Netty

博客分类：

- Thinking

Netty是什么？

本质：JBoss做的一个Jar包

目的：快速开发高性能、高可靠性的网络服务器和客户端程序

优点：提供异步的、事件驱动的网络应用程序框架和工具

通俗的说：一个好使的处理Socket的东东

如果没有Netty？

远古：java.net + java.io

近代：java.nio

其他：Mina，Grizzly

为什么不是Mina？

1、都是Trustin Lee的作品，Netty更晚；

2、Mina将内核和一些特性的联系过于紧密，使得用户在不需要这些特性的时候无法脱离，相比下性能会有所下降，Netty解决了这个设计问题；

3、Netty的文档更清晰，很多Mina的特性在Netty里都有；

4、Netty更新周期更短，新版本的发布比较快；

5、它们的架构差别不大，Mina靠apache生存，而Netty靠jboss，和jboss的结合度非常高，Netty有对google protocal buf的支持，有更完整的ioc容器支持(spring,guice,jbossmc和osgi)；

6、Netty比Mina使用起来更简单，Netty里你可以自定义的处理upstream events 或/和 downstream events，可以使用decoder和encoder来解码和编码发送内容；

7、Netty和Mina在处理UDP时有一些不同，Netty将UDP无连接的特性暴露出来；而Mina对UDP进行了高级层次的抽象，可以把UDP当成"面向连接"的协议，而要Netty做到这一点比较困难。

Netty的特性

设计

统一的API，适用于不同的协议（阻塞和非阻塞）

基于灵活、可扩展的事件驱动模型

高度可定制的线程模型

可靠的无连接数据Socket支持（UDP）

性能

更好的吞吐量，低延迟

更省资源

尽量减少不必要的内存拷贝

安全

完整的SSL/TLS和STARTTLS的支持

能在Applet与Android的限制环境运行良好

健壮性

不再因过快、过慢或超负载连接导致OutOfMemoryError

不再有在高速网络环境下NIO读写频率不一致的问题

易用

完善的JavaDoc，用户指南和样例

简洁简单

仅信赖于JDK1.5

看例子吧！

Server端：

Java代码

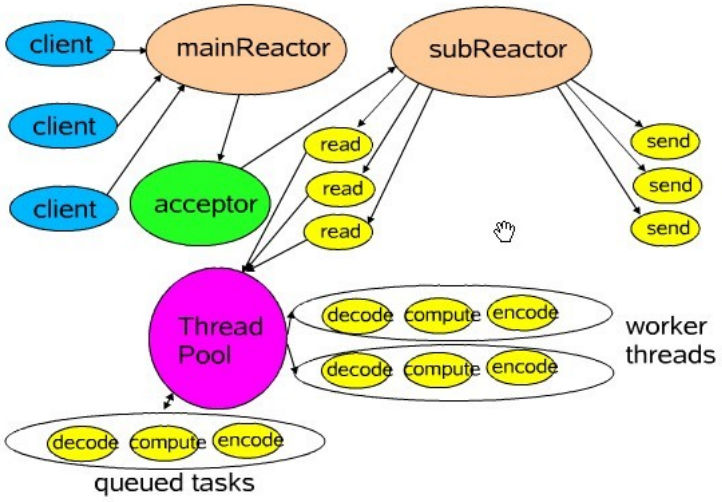
```java
package me.hello.netty;

import org.jboss.netty.bootstrap.ServerBootstrap;
import org.jboss.netty.channel.*;
import org.jboss.netty.channel.socket.nio.NioServerSocketChannelFactory;
import org.jboss.netty.handler.codec.string.StringDecoder;
import org.jboss.netty.handler.codec.string.StringEncoder;

import java.net.InetSocketAddress;
import java.util.concurrent.Executors;

/**
 * God Bless You!
 * Author: Fangniude
 * Date: 2013-07-15
 */
public class NettyServer {
        public static void main(String[] args) {
                ServerBootstrap bootstrap = new ServerBootstrap(new NioServerSocketChannelFactory(Executors.newCachedThreadPool(), Executors.

                // Set up the default event pipeline.
                bootstrap.setPipelineFactory(new ChannelPipelineFactory() {
                        @Override
                        public ChannelPipeline getPipeline() throws Exception {
                                return Channels.pipeline(new StringDecoder(), new StringEncoder(), new ServerHandler());
                        }
                });

                // Bind and start to accept incoming connections.
                Channel bind = bootstrap.bind(new InetSocketAddress(8000));
                System.out.println("Server已经启动，监听端口: " + bind.getLocalAddress() + ", 等待客户端注册。。。");
        }

        private static class ServerHandler extends SimpleChannelHandler {
                @Override
                public void messageReceived(ChannelHandlerContext ctx, MessageEvent e) throws Exception {
                        if (e.getMessage() instanceof String) {
                                String message = (String) e.getMessage();
                                System.out.println("Client发来:" + message);

                                e.getChannel().write("Server已收到刚发送的:" + message);

                                System.out.println("\n等待客户端输入。。。");
                        }

                        super.messageReceived(ctx, e);
                }

                @Override
                public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEvent e) throws Exception {
                        super.exceptionCaught(ctx, e);
                }

                @Override
                public void channelConnected(ChannelHandlerContext ctx, ChannelStateEvent e) throws Exception {
                        System.out.println("有一个客户端注册上来了。。。");
                        System.out.println("Client:" + e.getChannel().getRemoteAddress());
                        System.out.println("Server:" + e.getChannel().getLocalAddress());
                        System.out.println("\n等待客户端输入。。。");
                        super.channelConnected(ctx, e);
                }
        }
}
```

