作

（1）BGSAVE 命令会立即创建一个子进程来执行真正的保存操作，而不会阻塞主进程。这使得 Redis 能够在进行后台保存的同时继续响应客户端的请求。

（2）由于是后台操作，所以 BGSAVE 命令返回的并不是保存操作的结果，而是一个字符串 "Background saving started"，表示保存操作已经开始。

2、创建 RDB 文件

在执行 BGSAVE 命令后，Redis 会生成一个当前数据库的快照文件（RDB 文件），其中包含了数据库在执行 BGSAVE 命令时的数据状态。

这个 RDB 文件可以用来还原 Redis 数据库的状态，也可以作为备份用途，可以手动复制到其他位置或者定期备份。

3、适用场景

当需要进行数据库备份或者持久化操作时，可以使用 BGSAVE 命令，避免阻塞客户端请求。

4、注意事项：

（1）执行 BGSAVE 命令需要足够的内存空间来进行备份操作，因为 Redis 在执行 BGSAVE 命令时，会先 fork 出一个子进程，然后在子进程中进行数据的持久化操作。如果内存不足，可能会导致 BGSAVE 失败。

（2）由于 BGSAVE 是一个耗时的操作，如果 Redis 实例中的数据非常大，可能会占用较长的时间和系统资源，因此需要谨慎使用，避免影响正常的服务响应能力。

rdb\_bgsave\_in\_progress:1

在 Redis 中，rdb\_bgsave\_in\_progress 是一个内部的信息标识，用于表示是否有后台的 RDB（Redis 数据库持久化）保存操作正在进行中。当你执行命令 INFO 时，会返回一些有关 Redis 服务器状态的信息，其中就包括了 rdb\_bgsave\_in\_progress 这个字段。如果它的值为 1，表示当前有一个后台的 RDB 保存操作正在进行中；如果值为 0，则表示没有后台 RDB 保存操作正在进行。

## 执行Lua脚本

# Lua脚本语法

Lua 数组的索引从**1**开始（不是 0）

# 过期策略

## 基本概念

Redis 的过期策略是管理键生命周期和内存回收的核心机制，主要通过 **被动删除（惰性删除）** 和 **主动删除（定期删除）** 两种方式结合实现，同时配合 **内存淘汰策略** 防止内存溢出。

