Non blocking IO with Java NIO

Tam-CO

Agenda

1. Non blocking IO concept
2. Java NIO
3. Drawback of blocking IO
4. Non blocking comes to rescue
5. Code demo

Non blocking IO concept

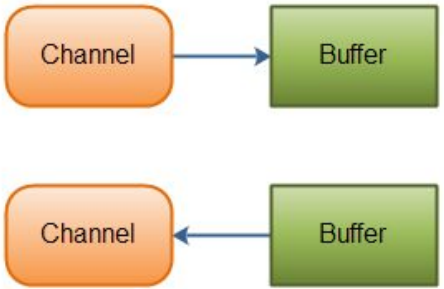
- Traditional IO (Blocking IO/Synchronous IO)
  - Start the access and wait for it to complete
  - Block the progress of program while communication is in progress
  - Leaving system resources idle
- Non blocking IO (Asynchronous IO)
  - A form of IO processing that permits other processing to continue before the **transmission** has finished

Java NIO

- NIO stands for **New IO** (not non-blocking IO), from Java 1.4
- An alternative to the standard Java IO and Java Networking API, offers a different way to working with IO
- NIO enables you to do non-blocking IO
- Some concepts in NIO
  - Channel
  - Buffer
  - Selector

Java NIO (cnt)

- Channel and Buffer
  - In standard IO you work with **streams**, read and write data to streams directly
  - In NIO, you work with channel and buffer. Data is always read from a channel into a buffer, or written from a buffer to a channel



Java NIO (cnt)

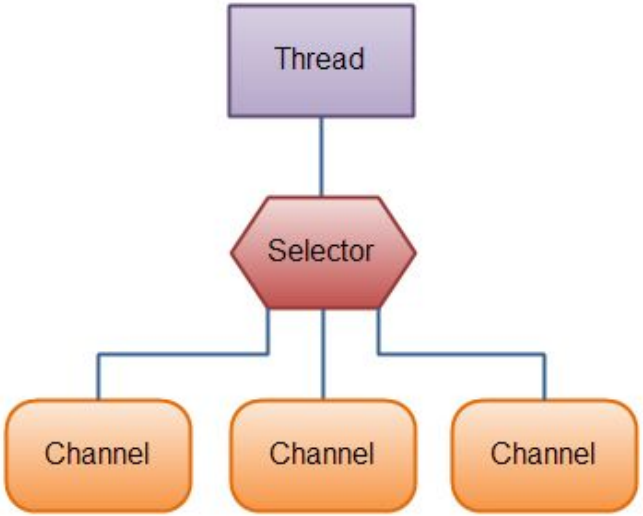
- Channel and Buffer
  - Some channels in NIO
    - FileChannel, DatagramChannel
    - SocketChannel, ServerSocketChannel
  - Some buffers in NIO
    - ByteBuffer, CharBuffer, DoubleBuffer
    - FloatBuffer, IntBuffer, LongBuffer, ShortBuffer

Java NIO (cnt)

- Selector
  - An object that can monitor muliple channels for events like **connection opened**, **data arrived**…
  - *someChannel.register(selector, selectionKeyEvent);*
  - Allow single thread can handle multiple channels
    - This is handy if your application has many connections with low traffic

Java NIO (cnt)

- Selector



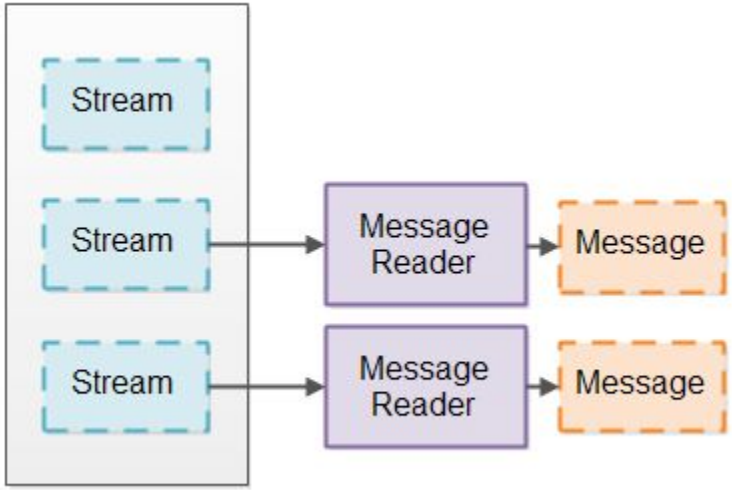
Java NIO (cnt)

- Enable non blocking mode
  - *someChannel.configureBlocking(false);*

Drawback of blocking IO

- Scenario
  - We need to read from multiple streams
- With single thread we cannot attempt to read from other stream if there is no data to read from the current stream
- So we need separate thread for each stream that needs to be read
- What happens if we have so many streams (e.g server connections) to read → We need so many threads. Threads are expensive
- How about thread pool? We may encounter reponsiveness problem

Drawback of blocking IO



Non blocking come to rescue

- With non-blocking mode we can use 1 thread to read/write from/to multiple channels
- But it has drawback itself
  - What happens if we have so many connections with high traffic?
  - 1M connections with high traffic that require 1M buffer?

Thank you