Netty

grayson

2022-07-13

1 Netty

Netty

2 Netty NIO

1. **NIO API** Selector ServerSockerChannel SocketChannel ByteBuffer
2. Java NIO Reactor
   NIO
3. NIO
4. **JDK NIO BUG** `epoll bug` Selector CPU 100% JDK 1.6 `update18` JDK 1.7 BUG
5. **JDK NIO** NIO
   NIO Netty NIO **UDP**

3

3.1

1. **API**
2.
3. **`ChannelHandler`**
4. **NIO Netty**
5. **Netty JDK NIO BUG** **NIO BUG**
6. **BUG**
7. Hadoop RPC Avro Netty

4

1.
   1. RPC Netty
      RPC
   2. Dubbo RPC Dubbo Dubbo Netty
2.
   1. Netty **TCP/UDP HTTP**
   2. **Netty**
3.

1. Hadoop **Avro** **RPC** **Netty**

5

Netty

1.
2. **Reactor**
3.
4.
5.

5.1

1. I/O **I/O**
2. I/O **I/O** **select**
3. / I/O

4. Netty I/O NioEventLoop **Selector** **SocketChannel**
I/O **I/O**
5. Netty **I/O** **N** **I/O**

5.2 Reactor

2

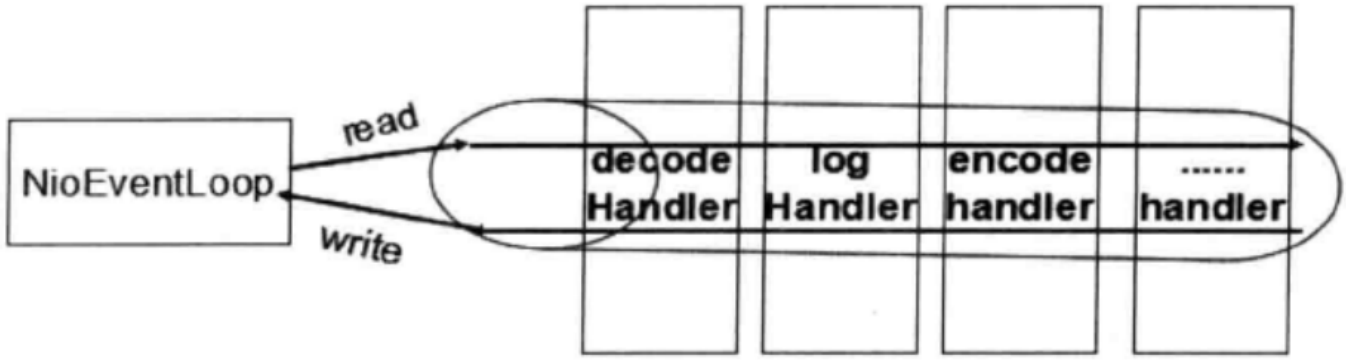
5.3

1.

2.

3. **CPU** **NIO**

4. **Netty** **I/O**



1. Netty NioEventLoop ChannelPipeline **fireChannelRead()**
   NioEventLoop **Handler**

5.4

1.
   1.
   2. CPU
   3.
2. Netty **Google Protobuf** **Netty**
3. 4.1

5.5

5.5.1 Linux

1. **CPU**

2. 
   1. **read()** **DMA**
   2.
   3. **send()**
      Socket
   4. **send()** **Socket**

>

3. DMA DMA Direct Memory Access DMA
   CPU I/O CPU

4.

> I/O

DMA

Socket

Linux **sendfile()**

Java NIO **FileChannel** **transferTo()** **FileChannel** **Channel** transferTo()

```
1. public abstract long transferTo(long position, long count, WritableByteChannel target) throws IOException;
```

FileChannel.transferTo()

DMA **Socket** **4** **3**

Linux 2.4 **Socket Buffer** **Descripter** **DMA** **Socket** **Socket** **DMA** **CPU**

5.5.2 Netty

Netty Linux Netty 5

1. **JVM**
2. **CompositeByteBuf** **Buffer** **Buffer** **Buffer**
3. **Unpooled.wrappedBuffer()** **byte** **ByteBuf**
4. ByteBuf.slice() **Unpooled.wrappedBuffer()** slice() **ByteBuf** **ByteBuf** **byte**
5. **Netty** **FileRegion** **FileRegion** **FileChannel.transferTo()** **Channel**

5.5.2.1

1. **JVM** **I/O** **VM**
2. JVM I/O
   1. **JVM** **JVM** **JVM**

2. **JVM GC** JVM GC

3. **Netty** **I/O** **JVM**

5.5.2.2 CompositeByteBuf

1. CompositeByteBuf **Netty** CompositeByteBuf **Buffer** **ByteBuf** **CompositeByteBuf** **ByteBuf**

   1. HTTP `header` `body` ByteBuf

   2. ByteBuf

      ```
      1. ByteBuf httpBuf = Unpooled.buffer(header.readableBytes() + body.readableBytes());
      2. httpBuf.writeBytes(header);
      3. httpBuf.writeBytes(body);
      ```

      `header` **`body`** **ByteBuf** **ByteBuf** **`header`** **`body`** **ByteBuf** **CPU**

   3. CompositeByteBuf

      ```
      1. CompositeByteBuf httpBuf = Unpooled.compositeBuffer();
      2. httpBuf.addComponents(header, body);
      ```

      1. CompositeByteBuf **`addComponents()`** **ByteBuf** **`byte`**

      2. CompositeByteBuf 

         1. CompositeByteBuf **Components** **Component** **ByteBuf** **ByteBuf** **CompositeByteBuf** **Component** **CompositeByteBuf**

