

原

JAVA NIO(五): 如何在5秒内写入10G的文本数据

2017年09月28日 21:29:27

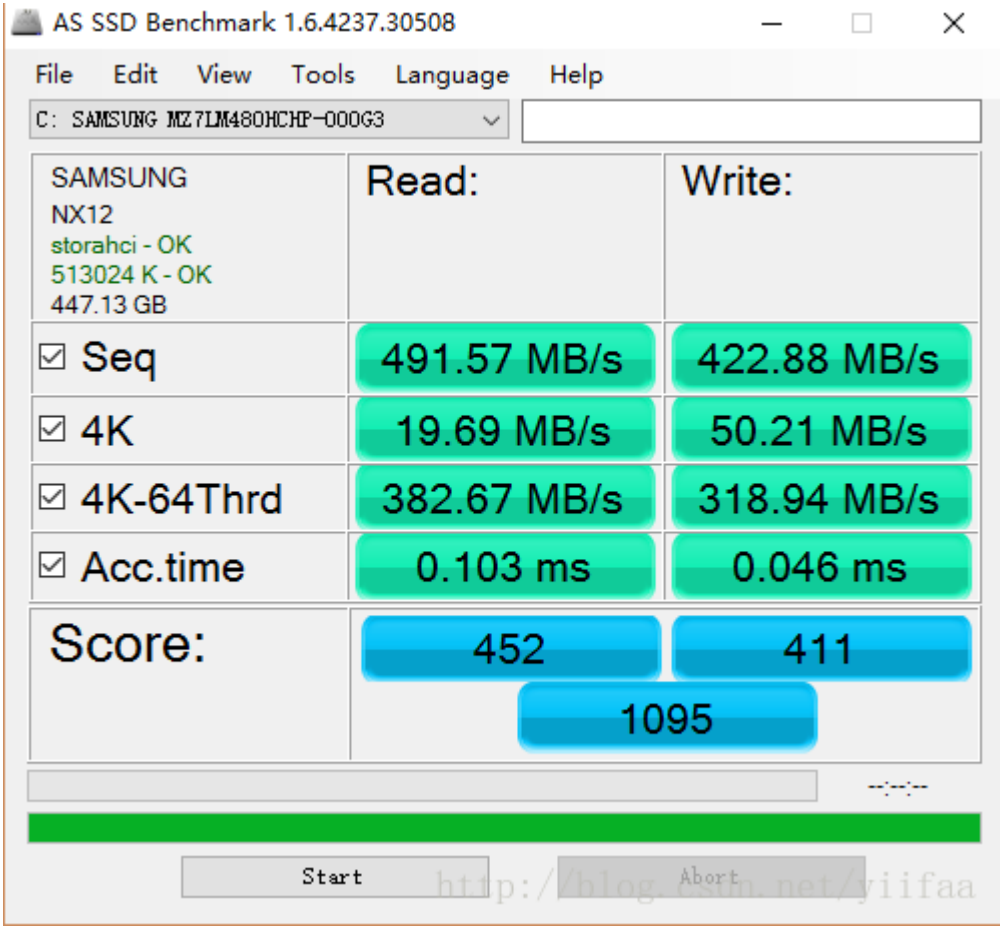
蚁方阵

阅读数: 1814

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/yiifaa/article/details/78128363>

首先说本机的性能，采用AS SSD Benchmark进行评测，写入能力大约在422M每秒，计划连续写入文本数据，直达到要求为止(5G数据与10G数据)，测试环境如下：

环境	版本
JDK	1.8.0_131
操作系统	Windows 10 专业版 x64
CPU	Inter i7-3740QM
内存	16G
硬盘	三星512G SSD



1. FileOutputStream与BufferedWriter

原以为FileOutputStream的性能会很低，BufferedWriter会有一定的性能提升，但结果却让我大吃一惊，测试数据如下：

测试编次	采用方式	文件大小	花费时间(秒)
1	BufferedWriter	4.5G	10.678399057
2	BufferedWriter	4.5G	10.808078377
3	FileOutputStream	4.5G	9.755711962
4	FileOutputStream	4.5G	9.457581885

BufferedWriter竟然还稍稍慢于FileOutputStream，并且FileOutputStream的性能如此惊人，已经完全达到了硬盘的性能巅峰，这说明JAVA的IO优化还是令人非常满意的，相关代码如下：

```
1 // FileOutputStream的写入方式类似，在此略
2 static void writeBuffer(File file) throws IOException {
3     FileOutputStream fos = new FileOutputStream(file);
4     BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(fos));
5     int i = 1000000;
6     while(i > 0) {
7         // word2048为字符串常量，刚好4800个字节
8         writer.write(word2048);
9         i --;
10    }
11    writer.close();
12    fos.close();
13 }
```

