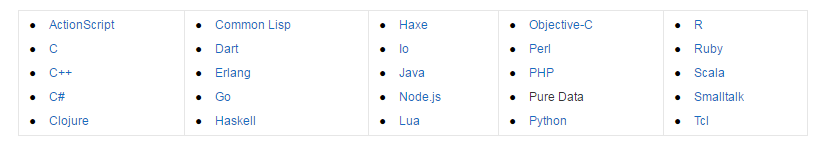
**Redis**

**版本：V 1.0**

# Redis简介

是一个开源的使用**C**语言编写，支持网络，可基于内存亦可持久化的日志key-value**数据库[非关系型数据库]**。并支持多种语言的**API**。 注意：非关系型数据库 mysql，oracle，sqlserver,db2关系型数据库。

**支持的语言：**



## Redis特性

**Redis远程的**：分为客户端，服务端。可以分别部署在不同的机器上，通过自定义协议进行传输和交互的。平时说的Redis通常指的是Redis的服务端。

**Redis基于内存的**：所有数据结构存在内存中。所有操作非常高速。性能优越于硬盘存储的mysql，因为存在内存中，所有也是比较吃内存的。

**Redis 非关系型数据库**：本质是数据库，存储数据，区别于Mysql。

关系型数据库在存储之前，必须要定义好所谓的数据字典，后续的存储数据按照存储字典来存储，而Redis就不需要了。

## Redis应用场景

1.**缓存：**当系统的接口数据比较慢的时候，可以把系统数据接口的数据缓存起来，当下次取的时候，可以直接从缓存中取就可以了。

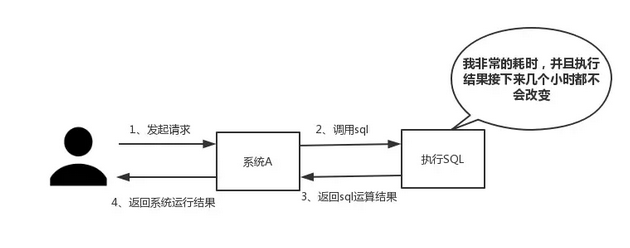
2.**数据存储：**redis有两种非常完备的持久化机制【**AOF和RDB**】，可以定期将数据持久化硬盘中，保障数据的完整性，安全性。

|  |
| --- |
| <https://db-engines.com/en/ranking> 使用排行榜 |

# 为什么使用redis

## 性能

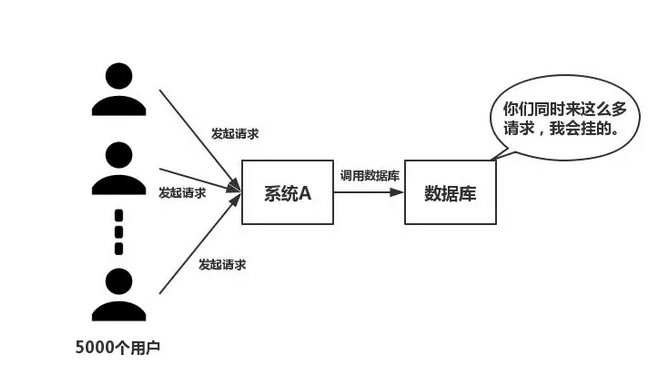
如下图所示，我们在碰到需要执行耗时特别久，且结果不频繁变动的SQL，就特别适合将运行结果放入缓存。这样，后面的请求就去缓存中读取，使得请求能够迅速响应。



|  |
| --- |
| 什么叫迅速：  ”在理想状态下，我们的页面跳转需要在瞬间解决，对于页内操作则需要在刹那间解决。另外，超过一弹指的耗时操作要有进度提示，并且可以随时中止或取消，这样才能给用户**最好的体验**。” 那么瞬间、刹那、一弹指具体是多少时间呢？ 根据《摩诃僧祗律》记载  一刹那者为一念，二十念为一瞬，二十瞬为一弹指，二十弹指为一罗预，二十罗预为一须臾，一日一夜有三十须臾。  那么，经过周密的计算，一瞬间为0.36 秒,一刹那有 0.018 秒.一弹指长达 7.2 秒。 |

## 并发

如下图所示，在大并发的情况下，所有的请求直接访问数据库，数据库会出现连接异常。这个时候，就需要使用redis做一个缓冲操作，让请求先访问到redis，而不是直接访问数据库。



