简述JAVA NIO

chao.zhu

- 阻塞式I/O的不足
- 非阻塞式I/O是如何弥补的
- 因为讲不了所以简单提一下nio实现及i/o模型

原始的WebServer可能是这样的

```
public class Initializer {
    public static void main(String[] args) throws IOException {
        Config.loadConfig();
        while (true) {
            try {
                Socket socket = Listener.listen(Config.port);
                String context = Reader.read(socket);
                Request request = ContextParser.parse(context);
                Action action = Dispatcher.dispatch(request);
                Response response = action.execute(request);
                Writer.write(socket, response);
                Closer.close(socket);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

阻塞式I/O

```
*/
public class Listener {
    private static ServerSocket serverSocket;

    public static Socket listen(int port) throws IOException {
        if (serverSocket == null) {
            serverSocket = new ServerSocket(port);
        }
        return serverSocket.accept();
    }
}
```

此处会阻塞住

```
public class Reader {
    public static String read(Socket socket) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        String line = bufferedReader.readLine();
        StringBuilder builder = new StringBuilder();
        while (line != null && line.length() > 0) {
            builder.append(line).append("\n");
            line = bufferedReader.readLine();
        }
        return builder.toString();
    }
}
```

此处也会阻塞住

为了处理并发，可能变成这个样子

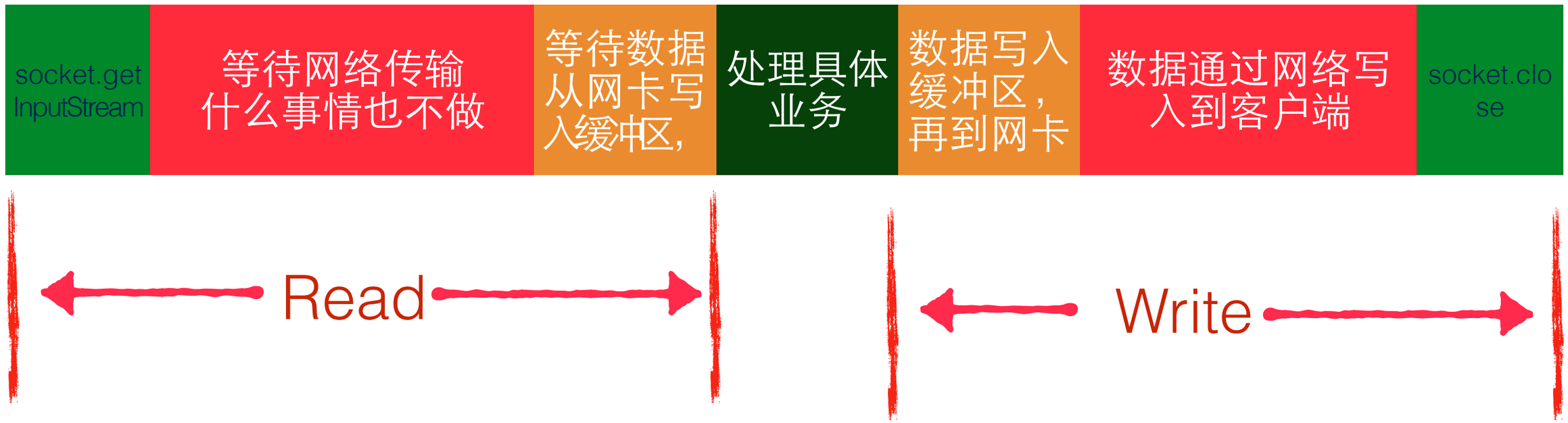
```
 * @version 1.0
 */
public class Initializer {
    public static void main(String[] args) throws IOException {
        Config.loadConfig();
        while (true) {
            try {
                Socket socket = Listener.listen(Config.port);
                //此处开启一个线程池处理请求
                Executor.excute(socket);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

