



深入浅出Netty

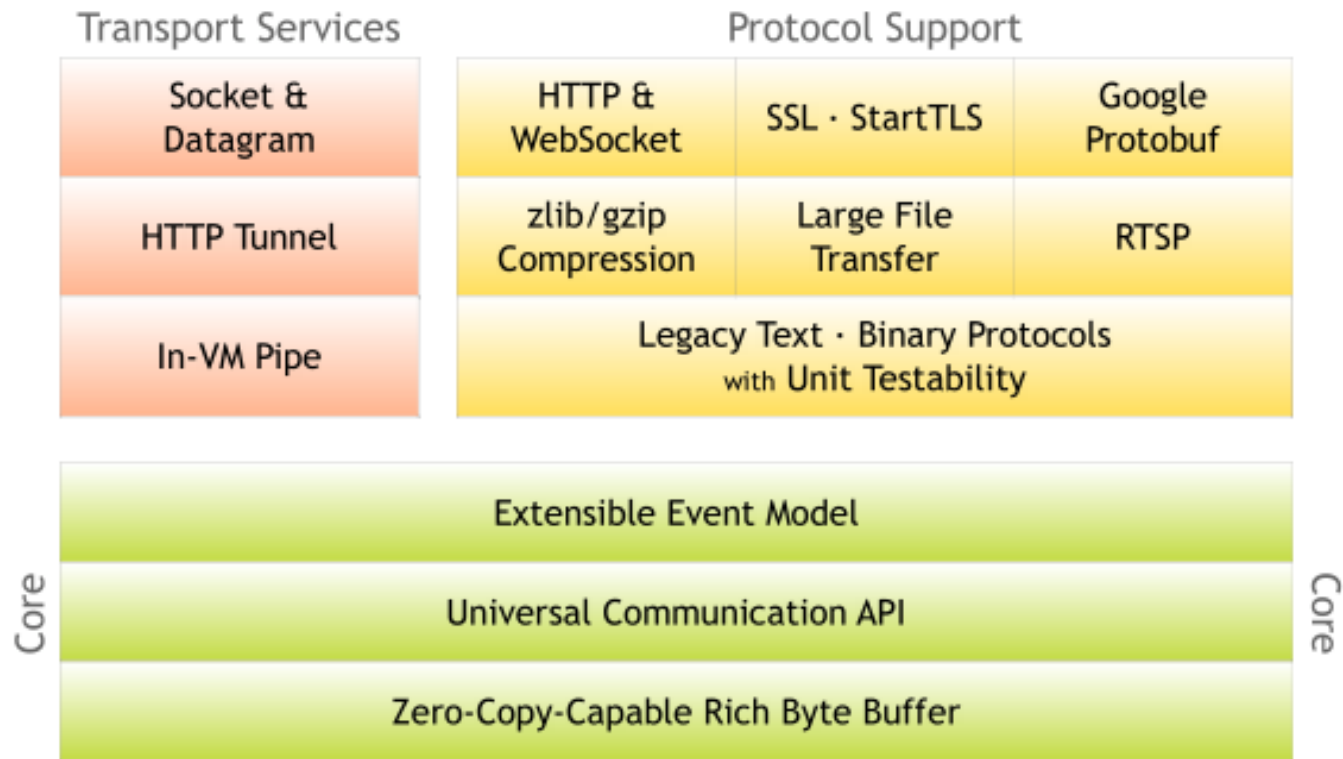
雷腾 L.T - leiteng@taobao.com
搜索算法与技术 – 搜索应用团队

Netty是什么？

Netty 提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序

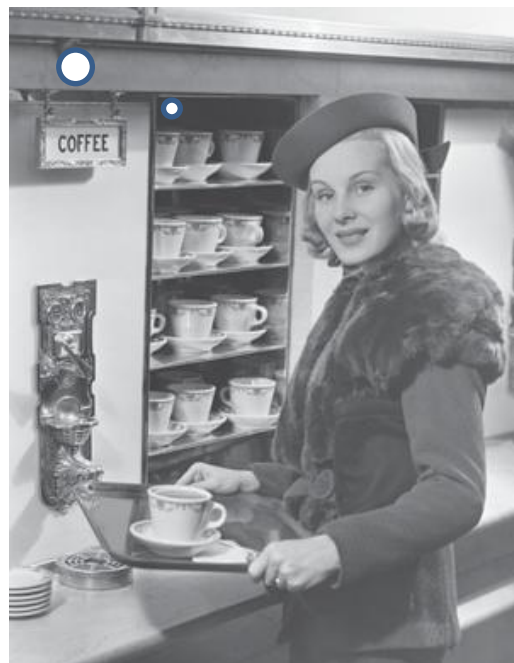


Netty的架构



Netty的特性

Netty 有些什
么特性呢？



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



Netty的特性

★ 安全

- 完整的SSL/ TLS和STARTTLS的支持
- 能在Applet与谷歌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Java doc , 用户指南和样例
- 简洁简单
- 仅依赖于JDK1.5



深入浅出Netty

But how to use it ?



