**一、引言**  
  
  
             学习redis 也有一段时间了，该接触的也差不多了。后来有一天，以前的同事问我，如何向redis中批量的增加数据，肯定是大批量的，为了这主题，我又重新找起了解决方案。目前的解决方案大都是从官网上查找和翻译的，每个实例也都调试了，正确无误。把结果告诉我同事的时候，我也更清楚这个主题如何操作了，里面的细节也更清楚了。当然也有人说可以通过脚本来做这个操作，没错，但是我对脚本语言还没有研究很透，就不来班门弄斧了。  
  
  
**二、管道的由来**  
  
             说起这个主题也是我同事帮的忙，关于批量增加增加数据到Redis服务器中，我已经写了一篇文章了，那篇文章只是介绍的操作，我们学技术，就要做到知其然知其所以然，所以就有了这篇文章。如果想查看我的上一篇文章，可以点击这里《[Redis进阶实践之十六 Redis大批量增加数据](http://www.cnblogs.com/PatrickLiu/p/8548580.html" \t "http://www.cnblogs.com/PatrickLiu/p/_blank)》  
  
  
        **1、请求/响应协议和RTT**  
  
                       Redis是使用 客户端-服务器（Client-Server） 模型的TCP服务器，称为请求/响应模式。  
  
                       这意味着通过以下步骤才能完成请求：  
  
                             1.1、客户端向服务器发送查询，并通常以阻塞的方式从套接字读取服务器响应。  
  
                             1.2、服务器处理命令并将响应发送回客户端。  
  
                       例如，这四个命令序列就是这样的：

[IMG_256](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

Client: INCR X

Server: 1

Client: INCR X

Server: 2

Client: INCR X

Server: 3

Client: INCR X

Server: 4

[IMG_257](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

                         客户端和服务器通过网络链路进行连接。这样的链接可以非常快（一个回送接口）或非常慢（通过互联网在两台主机之间建立很多跳转的连接）。无论网络延迟如何，数据包都会从客户端传输到服务器，然后从服务器传回客户端以进行回复。  
  
                          这个时间来回被称为RTT（往返时间）。当客户端需要连续执行多个请求时（例如，将许多元素添加到同一个列表或使用多个键填充数据库），很容易看到这会很影响性能。例如，如果RTT时间为250毫秒（在因特网上连接速度非常慢的情况下），即使服务器能够每秒处理100k个请求，此时我们也只能够每秒最多处理四个请求。  
  
                         如果使用的接口是本地回送接口（loopback），则RTT要短得多（例如，我的主机报告0.0,040毫秒ping 127.0.0.1），但如果您需要连续执行很多写操作，则仍然需要很多的时间。  
  
                        幸运的是，有一种方法可以改善这种做法。  
  
  
**2、Redis的管道**  
  
                       请求/响应服务器可以这样实现，即使客户端没有阅读上一条命令的回复，它也能够处理新的请求。通过这种方式，可以发送多个命令到服务器而无需等待回复，最后一步读取回复。  
  
                        这被称为管道技术，并且是被广泛使用的技术。例如，许多POP3协议的实现已经支持这个功能，显著加快了从服务器下载新电子邮件的过程。  
  
                        Redis自从早期的版本开始就支持管道的操作，因此无论您运行哪种版本，都可以使用Redis进行管道的操作。这是使用原始netcat实用程序的示例：

[root@linux ~]# (printf "PING\r\nPING\r\nPING\r\n"; sleep 1) | nc 192.168.127.130 6379

+PONG

+PONG

+PONG

                        （如果执行nc命令，提示：command not found，安装命令即可，即：yum install nc）  
  
                        这次我们没有为每次通话支付RTT的时间成本，只是把三命令作为了一个命令执行，最后只为这一次执行花费了时间。  
  
                        非常明确地说，通过管道的操作，我们第一个例子的操作顺序如下：

[IMG_258](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

Client: INCR X

Client: INCR X

Client: INCR X

Client: INCR X

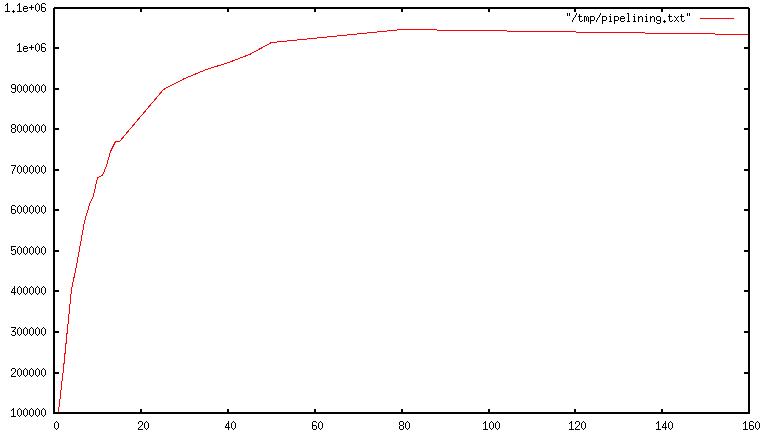
Server: 1

Server: 2

Server: 3

Server: 4

[IMG_259](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

                        重要提示：当客户端使用管道发送多条命令时，服务器将被迫使用内存排队答复。所以如果你需要使用管道发送大量的命令，最好将这些命令以合理的数目进行分组来批量发送，例如10k命令，读取回复，然后再发送另一个10k的命令，类似这样。速度几乎相同，所使用的额外内存的最大量将是将最大限度地排队此10k命令的回复所需的数量。  
  
