加餐1-Redis性能测试工具的使用

你好，我是蒋德钧。

咱们的课程已经更新过半了，在前面几个模块里，我带你从源码层面，分别了解和学习了Redis的数据结构、事件驱动框架和缓存算法的具体实现过程，相信你现在对Redis的数据类型和运行框架有了更加深入的认识。不过，阅读源码确实是一个比较烧脑的任务，需要你多花些时间钻研。所以，今天这节课，我们就通过加餐，来聊聊相对比较轻松的话题：Redis的性能测试工具。

我们在使用Redis的时候，经常会遇到需要评估Redis性能的场景。比如，当我们需要为部署Redis实例规划服务器配置规格时，或者当需要根据工作负载大小，决定Redis实例个数的时候，我们都需要了解Redis实例的运行性能。

那么这节课，我就来和你聊聊Redis的性能测试工具redis-benchmark，并带你了解下redis-benchmark的使用方法和基本实现。掌握了今天学习的内容之后，你既可以把redis-benchmark用在需要评估Redis性能的场景中，而且你还可以对redis-benchmark进行二次开发，添加新的功能特性，来满足实际业务场景中的需求。

好，下面，我们就先来看看redis-benchmark的使用。

redis-benchmark的使用

redis-benchmark这个工具是在Redis源码的redis-benchmark.c文件中实现的。这个工具实际上是模拟多个客户端给Redis server发送请求。这些请求可以包括Redis对不同数据类型的多种操作，比如对String类型的GET、SET操作，对List类型的LPUSH、LPOP操作，等等。在测试的过程中，redis-benchmark工具会记录每个请求的响应时间，最后会把请求响应时间的分布以及请求吞吐率统计并打印出来。

现在，我们可以先运行一下这个工具，一是对redis-benchmark有个直观的印象，二是可以来学习下这个工具的使用。

我在一台启动了Redis server的机器上，直接执行redis-benchmark命令，如下所示：

```
./redis-benchmark
```

然后，我们就可以得到性能测试结果。以下给出的代码片段只是展示了一部分的测试结果，是所测试的Redis server执行SET和GET两个命令的性能结果。

```
...
====== SET ======
  100000 requests completed in 1.43 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

95.80% <= 1 milliseconds
99.04% <= 2 milliseconds
```

```
99.37% <= 3 milliseconds
99.50% <= 4 milliseconds
99.61% <= 5 milliseconds
99.68% <= 6 milliseconds
99.84% <= 7 milliseconds
100.00% <= 8 milliseconds
100.00% <= 8 milliseconds
69832.40 requests per second

====== GET ======
  100000 requests completed in 1.22 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.79% <= 1 milliseconds
99.99% <= 2 milliseconds
100.00% <= 2 milliseconds
81766.15 requests per second
...
```

现在，我们来解读下这个测试结果，主要包括了两方面的信息。

一方面，测试结果会展示测试的命令操作，以及测试的配置。其中，测试配置包括一共发送的请求个数、使用的并发客户端个数、键值对的value大小等。在刚才运行的测试中，我们没有设置任何选项，所以redis-benchmark使用了默认配置。

这里，我把redis-benchmark常用的配置项列在了下面的表中，你可以看下。

配置项	含义	默认值
-h	待测试Redis server的IP地址	127.0.0.1
-p	待测试Redis server的端口号	6379
-c	并行发送请求的客户端数量	50个
-n	发送的请求总数	100000个
-d	value的大小	3字节
-r	SET/GET/INCR操作的key是否随机生成，SADD操作的值是否随机生成	用户自行设置
-P	是否使用pipeline功能，即一次命令发送的请求个数	1，即一个命令只发送一个命令
-t	测试的命令操作，用逗号隔开	用户自行设置

其中，和Redis server性能测试密切相关的选项主要是这几个：

- **-c，-n选项**

我们可以增加它们的选项值，从而增加给Redis server发送请求的客户端数量，以及发送给Redis server的

请求数量。这两个选项在对Redis server进行压力测试时，是非常重要的。因为在大压力情况下，Redis server通常要处理大并发客户端连接，以及大量的请求，通过增加这两个选项值，这样的测试结果能体现Redis server本身性能以及所使用的服务器硬件配置效果。

- **-d选项**

我们可以根据实际业务场景中的value大小，来设置这个选项值。默认情况下，redis-benchmark测试的value大小只有3字节，而通常业务场景下，value的大小从几字节、几十字节到几百字节，甚至上千字节不等。

value的字节数越多，对Redis server的内存访问、网络传输、RDB/AOF文件读写影响越大。所以，如果我们只是使用默认配置，这样测试的性能结果不一定能反映业务场景下Redis server的真实表现。

