

射---上G大文件处理

Evankaka

阅读数：9882

标签：

java

内存映射

原创文章，转载请注明出处<http://blog.csdn.net/evankaka> <https://blog.csdn.net/Evankaka/article/details/48464013>

转载请注明出处<http://blog.csdn.net/evankaka>

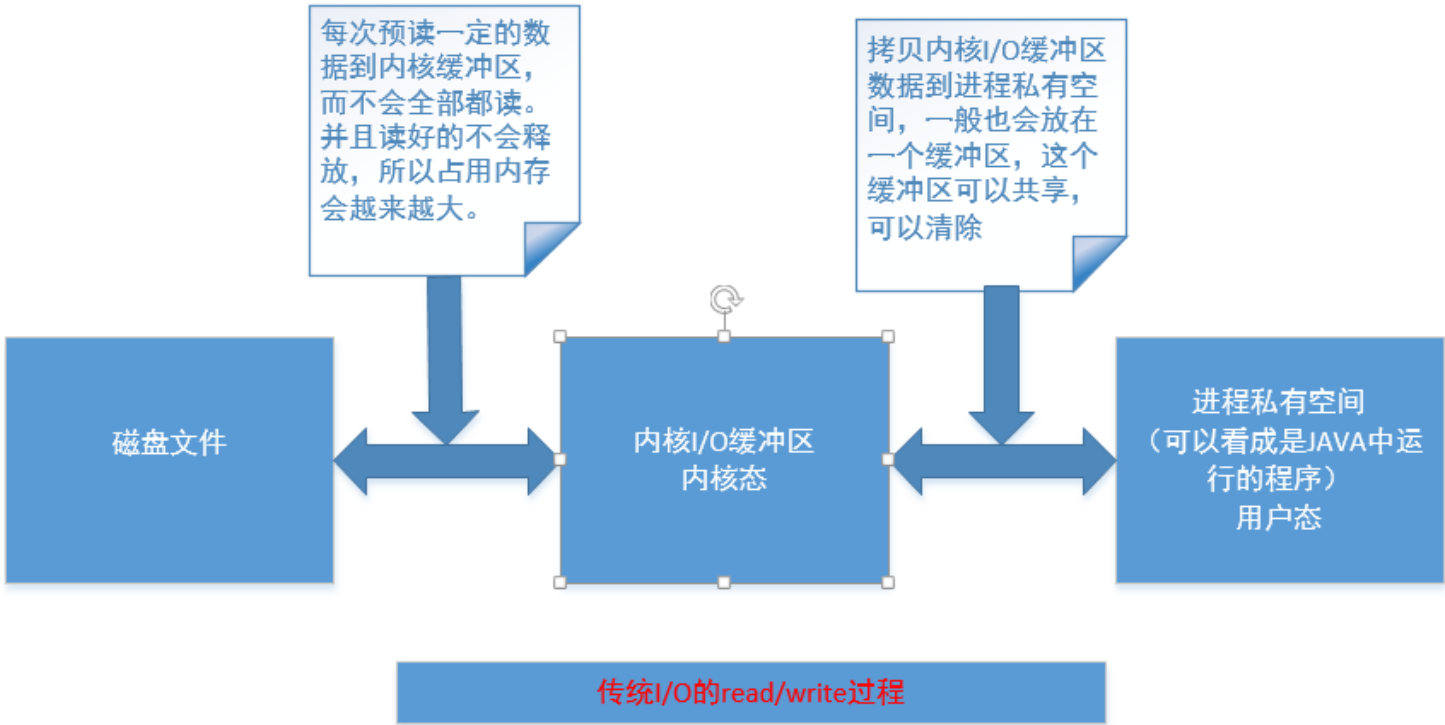
内存映射的原理及过程，与传统IO进行了对比，最后，用实例说明了结果。

射IO和内存映射文件是什么？

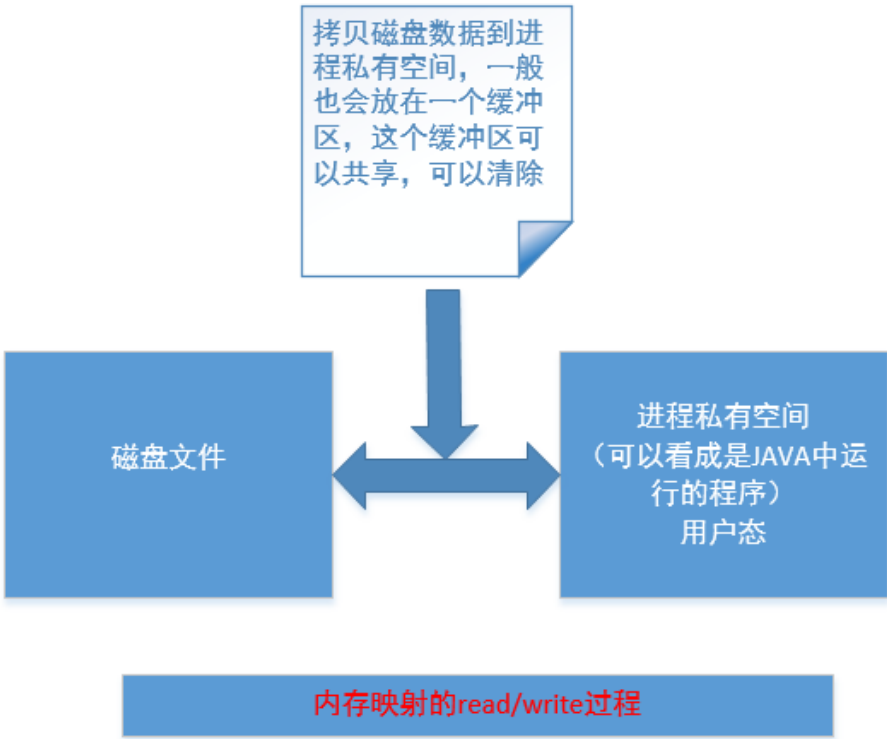
许Java程序直接从内存中读取文件内容，通过将整个或部分文件映射到内存，由操作系统来处理加载请求和写入文件，应用只需要和内存打交道，这使得IO操作非常快速。Java语言支持内存映射文件，通过java.nio包和MappedByteBuffer 可以从内存直接读写文件。

得到一块内存的映射。Win32提供了允许应用程序把文件映射到一个进程的函数 (CreateFileMapping)。内存映射文件与虚拟内存有些类似，通过内存映射文件可以保留一个地址空间的区域，同时将文件映射的物理存储器来自一个已经存在于磁盘上的文件，而且在对该文件进行操作之前必须首先对文件进行映射。使用内存映射文件处理存储于磁盘上的文件时，将不必再对文件执行I/O操作，使文件时能起到相当重要的作用。

都是调用操作系统提供的底层标准IO系统调用函数 read()、write()，此时调用此函数的进程（在JAVA中即java进程）由当前的用户态切换到内核态，然后OS的内核代码负责将相应的文件数据读取到内核IO缓冲区拷贝到进程的私有地址空间中去，这样便完成了一次IO操作。这么做是为了减少磁盘的IO操作，为了提高性能而考虑的，因为我们的程序访问一般都带有局部性，也就是所谓的局部性，即我们访问了文件的某一段数据，那么接下去很可能还会访问接下去的一段数据，由于磁盘IO操作的速度比直接访问内存慢了好几个数量级，所以OS根据局部性原理会在一次 read()系统调用中将数据读入内核IO缓冲区中，当继续访问的文件数据在缓冲区中时便直接拷贝数据到进程私有空间，避免了再次的低效率磁盘IO操作。其过程如下



IO操作最大的不同之处就在于它虽然最终也是要从磁盘读取数据，但是它并不需要将数据读取到OS内核缓冲区，而是直接将进程的用户私有地址空间中的一部分区域与文件对象建立起映射关系，这样，速度当然快了。



于性能，这对于建立高频电子交易系统尤其重要。内存映射文件通常比标准通过正常IO访问文件要快。另一个巨大的优势是内存映射IO允许加载不能直接访问的映射文件。经验表明，内存映射文件在两个进程中共享数据。尽管它也有不足——增加了页面错误的数目。由于操作系统只将一部分文件加载到内存，如果一个请求页面没有在内存中，它将导致页面错误。同样它可以被