在线程池中处理具体请求

这样写存在的问题

- 单线程只能线性处理请求
- 单线程不可能同时监听多个端口
- 多线程时，每个线程阻塞时间过长

从一个线程拿到socket到结束

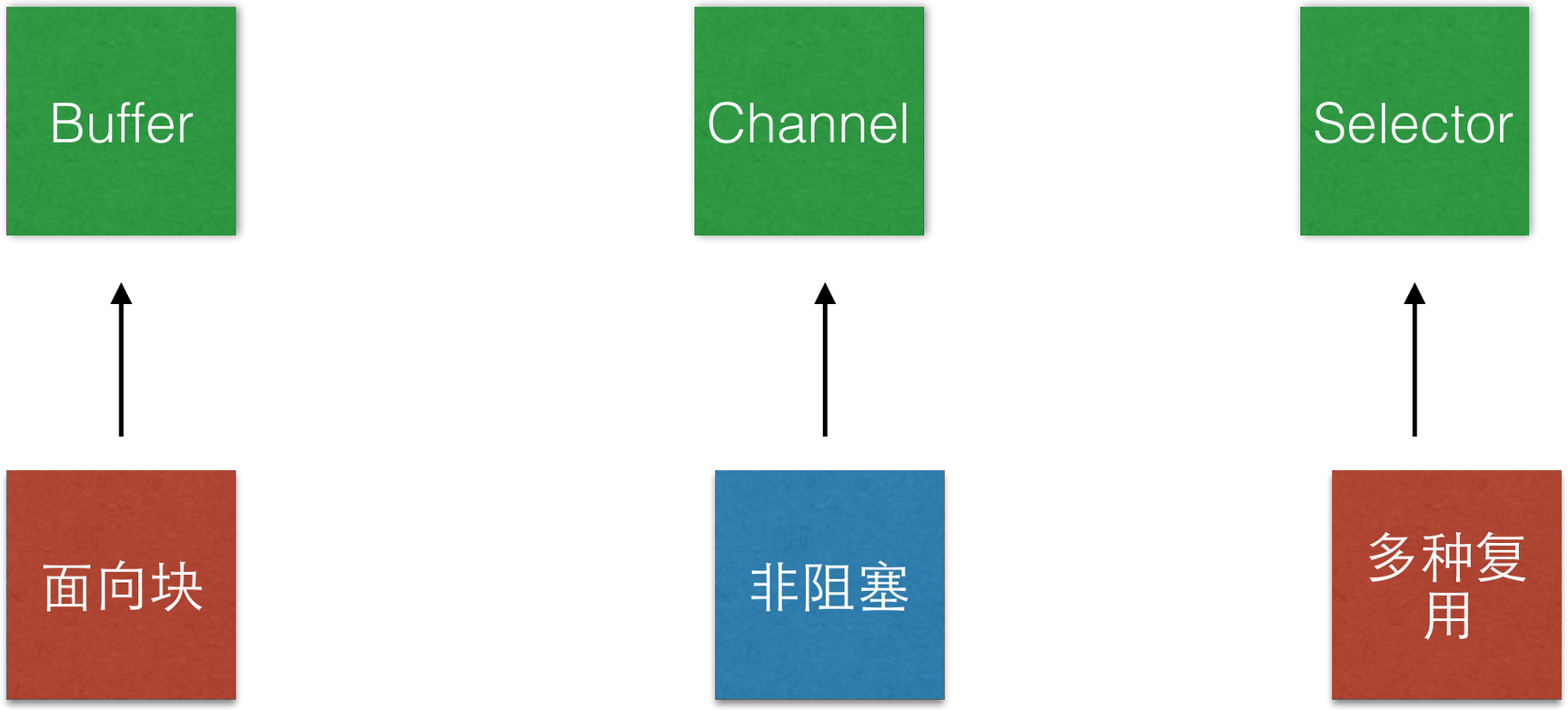


- Read与Write过程中线程什么也没有，但是仍然会参与线程切换
- 如果业务处理的时间很短，线程的大部分存活时间是没有必要的
- 但是阻塞的I/O，除此之外，别无他法

java从1.4推出了NIO

- NIO的一种解释是 New I/0
- 包含两大特性，面向缓冲区的块式I/O和非阻塞I/O
- 另一种解释是 Non-block I/O
- sun官方的解释是New I/O