Hello World in Netty

★ HelloWorldServer

```
ServerBootstrap bootstrap = new ServerBootstrap(new  
    NioServerSocketChannelFactory(  
        Executors.newCachedThreadPool(),  
        Executors.newCachedThreadPool()));
```

```
bootstrap.setPipelineFactory(new ChannelPipelineFactory() {  
    public ChannelPipeline getPipeline() {  
        ChannelPipeline pipeline = Channels.pipeline();  
        pipeline.addLast("decoder", new StringDecoder();  
        pipeline.addLast("encoder", new StringEncoder();  
        pipeline.addLast("handler", new HelloWorldServerHandler();  
        return pipeline;  
    }  
});
```

```
bootstrap.bind(new InetSocketAddress(8080));
```



Hello World in Netty

★ HelloWorldServerHandler

```
public void channelConnected(ChannelHandlerContext ctx,  
    ChannelStateEvent e) throws Exception {  
    e.getChannel().write("Hello, World");  
}
```

```
public void exceptionCaught(ChannelHandlerContext ctx,  
    ExceptionEvent e) {  
    logger.log(Level.WARNING, "Unexpected exception from  
    downstream.", e.getCause());  
    e.getChannel().close();  
}
```



Hello World in Netty

★ HelloWorldClient

```
ClientBootstrap bootstrap = new ClientBootstrap(new NioClientSocketChannelFactory(  
    Executors.newCachedThreadPool(), Executors.newCachedThreadPool()));
```

```
bootstrap.setPipelineFactory(new ChannelPipelineFactory() {  
    public ChannelPipeline getPipeline() {  
        ChannelPipeline pipeline = pipeline();  
        pipeline.addLast("decoder", new StringDecoder());  
        pipeline.addLast("encoder", new StringEncoder());  
        pipeline.addLast("handler", new HelloWorldClientHandler());  
        return pipeline;  
    }  
});
```

```
ChannelFuture future = bootstrap.connect(new InetSocketAddress("localhost", 8080));
```

```
future.getChannel().getCloseFuture().awaitUninterruptibly();
```

```
bootstrap.releaseExternalResources();
```



Hello World in Netty

★ HelloWorldClientHandler

```
public void messageReceived(ChannelHandlerContext ctx,  
    MessageEvent e) {
```

```
    String message = (String) e.getMessage();
```

```
    System.out.println(message);
```

```
    e.getChannel().close();
```

```
}
```

```
public void exceptionCaught(ChannelHandlerContext ctx,  
    ExceptionEvent e) {
```

```
    logger.log(Level.WARNING, "Unexpected exception from  
        downstream.", e.getCause());
```

```
    e.getChannel().close();
```

```
}
```



深入浅出Netty

刨根问底的勇气？



Netty源码分析

★ org.jboss.netty.bootstrap

- ★ Bootstrap : ChannelFactory , ChannelPipeline , ChannelPipelineFactory
 - ★ 初始化channel的辅助类
 - ★ 为具体的子类提供公共数据结构
- ★ ServerBootstrap: bind()
 - ★ 创建服务器端channel的辅助类
 - ★ 接收connection请求
- ★ ClientBootstrap : connect()
 - ★ 创建客户端channel的辅助类
 - ★ 发起connection请求
- ★ ConnectionlessBootstrap : connect() , bind()
 - ★ 创建无连接传输channel的辅助类 (UDP)
 - ★ 包括Client 和Server



Netty源码分析

★ org.jboss.netty.buffer

取代nio 中的java.nio.ByteBuffer , 相比ByteBuffer

- ★ 可以根据需要自定义buffer type
- ★ 内置混合的buffer type, 以实现zero-copy
- ★ 提供类似StringBuffer的动态dynamic buffer
- ★ 不需要调用flip方法
- ★ 更快的性能

推荐使用ChannelBuffers的静态工厂创建ChannelBuffer



Netty源码分析

- ★ [org.jboss.netty.channel](#)
channel核心api，包括异步和事件驱动等各种传送接口
- ★ [org.jboss.netty.channel.group](#)
channel group，帮助用户维护channel列表
- ★ [org.jboss.netty.channel.local](#)
一种虚拟传输方式，允许同一个虚拟机上的两个部分可以互相通信
- ★ [org.jboss.netty.channel.socket](#)
TCP, UDP接口，继承了核心的channel API
- ★ [org.jboss.netty.channel.socket.nio](#)
基于nio的Socket channel实现
- ★ [org.jboss.netty.channel.socket.oio](#)
基于老io的Socket channel实现
- ★ [org.jboss.netty.channel.socket.http](#)
基于http的客户端和相应的server端的实现，工作在有防火墙的情况



Netty源码分析

- ★ [org.jboss.netty.container](#)
各种容器的兼容
- ★ [org.jboss.netty.container.microcontainer](#)
JBoss Microcontainer 集成接口
- ★ [org.jboss.netty.container.osgi](#)
OSGi framework集成接口
- ★ [org.jboss.netty.container.spring](#)
Spring framework集成接口