## 过期键的设置方法

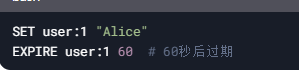
通过以下命令为键设置过期时间：

EXPIRE key seconds：设置键在指定秒数后过期。

EXPIREAT key timestamp：设置键在指定时间戳（秒级）过期。

PEXPIRE key milliseconds：毫秒级版本的 EXPIRE。

PEXPIREAT key timestamp：毫秒级版本的 EXPIREAT。



## 过期键的删除策略

Redis 采用 **被动删除 + 主动删除** 的组合策略，平衡性能和内存效率。

### 被动删除

原理：当客户端尝试访问一个键时，Redis 会检查该键是否已过期。如果过期，则直接删除并返回空值（nil）。

优点：对 CPU 友好，只在访问时触发删除操作。

缺点：如果键长期不被访问，即使已过期也会占用内存（内存泄漏风险）

### 主动删除

原理：Redis 每隔一段时间（默认每秒 10 次）运行 serverCron 任务，随机抽取部分设置了过期时间的键，删除其中已过期的键。

步骤：

随机选择 20 个键检查。

删除其中已过期的键。

如果发现超过 25% 的键已过期，重复步骤 1。

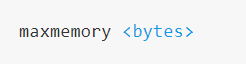
优点：减少内存浪费，防止大量过期键堆积。

缺点：可能占用额外 CPU 资源，尤其在大量键同时过期时。

## 内存淘汰策略

### 配置redis最大使用的内存

Redis提供了一个参数 maxmemory来配置Redis最大使用内存：



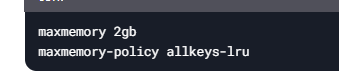
也可以通过命令config set maxmemory1GB来动态修改

### 淘汰策略



### 配置方式

在redis.conf文件中进行配置



## 过期键的持久化处理

### RDB 持久化

1、生成 RDB 文件时：过期键不会被保存到 RDB 文件中。

2、加载 RDB 文件时：主节点会过滤过期键，从节点不会（需依赖主节点同步删除）。

### AOF持久化

1、AOF 写入时：过期键被删除后，会追加一条 DEL 命令到 AOF 文件。

2、AOF 重写时：已过期的键不会被写入新 AOF 文件。

## 配置建议与最佳实践

1、合理设置过期时间：避免大量键同时过期（可加随机偏移，如 EXPIRE key 60 + rand(10)）。

2、监控内存使用：通过 INFO memory 命令观察内存和键淘汰情况。

3、选择淘汰策略：

（1）优先使用 allkeys-lru 或 allkeys-lfu（适合大多数场景）。

（2）需要保留持久数据时，用 volatile-\* 策略。

4、避免 noeviction：除非明确数据不可丢失且能接受写拒绝。

## LRU算法

### 基本概念

**LRU（Least Recently Used，最近最少使用）** 是一种常用的缓存淘汰算法，用于在缓存空间不足时决定哪些数据应该被移除。其核心思想是：**优先淘汰最近最少使用的数据**，保留最近频繁访问的数据。

### 核心思想

1、最近使用：如果一个数据最近被访问过，那么它在未来被访问的概率更高。

2、最少使用：如果一个数据长时间未被访问，那么它在未来被访问的概率更低。

基于这一思想，LRU 算法会优先淘汰那些长时间未被访问的数据。

### 实现方式

### LFU算法

# 为什么Redis需要把所有数据放到内存中

Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 Redis 具有快速和数据持久化的特征，如果不将数据放在内存中，磁盘I/O速度严重影响 Redis 的性能，在内存越来越便宜的今天，Redis 将会越来越受欢迎，如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

# Redis适合的场景

## 1、会话缓存

Redis 会话缓存指的是使用 Redis 缓存会话信息以提升 Web 应用程序性能和扩展性的技术。Web 应用程序通常需要为每个用户维护一个会话，以存储用户身份验证、用户配置、购物车、浏览器历史记录等信息。如果不进行缓存，这些信息将储存在内存中，并随着每个请求的到来而加载。对于高流量的 Web 应用程序，这可能会导致服务器性能下降。

通过将会话数据存储在 Redis 中，可以实现以下优势：

**提高性能**：Redis 使用内存作为主要的存储介质，因此能够更快地读取和写入数据。相对于读取和写入磁盘的传统数据库来说，Redis 可以极大地提高 Web 应用程序的性能。

**扩展性**：Redis 可以快速扩展，即使在高流量的情况下，也能保持快速响应。一些常见的 Web 应用程序扩展解决方案利用多台服务器和负载均衡器，会话共享策略可以借助 Redis 来实现会话的共享。

**可靠性**： Redis 支持数据备份和容错，可以根据需要设置主从复制和集群。

通过将会话数据存储在 Redis 中，Web 应用程序可以更快地响应客户端请求，并具备更好的扩展性和可靠性。

需要注意的是，在使用 Redis 会话缓存时，需要将 session 的数据序列化成二进制格式存储在 Redis 中，以确保能够正确地进行数据存取。同时，还需要注意会话过期机制，避免长时间未被使用的 session 占用过多的系统资源。

## 2、全页缓存

## 3、队列

Redis在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言 (如Python)对 list 的 push/pop 操作。

## 4、排行榜/计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合 (Set) 和有序集合 (SortedSet) 也使得我们在执行这些操作的时候变的非常简单，Redis 正好提供了这两种数据结构。

## 5、发布/订阅

最后(但肯定不是最不重要的)是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统。

# redis为什么做集群

1、防止单点故障

2、处理高并发

3、处理大量数据

# redis集群方式

## 主从复制模式

主从复制（又名读写分离）：是指将一台Redis服务器的数据，复制到其他的Redis服务器。前者称为主节点(master)，后者称为从节点(slave)，数据的复制是单向的，只能由主节点到从节点。默认情况下，每台Redis服务器都是主节点；且一个主节点可以有多个从节点(或没有从节点)，但一个从节点只能有一个主节点。**写请求到主Redis，读请求到从Redis**,读/写的路由需要负载均衡器(主Twemproxy/从Twemproxy) ,而主从Redis的负载均衡器需要做主备切换(keeplived)

