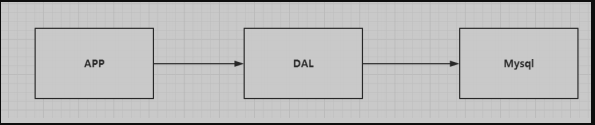
# Redis

# NoSqL概述

# 1.架构的演进

## 单机MySQL的年代



90年代 一个基本的网站访问量一般不会太大，单个数据库完全足够

那个时候，更多的去使用静态网页 Html ~ 服务器根本没有太大的压力！

思考一下，这种情况下：整个网站的瓶颈是什么？

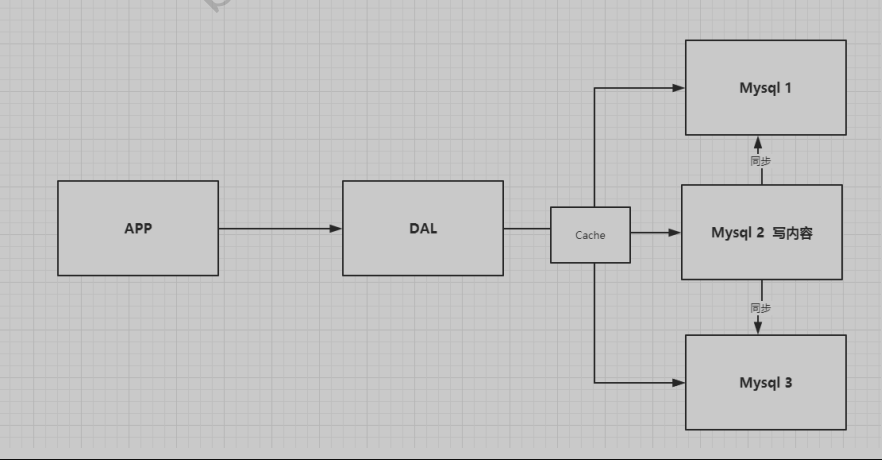
1 、数据量如果太大、一个机器放不下了！

2 、数据的索引 （B+ Tree），一个机器内存也放不下

3 、访问量（读写混合），一个服务器承受不了~

只要你开始出现以上的三种情况之一，那么你就必须要晋级！

## Memcached(缓存) + MySQL +垂直拆分(读写分离)



网站80%的情况都是在读取数据，每次都要查询数据库的话就十分的麻烦，所以我们希望减轻数据库的压力，我们可以使用缓存来保证效率。

发展过程：优化数据结构和索引--->文件缓存(IO)---->Memcached(当时热门技术)

## 分库分表+水平拆分+MySQL集群

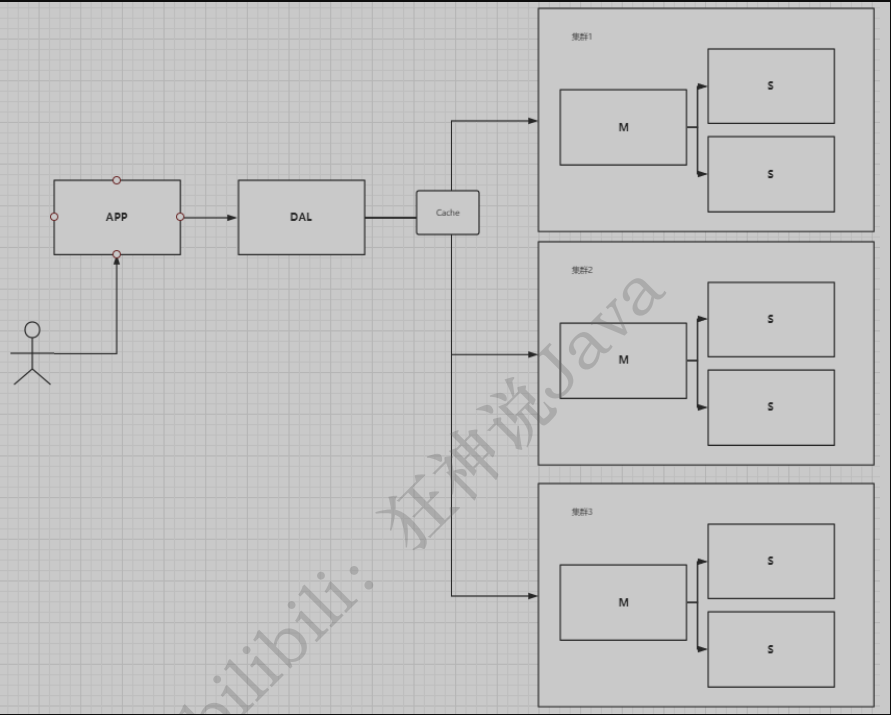
技术和业务在发展的同时，对人的要求也越来越高

本质：数据库（读+写）

早些年MyISAM 表锁 十分影响效率，高并发下就会出现严重的问题

早些年Innodb 行锁

慢慢的就开始使用分库分表解决写的压力，MySQL慢慢的退出表分区，这个并没有多少公司使用，还退出了MySQL的集群，有了集群就很好的满足了那个年代所有的需求



## 最近的年代

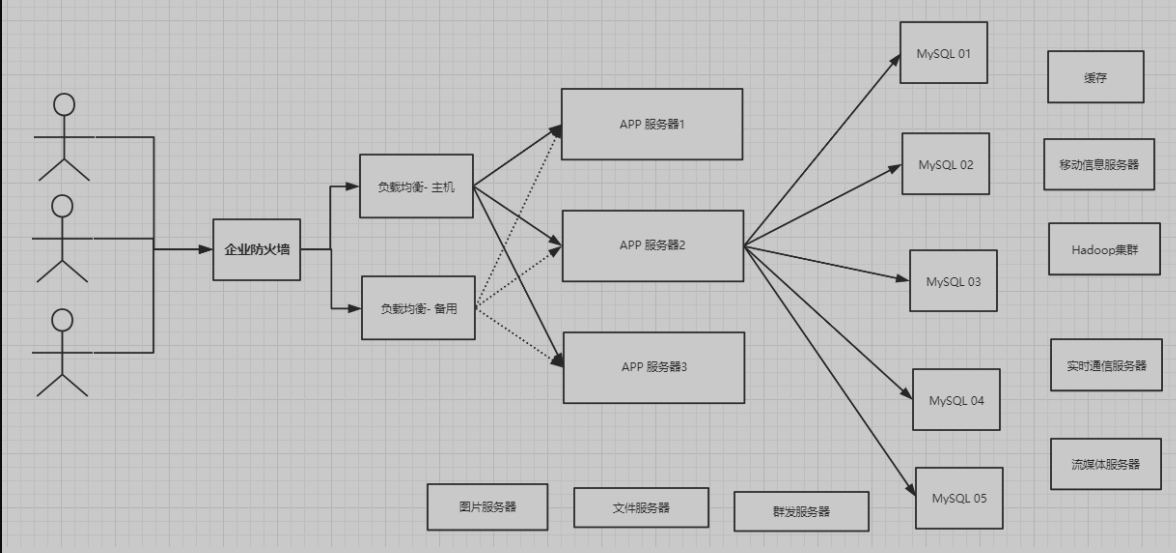
2010--2020 十年之间，世界已经发生了翻天覆地的变化；（定位，也是一种数据，音乐，热榜！）

MySQL 等关系型数据库就不够用了！数据量很多，变化很快~！

MySQL 有的使用它来村粗一些比较大的文件，博客，图片！数据库表很大，效率就低了！如果有一种数  
据库来专门处理这种数据,

MySQL压力就变得十分小（研究如何处理这些问题！）大数据的IO压力下，表几乎没法更大！

目前一个基本的互联网项目！



## 5.为什么要使用NoSQL

用户的个人信息，社交网络，地理位置。用户自己产生的数据，用户日志等等爆发式增长！

这时候我们就需要使用NoSQL数据库的，Nosql 可以很好的处理以上的情况！

# NoSQL

## 什么是NoSQL

NoSQL = Not Only SQL （不仅仅是SQL）

关系型数据库：表格 ，行 ，列

泛指非关系型数据库的，随着web2.0互联网的诞生！传统的关系型数据库很难对付web2.0时代！尤其  
是超大规模的高并发的社区！ 暴露出来很多难以克服的问题，NoSQL在当今大数据环境下发展的十分迅  
速，Redis是发展最快的，而且是我们当下必须要掌握的一个技术！