Netty源码分析

- ★ `org.jboss.netty.handler`
处理器
- ★ `org.jboss.netty.handler.codec`
编码解码器
- ★ `org.jboss.netty.handler.execution`
基于Executor的实现
- ★ `org.jboss.netty.handler.queue`
将event存入内部队列的处理
- ★ `org.jboss.netty.handler.ssl`
基于SSLEngine的SSL以及TLS实现
- ★ `org.jboss.netty.handler.stream`
异步写入大数据，不会产生outOfMemory 也不会花费很多内存
- ★ `org.jboss.netty.handler.timeout`
通过Timer来对读写超时或者闲置链接进行通知



Netty源码分析

- ★ `org.jboss.netty.handler.codec.base64` Base64 编码
- ★ `org.jboss.netty.handler.codec.compression` 压缩格式
- ★ `org.jboss.netty.handler.codec.embedder` 嵌入模式下编码和解码
- ★ `org.jboss.netty.handler.codec.frame` 评估流的数据的排列和内容
- ★ `org.jboss.netty.handler.codec.http.websocket` websocket编码解码
- ★ `org.jboss.netty.handler.codec.http` http的编码解码以及类型信息
- ★ `org.jboss.netty.handler.codec.oneone` 对象到对象编码解码
- ★ `org.jboss.netty.handler.codec.protobuf` Protocol Buffers的编码解码
- ★ `org.jboss.netty.handler.codec.replay` 在阻塞io中实现非阻塞解码
- ★ `org.jboss.netty.handler.codec.rtsp` RTSP的编码解码
- ★ `org.jboss.netty.handler.codec.serialization` 序列化对象到bytebuffer实现
- ★ `org.jboss.netty.handler.codec.string` 字符串编码解码，继承oneone

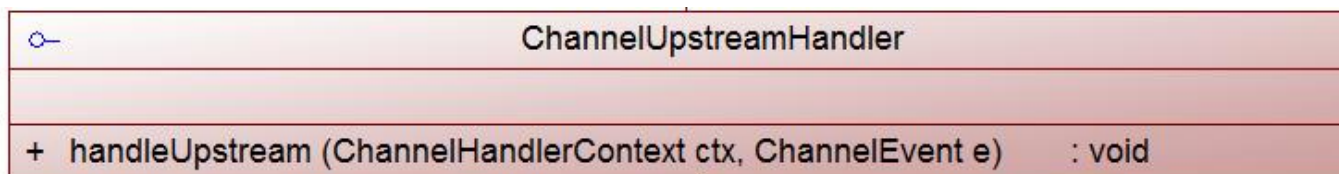


Netty源码分析

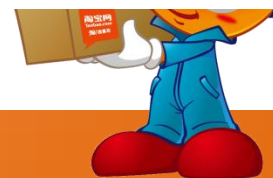
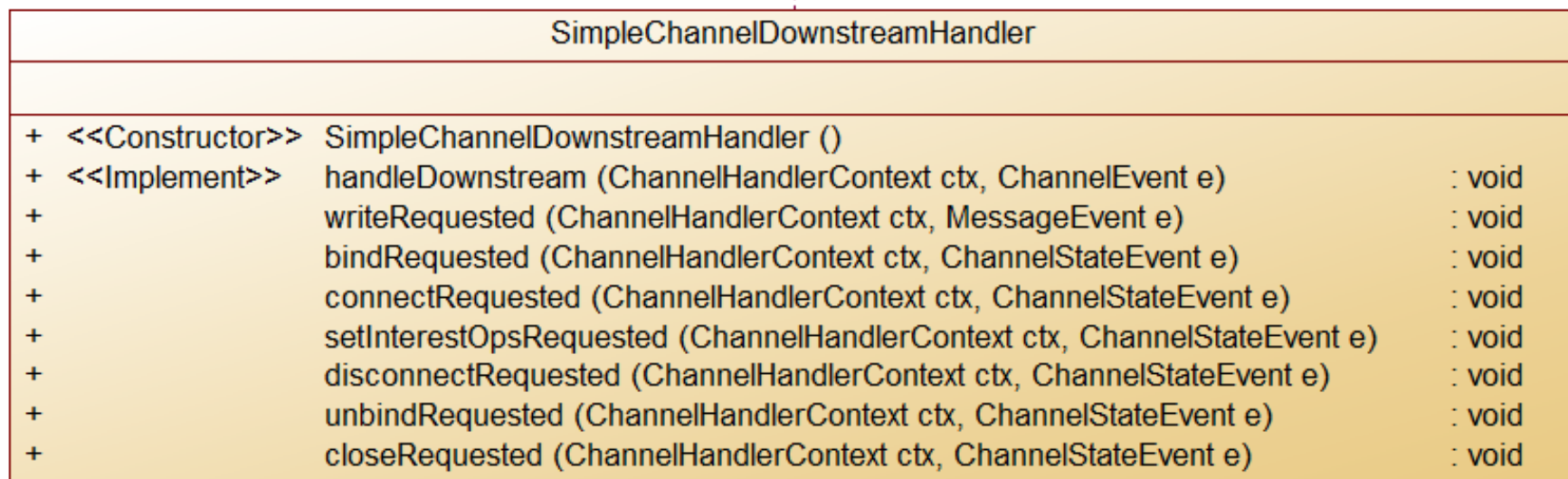
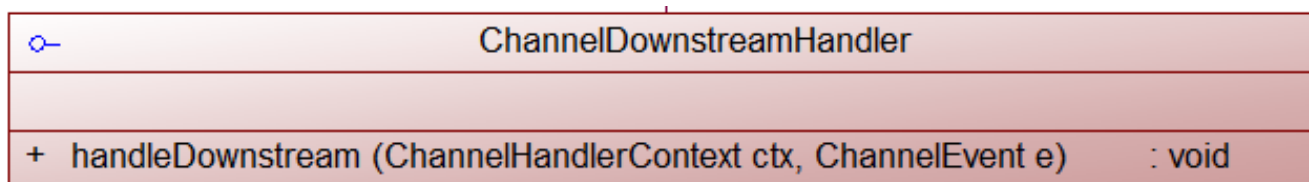
- ★ `org.jboss.netty.logging`
根据不同的log framework 实现的类
- ★ `org.jboss.netty.util`
netty util类
- ★ `org.jboss.netty.util.internal`
netty内部util类，不被外部使用



