实战Java高并发程序设计第8周

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

内容提要

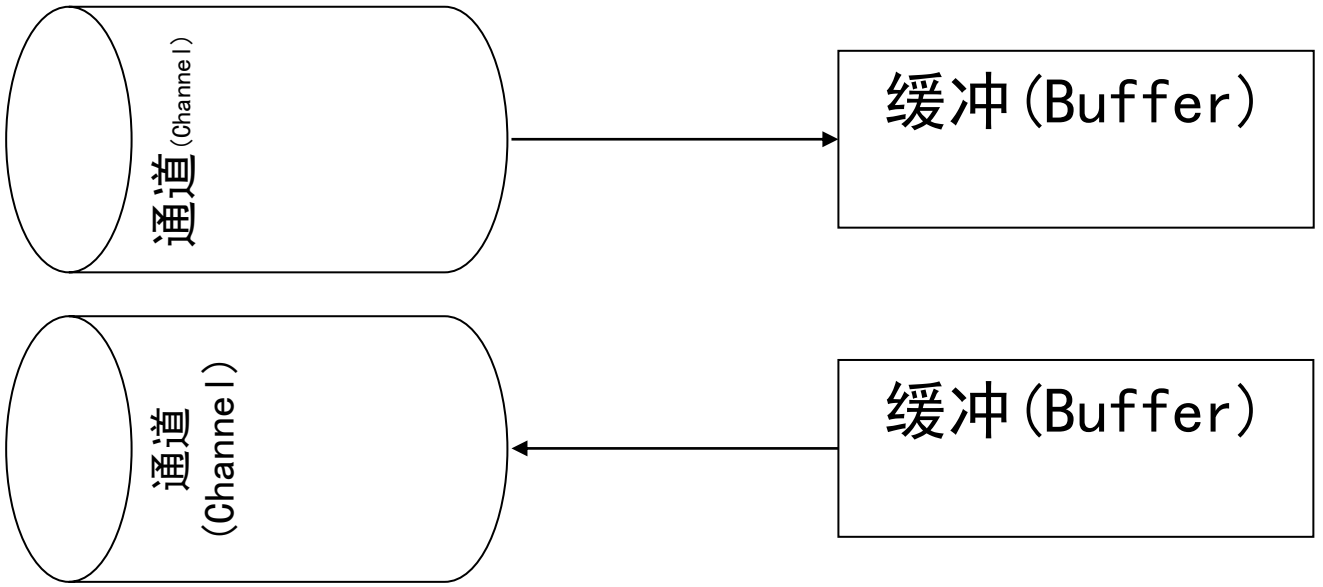
- 什么是NIO
- Buffer
- Channel
- 网络编程
- AIO