很多的数据类型用户的个人信息，社交网络，地理位置。这些数据类型的存储不需要一个固定的格式！  
不需要多月的操作就可以横向扩展的 ！ Map<String,Object> 使用键值对来控制！

## 2.NoSQL的特点

解耦！

1 、方便扩展（数据之间没有关系，很好扩展！）

2 、大数据量高性能（Redis 一秒写 8 万次，读取 11 万，NoSQL的缓存记录级，是一种细粒度的缓存，性能会比较高！）

3 、数据类型是多样型的！（不需要事先设计数据库！随取随用！如果是数据量十分大的表，很多人就无法设计了！）

## 3.NoSQL的四大分类

### 1.KV键值对：

新浪 Redis

美团 Redis + Tair

阿里，百度 Redis + memecache

### 文档型数据库（bson格式 和json一样）

MongoDB (一般必须掌握)

MongoDB 是一个基于分布式文件存储的数据库 C++编写 主要用来处理大量的文档

MongoDB 是一个介于关系型数据库和非关系型数据中中间的产品，MongoDB 是非关系型数据库中功能最丰富的，最想关系型数据库的

ConthDB

### 列存储数据库

Hbase 分布式文件系统

### 图关系型数据库

他不是存储图形，放的是关系，比如朋友圈社交网络，广告推荐

### 四者对比



# Redis入门

## redis是什么

Redis（Remote Dictionary Server )，即远程字典服务!

是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Val数据库，并提供多种语言的API。



redis会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，并且在此基础上实现了master-slave(主从)同步。

免费和开源！是当下最热门的 NoSQL 技术之一！也被人们称之为结构化数据库！

## Redis能干嘛？

1 、内存存储、持久化，内存中是断电即失、所以说持久化很重要（rdb、aof）

2 、效率高，可以用于高速缓存

3 、发布订阅系统

4 、地图信息分析

5 、计时器、计数器（浏览量！）

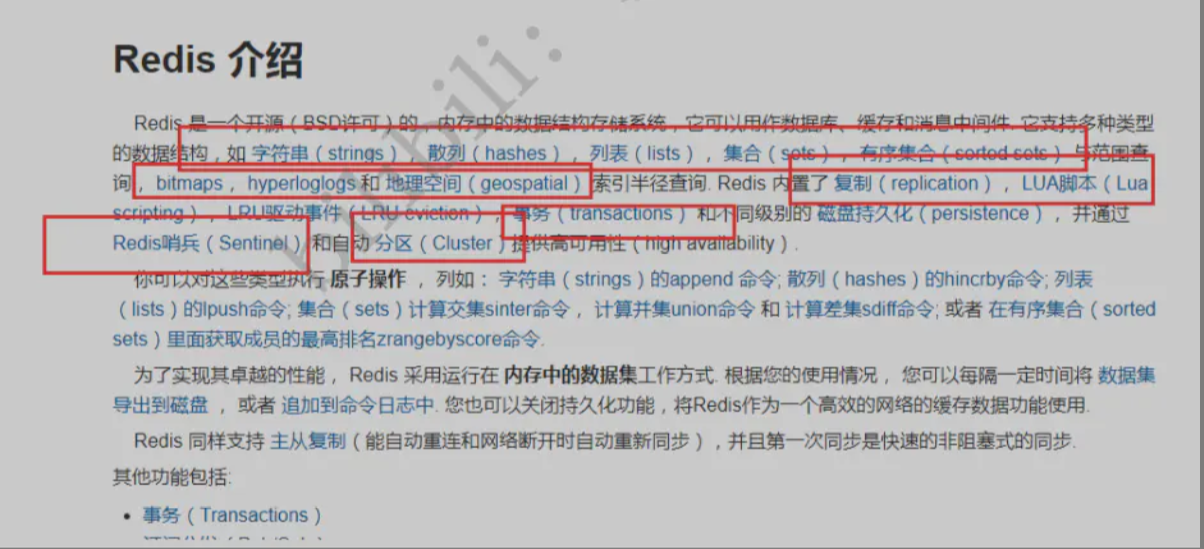
## 特性

1 、多样的数据类型

2 、持久化

3 、集群

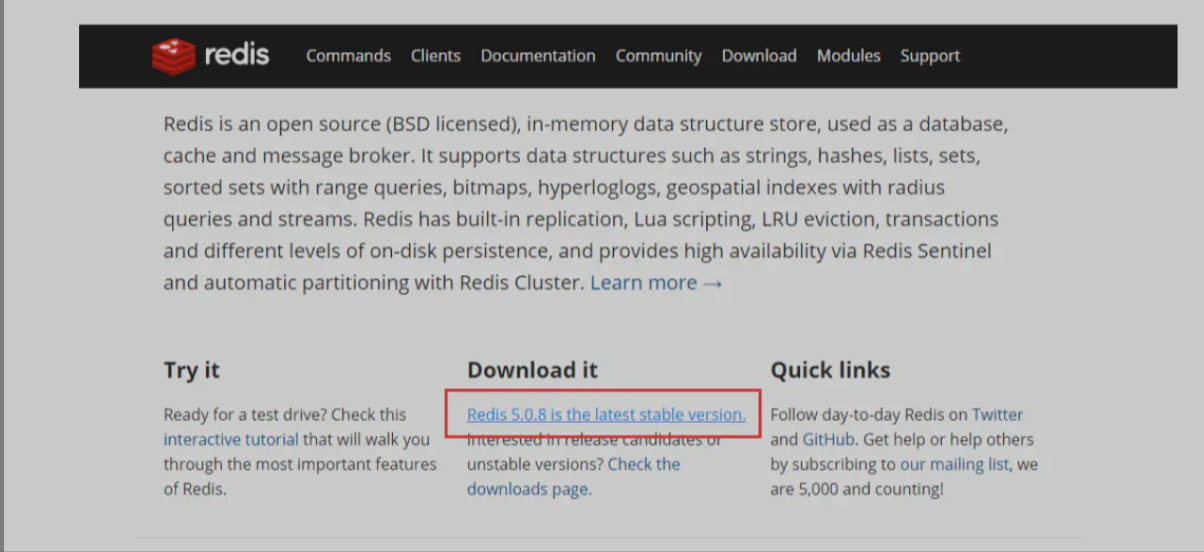
4 、事务



官网：[https://redis.io/](https://links.jianshu.com/go?to=https://redis.io/" \t "https://www.jianshu.com/p/_blank)

中文网：[http://www.redis.cn/](https://links.jianshu.com/go?to=http://www.redis.cn/" \t "https://www.jianshu.com/p/_blank)

下载地址：通过官网下载即可！



## Windows安装

1 、下载安装包：[https://github.com/dmajkic/redis/releases](https://links.jianshu.com/go?to=https://github.com/dmajkic/redis/releases" \t "_blank)