            ```
            1. private static final class Component {
            2.     final ByteBuf srcBuf; //          ByteBuf
            3.     final ByteBuf buf; // srcBuf                ByteBuf
            4.
            5.     int srcAdjustment; // CompositeByteBuf                  srcBuf
            6.     int adjustment; // CompositeByteBuf                    buf
            7.
            8.     int offset; // Component          CompositeByteBuf
            9.     int endOffset; // Component          CompositeByteBuf
            10.
            11.    Component(ByteBuf srcBuf, int srcOffset, ByteBuf buf, int bufOffset,
            ```

```
    int offset, int len, ByteBuf slice) {
    this.srcBuf = srcBuf;
    this.srcAdjustment = srcOffset - offset;
    this.buf = buf;
    this.adjustment = bufOffset - offset;
    this.offset = offset;
    this.endOffset = offset + len;
    this.slice = slice;
    }

    // ...
}
```

2. Component HTTP header body CompositeByteBuf Component

1. header body ByteBuf ByteBuf header body header offset ~ endOffset 0 ~ 6 body offset ~ endOffset 6 ~ 10 Component offset **endOffset** **ByteBuf** **offset** **endOffset** **Component** **ByteBuf**
2. Component srcAdjustment **adjustment** **CompositeByteBuf** **ByteBuf** **adjustment = readerIndex - offset** **CompositeByteBuf** **Component** **ByteBuf** header 1 body 2 Component 

5.5.2.3 Unpooled.wrappedBuffer

1. Unpooled **wrappedBuffer** **ByteBuf** byte[] ByteBuf ByteBuffer **ByteBuf** **ByteBuf** **byte**

5.5.2.4 ByteBuf.slice

1. Bytebuf.slice() **Unpooled.wrappedBuffer()** ByteBuf.slice() **ByteBuf** **ByteBuf**

2. HTTP slice() header body ByteBuf header body

```
1. ByteBuf header = httpBuf.slice(0, 6);
2. ByteBuf body = httpBuf.slice(6, 4);
```

`slice()` ByteBuf `readerIndex` `writerIndex`
ByteBuf ByteBuf ByteBuf ByteBuf

 8.png

5.5.2.5 FileRegion

1. Netty **FileRegion** **DefaultFileRegion** **DefaultFileRegion**
   NioSocketChannel **FileRegion** **FileChannel** **JDK NIO**
   `FileChannel.transferTo()` FileRegion

```
@Override
public long transferTo(WritableByteChannel target, long position) throws IOException {
    long count = this.count - position;
    if (count < 0 || position < 0) {
        throw new IllegalArgumentException(
                "position out of range: " + position +
                " (expected: 0 - " + (this.count - 1) + ')');
    }
    if (count == 0) {
        return 0L;
    }
    if (refCnt() == 0) {
        throw new IllegalReferenceCountException(0);
    }
    // Call open to make sure fc is initialized. This is a no-oop if we called it before.
    open();

    long written = file.transferTo( position: this.position + position, count, target);
    if (written > 0) {
        transferred += written;
    } else if (written == 0) {
        // If the amount of written data is 0 we need to check if the requested count is bigger then the
        // actual file itself as it may have been truncated on disk.
        //
        // See https://github.com/netty/netty/issues/8868
        validate( region: this, position);
    }
    return written;
}
```

---

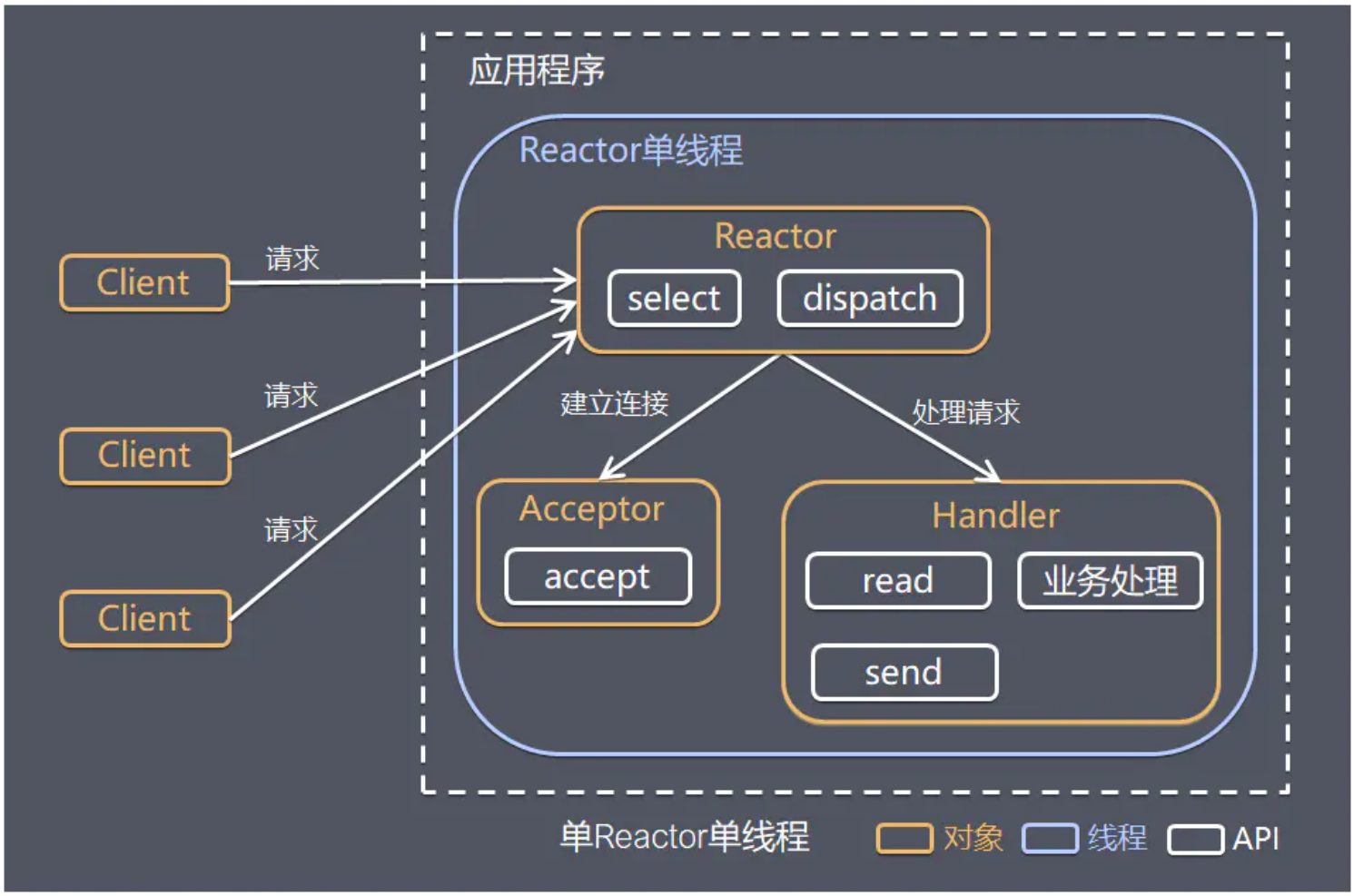
1. 03 RPC
2. Netty 2
3. Netty Netty
4. offer -Netty 15
5. ( ) Netty
6. 16 IO Netty

