Redis高级特性及应用场景

# redis高级特性

## redis中键的生存时间（expire）

Redis中可以使用expire命令设置一个键的生存时间，到时间后redis会自动删除它。

* 过期时间可以设置为秒或者毫秒精度。
* 过期时间分辨率总是 1 毫秒。
* 过期信息被复制和持久化到磁盘，当 Redis 停止时时间仍然在计算 (也就是说 Redis 保存了过期时间)。

|  |
| --- |
| # expire 设置生存时间（单位/秒）  expire key seconds(秒)  # ttl 查看键的剩余生存时间  ttl key  # persist 取消生存时间  persist key  expireat [key] unix时间戳1351858600  http://img.blog.csdn.net/20160503095547431 |

应用场景：网站实时性不强的数据缓存，手机验证码

## redis事务

redis中的事务是一组命令的集合。事务同命令一样都是redis的最小执行单元。一组事务中的命令要么都执行，要么都不执行。(例如：转账)

原理：先将属于一个事务的命令发送给redis进行缓存，最后再让redis依次执行这些命令。

|  |
| --- |
| #命令  multi //事务开始  .....  exec //事务结束，开始执行事务中的命令  discard //放弃事务  错误处理  语法错误：致命的错误，事务中的所有命令都不会执行  运行错误：不会影响事务中其他命令的执行  Redis 不支持回滚（roll back）：正因为redis不支持回滚功能，才使得redis在事务上可以保持简洁和快速。 |
| watch命令  作用：监控一个或者多个键，当被监控的键值被修改后阻止之后的一个事务的执行。  但是不能保证其它客户端不修改这一键值，所以我们需要在事务执行失败后重新执行事务中的命令。  注意：执行完事务的exec命令之后，watch就会取消对所有键值的监控  unwatch：取消监控 |

## redis数据排序

sort命令可以对列表类型，集合类型和有序集合类型进行排序。

sort key [desc] [limit offset count]

by 参考键(参考键可以是字符串类型或者是hash类型的某个字段，hash类型的格式为：键名->字段名)

* 如果参考键中不带\*号则不排序
* 如果某个元素的参考键不存在，则默认参考键的值为0

扩展 get参数

* get参数的规则和by参数的规则一样
* get # (返回元素本身的值)

扩展 store参数

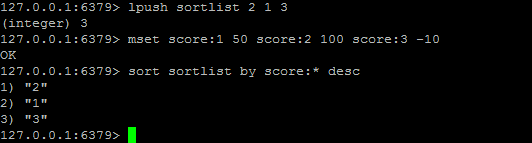
使用store 参数可以把sort的排序结果保存到指定的列表中

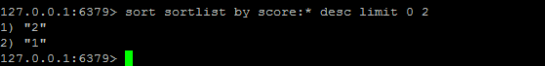
性能优化

1：尽可能减少待排序键中元素的数量

2：使用limit参数只获取需要的数据

3：如果要排序的数据数量很大，尽可能使用store参数将结果缓存。





## 发布和订阅模式

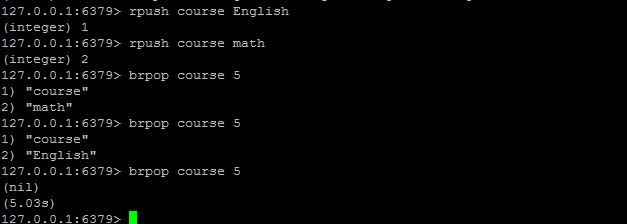
|  |
| --- |
| #发布：publish  publish channel message  #订阅：subscribe  subscribe channel [.....]  #取消订阅：unsubscribe  unsubscribe [channel]  #按照规则订阅：psubscribe  psubscribe channel ?  按照规则取消订阅：punsubscribe  注意：使用punsubscribe命令只能退订通过psubscribe 订阅的频道。  http://img.blog.csdn.net/20160503100018498  http://img.blog.csdn.net/20160503100007693 |