## 单线程redis为什么这么快？

(一)纯内存操作  
(二)单线程操作，避免了频繁的上下文切换  
(三)采用了非阻塞I/O**多路复用**机制

多路复用案例：

小明在紫禁城开了一家快递店，负责同城快送服务。小明因为资金限制，雇佣了一批快递员，然后小明发现资金不够了，只够买一辆车送快递。

经营方式一

客户每送来一份快递，小明就让一个快递员盯着，然后快递员开车去送快递。慢慢的小明就发现了这种经营方式存在下述问题

1、几十个快递员基本上时间都花在了抢车上了，大部分快递员都处在闲置状态，谁抢到了车，谁就能去送快递

2、随着快递的增多，快递员也越来越多，小明发现快递店里越来越挤，没办法雇佣新的快递员了

3、快递员之间的协调很花时间。

经营方式二

小明只雇佣一个快递员。然后呢，客户送来的快递，小明按送达地点标注好，然后依次放在一个地方。最后，那个快递员依次的去取快递，一次拿一个，然后开着车去送快递，送好了就回来拿下一个快递

上述两种经营方式对比，是不是明显觉得第二种，效率更高，更好呢。在上述比喻中:

每个快递员——————>每个线程

每个快递——————–>每个socket(I/O流)

快递的送达地点————–>socket的不同状态

客户送快递请求————–>来自客户端的请求

小明的经营方式————–>服务端运行的代码

一辆车———————->CPU的核数

结论：

经营方式一就是传统的并发模型，每个I/O流(快递)都有一个新的线程(快递员)管理。

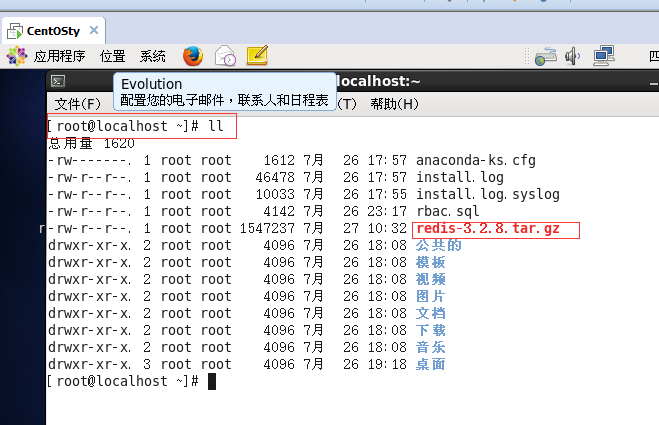
经营方式二就是I/O多路复用。只有单个线程(一个快递员)，通过跟踪每个I/O流的状态(每个快递的送达地点)，来管理多个I/O流。