图示

描述已自动生成

Redis主从复制，读写分离，提升读性能

Twemproxy代理实现请求的分发

Twemproxy集群(主备)解决单点故障

Keepalived实现Twemproxy的主备切换，以及虚拟ip。

### 优点

读写分离，分担读的压力。

### 缺点

1、主挂掉，整个集群就挂掉了。

2、不具备自动容错和恢复功能 :主从宕机会影响前端的部分请求失败，需要重启机器或者手动切换前端IP才能恢复

3、主机宕机数据丢失：宕机前部分有部分数据未同步到从机，切换IP后会引入数据不一致降低系统可用性

4、数据达到redis的容量，不方便扩展。

### 作用

1、数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。

2、故障恢复：当主节点出现问题时可以由从节点提供服务实现快速的故障恢复；实际上是一种服务的冗余。

3、负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写Redis数据时应用连接主节点，读Redis数据时应用连接从节点），分担服务器负载；尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高Redis服务器的并发量。

4、读写分离：可以用于实现读写分离，主库写、从库读，读写分离不仅可以提高服务器的负载能力，同时可根据需求的变化，改变从库的数量。

5、高可用基石：除了上述作用以外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是Redis高可用的基础。

## Sentinel（哨兵）模式

### 基本概念

主从复制模式中，主服务中断后，需要人工手动将从服务切换为主服务，以便继续提供服务。Redis 2.8中，哨兵模式解决了这一问题，哨兵模式可以实现**自动化的系统监控和故障恢复（主从切换）功能。**

图示

描述已自动生成

哨兵会持续去ping Redis的服务器，若多数哨兵ping不通服务器，表示这个服务器下线。

1、主观下线：一个哨兵觉得服务器下线

2、客观下线：多数哨兵认为服务器下线

3、故障转移：主从切换

## Cluster（集群）模式

图示

描述已自动生成

### 基本概念

Redis-Cluster采用无中心结构，集群中的每个节点都是平等的关系，都是对等的，每个节点都保存各自的数据和整个集群的状态。每个节点都和其他所有节点连接，而且这些连接保持活跃，这样就保证了我们只需要连接集群中的任意一个节点，就可以获取到其他节点的数据。

### 为什么需要redis cluster

哨兵模式基本已经可以实现高可用，读写分离 ，但是在这种模式下每台redis服务器都存储相同的数据，很浪费内存，所以在redis3.0上加入了cluster模式，实现的redis的分布式存储，也就是说每台redis节点上存储不同的内容。

### 数据分散存储

Redis 集群并没有使用传统的一致性哈希来分配数据，而是采用另外一种叫做 哈希槽（hash slot） 的方式来分配的。redis cluster 默认分配了 16384 个slot，当我们set一个key 时，会用CRC16算法来取模得到所属的slot，然后将这个key 分到哈希槽区间的节点上，具体算法就是：CRC16(key) % 16384。

### 容错机制-投票

为了防止主节点数据丢失，可以为每个主节点准备特定数目的从节点，主节点挂掉从节点可以升级为主节点(哨兵模式) 。容错机制指的是,如果半数以上master节点与故障节点通信超过(cluster-node-timeout),认为该节点故障，自动触发故障转移操作。 故障节点对应的从节点自动升级为主节点，如果某个主挂掉，而没有从节点可以使用，那么整个Redis集群进入宕机状态。

### 总结

1、多主多从，分摊写的压力。

2、分布式数据存储，解决存储大量数据问题

3、分散存储通过hash槽，一共有 16384 个槽位，会通过一个算法计算出槽位在各个主之间的具体分配

4、多个主之间不能进行数据交换，但是可以进行通信联系

5、投票机制-多主之间，超过半数通信故障，视为该故障主挂掉，主从切换，若该故障主没有从整个集群进入宕机状态。

# redis实现分布式锁

## 基本概念

日常开发中，秒杀下单、抢红包等业务场景，都需要用到分布式锁。而Redis非常适合作为分布式锁使用。

分布式锁就是控制分布式系统不同进程共同访问共享资源的一种锁的实现。如果不同的系统或同一个系统的不同主机之间共享了某个临界资源，往往需要互斥来防止彼此干扰，以保证一致性。

