Java 基础提升篇：NIO 基础详解

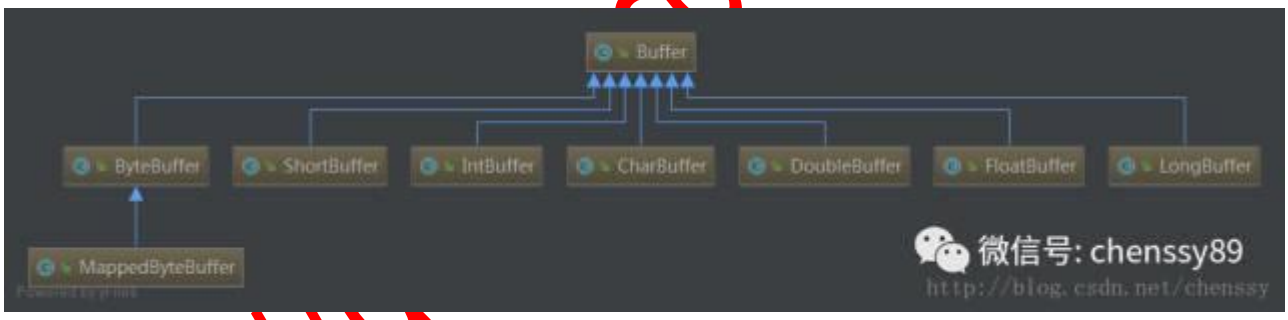
Netty 是基于 Java NIO 封装的网络通讯框架，只有充分理解了 Java NIO 才能理解好 Netty 的底层设计。Java NIO 由三个核心组件组件：

- Buffer
- Channel
- Selector

缓冲区 Buffer

Buffer 是一个数据对象，我们可以把它理解为固定数量的数据的容器，它包含一些要写入或者读出的数据。

在 Java NIO 中，任何时候访问 NIO 中的数据，都需要通过缓冲区（Buffer）进行操作。读取数据时，直接从缓冲区中读取，写入数据时，写入至缓冲区。NIO 最常用的缓冲区则是 ByteBuffer。下图是 Buffer 继承关系图：