Netty事件驱动模型



Netty事件驱动模型



Netty Pipeline 流处理

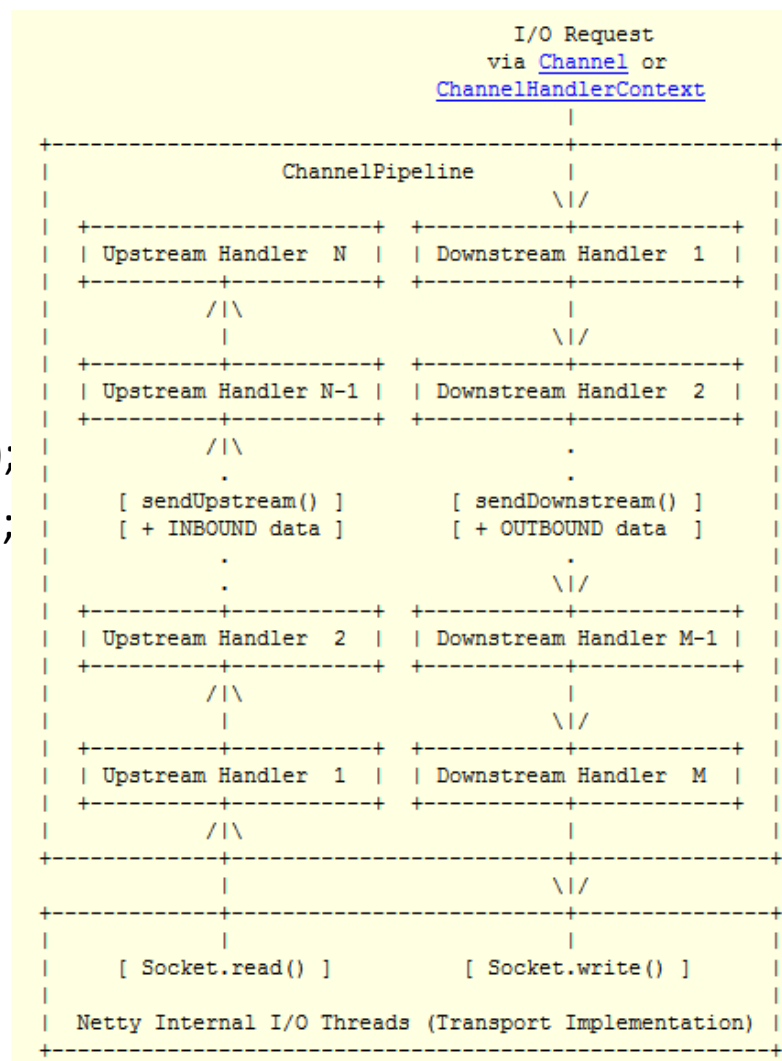
★ Upstream 接收请求

★ Downstream 发送请求

```
ChannelPipeline p = Channels.pipeline\(\);  
p.addLast("1", new UpstreamHandlerA());  
p.addLast("2", new UpstreamHandlerB());  
p.addLast("3", new DownstreamHandlerA());  
p.addLast("4", new DownstreamHandlerB());  
p.addLast("5", new UpstreamHandlerX());
```

