<http://blog.csdn.net/jiangxinyu/article/details/7885518/>

当我们读写文本文件的时候，采用Reader是非常方便的，比如FileReader，InputStreamReader和BufferedReader。其中最重要的类是InputStreamReader， 它是字节转换为字符的桥梁。你可以在构造器重指定编码的方式，如果不指定的话将采用底层操作系统的默认编码方式，例如GBK等。使用FileReader读取文件：

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. FileReader fr = **new** FileReader("ming.txt");
3. **int** ch = 0;
5. **while**((ch = fr.read())!=-1 )
7. {

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. System.out.print((**char**)ch);

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. }

其中read()方法返回的是读取得下个字符。当然你也可以使用read(char[] ch,int off,int length)这和处理二进制文件的时候类似。

事实上在FileReader中的方法都是从InputStreamReader中继承过来的。read()方法是比较好费时间的，如果为了提高效率我们可以使用BufferedReader对Reader进行包装，这样可以提高读取得速度，我们可以一行一行的读取文本，使用readLine()方法。

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("ming.txt")));  
String data = null;  
while((data = br.readLine())!=null)  
{  
System.out.println(data);   
}

了解了FileReader操作使用FileWriter写文件就简单了，这里不赘述。

Eg.我的综合实例

testFile:

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileNotFoundException;
4. **import** java.io.FileOutputStream;
5. **import** java.io.IOException;
6. **import** java.io.InputStreamReader;
8. **public** **class** testFile {
9. /\*\*
10. \* @param args
11. \*/
12. **public** **static** **void** main(String[] args) {
13. // TODO Auto-generated method stub
14. // file(内存)----输入流---->【程序】----输出流---->file(内存)
15. File file = **new** File("d:/temp", "addfile.txt");
16. **try** {
17. file.createNewFile(); // 创建文件
18. } **catch** (IOException e) {
19. // TODO Auto-generated catch block
20. e.printStackTrace();
21. }
23. // 向文件写入内容(输出流)
24. String str = "亲爱的小南瓜！";
25. **byte** bt[] = **new** **byte**[1024];
26. bt = str.getBytes();
27. **try** {
28. FileOutputStream in = **new** FileOutputStream(file);
29. **try** {
30. in.write(bt, 0, bt.length);
31. in.close();
32. // boolean success=true;
33. // System.out.println("写入文件成功");
34. } **catch** (IOException e) {
35. // TODO Auto-generated catch block
36. e.printStackTrace();
37. }
38. } **catch** (FileNotFoundException e) {
39. // TODO Auto-generated catch block
40. e.printStackTrace();
41. }
42. **try** {
43. // 读取文件内容 (输入流)
44. FileInputStream out = **new** FileInputStream(file);
45. InputStreamReader isr = **new** InputStreamReader(out);
46. **int** ch = 0;
47. **while** ((ch = isr.read()) != -1) {
48. System.out.print((**char**) ch);
49. }
50. } **catch** (Exception e) {
51. // TODO: handle exception
52. }
53. }
54. }