1 Reactor

1. Reactor **Reactor** **Reactor**
2. **Reactor** Reactor
    1. **Reactor**
    2. **Reactor**
    3. **Reactor**

1.1 Reactor

1.1.1



1. Reactor **Selector** **dispatch**
    1. **Acceptor** `accept` **Handler**
    2. **Handler**
2. Handler **$read \rightarrow (decode \rightarrow compute \rightarrow encode) \rightarrow send$**

1.1.2

1.1.2.1

1. Redis
   $O(1)$

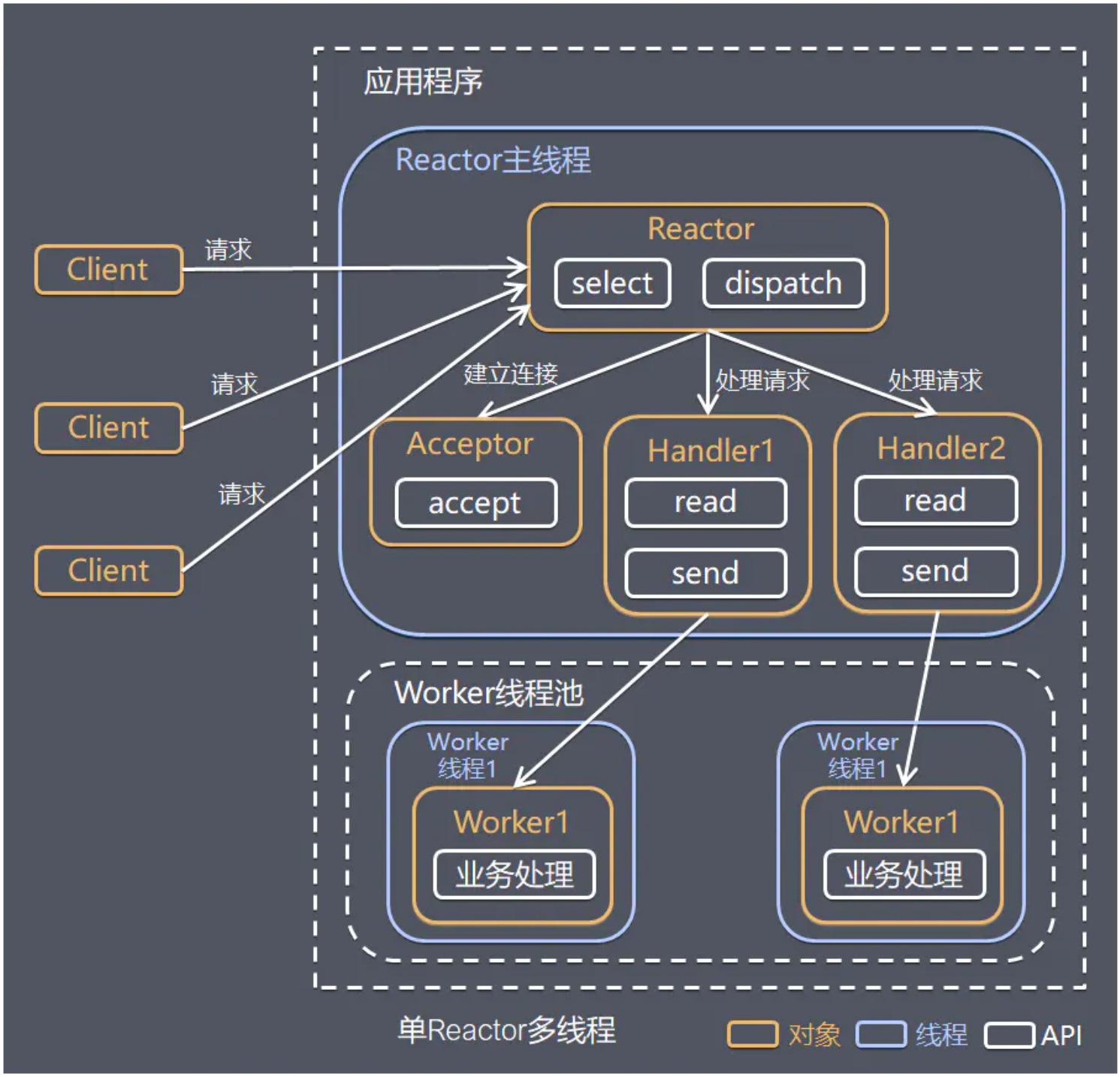
1.1.2.2

1. **CPU**
2. **Handler**
3.