Upstream: 1 → 2 → 5 顺序处理

Downstream: 4 → 3 逆序处理



Netty 通用通信API

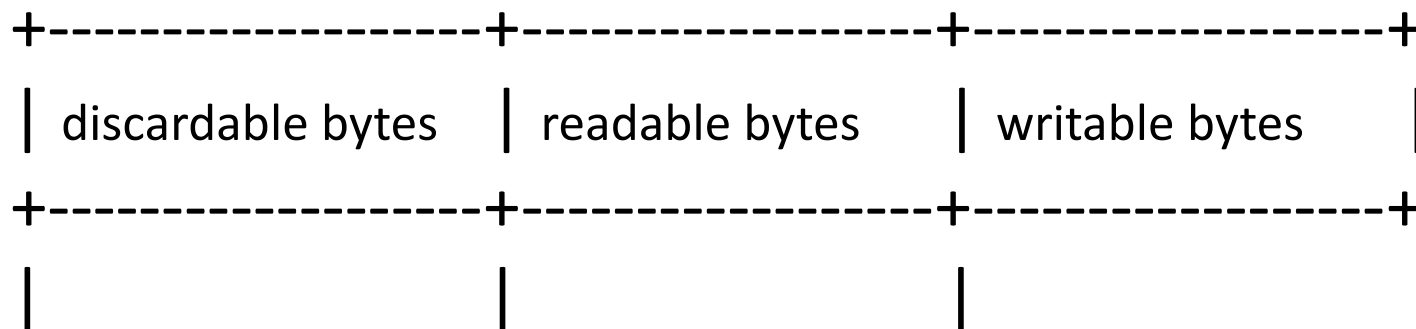
- ★ connect
- ★ bind
- ★ write
- ★ close
- ★ disconnect
- ★ unbind
- ★ isOpen
- ★ isBound
- ★ isConnected
- ★ isReadable
- ★ isWritable

Channel		
+ OP_NONE	: int	= 0
+ OP_READ	: int	= 1
+ OP_WRITE	: int	= 4
+ OP_READ_WRITE	: int	= OP_READ OP_WRITE
+ getId ()	: Integer	
+ getFactory ()	: ChannelFactory	
+ getParent ()	: Channel	
+ getConfig ()	: ChannelConfig	
+ getPipeline ()	: ChannelPipeline	
+ isOpen ()	: boolean	
+ isBound ()	: boolean	
+ isConnected ()	: boolean	
+ getLocalAddress ()	: SocketAddress	
+ getRemoteAddress ()	: SocketAddress	
+ write (Object message)	: ChannelFuture	
+ write (Object message, SocketAddress remoteAddress)	: ChannelFuture	
+ bind (SocketAddress localAddress)	: ChannelFuture	
+ connect (SocketAddress remoteAddress)	: ChannelFuture	
+ disconnect ()	: ChannelFuture	
+ unbind ()	: ChannelFuture	
+ close ()	: ChannelFuture	
+ getCloseFuture ()	: ChannelFuture	
+ getInterestOps ()	: int	
+ isReadable ()	: boolean	
+ isWritable ()	: boolean	
+ setInterestOps (int interestOps)	: ChannelFuture	
+ setReadable (boolean readable)	: ChannelFuture	



