Redis

NoSQL技术之一，是一个key-value存储

1、redis能干嘛

1、基于内存存储、持久化（rdb、aof）

2、效率高、用于高速缓存

3、发布订阅系统（简单的消息队列）

4、地图信息分析

5、计时器、计数器（浏览量！incr decre）

2、特性

1、多样的数据类型

2、持久化

3、集群

4、事务

3、windows部署redis

太简单。。百度

4、linux下部署redis

推荐使用linux

需要用到的命令:

```
yum install gcc-c++
```

安装的文件都在/usr/local/bin文件夹目录之下

redis默认不是后台启动。。

修改redis.conf 中 deamonize no==>deamonize yes

使用指定的配置文件开启redis 可能用到的命令：redis-server XXX(文件名)/redis.conf

查看当前服务进程信息

```
ps -ef|grep redis
```

5、性能测试

redis-benchmarke压力测试工具

```
redis-benchmark -h localhost -p 6379 -c 100 -n 100000
```

6、基础的知识

redis默认有16个数据库

默认使用的是第0个数据库

```
select 3  # 切换数据库
DBSIZE    # 查看DB大小
keys *    # 查看数据库所有的key
flushall  # 清空全部库
flushdb   # 清空当前库
```

不同的数据库存不同个的值

7、Redis为什么快

redis是将所有的数据放在内存中的，所以使用单线程操作效率就是最高的，多线程（上下文切换：耗时的操作），对于内存系统来说，如果没有上下文切换效率就是最高的，多次读写都是在一个CPU上，在内存的情况下，单线程就是最快的。

8、五大数据类型

8.1 String

####

```
APPEND key1 "hello" # 追加字符串，如果当前key不存在，就相当于set key
EXISTS key1 # 判断某一个key是否存在
STRLEN key1 # 获取字符串的长度
incr views # 自增1
decre views # 自减1
INCRBY views 10 # 设置自增步长
DECREBY views 10 # 设置自减步长
GETRANGE key1 0 3 # 截取字符串 [0,3]
GETRANGE key1 0 -1 # 获取全部字符串和get key是一样的
SETRANGE key2 1 xx # 替换指定位置开始的字符串
setex (set with expire) #设置过期时间
setnx (set if not exist) #不存在再设置
mset # 批量设置值
msetnx # 原子性操作 要么一起成功、要么一起失败


set user:1{name:123,age:3} #设置一个user：1 对象 值为 json字符串来保存一个对象
getset # 先get再set
```

String类似的使用场景：value除了是我们的字符串还可以是我们的数字

计数器

统计多单位的数量

粉丝数

对象缓存存储

8.2 List

```
lset # 将列表中制定下表的值替换为另外一个值 更新操作
exist list # 判断列表是否存在
lset list 0 item # 如果存在，更新当前下标的值
lset list 1 other # 如果不在，则报错

linsert
linsert mylist before "world" "other"
```

小结

实际上是一个链表，before Node after，left ，right都可以插入值

如果key不存在则创建key，如果存在则新增内容

如果移除了所有值，空链表，也不代表不存在

在两边插入或者改动值，效率最高！中间元素，相对来说会效率低一点

> 消息排队！消息队列（Lpush Rpop）左进右出 ，栈（Lpush Lpop）坐进左出

8.3 Set（集合）

set中的值不能重复

```
sadd myset "hello" # set 中添加元素
smembers myset # 查看set集合中所有值
sismenber myset hello # 判断某个值是否在set中
scard myset # 获取set集合中的个数

srem myset "hello" # 移除某一个值
srandmember # 随机抽出一个元素
srandmember myset 2 # 随机抽出两个元素
spop # 随机移除一个元素


smove myset myset2 "kuangshen" # 讲一个指定的值，移动到另外一个set集合


实战：
微博、B站共同关注（交集）
sdiff key1 key2（差集--不同的值）
sinter key1 key2（交集--相同的值）
sunion key1 key2（并集--所有的值）
```