## redis任务队列

任务队列：使用lpush和rpop(brpop)可以实现普通的任务队列。

brpop是列表的阻塞式(blocking)弹出原语。它是 RPOP命令的阻塞版本，当给定列表内没有任何元素可供弹出的时候，连接将被 BRPOP命令阻塞，直到等待超时或发现可弹出元素为止。当给定多个 key 参数时，按参数 key 的先后顺序依次检查各个列表，弹出第一个非空列表的尾部元素。

brpop key1 key2 key3 timeout



## redis管道

redis的pipeline(管道)功能在命令行中没有，但是redis是支持管道的，在Java的客户端(jedis)中是可以使用的。

|  |
| --- |
| #不使用管道方式，插入1000条数据耗时328毫秒  // 测试不使用管道  public static void testInsert() {  long currentTimeMillis = System.currentTimeMillis();  Jedis jedis = new Jedis("192.168.33.130", 6379);  for (int i = 0; i < 1000; i++) {  jedis.set("test" + i, "test" + i);  }  long endTimeMillis = System.currentTimeMillis();  System.out.println(endTimeMillis - currentTimeMillis);  } |
| #使用管道方式，插入1000条数据耗时37毫秒  // 测试管道  public static void testPip() {  long currentTimeMillis = System.currentTimeMillis();  Jedis jedis = new Jedis("192.168.33.130", 6379);  Pipeline pipelined = jedis.pipelined();  for (int i = 0; i < 1000; i++) {  pipelined.set("bb" + i, i + "bb");  }  pipelined.sync();  long endTimeMillis = System.currentTimeMillis();  System.out.println(endTimeMillis - currentTimeMillis);  } |

# redis配置优化

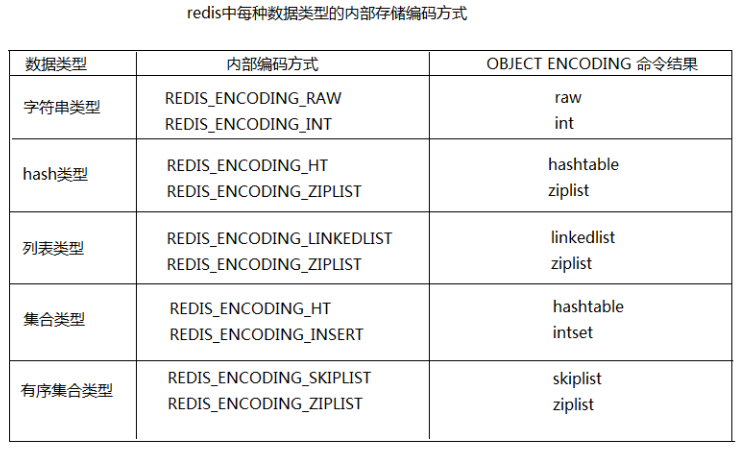
## 精简键名和键值

键名：尽量精简，但是也不能单纯为了节约空间而使用不易理解的键名。

键值：对于键值的数量固定的话可以使用0和1这样的数字来表示，（例如：male/female、right/wrong）

当业务场景不需要数据持久化时，关闭所有的持久化方式可以获得最佳的性能

内部编码优化： redis为每种数据类型都提供了两种内部编码方式，在不同的情况下redis会自动调整合适的编码方式。（如图所示)



## 修改Linux内核内存分配策略

redis在运行过程中可能会出现下面问题，错误日志：

|  |
| --- |
| WARNING overcommit\_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit\_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit\_memory=1'  redis在备份数据的时候，会fork出一个子进程，理论上child进程所占用的内存和parent是一样的，比如parent占用的内存为8G，这个时候也要同样分配8G的内存给child,如果内存无法负担，往往会造成redis服务器的down机或者IO负载过高，效率下降。所以内存分配策略应该设置为 1（表示内核允许分配所有的物理内存，而不管当前的内存状态如何）。  内存分配策略有三种可选值：0、1、2。  0， 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。  1， 不管需要多少内存，都允许申请。  2， 只允许分配物理内存和交换内存的大小。(交换内存一般是物理内存的一半)  向/etc/sysctl.conf添加：vm.overcommit\_memory = 1 //然后重启服务器  sysctl vm.overcommit\_memory=1 //立即生效 |

## 关闭Transparent Huge Pages(THP)

|  |
| --- |
| THP会造成内存锁影响redis性能，建议关闭  Transparent HugePages ：用来提高内存管理的性能  Transparent Huge Pages在32位的RHEL 6中是不支持的  #使用root用户执行下面命令，把这条命令添加到这个文件中/etc/rc.local  echo never > /sys/kernel/mm/transparent\_hugepage/enabled |

