<https://www.cnblogs.com/skywang12345/p/io_23.html#top>

[**java io系列23之 BufferedReader(字符缓冲输入流)**](http://www.cnblogs.com/skywang12345/p/io_23.html)

转载请注明出处：[**http://www.cnblogs.com/skywang12345/p/io\_23.html**](http://www.cnblogs.com/skywang12345/p/io_23.html)

更多内容请参考：[**java io系列01之 "目录"**](http://www.cnblogs.com/skywang12345/p/io_01.html)

### BufferedReader 介绍

BufferedReader 是缓冲字符输入流。它继承于Reader。  
BufferedReader 的作用是为其他字符输入流添加一些缓冲功能。

**BufferedReader 函数列表**

[复制代码](javascript:void(0);)

BufferedReader(Reader in)

BufferedReader(Reader in, int size)

void close()

void mark(int markLimit)

boolean markSupported()

int read()

int read(char[] buffer, int offset, int length)

String readLine()

boolean ready()

void reset()

long skip(long charCount)

[复制代码](javascript:void(0);)

### BufferedReader 源码分析(基于jdk1.7.40)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 package java.io;

2

3 public class BufferedReader extends Reader {

4

5 private Reader in;

6

7 // 字符缓冲区

8 private char cb[];

9 // nChars 是cb缓冲区中字符的总的个数

10 // nextChar 是下一个要读取的字符在cb缓冲区中的位置

11 private int nChars, nextChar;

12

13 // 表示“标记无效”。它与UNMARKED的区别是：

14 // (01) UNMARKED 是压根就没有设置过标记。

15 // (02) 而INVALIDATED是设置了标记，但是被标记位置太长，导致标记无效！

16 private static final int INVALIDATED = -2;

17 // 表示没有设置“标记”

18 private static final int UNMARKED = -1;

19 // “标记”

20 private int markedChar = UNMARKED;

21 // “标记”能标记位置的最大长度

22 private int readAheadLimit = 0; /\* Valid only when markedChar > 0 \*/

23

24 // skipLF(即skip Line Feed)是“是否忽略换行符”标记

25 private boolean skipLF = false;

26

27 // 设置“标记”时，保存的skipLF的值

28 private boolean markedSkipLF = false;

29

30 // 默认字符缓冲区大小

31 private static int defaultCharBufferSize = 8192;

32 // 默认每一行的字符个数

33 private static int defaultExpectedLineLength = 80;

34

35 // 创建“Reader”对应的BufferedReader对象，sz是BufferedReader的缓冲区大小

36 public BufferedReader(Reader in, int sz) {

37 super(in);

38 if (sz <= 0)

39 throw new IllegalArgumentException("Buffer size <= 0");

40 this.in = in;

41 cb = new char[sz];

42 nextChar = nChars = 0;

43 }

44

45 // 创建“Reader”对应的BufferedReader对象，默认的BufferedReader缓冲区大小是8k

46 public BufferedReader(Reader in) {

47 this(in, defaultCharBufferSize);

48 }

49

50 // 确保“BufferedReader”是打开状态

51 private void ensureOpen() throws IOException {

52 if (in == null)

53 throw new IOException("Stream closed");

54 }

55

56 // 填充缓冲区函数。有以下两种情况被调用：

57 // (01) 缓冲区没有数据时，通过fill()可以向缓冲区填充数据。

58 // (02) 缓冲区数据被读完，需更新时，通过fill()可以更新缓冲区的数据。

59 private void fill() throws IOException {

60 // dst表示“cb中填充数据的起始位置”。

61 int dst;

62 if (markedChar <= UNMARKED) {

63 // 没有标记的情况，则设dst=0。

64 dst = 0;

65 } else {

66 // delta表示“当前标记的长度”，它等于“下一个被读取字符的位置”减去“标记的位置”的差值；

67 int delta = nextChar - markedChar;