2 、下载完毕得到压缩包：

3 、解压到自己电脑上的环境目录下的就可以的！Redis 十分的小，只有5M

4 、开启Redis，双击运行服务即可！

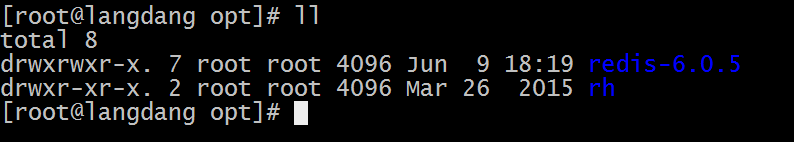
5 、使用redis客户单来来连接redis

记住一句话，Window下使用确实简单，但是Redis 推荐我们使用Linux去开发使用！

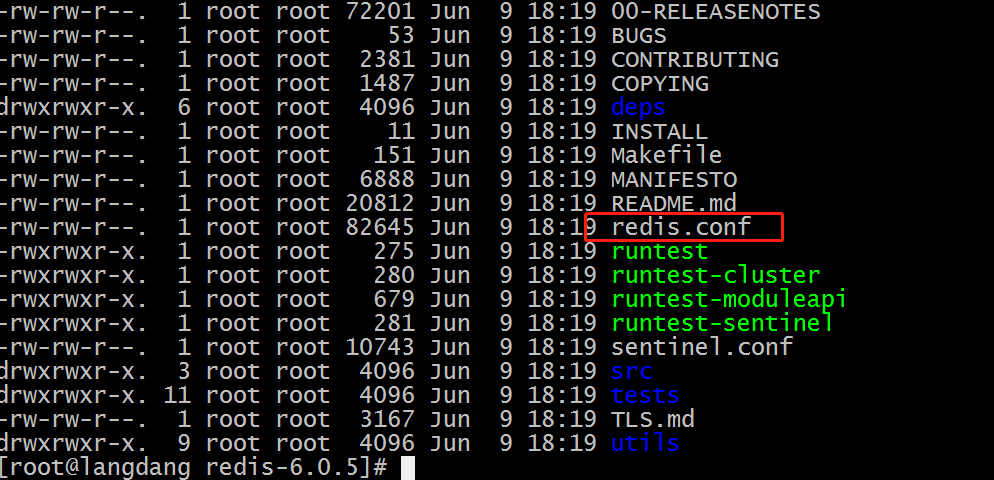
## 5.Linux安装

1 、下载安装包！ redis-5.0.8.tar.gz

2 、解压Redis的安装包！ 程序/opt



3 、进入解压后的文件，可以看到我们redis的配置文件



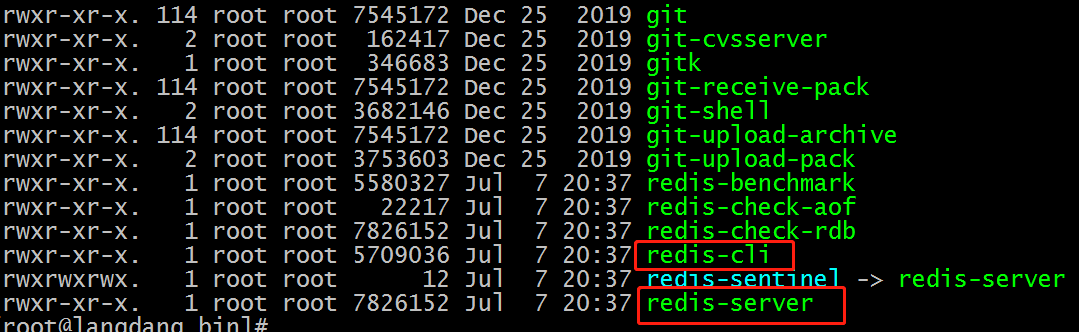
4 、基本的环境安装

yum install gcc-c++

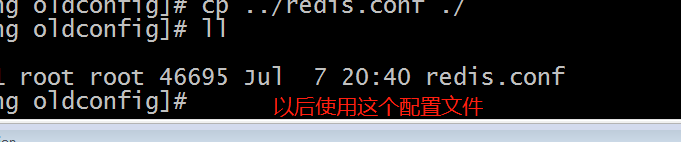
make

make install

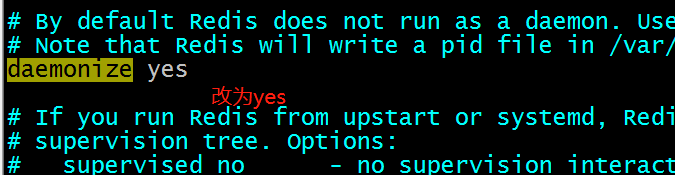
5 、redis的默认安装路径 /usr/local/bin



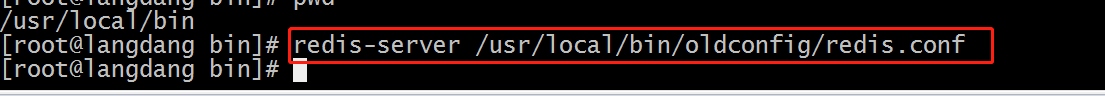
6 、将redis配置文件。复制到我们当前目录下



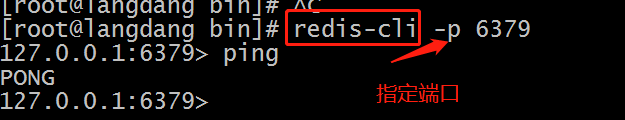
7 、redis默认不是后台启动的，修改配置文件！



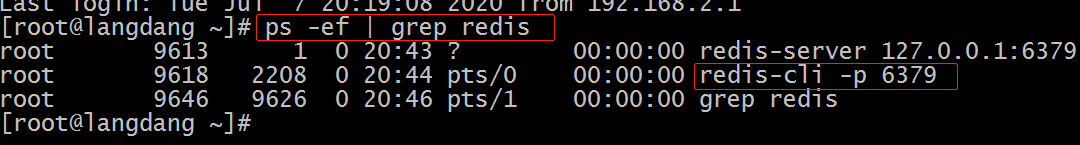
8 、启动Redis服务！



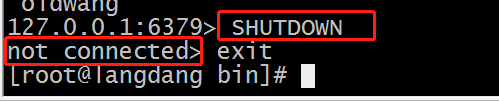
9 、使用redis-cli 进行连接测试！



10 、查看redis的进程是否开启！



11 、如何关闭Redis服务呢？ shutdown



# 4.Redis的使用

## 1.性能测试

redis-benchmark 是一个压力测试工具！官方自带的性能测试工具！

redis-benchmark 命令参数！

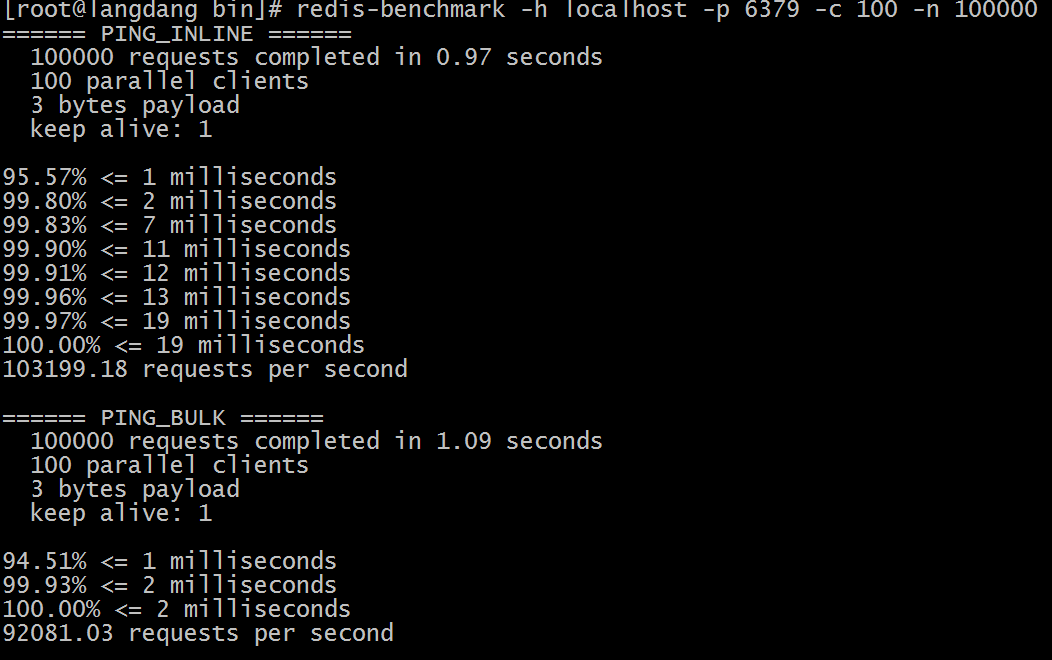


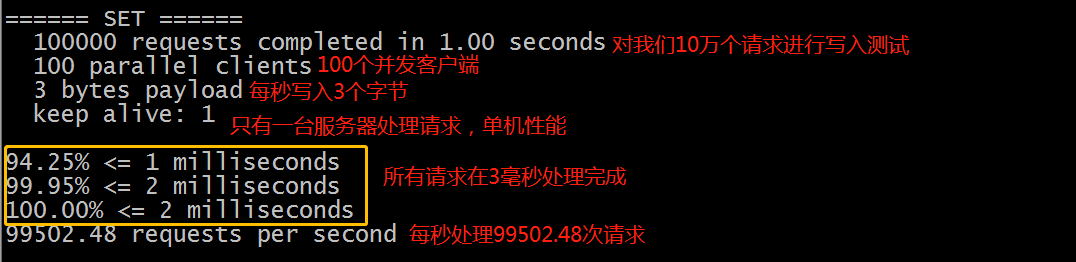
我们来简单测试下：

如何查看这些分析呢？

测试： 100 个并发连接 100000 请求

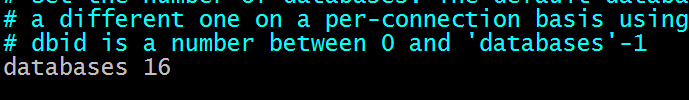
**redis-benchmark -h localhost -p 6379 -c 100 -n 100000**

**`**



## 2.基础知识

redis默认有 16 个数据库



默认使用的是第 0 个

可以使用 select 进行切换数据库！

|  |
| --- |
| 127.0.0.1:6379> select 3 切换数据库  127.0.0.1:6379[3]> DBSIZE 查看数据库大小  127.0.0.1:6379[3]> keys \* 查看数据库所有的key  127.0.0.1:6379[3]> FLUSHDB 清除当前数据库 flushdb  127.0.0.1:6379[3]> FLUSHALL 清除全部数据库的内容 |

为什么redis端口号是6379 粉丝效应

## 3.Redis为什么是单线程

明白Redis是很快的，官方表示，Redis是基于内存操作，CPU不是Redis性能瓶颈，Redi的瓶颈是根据机器的内存和网络带宽，既然可以使用单线程来实现，就使用单线程了！所有就使用了单线程了！

Redis 是C 语言写的，官方提供的数据为 100000+ 的QPS，完全不比同样是使用 key-vale的Memecache差！

## 4.为什么单线程这么快？

1 、误区 1 ：高性能的服务器一定是多线程的？

2 、误区 2 ：多线程（CPU上下文会切换！）一定比单线程效率高！

CPU>内存>硬盘的速度要有所了解！

核心：redis 是将所有的数据全部放在内存中的，所以说使用单线程去操作效率就是最高的，多线程（CPU上下文会切换：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上的，在内存情况下，这个就是最佳的方案！

# 五大基本数据类型

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件MQ。 它支持多种类型的数据结构，如 字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。 Redis 内置了 复制（replication），LUA脚本（Lua scripting）， LRU驱动事件（LRU eviction），事务（transactions） 和不同级别的 磁盘持久化（persistence）， 并通过Redis哨兵（Sentinel）和自动 分区（Cluster）提供高可用性（high availability）。

## RedisKey的基本命令

|  |
| --- |
| 127 .0.0.1:6379> keys \* 查看所有的key(empty list or set)  127 .0.0.1:6379> set name oldwang # set key  OK  127 .0.0.1:6379> keys \*  1 ) "name"  127 .0.0.1:6379> set age 1  OK  127 .0.0.1:6379> keys \*  1 ) "age"  2 ) "name"  127 .0.0.1:6379> EXISTS name 判断当前的key是否存在  (integer) 1  127 .0.0.1:6379> EXISTS name  (integer) 0  127 .0.0.1:6379> move name 1 移除当前的key  (integer) 1  127 .0.0.1:6379> keys \*  1 ) "age"  127 .0.0.1:6379> set name oldwang  OK  127 .0.0.1:6379> keys \*  1 ) "age"  2 ) "name"  127 .0.0.1:6379> get name"oldwang"  127 .0.0.1:6379> EXPIRE name 10 设置key的过期时间，单位是秒(integer) 1  127 .0.0.1:6379> ttl name 查看当前key的剩余时间  (integer) 4  127 .0.0.1:6379> ttl name  (integer) 3  127 .0.0.1:6379> ttl name  (integer) -  127 .0.0.1:6379> get name  (nil)  127 .0.0.1:6379> type name 查看当前key的一个类型！  string  127 .0.0.1:6379> type age  string |

## 2.String(字符串类型)详解

|  |
| --- |
