Санкт-Петербургский Национальный Исследовательский Университет ИТМО

Факультет Программной Инженерии и Компьютерной Техники

Реферат

По дисциплине

Программирование

На тему: «Асинхронные каналы nio»

Выполнил студент группы P3108, поток 2.5:

Петров Вячеслав Маркович

Преподаватель:

Гаврилов Антон Валерьевич

Санкт-Петербург 2024 г.

Оглавление

Введение

В предложенной работе будут рассматриваться классы AsynchronousFileChannel и AsynchronousSocketChannel: преимущества, недостатки и их особенности реализации на примере реальных кодов.

Если оценивать поверхностно, то: AsynchronousSocketChannel — класс в Java, который представляет асинхронный канал для потокового ввода-вывода по сети. А AsynchronousFileChannel — класс, который представляет асинхронный канал для чтения, записи и манипуляций с файлами. Оба класса были введены в Java 7, как часть пакета `java.nio.channels`.

Сразу видно, что оба класса ориентированы на похожие задачи, поэтому разберёмся в чём же их отличие и как выбрать между ними в зависимости от условий.

Future и CompletionHandler

Примеры реализации AsynchronousFileChannel и AsynchronousSocketChannel будут рассматриваться с помощью Future и CompletionHandler, поэтому стоит разобраться, в чём же эти интерфейсы отличаются и какой удобнее использовать для нужной задачи.

Особенности Future:

- Интерфейс представляет результат асинхронной операции, которая будет завершена в будущем.
- С помощью Future можно проверить, завершилась ли операция, - метод isDone() и получить результат - метод get().
- Предоставляет возможность блокировать поток выполнения, ожидая завершения операции.

Особенности CompletionHandler:

- Этот интерфейс используется для обработки результата асинхронной операции, когда она завершится.
- При использовании CompletionHandler, вы указываете код, который должен выполниться после завершения операции.
- В отличие от Future, позволяет задать различные действия для успешного завершения операции и для ошибки.

Тогда можно выделить главное различие этих интерфейсов: Future просто представляет результат асинхронной операции, в то время как CompletionHandler позволяет указать действия, которые нужно выполнить по завершении операции, включая обработку успешного завершения и ошибок.

Значит CompletionHandler следует использовать для более детальной обработки результатов, когда нам важно при успешной записи выполнить какое-то определенное условие, а Future просто для хранения результата, при этом имея возможность влиять на поток выполнения, однако при этом реализация CompletionHandler требует большего внимания от разработчика.

AsynchronousFileChannel

Главные характеристики:

1. Асинхронность:

AsynchronousFileChannel позволяет выполнять операции чтения и записи асинхронно, что означает, что ваш поток не блокируется во время выполнения этих операций. Это особенно полезно для повышения производительности ввода-вывода (I/O) в приложениях, где необходимо обрабатывать большое количество операций с файлами.

2. Особенности работы с файлами:

AsynchronousFileChannel не имеет текущей позиции в файле. Вместо этого позиция файла указывается для каждого метода чтения и записи, который инициирует асинхронные операции. А CompletionHandler указывается как параметр и вызывается для использования результата операции ввода-вывода. Этот класс также определяет методы чтения и записи, которые инициируют асинхронные операции, возвращая Future для представления ожидаемого результата операции.

Основные методы:

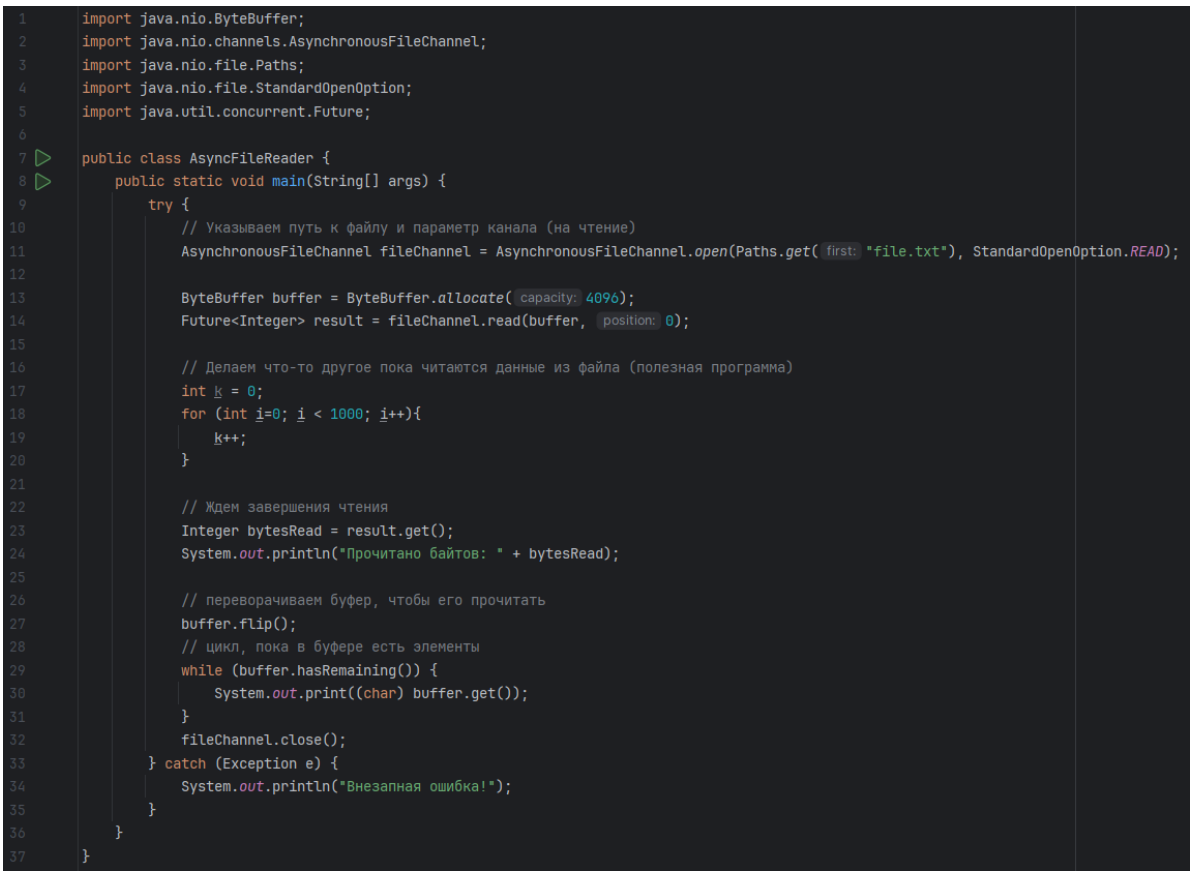
1. read(ByteBuffer dst, long position): Читает байты из файла в заданное положение в буфере.
2. write(ByteBuffer src, long position): Записывает байты в файл из заданного положения в буфере.
3. lock(): Получает блокировку файла.
4. force(boolean metaData): Принудительно записывает данные на диск.
5. size(): Возвращает размер файла.
6. truncate(long size): Укорачивает файл до указанного размера.

Можно выделить следующие преимущества:

1. Повышенная производительность - асинхронные операции позволяют другим частям программы продолжать выполняться, что может повысить общую производительность.
2. Гибкость - возможность использования как Future, так и CompletionHandler для обработки завершения операций.