- 为什么需要了解NIO和AIO？

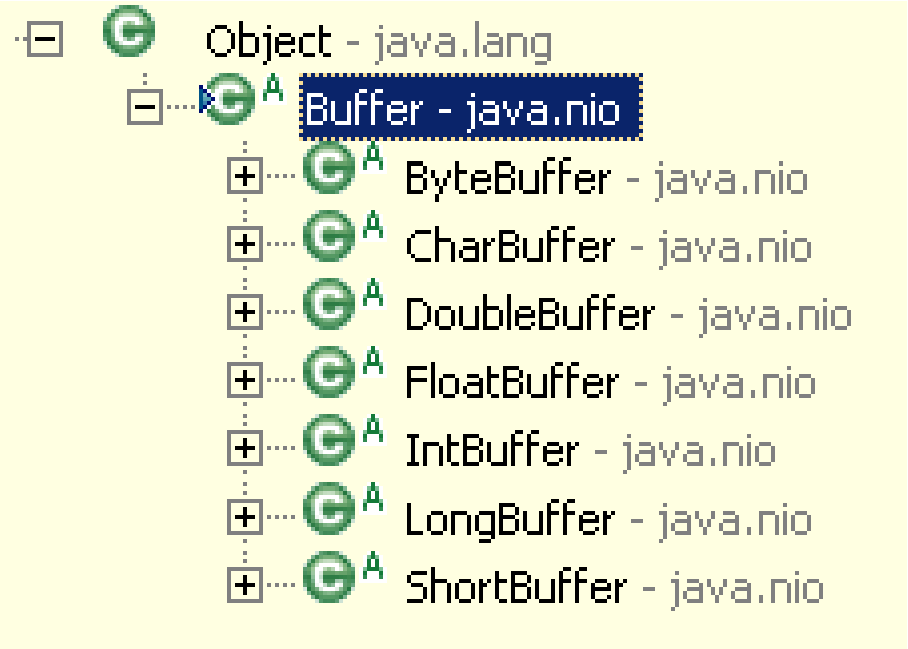
什么是NIO

- NIO是New I/O的简称，与旧式的基于流的I/O方法相对，从名字看，它表示新的一套Java I/O标准。它是在Java 1.4中被纳入到JDK中的，并具有以下特性：
  - NIO是基于块（Block）的，它以块为基本单位处理数据
  - 为所有的原始类型提供（Buffer）缓存支持
  - 增加通道（Channel）对象，作为新的原始 I/O 抽象
  - 支持锁和内存映射文件的文件访问接口
  - 提供了基于Selector的异步网络I/O

Buffer && Channel



Buffer


Object - java.lang
Buffer - java.nio
ByteBuffer - java.nio
CharBuffer - java.nio
DoubleBuffer - java.nio
FloatBuffer - java.nio
IntBuffer - java.nio
LongBuffer - java.nio
ShortBuffer - java.nio

Buffer

```
FileInputStream fin = new FileInputStream(new File("d:\\temp_buffer.tmp"));
FileChannel fc=fin.getChannel();
```

```
ByteBuffer byteBuffer=ByteBuffer.allocate(1024);
fc.read(byteBuffer);
```

```
fc.close();
byteBuffer.flip();
```

Buffer

NIO复制文件

```
public static void nioCopyFile(String resource, String destination)
                        throws IOException {
        FileInputStream fis = new FileInputStream(resource);
        FileOutputStream fos = new FileOutputStream(destination);
        FileChannel readChannel = fis.getChannel();                    //读文件通道
        FileChannel writeChannel = fos.getChannel();                   //写文件通道
        ByteBuffer buffer = ByteBuffer.allocate(1024);                 //读入数据缓存
        while (true) {
                buffer.clear();
                int len = readChannel.read(buffer);                    //读入数据
                if (len == -1) {
                        break;
                        //读取完毕
                }
                buffer.flip();
                writeChannel.write(buffer);
        //写入文件
        }
        readChannel.close();
        writeChannel.close();
}
```

Buffer

- Buffer中有3个重要的参数：位置（position）、容量（capactiy）和上限（limit）

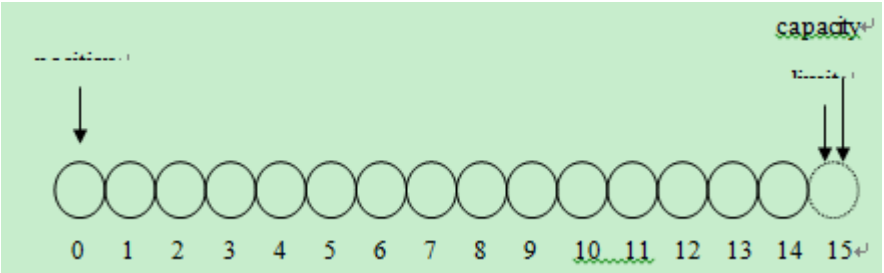
参数	写模式	读模式
位置(position)	当前缓冲区的位置，将从position的下一个位置写数据	当前缓冲区读取的位置，将从此位置后，读取数据
容量(capactiy)	缓存区的总容量上限	缓存区的总容量上限
上限(limit)	缓冲区的实际上限，它总是小于等于容量。通常情况下，和容量相等。	代表可读取的总容量，和上次写入的数据量相等。

