

纸上得来终觉浅 绝知此事要躬行——<https://github.com/sgq0085/learn>

- [博客](#)
- [微博](#)
- [相册](#)
- [收藏](#)
- [留言](#)
- [关于我](#)

[高效读取大数据文本文件（上亿行数据）](#)

- 博客分类:
- [Java基础-流与文件](#)

[LineNumberReaderRandomAccessFileBufferedRandomAccessFile](#) 文本读取大数据

一.前言

本文是对大数据文本文件读取（按行读取）的优化，目前常规的方案（限于JDK）有三种，第一种LineNumberReader，第二种RandomAccessFile，第三种是内存映射文件（详见<http://sgq0085.iteye.com/blog/1318622>）在RandomAccessFile基础上调用getChannel().map(...)。

1.LineNumberReader

按行读取，只能从第一行向后遍历，到需要读取的行时开始读入，直到完成；在我的测试用例中，读取1000W行数据每次5万行，用时93秒，效率实测比RandomAccessFile要高，但读取一亿跳数据时效率太低了（因为每次都要从头遍历），因为测试时超过1个小时，放弃测试；

2.RandomAccessFile

实际不适用于这种大数据读取，RandomAccessFile是为了磁盘文件的随机访问，所以效率很低，1000w行测试时用时140秒，一亿行数据测试用时1438秒但由于可以通过getFilePointer方法记录位置，并通过seek方法指定读取位置，所以从理论上比较适用这种大数据按行读取的场景；

RandomAccessFile只能按照8859_1这种方法读取，所以需要内容重新编码，方法如下

Java代码 🌟 ☆

```
1. new String(pin.getBytes("8859_1"), "")
```

3.内存映射文件

由于每行数据大小不同，内存映射文件在这种情况下不适用，其他情况请参考我的博客（详见<http://sgq0085.iteye.com/blog/1318622>）

二.解决方案

如果在RandomAccessFile基础上，整合内部缓冲区，效率会有提高，测试过程中1000w行数据用时1秒，1亿行数据用时103（比1438秒快了13倍左右）