2. ByteBuffer与直接缓冲区

几乎所有的人都推荐，nio性能极佳，那真实的性能到底怎样？

文件大小	采用方式	花费时间(秒)
4.5G	采用直接内存	8.598730553
4.5G	不采用直接内存	8.581370111

从上面的数据可以看出，采用ByteBuffer后，性能约有10%的提升，但令人惊讶的，采不采用直接缓冲区竟然没有差异，这与理论推测又有显著差异(具体请参见《JAVA NIO》第45页)，在这里还有一个可优化的地方，如何选择ByteBuffer的大小是决定写入速度的关键，相关代码如下：

```
1 FileOutputStream fos = new FileOutputStream(file);
2 FileChannel fc = fos.getChannel();
3 // 此数字可优化
4 int times = 100;
5 // word2048为字符串常量，刚好4800个字节
```

```
6 byte[] datas = word2048.getBytes();
7 ByteBuffer bbuf = ByteBuffer.allocate(4800 * times);
8 int i = 10000;
9 while(i > 0) {
10     for(int j = 0; j < times; j++) {
11         bbuf.put(datas);
12     }
13     bbuf.flip();
14     fc.write(bbuf);
15     bbuf.clear();
16     i --;
17 }
```

1
1

从这里来看，跟BIO相比，NIO性能提升并不明显。

3. FileChannel与文件空洞

在nio中，FileChannel可以决定文件的写入位置，这也是能产生文件空洞的主要方法，那么这样，产生大量的文件空洞，是否能加快文件的创建速度呢？

文件大小	采用方式	花费时间
5.1G	改变Channel Position 步进2	13.214593475S
11G	改变Channel Position 步进2	25.849560829S

可以看出，写入速度主要还是受限与磁盘IO，即使少写数据，依旧不能提升速度，反而还有较大幅度的下降，相关代码如下：

```
1 FileOutputStream fos = new FileOutputStream(file);
2 FileChannel fc = fos.getChannel();
3 ByteBuffer bbuf = ByteBuffer.allocateDirect(1024);
4 long value = 10 * 1024 * 1024;
5 // 为什么不一步到位？直接将position设置10G
6 for(int i = 1; i < 1025; i = i * 2) {
7     bbuf.put((byte)1);
8     bbuf.flip();
9     fc.position(i * value);
10    fc.write(bbuf);
11    bbuf.clear();
12 }
13 fc.close();
14 fos.close();
```

在这里需要强调的一点是，文件大小的速度不能增长太快，否则必然出现“IllegalArgumentException”错误，这也是上述代码中需要划分多次循环的原因。

4. 惊人的MappedByteBuffer

早就听说直接内存映射提升IO性能惊人，那到底有多惊人呢？请看测试数据：

文件大小	采用方式	花费时间(秒)
4.5G	内存映射	2.537650656
9G	内存映射	5.303423243

跟前面的最快的NIO方法相比，性能竟然提升了244%，写入速度竟然达到了1.8G每秒，这是怎么做到的？相关代码如下：

```
1 // 必须采用RandomAccessFile，并且是rw模式
2 RandomAccessFile acf = new RandomAccessFile(file, "rw");
3 FileChannel fc = acf.getChannel();
4 byte[] bs = word2048.getBytes();
5 int len = bs.length * 1000;
6 long offset = 0;
7 int i = 2000000;
8 while(i > 0) {
9     MappedByteBuffer mbuf = fc.map(FileChannel.MapMode.READ_WRITE, offset, len );
10    for(int j = 0; j < 1000; j ++ ) {
11        mbuf.put(bs);
12    }
13    offset = offset + len;
14    i = i - 1000;
15 }
16 fc.close();
```

当然，性能提升的代价也是很明显的，内存消耗至少增加了2G（直接内存，不是JAVA堆内存），而前面的方法内存消耗都很少，大多只在30M左右。

5. 其他的现象

- 1. 同样的文件名，删除了再创建，速度又有10~20%的提升；
- 2. 字符串转换为字节数组的速度极快，比直接写入字符串的速度更快，这也是BufferedWriter比FileOutputStream慢的原因；

结论

直接内存映射、直接缓冲区都能提升IO写入的性能，背后的核心技术还是分页技术(请参加操作系统原理)，如何组织分页的范围与写入的频次，是提升性能的关键，另外，写入对内存与CPU性能消耗都不高。