Netty Zero-Copy-Capable Buffer

★ 序列访问索引



$0 \leq \text{readerIndex} \leq \text{writerIndex} \leq \text{capacity}$

★ get & set : **not** modify the readerIndex or writerIndex

★ read & write : modify the readerIndex or writerIndex



★ ChannelBuffer

★ ChannelBuffers

一 隐藏具体类型

ChannelBuffers		
+ BIG_ENDIAN	: ByteOrder	= ByteOrder.BIG_ENDIAN
+ LITTLE_ENDIAN	: ByteOrder	= ByteOrder.LITTLE_ENDIAN
+ EMPTY_BUFFER	: ChannelBuffer	= new BigEndianHeapChannelBuffer(0)
- HEXDUMP_TABLE	: char[]	= new char[256 * 4]
- <<staticInitializer>>	_STATIC_INITIALIZER ()	: void
+ buffer (int capacity)		: ChannelBuffer
+ buffer (ByteOrder endianness, int capacity)		: ChannelBuffer
+ directBuffer (int capacity)		: ChannelBuffer
+ directBuffer (ByteOrder endianness, int capacity)		: ChannelBuffer
+ dynamicBuffer ()		: ChannelBuffer
+ dynamicBuffer (ChannelBufferFactory factory)		: ChannelBuffer
+ dynamicBuffer (int estimatedLength)		: ChannelBuffer
+ dynamicBuffer (ByteOrder endianness, int estimatedLength)		: ChannelBuffer
+ dynamicBuffer (int estimatedLength, ChannelBufferFactory factory)		: ChannelBuffer
+ dynamicBuffer (ByteOrder endianness, int estimatedLength, ChannelBufferFactory factory)		: ChannelBuffer
+ wrappedBuffer (byte array[])		: ChannelBuffer
+ wrappedBuffer (ByteOrder endianness, byte array[])		: ChannelBuffer
+ wrappedBuffer (byte array[], int offset, int length)		: ChannelBuffer
+ wrappedBuffer (ByteOrder endianness, byte array[], int offset, int length)		: ChannelBuffer
+ wrappedBuffer (ByteBuffer buffer)		: ChannelBuffer
+ wrappedBuffer (ChannelBuffer buffer)		: ChannelBuffer
+ wrappedBuffer (byte... arrays[])		: ChannelBuffer
+ wrappedBuffer (ByteOrder endianness, byte... arrays[])		: ChannelBuffer
- compositeBuffer (ByteOrder endianness, List<ChannelBuffer> components)		: ChannelBuffer
+ wrappedBuffer (ChannelBuffer... buffers)		: ChannelBuffer
+ wrappedBuffer (ByteBuffer... buffers)		: ChannelBuffer
+ copiedBuffer (byte array[])		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, byte array[])		: ChannelBuffer
+ copiedBuffer (byte array[], int offset, int length)		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, byte array[], int offset, int length)		: ChannelBuffer
+ copiedBuffer (ByteBuffer buffer)		: ChannelBuffer
+ copiedBuffer (ChannelBuffer buffer)		: ChannelBuffer
+ copiedBuffer (byte... arrays[])		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, byte... arrays[])		: ChannelBuffer
+ copiedBuffer (ChannelBuffer... buffers)		: ChannelBuffer
+ copiedBuffer (ByteBuffer... buffers)		: ChannelBuffer
+ copiedBuffer (CharSequence string, Charset charset)		: ChannelBuffer
+ copiedBuffer (CharSequence string, int offset, int length, Charset charset)		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, CharSequence string, Charset charset)		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, CharSequence string, int offset, int length, Charset charset)		: ChannelBuffer
+ copiedBuffer (char array[], Charset charset)		: ChannelBuffer
+ copiedBuffer (char array[], int offset, int length, Charset charset)		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, char array[], Charset charset)		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, char array[], int offset, int length, Charset charset)		: ChannelBuffer
- copiedBuffer (ByteOrder endianness, CharBuffer buffer, Charset charset)		: ChannelBuffer
+ copiedBuffer (String string, String charsetName)		: ChannelBuffer
+ copiedBuffer (ByteOrder endianness, String string, String charsetName)		: ChannelBuffer
+ unmodifiableBuffer (ChannelBuffer buffer)		: ChannelBuffer
+ hexDump (ChannelBuffer buffer)		: String
+ hexDump (ChannelBuffer buffer, int fromIndex, int length)		: String
+ hashCode (ChannelBuffer buffer)		: int
+ equals (ChannelBuffer bufferA, ChannelBuffer bufferB)		: boolean
+ compare (ChannelBuffer bufferA, ChannelBuffer bufferB)		: int
+ indexOf (ChannelBuffer buffer, int fromIndex, int toIndex, byte value)		: int
+ indexOf (ChannelBuffer buffer, int fromIndex, int toIndex, ChannelBufferIndexFinder indexFinder)		: int
+ swapShort (short value)		: short
+ swapMedium (int value)		: int
+ swapInt (int value)		: int
+ swapLong (long value)		: long
+ firstIndexOf (ChannelBuffer buffer, int fromIndex, int toIndex, byte value)		: int
- lastIndexOf (ChannelBuffer buffer, int fromIndex, int toIndex, byte value)		: int
+ firstIndexOf (ChannelBuffer buffer, int fromIndex, int toIndex, ChannelBufferIndexFinder indexFinder)		: int
- lastIndexOf (ChannelBuffer buffer, int fromIndex, int toIndex, ChannelBufferIndexFinder indexFinder)		: int
+ encodeString (ByteBuffer src, Charset charset)		: ByteBuffer
- decodeString (ByteBuffer src, Charset charset)		: String
- <<Constructor>>	ChannelBuffers ()	