面试的时候：

将复杂的东西简单化：生活化！

# Redis安装

## 导入redis的压缩包，root目录即可！



## 解压(解压到当前目录即可/root)

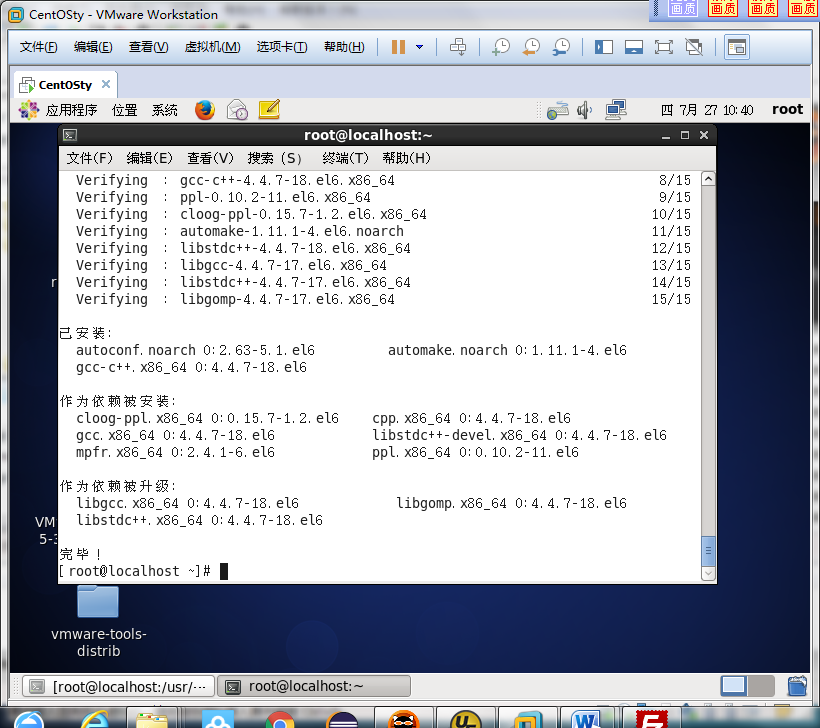
tar -zxvf redis-3.2.8.tar.gz

## 要添加依赖。（Redis是使用c语言编写的。）在root目录下添加。

yum install gcc-c++ automake autoconf,一般情况下只添加 gcc-c++即可。automake autoconf是为了确保不会出现其他问题而添加的依赖。

yum :最好在有外网的情况下安装使用

如果环境不允许，使用本地yum源。



## 创建安装目录

/opt 存在临时文件

/usr/local 安装应用程序的目录

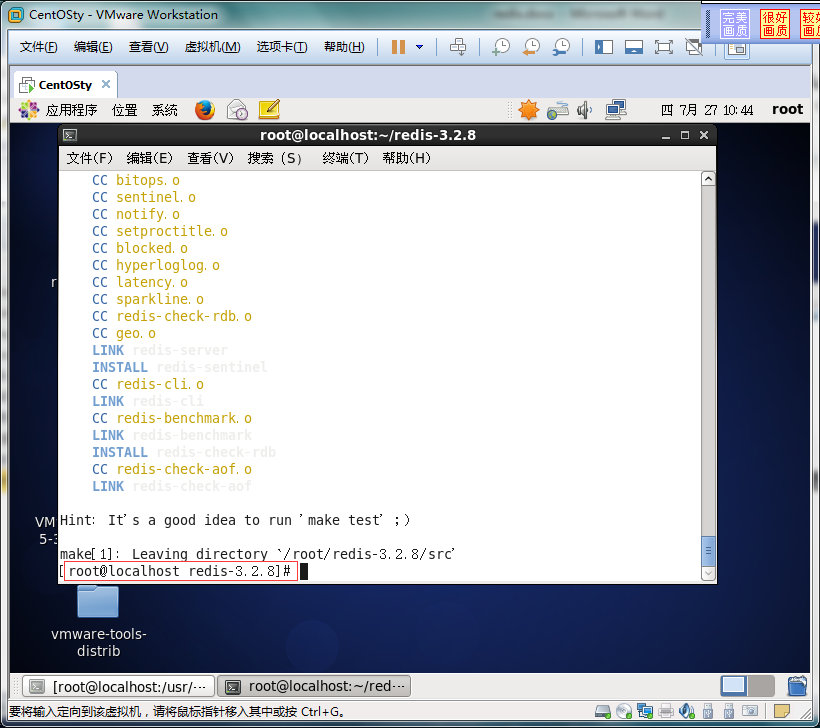
mkdir -p /usr/local/redis

## 进行编译：在解压后的redis 文件夹中，使用make 命令。

# cd /root/redis-3.2.8

# make

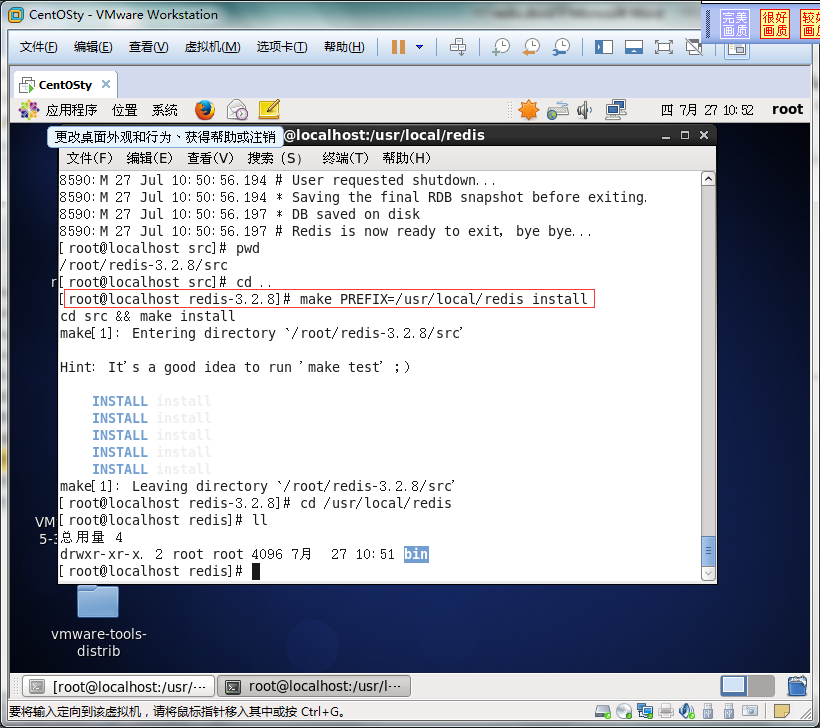
使用make 命令进行编译 :