## 修改linux中TCP 监听的最大容纳数量

在高并发环境下你需要一个高backlog值来避免慢客户端连接问题。注意Linux内核默默地将这个值减小到/proc/sys/net/core/somaxconn的值，所以需要确认增大somaxconn和tcp\_max\_syn\_backlog两个值来达到想要的效果。echo 511 > /proc/sys/net/core/somaxconn

注意：这个参数并不是限制redis的最大链接数。如果想限制redis的最大连接数需要修改maxclients，默认最大连接数为10000。

## 限制redis的内存大小

通过redis的info命令查看内存使用情况。如果不设置maxmemory或者设置为0，64位系统不限制内存，32位系统最多使用3GB内存。修改配置文件中的maxmemory和maxmemory-policy

|  |
| --- |
| maxmemory：最大内存  maxmemory-policy：内存不足时,数据清除策略  如果可以确定数据总量不大，并且内存足够的情况下不需要限制redis使用的内存大小。如果数据量不可预估，并且内存也有限的话，尽量限制下redis使用的内存大小，这样可以避免redis使用swap分区或者出现OOM错误。  注意：如果不限制内存，当物理内存使用完之后，会使用swap分区，这样性能较低，如果限制了内存，当到达指定内存之后就不能添加数据了，否则会报OOM错误。可以设置maxmemory-policy，内存不足时删除数据。  used\_memory是Redis使用的内存总量，它包含了实际缓存占用的内存和Redis自身运行所占用的内存（以字节（byte）为单位，其中used\_memory\_human上的数据和used\_memory是一样的值，它以M为单位显示，仅为了方便阅读）。  如果一个Redis实例的内存使用率超过可用最大内存(used\_memory >可用最大内存)，那么操作系统开始进行内存与swap空间交换，把内存中旧的或不再使用的内容写入硬盘上（硬盘上的这块空间叫Swap分区），以便腾出新的物理内存给新页或活动页(page)使用。  在硬盘上进行读写操作要比在内存上进行读写操作，时间上慢了近5个数量级，内存是0.1us(微秒)、而硬盘是10ms(毫秒)。如果Redis进程上发生内存交换，那么Redis和依赖Redis上数据的应用会受到严重的性能影响。 通过查看used\_memory指标可知道Redis正在使用的内存情况，如果used\_memory>可用最大内存，那就说明Redis实例正在进行内存交换或者已经内存交换完毕。管理员根据这个情况，执行相对应的应急措施。  通过减少Redis的内存占用率，来避免这样的问题，或者使用下面的技巧来避免内存交换发生：  #1。 尽可能的使用Hash数据结构。因为Redis在储存小于100个字段的Hash结构上，其存储效率是非常高的。所以在不需要集合(set)操作或list的push/pop操作的时候，尽可能的使用Hash结构。比如，在一个web应用程序中，需要存储一个对象表示用户信息，使用单个key表示一个用户，其每个属性存储在Hash的字段里，这样要比给每个属性单独设置一个key-value要高效的多。 通常情况下倘若有数据使用string结构，用多个key存储时，那么应该转换成单key多字段的Hash结构。 如上述例子中介绍的Hash结构应包含，单个对象的属性或者单个用户各种各样的资料。Hash结构的操作命令是HSET(key, fields, value)和HGET(key, field)，使用它可以存储或从Hash中取出指定的字段。    #2. 设置key的过期时间。一个减少内存使用率的简单方法就是，每当存储对象时确保设置key的过期时间。倘若key在明确的时间周期内使用或者旧key不大可能被使用时，就可以用Redis过期时间命令(expire,expireat, pexpire, pexpireat)去设置过期时间，这样Redis会在key过期时自动删除key。 假如你知道每秒钟有多少个新key-value被创建，那可以调整key的存活时间，并指定阀值去限制Redis使用的最大内存。    #3. 回收key。在Redis配置文件中(一般叫Redis.conf)，通过设置“maxmemory”属性的值可以限制Redis最大使用的内存，修改后重启实例生效。也可以使用客户端命令config set maxmemory 去修改值，这个命令是立即生效的，但会在重启后会失效，需要使用config rewrite命令去刷新配置文件。 若是启用了Redis快照功能，应该设置“maxmemory”值为系统可使用内存的45%，因为快照时需要一倍的内存来复制整个数据集，也就是说如果当前已使用45%，在快照期间会变成95%(45%+45%+5%)，其中5%是预留给其他的开销。 如果没开启快照功能，maxmemory最高能设置为系统可用内存的95%。    当内存使用达到设置的最大阀值时，需要选择一种key的回收策略，可在Redis.conf配置文件中修改“maxmemory-policy”属性值。 若是Redis数据集中的key都设置了过期时间，那么“volatile-ttl”策略是比较好的选择。但如果key在达到最大内存限制时没能够迅速过期，或者根本没有设置过期时间。那么设置为“allkeys-lru”值比较合适，它允许Redis从整个数据集中挑选最近最少使用的key进行删除(LRU淘汰算法)。Redis还提供了一些其他淘汰策略，如下：  volatile-lru： 使用LRU算法从已设置过期时间的数据集合中淘汰数据。  volatile-ttl：从已设置过期时间的数据集合中挑选即将过期的数据淘汰。  volatile-random：从已设置过期时间的数据集合中随机挑选数据淘汰。  allkeys-lru：使用LRU算法从所有数据集合中淘汰数据。  allkeys-random：从数据集合中任意选择数据淘汰  no-enviction：禁止淘汰数据。  通过设置maxmemory为系统可用内存的45%或95%(取决于持久化策略)和设置“maxmemory-policy”为“volatile-ttl”或“allkeys-lru”(取决于过期设置)，可以比较准确的限制Redis最大内存使用率，在绝大多数场景下使用这2种方式可确保Redis不会进行内存交换。倘若你担心由于限制了内存使用率导致丢失数据的话，可以设置noneviction值禁止淘汰数据。 |