Buffer

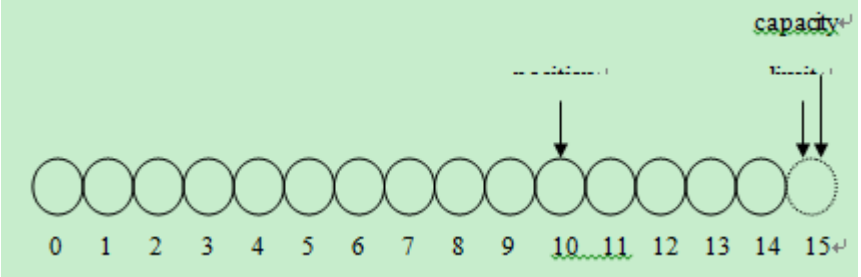
```
ByteBuffer b=ByteBuffer.allocate(15);                    //15个字节大小的缓冲区
System.out.println("limit="+b.limit()+" capacity="+b.capacity()+" position="+b.position());
for(int i=0;i<10;i++){                                                    //存入10个字节数据
        b.put((byte)i);
}
System.out.println("limit="+b.limit()+" capacity="+b.capacity()+" position="+b.position());
b.flip();                                                                 //重置position
System.out.println("limit="+b.limit()+" capacity="+b.capacity()+" position="+b.position());
for(int i=0;i<5;i++){
        System.out.print(b.get());
}
System.out.println();
System.out.println("limit="+b.limit()+" capacity="+b.capacity()+" position="+b.position());
b.flip();
System.out.println("limit="+b.limit()+" capacity="+b.capacity()+" position="+b.position());
```

Buffer

新建

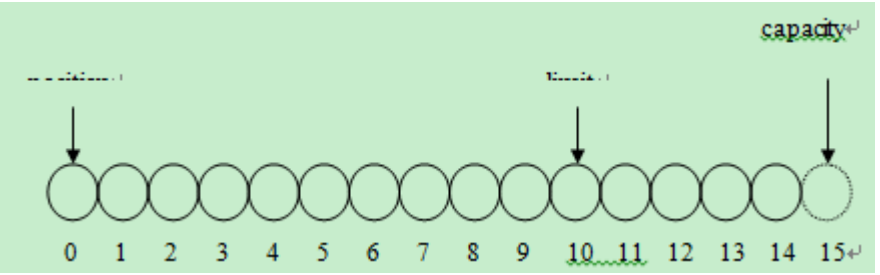


存入10byte



Buffer

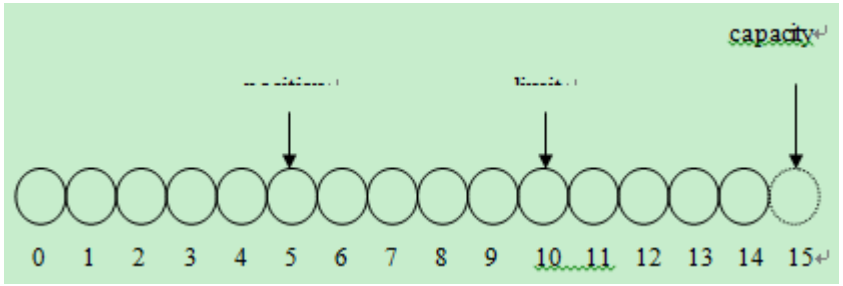
flip



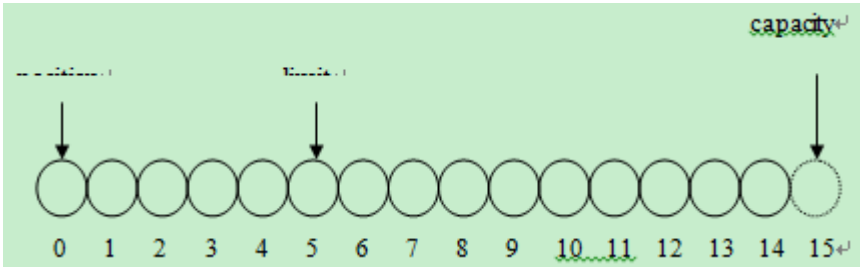
该操作会重置position，通常，将buffer从写模式转换为读模式时需要执行此方法
flip()操作不仅重置了当前的position为0，还将limit设置到当前position的位置

Buffer

5次读操作



flip



Buffer

- public final Buffer rewind()
  - 将position置零，并清除标志位（mark）
- public final Buffer clear()
  - 将position置零，同时将limit设置为capacity的大小，并清除了标志mark
- public final Buffer flip()
  - 先将limit设置到position所在位置，然后将position置零，并清除标志位mark
  - 通常在读写转换时使用