NIO的三大构成部分

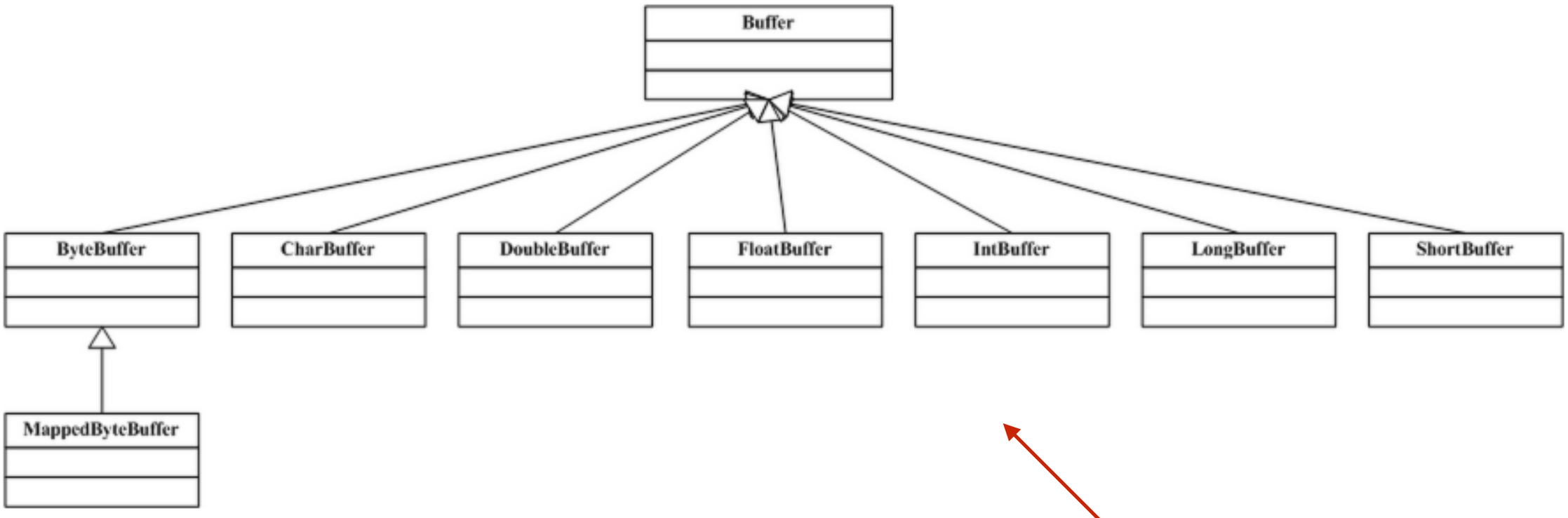


举个buffer的栗子

```
public class BufferTest {
    public static void main(String[] args) throws IOException {
        RandomAccessFile file = new RandomAccessFile(//
                BufferTest.class.getResource("/test.txt").getFile(), "
        FileChannel inChannel = file.getChannel();
        ByteBuffer byteBuffer = ByteBuffer.allocate(10);// buffer大小
        while (inChannel.read(byteBuffer) != -1) {
            byteBuffer.flip();
            // System.out.println(byteBuffer.toString());
            System.out.println(new String(byteBuffer.array()));
            // byteBuffer.compact();
            byteBuffer.clear();
        }
        file.close();
    }
}
```