Windows平台，UNIX，Solaris和其他类UNIX操作系统都支持内存映射IO和64位架构，你几乎可以将所有文件映射到内存并通过JAVA编程语言直接访问。

映射要点：

- 1. 只读IO。
 - 2. 应用，例如高频电子交易平台。
 - 3. 文件加载到内存。
 - 4. 映射文件，如果请求页面不在内存中的话。
 - 5. 寻址的大小。在32位机器中，你不能访问超过4GB或2 ^ 32（以上的文件）。
 - 6. 映射文件的多。
 - 7. 映射文件之外，并驻留共享内存，允许两个不同进程共享文件。
 - 8. 映射文件，所以即使在将内容写入内存后java程序崩溃了，它仍然会将它写入文件直到操作系统恢复。
 - 9. 映射文件缓冲而不是非直接缓冲。
 - 10. force()方法，这个方法意味着强制操作系统将内存中的内容写入磁盘，所以如果你每次写入内存映射文件都调用force()方法，你将不会体会到使用映射字节缓冲的好处，相反，它(的性能)将类似于非映射文件。
- 将会有很小的机率发生内存映射文件没有写入到磁盘，这意味着你可能会丢失关键数据。

缓冲区大小

不指定缓冲区大小

gwen
9月5日

```
public static void readFile1(String path) {
    long start = System.currentTimeMillis();//开始时间
    try {
        File file = new File(path);
        if (!file.exists()) {
            System.out.println("文件不存在");
            return;
        }
        FileReader fileReader = new FileReader(file);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            System.out.println(line);
        }
        bufferedReader.close();
        fileReader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    long end = System.currentTimeMillis();//结束时间
    System.out.println("传统IO读取数据，不指定缓冲区大小，总共耗时: "+(end - start)+"ms");
}
```