68 if (delta >= readAheadLimit) {

69 // 若“当前标记的长度”超过了“标记上限(readAheadLimit)”，

70 // 则丢弃标记！

71 markedChar = INVALIDATED;

72 readAheadLimit = 0;

73 dst = 0;

74 } else {

75 if (readAheadLimit <= cb.length) {

76 // 若“当前标记的长度”没有超过了“标记上限(readAheadLimit)”，

77 // 并且“标记上限(readAheadLimit)”小于/等于“缓冲的长度”；

78 // 则先将“下一个要被读取的位置，距离我们标记的置符的距离”间的字符保存到cb中。

79 System.arraycopy(cb, markedChar, cb, 0, delta);

80 markedChar = 0;

81 dst = delta;

82 } else {

83 // 若“当前标记的长度”没有超过了“标记上限(readAheadLimit)”，

84 // 并且“标记上限(readAheadLimit)”大于“缓冲的长度”；

85 // 则重新设置缓冲区大小，并将“下一个要被读取的位置，距离我们标记的置符的距离”间的字符保存到cb中。

86 char ncb[] = new char[readAheadLimit];

87 System.arraycopy(cb, markedChar, ncb, 0, delta);

88 cb = ncb;

89 markedChar = 0;

90 dst = delta;

91 }

92 // 更新nextChar和nChars

93 nextChar = nChars = delta;

94 }

95 }

96

97 int n;

98 do {

99 // 从“in”中读取数据，并存储到字符数组cb中；

100 // 从cb的dst位置开始存储，读取的字符个数是cb.length - dst

101 // n是实际读取的字符个数；若n==0(即一个也没读到)，则继续读取！

102 n = in.read(cb, dst, cb.length - dst);

103 } while (n == 0);

104

105 // 如果从“in”中读到了数据，则设置nChars(cb中字符的数目)=dst+n，

106 // 并且nextChar(下一个被读取的字符的位置)=dst。

107 if (n > 0) {

108 nChars = dst + n;

109 nextChar = dst;

110 }

111 }

112

113 // 从BufferedReader中读取一个字符，该字符以int的方式返回

114 public int read() throws IOException {

115 synchronized (lock) {

116 ensureOpen();

117 for (;;) {

118 // 若“缓冲区的数据已经被读完”，

119 // 则先通过fill()更新缓冲区数据

120 if (nextChar >= nChars) {

121 fill();

122 if (nextChar >= nChars)

123 return -1;

124 }

125 // 若要“忽略换行符”，

126 // 则对下一个字符是否是换行符进行处理。

127 if (skipLF) {

128 skipLF = false;

129 if (cb[nextChar] == '\n') {

130 nextChar++;

131 continue;

132 }

133 }

134 // 返回下一个字符

135 return cb[nextChar++];

136 }

137 }

138 }

139

140 // 将缓冲区中的数据写入到数组cbuf中。off是数组cbuf中的写入起始位置，len是写入长度

141 private int read1(char[] cbuf, int off, int len) throws IOException {

142 // 若“缓冲区的数据已经被读完”，则更新缓冲区数据。

143 if (nextChar >= nChars) {

144 if (len >= cb.length && markedChar <= UNMARKED && !skipLF) {

145 return in.read(cbuf, off, len);

146 }

147 fill();

148 }

149 // 若更新数据之后，没有任何变化；则退出。

150 if (nextChar >= nChars) return -1;

151 // 若要“忽略换行符”，则进行相应处理

152 if (skipLF) {

153 skipLF = false;

154 if (cb[nextChar] == '\n') {

155 nextChar++;

156 if (nextChar >= nChars)

157 fill();

158 if (nextChar >= nChars)

159 return -1;

160 }

161 }

162 // 拷贝字符操作

163 int n = Math.min(len, nChars - nextChar);

164 System.arraycopy(cb, nextChar, cbuf, off, n);

165 nextChar += n;

166 return n;