BufferedRandomAccessFile

网上已经有实现，代码如下：

Java代码 🌟 ☆

```
1. package com.gqshao.file.io;
2.
3. import java.io.File;
4. import java.io.FileNotFoundException;
5. import java.io.IOException;
6. import java.io.RandomAccessFile;
7. import java.util.Arrays;
8.
9. public class BufferedRandomAccessFile extends RandomAccessFile {
10.     static final int LogBuffSz_ = 16; // 64K buffer
11.     public static final int BuffSz_ = (1 << LogBuffSz_);
12.     static final long BuffMask_ = ~(((long) BuffSz_) - 1L);
13.
14.     private String path_;
15.
16.     /*
17.      * This implementation is based on the buffer implementation in Modula-3's
18.      * "Rd", "Wr", "RdClass", and "WrClass" interfaces.
19.      */
20.     private boolean dirty_; // true iff unflushed bytes exist
21.     private boolean syncNeeded_; // dirty_ can be cleared by e.g. seek, so track sync separately
22.     private long curr_; // current position in file
23.     private long lo_, hi_; // bounds on characters in "buff"
24.     private byte[] buff_; // local buffer
25.     private long maxHi_; // this.lo + this.buff.length
26.     private boolean hitEOF_; // buffer contains last file block?
27.     private long diskPos_; // disk position
28.
29.     /*
30.      * To describe the above fields, we introduce the following abstractions for
31.      * the file "f":
32.      *
33.      * len(f) the length of the file curr(f) the current position in the file
34.      * c(f) the abstract contents of the file disk(f) the contents of f's
35.      * backing disk file closed(f) true iff the file is closed
36.      *
37.      * "curr(f)" is an index in the closed interval [0, len(f)]. "c(f)" is a
38.      * character sequence of length "len(f)". "c(f)" and "disk(f)" may differ if
39.      * "c(f)" contains unflushed writes not reflected in "disk(f)". The flush
40.      * operation has the effect of making "disk(f)" identical to "c(f)".
41.      *
42.      * A file is said to be *valid* if the following conditions hold:
43.      *
44.      * V1. The "closed" and "curr" fields are correct:
45.      *
46.      * f.closed == closed(f) f.curr == curr(f)
```

```
47. *
48. * V2. The current position is either contained in the buffer, or just past
49. * the buffer:
50. *
51. * f.lo <= f.curr <= f.hi
52. *
53. * V3. Any (possibly) unflushed characters are stored in "f.buff":
54. *
55. * (forall i in [f.lo, f.curr): c(f)[i] == f.buff[i - f.lo])
56. *
57. * V4. For all characters not covered by V3, c(f) and disk(f) agree:
58. *
59. * (forall i in [f.lo, len(f)): i not in [f.lo, f.curr) => c(f)[i] ==
60. * disk(f)[i])
61. *
62. * V5. "f.dirty" is true iff the buffer contains bytes that should be
63. * flushed to the file; by V3 and V4, only part of the buffer can be dirty.
64. *
65. * f.dirty == (exists i in [f.lo, f.curr): c(f)[i] != f.buff[i - f.lo])
66. *
67. * V6. this.maxHi == this.lo + this.buff.length
68. *
69. * Note that "f.buff" can be "null" in a valid file, since the range of
70. * characters in V3 is empty when "f.lo == f.curr".
71. *
72. * A file is said to be *ready* if the buffer contains the current position,
73. * i.e., when:
74. *
75. * R1. !f.closed && f.buff != null && f.lo <= f.curr && f.curr < f.hi
76. *
77. * When a file is ready, reading or writing a single byte can be performed
78. * by reading or writing the in-memory buffer without performing a disk
79. * operation.
80. */
81.
82. /**
83. * Open a new <code>BufferedRandomAccessFile</code> on <code>file</code>
84. * in mode <code>mode</code>, which should be "r" for reading only, or
85. * "rw" for reading and writing.
86. */
87. public BufferedRandomAccessFile(File file, String mode) throws IOException {
88.     this(file, mode, 0);
89. }
90.
91. public BufferedRandomAccessFile(File file, String mode, int size) throws IOException {
92.     super(file, mode);
93.     path_ = file.getAbsolutePath();
94.     this.init(size);
