# Setting up Asynchronous Operations with NIO

**José Paumard**
PHD, JAVA CHAMPION, JAVA ROCK STAR
@JosePaumard https://github.com/JosePaumard

## Agenda

Introducing asynchronous operations

Setting up the problem: sync vs async

Selectors, socket channels

Using selectors to set up asynchronous network operations

## Introducing Asynchronous Operations

## Synchronous Read

When we call the read() method

- on a Reader from Java I/O

- or on a Channel from Java NIO

The method returns when the data is read

This is a synchronous call

Also called a blocking call

## Asynchronous Read

Asynchronous, or non-blocking behaves differently

We trigger a read operation

When the data is there, we are called back

So we can do something else in the meantime

## What Is the Difference?

Suppose our application needs to read data from many sources

- with synchronous read, each read operation is conducted in its own thread

- with asynchronous reads, a single thread can handle many read operations

## Less Thread Is Better

A thread is a system resource

It can be expensive to set up

And it can be expensive for a CPU to switch from one thread to another

## Understanding Selectors

The Selector is the entry object to set an asynchronous system

1) create a channel

2) configure it to be non-blocking

3) register this channel with this selector

4) get the registration key

Then the Channel will fire events

There are four events defined:

- READ, WRITE: the channel is ready for reading or writing

In the case of socket channels:

- CONNECT: a connection was established

- ACCEPT: a connection was accepted

The registration is configured to listen to certain events

It can also be done through the key

Once the system is set up

1) we call select() on the selector

2) from the selector, we get the keys that have events to be processed

## Setting up a Asynchronous Network Reader

```java
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.configureBlocking(false);

ServerSocket serverSocket = serverSocketChannel.socket();
serverSocket.bind(new InetSocketAddress(12345));
```

Let us first create a server socket channel and configure this channel to be non-blocking

Then create a server socket from this channel and bind it to a port

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

SelectionKey key =
    serverSocketChannel.register(selector,
                                 SelectionKey.OP_ACCEPT);
```

Then we need to create a selector object

And register our selector with the ACCEPT event

The returned key can be stored somewhere for future use

Unregister, check for validity, etc…

```java
int n = selector.select();
Set<SelectionKey> selectedKeys = selector.selectedKeys();


for(SelectionKey key: selectedKeys) {

    // do something with the corresponding channel

}
```

Our server socket channel is listening to incoming requests

The select() call is blocking until events are available, n is the number of keys with available events

The keys that got events are returned in a set

Each key can be processed in a loop

This is the first step: an incoming request has been received

1) we need to make sure that the key corresponds to a connection request

2) then we need to set up a connection with a socket, and listen to the events on this socket

```
if ((key.readyOps() & SelectionKey.OP_ACCEPT) == SelectionKey.OP_ACCEPT) {

   ServerSocketChannel channel = (ServerSocketChannel)key.channel();

   SocketChannel socketChannel = channel.accept();
   socketChannel.configureBlocking(false);
   socketChannel.register(selector, SelectionKey.OP_READ);
}
```

We make sure that the event is a connection request

The key object knows the channel that has fired the event

We then create a client socket channel, non-blocking

And register it to the selector

At this point our selector is registered with two channels, and will get the events from both channels

1) the first channel listens to incoming requests

2) the second channel has been created to handle the communication with a given client

So we need to add the handling of the READ event for this second channel

```java
else if ((key.readyOps() & SelectionKey.OP_READ) == SelectionKey.OP_READ) {

    SocketChannel channel = (SocketChannel)key.channel();
    channel.accept(byteBuffer);
    // do something with the buffer
    byteBuffer.clear();
    selectedKeys.remove(key);
    key.cancel();
    channel.close();
}
```

This is the READ handling part

First, get the corresponding channel and read it in a byte buffer

Then do something with this byte buffer

And clean up the resources: the buffer, the key and the channel

## Demo

Let us see some code!

Let replay this asynchronous server live and with some data

# Module Wrap Up

What did you learn?

The way Java NIO brings asynchronous operations to the platform

How to deal with Selectors and to set up an asynchronous server