167 }

168

169 // 对read1()的封装，添加了“同步处理”和“阻塞式读取”等功能

170 public int read(char cbuf[], int off, int len) throws IOException {

171 synchronized (lock) {

172 ensureOpen();

173 if ((off < 0) || (off > cbuf.length) || (len < 0) ||

174 ((off + len) > cbuf.length) || ((off + len) < 0)) {

175 throw new IndexOutOfBoundsException();

176 } else if (len == 0) {

177 return 0;

178 }

179

180 int n = read1(cbuf, off, len);

181 if (n <= 0) return n;

182 while ((n < len) && in.ready()) {

183 int n1 = read1(cbuf, off + n, len - n);

184 if (n1 <= 0) break;

185 n += n1;

186 }

187 return n;

188 }

189 }

190

191 // 读取一行数据。ignoreLF是“是否忽略换行符”

192 String readLine(boolean ignoreLF) throws IOException {

193 StringBuffer s = null;

194 int startChar;

195

196 synchronized (lock) {

197 ensureOpen();

198 boolean omitLF = ignoreLF || skipLF;

199

200 bufferLoop:

201 for (;;) {

202

203 if (nextChar >= nChars)

204 fill();

205 if (nextChar >= nChars) { /\* EOF \*/

206 if (s != null && s.length() > 0)

207 return s.toString();

208 else

209 return null;

210 }

211 boolean eol = false;

212 char c = 0;

213 int i;

214

215 /\* Skip a leftover '\n', if necessary \*/

216 if (omitLF && (cb[nextChar] == '\n'))

217 nextChar++;

218 skipLF = false;

219 omitLF = false;

220

221 charLoop:

222 for (i = nextChar; i < nChars; i++) {

223 c = cb[i];

224 if ((c == '\n') || (c == '\r')) {

225 eol = true;

226 break charLoop;

227 }

228 }

229

230 startChar = nextChar;

231 nextChar = i;

232

233 if (eol) {

234 String str;

235 if (s == null) {

236 str = new String(cb, startChar, i - startChar);

237 } else {

238 s.append(cb, startChar, i - startChar);

239 str = s.toString();

240 }

241 nextChar++;

242 if (c == '\r') {

243 skipLF = true;

244 }

245 return str;

246 }

247

248 if (s == null)

249 s = new StringBuffer(defaultExpectedLineLength);

250 s.append(cb, startChar, i - startChar);

251 }

252 }

253 }

254

255 // 读取一行数据。不忽略换行符

256 public String readLine() throws IOException {

257 return readLine(false);

258 }

259

260 // 跳过n个字符

261 public long skip(long n) throws IOException {

262 if (n < 0L) {

263 throw new IllegalArgumentException("skip value is negative");

264 }

265 synchronized (lock) {

266 ensureOpen();

267 long r = n;

268 while (r > 0) {

269 if (nextChar >= nChars)

270 fill();

271 if (nextChar >= nChars) /\* EOF \*/

272 break;

273 if (skipLF) {

274 skipLF = false;

275 if (cb[nextChar] == '\n') {

276 nextChar++;

277 }

278 }

279 long d = nChars - nextChar;

280 if (r <= d) {

281 nextChar += r;

282 r = 0;

283 break;

284 }

285 else {

286 r -= d;

287 nextChar = nChars;

288 }

289 }

290 return n - r;

291 }

292 }

293

294 // “下一个字符”是否可读

295 public boolean ready() throws IOException {

296 synchronized (lock) {

297 ensureOpen();

298

299 // 若忽略换行符为true；

300 // 则判断下一个符号是否是换行符，若是的话，则忽略

301 if (skipLF) {

302 if (nextChar >= nChars && in.ready()) {

303 fill();

304 }

305 if (nextChar < nChars) {

306 if (cb[nextChar] == '\n')

307 nextChar++;

308 skipLF = false;

309 }

310 }

311 return (nextChar < nChars) || in.ready();

312 }

313 }

314

315 // 始终返回true。因为BufferedReader支持mark(), reset()