95. }
96.
97. /**
98. * Open a new <code>BufferedRandomAccessFile</code> on the file named
99. * <code>name</code> in mode <code>mode</code>, which should be "r" for
100. * reading only, or "rw" for reading and writing.
101. */
102. public BufferedRandomAccessFile(String name, String mode) throws IOException {
103.     this(name, mode, 0);
104. }
105.
106. public BufferedRandomAccessFile(String name, String mode, int size) throws FileNotFoundException {
107.     super(name, mode);
108.     path_ = name;
109.     this.init(size);
110. }
111.
112. private void init(int size) {
113.     this.dirty_ = false;
114.     this.lo_ = this.curr_ = this.hi_ = 0;
115.     this.buff_ = (size > BuffSz_) ? new byte[size] : new byte[BuffSz_];
116.     this.maxHi_ = (long) BuffSz_;
117.     this.hitEOF_ = false;
118.     this.diskPos_ = 0L;
119. }
120.
121. public String getPath() {
122.     return path_;
123. }
124.
125. public void sync() throws IOException {
126.     if (syncNeeded_) {
127.         flush();
128.         getChannel().force(true);
129.         syncNeeded_ = false;
130.     }
131. }
132.
133. // public boolean isEOF() throws IOException
134. // {
135. //     assert getFilePointer() <= length();
136. //     return getFilePointer() == length();
137. // }
138.
139. public void close() throws IOException {
140.     this.flush();
141.     this.buff_ = null;
142.     super.close();
143. }
144.
145. /**
146. * Flush any bytes in the file's buffer that have not yet been written to
147. * disk. If the file was created read-only, this method is a no-op.
148. */
149. public void flush() throws IOException {
150.     this.flushBuffer();
151. }
152.
153. /* Flush any dirty bytes in the buffer to disk. */
154. private void flushBuffer() throws IOException {
155.     if (this.dirty_) {
156.         if (this.diskPos_ != this.lo_)
157.             super.seek(this.lo_);
158.         int len = (int) (this.curr_ - this.lo_);
159.         super.write(this.buff_, 0, len);
160.         this.diskPos_ = this.curr_;
161.         this.dirty_ = false;
162.     }
163. }
164.
165. /*
166. * Read at most "this.buff.length" bytes into "this.buff", returning the
167. * number of bytes read. If the return result is less than
168. * "this.buff.length", then EOF was read.
169. */
170. private int fillBuffer() throws IOException {
171.     int cnt = 0;
172.     int rem = this.buff_.length;
173.     while (rem > 0) {
174.         int n = super.read(this.buff_, cnt, rem);
175.         if (n < 0)
176.             break;
177.         cnt += n;
178.         rem -= n;
179.     }
180.     if ((cnt < 0) && (this.hitEOF_ = (cnt < this.buff_.length))) {
181.         // make sure buffer that wasn't read is initialized with -1
```



```
182.     Arrays.fill(this.buff_, cnt, this.buff_.length, (byte) 0xff);
183. }
184. this.diskPos_ += cnt;
185. return cnt;
186. }
187.
188. /*
189.  * This method positions <code>this.curr</code> at position <code>pos</code>.
190.  * If <code>pos</code> does not fall in the current buffer, it flushes the
191.  * current buffer and loads the correct one.<p>
192.  *
193.  * On exit from this routine <code>this.curr == this.hi</code> iff <code>pos</code>
194.  * is at or past the end-of-file, which can only happen if the file was
195.  * opened in read-only mode.
196.  */
197. public void seek(long pos) throws IOException {
198.     if (pos >= this.hi_ || pos < this.lo_) {
199.         // seeking outside of current buffer -- flush and read
200.         this.flushBuffer();
201.         this.lo_ = pos & BuffMask_; // start at BuffSz boundary
202.         this.maxHi_ = this.lo_ + (long) this.buff_.length;
203.         if (this.diskPos_ != this.lo_) {
204.             super.seek(this.lo_);
205.             this.diskPos_ = this.lo_;
206.         }
207.         int n = this.fillBuffer();
208.         this.hi_ = this.lo_ + (long) n;
209.     } else {