## 进行安装：

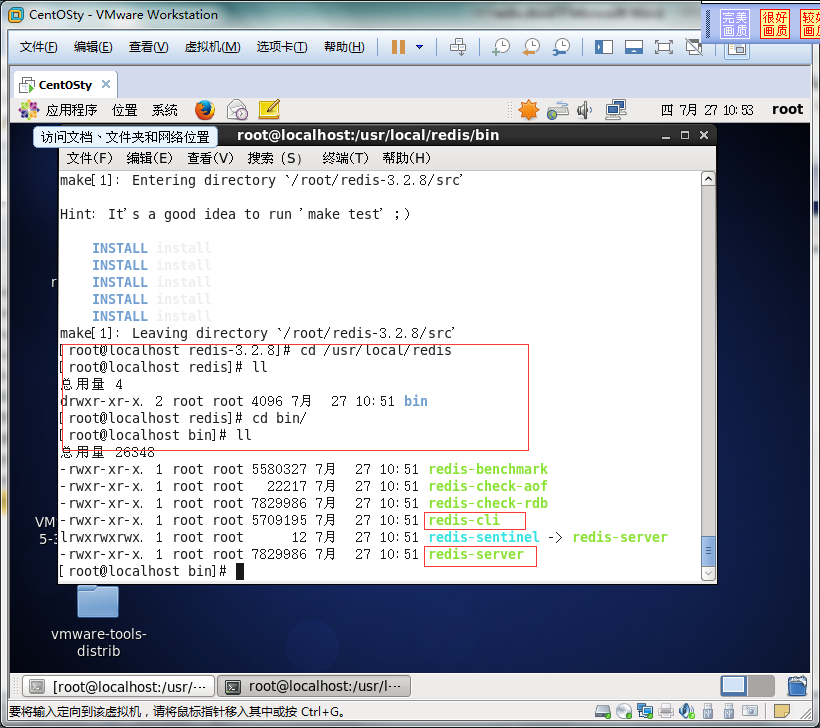
使用make install 安装路径 /usr/loca/redis

make install **PREFIX**=/usr/local/redis install 放在前后没有任何影响。



## 进入安装目录

cd /usr/local/redis 目录下会一个bin文件夹



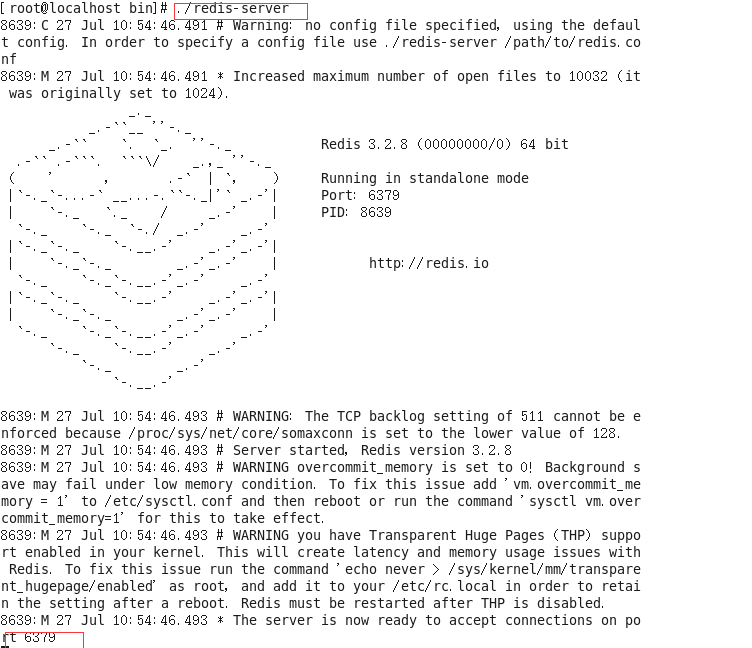
redis-cli :表示redis的客户端

redis-server :表示redis的服务器

## 开始启动服务

./redis-server 启动服务

Port 6379 :redis 默认的端口号



启动服务后则该窗口什么也不能做了！

因为redis启动方式默认前台启动。为了是窗口可以用，可以设置redis 为后台启动。

# redis 配置

redis.conf 配置文件

bind 绑定那个ip地址

protected-mode yes 保护模式是否开启

port 6379 端口号

tcp-backlog 511 确定了TCP连接中已完成队列

timeout 0 设置客户端空闲超时时间，服务端不会主动断开连接，不能小于0。

tcp-keepalive 300 每个一段时间300秒发送一次请求.看是否还活着

daemonize yes 是否在后台执行！

supervised no无监督交互