| 127.0.0.1:6379> set key1 v1 设置值  OK  127.0.0.1:6379> get key1 获取值  "v1"  127.0.0.1:6379> keys \* 获取所有key  1) "key1"  127.0.0.1:6379> EXISTS key1 判断某一个key是否存在  (integer) 1  127.0.0.1:6379> APPEND key1 "hello" 追加字符串如果 key不存在就相当于set key  (integer) 7  127.0.0.1:6379> get key1  "v1hello"  127.0.0.1:6379> STRLEN key1 获取字符串的长度  (integer) 7  127.0.0.1:6379> APPEND key1 oldwang  (integer) 14  127.0.0.1:6379> STRLEN key1  (integer) 14  ----------------------------------------------------------------------------------------------------------------  127.0.0.1:6379> set views 0 初始值为0  OK  127.0.0.1:6379> get views  "0"  127.0.0.1:6379> INCR views 自增1  (integer) 1  127.0.0.1:6379> INCR views  (integer) 2  127.0.0.1:6379> get views  "2"  127.0.0.1:6379> DECR views 自减1  (integer) 1  127.0.0.1:6379> DECR views  (integer) 0  127.0.0.1:6379> get views  "0"  127.0.0.1:6379> INCRBY views 10 可以设置步长指定增量  (integer) 10  127.0.0.1:6379> INCRBY views 10  (integer) 20  127.0.0.1:6379> DECRBY views 5 减去步长  (integer) 15  ----------------------------------------------------------------------------------------------------------------  字符串范围 range  127.0.0.1:6379> set key1 "hello oldwang" 设置key1的值  OK  127.0.0.1:6379> GETRANGE key1 0 3 截取字符串，  "hell"  127.0.0.1:6379> GETRANGE key1 0 -1 获取全部字符串，和get key是一样的  "hello oldwang"  127.0.0.1:6379>  ----------------------------------------------------------------------------------------------------------------  替换字符串  127.0.0.1:6379> SETRANGE key2 1 xxx 替换指定位置开始的字符串  (integer) 7  127.0.0.1:6379> get key2  "axxxefg"  -------------------------------------------------------------------------------------------------------------  setex (set with expire) 设置过期时间  setnx (set if not exists) 不存在设置。 分布式锁中经常使用  127.0.0.1:6379> set key3 60 设置过期时间  OK  127.0.0.1:6379> SETEX key3 30 "hello" 设置key3的值为hello 30S过期  OK  127.0.0.1:6379> ttl key3  (integer) 23  127.0.0.1:6379> SETNX mykey "redis" 如果mykey不存在就创建  (integer) 1  127.0.0.1:6379> keys \*  1) "key1"  2) "mykey"  3) "key2"  127.0.0.1:6379> SETNX mykey "hello" 如果存在就创建失败  (integer) 0  127.0.0.1:6379> keys \*  1) "key1"  2) "mykey"  3) "key2"  127.0.0.1:6379>  -------------------------------------------------------------------------------------------------------------  批量设置 mset  批量过期 mget  127.0.0.1:6379> mset key1 v1 key2 v2 key3 v3 同时设置多个值  OK  127.0.0.1:6379> keys \*  1) "key1"  2) "key3"  3) "key2"  127.0.0.1:6379> mget key1 key2 key3 同时获取多个值  1) "v1"  2) "v2"  3) "v3"  127.0.0.1:6379> MSETNX key1 v1 k4 v4 msetnx是一个原子性操作  (integer) 0  127.0.0.1:6379> get k4  (nil)  127.0.0.1:6379>  设置对象  set user:1{name:zhangsan,age；3} 设置一个user:1对象 值为json字符串保存对象  这里的key是一个巧妙的设计 user:{id}:{filed},如此设计在redis中是ok的  127.0.0.1:6379> mset user:1:name zhangsan user:1:age 2  OK  127.0.0.1:6379> mget user:1:name user:1:age  1) "zhangsan"  2) "2"  -------------------------------------------------------------------------------------------------------------  getset 先获取在设置  127.0.0.1:6379> getset db redis 如果不存在返回nil  (nil)  127.0.0.1:6379> getset db redis  "redis"  127.0.0.1:6379> getset db mysql 如果存在值 获取原来的值，并设置新的值  "redis"  127.0.0.1:6379> getset db mysql  "mysql"  127.0.0.1:6379> |