缓冲区大小

0读取数据, 指定缓冲区大小
gwen
9月5日

tFoundException

```
void readFile2(String path) throws FileNotFoundException {
    start = System.currentTimeMillis();//开始时间
    ufSize = 1024 * 1024 * 5;//5M缓冲区
    fin = new File(path); // 文件大小200M
    hannel fcin = new RandomAccessFile(fin, "r").getChannel();
    uffer rBuffer = ByteBuffer.allocate(bufSize);
    g enterStr = "\n";
    len = 0L;

    byte[] bs = new byte[bufSize];
    String tempString = null;
    while (fcin.read(rBuffer) != -1) { // 每次读5M到缓冲区
        int rSize = rBuffer.position();
        rBuffer.rewind();
        rBuffer.get(bs);//将缓冲区数据读到数组中
        rBuffer.clear();//清除缓冲
        tempString = new String(bs, 0, rSize);
        int fromIndex = 0;//缓冲区起始
        int endIndex = 0;//缓冲区结束
        //按行读缓冲区数据
        while ((endIndex = tempString.indexOf(enterStr, fromIndex)) != -1) {
            String line = tempString.substring(fromIndex, endIndex);//转换一行
            System.out.print(line);
            fromIndex = endIndex + 1;
        }
    }
    = System.currentTimeMillis();//结束时间
    t.println("传统IO读取数据,指定缓冲区大小, 总共耗时: "+(end - start)+"ms");

    ch (IOException e) {
        e.printStackTrace();
    }
}
```

射读大文件
nbingwen
15年9月15日
h

```
void readFile3(String path) {
    start = System.currentTimeMillis();//开始时间
    th = 0;
    FER_SIZE = 0x300000;// 3M的缓冲
    = new File(path);
    h = file.length();

    dByteBuffer inputBuffer = new RandomAccessFile(file, "r").getChannel().map(FileChannel.MapMode.READ_ONLY, 0, fileLength);// 读取大文件

    ] dst = new byte[BUFFER_SIZE]; // 每次读出3M的内容

    int offset = 0; offset < fileLength; offset += BUFFER_SIZE) {
    f (fileLength - offset >= BUFFER_SIZE) {
        for (int i = 0; i < BUFFER_SIZE; i++)
            dst[i] = inputBuffer.get(offset + i);
    else {
        for (int i = 0; i < fileLength - offset; i++)
            dst[i] = inputBuffer.get(offset + i);

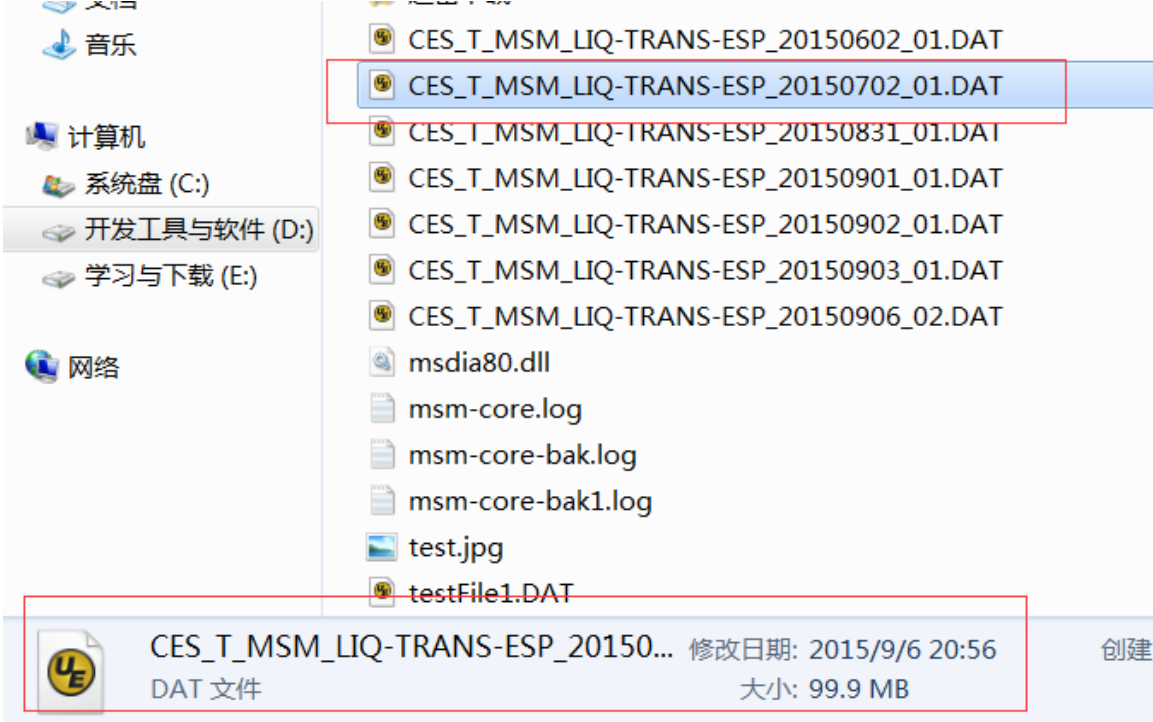
    / 将得到的3M内容给Scanner, 这里的XXX是指Scanner解析的分隔符
    canner scan = new Scanner(new ByteArrayInputStream(dst)).useDelimiter(" ");
    hile (scan.hasNext()) {
        // 这里为对读取文本解析的方法
        System.out.print(scan.next() + " ");

    can.close();
}
```