# redis使用场景

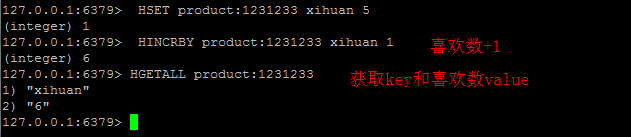
## 发布与订阅

在更新中保持用户对数据的映射是系统中的一个普遍任务。Redis的pub/sub功能使用了SUBSCRIBE、UNSUBSCRIBE和PUBLISH命令，让这个变得更加容易。

|  |
| --- |
| // 订阅频道数据  public static void testSubscribe() {  //连接Redis数据库  Jedis jedis = new Jedis("192.168.33.130", 6379);  JedisPubSub jedisPubSub = new JedisPubSub() {    // 当向监听的频道发送数据时，这个方法会被触发  @Override  public void onMessage(String channel, String message) {  System.out.println("收到消息" + message);  //当收到 "unsubscribe" 消息时，调用取消订阅方法  if ("unsubscribe".equals(message)) {  this.unsubscribe();  }  }    // 当取消订阅指定频道的时候，这个方法会被触发  @Override  public void onUnsubscribe(String channel, int subscribedChannels) {  System.out.println("取消订阅频道" + channel);  }    };  // 订阅之后，当前进程一致处于监听状态，当被取消订阅之后，当前进程会结束  jedis.subscribe(jedisPubSub, "ch1");  }      // 发布频道数据  public static void testPubSub() throws Exception {  //链接Redis数据库  Jedis jedis = new Jedis("192.168.33.130", 6379);  //发布频道 "ch1" 和消息 "hello redis"  jedis.publish("ch1", "hello redis");  //关闭连接  jedis.close();  } |

## 各种计数

商品维度计数（喜欢数，评论数，鉴定数，浏览数,etc），采用Redis 的类型: Hash.为product定义个key product:，为每种数值定义hashkey, 譬如喜欢数xihuan。



## 排行榜及相关问题

排行榜（leader board）按照得分进行排序。ZADD命令可以直接实现这个功能，而ZREVRANGE命令可以用来按照得分来获取前100名的用户，ZRANK可以用来获取用户排名，非常直接而且操作容易。

## 队列

在当前的编程中队列随处可见。除了push和pop类型的命令之外，Redis还有阻塞队列的命令，能够让一个程序在执行时被另一个程序添加到队列。你也可以做些更有趣的事情，比如一个旋转更新的RSS feed队列。

## 缓存

Redis缓存使用的方式与memcache相同。

网络应用不能无休止地进行模型的战争，看看这些Redis的原语命令，尽管简单但功能强大，把它们加以组合，所能完成的就更无法想象。当然，你可以专门编写代码来完成所有这些操作，但Redis实现起来显然更为轻松。