## 分布式锁的特征

### 1、互斥性

任意时刻，只有一个客户端能持有锁。

### 2、锁超时释放

持有锁超时，可以释放，防止不必要的资源浪费，也可以防止死锁。

### 3、可重入性

一个线程如果获取了锁之后,可以再次对其请求加锁。

### 4、高性能和高可用

加锁和解锁需要开销尽可能低，同时也要保证高可用，避免分布式锁失效。

### 5、安全性

锁只能被持有的客户端删除，不能被其他客户端删除。

## 使用场景

### 1、库存扣减

（1）电商秒杀、抢购场景

（2）防止超卖问题

（3）确保库存数据一致性

### 2、防止重复提交

（1）订单重复创建

（2）支付重复扣款

（3）表单重复提交

### 3、定时任务调度

（1）多节点部署时只允许一个节点执行

（2）避免任务重复执行

（3）分布式Cron作业

### 4、资源互斥访问

（1）配置文件更新

（2）数据库迁移脚本

（3）系统初始化操作

### 5、分布式ID生成

（1）确保ID唯一性

（2）序列号生成

（3）业务编号分配

### 6、缓存更新

（1）Redis缓存失效重建

（2）防止缓存击穿

（3）热点数据更新

## 方案一：SETNX + EXPIRE

setnx+ expire命令。即先用setnx来抢锁，如果抢到之后，再用expire给锁设置一个过期时间，防止锁忘记了释放。

SETNX 是SET IF NOT EXISTS的简写，日常命令格式是SETNX key value，如果 key不存在，则SETNX成功返回1，如果这个key已经存在了，则返回0。

假设某电商网站的某商品做秒杀活动，key可以设置为key\_resource\_id,value设置任意值，伪代码如下：

文本, 信件

描述已自动生成

但是这个方案中，setnx和expire两个命令分开了，**不是原子操作**。如果执行完setnx加锁，正要执行expire设置过期时间时，进程crash或者要重启维护了，那么这个锁就“长生不老”了，**别的线程永远获取不到锁了。**

## 方案二：SETNX + value值是(系统时间+过期时间)

https://juejin.cn/post/6936956908007850014

## 方案三：使用Lua脚本(包含SETNX + EXPIRE两条指令)

# redis持久化

## 基本概念

由于Redis是内存数据库，那么有很多内存的特点，例如掉电易失，或者进程退出，服务器中的数据也将消失不见，所以需要一种方法将数据从内存中写到磁盘，这一过程称之为数据持久化。

持久化有两种方式，一种是RDB，操作手段是将数据从内存中写到磁盘，生成一个经过压缩的RDB文件，另一种持久化方式叫AOF，是把Redis执行的命令行逐句记录下来，追加在类似日志的文件中。

## RDB和AOF的区别

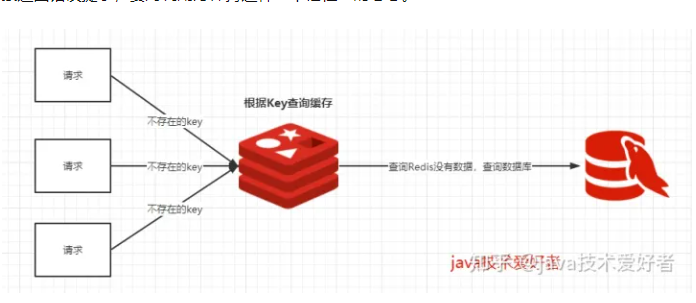
# 分布式缓存

## 缓存穿透

### 基本概念

我们使用Redis大部分情况都是通过Key查询对应的值，假如发送的请求传进来的key是不存在Redis中的，那么就查不到缓存，查不到缓存就会去数据库查询。假如有大量这样的请求，这些请求像“穿透”了缓存一样直接打在数据库上，这种现象就叫做缓存穿透。