pidfile /var/run/redis\_6379.pid 指定存储Redis进程号的文件路径

loglevel notice 日志级别 notice (仅试用于生产)

logfile "" 配置 log 文件地址,默认打印在命令行终端的窗口上，也可设为/dev/null屏蔽日志

databases 16 指定数据库个数

maxclient 最大客户端连接数

Maxmemory ：

设置Redis可以使用的内存量。一旦到达内存使用上限，Redis将会试图移除内部数据，移除规则 可以通过maxmemory-policy来指定。如果Redis无法根据移除规则来移除内存中的数据，或者设置了“不允许移除”，

那么Redis则会针对那些需要申请内存的指令返回错误信息，比如SET、LPUSH等。

Maxmemory-policy noeviction 默认配置

**（1）volatile-lru：使用LRU算法移除key，只对设置了过期时间的键**

**（2）allkeys-lru：使用LRU算法移除key**

**（3）volatile-random：在过期集合中移除随机的key，只对设置了过期时间的键**

**（4）allkeys-random：移除随机的key**

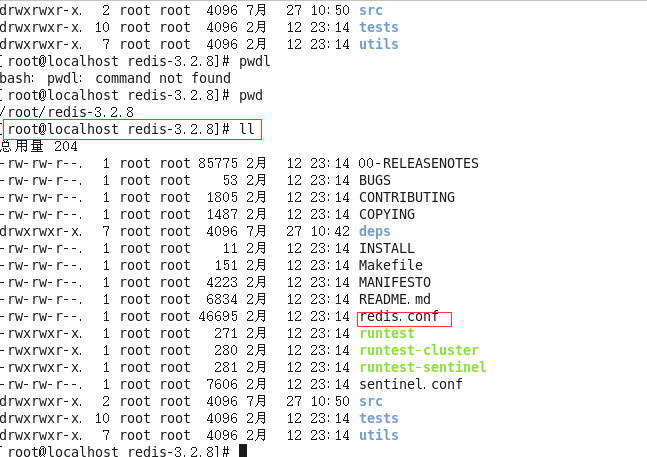
**（5）volatile-ttl：移除那些TTL值最小的key，即那些最近要过期的key**