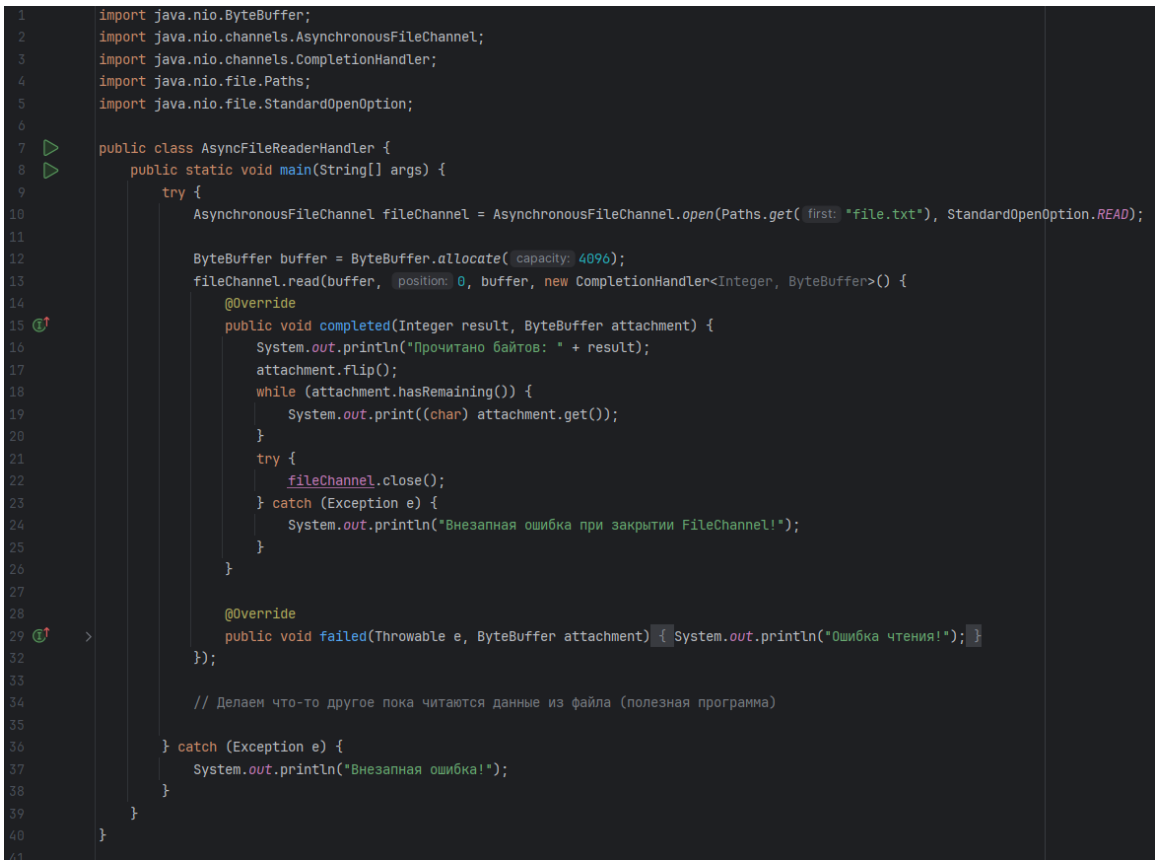
Примеры использования (с Future и CompletionHandler):

Ниже представлены листинги программ, использующих AsynchronousFileChannel с Future или CompletionHandler. Также добавлена ссылка на исходный код на https://github.com/sub-myitmo/example.



```java
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousFileChannel;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.concurrent.Future;

public class AsyncFileReader {
    public static void main(String[] args) {
        try {
            // Указываем путь к файлу и параметр канала (на чтение)
            AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(Paths.get( first: "file.txt"), StandardOpenOption.READ);

            ByteBuffer buffer = ByteBuffer.allocate( capacity: 4096);
            Future<Integer> result = fileChannel.read(buffer,  position: 0);

            // Делаем что-то другое пока читаются данные из файла (полезная программа)
            int k = 0;
            for (int i=0; i < 1000; i++){
                k++;
            }

            // Ждем завершения чтения
            Integer bytesRead = result.get();
            System.out.println("Прочитано байтов: " + bytesRead);

            // переворачиваем буфер, чтобы его прочитать
            buffer.flip();
            // цикл, пока в буфере есть элементы
            while (buffer.hasRemaining()) {
                System.out.print((char) buffer.get());
            }
            fileChannel.close();
        } catch (Exception e) {
            System.out.println("Внезапная ошибка!");
        }
    }
}
```

