```
            nonBlockingWriteBuffer.add(from);
        }
    }
```

5）SocketWrapperBase 类核心方法之 flush 原理

flush 方法用于将缓冲区的数据写出到 socket 中，flush 也支持阻塞和非阻塞操作，如果是阻塞操作，那么调用 flushBlocking 方法，如果是非阻塞，则调用 flushNonBlocking 方法。

```
public boolean flush(boolean block) throws IOException {
        boolean result = false;
        if (block) {
            flushBlocking();
        } else {
            result = flushNonBlocking();
        }
        return result;
    }
```

6）SocketWrapperBase 类核心方法之 flushBlocking 阻塞原理

flushBlocking 方法可以阻塞式地清空缓冲区。首先调用子类实现的 doWrite 方法将缓冲区的数据写出，如果 nonBlockingWriteBuffer 非阻塞缓冲区不为空，那么调用 write 方法将其中的数据写出到 socket 写缓冲区，然后调用 doWrite 方法将 socket 写缓冲区数据写出到客户端，flushBlocking 方法将总是把缓冲区清空。详细实现如下。

```
protected void flushBlocking() throws IOException {
        // 子类实现 socket 写操作
        doWrite(true);
        // 写非阻塞写缓冲区数据
        if (!nonBlockingWriteBuffer.isEmpty()) {
            nonBlockingWriteBuffer.write(this, true);
            if (!socketBufferHandler.isWriteBufferEmpty()) {
                doWrite(true);
            }
        }
    }
```

7）SocketWrapperBase 类核心方法之 flushNonBlocking 非阻塞原理

flushNonBlocking 方法可以非阻塞式地清空缓冲区。首先判断 socket 写缓冲区是否为空，如果不为空，首先调用子类实现的 doWrite 方法刷新数据，传入的参数为 false，表明该操作也是非阻塞式操作，同时继续判断写缓冲区是否仍然存在数据，如果存在，表明子类非阻塞式写操作失败，这时可能是由于 socket 内部的写缓冲区满了，不能写入数据导致的。如果写缓冲区的数据子类写入成功，那么继续将非阻塞缓冲区的数据写入 socket 写缓冲区中，再次调用 doWrite 方法将数据写出到 socket 中。详细实现如下。

```
protected boolean flushNonBlocking() throws IOException {
        boolean dataLeft = !socketBufferHandler.isWriteBufferEmpty();
        // 写缓冲区存在数据
        if (dataLeft) {
            doWrite(false);
            dataLeft = !socketBufferHandler.isWriteBufferEmpty();
```