### 示意图



### 分析

关键在于在Redis查不到key值，这和缓存击穿有根本的区别，区别在于缓存穿透的情况是传进来的key在Redis中是不存在的。假如有黑客传进大量不存在的key，那么大量的请求打在数据库上是很致命的问题，所以在日常开发中要对参数做好校验，一些非法的参数，不可能存在的key就直接返回错误提示，要对调用方保持这种“不信任”的心态。

### 解决方案

1、把无效的Key存进Redis中。如果Redis查不到数据，数据库也查不到，我们把这个Key值保存进Redis，设置value="null"，当下次再通过这个Key查询时就不需要再查询数据库。这种处理方式肯定是有问题的，假如传进来的这个不存在的Key值每次都是随机的，那存进Redis也没有意义。

2、使用布隆过滤器。布隆过滤器的作用是某个 key 不存在，那么就一定不存在，它说某个 key 存在，那么很大可能是存在(存在一定的误判率)。于是我们可以在缓存之前再加一层布隆过滤器，在查询的时候先去布隆过滤器查询 key 是否存在，如果不存在就直接返回。

## 缓存击穿

### 基本概念

其实跟缓存雪崩有点类似，缓存雪崩是大规模的key失效，而缓存击穿是一个热点的Key，有大并发集中对其进行访问，突然间这个Key失效了，导致大并发全部打在数据库上，导致数据库压力剧增。这种现象就叫做缓存击穿。

### 分析

关键在于某个热点的key失效了，导致大并发集中打在数据库上。所以要从两个方面解决，第一是否可以考虑热点key不设置过期时间，第二是否可以考虑降低打在数据库上的请求数量。

### 解决方案

1、如果业务允许的话，对于热点的key可以设置永不过期的key。

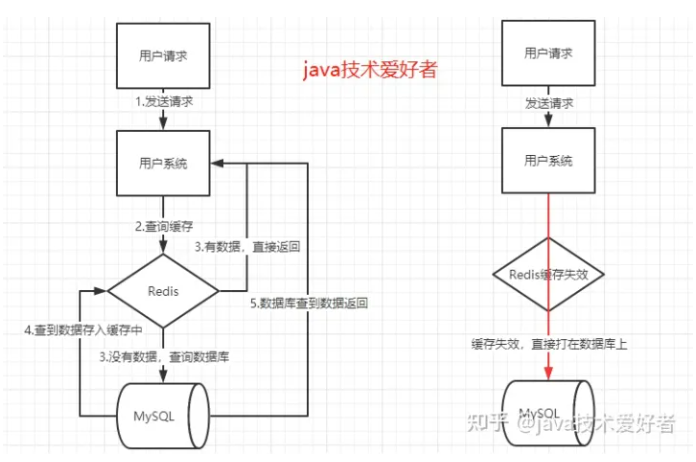
2、使用互斥锁。如果缓存失效的情况，只有拿到锁才可以查询数据库，降低了在同一时刻打在数据库上的请求，防止数据库打死，当然这样会导致系统的性能变差。

## 缓存雪崩

### 基本概念

当某一个时刻出现大规模的缓存失效的情况，那么就会导致大量的请求直接打在数据库上面，导致数据库压力巨大，如果在高并发的情况下，可能瞬间就会导致数据库宕机。这时候如果运维马上又重启数据库，马上又会有新的流量把数据库打死，这就是缓存雪崩。

### 示意图



### 分析

造成缓存雪崩的关键在于在同一时间大规模的key失效。为什么会出现这个问题呢，有几种可能，第一种可能是Redis宕机，第二种可能是采用了相同的过期时间。

### 解决方案

1、在原有的失效时间上加上一个随机值，比如1-5分钟随机。这样就避免了因为采用相同的过期时间导致的缓存雪崩。

2、使用熔断机制。当流量到达一定的阈值时，就直接返回“系统拥挤”之类的提示，防止过多的请求打在数据库上，至少能保证一部分用户是可以正常使用，其他用户多刷新几次也能得到结果。

3、提高数据库的容灾能力，可以使用分库分表，读写分离的策略。

4、为了防止Redis宕机导致缓存雪崩的问题，可以搭建Redis集群，提高Redis的容灾性。

## 总结

这三个问题在使用Redis的时候是肯定会遇到的，而且是非常致命性的问题，所以在日常开发中一定要注意，每次使用Redis时，都要对其保持严谨的态度。还有一个需要注意的是要做好熔断，一旦出现缓存雪崩，击穿，穿透这种情况，至少还有熔断机制保护数据库不会被打死。