210.         // seeking inside current buffer -- no read required
211.         if (pos < this.curr_) {
212.             // if seeking backwards, we must flush to maintain V4
213.             this.flushBuffer();
214.         }
215.     }
216.     this.curr_ = pos;
217. }
218.
219. public long getFilePointer() {
220.     return this.curr_;
221. }
222.
223. public long length() throws IOException {
224.     // max accounts for the case where we have written past the old file length, but not yet flushed our buffer
225.     return Math.max(this.curr_, super.length());
226. }
227.
228. public int read() throws IOException {
229.     if (this.curr_ >= this.hi_) {
230.         // test for EOF
231.         // if (this.hi_ < this.maxHi) return -1;
232.         if (this.hitEOF_)
233.             return -1;
234.
235.         // slow path -- read another buffer
236.         this.seek(this.curr_);
237.         if (this.curr_ == this.hi_)
238.             return -1;
239.     }
240.     byte res = this.buff_[(int) (this.curr_ - this.lo_)];
241.     this.curr_++;
242.     return ((int) res) & 0xFF; // convert byte -> int
243. }
244.
245. public int read(byte[] b) throws IOException {
246.     return this.read(b, 0, b.length);
247. }
248.
249. public int read(byte[] b, int off, int len) throws IOException {
250.     if (this.curr_ >= this.hi_) {
251.         // test for EOF
252.         // if (this.hi_ < this.maxHi) return -1;
253.         if (this.hitEOF_)
254.             return -1;
255.
256.         // slow path -- read another buffer
257.         this.seek(this.curr_);
258.         if (this.curr_ == this.hi_)
259.             return -1;
260.     }
261.     len = Math.min(len, (int) (this.hi_ - this.curr_));
262.     int buffOff = (int) (this.curr_ - this.lo_);
263.     System.arraycopy(this.buff_, buffOff, b, off, len);
264.     this.curr_ += len;
265.     return len;
266. }
267.
268. public void write(int b) throws IOException {
269.     if (this.curr_ >= this.hi_) {
270.         if (this.hitEOF_ && this.hi_ < this.maxHi_) {
271.             // at EOF -- bump "hi"
272.             this.hi_++;
273.         } else {
274.             // slow path -- write current buffer; read next one
275.             this.seek(this.curr_);
276.             if (this.curr_ == this.hi_) {
277.                 // appending to EOF -- bump "hi"
278.                 this.hi_++;
279.             }
280.         }
281.     }
282.     this.buff_[(int) (this.curr_ - this.lo_)] = (byte) b;
283.     this.curr_++;
284.     this.dirty_ = true;
285.     syncNeeded_ = true;
286. }
287.
288. public void write(byte[] b) throws IOException {
289.     this.write(b, 0, b.length);
290. }
291.
292. public void write(byte[] b, int off, int len) throws IOException {
293.     while (len > 0) {
294.         int n = this.writeAtMost(b, off, len);
295.         off += n;
296.         len -= n;
297.         this.dirty_ = true;
298.         syncNeeded_ = true;
299.     }
300. }
301.
302. /*
303.  * Write at most "len" bytes to "b" starting at position "off", and return
304.  * the number of bytes written.
305.  */
306. private int writeAtMost(byte[] b, int off, int len) throws IOException {
307.     if (this.curr_ >= this.hi_) {
308.         if (this.hitEOF_ && this.hi_ < this.maxHi_) {
309.             // at EOF -- bump "hi"
310.             this.hi_ = this.maxHi_;
311.         } else {
312.             // slow path -- write current buffer; read next one
313.             this.seek(this.curr_);
314.             if (this.curr_ == this.hi_) {
315.                 // appending to EOF -- bump "hi"
316.                 this.hi_ = this.maxHi_;
```

```
317.         }
318.     }
319. }
320. len = Math.min(len, (int) (this.hi_ - this.curr_));
321. int buffOff = (int) (this.curr_ - this.lo_);
322. System.arraycopy(b, off, this.buff_, buffOff, len);
323. this.curr_ += len;
324. return len;
325. }
326. }
```