# java中多种方式读文件

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. //------------------参考资料---------------------------------
2. //
3. //1、按字节读取文件内容
4. //2、按字符读取文件内容
5. //3、按行读取文件内容
6. //4、随机读取文件内容
8. **import** java.io.BufferedReader;
9. **import** java.io.File;
10. **import** java.io.FileInputStream;
11. **import** java.io.FileReader;
12. **import** java.io.IOException;
13. **import** java.io.InputStream;
14. **import** java.io.InputStreamReader;
15. **import** java.io.RandomAccessFile;
16. **import** java.io.Reader;
18. **public** **class** ReadFromFile {
19. /\*\*
20. \* 以字节为单位读取文件，常用于读二进制文件，如图片、声音、影像等文件。
21. \*
22. \* @param fileName
23. \*            文件的名
24. \*/
25. **public** **static** **void** readFileByBytes(String fileName) {
26. File file = **new** File(fileName);
27. InputStream in = **null**;
28. **try** {
29. System.out.println("以字节为单位读取文件内容，一次读一个字节：");
30. // 一次读一个字节
31. in = **new** FileInputStream(file);
32. **int** tempbyte;
33. **while** ((tempbyte = in.read()) != -1) {
34. System.out.write(tempbyte);
35. }
36. in.close();
37. } **catch** (IOException e) {
38. e.printStackTrace();
39. **return**;
40. }
41. **try** {
42. System.out.println("以字节为单位读取文件内容，一次读多个字节：");
43. // 一次读多个字节
44. **byte**[] tempbytes = **new** **byte**[100];
45. **int** byteread = 0;
46. in = **new** FileInputStream(fileName);
47. ReadFromFile.showAvailableBytes(in);
48. // 读入多个字节到字节数组中，byteread为一次读入的字节数
49. **while** ((byteread = in.read(tempbytes)) != -1) {
50. System.out.write(tempbytes, 0, byteread);
51. }
52. } **catch** (Exception e1) {
53. e1.printStackTrace();
54. } **finally** {
55. **if** (in != **null**) {
56. **try** {
57. in.close();
58. } **catch** (IOException e1) {
59. }
60. }
61. }
62. }
64. /\*\*
65. \* 以字符为单位读取文件，常用于读文本，数字等类型的文件
66. \*
67. \* @param fileName
68. \*            文件名
69. \*/
70. **public** **static** **void** readFileByChars(String fileName) {
71. File file = **new** File(fileName);
72. Reader reader = **null**;
73. **try** {
74. System.out.println("以字符为单位读取文件内容，一次读一个字节：");
75. // 一次读一个字符
76. reader = **new** InputStreamReader(**new** FileInputStream(file));
77. **int** tempchar;
78. **while** ((tempchar = reader.read()) != -1) {
79. // 对于windows下，rn这两个字符在一起时，表示一个换行。
80. // 但如果这两个字符分开显示时，会换两次行。
81. // 因此，屏蔽掉r，或者屏蔽n。否则，将会多出很多空行。
82. **if** (((**char**) tempchar) != 'r') {
83. System.out.print((**char**) tempchar);
84. }
85. }
86. reader.close();
87. } **catch** (Exception e) {
88. e.printStackTrace();
89. }
90. **try** {
91. System.out.println("以字符为单位读取文件内容，一次读多个字节：");
92. // 一次读多个字符
93. **char**[] tempchars = **new** **char**[30];
94. **int** charread = 0;
95. reader = **new** InputStreamReader(**new** FileInputStream(fileName));
96. // 读入多个字符到字符数组中，charread为一次读取字符数
97. **while** ((charread = reader.read(tempchars)) != -1) {
98. // 同样屏蔽掉r不显示
99. **if** ((charread == tempchars.length)
100. && (tempchars[tempchars.length - 1] != 'r')) {
101. System.out.print(tempchars);
102. } **else** {
103. **for** (**int** i = 0; i < charread; i++) {
104. **if** (tempchars[i] == 'r') {
105. **continue**;
106. } **else** {
107. System.out.print(tempchars[i]);
108. }
109. }
110. }
111. }
112. } **catch** (Exception e1) {
113. e1.printStackTrace();
114. } **finally** {
115. **if** (reader != **null**) {
116. **try** {
117. reader.close();
118. } **catch** (IOException e1) {
119. }
120. }
121. }
122. }
124. /\*\*
125. \* 以行为单位读取文件，常用于读面向行的格式化文件
126. \*
127. \* @param fileName
128. \*            文件名
129. \*/
130. **public** **static** **void** readFileByLines(String fileName) {
131. File file = **new** File(fileName);
132. BufferedReader reader = **null**;
133. **try** {
134. System.out.println("以行为单位读取文件内容，一次读一整行：");
135. reader = **new** BufferedReader(**new** FileReader(file));
136. String tempString = **null**;
137. **int** line = 1;
138. // 一次读入一行，直到读入null为文件结束
139. **while** ((tempString = reader.readLine()) != **null**) {
140. // 显示行号
141. System.out.println("line " + line + ": " + tempString);
142. line++;
143. }
144. reader.close();
145. } **catch** (IOException e) {
146. e.printStackTrace();
147. } **finally** {
148. **if** (reader != **null**) {
149. **try** {
150. reader.close();
151. } **catch** (IOException e1) {
152. }
153. }
154. }
155. }
157. /\*\*
158. \* 随机读取文件内容
159. \*
160. \* @param fileName
161. \*            文件名
162. \*/
163. **public** **static** **void** readFileByRandomAccess(String fileName) {
164. RandomAccessFile randomFile = **null**;
165. **try** {
166. System.out.println("随机读取一段文件内容：");
167. // 打开一个随机访问文件流，按只读方式
168. randomFile = **new** RandomAccessFile(fileName, "r");
169. // 文件长度，字节数
170. **long** fileLength = randomFile.length();
171. // 读文件的起始位置
172. **int** beginIndex = (fileLength > 4) ? 4 : 0;
173. // 将读文件的开始位置移到beginIndex位置。
174. randomFile.seek(beginIndex);
175. **byte**[] bytes = **new** **byte**[10];
176. **int** byteread = 0;
177. // 一次读10个字节，如果文件内容不足10个字节，则读剩下的字节。
178. // 将一次读取的字节数赋给byteread
179. **while** ((byteread = randomFile.read(bytes)) != -1) {
180. System.out.write(bytes, 0, byteread);
181. }
182. } **catch** (IOException e) {
183. e.printStackTrace();
184. } **finally** {
185. **if** (randomFile != **null**) {
186. **try** {
187. randomFile.close();
188. } **catch** (IOException e1) {
189. }
190. }
191. }
192. }
194. /\*\*
195. \* 显示输入流中还剩的字节数
196. \*
197. \* @param in
198. \*/
199. **private** **static** **void** showAvailableBytes(InputStream in) {
200. **try** {
201. System.out.println("当前字节输入流中的字节数为:" + in.available());
202. } **catch** (IOException e) {
203. e.printStackTrace();
204. }
205. }
207. **public** **static** **void** main(String[] args) {
208. String fileName = "C:/temp/newTemp.txt";
209. ReadFromFile.readFileByBytes(fileName);
210. ReadFromFile.readFileByChars(fileName);
211. ReadFromFile.readFileByLines(fileName);
212. ReadFromFile.readFileByRandomAccess(fileName);
213. }
214. }