客户端：

Java代码

```java
package me.hello.netty;

import org.jboss.netty.bootstrap.ClientBootstrap;
import org.jboss.netty.channel.*;
import org.jboss.netty.channel.socket.nio.NioClientSocketChannelFactory;
import org.jboss.netty.handler.codec.string.StringDecoder;
import org.jboss.netty.handler.codec.string.StringEncoder;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.InetSocketAddress;
import java.util.concurrent.Executors;

/**
 * God Bless You!
 * Author: Fangniude
 * Date: 2013-07-15
 */
public class NettyClient {

	public static void main(String[] args) {
		// Configure the client.
		ClientBootstrap bootstrap = new ClientBootstrap(new NioClientSocketChannelFactory(Executors.newCachedThreadPool(), Executors.

		// Set up the default event pipeline.
		bootstrap.setPipelineFactory(new ChannelPipelineFactory() {
			@Override
			public ChannelPipeline getPipeline() throws Exception {
				return Channels.pipeline(new StringDecoder(), new StringEncoder(), new ClientHandler());
			}
		});

		// Start the connection attempt.
		ChannelFuture future = bootstrap.connect(new InetSocketAddress("localhost", 8000));

		// Wait until the connection is closed or the connection attempt fails.
		future.getChannel().getCloseFuture().awaitUninterruptibly();

		// Shut down thread pools to exit.
		bootstrap.releaseExternalResources();
	}

	private static class ClientHandler extends SimpleChannelHandler {
		private BufferedReader sin = new BufferedReader(new InputStreamReader(System.in));

		@Override
		public void messageReceived(ChannelHandlerContext ctx, MessageEvent e) throws Exception {
			if (e.getMessage() instanceof String) {
				String message = (String) e.getMessage();
				System.out.println(message);

				e.getChannel().write(sin.readLine());

				System.out.println("\n等待客户端输入。。。");
			}

			super.messageReceived(ctx, e);
		}

		@Override
		public void channelConnected(ChannelHandlerContext ctx, ChannelStateEvent e) throws Exception {
			System.out.println("已经与Server建立连接。。。。");
			System.out.println("\n请输入要发送的信息：");
			super.channelConnected(ctx, e);

			e.getChannel().write(sin.readLine());
		}
	}
}
```