6
2

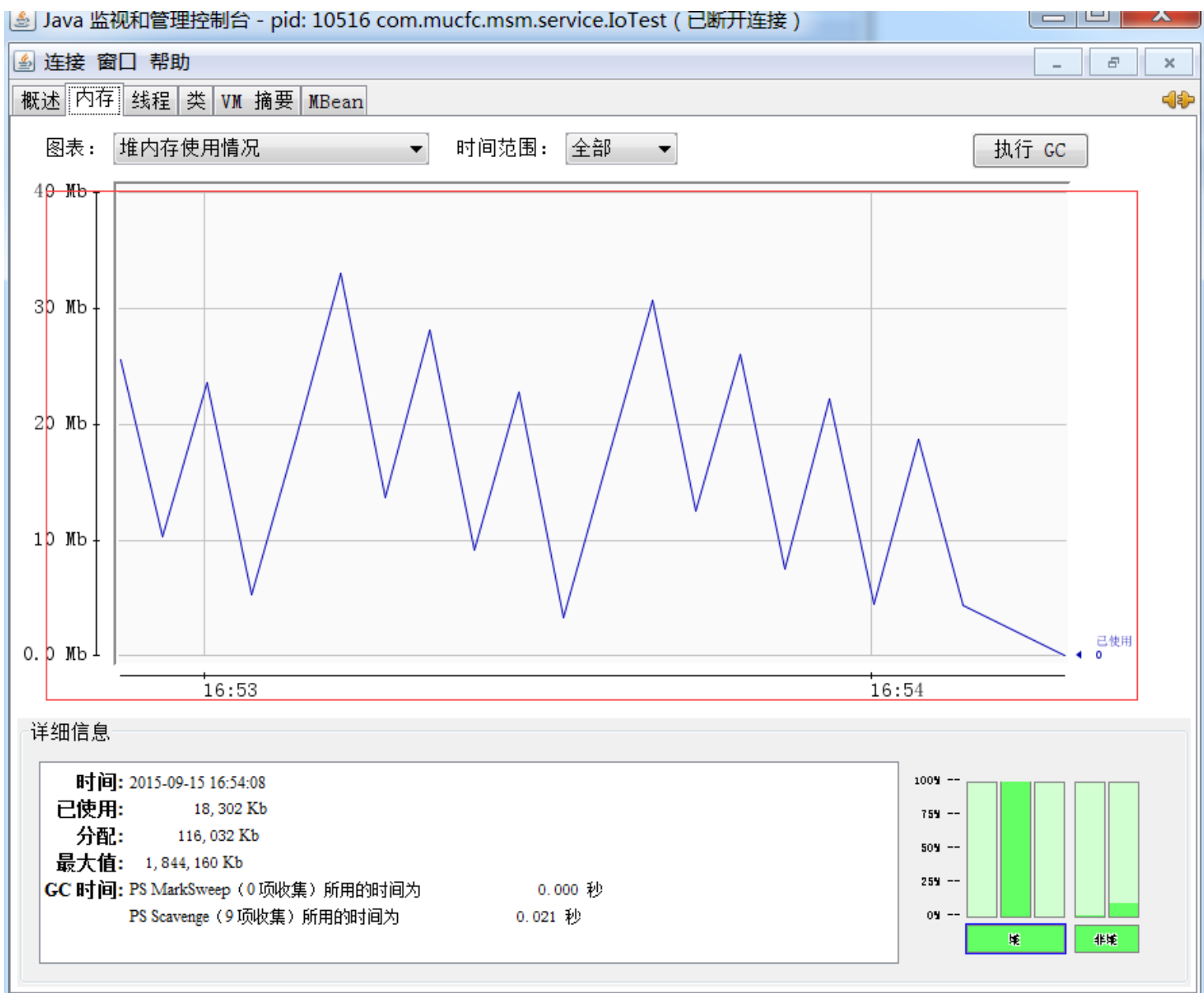
```
m.out.println();35 | long end = System.currentTimeMillis();//结束时间
m.out.println("NIO 内存映射读大文件，总共耗时: "+(end - start)+"ms");
Exception e) {
ntStackTrace();
```