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. //二、将内容追加到文件尾部
2. **import** java.io.FileWriter;
3. **import** java.io.IOException;
4. **import** java.io.RandomAccessFile;
6. /\*\*
7. \* 将内容追加到文件尾部
8. \*/
9. **public** **class** AppendToFile {
10. /\*\*
11. \* A方法追加文件：使用RandomAccessFile
12. \*
13. \* @param fileName
14. \*            文件名
15. \* @param content
16. \*            追加的内容
17. \*/
18. **public** **static** **void** appendMethodA(String fileName,
20. String content) {
21. **try** {
22. // 打开一个随机访问文件流，按读写方式
23. RandomAccessFile randomFile = **new** RandomAccessFile(fileName, "rw");
24. // 文件长度，字节数
25. **long** fileLength = randomFile.length();
26. // 将写文件指针移到文件尾。
27. randomFile.seek(fileLength);
28. randomFile.writeBytes(content);
29. randomFile.close();
30. } **catch** (IOException e) {
31. e.printStackTrace();
32. }
33. }
35. /\*\*
36. \* B方法追加文件：使用FileWriter
37. \*
38. \* @param fileName
39. \* @param content
40. \*/
41. **public** **static** **void** appendMethodB(String fileName, String content) {
42. **try** {
43. // 打开一个写文件器，构造函数中的第二个参数true表示以追加形式写文件
44. FileWriter writer = **new** FileWriter(fileName, **true**);
45. writer.write(content);
46. writer.close();
47. } **catch** (IOException e) {
48. e.printStackTrace();
49. }
50. }
52. **public** **static** **void** main(String[] args) {
53. String fileName = "C:/temp/newTemp.txt";
54. String content = "new append!";
55. // 按方法A追加文件
56. AppendToFile.appendMethodA(fileName, content);
57. AppendToFile.appendMethodA(fileName, "append end. n");
58. // 显示文件内容
59. ReadFromFile.readFileByLines(fileName);
60. // 按方法B追加文件
61. AppendToFile.appendMethodB(fileName, content);
62. AppendToFile.appendMethodB(fileName, "append end. n");
63. // 显示文件内容
64. ReadFromFile.readFileByLines(fileName);
65. }
66. }

## 1、判断文件是否存在，不存在创建文件

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. File file=**new** File(path+filename);
2. **if**(!file.exists())
3. {
4. **try** {
5. file.createNewFile();
6. } **catch** (IOException e) {
7. // TODO Auto-generated catch block
8. e.printStackTrace();
9. }
10. }

## 2、判断文件夹是否存在，不存在创建文件夹

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. File file =**new** File(path+filename);
2. //如果文件夹不存在则创建
3. **if**  (!file .exists())
4. {
5. file .mkdir();
6. }

# [java 写文件的三种方法比较](http://www.cnblogs.com/yezhenhan/archive/2012/09/10/2678690.html)

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **import** java.io.File;
3. **import** java.io.FileOutputStream;
5. **import** java.io.\*;
7. **public** **class** FileTest {
9. **public** FileTest() {
11. }
13. **public** **static** **void** main(String[] args) {
15. FileOutputStream out = **null**;
17. FileOutputStream outSTr = **null**;
19. BufferedOutputStream Buff=**null**;
21. FileWriter fw = **null**;
23. **int** count=1000;//写文件行数
25. **try** {
27. out = **new** FileOutputStream(**new** File(“C:/add.txt”));
29. **long** begin = System.currentTimeMillis();
31. **for** (**int** i = 0; i < count; i++) {
33. out.write(“测试java 文件操作\r\n”.getBytes());
35. }
37. out.close();
39. **long** end = System.currentTimeMillis();
41. System.out.println(“FileOutputStream执行耗时:” + (end - begin) + ” 豪秒”);
43. outSTr = **new** FileOutputStream(**new** File(“C:/add0.txt”));
45. Buff=**new** BufferedOutputStream(outSTr);
47. **long** begin0 = System.currentTimeMillis();
49. **for** (**int** i = 0; i < count; i++) {
51. Buff.write(“测试java 文件操作\r\n”.getBytes());
53. }
55. Buff.flush();
57. Buff.close();
59. **long** end0 = System.currentTimeMillis();
61. System.out.println(“BufferedOutputStream执行耗时:” + (end0 - begin0) + ” 豪秒”);
63. fw = **new** FileWriter(“C:/add2.txt”);
65. **long** begin3 = System.currentTimeMillis();
67. **for** (**int** i = 0; i < count; i++) {
69. fw.write(“测试java 文件操作\r\n”);
71. }
73. fw.close();
75. **long** end3 = System.currentTimeMillis();
77. System.out.println(“FileWriter执行耗时:” + (end3 - begin3) + ” 豪秒”);
79. } **catch** (Exception e) {
81. e.printStackTrace();
83. }
85. **finally** {
87. **try** {
89. fw.close();
91. Buff.close();
93. outSTr.close();
95. out.close();
97. } **catch** (Exception e) {
99. e.printStackTrace();
101. }
103. }
105. }
107. }

# [java中的getParentFile](http://blog.csdn.net/cys1991/article/details/7593790)

String name = "AAAA.txt";  
String lujing = "1"+"/"+"2";//定义路径  
File a = new File(lujing,name);

a.getParentFile().mkdirs();    //这里如果不加getParentFile(),创建的文件夹为"1/2/AAAA.txt/"

那么，a的意义就是“1/2/AAAA.txt”。  
  
这里a是File，但是File这个类在Java里表示的不只是文件，虽然File在英语里是文件的意思。Java里，File至少可以表示文件或文件夹（大概还有可以表示系统设备什么的，这里不考虑，只考虑文件和文件夹）。  
  
