# Redis批量建议

首先强烈建议开始在redis中存储的时候一定要考虑批量操作的情况，存储的数据类型考虑使用set和hmap，至于set和hmap的命令以及java api的使用建议浏览官方文档。

针对于有些较大的工程中已经使用了string或者list，而list中仅仅存储数据库中对应一条的数据的情况，由于此时修改数据存储的方式工程量比较大，容易出现问题，不能保证一次修改完成及正确修改，需要经过多次调试，此种情况建议考虑使用redis的pipeline管道实现高效批量的操作。

# Pipeline介绍

Redis 管道是客户端将多个命令打包一次性发送给服务器，不用等到单独命令的执行结果返回；而 Redis 管理需要服务器在执行所有的命令后返回结果，所有命令发送和接收只发生一次，大大减少了多个命令单个在网络的耗时。

管道技术是广泛使用的技术。使如，许多 POP3 协议实现已经支持此功能，大大加快了从服务器下载电子邮件的过程。Redis 从很早就开始支持管道技术，不管运行的什么版本，可理解为使用管道技术可以不考虑版本是否支持的情况。

管道技术不仅仅可以减少由于往返时间而导致的延迟成功，还大大提高了在给定 Redis 服务中每秒可执行的总操作量。这是因为，在不使用管道技术的情况下，为每个命令提供服务，需要调用系统 read()和write()，意味着从用户态到内核态，在多命令情况下，频繁的上下文切换需要消耗巨大的系统性能。使用管道技术，通常调用一次read()可以读取多个命令，并调用一次write()返回多个执行结果。使用管道技术最多可以提高10倍的性能。

# Pipeline应用

## 单机版jedis应用pipeline

单机版的jedis调用pipeline的使用案例，这里和普通的操作进行了对比:

|  |
| --- |
| Jedis jedis = new Jedis("localhost", 6379);  Logger logger = Logger.getLogger("javasoft");  long start = System.currentTimeMillis();  for (int i = 0; i < 10000; i++) {  jedis.set(String.valueOf(i), String.valueOf(i));  }  long end = System.currentTimeMillis();  logger.info("the jedis total time is:" + (end - start));  Pipeline pipe = jedis.pipelined(); // 先创建一个pipeline的链接对象  long start\_pipe = System.currentTimeMillis();  for (int i = 0; i < 10000; i++) {  pipe.set(String.valueOf(i), String.valueOf(i));  }  pipe.sync(); // 获取所有的response  long end\_pipe = System.currentTimeMillis();  logger.info("the pipe total time is:" + (end\_pipe - start\_pipe)); |

## 集群版redisTampate应用pipeline

集群版的redis初始化的类为redisCluster，redisCluster不支持使用pipeline，网上有三种解决方案，第一种是通过redisCluster最终获取redis实例，然后进行调用pipeline；第二种方式是调用github上封装的一个redis-exe的；第三种方式也就是目前我们使用的springboot支持的spring-data-redis中的redisTampate；

|  |
| --- |
| long start = System.currentTimeMillis();  for (int i = 0; i < 10; i++) {  redisTemplate.opsForValue().set("gcc\_test\_normal"+String.valueOf(i), String.valueOf(i));  }  long end = System.currentTimeMillis();  logger.info("the jedis total time is:" + (end - start));  long start\_pipe = System.currentTimeMillis();  List<Object> list = redisTemplate.executePipelined(new SessionCallback<Object>() {  @Override  public <K, V> Object execute(RedisOperations<K, V> operations) throws DataAccessException {  for (int i = 0; i < 10; i++) {  redisTemplate.opsForValue().set("gcc\_test\_pipeline"+String.valueOf(i), String.valueOf(i));  }  return null;  }  });  long end\_pipe = System.currentTimeMillis();  logger.info("the pipe total time is:" + (end\_pipe - start\_pipe)); |