三.测试

1.FileUtil

用于封装三种方案（LineNumberReader、RandomAccessFile、BufferedRandomAccessFile）的文件读取

Java代码  

```
1. package com.gqshao.file.util;
2.
3. import com.google.common.collect.Lists;
4. import com.google.common.collect.Maps;
5. import com.gqshao.file.io.BufferedRandomAccessFile;
6. import org.apache.commons.io.IOUtils;
7. import org.apache.commons.lang3.StringUtils;
8.
9. import java.io.*;
10. import java.util.List;
11. import java.util.Map;
12.
13. public class FileUtil {
14.
15.     /**
16.      * 通过BufferedRandomAccessFile读取文件,推荐
17.      *
18.      * @param file    源文件
19.      * @param encoding 文件编码
20.      * @param pos      偏移量
21.      * @param num      读取量
22.      * @return pins 文件内容， pos当前偏移量
23.      */
24.     public static Map<String, Object> BufferedRandomAccessFileReadLine(File file, String encoding, long pos, int num) {
25.         Map<String, Object> res = Maps.newHashMap();
26.         List<String> pins = Lists.newArrayList();
27.         res.put("pins", pins);
28.         BufferedRandomAccessFile reader = null;
29.         try {
30.             reader = new BufferedRandomAccessFile(file, "r");
31.             reader.seek(pos);
32.             for (int i = 0; i < num; i++) {
33.                 String pin = reader.readLine();
34.                 if (StringUtils.isBlank(pin)) {
35.                     break;
36.                 }
37.                 pins.add(new String(pin.getBytes("8859_1"), encoding));
38.             }
39.             res.put("pos", reader.getFilePointer());
40.         } catch (Exception e) {
41.             e.printStackTrace();
42.         } finally {
43.             IOUtils.closeQuietly(reader);
44.         }
45.         return res;
46.     }
47.
48.     /**
49.      * 通过RandomAccessFile读取文件，能出来大数据文件，效率低
50.      *
51.      * @param file    源文件
52.      * @param encoding 文件编码
53.      * @param pos      偏移量
54.      * @param num      读取量
55.      * @return pins 文件内容， pos当前偏移量
56.      */
57.     public static Map<String, Object> readLine(File file, String encoding, long pos, int num) {
58.         Map<String, Object> res = Maps.newHashMap();
59.         List<String> pins = Lists.newArrayList();
60.         res.put("pins", pins);
61.         RandomAccessFile reader = null;
62.         try {
63.             reader = new RandomAccessFile(file, "r");
64.             reader.seek(pos);
65.             for (int i = 0; i < num; i++) {
66.                 String pin = reader.readLine();
67.                 if (StringUtils.isBlank(pin)) {
68.                     break;
69.                 }
70.                 pins.add(new String(pin.getBytes("8859_1"), encoding));
71.             }
72.             res.put("pos", reader.getFilePointer());
73.         } catch (Exception e) {
74.             e.printStackTrace();
75.         } finally {
76.             IOUtils.closeQuietly(reader);
77.         }
78.         return res;
79.     }
80.
81.     /**
82.      * 使用LineNumberReader读取文件，1000w行比RandomAccessFile效率高，无法处理1亿条数据
83.      *
84.      * @param file    源文件
85.      * @param encoding 文件编码
86.      * @param index    开始位置
87.      * @param num      读取量
88.      * @return pins 文件内容
89.      */
90.     public static List<String> readLine(File file, String encoding, int index, int num) {
91.         List<String> pins = Lists.newArrayList();
92.         LineNumberReader reader = null;
93.         try {
94.             reader = new LineNumberReader(new InputStreamReader(new FileInputStream(file), encoding));
95.             int lines = 0;
96.             while (true) {
97.                 String pin = reader.readLine();
98.                 if (StringUtils.isBlank(pin)) {
99.                     break;
100.                }
101.                if (lines >= index) {
102.                    if (StringUtils.isNotBlank(pin)) {
103.                        pins.add(pin);
104.                    }

```



```
105.     }
106.     if (num == pins.size()) {
107.         break;
108.     }
109.     lines++;
110. }
111. } catch (Exception e) {
112.     e.printStackTrace();
113. } finally {
114.     IOUtils.closeQuietly(reader);
115. }
116. return pins;
117. }
118.
119.
120. }
```