8.4 Hash（哈希）

Map集合，key，这时候这个值是一个map集合！本质和String没有太大区别

```
hset myhash field1 "kuangshen" # set一个具体 key-value
hget myhash field1 # 获取一个字段值
hmset myhash field1 hello field2 world # set多个key-value
hmget myhash field1 field2 # 获取多个字段值
hgetall myhash # 获取全部的数据
hdel # 删除
hlen # 获取hash表中字段的数量
hexist # 判断hash中指定字段是否存在
```

8.5 Zset（有序集合）

在set的基础上，增加了一个值，set k1 v1　zset k1 score1 v1

```
zrangebyscore salary -inf +inf # 从小到大排序
zvrangebyscore #从小到大排序
zcount # 获取指定区间成员数量
```

set 排序 存储班级成绩表 工资表排序

普通消息 带权重进行判断

排行榜应用实现　取 TOP N 测试

9、三种特殊的数据类型

geospatial

```
geoadd # 添加地理位置（两级无法直接添加）我们一般会下载城市数据，通过java程序一次性导入
#参数 key 值（纬度、经度、名称）
geopos # 获取当前定位
geodist # 两人之间的距离（直线）
georadius# 我附近的人？（获取所有附近的人的地址，定位！）通过半径来查询
georadiusmember #找出位于指定元素周围的其他元素
geohash  # 将二维的经纬度转换为一堆的字符串，如果两个字符串越接近，那么则距离越近
```

GEO底层的实现原理是Zset

Hyperloglog

基数：不重复的元素

```
PFadd # 创建元素
PFcount # 统计基数数量
PFMERGE #  合并两组
```

Bitmaps

位存储 位图，数据结构，都是二进制位来进行记录，就只有0和1 两个状态

```
setbit sign 0 #存
getbit # 取
bitcount # 统计
```

10、事务

事务：一组命令的集合！一个事务中的所有命令都会被序列化，在事务执行的过程中，会按照顺序执行

一次性、顺序性、排他性！

Redis单条命令是保存原子性的，但是事务不保证原子性

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

Redis的事务：

开启事务（multi）、命令入队、执行事务（exec）

每次执行完事务之后要重新开启

```
discard # 放弃事务
```

编译型错误，全部都不会运行

运行时异常，只有错的报错，其他正常运行

11、监控

悲观锁：

很悲观，认为什么时候都会出问题，无论什么时候都会加锁

乐观锁：

很乐观，什么时候都没问题，所以不会上锁！更新数据的时候去判断一下，在此期间是否有人修改过这个数据

获取version

更新的时候比较version

redis监控测试

watch监控 unwatch取消监控

12、Jedis

用java操作Redis Jedis是java连接开发工具

Jedis包

13、springboot集成Redis

原来使用的jedis被替换成lettuce

jedis：采用直连，操作时不安全的，避免不安全使用jedis pool连接池 BIO

lettuce：采用netty，实例可以再多个线程中进行共享，不存在线程不安全的情况！可以减少线程数据了，更像Nio模式

opsForValue操作字符串

opsForList 操作list

opsForSet

opsForHash

opsForZSet

opdforGeo

opsForHyperLoglog

14、Redis.conf详解

```
conifg set requirepass "123456" # 设置redis密码
auth 123456 # 使用密码进行登陆

maxclients 10000 # 设置能连接上redis的最大客户端的数量
maxmemory<bytes> # redis 配置最大的内存容量
maxmemory-policy noeviction # 内存达到上线之后的处理策略

1、volatile-lru # 只对设置了过期时间的key进行LRU（默认值）
2、allkeys-lru # 删除lru算法的key
3、volatile-random # 随机删除即将过期的key
4、allkeys-random # 随机删除
5、volatile-ttl # 删除即将过期的
6、noevication # 永不过期，返回错误
```

append only 模式 aof配置

```
appendonly no # 默认是不开启aof模式的，默认是使用rdb方式持久化的，在大部分所有的情况下，rdb完全够用
appendfilename "appendonly.aof" # 持久化的文件的名字
# appendfsync always #每次修改都会sync，消耗性能
appendfsync everysec #每秒执行依次sync，可能会丢失这1s的数据！
# appendfsync no #不执行sync，这个时候操作系统自动同步数据，速度最快
```