1.2 Reactor

1.2.1

> Reactor Reactor



1. Reactor **Selector** **dispatch**
    1. **Acceptor** `accept` **Handler** **Handler** `read` `write`
2. **Handler** **Handler** `send`

1.2.2

1.2.2.1

1. **CPU**

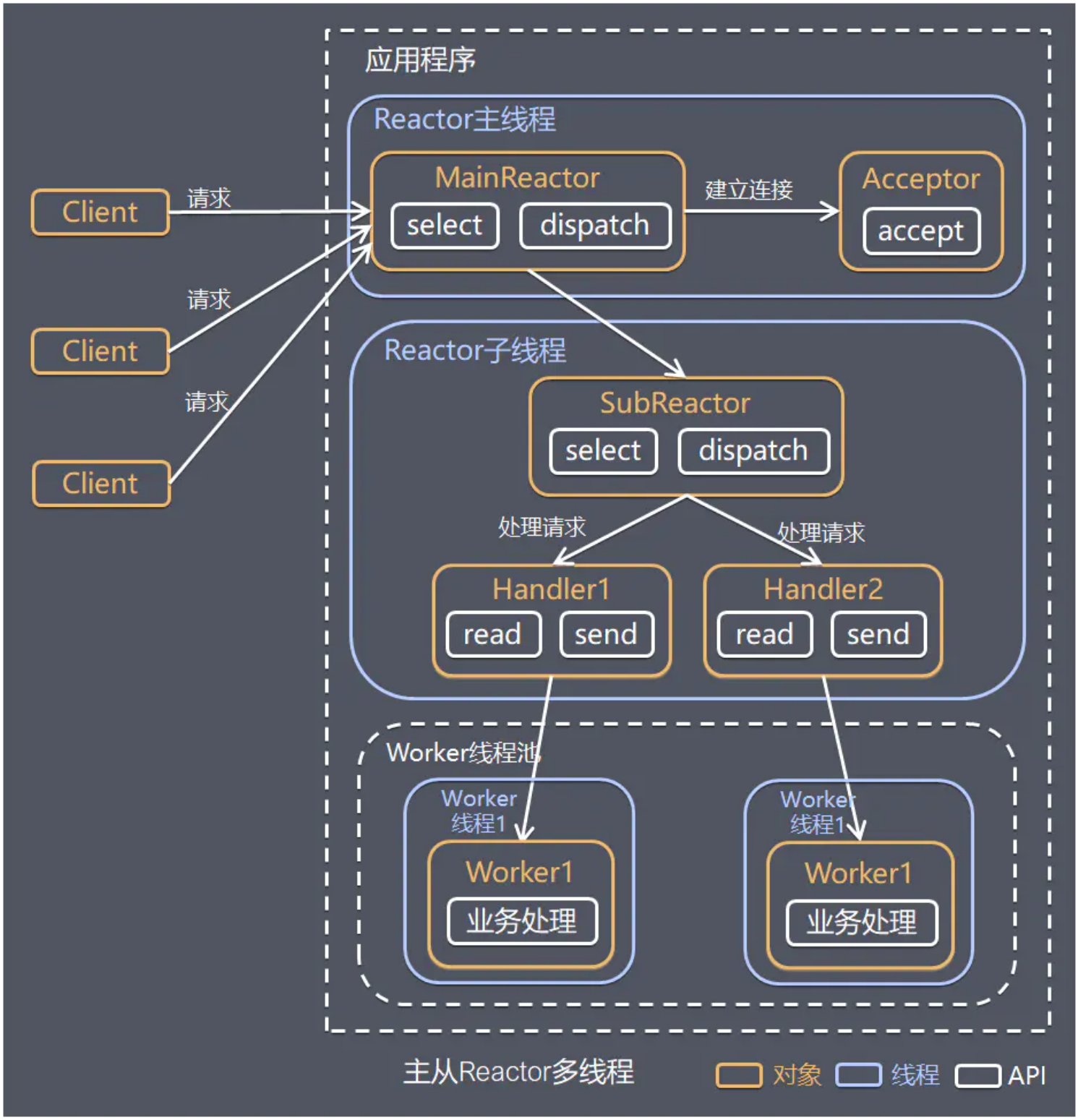
1.2.2.2

1.
2. Reactor Reactor

1.3 Reactor

1.3.1

> Reactor Reactor Reactor
> Reactor



1. **Reactor Reactor Selector dispatch Reactor Reactor MainReactor SubReactor**
2. Reactor **MainReactor Selector Acceptor Acceptor MainReactor SubReactor**
3. **SubReactor MainReactor Selector Handler**
4. **Handler $read \rightarrow \rightarrow send$**

1.3.2

1.3.2.1

1.
2. **Reactor**
3. Ngnix Ngnix Memcached Netty

1.3.2.2

1.

1.4

1. Reactor
    1. **Reactor**
    2. **Reactor** **1**
    3. **Reactor**
2. Reactor
    1. Reactor
    2.
    3. **Reactor** **CPU**
    4. **Reactor**

2 Netty

> Netty Netty Reactor
> Reactor Reactor

2.1

2.2

1. **NioEventLoopGroup** **Reactor** **BossGroup** **WorkerGroup** **BossGroup** **TCP** **WorkerGroup** **IO**

    > 1. NioEventLoopGroup NioEventLoop
    > 2. NioEventLoop NioEventLoop Selector Socket

2. BossGroup NioEventLoop
    1. **accept**

2. `accept` **NioSocketChannel** **WorkerGroup** **NioEventLoop** **Selector**
3.

3. WorkerGroup NioEventLoop
   1. `read` `write`
   2. **IO** read write **Pipeline** **NioSocketChannel** **Pipeline** **NioSocketChannel** **Channel** **Handler**
   3.