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

统计多单位数量

粉丝数

对象缓存存储

## 3.List列表类型详解

基本的数据类型，列表

在redis中我们可以把List玩成 栈，队列，阻塞队列

所有的List命令都是L开头的

|  |
| --- |
| 127.0.0.1:6379> LPUSH list one 将一个值或多个值插入到列表的头部（左边）  (integer) 1  127.0.0.1:6379> LPUSH list two  (integer) 2  127.0.0.1:6379> LPUSH list three  (integer) 3  127.0.0.1:6379> LRANGE list 0 -1 获取所有值  1) "three"  2) "two"  3) "one"  127.0.0.1:6379> LRANGE list 0 1 通过区间获取具体的值  1) "three"  2) "two"  127.0.0.1:6379> RPUSH list right 将一个值或多个值插入到列表的尾部（右边）  (integer) 4  127.0.0.1:6379> LRANGE list 0 -1  1) "three"  2) "two"  3) "one"  4) "right"  LPOP  RPOP  127.0.0.1:6379> LRANGE list 0 -1  1) "three"  2) "two"  3) "one"  4) "right"  127.0.0.1:6379> LPOP list 移除List的第一个元素  "three"  127.0.0.1:6379> RPOP list 移除List的最后一个元素  "right"  127.0.0.1:6379>  127.0.0.1:6379> LINDEX list 1 通过下标获取List中某一个值  "one"  127.0.0.1:6379> LINDEX list 0  "two"  127.0.0.1:6379>  Llen  127.0.0.1:6379> LPUSH list one two three  (integer) 3  127.0.0.1:6379> LLEN list 获取List的长度  (integer) 3  127.0.0.1:6379>  移除指定的值  Lrem  127.0.0.1:6379> LREM list 1 one 移除List集合中指定个数的value 精确匹配  (integer) 1  127.0.0.1:6379> LRANGE list 0 -1  1) "three"  2) "three"  3) "two"  127.0.0.1:6379> LREM list 2 three  (integer) 2  127.0.0.1:6379>  127.0.0.1:6379> RPUSH list "hello"  (integer) 1  127.0.0.1:6379> RPUSH list "hello1"  (integer) 2  127.0.0.1:6379> RPUSH list "hello2"  (integer) 3  127.0.0.1:6379> RPUSH list "hello3"  (integer) 4  127.0.0.1:6379> LTRIM list 1 2 通过下标截取指定的长度，这个List已经改变了，只剩下截取的元素  OK  127.0.0.1:6379> LRANGE list 0 -1  1) "hello1"  2) "hello2"  127.0.0.1:6379>  RPOPLPUSH 移除列表的最后一个元素，移动到新的列表中  127.0.0.1:6379> RPUSH list "hello1"  (integer) 1  127.0.0.1:6379> RPUSH list "hello2"  (integer) 2  127.0.0.1:6379> RPUSH list "hello3"  (integer) 3  127.0.0.1:6379> RPOPLPUSH list mylist 移除列表的最后一个元素，移动到新的列表中  "hello3"  127.0.0.1:6379> LRANGE list 0 -1  1) "hello1"  2) "hello2"  127.0.0.1:6379> LRANGE mylist 0 -1   1. "hello3"   Lset 将列表中指定下标的值替换为另一个值   1. 127.0.0.1:6379> EXISTS list 判断是否存在 2. (integer) 0 3. 127.0.0.1:6379> LSET list 0 one 如果不存在更新就会保存 4. (error) ERR no such key 5. 127.0.0.1:6379> LPUSH list one 6. (integer) 1   127.0.0.1:6379> LSET list 0 two 如果存在更新当前下标的值   1. OK 2. 127.0.0.1:6379> LRANGE list 0 0 3. 1) "two" 4. 127.0.0.1:6379>   LInsert 将某个具体的value插入到列表中某个元素的前面或后面  127.0.0.1:6379> FLUSHALL  OK  127.0.0.1:6379> RPUSH list hello  (integer) 1  127.0.0.1:6379> RPUSH list world  (integer) 2  127.0.0.1:6379> LINSERT list before world other 插入到指定元素的前边  (integer) 3  127.0.0.1:6379> LRANGE list 0 -1  1) "hello"  2) "other"  3) "world"  127.0.0.1:6379> LINSERT list after hello new 插入到指定元素的后边  (integer) 4  127.0.0.1:6379> LRANGE list 0 -1  1) "hello"  2) "new"  3) "other"  4) "world"  127.0.0.1:6379> |

List实际上是一个列表 是Node before，after left right都可以插入

如果key不存在 创建新的列表

如果key存在 新增内容

如果移除了所以的值就是空链表 也代表不存在

在两边插入或者改动效率最高，中间元素相对来说效率会低一点

场景，消息队列(lpush rpop) 栈(lpush,lpop)

## Set集合类型详解

Set中的值不能重复

|  |
| --- |
| 127.0.0.1:6379> SADD list one set中添加元素  (integer) 1  127.0.0.1:6379> SADD list two  (integer) 1  127.0.0.1:6379> SMEMBERS list 查看set中所有值  1) "two"  2) "one"  127.0.0.1:6379> SISMEMBER list list 判断某一个值是否存在  (integer) 0  127.0.0.1:6379> SISMEMBER list one  (integer) 1  127.0.0.1:6379>  127.0.0.1:6379> SCARD list 获取set集合中的个数  (integer) 2  127.0.0.1:6379>  127.0.0.1:6379> SCARD list  (integer) 2  127.0.0.1:6379> SREM list one 移除集合中的元素  (integer) 1  127.0.0.1:6379> SCARD list  (integer) 1  127.0.0.1:6379> SMEMBERS list  1) "two"  127.0.0.1:6379>  Set 抽随机  127.0.0.1:6379> SMEMBERS list 随机抽选出一个元素  1) "two"  127.0.0.1:6379> SRANDMEMBER list  "two"  127.0.0.1:6379> SRANDMEMBER list 1 随机抽选一个元素  "two"  127.0.0.1:6379> SRANDMEMBER list  "two"  127.0.0.1:6379>  删除随机的key  127.0.0.1:6379> SMEMBERS list  1) "two"  2) "xx"  3) "oo"  127.0.0.1:6379> SPOP list 随机删除集合中的元素  "two"  127.0.0.1:6379> SMEMBERS list  1) "xx"  2) "oo"  127.0.0.1:6379> SPOP list  "oo"  127.0.0.1:6379>  将一个指定的值移动到另外一个集合  127.0.0.1:6379> SADD list1 set2  (integer) 1  127.0.0.1:6379> keys \*  1) "list1"  2) "list"  127.0.0.1:6379> SMOVE list list2 a 将一个指定的值移动到另外一个集合  (integer) 1  127.0.0.1:6379> SMEMBERS list2  1) "a"  127.0.0.1:6379>  共同关注（并集）  数字集合类  差集 SDIFF  交集 SINTER  并集 SUNION  127.0.0.1:6379> SADD k1 a  (integer) 1  127.0.0.1:6379> SADD k1 b  (integer) 1  127.0.0.1:6379> SADD k1 c  (integer) 1  127.0.0.1:6379> SADD k2 c  (integer) 1  127.0.0.1:6379> SADD k2 d  (integer) 1  127.0.0.1:6379> SADD k2 e  (integer) 1  127.0.0.1:6379> SDIFF k1 k2 差集  1) "b"  2) "a"  127.0.0.1:6379> SINTER k1 k2 交集  1) "c"  127.0.0.1:6379> SUNION k1 k2 并集  1) "b"  2) "c"  3) "a"  4) "e"  5) "d"  127.0.0.1:6379> |