6
2



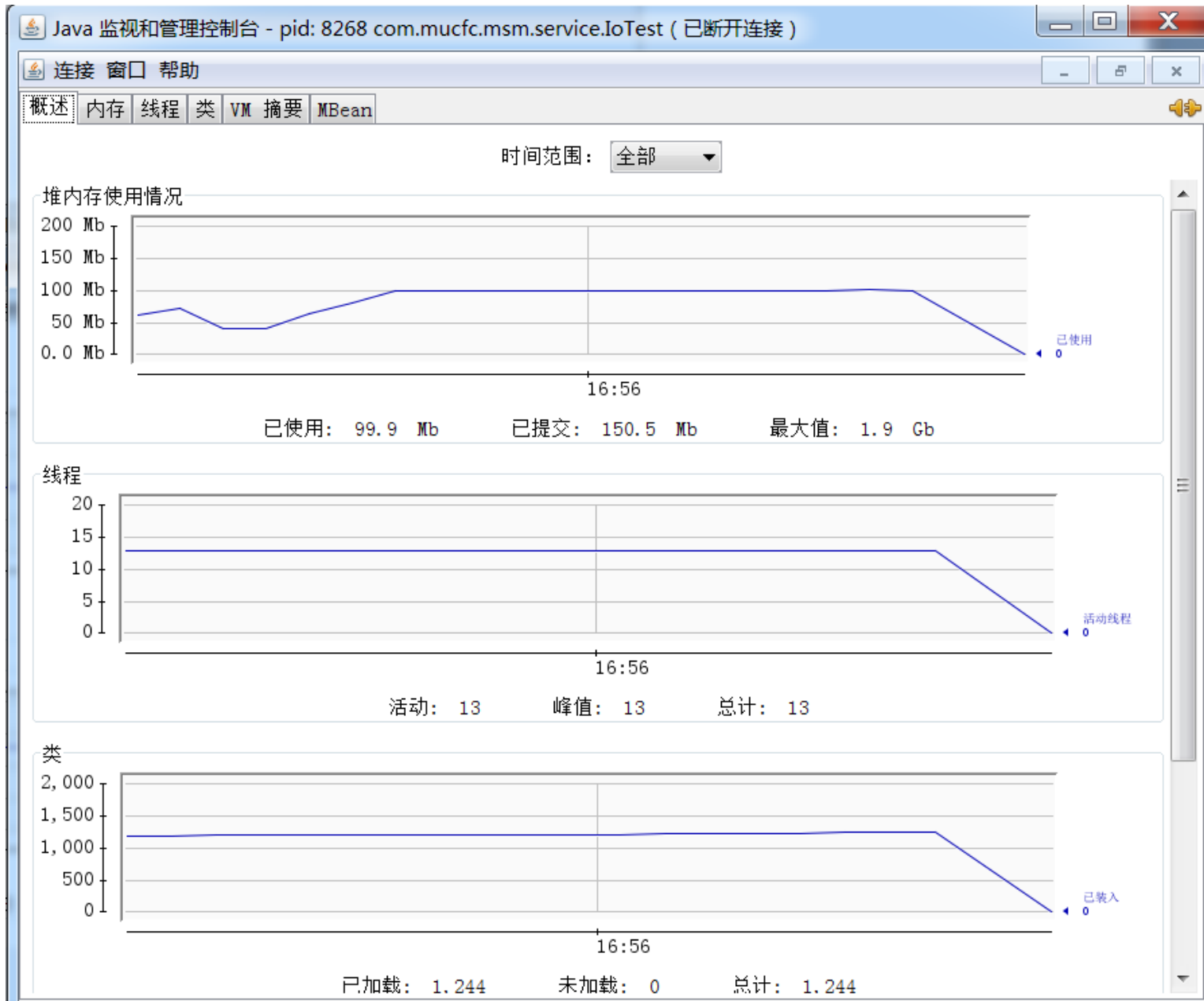
```
void main(String args[]) {
g path = "D:" + File.separator + "CES_T_MSM_LIQ-TRANS-ESP_20150702_01.DAT";
ile1(path);
dFile2(path);
dFile3(path);
```