每一个 Java 基本类型都对应着一种 Buffer，他们都包含这相同的操作，只不过是所处理的数据类型不同而已。

通道 Channel

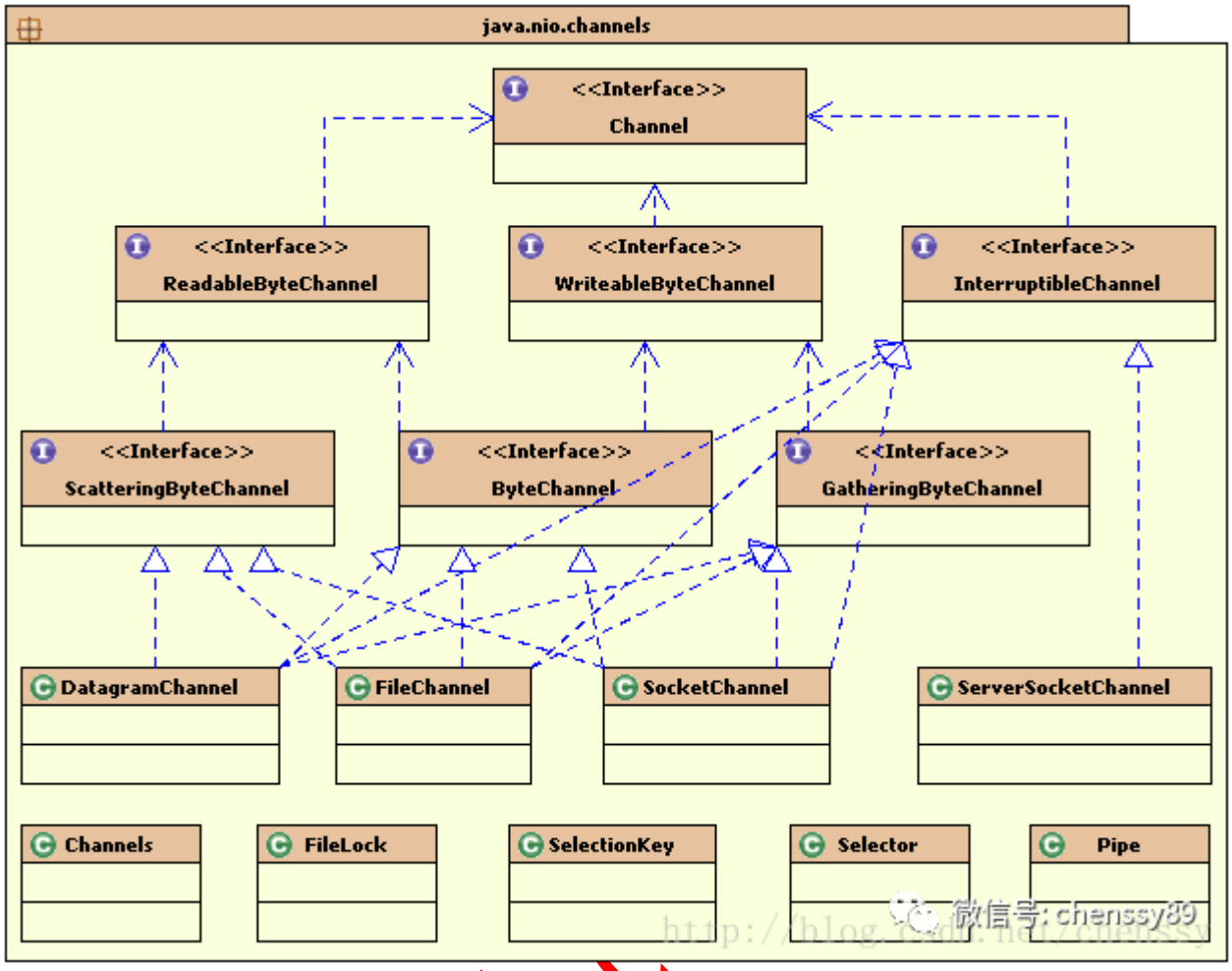
Channel 是一个通道，它就像自来水管一样，网络数据通过 Channel 这根水管读取和写入。传统的 IO 是基于流进行操作的，和 Channle 类似，但又有些不同：

区别	流	通过 Channel
是否支持异步	不支持	支持
是否可双向传输数据	不能，只能单向	可以，既可以从通道读取数据，也可以向通道写入数据
是否结合 Buffer 使用	不	必须结合 Buffer 使用
性能	较低	较高

正如上面说到的，Channel 必须要配合 Buffer 一起使用，我们永远不可能将数据直接写入到 Channel 中，同样也不可能直接从 Channel 中读取数据。都是通过从 Channel 读取数据到 Buffer 中或者从 Buffer 写入数据到 Channel 中，如下：

简单点说，Channel 是数据的源头或者数据的目的地，用于向 buffer 提供数据或者读取 buffer 数据，并且对 I/O 提供异步支持。

下图是 Channel 的类图:



Channel 为最顶层接口，所有子 Channel 都实现了该接口，它主要用于 I/O 操作的连接。定义如下：

```
public interface Channel extends Closeable {
    /**
     * 判断此通道是否处于打开状态。
     */
    public boolean isOpen();
    /**
     *关闭此通道。
     */
    public void close() throws IOException;
}
```