**（6）noeviction：不进行移除。针对写操作，只是返回错误信息**

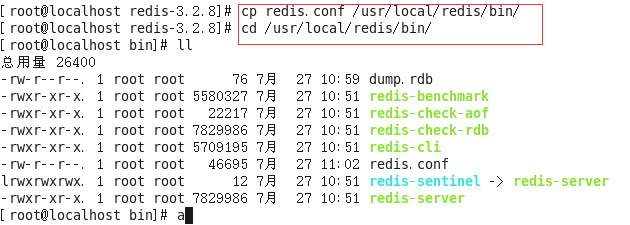
cluster-enabled yes 是否开启集群

## 设置redis后台启动

先使用ctrl+c退出前台启动

1. 回到redis的解压目录去找redis.conf配置文件
2. 
3. 找到之后，拷贝到安装目录

cp redis.conf /usr/local/redis/bin



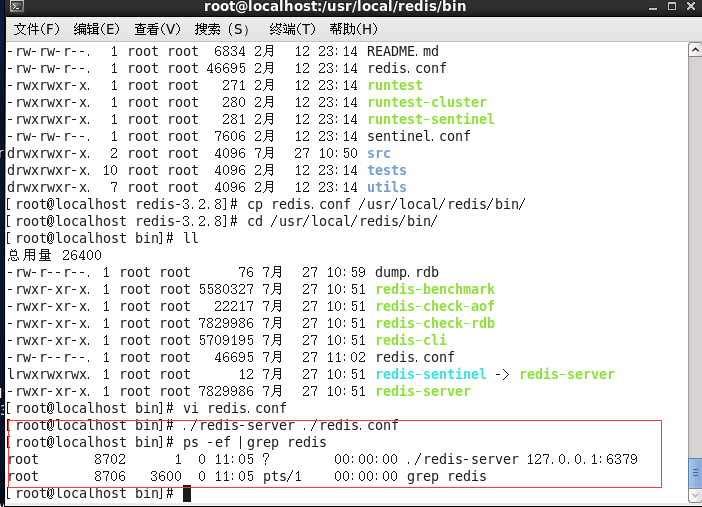
要修改redis.conf 配置文件

daemonize no – daemonize yes 此属性表示设置redis的启动方式。

no表示前台启动，yes表示后台启动。

修改完成之后，再从新启动redis

./redis-server ./redis.conf



如果将redis的进程kill 掉。 则再去找 redis的服务进程。就找不到了！



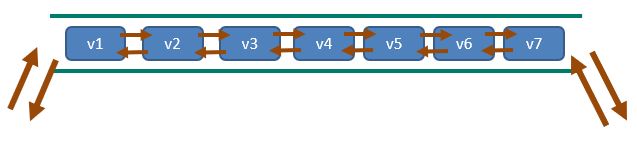
# Redis的五种数据类型

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。 Map key-value
  1. String str = “hello”; redis key – value String s = new String(“”);
* String类型是二进制安全的。意味着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，一个Redis中字符串value最多可以是512M.

List：

* 单键多值
* Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素导列表的头部（左边）或者尾部（右边）。
* 它的底层实际是个**双向链表，对两端的操作性能很高，通过索引下标的操作中间的节点性能会较差。**

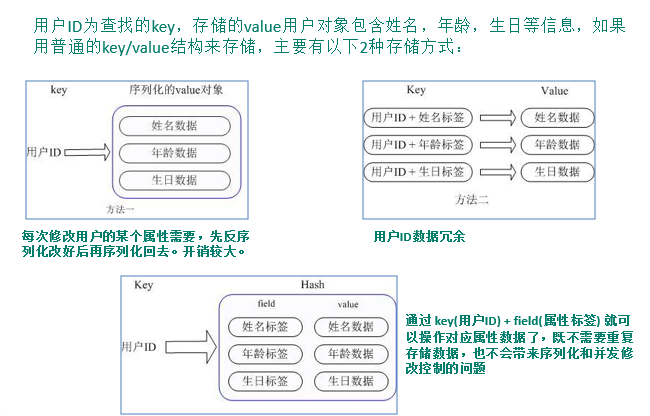


Set：

* Redis set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以**自动排重的**，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择。

Hash：

* Redis hash 是一个键值对集合。
* 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。
* 类似Java里面的Map<String,Object>



Zset：

* Redis有序集合zset与普通集合set非常相似，是一个没有重复元素的字符串集合。不同之处是有序集合的没有成员都关联了一个**评分（score）** ，这个**评分（score）**被用来按照从最低分到最高分的方式排序集合中的成员。集合的成员是唯一的，但是评分可以是重复了 。
* 因为元素是有序的, 所以你也可以很快的根据评分（score）或者次序（position）来获取一个范围的元素。访问有序集合的中间元素也是非常快的,因此你能够使用有序集合作为一个没有重复成员的智能列表。

使用场景：

|  |  |  |
| --- | --- | --- |
| 数据类型 | | 项目中的应用 |
| String | 字符串 | 比如说当一个ip地址访问网站超过了预定的次数，可以禁止访问，则这个预定次数就可以使用String来存储 |
| List | 列表 | 实现最新消息信息排列展示【消息队列】 |
| Set | 集合 | 特殊之处在于set是可以**自动排重**的。比如在微博应用中，每个人的好友存在一个集合（set）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。 |
| Hash | 散列 | 存储用户信息：key(用户ID) + field(属性标签) 操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题 |
| Zset | 有序集合 | 以某个条件为权重，比如按顶的次数排序。  需要精准设定过期时间的应用  使用sorted set的设置过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据。 |