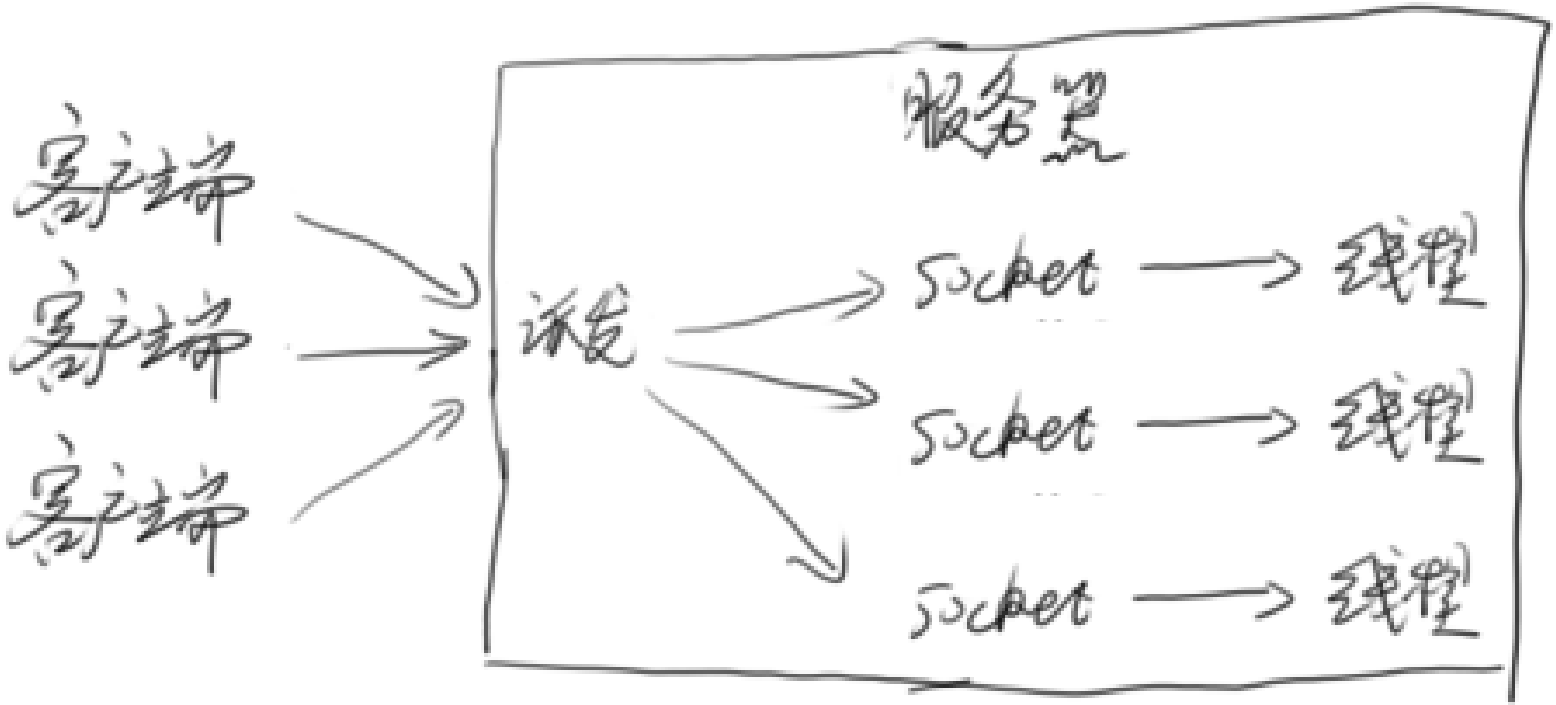
Buffer

- 文件映射到内存

```
RandomAccessFile raf = new RandomAccessFile("C:\\mapfile.txt", "rw");
FileChannel fc = raf.getChannel();
//将文件映射到内存中
MappedByteBuffer mbb = fc.map(FileChannel.MapMode.READ_WRITE, 0, raf.length());
while(mbb.hasRemaining()){
        System.out.print((char)mbb.get());
}
mbb.put(0,(byte)98);    //修改文件
raf.close();
```

网络编程

- 多线程网络服务器的一般结构



网络编程

- 简单案例 EchoServer

```
public static void main(String args[]) {
        ServerSocket echoServer = null;
        Socket clientSocket = null;
        try {
            echoServer = new ServerSocket(8000);
        } catch (IOException e) {
            System.out.println(e);
        }
        while (true) {
            try {
                clientSocket = echoServer.accept();
                System.out.println(clientSocket.getRemoteSocketAddress() + " connect!");
                tp.execute(new HandleMsg(clientSocket));
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
```