316 public boolean markSupported() {

317 return true;

318 }

319

320 // 标记当前BufferedReader的下一个要读取位置。关于readAheadLimit的作用，参考后面的说明。

321 public void mark(int readAheadLimit) throws IOException {

322 if (readAheadLimit < 0) {

323 throw new IllegalArgumentException("Read-ahead limit < 0");

324 }

325 synchronized (lock) {

326 ensureOpen();

327 // 设置readAheadLimit

328 this.readAheadLimit = readAheadLimit;

329 // 保存下一个要读取的位置

330 markedChar = nextChar;

331 // 保存“是否忽略换行符”标记

332 markedSkipLF = skipLF;

333 }

334 }

335

336 // 重置BufferedReader的下一个要读取位置，

337 // 将其还原到mark()中所保存的位置。

338 public void reset() throws IOException {

339 synchronized (lock) {

340 ensureOpen();

341 if (markedChar < 0)

342 throw new IOException((markedChar == INVALIDATED)

343 ? "Mark invalid"

344 : "Stream not marked");

345 nextChar = markedChar;

346 skipLF = markedSkipLF;

347 }

348 }

349

350 public void close() throws IOException {

351 synchronized (lock) {

352 if (in == null)

353 return;

354 in.close();

355 in = null;

356 cb = null;

357 }

358 }

359 }

[复制代码](javascript:void(0);)

**说明**：  
要想读懂BufferReader的源码，就要先理解它的思想。BufferReader的作用是为其它Reader提供缓冲功能。创建BufferReader时，我们会通过它的构造函数指定某个Reader为参数。BufferReader会将该Reader中的数据分批读取，每次读取一部分到缓冲中；操作完缓冲中的这部分数据之后，再从Reader中读取下一部分的数据。  
为什么需要缓冲呢？原因很简单，效率问题！缓冲中的数据实际上是保存在内存中，而原始数据可能是保存在硬盘或NandFlash中；而我们知道，从内存中读取数据的速度比从硬盘读取数据的速度至少快10倍以上。  
那干嘛不干脆一次性将全部数据都读取到缓冲中呢？第一，读取全部的数据所需要的时间可能会很长。第二，内存价格很贵，容量不想硬盘那么大。

下面，我就BufferReader中最重要的函数fill()进行说明。其它的函数很容易理解，我就不详细介绍了，大家可以参考源码中的注释进行理解。我们先看看fill()的源码：

[复制代码](javascript:void(0);)

1 private void fill() throws IOException {

2 int dst;

3 if (markedChar <= UNMARKED) {

4 /\* No mark \*/

5 dst = 0;

6 } else {

7 /\* Marked \*/

8 int delta = nextChar - markedChar;

9 if (delta >= readAheadLimit) {

10 /\* Gone past read-ahead limit: Invalidate mark \*/

11 markedChar = INVALIDATED;

12 readAheadLimit = 0;

13 dst = 0;

14 } else {

15 if (readAheadLimit <= cb.length) {

16 /\* Shuffle in the current buffer \*/

17 System.arraycopy(cb, markedChar, cb, 0, delta);

18 markedChar = 0;

19 dst = delta;

20 } else {

21 /\* Reallocate buffer to accommodate read-ahead limit \*/

22 char ncb[] = new char[readAheadLimit];

23 System.arraycopy(cb, markedChar, ncb, 0, delta);

24 cb = ncb;

25 markedChar = 0;

26 dst = delta;

27 }

28 nextChar = nChars = delta;

29 }

30 }

31

32 int n;

33 do {

34 n = in.read(cb, dst, cb.length - dst);

35 } while (n == 0);

36 if (n > 0) {

37 nChars = dst + n;

38 nextChar = dst;

39 }

40 }

[复制代码](javascript:void(0);)