---

1. Netty Netty
2. Netty
3. Netty 2

2.0 Channel

1

1. Channel **Netty**

2. **Netty** **Channel** **EventLoop** **ByteBufAllocator** **ChannelPipeline**

1.1

1.1.1 **JDK NIO** **Channel**

1. JDK `SocketChannel` `ServerSocketChannel` **Channel**

2. JDK `SocketChannel` `ServerSocketChannel` **I/O** **SPI** **SPI** **SocketChannel** **ServerSocketChannel** **Channel**

> SPI API
>
> 1. **SPI**
>   1. SPI **Service Provider Interface** **Java** **API**
>   2. SPI
>   3. SPI
> 2. **API**
>   1. API **Application Programming Interface**
>   2. API

3. **Netty** **Channel** **Netty** I/O ChannelPipeline TCP **JDK** **SocketChannel** **ServerSocketChannel**

4. **Channel**

1.1.2

1. **Channel** **Facade** **I/O** **I/O**
2. **Channel** **SocketChannel** **ServerSocketChannel**

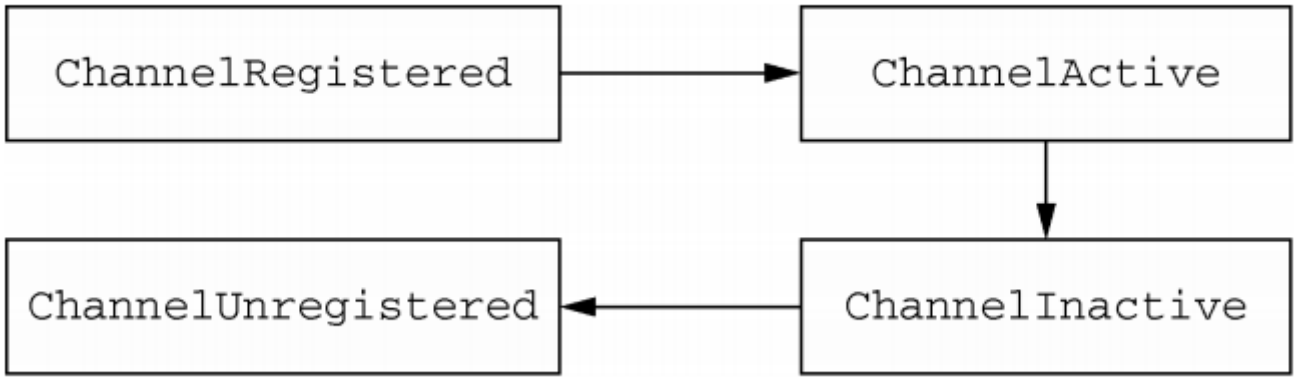
3. **Channel** **Channel**

1.1.3

1. Channel **ChannelInboundHandler**

    1. `ChannelUnregistered` Channel **EventLoop**
    2. `ChannelRegistered` Channel **EventLoop**
    3. `ChannelActive` Channel
    4. `ChannelInactive` Channel

2. Channel **ChannelPipeline**
   ChannelHandler



---

1. Netty 2
2. Java SPI
3. Netty

2.1 ChannelHandler ChannelPipeline

1 ChannelHandler

1.1

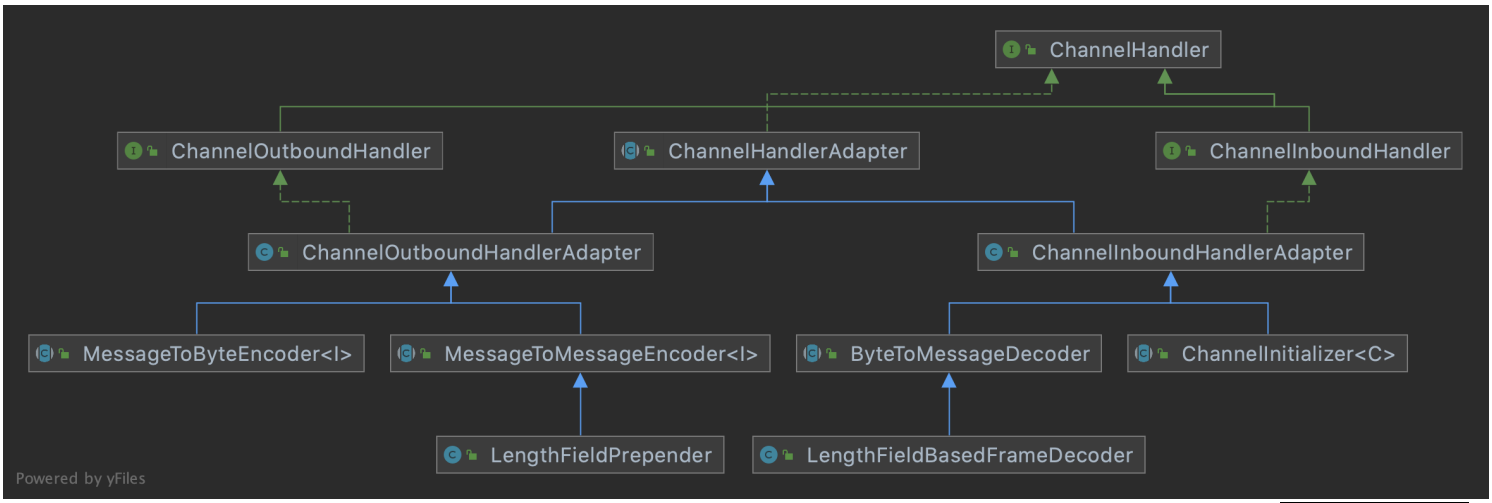
1. ChannelHandler **Servlet** **Filter** **I/O** **I/O**
2.

> Pass Through

3. ChannelHandler
4. ChannelHandler
   1. Shareable **ChannelPipeline** **ChannelHandler**
   2. Skip **Skip**
5. ChannelHandler **ChannelHandler**
   ChannelHandler **ChannelHandler**
   Netty ChannelHandlerAdapter
   ChannelHandler **ChannelHandlerAdapter**
   ChannelHandlerAdapter @Skip
   ChannelHandler

```
1. @Skip
2. @Override
3. public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
4.     ctx.fireExceptionCaught(cause);
5. }
```

