Java.nio   
How to build a simple non-blocking server in Java?

* [Building non-blocking server in Java](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#building-non-blocking-server-in-java)
  + [What is non-blocking server?](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#what-is-non-blocking-server)
  + [java.nio](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#javanio)
  + [Bufor](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#bufor)
  + [Channel](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#channel)
  + [Selector](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#selector)
  + [Create NIO Server](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#create-nio-server)
  + [Create Client](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#create-client)
  + [Connecting client to our server](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#connecting-client-to-our-server)
  + [Summary](https://www.devdiaries.net/blog/java.nio-How-To-Build-a-non-blocking-server-in-java/#summary)

Building non-blocking server in Java

Some time ago (after Spring 5 release with WebFlux) I started getting interested in non-blocking http servers (Java reactive frameworks are based on this). That’s why I decided to write a post in which I will show how to create a non-blocking server and a client. We will send messages from the client to the server and display them.

Let’s start!

What is non-blocking server?

In the traditional approach, the server listens in the loop for any traffic on a given port. As soon as a new request appears, it delegates the request to previously created thread pool. This approach has some disadvantages. Firstly, the number of concurrently served clients have to be at most equal to the size of the thread pool. Moreover, if any client has weak internet connection - then the thread assigned to his request wastes most of the time waiting for more bits.

In non-blocking approach - one thread can handle multiple queries at a time. How? Thanks to the non-blocking IO implemented in **java.nio.package**.

java.nio

Java New IO (nio) was created in JDK 1.4 to allow all programmers to implement very fast input/output without having to deal with custom native code.

It was built based on three main functionalities: **buffers, channels and selectors**.

Buffer

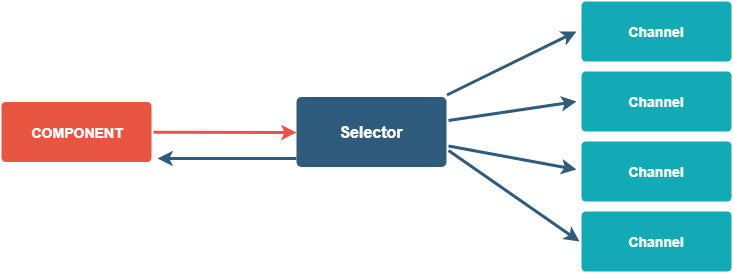
Buffer is a block of memory used to temporarily store data while it is being moved from one place to another.

Channel

Channel represents a connection to an objects that are capable of performing I/O operations, such as files and sockets. It uses buffers from which it reads the data to send and writes received information.

Selector

Selector is one of Java NIO class. The priciple of selector is very simple. After creation, we have to register in selector all the channels that we want to listen to. As a result of this operation, each channel is assigned with **selectionKey**. SelectionKey is an object that identyfying channel and contains information about channel’ status (e.g readiness to accept request). Each key holds information about who is making the request and what type of the request is. This is, each instance of Selector can monitor more socket channels and thus more connections. When something happens on the channel, the selector informs the application to process the request.



Create NIO Server

Let’s code! We will create our non-blocking server and client. Server will accept connections on port 8089 on localhost. We set it by using ServerSocket’s **bind()** method. To make the server non-blocking we will set ServerSocketchannel’s **configureBlocking()** method to **false**. Take a look at the following implementation:

**public** **class** **NonBlockingServer** **{**

**private** **static** Selector selector **=** **null;**

**public** **static** **void** **main(**String**[]** args**)** **{**

**try** **{**

selector **=** Selector**.**open**();**

*// We have to set connection host, port and non-blocking mode*

ServerSocketChannel socket **=** ServerSocketChannel**.**open**();**

ServerSocket serverSocket **=** socket**.**socket**();**

serverSocket**.**bind**(new** InetSocketAddress**(**"localhost"**,** 8089**));**

socket**.**configureBlocking**(false);**

**int** ops **=** socket**.**validOps**();**

socket**.**register**(**selector**,** ops**,** **null);**

**while** **(true)** **{**

selector**.**select**();**

Set**<**SelectionKey**>** selectedKeys **=** selector**.**selectedKeys**();**

Iterator**<**SelectionKey**>** i **=** selectedKeys**.**iterator**();**

**while** **(**i**.**hasNext**())** **{**

SelectionKey key **=** i**.**next**();**

**if** **(**key**.**isAcceptable**())** **{**

*// New client has been accepted*

handleAccept**(**socket**,** key**);**

**}** **else** **if** **(**key**.**isReadable**())** **{**

*// We can run non-blocking operation READ on our client*

handleRead**(**key**);**

**}**

i**.**remove**();**

**}**

**}**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**private** **static** **void** **handleAccept(**ServerSocketChannel mySocket**,**