# 常用命令：

## 非数据类型常用命令

1. 切换库
   1. select 0
2. 启动服务器，用客户端访问
   1. ./redis-server redis.conf
   2. ./redis-cli
3. 测试验证连接是否正常
   1. ping
4. 查看当前库的所有键
   1. keys \*
5. 判断key是否存在
   1. **exists <key>**
6. 查看键的类型
   1. type <key>
7. 删除某个键
   1. del <key>
8. 设置key的过期时间单位为 秒
   1. **expire** <key> <seconds>
9. 查看还有多少秒过期，-1表示永不过期，-2表示已过期
   1. **ttl <key>**
10. 查看当前数据库的key的数量
    1. dbsize
11. 清空当前库
    1. **flushdb**
12. 通杀全部库
    1. flushall

## String 类型常用命令

1. 添加键值对
   1. **set <key>**
2. 查询对应键值
   1. **get <key>**
3. 将给定的<value> 追加到原值的末尾
   1. append <key> <value>
4. 获得值的长度
   1. strlen <key>
5. 只有在 key 不存在时设置 key 的值
   1. setnx <key> <value>
6. 将 key 中储存的数字值增1,只能对数字值操作，如果为空，新增值为1
   1. incr <key>
7. 将 key 中储存的数字值减1,只能对数字值操作，如果为空，新增值为-1
   1. decr <key>
8. 将 key 中储存的数字值增减。自定义步长。
   1. **incrby / decrby** <key> <步长>
9. 同时设置一个或多个 key-value对
   1. mset <key1> <value1> <key2> <value2> .....
10. 同时获取一个或多个 value
    1. mget <key1> <key2> <key3> .....
11. 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。
    1. msetnx <key1> <value1> <key2> <value2> .....
12. 获得值的范围
    1. getrange <key> <起始位置> <结束位置>
13. 用 <value> 覆写<key> 所储存的字符串值，从<起始位置>开始。
    1. setrange <key> <起始位置> <value>
14. 设置键值的同时，设置过期时间，单位秒。
    1. **setex <key> <过期时间> <value>**
15. 以新换旧，设置了新值同时获得旧值
    1. getset <key> <value>

## List 类型常用命令

1. 从左边/右边插入一个或多个值。
   1. **lpush/rpush <key> <value1> <value2> <value3> ....**
2. 从左边/右边吐出一个值。值在键在，值光键亡
   1. **lpop/rpop <key>**
3. 按照索引下标获得元素(从左到右) 元素不会丢失！
   1. lrange <key> <start> <stop>
4. 获得列表长度
   1. llen <key>
5. 在<value>的后面插入<newvalue> 插入值
   1. linsert <key> before <value> <pivot> <newvalue>

## Set 类型常用命令

1. 将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略。
   1. **sadd <key> <value1> <value2> .....**
2. 取出该集合的所有值。
   1. **smembers <key>**
3. 判断集合<key>是否为含有该<value>值，有返回1，没有返回0
   1. sismember <key> <value>
4. 删除集合中的某个元素。
   1. **srem <key> <value1> <value2> ....**
5. 随机从该集合中吐出一个值。
   1. spop <key>
6. **随机从该集合中取出n个值，不会从集合中删除。**
   1. srandmember <key> <n>
7. 返回两个集合的交集元素。
   1. sinter <key1> <key2>
8. 返回两个集合的并集元素。
   1. sunion <key1> <key2>
9. 返回两个集合的差集元素，返回的结果跟key的顺序有关系
   1. sdiff <key1> <key2>

## Hash 类型常用命令

1. 给<key>集合中的 <field>键赋值<value>
   1. hset <key> <field> <value>
2. 从<key1>集合<field> 取出 value
   1. hget <key1> <field>
3. 批量设置hash的值
   1. hmset <key1> <field1> <value1> <field2> <value2>...
4. 批量取出hash的值
   1. hmget <key1> <field1> <field2>...
5. 查看哈希表 key 中，给定域 field 是否存在。 有返回1，没有返回0
   1. **hexists key <field>**
6. 列出该hash集合的所有field
   1. hkeys <key>
7. 列出该hash集合的所有value
   1. **hvals <key>**