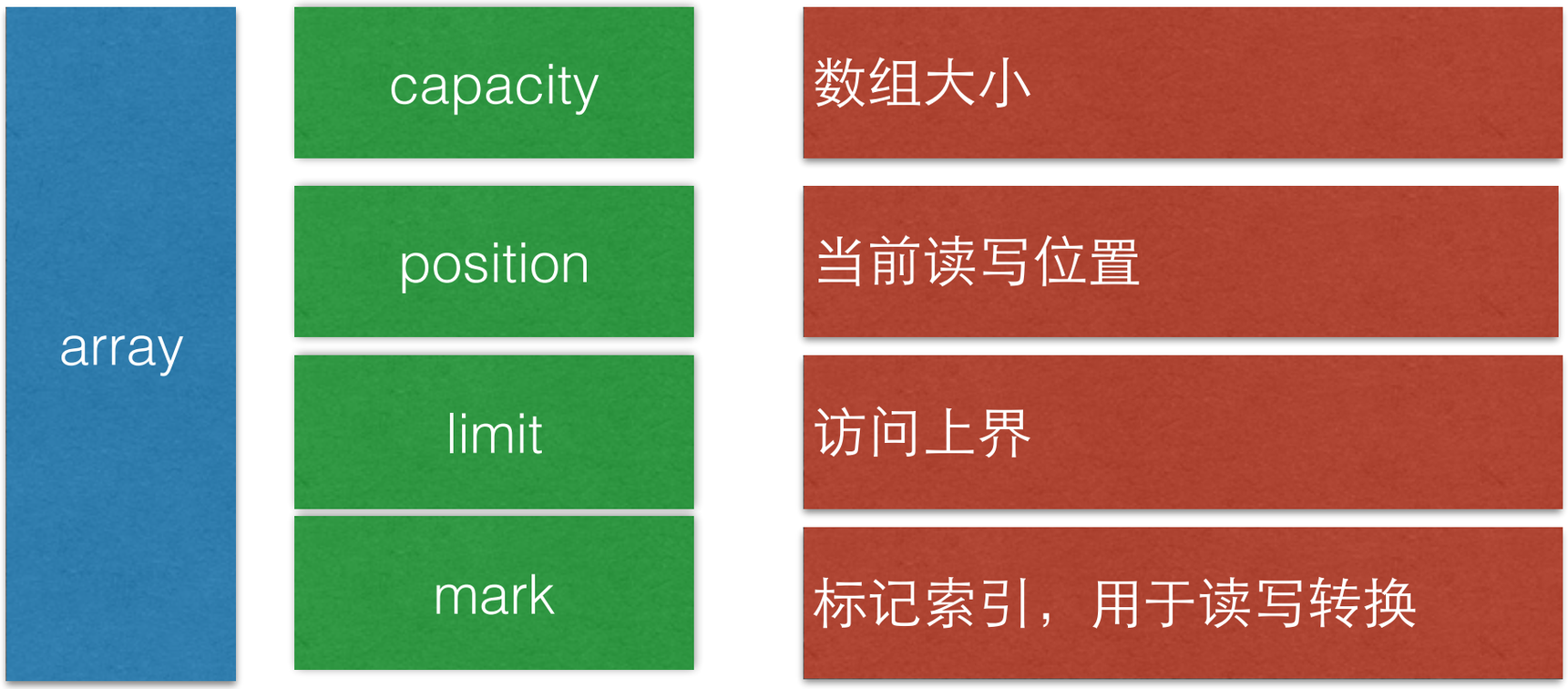
除了ByteBuffer还有很多，就不一一细说了

Buffer缓冲区的家谱如下图：



这个图是网上盗的
地址：http://www.cnblogs.com/

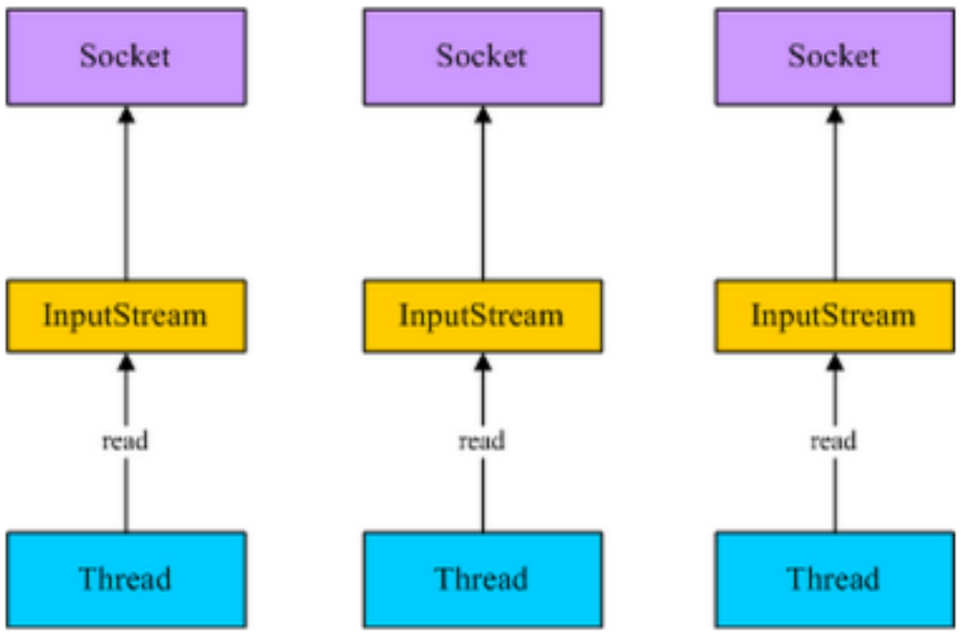