微博，A用户将所有关注的人放在set集合中，将粉丝放在一个集合中，

场景 共同关注，共同爱好，二度好友，推荐好友

## Hash哈希类型详解

Map集合 key-<key value> 这时候这个值就是一个map集合 hash本质和string类型没有太大区别，还是一个简单的集合

|  |
| --- |
| 127.0.0.1:6379> HSET hash field1 old set一个具体的 key-value  (integer) 1  127.0.0.1:6379> HGET hash field1  "old"  127.0.0.1:6379> HMSET hash field1 wang field2 oldwang set多个key-value  OK  127.0.0.1:6379> HMGET hash field1 field2 获取多个字段的值  1) "wang"  2) "oldwang"  127.0.0.1:6379> HGETALL hash 获取hash中全部的数据  1) "field1"  2) "wang"  3) "field2"  4) "oldwang"  127.0.0.1:6379>  127.0.0.1:6379> HDEL hash field1 删除指定的key字段，对应的value值也就消失了  (integer) 1  127.0.0.1:6379> HGETALL hash  1) "field2"  2) "oldwang"  127.0.0.1:6379>  Hlen  127.0.0.1:6379> HGETALL hash  1) "field2"  2) "oldwang"  3) "f1"  4) "hello"  5) "f2"  6) "world"  127.0.0.1:6379> HLEN hash 获取hash表的字段数量  (integer) 3  127.0.0.1:6379>  127.0.0.1:6379> HEXISTS hash f1 判断hash指定字段是否存在  (integer) 1  127.0.0.1:6379> HEXISTS hash f3  (integer) 0  127.0.0.1:6379>  只获取所有的field  只获取所有的值value  127.0.0.1:6379> HKEYS hash 只获取所有的field  1) "field2"  2) "f1"  3) "f2"  127.0.0.1:6379> HVALS hash 只获取所有的值value  1) "oldwang"  2) "hello"  3) "world"  127.0.0.1:6379>  127.0.0.1:6379> hset hash field3 5  (integer) 1  127.0.0.1:6379> HINCRBY hash field3 1 指定增量  (integer) 6  127.0.0.1:6379> HINCRBY hash field3 -1  (integer) 5  127.0.0.1:6379> HSETNX hash field4 hello 如果不存在则可以设置  (integer) 1  127.0.0.1:6379> HSETNX hash field4 hellow 如果不存在不能设置  (integer) 0  127.0.0.1:6379> |

Hash存储变更数据 尤其是用户信息保存 经常变动的信息

## Zset有序集合详解

在set集合的基础上增加了一个值

|  |
| --- |
| 127.0.0.1:6379> ZADD set 1 one 添加一个值  (integer) 1  127.0.0.1:6379> ZADD set 2 two 3 three 添加多个值  (integer) 2  127.0.0.1:6379> ZRANGE set 0 -1  1) "one"  2) "two"  3) "three"  127.0.0.1:6379>  127.0.0.1:6379> ZADD salary 2500 xiaohong 添加用户  (integer) 1  127.0.0.1:6379> ZADD salary 5000 zhangsan  (integer) 1  127.0.0.1:6379> ZADD salary 50000 oldwang  (integer) 1  127.0.0.1:6379> ZRANGEBYSCORE salary -inf +inf 显示全部用户 从小到大  1) "xiaohong"  2) "zhangsan"  3) "oldwang"  127.0.0.1:6379> ZRANGEBYSCORE salary -inf +inf withscores 显示全部用户并且附带成绩  1) "xiaohong"  2) "2500"  3) "zhangsan"  4) "5000"  5) "oldwang"  6) "50000"  127.0.0.1:6379> ZRANGEBYSCORE salary -inf 2500 withscores 显示工资小于2500的员工升序  1) "xiaohong"  2) "2500"  127.0.0.1:6379>  127.0.0.1:6379> ZREVRANGE salary 0 -1 从大到小进行排序  1) "oldwang"  2) "zhangsan"  移除元素  127.0.0.1:6379> ZRANGE salary 0 -1  1) "xiaohong"  2) "zhangsan"  3) "oldwang"  127.0.0.1:6379> ZREM salary xiaohong 移除有序集合中的指定元素  (integer) 1  127.0.0.1:6379> ZRANGE salary 0 -1  1) "zhangsan"  2) "oldwang"  127.0.0.1:6379>  127.0.0.1:6379> ZCARD salary 获取有序集合中的个数  (integer) 2  127.0.0.1:6379>  区间计算  127.0.0.1:6379> ZADD set 1 a  (integer) 1  127.0.0.1:6379> ZADD set 2 b  (integer) 1  127.0.0.1:6379> ZADD set 3 c  (integer) 1  127.0.0.1:6379> ZCOUNT set 1 3 获取集合的区间个数  (integer) 3  127.0.0.1:6379> ZCOUNT set 1 2  (integer) 2  127.0.0.1:6379> |

案例 存储班级成绩表，工资表 带权重进排序，排行榜应用实现 TopN

# 6.三种特殊数据类型

## 1.Geospatial 地理位置

朋友的定位，附近的人，打车距离

Redis的Geo 在redis3.2版本就推出了，这个功能可以推算地理位置信息，

可以查一些数据 [http://www.jsons.cn/lngcodeinfo/0706D99C19A781A3/](https://links.jianshu.com/go?to=http://www.jsons.cn/lngcodeinfo/0706D99C19A781A3/" \t "https://www.jianshu.com/p/_blank)



官方文档：[https://www.redis.net.cn/order/3685.html](https://links.jianshu.com/go?to=https://www.redis.net.cn/order/3685.html" \t "https://www.jianshu.com/p/_blank)