Netty整体架构



Netty组件

ChannelFactory

Boss

Worker

Channel

ChannelEvent

Pipeline

ChannelContext

Handler

Sink

Server端核心类

NioServerSocketChannelFactory

NioServerBossPool

NioWorkerPool

NioServerBoss

NioWorker

NioServerSocketChannel

NioAcceptedSocketChannel

DefaultChannelPipeline

NioServerSocketPipelineSink

Channels

ChannelFactory

Channel工厂，很重要的类

保存启动的相关参数

NioServerSocketChannelFactory

NioClientSocketChannelFactory

NioDatagramChannelFactory

这是Nio的，还有Oio和Local的

SelectorPool

Selector的线程池

NioServerBossPool 默认线程数：1

NioClientBossPool 1

NioWorkerPool 2 * Processor

NioDatagramWorkerPool

Selector

选择器，很核心的组件

NioServerBoss

NioClientBoss

NioWorker

NioDatagramWorker

Channel

通道

NioServerSocketChannel

NioClientSocketChannel

NioAcceptedSocketChannel

NioDatagramChannel

Sink

负责和底层的交互

如bind，Write，Close等

NioServerSocketPipelineSink

NioClientSocketPipelineSink

NioDatagramPipelineSink

Pipeline

负责维护所有的Handler

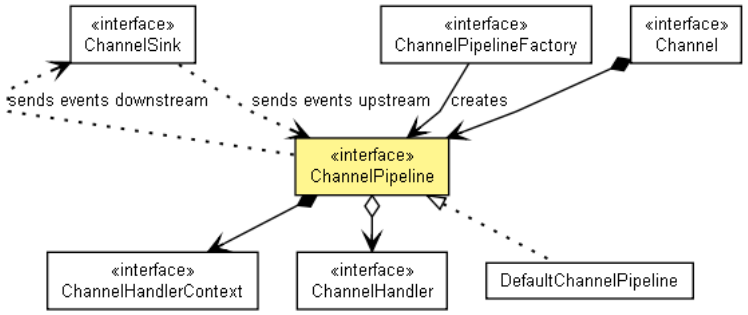
ChannelContext

一个Channel一个，是Handler和Pipeline的中间件

Handler

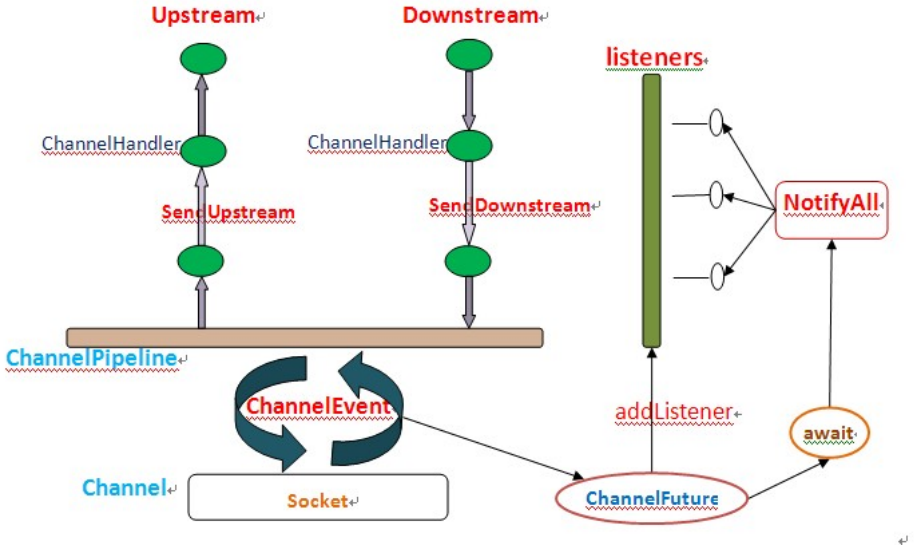
对Channel事件的处理器

ChannelPipeline

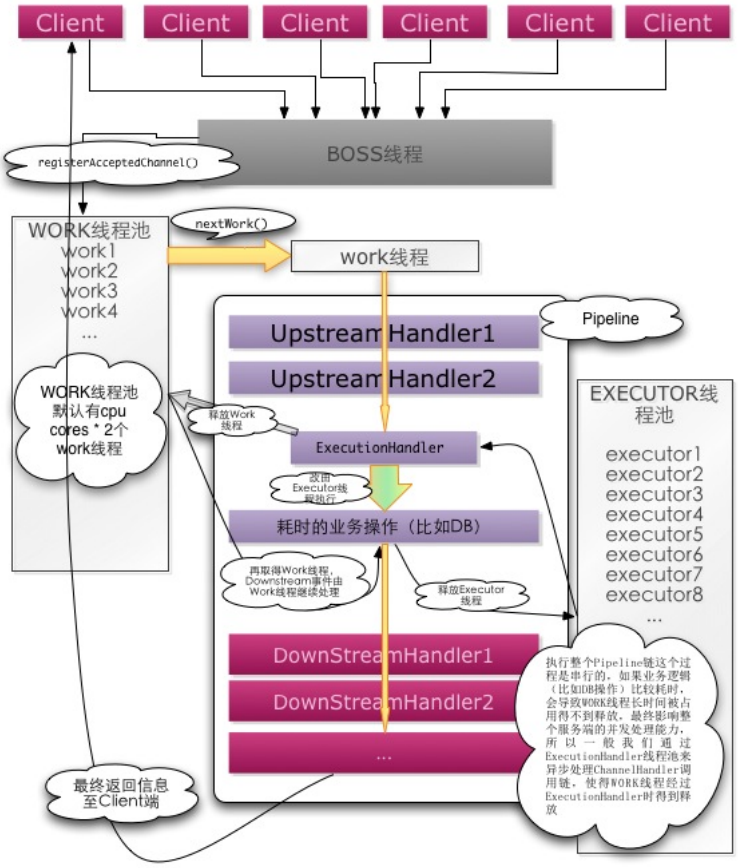



```
                                      I/O Request
                                    via Channel or
                                ChannelHandlerContext
                                         |
+----------------------------------------+---------------+
|                ChannelPipeline         |               |
|                                       \|/              |
|  +---------------------+  +-----------+------------+  |
|  | Upstream Handler  N |  | Downstream Handler  1  |  |