也就是说，在“1/2/AAAA.txt”真正出现在磁盘结构里之前，它既可以表示这个文件，也可以表示这个路径的文件夹。那么，如果没有getParentFile(),直接执行a.mkdirs()，就是说，创建“1/2/AAAA.txt”代表的文件夹，也就是“1/2/AAAA.txt/”，在此之后，执行a.createNewFile()，试图创建a文件，然而以a为名的文件夹已经存在了，所以createNewFile()实际是执行失败的。你可以用System.out.println(a.createNewFile())这样来检查是不是真正创建文件成功。  
  
所以，这里，你想要创建的是“1/2/AAAA.txt”这个文件。在创建AAAA.txt之前，必须要1/2这个目录存在。所以，要得到1/2，就要用a.getParentFile()，然后要创建它，也就是a.getParentFile().mkdirs()。在这之后，a作为文件所需要的文件夹大概会存在了（有特殊情况会无法创建的，这里不考虑），就执行a.createNewFile()创建a文件。

# Java RandomAccessFile的使用

Java的RandomAccessFile提供对文件的读写功能，与普通的输入输出流不一样的是RamdomAccessFile可以任意的访问文件的任何地方。这就是“Random”的意义所在。

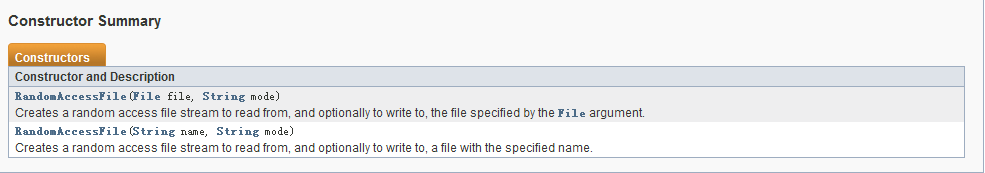
RandomAccessFile的对象包含一个记录指针，用于标识当前流的读写位置，这个位置可以向前移动，也可以向后移动。RandomAccessFile包含两个方法来操作文件记录指针。

long getFilePoint():记录文件指针的当前位置。

void seek(long pos):将文件记录指针定位到pos位置。

RandomAccessFile包含InputStream的三个read方法，也包含OutputStream的三个write方法。同时RandomAccessFile还包含一系列的readXxx和writeXxx方法完成输入输出。

RandomAccessFile的构造方法如下



mode的值有四个

"r":以只读文方式打开指定文件。如果你写的话会有IOException。

"rw":以读写方式打开指定文件，不存在就创建新文件。

"rws":不介绍了。

"rwd":也不介绍。

**[java]** [view plain](http://blog.csdn.net/jiangxinyu/article/details/7885518/) [copy](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. /\*\*
2. \* 往文件中依次写入3名员工的信息，
3. \* 每位员工有姓名和员工两个字段 然后按照
4. \* 第二名，第一名，第三名的先后顺序读取员工信息
5. \*/
6. **import** java.io.File;
7. **import** java.io.RandomAccessFile;
9. **public** **class** RandomAccessFileTest {
10. **public** **static** **void** main(String[] args) **throws** Exception {
11. Employee e1 = **new** Employee(23, "张三");
12. Employee e2 = **new** Employee(24, "lisi");
13. Employee e3 = **new** Employee(25, "王五");
14. File file = **new** File("employee.txt");
15. **if** (!file.exists()) {
16. file.createNewFile();
17. }
18. // 一个中文占两个字节 一个英文字母占一个字节
19. // 整形 占的字节数目 跟cpu位长有关 32位的占4个字节
20. RandomAccessFile randomAccessFile = **new** RandomAccessFile(file, "rw");
21. randomAccessFile.writeChars(e1.getName());
22. randomAccessFile.writeInt(e1.getAge());
23. randomAccessFile.writeChars(e2.getName());
24. randomAccessFile.writeInt(e2.getAge());
25. randomAccessFile.writeChars(e3.getName());
26. randomAccessFile.writeInt(e3.getAge());
27. randomAccessFile.close();
29. RandomAccessFile raf2 = **new** RandomAccessFile(file, "r");
30. raf2.skipBytes(Employee.LEN \* 2 + 4);
31. String strName2 = "";
32. **for** (**int** i = 0; i < Employee.LEN; i++) {
33. strName2 = strName2 + raf2.readChar();
34. }
35. **int** age2 = raf2.readInt();
36. System.out.println("strName2 = " + strName2.trim());
37. System.out.println("age2 = " + age2);
39. raf2.seek(0);
40. String strName1 = "";
41. **for** (**int** i = 0; i < Employee.LEN; i++) {
42. strName1 = strName1 + raf2.readChar();
43. }
44. **int** age1 = raf2.readInt();
45. System.out.println("strName1 = " + strName1.trim());
46. System.out.println("age1 = " + age1);
48. raf2.skipBytes(Employee.LEN \* 2 + 4);
49. String strName3 = "";
50. **for** (**int** i = 0; i < Employee.LEN; i++) {
51. strName3 = strName3 + raf2.readChar();
52. }
53. **int** age3 = raf2.readInt();
54. System.out.println("strName3 = " + strName3.trim());
55. System.out.println("age3 = " + age3);
56. }
57. }
59. **class** Employee {
60. // 年龄
61. **public** **int** age;
62. // 姓名
63. **public** String name;
64. // 姓名的长度
65. **public** **static** **final** **int** LEN = 8;
67. **public** Employee(**int** age, String name) {
68. **this**.age = age;
70. // 对name字符长度的一个处理
71. **if** (name.length() > LEN) {
72. name = name.substring(0, LEN);
73. } **else** {
74. **while** (name.length() < LEN) {
75. name = name + "/u0000";
76. }
77. }
78. **this**.name = name;
79. }
81. **public** **int** getAge() {
82. **return** age;
83. }
85. **public** String getName() {
86. **return** name;
87. }
89. }