1.2



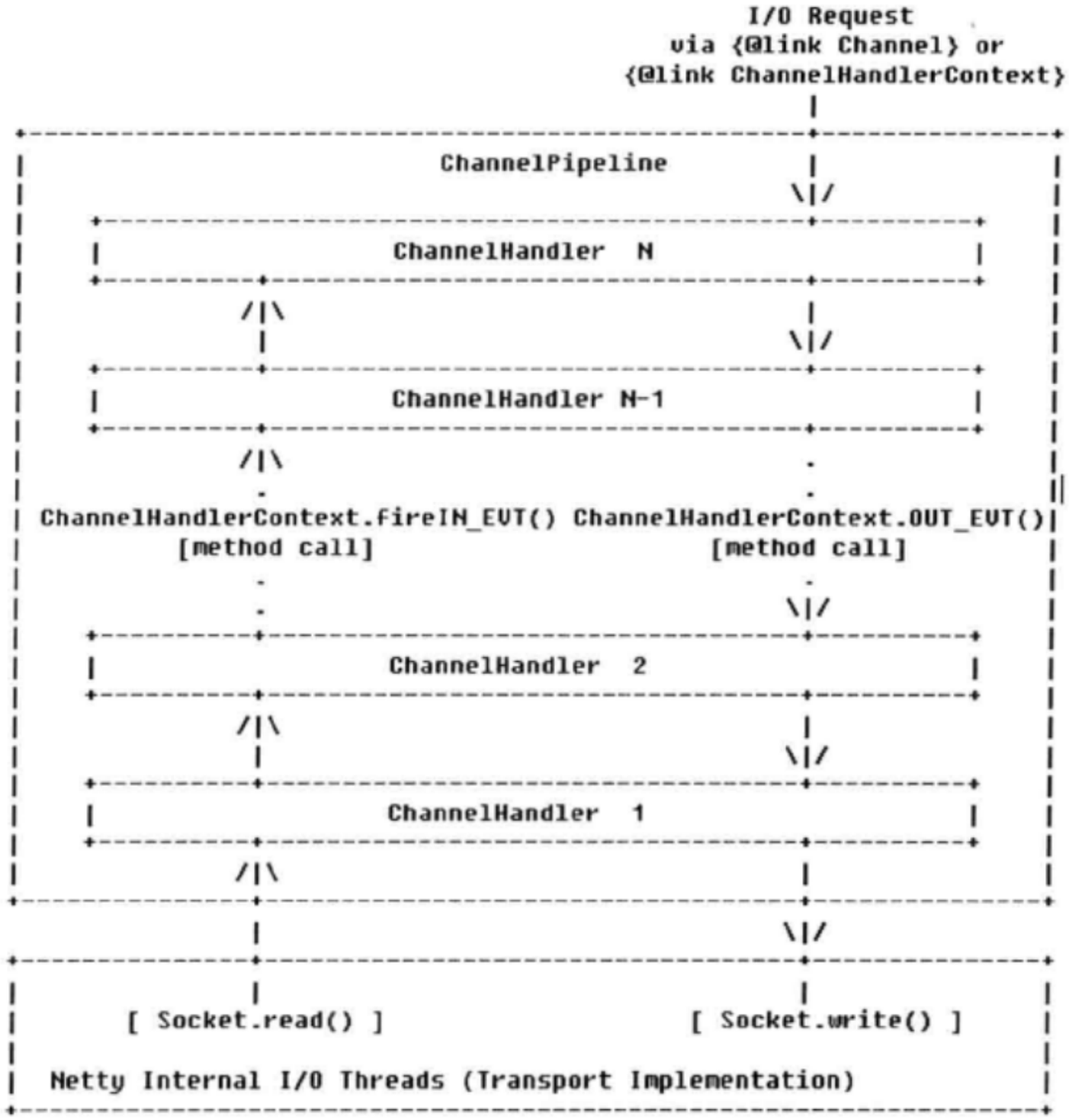
2 ChannelPipeline

2.1

ChannelPipeline **ChanelHandler** **ChannelHandler**

2.1.1

ChannelPipeline **ChannelHandler**



1. SocketChannel `read()` **ByteBuf** **ChannelRead** **IO** **NioEventLoop** **ChannelPipeline** **`fireChannelRead()`** **ChannelPipeline**
2. **HeadHandler** **ChannelHandler1** **ChannelHandler2** … **TailHandler** **ChannelHandler**
3. **ChannelHandlerContext** **`write()`** **TailHandler** **ChannelHandlerN** … **ChannelHandler1** **HeadHandler** Future

2.1.2

> ChannelHandlerContext

Netty inbound **outbound**

1. inbound **I/O** **TCP** inbound
   1. fireChannelRegistered() **Channel**
   2. fireChannelActive() **TCP** **Channel**
   3. fireChannelRead()
   4. fireChannelReadComplete()
   5. fireExceptionCaught()
   6. fireUserEventTriggered()
   7. fireChannelWritabilityChanged() **Channel**
   8. fireChannelInactive() **TCP**
2. outbound **I/O** outbound
   1. bind()
   2. connect()
   3. write()
   4. flush()
   5. read()
   6. disconnect()
   7. close() **Channel**

2.1.3

1. ChannelPipeline **ChannelHandler** ChannelHandler ChannelHandler **ChannelHandler** **ChannelHandlerAdapter**
2.
   1. **Channel Active** **TCP**

```
1. public class MyInboundHandler extends ChannelInboundHandlerAdapter {
2.     @Override
3.     public void channelActive(ChannelHandlerContext ctx) throws Exception {
```

```
        System.out.println("TCP connected!");
        ctx.fireChannelActive();
    }
}
```

2. **Channel Close**

```
public class MyOutboundHandler extends ChannelOutboundHandlerAdapter {
    @Override
    public void close(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        System.out.println("TCP closing ...");
        ctx.close(promise);
    }
}
```

2.1.3 ChannelPipeline

1. **ChannelPipeline** **ServerBootstrap** **Bootstrap** **Netty** **Channel** **ChannelPipeline** **ChannelPipeline**

```
channelPipeline = ch.pipeline();
channelPipeline.addLast("decoder", new MyProtocolDecoder());
channelPipeline.addLast("encoder", new MyProtocolEncoder());
```

2. ChannelHandler MessageToMessageDecoder **ByteToMessageDecoder** **ByteBuf** **POJO** ChannelPipeline **addBefore()** **addAfter()**