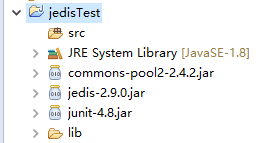
## Zset 类型常用命令

1. 将一个或多个 member 元素及其 score 值加入到有序集 key 当中。
   1. **zadd <key> <score1> <value1> <score2> <value2>...**
2. 返回有序集 key 中，下标在<start> <stop>之间的元素带WITHSCORES，可以让分数一起和值返回到结果集。
   1. **zrange <key> <start> <stop> [WITHSCORES]** 
      1. WITHSCORES 如果在命令行上加上该选项，则将score 和 value 一同取出，如果不加该选项，则只取value值！
3. 返回有序集 key 中，所有 score 值介于 min 和 max 之间(包括等于 min 或 max )的成员。有序集成员按 score 值递增(从小到大)次序排列。
   1. zrangebyscore key min max [withscores] [limit offset count]
   2. zrevrangebyscore key max min [withscores] [limit offset count] 从大到小
4. 为元素的score加上增量
   1. zincrby <key> <increment> <member>
5. 删除该集合下，指定值的元素
   1. zrem <key> <value>
6. 统计该集合，分数区间内的元素个数
   1. zcount <key> <min> <max>
7. 返回该值在集合中的排名，从**0**开始。
   1. **zrank <key> <value>**

# 使用eclipse 连接操作redis

1. 关闭防火墙
2. 修改bind 标签 为\* ，或者直接注释点 可以使用windows客户端连接

## 创建工程



1. 导入jar包

|  |
| --- |
| commons-pool2-2.4.2.jar  jedis-2.9.0.jar  junit-4.8.jar |

1. 创建测试类
   1. Jedis：这个类是用来操作redis的。该类中包含了操作redis的所有方法。

|  |
| --- |
| **public** **class** TestJedis {  @Test  **public** **void** connection(){  // 创建连接使用Jedis类  Jedis jd = **new** Jedis("192.168.67.201", 6379);  System.***out***.println(jd.ping());  jd.set("gyy", "大美女");  jd.set("ty", "清纯小美女");  System.***out***.println(jd.get("gyy"));  }  } |

1. 使用连接池进行测试

|  |
| --- |
| JedisPool jp = **null**;  Jedis jd = **null**;  @Before  **public** **void** connection() {  jp = **new** JedisPool("192.168.67.201", 6379);  jd = jp.getResource();  System.***out***.println(jd.ping());  }  // 测试String 添加单条数据  @Test  **public** **void** tStr() {  jd.set("name0", "陈乔恩");  System.***out***.println(jd.get("name0"));  }  // 测试String 添加多条数据 json {"name":"value"}  @Test  **public** **void** tStrs() {  jd.mset("name1", "赵丽颖", "sex", "女");  System.***out***.println(jd.mget("name1", "sex"));  }  // 测试hash 添加单条数据  @Test  **public** **void** tHash() {  jd.hset("h01", "name2", "妲己");  System.***out***.println(jd.hget("h01", "name2"));  }  // 测试hash 添加多条数据  @Test  **public** **void** tHashs() {  Map map = **new** HashMap<>();  map.put("name3", "大乔");  map.put("name4", "曹操");  jd.hmset("h02", map);  System.***out***.println(jd.hmget("h02", "name3", "name4"));  }  // 测试list  @Test  **public** **void** tlist() {  // jd.lpush("list01", "赵云");  // jd.lpush("list01", "张飞");  // jd.lpush("list01", "关羽");  // jd.lpush("list01", "刘备");  // 再次运行结果是:  jd.lpush("list01", "赵云");  jd.lpush("list02", "张飞");  jd.lpush("list03", "关羽");  jd.lpush("list04", "刘备");  // 从新添加一个集合  jd.lpush("list05", "凯");  jd.lpush("list06", "虞姬");  jd.lpush("list07", "孙悟空");  System.***out***.println(jd.lrange("list05", 0, 5));  }  // 测试set  @Test  **public** **void** tset() {  jd.sadd("set01", "太乙真人");  jd.sadd("set01", "蔡文姬");  jd.sadd("set01", "扁鹊");  System.***out***.println(jd.smembers("set01"));  }  // 测试 sorted Set 对数据进行排序  @Test  **public** **void** tsortSet() {  Map<String, Double> map = **new** HashMap<>();  map.put("100", **new** Double(1));  map.put("98", **new** Double(3));  map.put("99", **new** Double(2));  jd.zadd("sorted01", map);  // 0,1 取得集合中的元素 ，主要排序是根据Double数值进行排序。  System.***out***.println(jd.zrange("sorted01", 0, 1));  } |