根据fill()中的if...else...，我将fill()分为**4种情况**进行说明。

**情况1**：读取完缓冲区的数据，并且缓冲区没有被标记  
执行流程如下，  
(01) 其它函数调用 fill()，来更新缓冲区的数据  
(02) fill() 执行代码 if (markedChar <= UNMARKED) { ... }  
为了方便分析，我们将这种情况下fill()执行的操作等价于以下代码：

[复制代码](javascript:void(0);)

1 private void fill() throws IOException {

2 int dst;

3 if (markedChar <= UNMARKED) {

4 /\* No mark \*/

5 dst = 0;

6 }

7

8 int n;

9 do {

10 n = in.read(cb, dst, cb.length - dst);

11 } while (n == 0);

12

13 if (n > 0) {

14 nChars = dst + n;

15 nextChar = dst;

16 }

17 }

[复制代码](javascript:void(0);)

**说明**：  
这种情况发生的情况是 — — Reader中有很长的数据，我们每次从中读取一部分数据到缓冲中进行操作。每次当我们读取完缓冲中的数据之后，并且此时BufferedReader没有被标记；那么，就接着从Reader(BufferReader提供缓冲功能的Reader)中读取下一部分的数据到缓冲中。  
其中，判断是否读完缓冲区中的数据，是通过“比较nextChar和nChars之间大小”来判断的。其中，nChars 是缓冲区中字符的总的个数，而 nextChar 是缓冲区中下一个要读取的字符的位置。  
判断BufferedReader有没有被标记，是通过“markedChar”来判断的。  
理解这个思想之后，我们再对这种情况下的fill()的代码进行分析，就特别容易理解了。  
(01) if (markedChar <= UNMARKED) 它的作用是判断“BufferedReader是否被标记”。若被标记，则dst=0。  
(02) in.read(cb, dst, cb.length - dst) 等价于 in.read(cb, 0, cb.length)，意思是从Reader对象in中读取cb.length个数据，并存储到缓冲区cb中，而且从缓冲区cb的位置0开始存储。该函数返回值等于n，也就是n表示实际读取的字符个数。若n=0(即没有读取到数据)，则继续读取，直到读到数据为止。  
(03) nChars=dst+n 等价于 nChars=n；意味着，更新缓冲区数据cb之后，设置nChars(缓冲区的数据个数)为n。  
(04) nextChar=dst 等价于 nextChar=0；意味着，更新缓冲区数据cb之后，设置nextChar(缓冲区中下一个会被读取的字符的索引值)为0。

**情况2**：读取完缓冲区的数据，缓冲区的标记位置>0，并且“当前标记的长度”超过“标记上限(readAheadLimit)”  
执行流程如下，  
(01) 其它函数调用 fill()，来更新缓冲区的数据  
(02) fill() 执行代码 if (delta >= readAheadLimit) { ... }  
为了方便分析，我们将这种情况下fill()执行的操作等价于以下代码：

[复制代码](javascript:void(0);)

1 private void fill() throws IOException {

2 int dst;

3 if (markedChar > UNMARKED) {

4 int delta = nextChar - markedChar;

5 if (delta >= readAheadLimit) {

6 markedChar = INVALIDATED;

7 readAheadLimit = 0;

8 dst = 0;

9 }

10 }

11

12 int n;

13 do {

14 n = in.read(cb, dst, cb.length - dst);

15 } while (n == 0);

16 if (n > 0) {

17 nChars = dst + n;

18 nextChar = dst;

19 }

20 }

[复制代码](javascript:void(0);)