Buffer的实现相对简单，所以说一点点



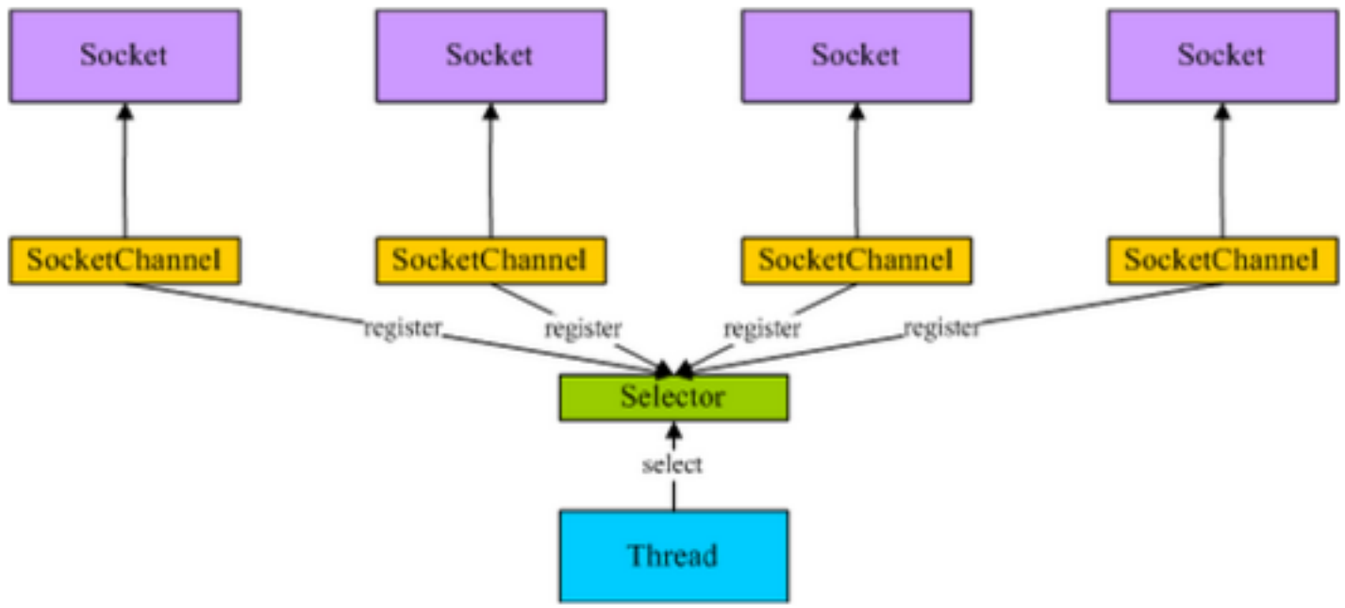
Buffer的大小会影响读写速度

- 操作文件 时一般设置为4k，读写速度效率最高
- 这个是我们老师说的
- 这个值跟硬盘文件格式有关，ntfs的块大小一般为4k
- 文件不会共用块，新文件总是从新的块开始写入

