在《Fundamental Networking in Java》一书的 chapter 4-Scalable IO 中的 `14.7 Exceptions In New IO` 中(P 114) 是这样描述 `ClosedChannelException` 的：

Thrown by any channel operation if the channel is already closed, or by SocketChannel.write if the socket has been shutdown for output, or SocketChannel.read if the socket has been shutdown for input.

翻译成中文大意是：由任意通道的操作抛出,如果通道已经关闭,或者由 `SocketChannel.write` 抛出,如果套接字已经停止输出,或者 `SocketChannel.read` 抛出,如果套接字已经停止输入.

初看这句话是没有任何问题的.我们可以把这句话看成以下三种情况：

- 通道关闭， `SocketChannel.close()`
- `SocketChannel.shutdownOutput()` -> `SocketChannel.write()` 异常
- `SocketChannel.shutdownInput()` -> `SocketChannel.read()` 异常

不过实践是检验真理的唯一标准，下面就通过试验来验证，JDK版本：jdk1.7.0_09

示例1：通道关闭.

```
1.  public class ClosedChannelExceptionTest {
2.      public static void main(String[] args) throws IOException {
3.          ServerSocketChannel server = ServerSocketChannel.open();
4.
5.          // 关闭通道
6.          server.close();
7.
8.          // java.nio.channels.ClosedChannelException
9.          server.configureBlocking(true);
10.     }
11. }
```

示例2：SocketChannel.shutdownOutput()

```
1.  private void write(SocketChannel channel) throws IOException {
2.      String msg = "hello";
3.      ByteBuffer buffer = ByteBuffer.wrap(msg.getBytes());
4.      // XXX: 停止输出
5.      channel.shutdownOutput();
6.      channel.write(buffer);    // java.nio.channels.ClosedChannelException
7.  }
```

示例3：SocketChannel.shutdownInput()

```
1.  private void read(SocketChannel channel) throws IOException {
2.      ByteBuffer buffer = ByteBuffer.allocate(10);
3.      // XXX: 停止输入
4.      channel.shutdownInput();
5.      // 不抛出异常, 返回-1
6.      int n = channel.read(buffer);
7.      System.out.println("receive " + n + " bytes.");
8.  }
```

可以看到前两个示例中是没有问题的,抛出了 `ClosedChannelException` .而在示例3中, 执行 `channel.shutdownInput()` 之后 `channel.read(buffer)` 并未抛出异常.

原因分析：主要查看 `sun.nio.ch.SocketChannelImpl` 类(如果是反编译的话代码看起来会有些区别).

- `shutdownInput()` 、 `shutdownOutput()` 主要是对 `isInputOpen` 、 `isOutputOpen` 两个变量的操作.

```
@Override
public SocketChannel shutdownInput() throws IOException {
    synchronized (stateLock) {
        if (!isOpen())
            throw new ClosedChannelException();
        if (!isConnected())
            throw new NotYetConnectedException();
        if (isInputOpen) {
            Net.shutdown(fd, Net.SHUT_RD);
            if (readerThread != 0)
                NativeThread.signal(readerThread);
            isInputOpen = false;
        }
        return this;
    }
}

@Override
public SocketChannel shutdownOutput() throws IOException {
    synchronized (stateLock) {
        if (!isOpen())
            throw new ClosedChannelException();
        if (!isConnected())
            throw new NotYetConnectedException();
        if (isOutputOpen) {
            Net.shutdown(fd, Net.SHUT_WR);
            if (writerThread != 0)
                NativeThread.signal(writerThread);
            isOutputOpen = false;
        }
        return this;
    }
}
```

- `read()` 方法

```
public int read(ByteBuffer buf) throws IOException {

    if (buf == null)
        throw new NullPointerException();

    synchronized (readLock) {
        if (!ensureReadOpen())
            return -1;
```

`ensureReadOpen()` 方法:

```
private boolean ensureReadOpen() throws ClosedChannelException {
    synchronized (stateLock) {
        if (!isOpen())
            throw new ClosedChannelException();
        if (!isConnected())
            throw new NotYetConnectedException();
        if (!isInputOpen)
            return false;
        else
            return true;
    }
}
```

- `write()` 方法

```
public int write(ByteBuffer buf) throws IOException {
    if (buf == null)
        throw new NullPointerException();
    synchronized (writeLock) {
        ensureWriteOpen();
```

`ensureWriteOpen()` 方法：

```
private void ensureWriteOpen() throws ClosedChannelException {
    synchronized (stateLock) {
        if (!isOpen())
            throw new ClosedChannelException();
        if (!isOutputOpen)
            throw new ClosedChannelException();
        if (!isConnected())
            throw new NotYetConnectedException();
    }
}
```

- ensureReadOpen vs ensureWriteOpen

  将上面的两个方法对比一下：

```
private boolean ensureReadOpen() throws ClosedChannelException {
    synchronized (stateLock) {
        if (!isOpen())
            throw new ClosedChannelException();
        if (!isConnected())
            throw new NotYetConnectedException();
        if (!isInputOpen)
            return false;
        else
            return true;
    }
}

private void ensureWriteOpen() throws ClosedChannelException {
    synchronized (stateLock) {
        if (!isOpen())
            throw new ClosedChannelException();
        if (!isOutputOpen)
            throw new ClosedChannelException();
        if (!isConnected())
            throw new NotYetConnectedException();
    }
}
```

不抛出异常

抛出异常

- 低版本的 JDK

  考虑到本书使用的 JDK 的版本问题，查了下 JDK 1.4 中的 `ensureReadOpen` 方法，其实和上面是差不多的：

```
private boolean ensureReadOpen() throws ClosedChannelException {
    synchronized (this.stateLock) {
        if (!(isOpen()))
            throw new ClosedChannelException();
        if (!(isConnected())) {
            throw new NotYetConnectedException();
        }
        return (this.isInputOpen);
    }
}
```

综合上面,可以看到如果关闭了输入流(`SocketChannel.shutdownInput()`),那么 `read()` 方法直接返回 -1.而不是像 `write()` 方法那样,抛出异常.这里可以看到 `read()` 、 `write()` 方法之间的一些小差异.注意一下书中的小错误."SocketChannel.read if the socket has been shutdown for input"是有问题的.也就是 `channel.shutdownInput()` -> `SocketChannel.read` 抛出异常是不对的.