有效的经度从-180度到180度。

有效的纬度从-85.05112878度到85.05112878度。

|  |
| --- |
| Getadd 添加地理位置 规则南北极无法添加，一般会通过java程序一次性导入  参数 key 值 (维度，精度，名称)  127.0.0.1:6379> GEOADD china:city 116.40 39.90 beijing  (integer) 1  127.0.0.1:6379> GEOADD china:city 121.47 31.23 shanghai  (integer) 1  127.0.0.1:6379> GEOADD china:city 106.50 29.53 chongqiang 114.05 22 shenzheng  (integer) 2  127.0.0.1:6379> GEOADD china:city 120.16 30.24 hangzhou 108.96 34.26 xian  (integer) 2  127.0.0.1:6379>  (error) ERR invalid longitude,latitude pair 39.900000,116.400000 超出范围  GEOPOS 命令 - 从key里返回所有给定位置元素的位置（经度和纬度）  127.0.0.1:6379> GEOPOS china:city beijing 获取指定的城市经度和纬度  1) 1) "116.39999896287918091"  2) "39.90000009167092543"  127.0.0.1:6379> GEOPOS china:city chongqiang  1) "106.49999767541885376"  2) "29.52999957900659211"  127.0.0.1:6379>  两个人之间的距离  GEODIST 命令 - 返回两个给定位置之间的距离  指定单位的参数 unit 必须是以下单位的其中一个：  m 表示单位为米。  km 表示单位为千米。  mi 表示单位为英里。  ft 表示单位为英尺。  127.0.0.1:6379> GEODIST china:city beijing shanghai 查看北京到上海的距离  "1067378.7564"  127.0.0.1:6379>  附近的人（获取所有附近的人地址，定位） 通过半径查询  GEORADIUS 命令 - 以给定的经纬度为中心， 找出某一半径内的元素  127.0.0.1:6379> GEORADIUS china:city 110 30 1000 km 获取以100 30这个经纬度为中心寻找1000KM的城市，  1) "chongqiang"  2) "xian"  3) "shenzheng"  4) "hangzhou"  127.0.0.1:6379> GEORADIUS china:city 110 30 300 km withdist 显示到中间的距离位置  (empty list or set)  127.0.0.1:6379> GEORADIUS china:city 110 30 500 km  1) "chongqiang"  2) "xian"  127.0.0.1:6379>  127.0.0.1:6379> GEORADIUS china:city 110 30 500 km withcoord 显示他人定位信息  1) 1) "chongqiang"  2) "341.9374"  3) 1) "106.49999767541885376"  2) "29.52999957900659211"  127.0.0.1:6379> GEORADIUS china:city 110 30 500 km withcoord withdist count 2 筛选指定的结果  1) 1) "chongqiang"  2) "341.9374"  3) 1) "106.49999767541885376"  2) "29.52999957900659211"  2) 1) "xian"  2) "483.8340"  3) 1) "108.96000176668167114"  2) "34.25999964418929977"  127.0.0.1:6379>  GEORADIUSBYMEMBER 命令 - 找出位于指定范围内的元素，中心点是由给定的位置元素决定  127.0.0.1:6379> GEORADIUSBYMEMBER china:city beijing 1000 km 找出位于指定元素周围的内容  1) "beijing"  2) "xian"  127.0.0.1:6379> GEORADIUSBYMEMBER china:city shanghai 400 km  1) "hangzhou"  2) "shanghai"  127.0.0.1:6379> GEOHASH 命令 - 返回一个或多个位置元素的 Geohash 表示 127.0.0.1:6379> GEOHASH china:city beijing 将二维的经纬度转为一维的字符串，两个字符串越接近，说明位置越接近  1) "wx4fbxxfke0"  127.0.0.1:6379> |

GEO原理：底层的实现原理其实就是Zset 所以我们可以使用Zset操作

|  |
| --- |
| 127.0.0.1:6379> ZRANGE china:city 0 -1 查看地图中全部元素  1) "chongqiang"  2) "xian"  3) "shenzheng"  4) "hangzhou"  5) "shanghai"  6) "beijing"  127.0.0.1:6379> ZREM china:city beijing 移除指定元素  (integer) 1  127.0.0.1:6379> ZRANGE china:city 0 -1  1) "chongqiang"  2) "xian"  3) "shenzheng"  4) "hangzhou"  5) "shanghai"  127.0.0.1:6379> |

## 2.Hyperloglog

简介：Redis2.8.9更新了Hyperloglog

Redis hyperLogLog 基数统计算法

优点：占用的内存是固定的2^64不同的元素的技术，只需要费12KB内存，如果要从内存角度比较的话HyperLogLog是首选

注意：0.81%的错误率

网页的UV 传统方式，set保存用户的ID，然后就可以统计Set中的元素数量作为标准判断

这种方式如果保存大量的数据，就会比较麻烦，我们的目的是为计数，而不是保存用户的ID

|  |
| --- |
| 127.0.0.1:6379> PFADD mykey a b c d e f g h j i 创建第一组元素  (integer) 1  127.0.0.1:6379> PFCOUNT mykey 统计基数数量  (integer) 10  127.0.0.1:6379> PFADD mykey2 i j z x c v b n m 创建第二组元素  (integer) 1  127.0.0.1:6379> PFCOUNT mykey2  (integer) 9  127.0.0.1:6379> PFMERGE mykey mykey2 合并两组元素  OK  127.0.0.1:6379> PFCOUNT mykey  (integer) 15  127.0.0.1:6379> |

## 3.Bitmap

位存储 统计用户信息，活跃，不活跃 两个状态都可以使用Bitmap

Bitmap位图 数据结构，都是二进制来进行记录，就只有0，和1两个状态

使用BitMaps记录打卡记录

|  |
| --- |
| 127.0.0.1:6379> SETBIT sign 0 1  (integer) 0  127.0.0.1:6379> SETBIT sign 1 0  (integer) 0  127.0.0.1:6379> SETBIT sign 2 0  (integer) 0  127.0.0.1:6379> SETBIT sign 3 1  (integer) 0  127.0.0.1:6379> SETBIT sign 4 1  (integer) 0  127.0.0.1:6379> SETBIT sign 5 1  (integer) 0  查看某一天是否打卡  127.0.0.1:6379> GETBIT sign 3  (integer) 1  127.0.0.1:6379> GETBIT sign 1  (integer) 0  统计操作 统计打卡天数  127.0.0.1:6379> BITCOUNT sign  (integer) 4 |

# Redis基本事务操作

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，在事务的执行过程中会按照顺序执行，一次性，顺序性，排他性，执行一系列的命令

Redis事务没有隔离级别的概念。所有的命令在事务中，并没有被执行，只有发起命令的时候才会执行Exec

Redis单条命令保证原子性 但是事务不保证原子性

|  |
| --- |
| Redis的事务：  开启事务（MULTI）  命令入队（其他命令）  执行事务（exec）  注意：Redis的事务是一次性的  127.0.0.1:6379> MULTI 开启事务  OK  127.0.0.1:6379> set k1 v1 命令入队  QUEUED  127.0.0.1:6379> set k2 v2 命令入队  QUEUED  127.0.0.1:6379> get k2 命令入队  QUEUED  127.0.0.1:6379> set k3 v3 命令入队  QUEUED  127.0.0.1:6379> exec 执行事务  1) OK  2) OK  3) "v2"  4) OK  127.0.0.1:6379>  127.0.0.1:6379> MULTI 开启事务  OK  127.0.0.1:6379> set k1 v1  QUEUED  127.0.0.1:6379> set k2 v2  QUEUED  127.0.0.1:6379> set k3 v4  QUEUED  127.0.0.1:6379> DISCARD 取消事务  OK  127.0.0.1:6379> get k4 事务队列中的命令都不会执行  (nil)  127.0.0.1:6379>  编译异常 命令有错 事务中所有的命令都不会执行  运行异常 如果事务队列中存在语法性错误，那么执行命令的时候，其他命令是可以正常执行，错误命令抛出异常  编译异常  127.0.0.1:6379> MULTI  OK  127.0.0.1:6379> set k1 v1  QUEUED  127.0.0.1:6379> set k2 v2  QUEUED  127.0.0.1:6379> set k3 v3  QUEUED  127.0.0.1:6379> GETSET k3 错误命令  (error) ERR wrong number of arguments for 'getset' command  127.0.0.1:6379> set k4 v4  QUEUED  127.0.0.1:6379> set k5 v5  QUEUED  127.0.0.1:6379> EXEC 执行事务报错了  (error) EXECABORT Transaction discarded because of previous errors.  127.0.0.1:6379> get k5 所有的命令都不会被执行  (nil)  127.0.0.1:6379>  运行异常  127.0.0.1:6379> set k1 "v1"  OK  127.0.0.1:6379> MULTI  OK  127.0.0.1:6379> INCR k1 会执行失败  QUEUED  127.0.0.1:6379> set k2 v2  QUEUED  127.0.0.1:6379> set k4 v3  QUEUED  127.0.0.1:6379> get k4  QUEUED  127.0.0.1:6379> EXEC  1) (error) ERR value is not an integer or out of range 虽然第一条命令保存了，但是依旧正常执行成功了  2) OK  3) OK  4) "v3"  127.0.0.1:6379> |