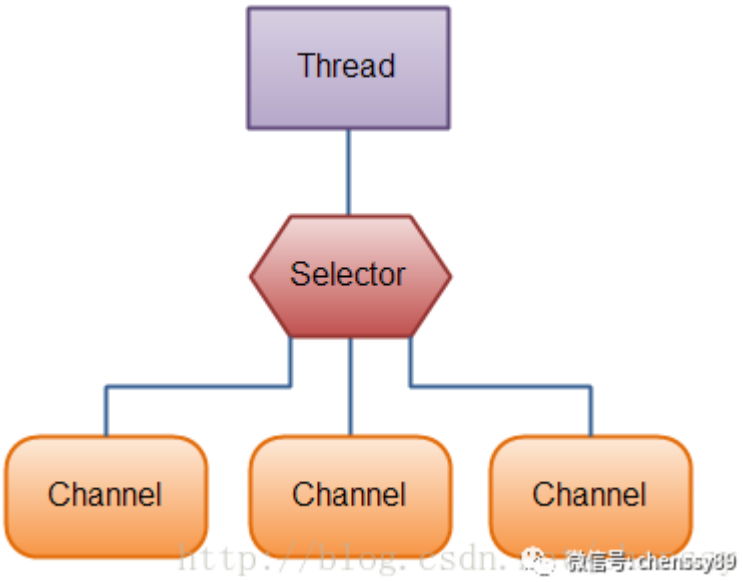
最为重要的 Channel 实现类为：

- FileChannel：一个用来写、读、映射和操作文件的通道
- DatagramChannel：能通过 UDP 读写网络中的数据
- SocketChannel: 能通过 TCP 读写网络中的数据
- ServerSocketChannel：可以监听新进来的 TCP 连接，像 Web 服务器那样。对每一个新进来的连接都会创建一个 SocketChannel

多路复用器 Selector

多路复用器 Selector，它是 Java NIO 编程的基础，它提供了选择已经就绪的任务的能力。从底层来看，Selector 提供了询问通道是否已经准备好执行每个 I/O 操作的能力。简单来讲，Selector 会不断地轮询注册在其上的 Channel，如果某个 Channel 上面发生了读或者写事件，这个 Channel 就处于就绪状态，会被 Selector 轮询出来，然后通过 SelectionKey 可以获取就绪 Channel 的集合，进行后续的 I/O 操作。

Selector 允许一个线程处理多个 Channel ，也就是说只要一个线程复杂 Selector 的轮询，就可以处理成千上万个 Channel ，相比于多线程来处理势必会减少线程的上下文切换问题。下图是一个 Selector 连接三个 Channel ：



实例

服务端

```
public class NIOServer {
    /*接受数据缓冲区*/
    private ByteBuffer sendbuffer = ByteBuffer.allocate(1024);
    /*发送数据缓冲区*/
    private  ByteBuffer receivebuffer = ByteBuffer.allocate(1024);
    private Selector selector;
    public NIOServer(int port) throws IOException {
        // 打开服务器套接字通道
        ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
```

```
        // 服务器配置为非阻塞
        serverSocketChannel.configureBlocking(false);
        // 检索与此通道关联的服务器套接字
        ServerSocket serverSocket = serverSocketChannel.socket();
        // 进行服务的绑定
        serverSocket.bind(new InetSocketAddress(port));
        // 通过 open()方法找到 Selector
        selector = Selector.open();
        // 注册到 selector，等待连接
        serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
        System.out.println("Server Start----:");
    }
    //
    private void listen() throws IOException {
        while (true) {
            selector.select();
            Set<SelectionKey> selectionKeys = selector.selectedKeys();
            Iterator<SelectionKey> iterator = selectionKeys.iterator();
            while (iterator.hasNext()) {
                SelectionKey selectionKey = iterator.next();
                iterator.remove();
                handleKey(selectionKey);
            }
        }
    }
    private void handleKey(SelectionKey selectionKey) throws IOException {
        // 接受请求
        ServerSocketChannel server = null;
        SocketChannel client = null;
        String receiveText;
        String sendText;
        int count=0;
        // 测试此键的通道是否已准备好接受新的套接字连接。
        if (selectionKey.isAcceptable()) {
            // 返回为之创建此键的通道。
            server = (ServerSocketChannel) selectionKey.channel();
            // 接受到此通道套接字的连接。
            // 此方法返回的套接字通道（如果有）将处于阻塞模式。
```

```
            client = server.accept();
            // 配置为非阻塞
            client.configureBlocking(false);
            // 注册到 selector，等待连接
            client.register(selector, SelectionKey.OP_READ);
        } else if (selectionKey.isReadable()) {
            // 返回为之创建此键的通道。
            client = (SocketChannel) selectionKey.channel();
            //将缓冲区清空以备下次读取
            receivebuffer.clear();
            //读取服务器发送来的数据到缓冲区中
            count = client.read(receivebuffer);
            if (count > 0) {
                receiveText = new String( receivebuffer.array(),0,count);
                System.out.println("服务器端接受客户端数据--:"+receiveText);
                client.register(selector, SelectionKey.OP_WRITE);
            }
        } else if (selectionKey.isWritable()) {
            //将缓冲区清空以备下次写入
            sendbuffer.clear();
            // 返回为之创建此键的通道。
            client = (SocketChannel) selectionKey.channel();
            sendText="message from server--";
            //向缓冲区中输入数据
            sendbuffer.put(sendText.getBytes());
            //将缓冲区各标志复位,因为向里面 put 了数据标志被改变要想从中读取数据发向服务
器,就要复位
            sendbuffer.flip();
            //输出到通道
            client.write(sendbuffer);
            System.out.println("服务器端向客户端发送数据--："+sendText);
            client.register(selector, SelectionKey.OP_READ);
        }
    }
    /**
    * @param args
    * @throws IOException
    */
```

```
    public static void main(String[] args) throws IOException {
        int port = 8080;
        NIOServer server = new NIOServer(port);
        server.listen();
    }
}
```