# [高效的RandomAccessFile](http://zhang-xiujiao.iteye.com/blog/1150751)

<http://zhang-xiujiao.iteye.com/blog/1150751>

**主体：**

RandomAccessFile类。其I/O性能较之其它常用开发语言的同类性能差距甚远，严重影响程序的运行效率。

开发人员迫切需要提高效率，下面分析RandomAccessFile等文件类的源代码，找出其中的症结所在，并加以改进优化，创建一个"性/价比"俱佳的随机文件访问类BufferedRandomAccessFile。

在改进之前先做一个基本测试：逐字节COPY一个12兆的文件（这里牵涉到读和写）。

|  |  |  |
| --- | --- | --- |
| 读 | 写 | 耗用时间（秒） |
| RandomAccessFile | RandomAccessFile | 95.848 |
| BufferedInputStream + DataInputStream | BufferedOutputStream + DataOutputStream | 2.935 |

我们可以看到两者差距约32倍，RandomAccessFile也太慢了。先看看两者关键部分的源代码，对比分析，找出原因。

**1．1．[RandomAccessFile]**

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **class** RandomAccessFile **implements** DataOutput, DataInput {
2. **public** **final** **byte** readByte() **throws** IOException {
3. **int** ch = **this**.read();
4. **if** (ch < 0)
5. **throw** **new** EOFException();
6. **return** (**byte**)(ch);
7. }
8. **public** **native** **int** read() **throws** IOException;
9. **public** **final** **void** writeByte(**int** v) **throws** IOException {
10. write(v);
11. }
12. **public** **native** **void** write(**int** b) **throws** IOException;
13. }

可见，RandomAccessFile每读/写一个字节就需对磁盘进行一次I/O操作。

**1．2．[BufferedInputStream]**

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **class** BufferedInputStream **extends** FilterInputStream {
2. **private** **static** **int** defaultBufferSize = 2048;
3. **protected** **byte** buf[]; // 建立读缓存区
4. **public** BufferedInputStream(InputStream in, **int** size) {
5. **super**(in);
6. **if** (size <= 0) {
7. **throw** **new** IllegalArgumentException("Buffer size <= 0");
8. }
9. buf = **new** **byte**[size];
10. }
11. **public** **synchronized** **int** read() **throws** IOException {
12. ensureOpen();
13. **if** (pos >= count) {
14. fill();
15. **if** (pos >= count)
16. **return** -1;
17. }
18. **return** buf[pos++] & 0xff; // 直接从BUF[]中读取
19. }
20. **private** **void** fill() **throws** IOException {
21. **if** (markpos < 0)
22. pos = 0;        /\* no mark: throw away the buffer \*/
23. **else** **if** (pos >= buf.length)  /\* no room left in buffer \*/
24. **if** (markpos > 0) {   /\* can throw away early part of the buffer \*/
25. **int** sz = pos - markpos;
26. System.arraycopy(buf, markpos, buf, 0, sz);
27. pos = sz;
28. markpos = 0;
29. } **else** **if** (buf.length >= marklimit) {
30. markpos = -1;   /\* buffer got too big, invalidate mark \*/
31. pos = 0;    /\* drop buffer contents \*/
32. } **else** {        /\* grow buffer \*/
33. **int** nsz = pos \* 2;
34. **if** (nsz > marklimit)
35. nsz = marklimit;
36. **byte** nbuf[] = **new** **byte**[nsz];
37. System.arraycopy(buf, 0, nbuf, 0, pos);
38. buf = nbuf;
39. }
40. count = pos;
41. **int** n = in.read(buf, pos, buf.length - pos);
42. **if** (n > 0)
43. count = n + pos;
44. }
45. }

**1．3．[BufferedOutputStream]**

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **class** BufferedOutputStream **extends** FilterOutputStream {
2. **protected** **byte** buf[]; // 建立写缓存区
3. **public** BufferedOutputStream(OutputStream out, **int** size) {
4. **super**(out);
5. **if** (size <= 0) {
6. **throw** **new** IllegalArgumentException("Buffer size <= 0");
7. }
8. buf = **new** **byte**[size];
9. }
10. **public** **synchronized** **void** write(**int** b) **throws** IOException {
11. **if** (count >= buf.length) {
12. flushBuffer();
13. }
14. buf[count++] = (**byte**)b; // 直接从BUF[]中读取
15. }
16. **private** **void** flushBuffer() **throws** IOException {
17. **if** (count > 0) {
18. out.write(buf, 0, count);
19. count = 0;
20. }
21. }
22. }

可见，Buffered I/O putStream每读/写一个字节，若要操作的数据在BUF中，就直接对内存的buf[]进行读/写操作；否则从磁盘相应位置填充buf[]，再直接对内存的buf[]进行读/写操作，绝大部分的读/写操作是对内存buf[]的操作。