2.RandomAccessFileTest

测试方法，涉及到的randomFile只是一个掺杂中文的文本文件，可以自己随便写一个

Java代码 🌸 ☆

```
1. package com.gqshao.file;
2.
3. import com.gqshao.file.util.FileUtil;
4. import org.apache.commons.collections.CollectionUtils;
5. import org.apache.commons.collections.MapUtils;
6. import org.apache.commons.io.IOUtils;
7. import org.junit.Test;
8. import org.slf4j.Logger;
9. import org.slf4j.LoggerFactory;
10.
11. import java.io.*;
12. import java.util.List;
13. import java.util.Map;
14.
15. public class RandomAccessFileTest {
16.
17.     private static final Logger logger = LoggerFactory.getLogger(RandomAccessFileTest.class);
18.
19.     private static final String ENCODING = "UTF-8";
20.     private static final int NUM = 50000;
21.
22.     private static File file = new File(ClassLoader.getResource("").getPath() + File.separator + "test.txt");
23.     private static File randomFile = new File(ClassLoader.getResource("").getPath() + File.separator + "RandomFile.txt");
24.
25.     /**
26.      * 生成1000w随机文本文件
27.      */
28.     @Test
29.     public void makePin() {
30.         String prefix = "_$#";
31.         OutputStreamWriter out = null;
32.         try {
33.             out = new OutputStreamWriter(new FileOutputStream(file, true), ENCODING);
34.             // 在1500w里随机1000w数据
35.             for (int j = 0; j < 100000000; j++) {
36.                 out.write(prefix + (int) (130000000 * Math.random()) + "\n");
37.             }
38.         } catch (Exception e) {
39.             e.printStackTrace();
40.         } finally {
41.             IOUtils.closeQuietly(out);
42.         }
43.         logger.info(file.getAbsolutePath());
44.     }
45.
46.     /**
47.      * 测试RandomAccessFile读取文件
48.      */
49.     @Test
50.     public void testRandomAccessRead() {
51.         long start = System.currentTimeMillis();
52.         //
53.         logger.info(String.valueOf(file.exists()));
54.         long pos = 0L;
55.         while (true) {
56.             Map<String, Object> res = FileUtil.readLine(file, ENCODING, pos, NUM);
57.             // 如果返回结果为空结束循环
58.             if (MapUtils.isEmpty(res)) {
59.                 break;
60.             }
61.             Object po = res.get("pins");
62.             List<String> pins = (List<String>) res.get("pins");
63.             if (CollectionUtils.isNotEmpty(pins)) {
64.                 // logger.info(Arrays.toString(pins.toArray()));
65.                 if (pins.size() < NUM) {
66.                     break;
67.                 }
68.             } else {
69.                 break;
70.             }
71.             pos = (Long) res.get("pos");
72.         }
73.         logger.info(((System.currentTimeMillis() - start) / 1000) + "");
74.     }
75.
76.     /**
77.      * 测试RandomAccessFile读取文件
78.      */
79.     @Test
80.     public void testBufferedRandomAccessRead() {
81.         long start = System.currentTimeMillis();
82.         //
83.         logger.info(String.valueOf(file.exists()));
84.         long pos = 0L;
85.         while (true) {
86.             Map<String, Object> res = FileUtil.BufferedRandomAccessFileReadLine(file, ENCODING, pos, NUM);
87.             // 如果返回结果为空结束循环
88.             if (MapUtils.isEmpty(res)) {
89.                 break;
90.             }
91.             List<String> pins = (List<String>) res.get("pins");
92.             if (CollectionUtils.isNotEmpty(pins)) {
93.                 // logger.info(Arrays.toString(pins.toArray()));
94.                 if (pins.size() < NUM) {
95.                     break;
96.                 }
97.             } else {
98.                 break;
99.             }
100.             pos = (Long) res.get("pos");
101.         }
102.         logger.info(((System.currentTimeMillis() - start) / 1000) + "");
103.     }
104.
105.     /**
106.      * 测试普通读取文件
107.      */
108.     @Test
```



```
109. public void testCommonRead() {
110.     long start = System.currentTimeMillis();
111.     logger.info(String.valueOf(randomFile.exists()));
112.     int index = 0;
113.     while (true) {
114.         List<String> pins = FileUtil.readLine(file, ENCODING, index, NUM);
115.         if (CollectionUtils.isNotEmpty(pins)) {
116. //             logger.info(Arrays.toString(pins.toArray()));
117.             if (pins.size() < NUM) {
118.                 break;
119.             }
120.         } else {
121.             break;
122.         }
123.         index += NUM;
124.     }
125.     logger.info(((System.currentTimeMillis() - start) / 1000) + "");
126. }
127. }
```

4

顶

1

踩

分享到： 

[MurmurHash一致性Hash算法JAVA版](#) | [Thrift 简单使用](#)

- 2015-05-29 11:41
- 浏览 5926
- [评论\(3\)](#)
- 分类: [编程语言](#)
- [查看更多](#)