# redis中的数据与数据库的数据保持同步

## 1、****缓存失效策略（Cache Invalidation）****

### 核心思想

当数据库数据变更时，主动使对应的缓存失效，下次访问时重新加载

### 实现步骤

1、更新数据库：先执行数据库更新操作。

2、删除缓存：立即删除对应的Redis缓存。

3、下次读取：当有请求访问数据时，发现缓存不存在，从数据库加载最新数据并写入缓存

### 代码示例



### 适用场景

读多写少的场景。

对数据实时性要求不高的情况。

### 优缺点

优点：实现简单，缓存按需加载。

缺点：存在缓存击穿风险（多个请求同时访问未命中的缓存）。

## 2、****写穿透策略（Write-Through）****

### 核心思想

每次写操作同时更新数据库和缓存，确保缓存数据最新。

### 实现步骤

1、更新数据库：执行数据库更新。

2、更新缓存：将新数据同步写入Redis。

### 代码示例



### 适用场景

写操作频繁但数据一致性要求高的场景（如金融交易）。

### 优缺点

优点：缓存数据始终最新。

缺点：写操作延迟增加，频繁写操作可能导致缓存抖动。

## 3、延迟双删策略

### 核心思想

在数据库更新后，先删除缓存，延迟一段时间再次删除，防止旧数据回填。

### 实现步骤

1、删除缓存：更新数据库前先删除缓存。

2、更新数据库：执行数据库更新。

3、延迟二次删除：延迟一段时间（如500ms）后再次删除缓存。

### 代码示例



### 适用场景

高并发环境，避免缓存与数据库短暂不一致。

### 注意事项

延迟时间需根据业务峰值调整（通常大于主从同步时间）。

## 4、基于消息队列的异步同步

### 核心思想

通过消息队列解耦数据库更新与缓存操作。

### 实现步骤

发布变更事件：数据库更新后，向消息队列发送事件（如订单更新）。

消费事件更新缓存：消费者读取事件并更新或删除缓存。

### 代码示例

使用RabbitMQ



### 适用场景

大型分布式系统，需要高可靠性和解耦。

### 优缺点

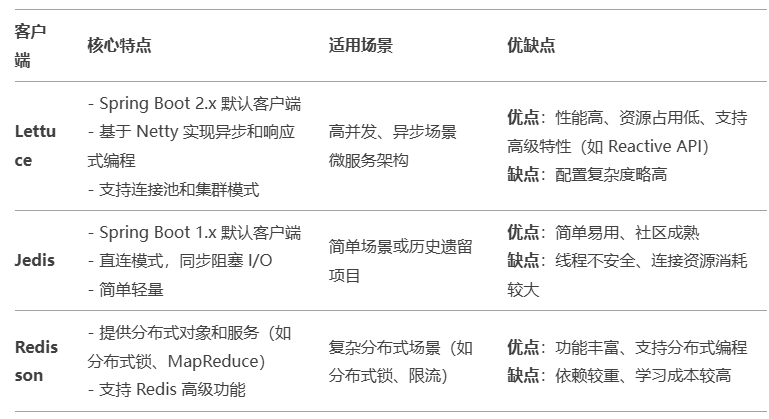
优点：系统解耦，支持最终一致性。

缺点：架构复杂度增加。

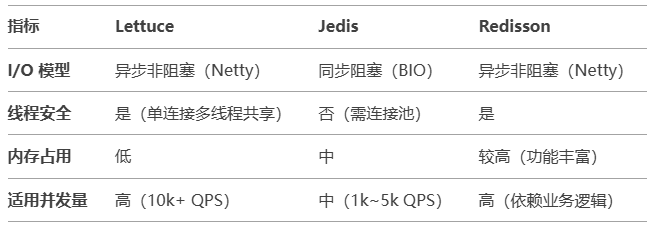
# Redis支持的Java客户端都有哪些

Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

# redis客户端对比



## 性能与资源消耗对比



## 建议

1、优先选择 Lettuce：适合大多数场景，性能优异且为 Spring Boot 默认集成。

2、复杂分布式需求：选择 Redisson，利用其分布式对象和服务简化开发。

3、维护旧项目：沿用 Jedis，但建议逐步迁移到 Lettuce。

# Redisson

## 基本概念

**Redisson** 是一个基于Netty框架的高级Redis客户端，专为分布式系统设计。它不仅支持基础的Redis操作，还提供了丰富的分布式对象和服务，简化了分布式环境下的开发工作。Redisson通过封装复杂的分布式逻辑，使开发者能够更专注于业务实现。