SelectionKey key**)** **throws** IOException **{**

System**.**out**.**println**(**"Connection Accepted..."**);**

*// Accept the connection and set non-blocking mode*

SocketChannel client **=** mySocket**.**accept**();**

client**.**configureBlocking**(false);**

*// Register that client is reading this channel*

client**.**register**(**selector**,** SelectionKey**.**OP\_READ**);**

**}**

**private** **static** **void** **handleRead(**SelectionKey key**)**

**throws** IOException **{**

System**.**out**.**println**(**"Reading..."**);**

*// create a ServerSocketChannel to read the request*

SocketChannel client **=** **(**SocketChannel**)** key**.**channel**();**

*// Create buffer to read data*

ByteBuffer buffer **=** ByteBuffer**.**allocate**(**1024**);**

client**.**read**(**buffer**);**

*// Parse data from buffer to String*

String data **=** **new** String**(**buffer**.**array**()).**trim**();**

**if** **(**data**.**length**()** **>** 0**)** **{**

System**.**out**.**println**(**"Received message: " **+** data**);**

**if** **(**data**.**equalsIgnoreCase**(**"exit"**))** **{**

client**.**close**();**

System**.**out**.**println**(**"Connection closed..."**);**

**}**

**}**

**}**

**}**

You certainly noticed two important methods in the main loop of our server: **isAcceptable()** - checking if client is requesting a connection and **isReadable()** - method to read data when client has prepared data. IsReadable()will read data from the channel and put it into buffer. Next, we will send data from buffer onto the screen.

Create Client

Our client is simple. We also use SocketChannel to connect to the channel and send messages in the buffer. At the end we close SocketChannel.

**public** **class** **ServerClient** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

**try** **{**

String**[]** messages **=** **{**"I like non-blocking servers"**,** "Hello non-blocking world!"**,** "One more message.."**,** "exit"**};**

System**.**out**.**println**(**"Starting client..."**);**

SocketChannel client **=** SocketChannel**.**open**(new** InetSocketAddress**(**"localhost"**,** 8089**));**

**for** **(**String msg **:** messages**)** **{**

System**.**out**.**println**(**"Prepared message: " **+** msg**);**

ByteBuffer buffer **=** ByteBuffer**.**allocate**(**1024**);**

buffer**.**put**(**msg**.**getBytes**());**

buffer**.**flip**();**

**int** bytesWritten **=** client**.**write**(**buffer**);**

System**.**out**.**println**(**String**.**format**(**"Sending Message: %s\nbufforBytes: %d"**,** msg**,** bytesWritten**));**

**}**

client**.**close**();**

System**.**out**.**println**(**"Client connection closed"**);**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

Connecting client to our server

We have a client and a server ready. Let’s run it! I wll start with the server (it must be ready when the client sends a message). Below you can see the result of client<->server communication.

**Client:**

Starting client**...**

Prepared message: I like non**-**blocking servers

Sending Message: I like non**-**blocking servers

bufforBytes: 27

Prepared message: Hello non**-**blocking world**!**

Sending Message: Hello non**-**blocking world**!**

bufforBytes: 25

Prepared message: One more message**..**

Sending Message: One more message**..**

bufforBytes: 18

Prepared message: exit

Sending Message: exit

bufforBytes: 4

Client connection closed

**Server:**

Connection Accepted**...**

Reading**...**

Received message: I like non**-**blocking servers

Reading**...**

Received message: Hello non**-**blocking world**!**

Reading**...**

Received message: One more message**..**

Reading**...**

Received message: exit

Connection closed**...**