**说明**：  
这种情况发生的情况是 — — BufferedReader中有很长的数据，我们每次从中读取一部分数据到缓冲区中进行操作。当我们读取完缓冲区中的数据之后，并且此时，BufferedReader存在标记时，同时，“当前标记的长度”大于“标记上限”；那么，就发生情况2。此时，我们会丢弃“标记”并更新缓冲区。  
(01) delta = nextChar - markedChar；其中，delta就是“当前标记的长度”，它是“下一个被读取字符的位置”减去“被标记的位置”的差值。  
(02) if (delta >= readAheadLimit)；其中，当delta >= readAheadLimit，就意味着，“当前标记的长度”>=“标记上限”。为什么要有标记上限，即readAheadLimit的值到底有何意义呢？  
我们标记一个位置之后，更新缓冲区的时候，被标记的位置会被保存；当我们不停的更新缓冲区的时候，被标记的位置会被不停的放大。然后内存的容量是有效的，我们不可能不限制长度的存储标记。所以，需要readAheadLimit来限制标记长度！  
(03) in.read(cb, dst, cb.length - dst) 等价于 in.read(cb, 0, cb.length)，意思是从Reader对象in中读取cb.length个数据，并存储到缓冲区cb中，而且从缓冲区cb的位置0开始存储。该函数返回值等于n，也就是n表示实际读取的字符个数。若n=0(即没有读取到数据)，则继续读取，直到读到数据为止。  
(04) nChars=dst+n 等价于 nChars=n；意味着，更新缓冲区数据cb之后，设置nChars(缓冲区的数据个数)为n。  
(05) nextChar=dst 等价于 nextChar=0；意味着，更新缓冲区数据cb之后，设置nextChar(缓冲区中下一个会被读取的字符的索引值)为0。

**情况3**：读取完缓冲区的数据，缓冲区的标记位置>0，“当前标记的长度”没超过“标记上限(readAheadLimit)”，并且“标记上限(readAheadLimit)”小于/等于“缓冲的长度”；  
执行流程如下，  
(01) 其它函数调用 fill()，来更新缓冲区的数据  
(02) fill() 执行代码 if (readAheadLimit <= cb.length) { ... }  
为了方便分析，我们将这种情况下fill()执行的操作等价于以下代码：

[复制代码](javascript:void(0);)

1 private void fill() throws IOException {

2 int dst;

3 if (markedChar > UNMARKED) {

4 int delta = nextChar - markedChar;

5 if ((delta < readAheadLimit) && (readAheadLimit <= cb.length) ) {

6 System.arraycopy(cb, markedChar, cb, 0, delta);

7 markedChar = 0;

8 dst = delta;

9

10 nextChar = nChars = delta;

11 }

12 }

13

14 int n;

15 do {

16 n = in.read(cb, dst, cb.length - dst);

17 } while (n == 0);

18 if (n > 0) {

19 nChars = dst + n;

20 nextChar = dst;

21 }

22 }

[复制代码](javascript:void(0);)

**说明**：  
这种情况发生的情况是 — — BufferedReader中有很长的数据，我们每次从中读取一部分数据到缓冲区中进行操作。当我们读取完缓冲区中的数据之后，并且此时，BufferedReader存在标记时，同时，“当前标记的长度”小于“标记上限”，并且“标记上限”小于/等于“缓冲区长度”；那么，就发生情况3。此时，我们保留“被标记的位置”(即，保留被标记位置开始的数据)，并更新缓冲区(将新增的数据，追加到保留的数据之后)。

**情况4**：读取完缓冲区的数据，缓冲区的标记位置>0，“当前标记的长度”没超过“标记上限(readAheadLimit)”，并且“标记上限(readAheadLimit)”大于“缓冲的长度”；  
执行流程如下，  
(01) 其它函数调用 fill()，来更新缓冲区的数据  
(02) fill() 执行代码 else { char ncb[] = new char[readAheadLimit]; ... }  
为了方便分析，我们将这种情况下fill()执行的操作等价于以下代码：

[复制代码](javascript:void(0);)

1 private void fill() throws IOException {

2 int dst;

3 if (markedChar > UNMARKED) {

4 int delta = nextChar - markedChar;

5 if ((delta < readAheadLimit) && (readAheadLimit > cb.length) ) {

6 char ncb[] = new char[readAheadLimit];

7 System.arraycopy(cb, markedChar, ncb, 0, delta);

8 cb = ncb;

9 markedChar = 0;

10 dst = delta;

