Intro

An event-driven asynchronous network application framework

Architecture

Several important concepts

- A Selector
- A task queue (mpsc_queue: Lock-free multiple producers and single consumer)
- A delayed task queue (delay_queue: A priority queue with a binary heap structure. Complexity: O(log n)).
- EventLoop is bound to a Thread that avoids the thread contention in the pipeline.

Boss: the mainReactor and **Worker: the subReactor**

- **The Boss and Worker share the EventLoop code logic**. **The Boss handles the accept event, and the Worker handles read and write events.**
- After the Boss listens to and accepts the connection (channel), it hands the channel to the Worker by polling. The Worker is responsible for processing the subsequent I/O events of the channel, such as read and write.
- In the case of no multi-port binding, only one EventLoop needs to be included in BossEventLoopGroup, and only one can be used.
- WorkerEventLoopGroup generally contains multiple EventLoop, and the number is generally two times the CPU core number. The most important thing is to find the best value according to the scenario.
- Channels are divided into two types: ServerChannel and Channel. ServerChannel corresponds to ServerSocketChannel, and Channel corresponds to a network connection.

[Log entry 21.09.2024]

interface EventLoopGroup extends EventExecutorGroup = Special EventExecutorGroup which allows registering Channels that get processed for later selection during the event looping.

The EventLoopGroup is a special type of EventExecutorGroup because it allows registration of IO Channels (io.netty.channel.Channel).

interface EventLoop extends EventLoopGroup = Will handle all the I/O operations for a (io.netty.channel.Channel) once registered.

class DefaultEventLoop = simplest implementation of actual event loop (void run() method is where the actual loop runs)

class NioEventLoop = event loop “group” that works with java.nio.channels.Channels and java.nio.channels.Selector (void run() method is where the actual loop runs)

So there are a bunch of EventExecutors that are grouped together - EventExecutorGroup. The EventExecutorGroup instances contain EventExecutors and provide an API to somehow use and manage this EventExecutor group.

[Log entry 22.09.2024]

Event loops are also event task executors so each event loop has two input sources (channels and task queues).

In NioEventLoop's case - priority is given to queued tasks over channels, for some reason. If during an event loop iteration there are pending tasks, selector's select() will not be called - the event loop will handle all the pending tasks first and during the next event loop iteration, if there are no pending tasks - the EL will handle channels.

[Log entry 23.09.2024]

class EpollEventLoop = uses epoll under the covers. Only works on Linux

public void run() = method that runs actual event loop.

private long epollWait(long deadlineNanos) = method to wait for IO events.

private processReady(EpollEventArray events, int ready) = method that handles IO events gathered by epollWait() method. Socket Channels are handled here. This method is called by run() method

EPOLLIN - epoll's read operation code
EPOLLOUT = epoll's write operation code

References:

- https://www.alibabacloud.com/blog/598081