2.1.4

1. ChannelPipeline **ChannelHandler** ChannelHandler ChannelPipeline ChannelHandler
2. **ChannelPipeline** N **ChannelPipeline** **ChannelHandler** **ChannelPipeline** **ChannelHandler**

---

1. Netty 2
2.

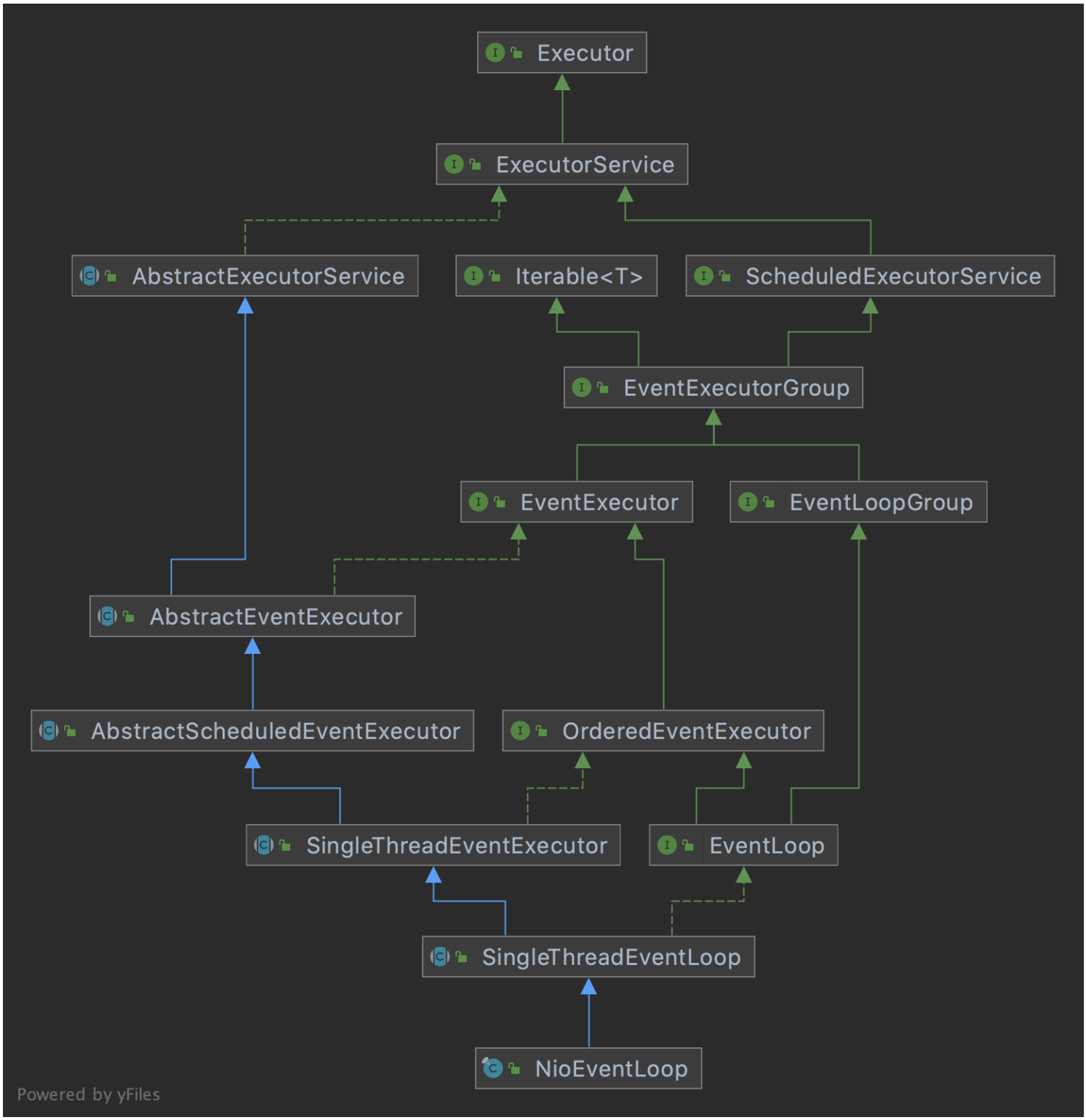
2.2 EventLoop EventLoopGroup

1

Netty NioEventLoop I/O **I/O**

1. **Task**
    1. **NioEventLoop** `execute()` Netty Task **I/O**
       Task **I/O**
2.
    1. **NioEventLoop** `schedule()`

2



1. Netty 2

2.3 Future Promise

1 Future

1.1

1. Netty Future **I/O** **ChannelFuture** **Channel**
2. ChannelFuture **I/O** ** `uncompleted` `completed` **
    1. **I/O** **ChannelFuture** `uncompleted` **I/O**
    2. **I/O** **ChannelFuture** `completed` **
3. ChannelFuture **GenericFutureListener** **`remove()`** **GenericFutureListener** **I/O** **I/O** **ChannelFuture** **GenericFutureListener** **`operationComplete()`** **ChannelFuture** **ChannelFuture**
4. **ChannelHandler** **ChannelFuture** **`await()`** **I/O** **I/O** **I/O** **I/O** **I/O**

```
/**
 * BAD - NEVER DO THIS
 */
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
    ChannelFuture channelFuture = ctx.channel().close();
    channelFuture.awaitUninterruptibly();
    // Perform post-closure operation
    // ...
}

/**
 * GOOD
 */
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
    ChannelFuture channelFuture = ctx.channel().close();
    channelFuture.addListener(new ChannelFutureListener() {
        @Override
        public void operationComplete(ChannelFuture future) throws Exception {
            // Perform post-closure operation
```

```
            // ...
        }
    })
}
```