  
    **3、这不仅仅是RTT的问题**  
  
                       管道不仅仅是为了减少往返时间所带来的延迟成本，它实际上可以提高您在给定的Redis服务器上每秒执行的总操作量。这是事实，即在不使用管道的情况下，从访问数据结构和生成答复的角度来看，每个命令的执行成本都不高的，但从执行套接字 I/O 操作的角度来看，这是非常昂贵的。当涉及调用read()和write()调用的时候，这个调用操作意味着要切换操作环境，要从用户登陆切换到内核登陆。最后来看，其实上下文切换才是导致速度大幅度的降低的罪魁祸首。  
  
                       当使用Redis的管道的时候，许多命令通常通过对一个read()函数的系统的调用来读取，并且通过对一个write()函数的系统的调用来传递多个响应。因此，每秒执行的总查询数量随着管道的操作呈线性增加，并最终达到未使用管道的基线的10倍，如下图所示：   
  
                          
  
  
  
      **4、一些真实世界的代码示例**  
  
                         在以下基准测试中，我们将使用支持管道的Redis Ruby客户端来测试由于管道而导致的速度提升：

[IMG_261](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

require 'rubygems'

require 'redis'

def bench(descr)

start = Time.now

yield

puts "#{descr} #{Time.now-start} seconds"

end

def without\_pipelining

r = Redis.new

10000.times {

r.ping

}

end

def with\_pipelining

r = Redis.new

r.pipelined {

10000.times {

r.ping

}

}

end

bench("without pipelining") {

without\_pipelining

}

bench("with pipelining") {

with\_pipelining

}

[IMG_262](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

                         运行上述简单脚本将在我的Mac OS X系统中提供以下图形，通过环回接口运行，其中管道将提供最小的改进，其他保持不变，因为RTT已经非常低：

without pipelining 1.185238 seconds

with pipelining 0.250783 seconds

                        正如您所看到的，使用管道，我们将传输速度改提升五倍。  
  
  
      **5、管道VS脚本**  
  
                        使用Redis脚本（Redis版本2.6或更高版本中可用），可以在服务器端更高效执行处理大量的管道用例的工作。 脚本的一大优点是它能够以最小的延迟读取和写入数据，使得读取，计算，写入等操作非常快速（在这种情况下，管道操作不起作用，因为客户端在调用写入命令之前需要读取命令的回复）。  
  
                       有时，应用程序可能还想在管道中发送EVAL或EVALSHA命令。这是完全可能的，并且Redis通过SCRIPT LOAD命令明确是支持的（它保证可以在没有失败风险的情况下调用EVALSHA）。  
  
  
    **6、 EVALSHA sha1 numkeys key [key ...] arg [arg ...]**  
  
                      Redis可以使用该命令的版本是2.6.0，或者更高的版本。  
  
                     时间复杂度：取决于执行的脚本。  
  
                     通过其SHA1摘要评估缓存在服务器端的脚本。使用SCRIPT LOAD命令将脚本缓存在服务器端。该命令在其他方面与EVAL相同。  
  
  
       **7、附录：为什么即使在回送接口上，一个繁忙的循环也很慢？**  
  
                    即使在本页面介绍的所有背景下，您仍然可能想知道为什么如在下所示的Redis基准测试中（在伪代码中），即使在回送接口中执行，并且服务器和客户端在同一物理机器上运行时，也很慢：

FOR-ONE-SECOND:

Redis.SET("foo","bar")

END

                   毕竟，如果Redis进程和基准测试都在同一个框中运行，那么这不仅仅是通过内存将消息从一个地方复制到另一个地方，而没有任何实际的延迟和实际网络？  
  
                   原因是系统中的进程并不总是在运行，实际上是内核调度器让进程运行的，所以会发生如下的情况，例如，当基准测试程序被允许运行，从Redis服务器读取回复（与最后执行的命令相关），并写入新的命令。该命令现在位于回送接口缓冲区中，但为了被服务器读取，内核调度器应该安排服务器进程（当前在系统调用中阻塞）运行，等等。 因此，实际上，由于内核调度程序的工作原理，回送接口仍然会有网络延迟的。  
  
                   基本上，使用一个繁忙的循环来执行基准测试是一件愚蠢的事情，可以在网络服务器中测量性能时完成相关测试。明智的做法是避免以这种方式做基准测试。  
  
  
**三、结束**  
  
               大批量插入数据的文章就写到这里了，这篇文章也介绍了 管道的一些底层的机制，对大家，对我们以后使用Redis 会有好处。等以后我对脚本语言，ruby，或者python学有所成的时候，在通过这些工具来做一些脚本执行批量插入Redis 的实力吧，也会把相应的感受和心得写出来。继续努力吧。对了，如果大家想观看英文，可以《[点击这里](https://redis.io/topics/pipelining" \t "http://www.cnblogs.com/PatrickLiu/p/_blank)》。