### redis 安装

安装：brew install redis

启动：cd /usr/local/Cellar/redis

redis-server

或用后台服务启动：

brew services start redis

安装成功后可以使用客户端程序redis-cli和redis服务交互

### 基础

redis有五种数据类型：string,list,hash,set,zset

#### key

键相关命令：

del key //key存在时删除key

keys foo\* //查询所有以foo开头的key

keys \* //查询所有key

exists key //判断key是否存在

expire key time(秒) //为key设置过期时间，time是秒数

expireat key timestamp //设置过期时间，timestamp是秒级时间戳

ttl key //返回key剩余生存时间（秒）

rename key newkey //改key名

type key //获取key对应value的类型

#### string

一个key对应一个value

set key value //设置键值：若key已存在，会覆盖

get key m //获取键值

setnx key value //设置键值，若key已存在，会设置失败

mset key1 value1 key2 value2 //同时设置多个key-value

msetnx key1 vlaue1 key2 value2//同时设置多个，只有key全部都不存在时成功

mget key1 key2 //同时获取多个key的值

setex key timeout value //设置key的值，并设置过期时间（秒）（若key存在会覆盖）

##Attention: 数字值在 Redis 中以字符串的形式保存

incr key //将value值加1

incrby key increment //将value加指定值

decr key //value值减1

decrby key decrement // value值减指定值

getset key newvalue //设置新值，返回旧值

strlen key //返回value长度

getrange key start end //返回从start到end范围内的字符串(闭包)