- **-r选项**

我们可以设置访问的key的随机性。如果不设置这个选项，那么，redis-benchmark访问的key是相同的，都是"key:__rand_int__"。比如，我们运行./redis-benchmark -t set -n 1000命令，测试1000次SET操作的性能。运行完之后，我们使用keys命令查看Redis数据库中的key，可以看到，其实这1000次SET操作都是访问同一个key，也就是"key:__rand_int__"。这个过程如下所示。

```
./redis-benchmark -t set -n 1000
...  //测试性能结果

/redis-cli keys \*
1) "key:__rand_int__"
```

当使用相同的key进行测试时，这会影响到我们评估Redis server随机访问性能的效果。而且，在实际业务场景中，key通常是随机的，所以，我们在实际测试过程中也需要把-r选项使用起来。

比如，我们执行./redis-benchmark -t set -n 1000 -r 10命令，测试1000次SET操作的性能。在这种情况下，这1000次SET操作实际访问的key，它们的值是在"key:000000000000"和"key:000000000009"之间，也就是说，-r选项的值N指定了key中的数字取值范围在大于等于0到小于N之间。这个过程如下所示：

```
./redis-benchmark -t set -n 1000 -r 10
...  //测试性能结果

./redis-cli keys \*
 1) "key:000000000000"
 2) "key:000000000002"
 3) "key:000000000007"
 4) "key:000000000003"
 5) "key:000000000008"
 6) "key:000000000006"
 7) "key:000000000001"
 8) "key:000000000004"
 9) "key:000000000005"
10) "key:000000000009"
```

- **-P选项**

我们可以通过该选项来设置Redis客户端以批处理的形式，让一个请求发送多个操作给Redis server，从而可以测试批处理发送操作，给Redis server吞吐率带来的性能提升效果。

比如，我们执行`./redis-benchmark -t set -n 1000000`命令，测试一百万次SET操作的性能。然后，我们再执行`./redis-benchmark -t set -n 1000000 -P 10`命令，同样测试一百万次SET操作的性能，不过此时，我们一个请求会发送10个操作。

下面的代码片段就展示了在这两种方式下，Redis server的性能结果。你可以看到，不批量发送操作的吞吐率是每秒68898个操作，而每次批量发送10个操作的吞吐率是每秒375798个操作。所以，批量发送操作能有效提升Redis server的性能。

```
./redis-benchmark -t set -n 1000000 -q
SET: 68898.99 requests per second

./redis-benchmark -t set -n 1000000 -q -P 10
SET: 375798.56 requests per second
```

好了，了解了redis-benchmark的主要配置选项，以及这其中和性能评估密切相关的选项后，我们再来看下**redis-benchmark运行后包含的另一方面信息，也就是测试性能结果信息**。

redis-benchmark运行后提供的性能结果包括两部分：一是**操作的延迟分布**。这部分信息展示了不同百分比的操作，它们的延迟最大值。二是**server的吞吐率**，也就是每秒完成的操作数。下面的代码片段就展示了，我们测试1000次SET操作后的性能结果。其中，98.65%的操作延迟小于等于1毫秒，99.17%的操作延迟小于等于2毫秒，而所有操作（也就是100%操作）的延迟都小于等于3毫秒。

```
./redis-benchmark -t set -n 10000
====== SET ======
  10000 requests completed in 0.13 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

98.65% <= 1 milliseconds
99.17% <= 2 milliseconds
100.00% <= 3 milliseconds
75187.97 requests per second
```

这里，你需要注意的是，在redis-benchmark的测试结果中，**延迟分布对于Redis来说，是非常重要的信息**。因为Redis通常需要服务大量的并发客户端，而以百分比统计的延迟分布，可以告诉我们这其中有多少比例的操作，它们的延迟较高。

为了帮助你更好地理解百分比延迟分布的作用，我给你举个例子。假设redis-benchmark的测试结果显示99%的操作延迟小于等于1毫秒，而所有操作，也就是100%的操作延迟小于等于5毫秒，那么就表明有1%的操作延迟是在1毫秒到5毫秒之间的。如果某个操作的延迟正好是5毫秒，那么和其他99%的操作相比，它的延迟就增加了5倍，这样一来，发送这个操作的客户端就会受到明显的性能影响。

我们再假设Redis server处理的请求数一共是100万个请求，那么1%的操作影响的就是1万个请求。而且Redis server处理的请求越多，这个影响的范围就越大。所以，这个以百分比统计的延迟分布可以帮助我们更加全面地评估Redis server的性能表现。