评论

3 楼 [wx3957156](#) 2015-06-11



2 楼 [sgq0085](#) 2015-06-01

zx_code 写道

lz，有没有尝试多线程读取，估计性能更快

出错的概率更高吧？ 这里涉及到偏移量在各个线程中的传递，可能会出问题

1 楼 [zx_code](#) 2015-06-01

lz，有没有尝试多线程读取，估计性能更快

发表评论



[您还没有登录,请您登录后再发表评论](#)

相关资源推荐

[高效读取大数据文本文件（上亿行数据）](#)

[一个文本文件，找出前10个经常出现的词，但这次文件比较长，说是上亿行或十亿行，总之无法一次...](#)

[c++读取文本文件最高效的方法](#)

[文本文件大数据查询](#)

[C# 读取大文件 （可以读取3GB大小的txt文件）](#)

[Pandas100秒处理一亿行数据](#)

[Java读取和操作大数据文本数据](#)

[python 高效去重复 支持GB级别大文件](#)

[mysql分区技术-单表的大数据处理](#)

[亿级别---数据生成及高效率导入](#)

[Java 高效读取大数据文件—最优方法](#)

[Java对大文件的高效读取方法](#)

[C#实现大数据量TXT文本数据快速高效去重](#)

[基于大文本文件的文本读取查找](#)

[大数据算法：对5亿数据进行排序](#)

[较大数据文件的读取优化过程](#)

[大数据排序或取重或去重相关问题](#)



[SQLServer大量数据高效率分页](#)

[Java_IO流_按行读取文本文件的内容并按行写入到另一文件](#)

[java读取大文本文件](#)



sgq0085

- 浏览: 279680 次
- 性别: 
- 来自: 吉林→上海
-  我现在离綫

最近访客

[更多访客>>](#)



[farwind](#)



[linkyhu](#)



[nucleus](#)



[aaron198](#)

文章分类

- [全部博客 \(174\)](#)
- [Java基础-接口与内部类 \(5\)](#)
- [Java基础-流与文件 \(10\)](#)
- [Java基础-JDBC \(12\)](#)
- [Java基础-XML解析 \(7\)](#)
- [Java基础-多线程 \(11\)](#)
- [Java基础-网络 \(6\)](#)
- [Java基础-注解 \(5\)](#)
- [Hibernate 研究记录 \(7\)](#)
- [JavaScript 研究记录 \(6\)](#)

- [ECMAScript 研究记录 \(7\)](#)
- [CSS 研究记录 \(9\)](#)
- [Maven 研究记录 \(8\)](#)
- [SQL 随笔 \(5\)](#)
- [权限控制和单点登陆 \(8\)](#)
- [Hadoop 研究记录 \(6\)](#)
- [随想杂谈 \(33\)](#)
- [JAVA EE \(4\)](#)
- [测试 \(3\)](#)
- [Redis \(10\)](#)
- [Memcached \(2\)](#)
- [MongoDB \(6\)](#)
- [ElasticSearch \(3\)](#)

社区版块

- [我的资讯](#) (0)
- [我的论坛](#) (77)
- [我的问答](#) (25)

存档分类

- [2018-05](#) (3)
- [2018-03](#) (1)
- [2018-02](#) (1)
- [更多存档...](#)

最新评论

- [sgq0085](#): 无尘灬, 写道楼主，在吗？可以加你qq咨询一下问题吗？公司禁用Q ...
[Shiro通过Redis管理会话实现集群](#)
- [无尘灬](#): 楼主，在吗？可以加你qq咨询一下问题吗？
[Shiro通过Redis管理会话实现集群](#)
- [zhouminsen](#): 感谢楼主的无私奉献
[Shiro通过Redis管理会话实现集群](#)
- [tonny1228](#): 经测试还是运行在local
[远程调用执行Hadoop Map/Reduce](#)
- [asdobby](#): 楼主，个人感觉每次调用SessionDAO的doUpdate方 ...
[Shiro通过Redis管理会话实现集群](#)