Guava学习之CharSequenceReader

CharSequenceReader类是以CharSequence的形式读取字符。CharSequenceReader类继承自Reader类，除了remaining()、hasRemaining()以及checkOpen()函数之后，其他的函数都是重写Reader类中的函数。

CharSequenceReader类声明没有用public关键字，所以我们暂时还不能调用这个类

CharSequenceReader类有下面三个成员变量

```
private CharSequence seq;  //存放字符序列
private int pos;   //存放上述字符序列的下一次读
private int mark;   //可以记录pos的位置,用于下一次重置到pos位置
```

CharSequenceReader类只有一个构造函数如下：

```
public CharSequenceReader(CharSequence seq) {
  this.seq = checkNotNull(seq);
    }
```

这个函数非常的简单，就是将seq赋值给CharSequenceReader类的seq，使得其他方法可以操作seq。

```
private void checkOpen() throws IOException {
  if (seq == null) {
    throw new IOException("reader closed");
  }
}

private boolean hasRemaining() {
  return remaining() > 0;
}

private int remaining() {
  return seq.length() - pos;
  }
```

上面三个函数是CharSequenceReader类专有的。checkOpen()函数主要是判断seq是否没有被清空，如果被清空了，抛出空指针异常；否则什么事都不做。其他的函数实现几乎都用到了checkOpen。hasRemaining函数主要是判断当前是否还有数据可以读。在CharSequenceReader类中提供了三个用于读seq中数据的函数，原型如下：

```
public synchronized int read(CharBuffer target) throws IOException
public synchronized int read() throws IOException
public synchronized int read(char[] cbuf, int off, int len) throws IOException
```

上述上个函数都是重写自Reader类相关的函数，read(CharBuffer target) 函数实现如下：

```
@Override
public synchronized int read(CharBuffer target) throws IOException {
  checkNotNull(target);
  checkOpen();
  if (!hasRemaining()) {
    return -1;
  }
  int charsToRead = Math.min(target.remaining(), remaining());
  for (int i = 0; i < charsToRead; i++) {
    target.put(seq.charAt(pos++));
  }
  return charsToRead;
}
```

先判断seq中是否没被清空；接着判断seq中还有数据可读与否，如果没有数据可读，将返回-1,；否则得到target和seq的最小可读空间大小，并从seq读取相应的数据于target中。其实target.put函数是将一个字符存放在内部char数组的相应位置上面去。read函数最后返回本次读取字符的个数。read()函数实现如下：

```
@Override
public synchronized int read() throws IOException {
  checkOpen();
  return hasRemaining() ? seq.charAt(pos++) : -1;
}
```

可以看出，这个函数实现相当的简单，先判断seq中是否没被清空；接着判断seq中还有数据可读与否，如果有数据可读，则返回seq中下标为pos的数据，且pos向后移动一个位置；如果没有数据可读，则之间返回-1。read()函数最多只返回一个字符。read(char[] cbuf, int off, int len)函数实现如下：

```
@Override
public synchronized int read(char[] cbuf, int off, int len)
                                throws IOException {
    checkPositionIndexes(off, off + len, cbuf.length);
    checkOpen();
    if (!hasRemaining()) {
      return -1;
    }

    int charsToRead = Math.min(len, remaining());
    for (int i = 0; i < charsToRead; i++) {
      cbuf[off + i] = seq.charAt(pos++);
    }
    return charsToRead;
}
```

大题和read(CharBuffer target)函数实现类似。它主要是将读取的到字符存放在cbuf下标从off开始的位置，并且依次读取charsToRead个，最后返回本次读取到的字符个数。

接下来说说skip(long n)函数，它的实现如下：

```
  @Override
  public synchronized long skip(long n) throws IOException {
    checkArgument(n >= 0, "n (%s) may not be negative", n);
    checkOpen();
    // safe because remaining is an int
    int charsToSkip = (int) Math.min(remaining(), n);
    pos += charsToSkip;
    return charsToSkip;
      }
```

主要是通过移动pos指标，从而达到忽略seq中charsToSkip 个字符。上面我们就说了CharSequen

ceReader类中大多数的函数都是重写Reader类的，阅读Reader类中的skip(long n)函数我们可以看到，它忽略了charsToSkip个字符的同时还保存了本次被忽略的charsToSkip个字符于char skipBuffer[]数组中。

剩下的几个函数为

```
@Override
public synchronized boolean ready() throws IOException {
  checkOpen();
  return true;
}

@Override
public boolean markSupported() {
  return true;
}

@Override
public synchronized void mark(int readAheadLimit) throws IOException {
  checkArgument(readAheadLimit >= 0,
      "readAheadLimit (%s) may not be negative", readAheadLimit);
  checkOpen();
  mark = pos;
}

@Override
public synchronized void reset() throws IOException {
  checkOpen();
  pos = mark;
}

@Override
public synchronized void close() throws IOException {
  seq = null;
  }
```

都是比较简单的，ready()函数简单的判断seq是否没被清空，从而返回true或者抛出异常（见checkOpen()函数）。markSupported()函数只是简单的标识CharSequenceReader类支持标记这个方法。mark函数主要是记录下seq当前的下标。reset()函数是将当前的下标重置为mark。close()函数主要是清空seq中的数据。

我们可以从源码中发现，CharSequenceReader类中绝大部分的函数都是用了synchronized修饰的，这使得每一次只有一个线程执行相关的函数(完)