Рисунок 1 – чтение из файла с Future (https://github.com/sub-myitmo/example/blob/main/src/AsyncFileReader.java)



```java
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousFileChannel;
import java.nio.channels.CompletionHandler;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class AsyncFileReaderHandler {
    public static void main(String[] args) {
        try {
            AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(Paths.get( first: "file.txt"), StandardOpenOption.READ);

            ByteBuffer buffer = ByteBuffer.allocate( capacity: 4096);
            fileChannel.read(buffer,  position: 0, buffer, new CompletionHandler<Integer, ByteBuffer>() {
                @Override
                public void completed(Integer result, ByteBuffer attachment) {
                    System.out.println("Прочитано байтов: " + result);
                    attachment.flip();
                    while (attachment.hasRemaining()) {
                        System.out.print((char) attachment.get());
                    }
                    try {
                        fileChannel.close();
                    } catch (Exception e) {
                        System.out.println("Внезапная ошибка при закрытии FileChannel!");
                    }
                }

                @Override
                public void failed(Throwable e, ByteBuffer attachment) { System.out.println("Ошибка чтения!"); }
            });

            // Делаем что-то другое пока читаются данные из файла (полезная программа)

        } catch (Exception e) {
            System.out.println("Внезапная ошибка!");
        }
    }
}
```

Рисунок 2 - чтение из файла с CompletionHandler (https://github.com/sub-myitmo/example/blob/main/src/AsyncFileReaderHandler.java)

```java
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousFileChannel;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.concurrent.Future;

public class AsyncFileWriter  {
    public static void main(String[] args) {
        try {
            // Указываем путь к файлу и параметр канала (запись и создание)
            AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(Paths.get( first: "file2.txt"), StandardOpenOption.WRITE, StandardOpenOption.CREATE);

            ByteBuffer buffer = ByteBuffer.wrap("Hello world from AsyncFileWriter!".getBytes());
            Future<Integer> result = fileChannel.write(buffer,  position: 0);

            // Делаем что-то другое пока данные записываются в файл (полезная программа)
            int k = 0;
            for (int i=0; i < 1000; i++){
                k++;
            }

            // Ждем завершения записи
            Integer bytesWritten = result.get();
            System.out.println("Записано байтов: " + bytesWritten);

            fileChannel.close();
        } catch (Exception e) {
            System.out.println("Внезапная ошибка!");
        }
    }
}
```

Рисунок 3 - запись в файл с Future (https://github.com/sub-myitmo/example/blob/main/src/AsyncFileWriter.java)

```java
public class AsyncFileWriterHandler {
    public static void main(String[] args) {

        try {
            // Открытие асинхронного файла для записи
            AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(Paths.get( first: "file2.txt"), StandardOpenOption.WRITE, StandardOpenOption.CREATE);
            ByteBuffer buffer = ByteBuffer.wrap("Hello world from AsyncFileWriterHandler!".getBytes());

            // Запуск асинхронной записи с использованием CompletionHandler
            fileChannel.write(buffer,  position: 0, buffer, new CompletionHandler<Integer, ByteBuffer>() {
                @Override
                public void completed(Integer result, ByteBuffer attachment) {
                    System.out.println("Байтов записано: " + result);
                    try {
                        fileChannel.close();
                    } catch (IOException e) {
                        System.out.println("Внезапная ошибка при закрытии FileChannel!");
                    }
                }

                @Override
                public void failed(Throwable exc, ByteBuffer attachment) {
                    System.out.println("Ошибка записи!");
                    try {
                        fileChannel.close();
                    } catch (IOException e) {
                        System.out.println("Внезапная ошибка при закрытии FileChannel!");
                    }
                }
            });

            // Делаем что-то другое пока данные записываются в файл (полезная программа)

        } catch (IOException e) {
            System.out.println("Внезапная ошибка!");
        }
    }
}
```