Netty数据流分析

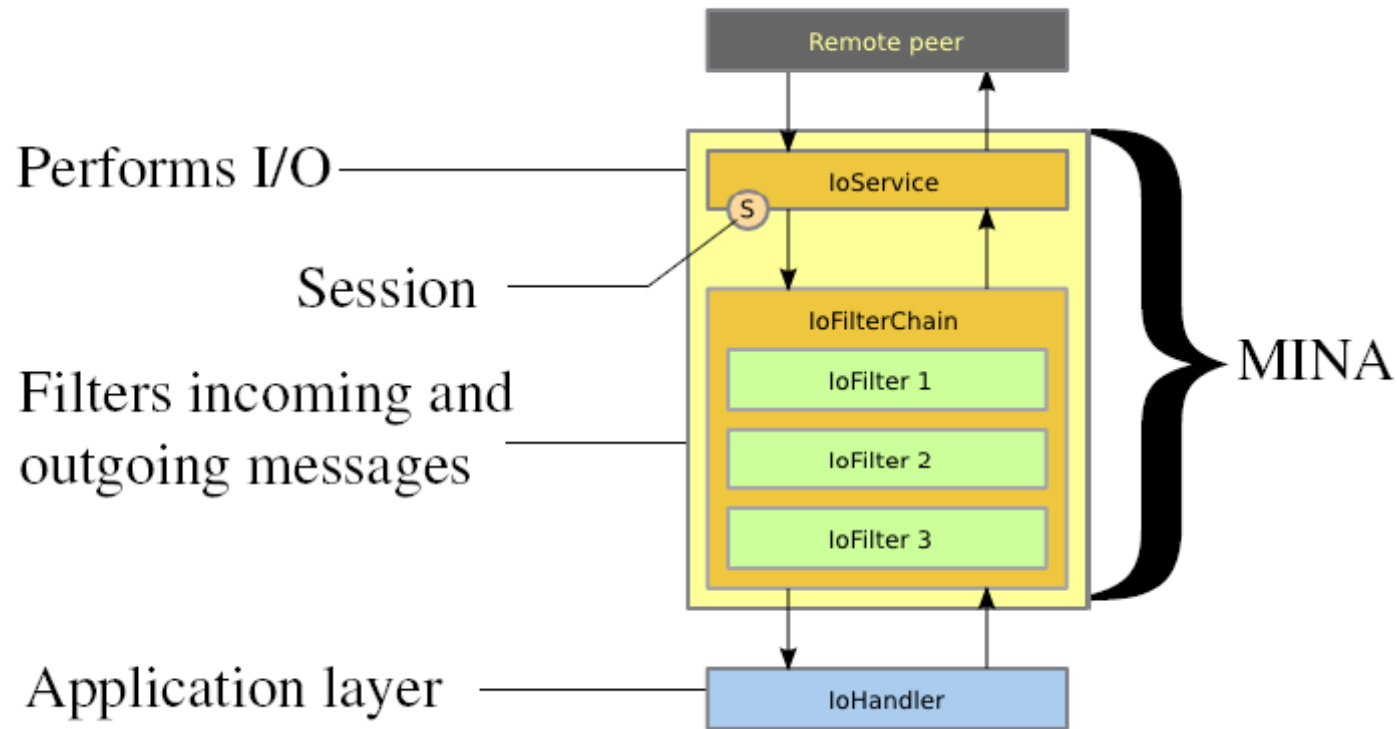
- ★ 服务器启动
- ★ 服务器主通道监听
- ★ 服务器子通道开通
- ★ 客户端启动
- ★ 客户端主通道监听
- ★ 客户端子通道开通



Netty VS Mina



Mina 架构



Mina 服务器端架构

★ IO Acceptor

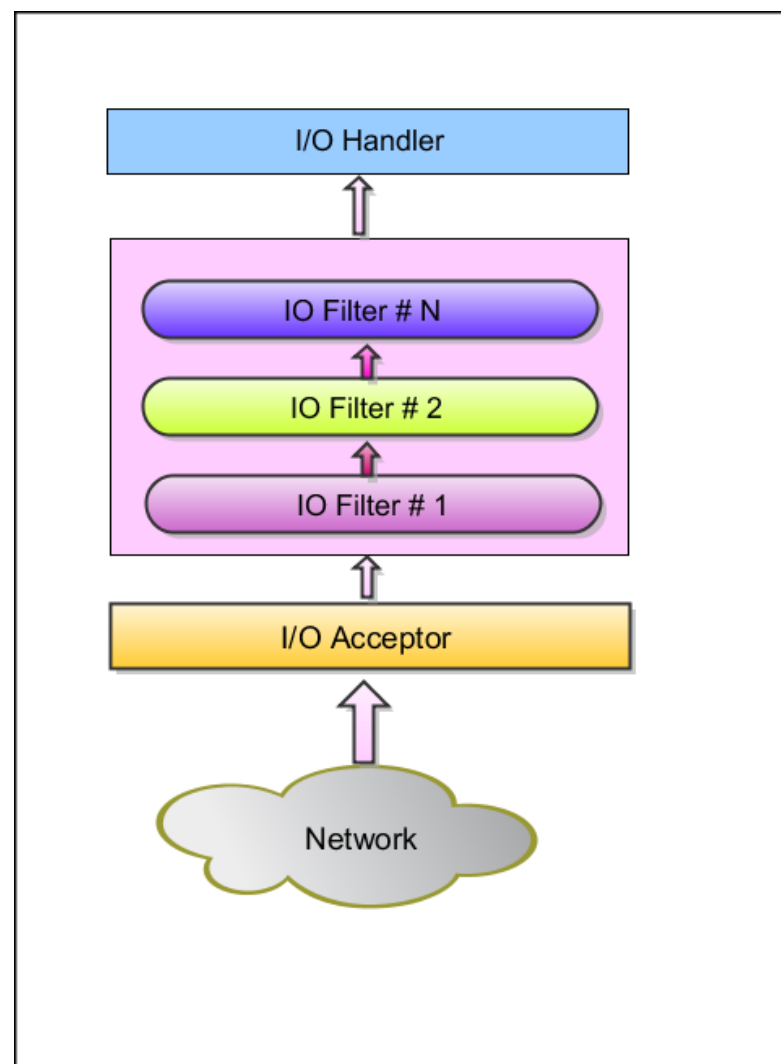
- 实现了IoService
- 监听网络请求或数据包
- 新连接，创建session