网络编程

```
static class HandleMsg implements Runnable{
        省略部分信息
    public void run(){
    try {
    is = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        os = new PrintWriter(clientSocket.getOutputStream(), true);
        // 从InputStream当中读取客户端所发送的数据
        String inputLine = null;
        long b=System.currentTimeMillis();
        while ((inputLine = is.readLine()) != null) {
            os.println(inputLine);
        }
        long e=System.currentTimeMillis();
        System.out.println("spend:"+(e-b)+"ms");
      } catch (IOException e) {
        e.printStackTrace();
      }finally{
        关闭资源
}
   }
```

网络编程

- EchoServer的客户端

```
public static void main(String[] args) throws IOException {
    Socket client = null;
    PrintWriter writer = null;
    BufferedReader reader = null;
    try {
        client = new Socket();
        client.connect(new InetSocketAddress("localhost", 8000));
        writer = new PrintWriter(client.getOutputStream(), true);
        writer.println("Hello!");
        writer.flush();

        reader = new BufferedReader(new InputStreamReader(client.getInputStream()));
        System.out.println("from server: " + reader.readLine());
    } catch
    } finally {
        //省略资源关闭
    }
}
```

网络编程

- 问题：
  - 为每一个客户端使用一个线程，如果客户端出现延时等异常，线程可能会被占用很长时间。因为数据的准备和读取都在这个线程中。
  - 此时，如果客户端数量众多，可能会消耗大量的系统资源
- 解决
  - 非阻塞的NIO
  - 数据准备好了在工作

网络编程-模拟低效的客户端

```
private static ExecutorService  tp=Executors.newCachedThreadPool();
private static final int sleep_time=1000*1000*1000;
public static class EchoClient implements Runnable{
        public void run(){
    try {
        client = new Socket();
        client.connect(new InetSocketAddress("localhost", 8000));
        writer = new PrintWriter(client.getOutputStream(), true);
        writer.print("H");
        LockSupport.parkNanos(sleep_time);
        writer.print("e");
        LockSupport.parkNanos(sleep_time);
        writer.print("l");
        LockSupport.parkNanos(sleep_time);
        writer.print("l");
        LockSupport.parkNanos(sleep_time);
        writer.print("o");
        LockSupport.parkNanos(sleep_time);
        writer.print("!");
        LockSupport.parkNanos(sleep_time);
        writer.println();
        writer.flush();
```

网络编程

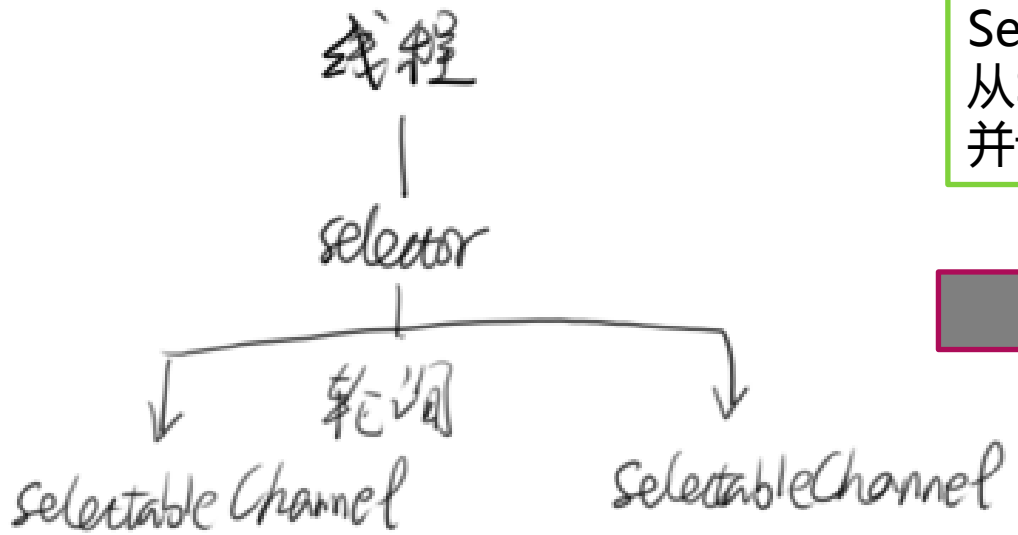
- 服务器输出如下：

```
spend:6000ms
spend:6000ms
spend:6000ms
spend:6001ms
spend:6002ms
spend:6002ms
spend:6002ms
spend:6002ms
spend:6003ms
spend:6003ms
```