Рисунок 4 - запись в файл с Future (https://github.com/sub-myitmo/example/blob/main/src/AsyncFileWriterHandler.java)

AsynchronousSocketChannel

Главные характеристики:

1. Асинхронность:

AsynchronousSocketChannel позволяет выполнять операции ввода-вывода (I/O) асинхронно, что означает, что ваш поток не блокируется во время выполнения этих операций. Такие каналы поддерживают одновременное чтение и запись, хотя в любой момент времени может выполняться не более одной операции чтения и одной операции записи. Если поток инициирует операцию чтения до завершения предыдущей операции чтения, то будет выброшен ReadPendingException. Аналогичным образом, попытка инициировать операцию записи до завершения предыдущей записи приведет к выдаче ошибки WritePendingException.

2. Подключение к socket:

Соединение AsynchronousSocketChannel создаётся при подключении к сокету объекта AsynchronousServerSocketChannel. Невозможно создать асинхронный канал сокета для произвольного, существующего файла socket.

Основные методы:

1. connect(SocketAddress remote): Асинхронно подключается к удаленному адресу.
2. read(ByteBuffer dst): Читает байты из канала в заданное положение в буфере.
3. write(ByteBuffer src): Записывает байты в канал из заданного положения в буфере.
4. close(): Закрывает канал.

Можно выделить следующие преимущества:

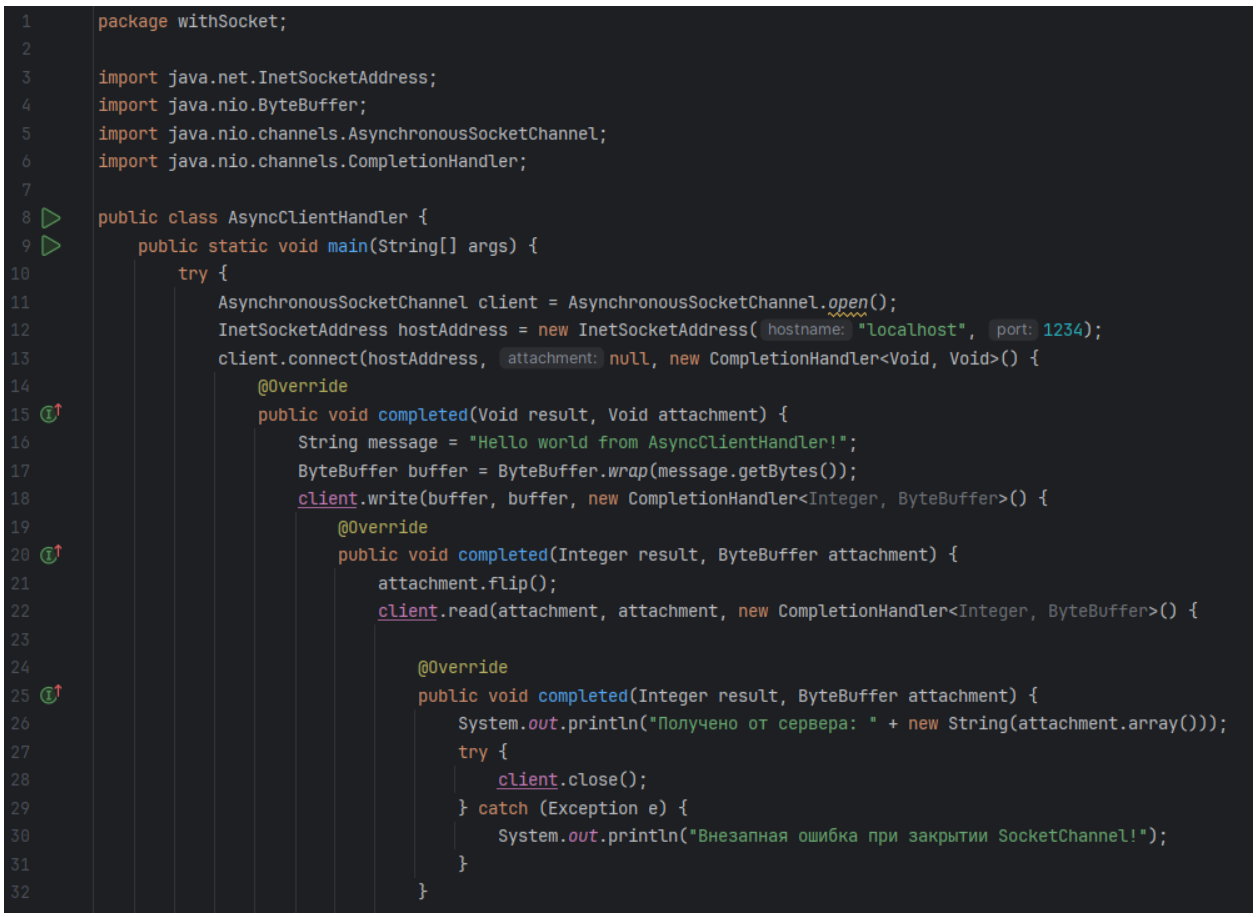
1. Повышенная производительность - асинхронные операции позволяют другим частям программы продолжать выполняться, что может повысить общую производительность.
2. Гибкость - возможность использования как Future, так и CompletionHandler для обработки завершения операций.
3. Удобство для обработки большого количества соединений.