**1．3．小结**

内存存取时间单位是纳秒级（10E-9），磁盘存取时间单位是毫秒级（10E-3），同样操作一次的开销，内存比磁盘快了百万倍。理论上可以预见，即使对内存操作上万次，花费的时间也远少对于磁盘一次I/O的开销。显然后者是通过增加位于内存的BUF存取，减少磁盘I/O的开销，提高存取效率的，当然这样也增加了BUF控制部分的开销。从实际应用来看，存取效率提高了32倍。

根据1.3得出的结论，现试着对RandomAccessFile类也加上缓冲读写机制。

随机访问类与顺序类不同，前者是通过实现DataInput/DataOutput接口创建的，而后者是扩展FilterInputStream/FilterOutputStream创建的，不能直接照搬。

**2．1．开辟缓冲区BUF[默认：1024字节]，用作读/写的共用缓冲区。**

**2．2．先实现读缓冲。**

读缓冲逻辑的基本原理：

* A 欲读文件POS位置的一个字节。
* B 查BUF中是否存在？若有，直接从BUF中读取，并返回该字符BYTE。
* C 若没有，则BUF重新定位到该POS所在的位置并把该位置附近的BUFSIZE的字节的文件内容填充BUFFER，返回B。

以下给出关键部分代码及其说明：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **class** BufferedRandomAccessFile **extends** RandomAccessFile {
2. //  byte read(long pos)：读取当前文件POS位置所在的字节
3. //  bufstartpos、bufendpos代表BUF映射在当前文件的首/尾偏移地址。
4. //  curpos指当前类文件指针的偏移地址。
5. **public** **byte** read(**long** pos) **throws** IOException {
6. **if** (pos < **this**.bufstartpos || pos > **this**.bufendpos ) {
7. **this**.flushbuf();
8. **this**.seek(pos);
9. **if** ((pos < **this**.bufstartpos) || (pos > **this**.bufendpos))
10. **throw** **new** IOException();
11. }
12. **this**.curpos = pos;
13. **return** **this**.buf[(**int**)(pos - **this**.bufstartpos)];
14. }
15. // void flushbuf()：bufdirty为真，把buf[]中尚未写入磁盘的数据，写入磁盘。
16. **private** **void** flushbuf() **throws** IOException {
17. **if** (**this**.bufdirty == **true**) {
18. **if** (**super**.getFilePointer() != **this**.bufstartpos) {
19. **super**.seek(**this**.bufstartpos);
20. }
21. **super**.write(**this**.buf, 0, **this**.bufusedsize);
22. **this**.bufdirty = **false**;
23. }
24. }
25. // void seek(long pos)：移动文件指针到pos位置，并把buf[]映射填充至POS所在的文件块。
26. **public** **void** seek(**long** pos) **throws** IOException {
27. **if** ((pos < **this**.bufstartpos) || (pos > **this**.bufendpos)) { // seek pos not in buf
28. **this**.flushbuf();
29. **if** ((pos >= 0) && (pos <= **this**.fileendpos) && (**this**.fileendpos != 0)) {   // seek pos in file (file length > 0)
30. **this**.bufstartpos =  pos \* bufbitlen / bufbitlen;
31. **this**.bufusedsize = **this**.fillbuf();
32. } **else** **if** (((pos == 0) && (**this**.fileendpos == 0)) || (pos == **this**.fileendpos + 1)) {   // seek pos is append pos
33. **this**.bufstartpos = pos;
34. **this**.bufusedsize = 0;
35. }
36. **this**.bufendpos = **this**.bufstartpos + **this**.bufsize - 1;
37. }
38. **this**.curpos = pos;
39. }
40. // int fillbuf()：根据bufstartpos，填充buf[]。
41. **private** **int** fillbuf() **throws** IOException {
42. **super**.seek(**this**.bufstartpos);
43. **this**.bufdirty = **false**;
44. **return** **super**.read(**this**.buf);
45. }
46. }

至此缓冲读基本实现，逐字节COPY一个12兆的文件（这里牵涉到读和写，用BufferedRandomAccessFile试一下读的速度）：

|  |  |  |
| --- | --- | --- |
| 读 | 写 | 耗用时间（秒） |
| RandomAccessFile | RandomAccessFile | 95.848 |
| BufferedRandomAccessFile | BufferedOutputStream + DataOutputStream | 2.813 |
| BufferedInputStream + DataInputStream | BufferedOutputStream + DataOutputStream | 2.935 |

可见速度显著提高，与BufferedInputStream+DataInputStream不相上下。

**2．3．实现写缓冲。**

写缓冲逻辑的基本原理：

* A欲写文件POS位置的一个字节。
* B 查BUF中是否有该映射？若有，直接向BUF中写入，并返回true。
* C若没有，则BUF重新定位到该POS所在的位置，并把该位置附近的 BUFSIZE字节的文件内容填充BUFFER，返回B。