多路复用与非阻塞式I/O才是重点



Block



Non-Block

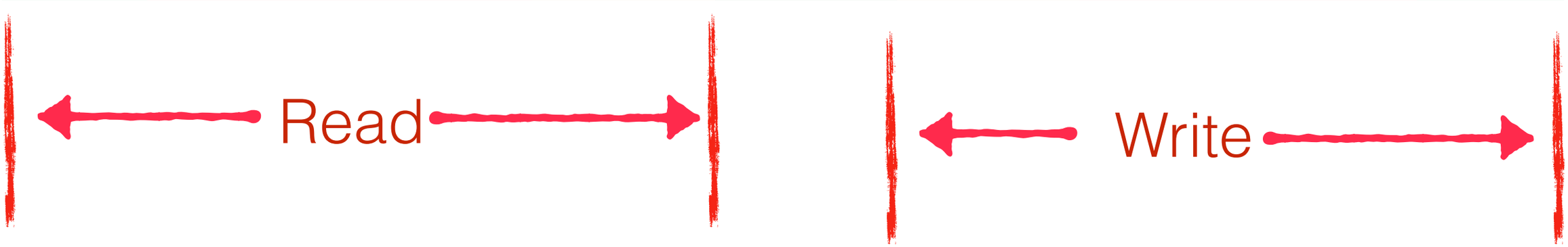
网址：这两个图也是盗的 http://www.cnblogs.com/zhuYears/archive/

Shut up mouth,show CODE

```
public class NioServer {
    public static void main(String[] args) throws IOException {
        Selector selector = Selector.open();
        Dispatcher.initRegister(selector);
        while (true) {
            int keyCount = selector.select();
            Iterator<SelectionKey> iterator = selector.selectedKeys().iterator();
            while (iterator.hasNext()) {
                SelectionKey key = iterator.next();
                Dispatcher.dispatch(key);
                iterator.remove();
            }
        }
    }
}
```

NIO到底快在哪里


socket.get InputStream
等待网络传输 什么事情也不做
等待数据从网卡写入缓冲区,
处理具体业务
数据写入缓冲区，再到网卡
数据通过网络写入到客户端
socket.close


Read
Write


socketChannel.read
我没有这一步啊 没有这一步啊
等待数据从网卡写入缓冲区,
处理具体业务
数据写入缓冲区，再到网卡
你丫还在等啊 但是我不用啊
socketChannerl.close

如此神奇的selector
是怎么实现的

- KQueueSelector （JDK1.7 默认 mac源码）
- PollSelector（Linux2.6以下默认）
- EPollSelectorProvider(jdk 1.5 以上Linux2.6及以上默认)

Linux三种多种复用实现

- select 1999
- poll 1999 Niels Provos
- epoll   2.6 Linux   On 11 July 2001 Davide Libenzi
- kqueue 是epoll在freeBsd下的实现
- 或者反来说 FreeBSD 4.3 2002 Jonathan Lemon 2000

详情参考博客:
http://www.cnblogs.com/Anker/p/3265058.html

Windows下有一种东西叫IOCP

- 这个东西还没有了解过，只知道它是异步I/O
- java7里新新推出 AI/O与之相似

Windows下有一种东西叫IOCP

- 这个东西还没有了解过，只知道它是异步I/O
- java7里新新推出 AI/O与之相似

两个名词

- 基于多路复用(multiple)的i/o模型被称之为Reactor Pattern
- 基于异步(asynchronous)I/O的模型被称之为 Preactor Pattern

提个资料说一下为什么这么设计

- C10K Problem
- Dan Kegel http://www.kegel.com/c10k.html

No more,
thanks for your attention.