快照

持久化，在规定的时间内执行了多少次操作，则会持久化到文件.rdb.aof

```
stop-write-on-bgsave-error yes #持久化如果错误是否继续进行工作
save 900 1 # 如果900s内，如果至少有一个key进行了修改，我们即进行持久化操作
save 300 10
save 60 10000
rdbcompression yes # 是否压缩rdb文件，需要消耗一些cpu资源
rdbchecksum yes # 保存rdb文件的时候，进行错误的检查校验
dir ./ # rdb文件的保存目录
```

15、Redis持久化

Redis是内存数据库断电即失，所以必须要有持久化

RDB（RedisDataBase）

在指定的时间间隔内将内存中的数据集快照写入磁盘，也就是行话讲的snapshot快照

rdb保存的文件是dump.rdb

触发机制

- save的规则满足如下情况，会自动触发rdb规则
- 执行flushdb命令
- 退出redis也会产生rdb文件

如何恢复rdb文件

只需要将rdb文件放在我们redis启动目录就可以，redis启动的时候会自动检查dump.rdb恢复其中的数据

查看需要存在的位置

```
127.0.0.1:6379> config get dir
1)"dir"
2)"/usr/local/bin" # 如果在这个目录下存在dump.rdb文件，启动就会自动恢复其中的数据
```

优点：

适合大规模的数据恢复

如果你对数据的完整性要求不高

缺点：

需要一定的时间间隔进程操作，如果redis宕机了，最后一次的修改数据就没有了！

fork进程的时候会占用一定的内容空间

AOF（AppendOnlyFile）

将我们的所有写操作命令都记录下来，history，恢复的时候就把这个文件全部执行一遍

默认不开启的我们需要手动进行配置我们只需要把 appendonly 改为 yes就行了

重启redis就可以生效了

如果aof有错误，redis是启动不起来的，我们要修复aof文件

redis给我们提供了这个工具，redis-check-aof --fix

优点：

每一次修改都同步、文件的完整性更好

每秒同步依次，可能会丢失一秒的数据

从不同步，效率最高

缺点：

相对于数据文件来说，aof远远大于rdb，修复的速度也比rdb慢

aof运行效率也比rdb低，所以默认使用的是rdb持久化

重写规则说明

如果aof文件大于64m，太大了~fork一个新的进程来将我们的文件进行重写

16、Redis发布订阅

redis发布订阅（pub/sub）是一种消息通信模式，发送者（pub）发送消息，订阅者（sub）接收消息

1：发送者 2：频道 3：接受者

```
subscribe kuangshenshuo(channel) # 订阅一个频道
publish kuangshenshuo "hello" # 发布者发布消息到频道
```

应用场景

实时消息系统，实时聊天（频道当聊天室，将信息回显给所有人），订阅关注系统

稍微复杂的场景就会使用消息中间件MQ

17、Redis主从复制

主从复制是指将一台Redis服务器的数据，复制到其他的Redis鸟服务器，前者成为主节点，后者称为从节点（slave/follow），数据的复制是单向的，只能由主节点到从节点，Master以写为主，slave以读为主

默认情况下，每台Redis服务器都是主节点

主从复制的作用主要包括：

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：当主节点出现问题的时候，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（写Redis 数据的时候应用连接主节点，读Redis数据时应用连接从节点）分担服务器负载，尤其是少写多读的场景下，大大提高Redis服务器的并发量