下面给出关键部分代码及其说明：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. // boolean write(byte bw, long pos)：向当前文件POS位置写入字节BW。
2. // 根据POS的不同及BUF的位置：存在修改、追加、BUF中、BUF外等情况。在逻辑判断时，把最可能出现的情况，最先判断，这样可提高速度。
3. // fileendpos：指示当前文件的尾偏移地址，主要考虑到追加因素
4. **public** **boolean** write(**byte** bw, **long** pos) **throws** IOException {
5. **if** ((pos >= **this**.bufstartpos) && (pos <= **this**.bufendpos)) { // write pos in buf
6. **this**.buf[(**int**)(pos - **this**.bufstartpos)] = bw;
7. **this**.bufdirty = **true**;
8. **if** (pos == **this**.fileendpos + 1) { // write pos is append pos
9. **this**.fileendpos++;
10. **this**.bufusedsize++;
11. }
12. } **else** { // write pos not in buf
13. **this**.seek(pos);
14. **if** ((pos >= 0) && (pos <= **this**.fileendpos) && (**this**.fileendpos != 0)) { // write pos is modify file
15. **this**.buf[(**int**)(pos - **this**.bufstartpos)] = bw;
16. } **else** **if** (((pos == 0) && (**this**.fileendpos == 0)) || (pos == **this**.fileendpos + 1)) { // write pos is append pos
17. **this**.buf[0] = bw;
18. **this**.fileendpos++;
19. **this**.bufusedsize = 1;
20. } **else** {
21. **throw** **new** IndexOutOfBoundsException();
22. }
23. **this**.bufdirty = **true**;
24. }
25. **this**.curpos = pos;
26. **return** **true**;
27. }

至此缓冲写基本实现，逐字节COPY一个12兆的文件，（这里牵涉到读和写，结合缓冲读，用BufferedRandomAccessFile试一下读/写的速度）：

|  |  |  |
| --- | --- | --- |
| 读 | 写 | 耗用时间（秒） |
| RandomAccessFile | RandomAccessFile | 95.848 |
| BufferedInputStream + DataInputStream | BufferedOutputStream + DataOutputStream | 2.935 |
| BufferedRandomAccessFile | BufferedOutputStream + DataOutputStream | 2.813 |
| BufferedRandomAccessFile | BufferedRandomAccessFile | 2.453 |

可见综合读/写速度已超越BufferedInput/OutputStream+DataInput/OutputStream。

# [高效的RandomAccessFile【续】](http://zhang-xiujiao.iteye.com/blog/1150762)

<http://zhang-xiujiao.iteye.com/blog/1150762>

**优化BufferedRandomAccessFile。**

优化原则：

* 调用频繁的语句最需要优化，且优化的效果最明显。
* 多重嵌套逻辑判断时，最可能出现的判断，应放在最外层。
* 减少不必要的NEW。

这里举一典型的例子：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **void** seek(**long** pos) **throws** IOException {
2. ...
3. **this**.bufstartpos =  pos \* bufbitlen / bufbitlen; // bufbitlen指buf[]的位长，例：若bufsize=1024，则bufbitlen=10。
4. ...
5. }

seek函数使用在各函数中，调用非常频繁，上面加重的这行语句根据pos和bufsize确定buf[]对应当前文件的映射位置，用"\*"、"/"确定，显然不是一个好方法。

* 优化一：this.bufstartpos = (pos << bufbitlen) >> bufbitlen;
* 优化二：this.bufstartpos = pos & bufmask; // this.bufmask = ~((long)this.bufsize - 1);

两者效率都比原来好，但后者显然更好，因为前者需要两次移位运算、后者只需一次逻辑与运算（bufmask可以预先得出）。

至此优化基本实现，逐字节COPY一个12兆的文件，（这里牵涉到读和写，结合缓冲读，用优化后BufferedRandomAccessFile试一下读/写的速度）：

|  |  |  |
| --- | --- | --- |
| 读 | 写 | 耗用时间（秒） |
| RandomAccessFile | RandomAccessFile | 95.848 |
| BufferedInputStream + DataInputStream | BufferedOutputStream + DataOutputStream | 2.935 |
| BufferedRandomAccessFile | BufferedOutputStream + DataOutputStream | 2.813 |
| BufferedRandomAccessFile | BufferedRandomAccessFile | 2.453 |
| BufferedRandomAccessFile优 | BufferedRandomAccessFile优 | 2.197 |

可见优化尽管不明显，还是比未优化前快了一些，也许这种效果在老式机上会更明显。

以上比较的是顺序存取，即使是随机存取，在绝大多数情况下也不止一个BYTE，所以缓冲机制依然有效。而一般的顺序存取类要实现随机存取就不怎么容易了。

**需要完善的地方**

提供文件追加功能：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **boolean** append(**byte** bw) **throws** IOException {
2. **return** **this**.write(bw, **this**.fileendpos + 1);
3. }

提供文件当前位置修改功能：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **boolean** write(**byte** bw) **throws** IOException {
2. **return** **this**.write(bw, **this**.curpos);
3. }

返回文件长度（由于BUF读写的原因，与原来的RandomAccessFile类有所不同）：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **long** length() **throws** IOException {
2. **return** **this**.max(**this**.fileendpos + 1, **this**.initfilelen);
3. }

返回文件当前指针（由于是通过BUF读写的原因，与原来的RandomAccessFile类有所不同）：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **long** getFilePointer() **throws** IOException {
2. **return** **this**.curpos;
3. }