想对作者说点什么

 **车龙梁Adam:** 面试可能会问到此类问题 (08-31 22:34 #1楼)

<div>说说java NIO的一些个人总结</div> <div>首先了解下所谓的java nio是个什么东西！ IO是靠字符或字节进行传输，比较慢！ 而NIO是靠块， 也就相当于一个Buffer， 一块一块 的...</div>	<div> 7473</div>	1
<div>月薪30K+！ 区块链以太坊开发必须会</div> <div>区块链DApp开发学习路线图，月薪4万很轻松</div>		1
<div>Java NIO 读取文件、写入文件、读取写入混合</div> <div>前言 Java NIO(new/inputstream outputstream)使用通道、缓冲来操作流，所以要深刻理解这些概念，尤其是，缓冲中的数据结构（当...</div>	<div> 4061</div>	
<div>javanio中FileChannel写入文件write，追加文件，以及多文件合并</div> <div>FileChannel 追加写入文件实现方法如下： File file = new File(filename) ; if(!file.exists()){ ...</div>	<div> 6018</div>	
<div>java NIO读写文件</div> <div>/** * 从文件中读取数据 */ public static void readDataFormFile(String path) { try (FileInputStream fis...</div>	<div> 586</div>	
<div>java写入文件的三种方式</div> <div>java写入文件的三种方式</div>	<div> 1291</div>	
<div>Java8读文件仅需一行代码</div> <div>Java8读文件仅需一行代码JDK7中引入了新的文件操作类java.nio.file.File，它包含了很多有用的方法来操作文件，比如检查文件是否为...</div>	<div> 2.1万</div>	
<div>Java字符串写入文件三种方式</div> <div>1、使用FileWriterString str="hello world!"; FileWriter writer; try { writer...</div>	<div> 1.5万</div>	
<div>Java读写大文本文件（2GB以上）</div> <div>Java读写大文本文件（2GB以上） 如下的程序，将一个行数为fileLines的文本文件平均分为splitNum个小文本文件，其中换行符'r'是linu...</div>	<div> 6992</div>	
<div>生成任意大小文件</div> <div>1.生成任意大小文件： dd if=/dev/zero of=data.file bs=1M count=1 创建一个1M大小的文件data.file bs指定大小， if即input file， ...</div>	<div> 512</div>	
<div>相关热词</div> <div>javall与javajava的~javajava和--</div>		

个人资料



蚁方阵

关注

原创356

粉丝140

喜欢92

评论125

等级： 博客 7

访问： 92万+

积分： 1万+

排名： 2073

勋章： 恒

最新文章

Spark-SQL导出查询结果的两种方式

HttpClient提交表单出现中文乱码的解决办法

一种快速切割字符串的方法

Spark： flatmap函数提示“ambiguous implicit values”的解决办法

Scala守卫语句的两种用法

个人分类

服务器管理4篇

代码管理8篇

信息安全10篇

项目构建18篇

STI团队建设2篇

展开

归档

2018年9月7篇

2018年8月12篇

2018年7月3篇

2017年12月13篇

2017年11月11篇

展开

热门文章

Webpack引入jquery及其插件的几种方法
阅读量： 54215

输入框事件监听(一)：keydown、keyup、input
阅读量： 33725

快速禁止Chrome浏览器缓存
阅读量： 25881

Spring Boot(三)：RestTemplate提交表单数据的三种方法
阅读量： 25447

Webpack错误解决(一)：ERROR in Entry module not found
阅读量： 23979

最新评论

输入框事件监听(五)：如何感知JS...
u013605060：[reply]u013605060[/reply] 如果看到请务必回复我，谢谢

输入框事件监听(五)：如何感知JS...
u013605060：如果是jq根据id的那种赋值，这块管用吗？

设置Spring Boot上下文(...
zhang1072407657：我理解是内置tomcat启动的话，每个对tomcat的访问，都是从根目录+上下文+xxx这种形势么...

Spring Boot(六)：如何...
zhang1072407657：[reply]SWhard[/reply] 可不可以指导下，怎么访问图片，内置tomcat 启动的...

Spring Boot(六)：如何...
zhang1072407657：我把图片上传到tomcat外，然后继承WebMvcConfigurerAdapter重写addRe...

联系我们




请扫描二维码联系客服

 webmaster@csdn.net

 400-660-0108

 QQ客服  客服论坛

[关于](#) [招聘](#) [广告服务](#) [网站地图](#)
©2018 CSDN版权所有 京ICP证09002463号
 百度提供搜索支持



经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

CSDN APP