append key sth //追加sth到key的value后面

#### hash

适合存储对象

hget key field //获取key中field字段对应的value，若key或field不存在，返回nil

hmget key field1 field2…//获取多个field的值

hset key field value //为key中的field字段赋值value，不存在创建，存在则覆盖

hmset key field1 value3 field2 value4 //设置多个field的value，会覆盖旧值

hgetall key //获取key中所有field及value

hkeys key //获取key中所有filed

hvals key //获取key中所有vlaue

hsetnx key field value //若field存在则设置失败

hincrby key field num //为field加num，num可是负数

hexists key field //判断key中是否存在field

hdel key field1 field2 //删除一个或多个字段

hscan key cursor match pattern // 支持模糊搜素

cursor: 游标，一般从0开始

pattern: 匹配规则

例如： hset test field\_1 10

hset test field\_2 20

hscan test 0 match field\_\*

可查询出field\_1,field\_2

#### list （有序列表）

//将一个值或多个值插入到列表头部(顺序为value1🡪vlaue2)，若key不存在，则会创建，若key已存在，但并不是列表类型，则报错。返回列表的长度

lpush key value1 value2 …

lpop key //移除列表第一个元素

lpushx key value1 value2… //key的列表存在是插入，不存在时插入失败

lrem key count value //移除count个value

count > 0 从表头开始向表尾搜索，移除与 VALUE 相等的元素，数量为 COUNT 。

count < 0 : 从表尾开始向表头搜索，移除与 VALUE 相等的元素，数量为 COUNT 的绝对值。

count = 0 : 移除表中所有与 VALUE 相等的值。

lrange key start end //返回指定区间内的元素

ltrim key start end //对列表进行修剪，只留下区间内的元素（跟lrange区别）

llen key //返回列表长度

lset key index value //通过index索引来更改列表元素的值

lindex key index //通过index获取列表某个元素

rpush key value1 value2 //从列表后端添加一个或多个值

rpop key //从列表后端移除一个元素

rpushx key vlaue1 value2 //为已存在的列表添加一个或多个值

#### set （无序不重复集合）

sadd key vlaue1 value2 //向集合添加一个或多个值，集合中已存在则忽略，返回插入的数量

smembers key // 获取集合中所有元素

srem key value1 value2 //移除集合中一个或多个元素，返回移除的数量

spop key //随机移除一个元素（因为set是无序的），返回被移除的元素

sdiff key1 key2 key3… //返回第一个集合里面不存在于另外集合里面的元素

sinter key1 key2 … //返回多个集合的交集

sunion key1 key2 … //返回多个集合的并集

scard key //返回集合中元素的个数

sismember key value //判断value是否是集合的成员，是返回1，其他返回0

srandmember key [count] //随机返回集合中指定count个元素，count不传返回一个

#### zset

每个元素都关联一个double类型的分数（score），redis正是通过score来为集合成员进行从小到大排序的。

有序集合成员是唯一的，但score是可以重复的。

zadd key score1 value1 score2 value2… //添加元素到有序集合中

zincrby key increment value //把value对应的score加上increment

zcard key //返回集合中元素数量

zcount key min max //返回score在min~max之间的元素数量

zrange key start end [WITHSCORES] //先按score从小到大排序，start/end是索引，从0开始，withscores参数会把score跟在每个元素后面打印出来

zrevrange key start end [WITHSCORES] //按score从大到小排序，其他同zrange一样

zrangebyscore key min max [withscores] //返回score在min~max范围内的所有成员

zrevrangebyscore key max min //返回score在max~min范围内的成员，从大到小排序

默认是开区间（小于等于或大于等于），可以给参数加‘（’来使用闭区间：

zrangebyscore zset (1 5 //1<score<=5

zrank key member //返回某个元素的索引（有序集合按score值从小到大排序）

zrevrank key member //返回某个元素的索引（有序集合按score值从大到小排序）

zscore key member //返回member的score值

zrem key member1 member2… //删除一个或多个元素

zremrangebyrank key start end //移除索引从start~end的元素（按score从小到大排序）

zremrangebyscore key min max //移除score值在min~max之间的元素

#### 每种数据类型的应用

* + 1. string

value可以是String也可以是数字。一般做一些复杂的计数功能的缓存。

* + 1. hash

key-value结构，方便存放对象

* + 1. list

简单消息队列，先进先出；

lrange 基于redis分页，性能极佳，用户体验好；

* + 1. set

去重

交集、并集、差集等功能

* + 1. zset

有权重参数score，可以按score排序，可以应用于各种排序需求。

#### 发布订阅

* + 1. psubscribe pattern 订阅一个或多个符合给定模式的频道

例如：psubscribe news.\* tweet.\*

* + 1. publish channel message 将message发送到指定的频道，psubscribe页面就会显示发送的信息
    2. subscribe channel [channel…] 订阅给定的一个或多个频道的信息

subscribe new.it //频道是确定的

* + 1. pubsub channels [pattren] 列出（匹配pattern的）当前活跃的频道（不包含psubscribe订阅的频道）

pattern 不写，列出全部活跃的频道

pubsub channels news.\* //列出匹配news.\*的所有频道

* + 1. pubsub numsub [channel…] //返回给定频道的订阅者数量（即通过subscribe订阅的，不含psubscribe订阅的频道）
    2. pubsub numpat //返回客户端订阅的所有模式的总和（只返回psubscribe订阅的频道总数）
    3. punsubscribe pattern //退订所有符合模式的频道
    4. unsubscribe channel… //退订给定的频道

### 应用及问题

#### 延时任务

场景： 1、生成订单30分钟未支付，则自动取消

2、生成订单60秒后,给用户发短信

延时任务和定时任务区别：

1、定时任务有明确的触发时间，延时任务没有

2、定时任务有执行周期，而延时任务在某事件触发后一段时间内执行，没有执行周期

3、定时任务一般执行的是批处理操作是多个任务，而延时任务一般是单个任务

利用redis的zset（有序集合）实现：

score存超时时间戳，value存订单号，zrange查询score最小的成员，然后zrem删除

优点:

(1)由于使用Redis作为消息通道，消息都存储在Redis中。如果发送程序或者任务处理程序挂了，重启之后，还有重新处理数据的可能性。

(2)做集群扩展相当方便

(3)时间准确度高

#### 缓存击穿

缓存击穿是数据库中存在，缓存中不存在，热点数据，并发量大，还未存入缓存时发生的

缓存击穿是针对某一个key的缓存，缓存在某个时间点过期的时候，恰好在这个时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端DB压垮