Примеры использования (с Future и CompletionHandler):

Ниже представлены листинги программ, использующих AsynchronousSocketChannel с Future или CompletionHandler (клиентская и серверная части). Также добавлена ссылка на исходный код на https://github.com/sub-myitmo/example.

```java
package withSocket;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousSocketChannel;
import java.util.concurrent.Future;

public class AsyncClient {
    public static void main(String[] args) {
        try {
            AsynchronousSocketChannel client = AsynchronousSocketChannel.open();
            InetSocketAddress hostAddress = new InetSocketAddress( hostname: "localhost",  port: 1234);
            Future<Void> future = client.connect(hostAddress);
            future.get();  // Ожидаем подключения

            String message = "Hello world from AsyncClient!";
            ByteBuffer buffer = ByteBuffer.wrap(message.getBytes());
            Future<Integer> writeResult = client.write(buffer);
            writeResult.get();  // Ожидаем завершения записи

            buffer.flip();
            Future<Integer> readResult = client.read(buffer);
            readResult.get();  // Ожидаем завершения чтения

            System.out.println("Получено от сервера: " + new String(buffer.array()));

            client.close();
        } catch (Exception e) {
            System.out.println("Внезапная ошибка!");
        }
    }
}
```

Рисунок 5 – клиентская часть с Future (https://github.com/sub-myitmo/example/blob/main/src/withSocket/AsyncClient.java)



```java
package withSocket;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;

public class AsyncClientHandler {
    public static void main(String[] args) {
        try {
            AsynchronousSocketChannel client = AsynchronousSocketChannel.open();
            InetSocketAddress hostAddress = new InetSocketAddress( hostname: "localhost",  port: 1234);
            client.connect(hostAddress,  attachment: null, new CompletionHandler<Void, Void>() {
                @Override
                public void completed(Void result, Void attachment) {
                    String message = "Hello world from AsyncClientHandler!";
                    ByteBuffer buffer = ByteBuffer.wrap(message.getBytes());
                    client.write(buffer, buffer, new CompletionHandler<Integer, ByteBuffer>() {
                        @Override
                        public void completed(Integer result, ByteBuffer attachment) {
                            attachment.flip();
                            client.read(attachment, attachment, new CompletionHandler<Integer, ByteBuffer>() {

                                @Override
                                public void completed(Integer result, ByteBuffer attachment) {
                                    System.out.println("Получено от сервера: " + new String(attachment.array()));
                                    try {
                                        client.close();
                                    } catch (Exception e) {
                                        System.out.println("Внезапная ошибка при закрытии SocketChannel!");
                                    }
                                }
```