## 核心特性

### 1、分布式对象

如锁、集合、队列、原子变量等。

### 2、多种部署模式支持

单节点、主从、哨兵、集群、云托管。

### 3、响应式与异步API

支持Reactive编程模型。

### 4、丰富的功能模块

分布式锁、Bloom过滤器、HyperLogLog等。

### 5、与Spring框架深度整合

轻松集成Spring Boot、Spring Cache。

## 主要功能

### 1、****分布式锁（RLock）****

Redisson的分布式锁解决了多节点环境下的资源竞争问题，支持自动续期和可重入。

### ****2、分布式集合****

RMap：分布式Map，支持本地缓存和淘汰策略。

RList：分布式列表，支持元素顺序操作。

RSet：分布式无序集合，支持交并差运算。

### **3、**发布/订阅****

支持消息的发布与订阅，适用于事件驱动架构。

### **4、其他高级功能**

分布式计数器（RAtomicLong）：原子增减操作。

限流器（RRateLimiter）：控制访问速率。

分布式调度服务（RScheduler）：支持定时任务。

## 优点

1、提供丰富的分布式对象，简化开发。

2、高可用性，支持多种部署模式。

3、完善的文档和活跃的社区支持。

## 缺点

1、学习曲线较陡，功能复杂。

2、依赖Netty，可能需处理版本冲突。

## 适用场景

1、分布式锁和同步机制。

2、分布式集合和队列管理。

3、高并发环境下的数据缓存与处理。

# 一致性哈希

## 基本概念

一种特殊的哈希算法，目的是解决分布式缓存的问题。在移除或者添加一个服务器时，能够尽可能小地改变已存在的服务请求与处理请求服务器之间的映射关系。一致性哈希解决了简单哈希算法在分布式[哈希表](https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/5981869?fromModule=lemma_inlink)( Distributed Hash Table，DHT) 中存在的动态伸缩等问题。

# 哈希槽的概念

Redis 集群没有使用一致性 hash,而是引入了哈希槽的概念，Redis 集群有 16384个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置在哪个槽，集群的每个节点负责一部分的hash槽。

# Redis集群会有写操作丢失吗

Redis 并不能保证数据的强一致性,这意味这在实际中集群在特定的条件下可能会丢失写操作。

# Redis集群之间是如何复制的

异步复制

# Redis集群最大节点个数是多少

16384个

# Redis集群如何选择数据库

Redis 集群目前无法做数据库选择，默认在 0 数据库。

# Redis中的管道有什么用

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应，这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

# 怎么理解Redis的事务

事务是一个单独的隔离操作: 事务中的所有命令都会序列化、按顺序地执行，事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作: 事务中的命令要么全部被执行，要么全部都不执行。

# Redis事务相关的命令

MULTI、EXEC、DISCARD、WATCH

# Redis key的过期时间和永久有效分别怎么设置

EXPIRE 和 PERSIST 命令

# Redis如何做内存优化

尽可能使用散列表 (hashes) ，散列表(是说散列表里面存储的数少) 使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key,而是应该把这个用户的所有信息存储到一张散列表里面。

# Redis回收进程如何工作的

一个客户端运行了新的命令，添加了新的数据。Redi 检查内存使用情况，如果大于 maxmemory 的限制，则根据设定好的策略进行回收。一个新的命令被执行，等等。所以我们不断地穿越内存限制的边界通过不断达到边界然后不断地回收到边界以下。如果一个命令的结果导致大量内存被使用(例如很大的集合的交集保存到一个新的键)，不用多久内存限制就会被这个内存使用量超越。