缓冲区大小，总共耗时：80264ms



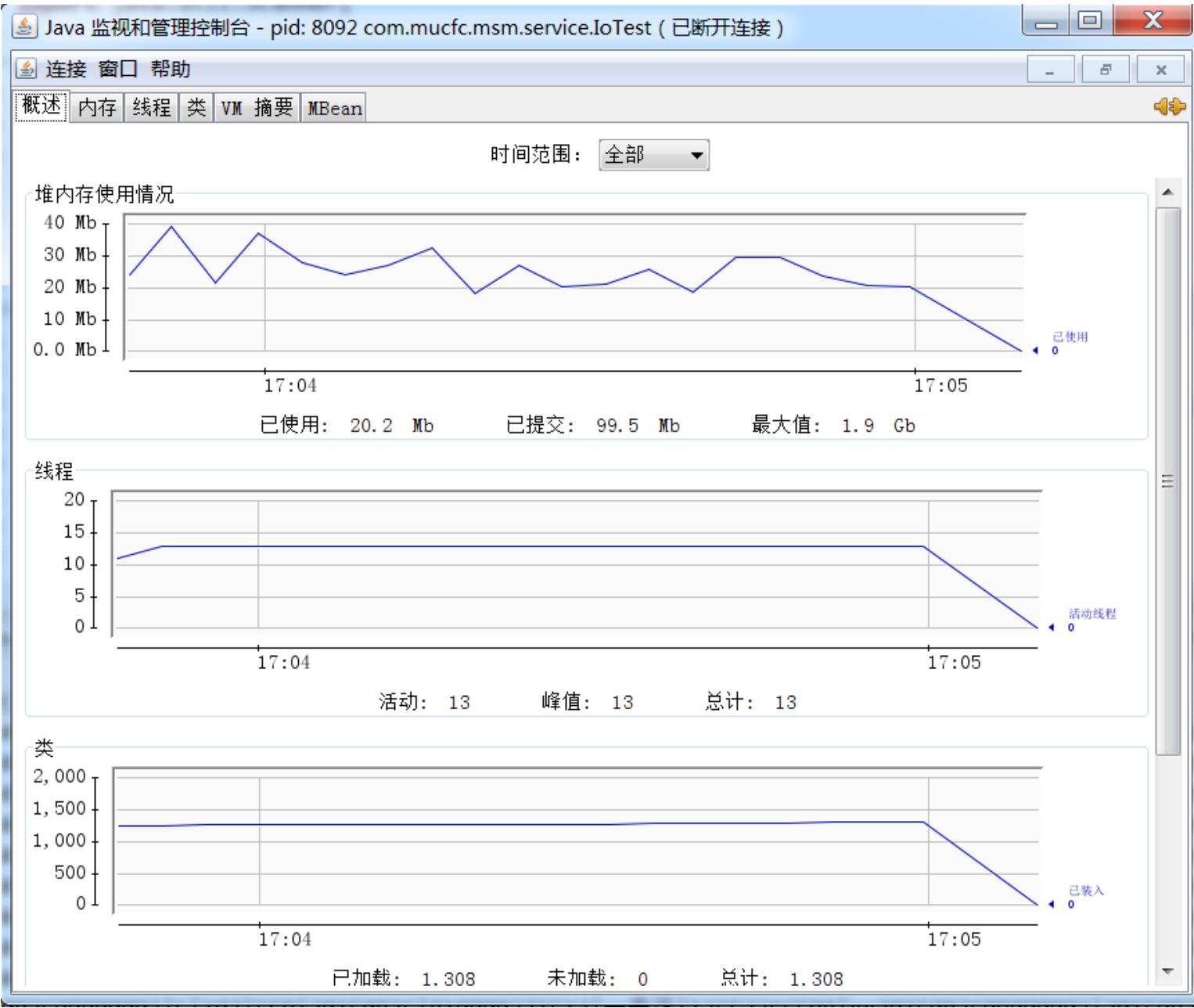
大小, 总共耗时: 80612ms

6
2



共耗时: 90955ms

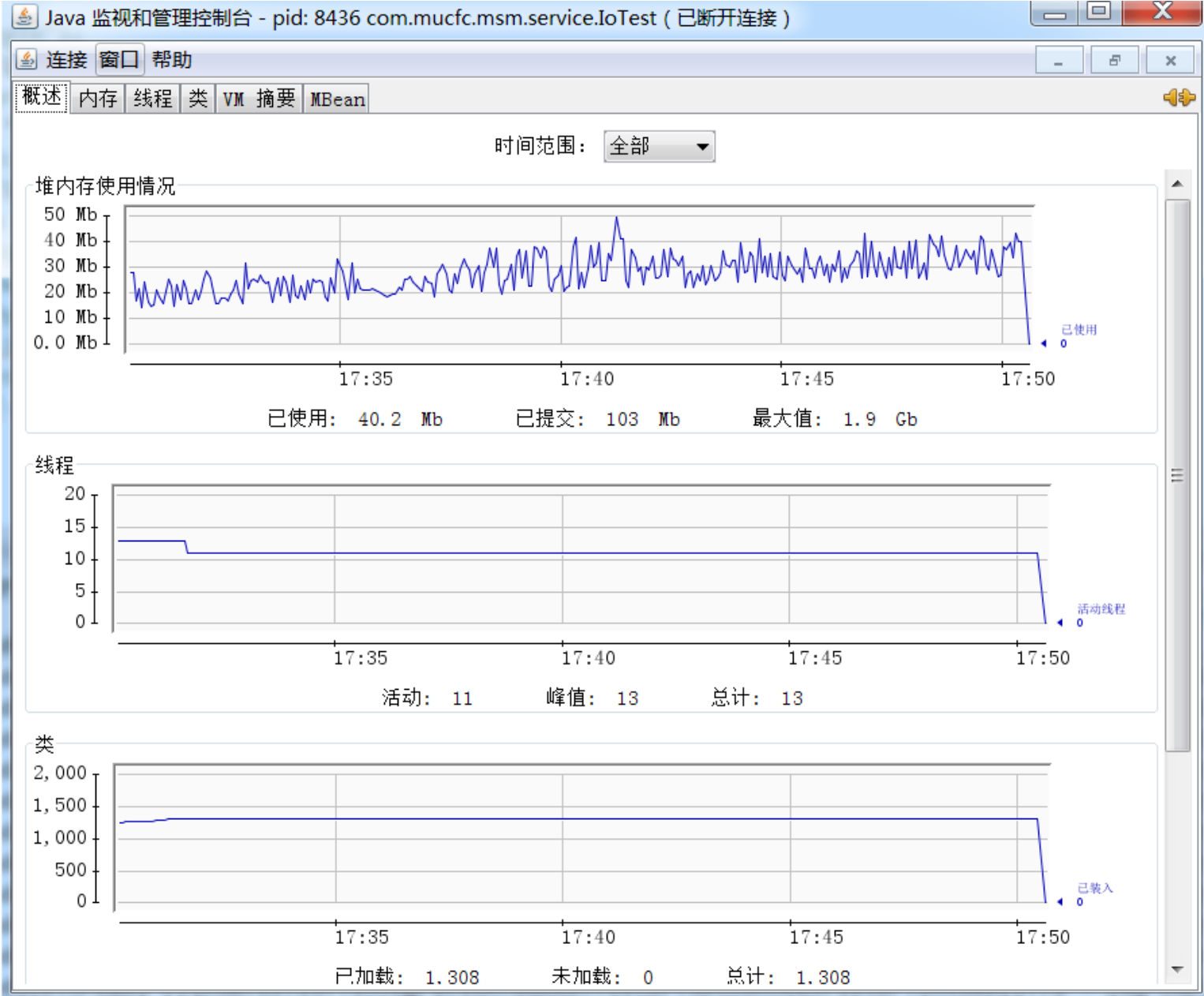
6
2



传统IO快多少，甚至还更加慢了，有可能是因为磁盘IO操作多了，反而降低了其效率，内存映射看来还是对大文件比较有好的效果。小文件基本上是没有多大的差别的。

区大小，总共耗时：1245111ms

耗时：1223877ms（大概20分钟多点）



了 (11-22 21:57 #2楼)	6	
(09-16 16:41 #1楼)	2	
<div>文件其实很容易</div> <div>直接认为，这太难了，这涉及到内存的容量，硬盘的读取速度以及虚拟内存、页失败载入等概念，其...</div>		
<div>文件原理与DirectMemory</div> <div>据说有一个新功能是内存映射文件，日常编程中并不是经常用到，但是在处理大文件时是比较理想的...</div>		
<div>读取并处理超大文件</div> <div>文件的方法，但由于是单线程读写，所以当读取数据量比较大的文件时性能肯定会有是个大问题，所以...</div>		
<div>牛，完美解决中文乱码问题</div> <div>个开发任务，要将百万行级别的txt数据插入到数据库中，由于内存方面的原因，因此不可能一次读取...</div>		
<div>dByteBuffer进行缓冲</div> <div>ffer进行缓冲，这样可以保证边读取大文件，边进行处理 package sean; import java.io.ByteArrayInp...</div>		
<div>件</div> <div>效率要比传统IO效率要更高 两者主要区别 IO NIO 面向流 面向缓冲 阻塞IO 非...</div>		