|  +----------+----------+  +-----------+------------+  |
|            /|\                         |               |
|             |                         \|/              |
|  +----------+----------+  +-----------+------------+  |
|  | Upstream Handler N-1 |  | Downstream Handler  2  |  |
|  +----------+----------+  +-----------+------------+  |
|            /|\                         .               |
|             .                          .               |
|     [ sendUpstream() ]        [ sendDownstream() ]     |
|     [ + INBOUND data ]        [ + OUTBOUND data  ]     |
|             .                          .               |
|             .                         \|/              |
|  +----------+----------+  +-----------+------------+  |
|  | Upstream Handler  2 |  | Downstream Handler M-1 |  |
|  +----------+----------+  +-----------+------------+  |
|            /|\                         |               |
|             |                         \|/              |
|  +----------+----------+  +-----------+------------+  |
|  | Upstream Handler  1 |  | Downstream Handler  M  |  |
|  +----------+----------+  +-----------+------------+  |
|            /|\                         |               |
+-------------+--------------------------+---------------+
              |                         \|/
+-------------+--------------------------+---------------+
|  Netty Internal I/O Threads (Transport Implementation) |
+--------------------------------------------------------+
```

优秀的设计----事件驱动



优秀的设计----线程模型



注意事项

解码时的Position

Channel的关闭

更多Handler

Channel的关闭

用完的Channel，可以直接关闭；

1、ChannelFuture加Listener

2、writeComplete

一段时间没用，也可以关闭

TimeoutHandler