Рисунок 6 – клиент (часть 1) с CompletionHandler (https://github.com/sub-myitmo/example/blob/main/src/withSocket/AsyncClientHanler.java)

```java
                                @Override
                                public void failed(Throwable e, ByteBuffer attachment) {
                                    System.out.println("Ошибка чтения!");
                                }
                            });
                        }

                        @Override
                        public void failed(Throwable e, ByteBuffer attachment) {
                            System.out.println("Ошибка записи!");
                        }
                    });
                }

                @Override
                public void failed(Throwable exc, Void attachment) { System.out.println("Ошибка соединения!"); }
            });

            // Делаем что-то другое пока выполняются операции при подключении (полезная программа)

        } catch (Exception e) {
            System.out.println("Внезапная ошибка!");
        }
    }
}
```

Рисунок 7 – клиент (часть 2) с CompletionHandler (https://github.com/sub-myitmo/example/blob/main/src/withSocket/AsyncClientHanler.java)

```java
package withSocket;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousServerSocketChannel;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;

public class AsyncServer {
    public static void main(String[] args) {
        try {
            final AsynchronousServerSocketChannel serverChannel = AsynchronousServerSocketChannel.open();
            InetSocketAddress hostAddress = new InetSocketAddress( hostname: "localhost",  port: 1234);
            serverChannel.bind(hostAddress);

            System.out.println("Порт прослушивания: " + hostAddress);

            serverChannel.accept( attachment: null, new CompletionHandler<AsynchronousSocketChannel, Void>() {
                @Override
                public void completed(AsynchronousSocketChannel result, Void attachment) {
                    serverChannel.accept( attachment: null,  handler: this);  // Accept the next connection
                    handleClient(result);
                }

                @Override
                public void failed(Throwable exc, Void attachment) {
                    System.out.println("Не удалось установить соединение!");
                }
            });

            // Чтобы сервер не завершился немедленно
            while (true) {
                Thread.sleep( millis: 1000);
            }
        } catch (Exception e) {
            System.out.println("Внезапная ошибка!");
        }
    }
```

Рисунок 8 – сервер (часть 1) (https://github.com/sub-myitmo/example/blob/main/src/withSocket/AsyncServer.java)

```java
    1 usage
    private static void handleClient(AsynchronousSocketChannel clientChannel) {
        ByteBuffer buffer = ByteBuffer.allocate( capacity: 4096);
        clientChannel.read(buffer, buffer, new CompletionHandler<Integer, ByteBuffer>() {
            @Override
            public void completed(Integer result, ByteBuffer attachment) {
                attachment.flip();
                String message = new String(attachment.array()).trim();
                System.out.println("Получено от клиента: " + message);

                // Отправить сообщение обратно клиенту
                clientChannel.write(ByteBuffer.wrap(("Сервер принял ваше сообщение").getBytes()));
            }

            @Override
            public void failed(Throwable exc, ByteBuffer attachment) {
                System.out.println("Ошибка при чтении от клиента!");
            }
        });
    }
}
```

Рисунок 9 – сервер (часть 2) (https://github.com/sub-myitmo/example/blob/main/src/withSocket/AsyncServer.java)

Заключение

После рассмотрения преимуществ каждого классов и их реализаций можно сказать, что AsynchronousFileChannel следует использовать для асинхронной работы с файлами, а AsynchronousSocketChannel - для асинхронного взаимодействия с клиентами, то есть чтобы обрабатывать большое количество соединений одновременно за счет неблокирующих операций, тем самым делая его подходящим для масштабируемых серверных приложений.