网络编程-NIO

- 把数据准备好了再通知我

Channel有点类似于流，一个Channel可以和文件或者网络Socket对应



Selector准备好数据后，返回SelectionKey
SelectionKey表示一对Selector和Channel的关系，
从SelectionKey中可以得到Channnel（数据已经准备），并读取数据

select()和selectNow()

网络编程

- 参考代码
- 总结：
  - NIO会将数据准备好后，再交由应用进行处理，数据的读取过程依然在应用线程中完成
  - 节省数据准备时间（因为Selector可以复用）

网络编程 AIO

- 读完了再通知我
- 不会加快IO，只是在读完后进行通知
- 使用回调函数，进行业务处理

网络编程 AIO

- AsynchronousServerSocketChannel

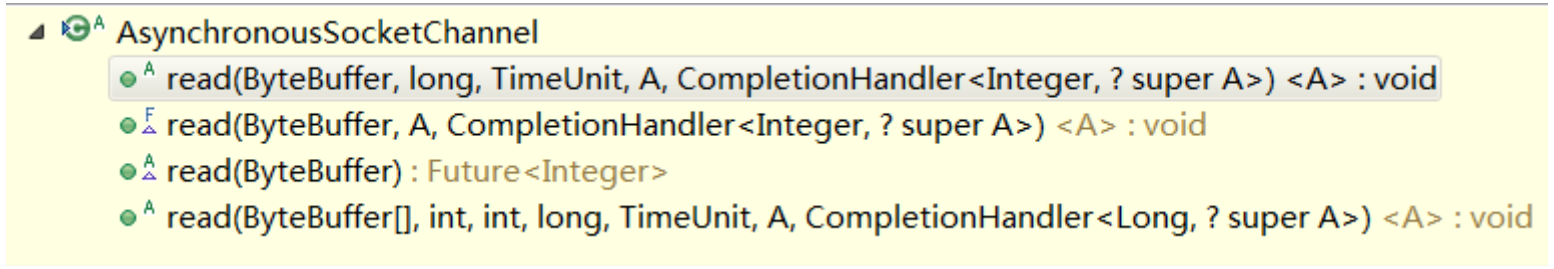
```
server = AsynchronousServerSocketChannel.open().bind(new InetSocketAddress(PORT));
```

使用server上的accept方法

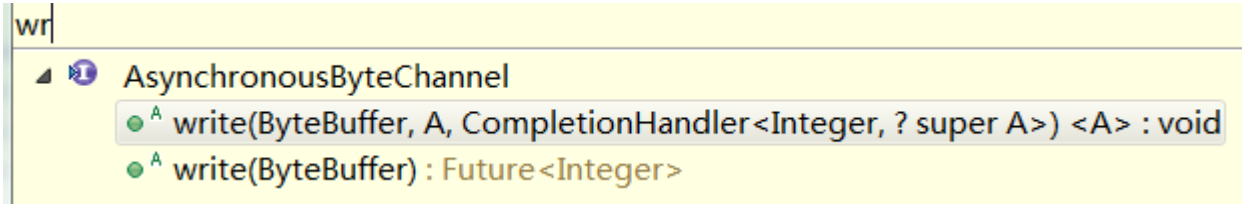
```
public abstract <A> void accept(A attachment,
                                CompletionHandler<AsynchronousSocketChannel,? super A>
handler);
```

网络编程 AIO

- AsynchronousSocketChannel
  - read



  - write



```
server.accept(null, new CompletionHandler<AsynchronousSocketChannel, Object>() {
    final ByteBuffer buffer = ByteBuffer.allocate(1024);
    public void completed(AsynchronousSocketChannel result, Object attachment) {
        System.out.println(Thread.currentThread().getName());
        Future<Integer> writeResult=null;
        try {
            buffer.clear();
            result.read(buffer).get(100, TimeUnit.SECONDS);
            buffer.flip();
            writeResult=result.write(buffer);
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        } catch (TimeoutException e) {
            e.printStackTrace();
        } finally {
            try {
                server.accept(null, this);
                writeResult.get();
                result.close();
            } catch (Exception e) {
                System.out.println(e.toString());
            }
        }
    }

    @Override
    public void failed(Throwable exc, Object attachment) {
        System.out.println("failed: " + exc);
    }
});
```

Thanks

FAQ时间