好了，到这里，我们就可以通过运行redis-benchmark这个工具，来了解我们所测试的Redis server处理不同请求操作的延迟分布和吞吐率了。

那么接下来，我们再来了解下redis-benchmark是怎么实现的。

redis-benchmark的实现

redis-benchmark本身可以单独运行，这是因为它本身就自带main函数。我们了解它的main函数，就可以了解redis-benchmark的基本实现。

它的main函数的主要执行流程可以分成三步。

第一步，main函数设置各种配置参数的默认值，比如待测试的Redis server的IP、端口号、客户端数量、value大小，等等。紧接着，main函数会调用parseOptions函数，解析通过redis-benchmark命令传入的各项参数，这就包括了我刚才给你介绍的redis-benchmark的基本配置项。

另外在这一步中，main函数还会调用aeCreateEventLoop函数创建一个事件循环，如下所示。redis-benchmark在实际运行时，会通过这个事件循环流程，来处理客户端的读写事件。

```
config.el = aeCreateEventLoop(1024*10);
```

第二步，main函数会检查redis-benchmark命令参数中是否包含了其他命令，如果有的话，那么redis-benchmark工具会调用benchmark函数（在redis-benchmark.c文件中），来实际测试这些命令操作。

benchmark函数会调用createClient函数（在redis-benchmark.c文件中）创建一个客户端。然后，它再调用createMissingClients函数（在redis-benchmark.c文件中），检查是否有多个并发客户端要创建。如果是的话，createMissingClients函数也会调用createClient函数，来创建剩余的客户端。

这里，**你需要注意的是**，createClient函数在创建完客户端后，只要redis-benchmark没有设置idle模式，也就是只创建客户端而不发送请求，那么，它就会调用aeCreateFileEvent函数在客户端上注册写事件。这里的写事件回调函数是writeHandler（在redis-benchmark.c文件中），负责向Redis server发送命令操作，如下所示：

```
if (config.idlemode == 0)
    aeCreateFileEvent(config.el,c->context->fd,AE_WRITABLE,writeHandler,c);
```

而writeHandler函数完成命令操作发送后，会调用aeDeleteFileEvent函数将当前客户端上监听的写事件删除，同时，创建当前客户端上监听的读事件，读事件的回调函数是readHandler（在redis-benchmark.c文件中），负责读取Redis server的返回结果。

```
if (sdslen(c->obuf) == c->written) {
   aeDeleteFileEvent(config.el,c->context->fd,AE_WRITABLE);
   aeCreateFileEvent(config.el,c->context->fd,AE_READABLE,readHandler,c);
 }
```

那么，再回到benchmark函数中，在创建完客户端后，紧接着，benchmark函数会调用aeMain函数进入刚才第一步中创建的事件循环流程，开始处理读写事件。如果事件循环流程结束了，benchmark函数调用showLatencyReport函数（在redis-benchmark.c文件中）打印测试结果，并调用freeAllClients函数（在redis-benchmark.c文件中）释放所有客户端。

好了，到这里，你就了解了，benchmark函数是如何使用事件驱动框架来完成操作测试的。

实际上，如果redis-benchmark命令运行时自带了测试操作，此时，在main函数的第二步中，在完成这些操作测试后，redis-benchmark工具就运行结束了，而不会再测试它的-t选项设置的命令操作了。

而如果redis-benchmark命令运行时没有自带测试操作，那么main函数就会进入第三步。

在**第三步**中，main函数会调用test_is_selected函数（在redis-benchmark.c文件中），判断-t选项中设置了哪些命令操作，然后main函数调用benchmark函数来完成这些操作的测试。

这样一来，redis-benchmark工具的基本执行流程就结束了。

小结

今天这节课我给你介绍了redis-benchmark工具的使用。redis-benchmark是常用的Redis性能测试工具，它可以通过设置并发客户端、总操作数、value大小、key的随机性、批量发送等配置项，来给Redis server施加不同的压力。

redis-benchmark工具本身提供了一些常见命令的测试，比如SET、GET、LPUSH，等等。这些命令的测试是redis-benchmark在它的实现文件中固定写好的。你可以在redis-benchmark.c文件中的main函数里面，找到这些命令。而如果我们想要测试不在固定测试命令集中的其他命令，我们可以在redis-benchmark命令的最后，设置其他的Redis命令，从而可以测试其他命令的性能结果。

最后，我也给你介绍了redis-benchmark的基本实现。它其实是启动多个客户端向Redis server发送命令操作。这个过程中，redis-benchmark使用了事件驱动框架。每当启动一个测试客户端，这个客户端会在事件驱动框架中创建写事件和读事件。写事件对应了测试客户端向Redis server发送操作命令，而读事件对应了测试客户端从Redis server读取响应结果。

从这里，你可以看到，Redis实现的事件驱动框架不仅用在server的运行过程中，而且还用在了性能测试工

具实现的客户端中。

每课一问

你在实际工作中，还用过什么其他的Redis性能测试工具吗？欢迎在留言区分享，我们一起交流探讨。