<div>nio的完全发挥</div> <div>读写大文件的方法，有兴趣的可以看看`package com.nio; import java.io.BufferedReader; import jav...</div>		
<div>牛</div> <div>的TXT文件，并入库。用之前的FileInputStream、BufferedReader显然不行了，虽然readLine这方法...</div>		
<div></div> <div>ava.io.FileInputStream; import java.io.FileOutputStream; impor...</div>		
<div></div> <div>使用java nio进行读写， 根据个人的需求 可能需要将一个超大文件读写形成很多较小的文件进行分析...</div>		
<div>模板+参数 java收集控制台一行 java代码抽奖 java 对象动态堆</div>		
<div>的~ java java和--</div>		

卡

文件的操作，很多时候都是读写超大文件，记录如下： 一、读文件 import java.io.BufferedOutput...

实现

高效地读取大文件。Java——回归基础。 2、在内存中读取 读取文件行的标准方式是在内存中读取...

卡可以提高效率

理大文件可以提高效率。 为什么呢？ 我们先来看看如果不使用内存映射文件的处理流程是怎样的， ...

卡nio 之MappedByteBuffer，高效文件/内存映射

7023 java处理大文件，一般用BufferedReader,BufferedReaderInputStre...

卡

读取一个200M的文本格式文件，而且还需要对文件的内容做解析，进行分词。如果用JVM的默认设...