11

12 nextChar = nChars = delta;

13 }

14 }

15

16 int n;

17 do {

18 n = in.read(cb, dst, cb.length - dst);

19 } while (n == 0);

20 if (n > 0) {

21 nChars = dst + n;

22 nextChar = dst;

23 }

24 }

[复制代码](javascript:void(0);)

**说明**：  
这种情况发生的情况是 — — BufferedReader中有很长的数据，我们每次从中读取一部分数据到缓冲区中进行操作。当我们读取完缓冲区中的数据之后，并且此时，BufferedReader存在标记时，同时，“当前标记的长度”小于“标记上限”，并且“标记上限”大于“缓冲区长度”；那么，就发生情况4。此时，我们要先更新缓冲区的大小，然后再保留“被标记的位置”(即，保留被标记位置开始的数据)，并更新缓冲区数据(将新增的数据，追加到保留的数据之后)。

### 示例代码

关于BufferedReader中API的详细用法，参考示例代码(**BufferedReaderTest.java**)：

[复制代码](javascript:void(0);)

1 import java.io.BufferedReader;

2 import java.io.ByteArrayInputStream;

3 import java.io.File;

4 import java.io.InputStream;

5 import java.io.FileReader;

6 import java.io.IOException;

7 import java.io.FileNotFoundException;

8 import java.lang.SecurityException;

9

10 /\*\*

11 \* BufferedReader 测试程序

12 \*

13 \* @author skywang

14 \*/

15 public class BufferedReaderTest {

16

17 private static final int LEN = 5;

18

19 public static void main(String[] args) {

20 testBufferedReader() ;

21 }

22

23 /\*\*

24 \* BufferedReader的API测试函数

25 \*/

26 private static void testBufferedReader() {

27

28 // 创建BufferedReader字符流，内容是ArrayLetters数组

29 try {

30 File file = new File("bufferedreader.txt");

31 BufferedReader in =

32 new BufferedReader(

33 new FileReader(file));

34

35 // 从字符流中读取5个字符。“abcde”

36 for (int i=0; i<LEN; i++) {

37 // 若能继续读取下一个字符，则读取下一个字符

38 if (in.ready()) {

39 // 读取“字符流的下一个字符”

40 int tmp = in.read();

41 System.out.printf("%d : %c\n", i, tmp);

42 }

43 }

44

45 // 若“该字符流”不支持标记功能，则直接退出

46 if (!in.markSupported()) {

47 System.out.println("make not supported!");

48 return ;

49 }

50

51 // 标记“当前索引位置”，即标记第6个位置的元素--“f”

52 // 1024对应marklimit

53 in.mark(1024);

54

55 // 跳过22个字符。

56 in.skip(22);

57

58 // 读取5个字符

59 char[] buf = new char[LEN];

60 in.read(buf, 0, LEN);

61 System.out.printf("buf=%s\n", String.valueOf(buf));

62 // 读取该行剩余的数据

63 System.out.printf("readLine=%s\n", in.readLine());

64

65 // 重置“输入流的索引”为mark()所标记的位置，即重置到“f”处。

66 in.reset();

67 // 从“重置后的字符流”中读取5个字符到buf中。即读取“fghij”

68 in.read(buf, 0, LEN);

69 System.out.printf("buf=%s\n", String.valueOf(buf));

70

71 in.close();

72 } catch (FileNotFoundException e) {

73 e.printStackTrace();

74 } catch (SecurityException e) {

75 e.printStackTrace();

76 } catch (IOException e) {

77 e.printStackTrace();

78 }

79 }

80 }

[复制代码](javascript:void(0);)

程序中读取的**bufferedreader.txt**的内容如下：

abcdefghijklmnopqrstuvwxyz

0123456789

ABCDEFGHIJKLMNOPQRSTUVWXYZ

**运行结果**：  
0 : a  
1 : b  
2 : c  
3 : d  
4 : e  
buf=01234  
readLine=56789  
buf=fghij