4.高可用（集群）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

一般来说只使用一台reids是不够的

1：从结构上，单个Redis服务器会发生单点故障，并且一台服务器需要处理所有的请求负载，压力较大

2：从容量上，单个Redis服务器内存容量有限，一般来说，单台Redis最大使用内存不应该超过20G

环境配置

只配置从库

```
info replication  # 查看当前库的信息
```

一主二从

```
slaveof 127.0.0.1 6379 # 认主机
```

复制原理

slave启动成功连接到master后会发送一个sync同步命令

master接到命令，启动后台的存盘进程，同时收集所有接收到的用于修改数据集命令，在后台进程执行完毕之后，master将传送整个数据文件到slave，并完成一次完全同步。

全量复制：而slave服务在接收到数据库文件数据后，将其存盘并加载到内存中

增量复制：master继续将新的所有收集到的修改命令依次传给slave，完成同步

但是只要是重新链接master，一次完全同步（全量复制）将被自动执行

slaveof no one 手动设置老大

哨兵模式（自动选取老大模式）

能够后台控主机是否故障，如果故障了根据投票数自动将从数据库转换为主库

哨兵模式是一种特殊的模式，首先redis提供了哨兵的命令，哨兵是一个独立的进程，作为进程，他会独立运行，其原理是哨兵通过发送命令，等待redis服务器响应，从而监控运行的多个redis实例

配置哨兵配置文件sentinel.conf

sentinel monitor 被监控的设备 host port 1

sentinel monitor myredis 127.0.0.1 6379 1 后面这个1 代表主机挂了，slave投票看谁接替成为主机，票数最多的，就会成为主机

启动哨兵：

redis-sentinel kconfig/sentinel.conf

master 节点断开了，这个时候就会从从机中随机选择一个服务器！（这里面有一个投票算法）

如果此时主机连接回来，只能归并到新的主机下当做从机

优点：

哨兵集群，基于主从复制模式，所有的主从配置优点，他都有

主从可以切换，故障可以转移，系统的可用性更强

哨兵模式就是主从模式的升级，手动到自动，更加简装

缺点

Redis不好在线扩容，集群的容量一旦到达上限，在线扩容十分麻烦

实现哨兵模式的配置其实很麻烦，里面有很多选择

18、缓存穿透和雪崩（面试高频，工作高用！）

缓存穿透

用户想要查询的数据在缓存和持久层数据库中都不存在，于是本次查询失败，当用户很多的时候会，缓存都没有命中（秒杀），于是都去请求了持久层数据库，这回给持久层数据库造成很大的压力，这时候就相当于出现了缓存穿透

解决方案：

布隆过滤器：

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

缓存击穿（量太大，缓存过期）

一个key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就被穿破缓存，直接请求数据库，就像在一个屏障上开了一个洞

解决方案：

设置热点数据永不过期

从缓存层面来看，没有设置过期时间，所以不会出现热点key过期后产生的问题

加互斥锁：

分布式锁：使用分布式锁，保证对于每个key同时只有一个线程取查询后端服务，其他线程没有获得分布式锁的权限，因此只需要等待即可，这种方式将高并发的压力转移到分布式锁，因此对分布式锁的考验很大

缓存雪崩

是指某一个时间段，缓存集中过期失效，或者redis宕机

产生雪崩的原因之一，双十一，比较热点的数据库都放在缓存当中，缓存时间一个小时之后，缓存数据过期，所有的查询都在持久层数据库上，数据库就会承受周期性的压力波峰，所有的请求都会在存储层，存储层的调用量会暴增，造成存储层也会挂掉的情况

解决方案

redis高可用

这个思想的含义是既然redis有可能挂掉，那我多增设几台redis，这样一台挂掉之后其他的还可以继续工作，其实就是搭建的集群（异地多活）

限流降级

在缓存失效的时候，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

数据预热

数据加热的含义就是在正式部署之前，我先把可能的数据先预先访问一遍，这样部分可能大量访问的数据就会加载到缓存中，在即将发生大并发访问前手动出发加载缓存不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