如果黑客每次请求缓存中不存在的数据，导致每次都要去数据库查询，缓存就失去意义了，大流量情况数据库可能挂掉,这就是缓存击穿。

互斥锁：

setnx用法：setnx key value //key存在会设置失败

在根据key获得的value值为空时，先锁上，再从数据库加载，加载完毕，释放锁。若其他线程发现获取锁失败，则睡眠50ms后重试。

伪代码：

|  |
| --- |
| function lock(lockName,timeout){  let identifier = uuid();  if(redis.setnx(lockName,identifier)){//加锁成功  redis.expire(lockName,timeout);//设置过期时间  return identifier;  }else if(redis.ttl(lockName) === -1){//若没设置过期时间，防死锁  redis.expire(lockName,timeout);//设置过期时间  }  return false;  }  function releaseLock(lockName,identifier){  if(redis.get(lockName) === identifier){//若锁没有被修改  redis.del(lockName);  return true;  }else{  return false;  }  } |

#### 缓存雪崩

缓存大面积同时过期，造成一时间大量请求同时访问数据库，造成异常

解决：过期时间加随机数，避免同时失效的情况

缓存雪崩针对很多key

#### 缓存穿透

缓存没有，数据库也没有

比如黑客请求id为-1的数据

解决：

接口校验；

从缓存取不到的数据，在数据库中也没有取到，这时也可以将key-value对写为key-null，缓存有效时间可以设置短点，如30秒，

#### 数据一致性

方式1：先更新数据库，再更新缓存 不可取

原因1：

若A、B同时请求更新，那么：

* A更新了数据库
* B更新了数据库
* B更新了缓存
* A更新了缓存

B在A之后更新的数据库，B的数据最新，但是网络等原因，B先更新了缓存，A后更新的缓存，A更新的就是旧数据，这样就产生了脏数据

原因：2：

如果写入缓存的数据是需要经过复杂计算，无疑是性能浪费，不如直接删除。

方式2：先删缓存，再更新数据库

因为写操作一般比读操作慢，可能会导致：

* 请求A要进行写操作，删除缓存
* 请求B查询缓存，不存在
* 请求B去查询数据库，获取旧数据
* 请求B将旧数据写入缓存
* 请求A将新值写入数据库

这样如果没有设置缓存过期时间的话，数据永远都是脏数据

解决：延时双删：

* 先删缓存
* 写数据库
* 休眠1s(一定时间)，再删缓存 //（休眠时间一般在读数据的耗时基础上加几百ms即可）

问题：如果写数据库之后删缓存失败，就还是会存在脏数据问题

方式3：先更新数据库，后删缓存

最优解

### 总结

#### 性能

我们在碰到需要执行耗时特别久，且结果不频繁变动的SQL，特别适合将结果放到缓存。后面的请求就去缓存获取。



#### 并发

大并发情况时，所有请求都直接操作数据库，数据库连接会出现问题。用redis做一个缓冲，能减轻数据库的压力。

#### redis过期策略和内存淘汰机制

比如你的redis只能有10G容量，如果超过了，redis会删除旧数据，那它是如何删的；

假如你设置了过期时间，但是时间到期之后，内存占用率还是很高，why？

redis过期策略：

定期删除+惰性删除策略

why not 定时删除？

用定时器来监视key，过期则自动删除。虽然内存及时释放了，但十分消耗CPU资源。在大并发的情景中，CPU应该应用在处理请求，而不是删除key，kill.

定期删除+惰性删除策略 工作机制：

定期删除，redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。

惰性删除来解决剩下问题：

获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。

但是如果定期删除没删掉，也没有去获取过key，两种机制都没删除掉，内存就会越来越高，这时候要采用内存淘汰机制：

redis.conf配置文件里：



volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的key。这种情况一般是把redis既当缓存，又做持久化存储的时候才用。不推荐

allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的key。推荐使用。

allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个key。你不删最少使用Key,去随机删。

volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个key。不推荐

volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的key优先移除。不推荐

noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。不用

原则：删除最少使用的为优先条件

#### redis是单线程的