★ IO Filter

- 过滤和传输
- 编码解码

★ IO Handler

- 实现业务逻辑



Mina 客户架构

★ IO Connector

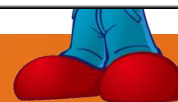
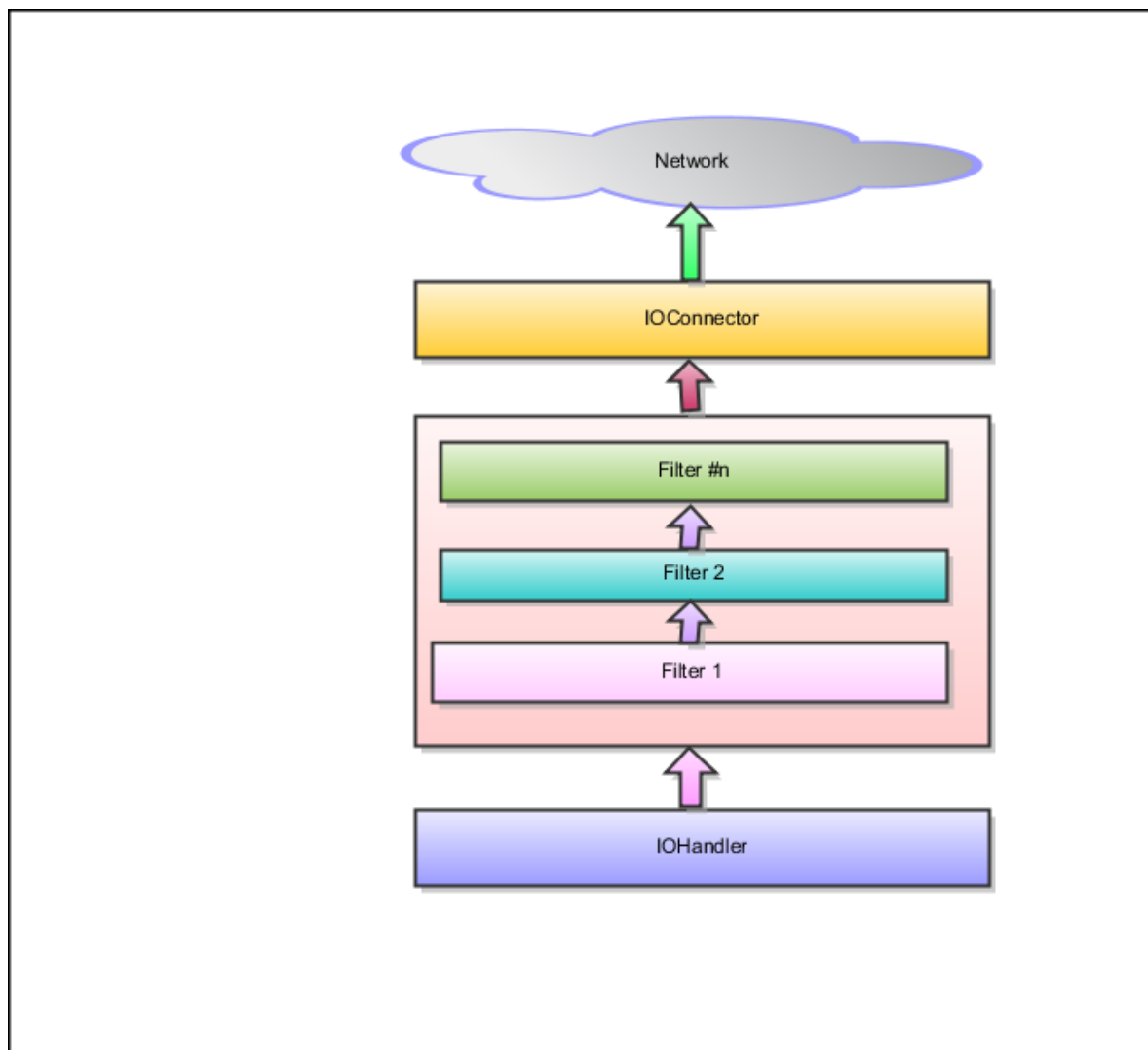
- 实现IO Server
- 连接到Socket
- bind到Server
- 创建session

★ IO Filter

- 过滤和传输
- 编码解码

★ IO Handler

- 实现业务逻辑



Netty VS Mina

- ★ Netty基于Pipeline处理，Mina基于Filter过滤
- ★ Netty的事件驱动模型具有更好的扩展性和易用性
- ★ Https，SSL，PB，RSTP，Text & Binary等协议支持
- ★ Netty中UDP传输有更好的支持
- ★ 官方测试Netty比Mina性能更好





Q & A

雷腾 L.T

Email : leiteng@taobao.com