提供对当前位置的多个字节的缓冲写功能：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **void** write(**byte** b[], **int** off, **int** len) **throws** IOException {
2. **long** writeendpos = **this**.curpos + len - 1;
3. **if** (writeendpos <= **this**.bufendpos) { // b[] in cur buf
4. System.arraycopy(b, off, **this**.buf, (**int**)(**this**.curpos - **this**.bufstartpos), len);
5. **this**.bufdirty = **true**;
6. **this**.bufusedsize = (**int**)(writeendpos - **this**.bufstartpos + 1);
7. } **else** { // b[] not in cur buf
8. **super**.seek(**this**.curpos);
9. **super**.write(b, off, len);
10. }
11. **if** (writeendpos > **this**.fileendpos)
12. **this**.fileendpos = writeendpos;
13. **this**.seek(writeendpos+1);
14. }
15. **public** **void** write(**byte** b[]) **throws** IOException {
16. **this**.write(b, 0, b.length);
17. }

提供对当前位置的多个字节的缓冲读功能：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. **public** **int** read(**byte** b[], **int** off, **int** len) **throws** IOException {
2. **long** readendpos = **this**.curpos + len - 1;
3. **if** (readendpos <= **this**.bufendpos && readendpos <= **this**.fileendpos ) { // read in buf
4. System.arraycopy(**this**.buf, (**int**)(**this**.curpos - **this**.bufstartpos), b, off, len);
5. } **else** { // read b[] size > buf[]
6. **if** (readendpos > **this**.fileendpos) { // read b[] part in file
7. len = (**int**)(**this**.length() - **this**.curpos + 1);
8. }
9. **super**.seek(**this**.curpos);
10. len = **super**.read(b, off, len);
11. readendpos = **this**.curpos + len - 1;
12. }
13. **this**.seek(readendpos + 1);
14. **return** len;
15. }
16. **public** **int** read(**byte** b[]) **throws** IOException {
17. **return** **this**.read(b, 0, b.length);
18. }
19. **public** **void** setLength(**long** newLength) **throws** IOException {
20. **if** (newLength > 0) {
21. **this**.fileendpos = newLength - 1;
22. } **else** {
23. **this**.fileendpos = 0;
24. }
25. **super**.setLength(newLength);
26. }
28. **public** **void** close() **throws** IOException {
29. **this**.flushbuf();
30. **super**.close();
31. }

至此完善工作基本完成，试一下新增的多字节读/写功能，通过同时读/写1024个字节，来COPY一个12兆的文件，（这里牵涉到读和写，用完善后BufferedRandomAccessFile试一下读/写的速度）：

|  |  |  |
| --- | --- | --- |
| 读 | 写 | 耗用时间（秒） |
| RandomAccessFile | RandomAccessFile | 95.848 |
| BufferedInputStream + DataInputStream | BufferedOutputStream + DataOutputStream | 2.935 |
| BufferedRandomAccessFile | BufferedOutputStream + DataOutputStream | 2.813 |
| BufferedRandomAccessFile | BufferedRandomAccessFile | 2.453 |
| BufferedRandomAccessFile优 | BufferedRandomAccessFile优 | 2.197 |
| BufferedRandomAccessFile完 | BufferedRandomAccessFile完 | 0.401 |

**与MappedByteBuffer+RandomAccessFile的对比？**

JDK1.4+提供了NIO类 ，其中MappedByteBuffer类用于映射缓冲，也可以映射随机文件访问，可见JAVA设计者也看到了RandomAccessFile的问题，并加以改进。怎么通过MappedByteBuffer+RandomAccessFile拷贝文件呢？下面就是测试程序的主要部分：

Java代码  [收藏代码](http://blog.csdn.net/jiangxinyu/article/details/7885518/)

1. RandomAccessFile rafi = **new** RandomAccessFile(SrcFile, "r");
2. RandomAccessFile rafo = **new** RandomAccessFile(DesFile, "rw");
3. FileChannel fci = rafi.getChannel();
4. FileChannel fco = rafo.getChannel();
5. **long** size = fci.size();
6. MappedByteBuffer mbbi = fci.map(FileChannel.MapMode.READ\_ONLY, 0, size);
7. MappedByteBuffer mbbo = fco.map(FileChannel.MapMode.READ\_WRITE, 0, size);
8. **long** start = System.currentTimeMillis();
9. **for** (**int** i = 0; i < size; i++) {
10. **byte** b = mbbi.get(i);
11. mbbo.put(i, b);
12. }
13. fcin.close();
14. fcout.close();
15. rafi.close();
16. rafo.close();
17. System.out.println("Spend: "+(**double**)(System.currentTimeMillis()-start) / 1000 + "s");

试一下JDK1.4的映射缓冲读/写功能，逐字节COPY一个12兆的文件，（这里牵涉到读和写）：

|  |  |  |
| --- | --- | --- |
| 读 | 写 | 耗用时间（秒） |
| RandomAccessFile | RandomAccessFile | 95.848 |
| BufferedInputStream + DataInputStream | BufferedOutputStream + DataOutputStream | 2.935 |
| BufferedRandomAccessFile | BufferedOutputStream + DataOutputStream | 2.813 |
| BufferedRandomAccessFile | BufferedRandomAccessFile | 2.453 |
| BufferedRandomAccessFile优 | BufferedRandomAccessFile优 | 2.197 |
| BufferedRandomAccessFile完 | BufferedRandomAccessFile完 | 0.401 |
| MappedByteBuffer+ RandomAccessFile | MappedByteBuffer+ RandomAccessFile | 1.209 |

确实不错，看来NIO有了极大的进步。建议采用 MappedByteBuffer+RandomAccessFile的方式。