客户端

```
public class NIOClient {
    /*接受数据缓冲区*/
    private static ByteBuffer sendbuffer = ByteBuffer.allocate(1024);
    /*发送数据缓冲区*/
    private static ByteBuffer receivebuffer = ByteBuffer.allocate(1024);
    public static void main(String[] args) throws IOException {
        // 打开 socket 通道
        SocketChannel socketChannel = SocketChannel.open();
        // 设置为非阻塞方式
        socketChannel.configureBlocking(false);
        // 打开选择器
        Selector selector = Selector.open();
        // 注册连接服务端 socket 动作
        socketChannel.register(selector, SelectionKey.OP_CONNECT);
        // 连接
        socketChannel.connect(new InetSocketAddress("127.0.0.1", 8080));
        Set<SelectionKey> selectionKeys;
        Iterator<SelectionKey> iterator;
        SelectionKey selectionKey;
        SocketChannel client;
        String receiveText;
        String sendText;
        int count=0;
        while (true) {
            //选择一组键，其相应的通道已为 I/O 操作准备就绪。
            //此方法执行处于阻塞模式的选择操作。
            selector.select();
            //返回此选择器的已选择键集。
            selectionKeys = selector.selectedKeys();
            //System.out.println(selectionKeys.size());
```

```
            iterator = selectionKeys.iterator();
            while (iterator.hasNext()) {
                selectionKey = iterator.next();
                if (selectionKey.isConnectable()) {
                    System.out.println("client connect");
                    client = (SocketChannel) selectionKey.channel();
                    // 判断此通道上是否正在进行连接操作。
                    // 完成套接字通道的连接过程。
                    if (client.isConnectionPending()) {
                        client.finishConnect();
                        System.out.println("完成连接!");
                        sendbuffer.clear();
                        sendbuffer.put("Hello,Server".getBytes());
                        sendbuffer.flip();
                        client.write(sendbuffer);
                    }
                    client.register(selector, SelectionKey.OP_READ);
                } else if (selectionKey.isReadable()) {
                    client = (SocketChannel) selectionKey.channel();
                    //将缓冲区清空以备下次读取
                    receivebuffer.clear();
                    //读取服务器发送来的数据到缓冲区中
                    count=client.read(receivebuffer);
                    if(count>0){
                        receiveText = new String( receivebuffer.array(),0,count);
                        System.out.println("客户端接受服务器端数据--:"+receiveText);
                        client.register(selector, SelectionKey.OP_WRITE);
                    }
                } else if (selectionKey.isWritable()) {
                    sendbuffer.clear();
                    client = (SocketChannel) selectionKey.channel();
                    sendText = "message from client--";
                    sendbuffer.put(sendText.getBytes());
                    //将缓冲区各标志复位,因为向里面 put 了数据标志被改变要想从中读取数据
发向服务器,就要复位
                    sendbuffer.flip();
                    client.write(sendbuffer);
                    System.out.println("客户端向服务器端发送数据--："+sendText);
```

```
                    client.register(selector, SelectionKey.OP_READ);
                }
            }
            selectionKeys.clear();
        }
    }
}
```

运行结果