ByteBuffer

的作用可以把一个文件映射到内存中，然后就能像访问数组一样去读取这个文件。这样不用多线程也...

卡

face，但又没啥材料，就写点关于技术类的吧。 记得半年前做毕业设计，遇到过这样一个情况，需...

卡("")用法

意思，它把给定的字符串解析成Java的各种基本数据类型primitive types，用于分解字符串的默认的分...

[illegible]

没有提供同步的机制，需要借助其他手段或者自行实现。 以下转自:<http://itindex.net/detail/49906-java-...>

映射文件 (Memory Mapped File) 或者MappedByteBuffer 1.4万

字映射文件 (Memory Mapped Files) 就已经在java.nio包中, 但它对很多程序开发者来说仍然是一个...

映射文件及文件通道到通道批量传输数据

文件MappedByteBuffer与zerocopy

1.方法-块读取/异步/优化分析算法/内存文件映射的原理和使用

射技术解决问题: ☐ 1.不要复制文件中所有的数据,只需要修改文件中局部的数据。 ☐ 2.并行/分段...

文件处理  90

[ankaka/article/details/48464013](#) 摘要: 本文主要讲了java中内存映射的原理及过程, 与传...

大文件的读写

富的操作手段，如：1. FILE *fp, fstream...; (C/C++) 2. CFile, CStdioFile...; (MFC) 3. CreateFi...

文件示例 🔍 👁 223


，但是，如果文件大小超过内存大小那怎么办呢 其实，我们在把内存映射文件映射到进程的地址空...

文件非常特别，它允许Java程序直接从内存中读取文件内容，通过将整个或部分文件映射到内存，由...

工具的实现(使用内存映射文件) 06-10

拷贝操作接口使用了内存映射模型的方式实现，可以指定线程数量，可以在拷贝过程中查看整体的进度信息（进度、速度、剩余时间、已用时间...

射技术解决问题: □ 1.不要复制文件中所有的数据,只需要修改文件中局部的数据。 □ 2.并行/分段处理大文件。 如下代码示使用javaNIO局部修改文件中指定位置的...

iter()方法使用  561

类java.util.Scanner来完成的。默认情况下，Scanner是使用“空白”作为分隔符将输入分解为标记，然后使用它所提供的不同的next方法将得到的标记转换...

inner的用法  2.7万

File; import java.io.FileInputStream; import java.io.RandomAccess... 913

大文件支持 GB级别-修正版 10-24

又开发的类。采用读文件的缓存 fbb 1024*5 行缓存 bb 256 字节 设计思想：每次通过nio读取字节到 fbb中 然后对fbb自己中的内容进行行判...

块内存的映射。与虚拟内存有些类似，通过内存映射文件可以保留一个地址空间的区域，同时将物理存储器提交给此区域，内存文件映射的物理存储器来自一个已经存在的磁盘上的文件，而且在...

涉及到io处理、并发编程、生产者/消费者模式的理解，是一个很好的综合应用场景，为此，花点时间做一些实践，对相关的知识做一次梳理和集成，总结一...

附件 06-09

uffer, 高效文件/内存映射

的文件内存映射方案, 读写性能极高。NIO最主要的就是实现了对异步操作的支持。其中一种通过把一个套接字通道(SocketChannel)注册到一个选...

Java并发编程与技术内幕：ForkJoin 框架初探

Mysql各种超时时间理解

Java并发编程与技术内幕:volatile的那些事
常见限流算法研究与实现

Java 8编程进阶-Stream之函数式编程

博主专栏



Cocos2d-x游戏开发实战

阅读量：250285 23 篇



零基础学习Spring

阅读量：207396 18 篇



Servlet/JSP基础指南

阅读量：124296 13 篇



跟我学AngularJs

阅读量：99411 9 篇



Java并发编程与技术内幕

阅读量：89787 8 篇

热门文章

- Jenkins详细安装与构建部署使用教程

阅读量：108362
- Java多线程学习（吐血超详细总结）

阅读量：104733
- Spring+Mybatis+SpringMVC+Maven+MySql搭建实例

阅读量：61110
- Servlet入门总结及第一个Servlet程序

阅读量：37557
- Kafka在Windows安装运行

阅读量：36806

最新评论

- Java并发编程与技术内幕:Con...

u013271656：很棒
- Mybatis自动生成代码

qq_39530754： 向大佬学习
- Docker部署JavaWeb项目...

qq_39530754： 很佩服你这样的大神！
- Jenkins详细安装与构建部署使...

BayBaymax：[code=objc] 感谢作者的无私分享，分享目前主流的技术干货教程：SpringBoot+Sp...
- Java多线程学习（吐血超详细总结）

l912943297：[reply]Dxx_23[/reply] 顶一个！

个人分类

Spring	25篇
Java并发编程与技术内幕	17篇
Hadoop实战演练	10篇
Hive/Hbase编程指南	9篇
Spark技术研究	7篇
展开	

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

6
2



CSDN APP

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

6
2