# Redis实现乐观锁

监控：watch （面试常问）

悲观锁：

很悲观，认为什么时候都可能出现问题，无论什么时候都加锁

乐观锁：

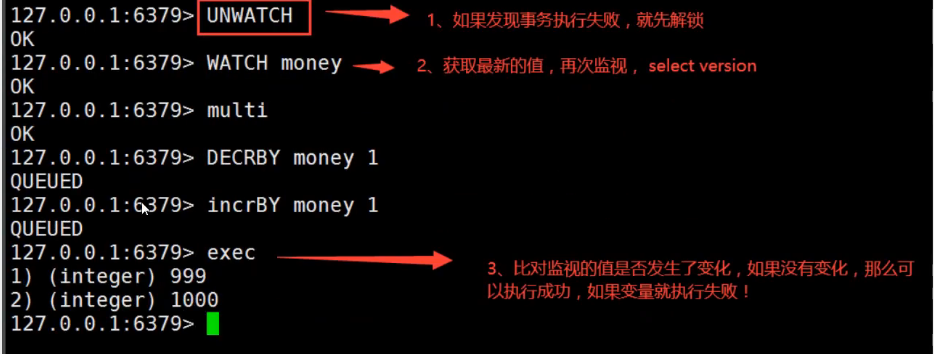
很乐观，认为什么都不会出现问题，所以不会加锁，更新数据的时候判断以下在此期间是否有人修改过这个数据。

获取version

更新的时候比较version

正常执行成功

|  |
| --- |
| 127.0.0.1:6379> set money 100  OK  127.0.0.1:6379> set out 0  OK  127.0.0.1:6379> WATCH money 监视money对象  OK  127.0.0.1:6379> MULTI 事务正常结束，数据期间没有变动，这个时候正常执行成功  OK  127.0.0.1:6379> DECRBY money 20  QUEUED  127.0.0.1:6379> INCRBY out 20  QUEUED  127.0.0.1:6379> EXEC  1) (integer) 80  2) (integer) 20  测试多线程修改值监视失败，使用watch当做redis的乐观锁操作  127.0.0.1:6379> WATCH money 监视money  OK  127.0.0.1:6379> MULTI  OK  127.0.0.1:6379> DECRBY money 10  QUEUED  127.0.0.1:6379> INCRBY out 10  QUEUED  127.0.0.1:6379> EXEC 执行之前另外一个线程修改我们的值这个时候就会导致食物执行失败  (nil)  如果修改失败获取最新的值就行 |



# Jedis操作Redis

我们要使用java来操作redis

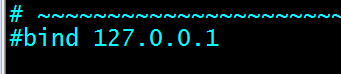
Jedis是redis官方推荐的java连接开发工具，使用java操作Redis中间件，

导入依赖

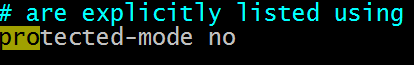
|  |
| --- |
| <dependencies>  <!-- 导入jedis的依赖包 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>  <!--导入fastjson-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.28</version>  </dependency> </dependencies> |

注意：Windows上连接Linux上的Redis需要对配置文件进行修改

１.修改redis.conf配置文件，将绑定的ip地址端口号给注释见下图



1. 由于Linux上的redis处于安全保护模式，这就让你无法从虚拟机外部去轻松建立连接， 这里就有两种解决方法，一种是在redis.conf中设置保护模式为no，见下图



编码测试：

连接数据库

|  |
| --- |
| //测试连接  Public static void main(String[] args) {  //1.new Jedis对象  Jedis jedis = new Jedis("192.168.2.110",6379);  //测试连接  System.out.println(jedis.ping()); } |

常用API

Key相关操作

|  |
| --- |
| */*\*\*  \* redis key相关操作  \*/ public class RedisKeyOperation {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  System.out.println(jedis.ping());  System.out.println("清空数据" + jedis.flushDB());  System.out.println("判断某个键是否存在" + jedis.exists("username"));  System.out.println("新增<'username','oldwang'>的键值对" + jedis.set("username","oldwang"));  System.out.println("新增<'password','password'>的键值对" + jedis.set("password","password"));  System.out.println("系统中所有的键如下");  Set<String> keys = jedis.keys("\*");  System.out.println(keys);  System.out.println("删除password" + jedis.del("password"));  System.out.println("判断password是否存在" + jedis.exists("password"));  System.out.println("查看username所存储的值的类型" + jedis.type("username"));  System.out.println("随机返回key空间一个" + jedis.randomKey());  System.out.println("重命名key" + jedis.rename("username","name"));  System.out.println("取出改名后的name" + jedis.get("name"));  System.out.println("按索引查询" + jedis.select(0));  System.out.println("删除当前选中数据库所有的key" + jedis.flushDB());  System.out.println("返回当前数据库中key的数目" + jedis.dbSize());  System.out.println("删除所有数据库中的所有key" + jedis.flushAll());  } } |

String相关操作

|  |
| --- |
| */*\*\*  \* String相关操作  \*/ public class RedisStringOperation {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  jedis.flushDB();  System.out.println("================增加数据===============");  System.out.println(jedis.set("key1","value1"));  System.out.println(jedis.set("key2","value2"));  System.out.println(jedis.set("key3","value3"));  System.out.println("删除key2" + jedis.del("key2"));  System.out.println("获取key2" + jedis.get("key2"));  System.out.println("修改key1的值" + jedis.set("key1","oldwangchange"));  System.out.println("在key3后边加入值"+ jedis.append("key3","append"));  System.out.println("key3的值"+jedis.get("key3"));  System.out.println("增加多个键值对" + jedis.mset("key01","value01","key02","value02"));  System.out.println("获取多个键值对" + jedis.mget("key01","key02"));  System.out.println("删除多个键值对" + jedis.del("key01","key02"));  System.out.println("获取多个键值对" + jedis.mget("key01","key02"));   jedis.flushDB();  System.out.println("====================新增键值对防止覆盖原来值===============");  System.out.println(jedis.setnx("key1","value1"));  System.out.println(jedis.setnx("key2","value2"));  System.out.println(jedis.setnx("key2","value-new"));  System.out.println(jedis.get("key1"));  System.out.println(jedis.get("key2"));   System.out.println("====================新增键值设置有效时间===============");  System.out.println(jedis.setex("key3",2,"value3"));  System.out.println(jedis.get("key3"));  try {  TimeUnit.SECONDS.sleep(3);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println(jedis.get("key3"));   System.out.println("====================获取原值，更新新值===============");  System.out.println(jedis.getSet("key2","key2GetSet"));  System.out.println(jedis.get("key2"));  System.out.println("获取key的字符串" + jedis.getrange("key2",2,4));  } } |

List操作

|  |
| --- |
| */*\*\*  \* List相关操作  \*/ public class RedisListOperation {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  jedis.flushDB();   System.out.println("======添加List=====");  jedis.lpush("collections","ArrayList","Vector","stack");  jedis.lpush("collections","HashSet");  jedis.lpush("collections","TreeSet");  jedis.lpush("collections","TreeMap");  System.out.println("collections的内容" + jedis.lrange("collections",0,-1));  System.out.println("collections区间0-3的元素" + jedis.lrange("collections",0,3));  System.out.println("==============================");  //删除列表指定的值，第二个参数为删除的个数（有重复时） 后add进去的元素先被删除，类似于出栈  System.out.println("删除指定元素个数" + jedis.lrem("collections",2,"HashSet"));  System.out.println("删除0-3之间之外的元素" + jedis.ltrim("collections",0,3));  System.out.println("collections列表出栈(左端)" + jedis.lpop("collections"));  } } |

其余API与所学命令一致

# 通过Jedis再次理解事务