2 Promise

2.1

1. Promise **Future** Future Netty **Promise** **Future** **I/O**

```
/**
 * Marks this future as a success and notifies all
 * listeners.
 *
 * If it is success or failed already it will throw an {@link IllegalStateException}.
 */
Promise<V> setSuccess(V result);

/**
 * Marks this future as a success and notifies all
 * listeners.
 *
 * @return {@code true} if and only if successfully marked this future as
 *         a success. Otherwise {@code false} because this future is
 *         already marked as either a success or a failure.
 */
boolean trySuccess(V result);

/**
 * Marks this future as a failure and notifies all
 * listeners.
 *
 * If it is success or failed already it will throw an {@link IllegalStateException}.
 */
Promise<V> setFailure(Throwable cause);

/**
 * Marks this future as a failure and notifies all
 * listeners.
 *
 * @return {@code true} if and only if successfully marked this future as
 *         a failure. Otherwise {@code false} because this future is
```

```
33.  *        already marked as either a success or a failure.
34.  */
35. boolean tryFailure(Throwable cause);
```

2. Netty **I/O** **Promise** ChannelHandlerCntext `write()` ChannelPromise

```
1. @Override
2. public ChannelFuture write(Object msg) {
3.     return write(msg, newPromise());
4. }
```

I/O **Promise**

```
1. @Override
2. public ChannelFuture write(Object msg, ChannelPromise promise) {
3.     Channel channel = channel();
4.     if (!channel.isActive()) {
5.         // Mark the write request as failure if the channel is inactive.
6.         if (channel.isOpen()) {
7.             promise.tryFailure(NOT_YET_CONNECTED_EXCEPTION);
8.         } else {
9.             promise.tryFailure(CLOSED_CHANNEL_EXCEPTION);
10.        }
11.        // Release message now to prevent resource-leak.
12.        ReferenceCountUtil.release(msg);
13.    } else {
14.        outBoundBuffer.addMessage(msg, promise);
15.    }
16. }
```

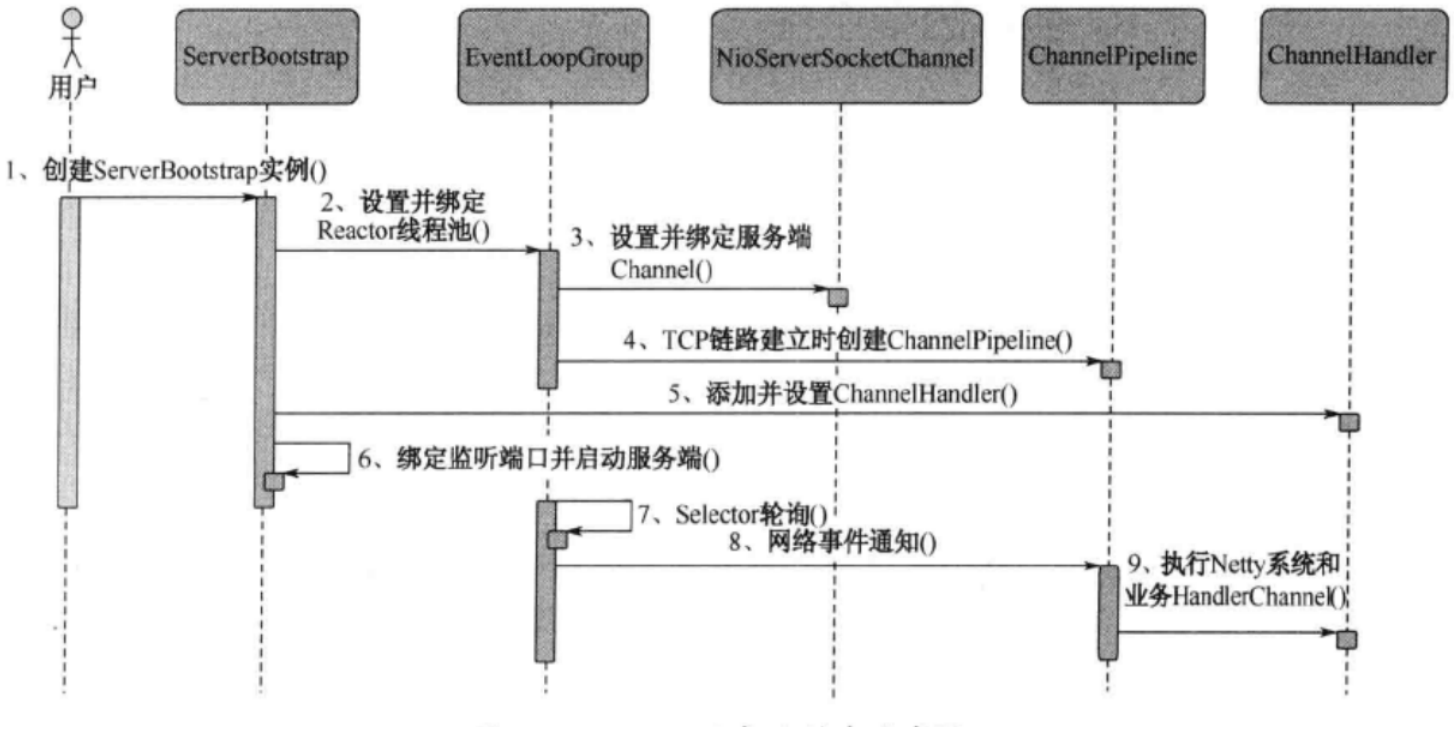
---

1. Netty 2

3.0

1

> Netty NIO ServerBootstrap
> Socket ServerBootstrap Netty



1. **ServerBootstrap**

   1. ServerBootstrap **Netty**
      API

2. **Reactor**

   1. **Netty Reactor EventLoopGroup EventLoop** EventLoop
      Selector Channel Selector EventLoop `run()`
   2. **EventLoop I/O Task Task EventLoop**
      EventLoop
      I/O

3. **Channel**

   1. NIO **ServerSocketChannel Netty NIO NioServerSocketChannel**
   2. Channel **Channel**

3. Netty ServerBootstrap **channel()** **Channel** Netty **NioServerSocketChannel**

```
public B channel(Class<? extends C> channelClass) {
    return channelFactory(new ReflectiveChannelFactory<C>(
            ObjectUtil.checkNotNull(channelClass, "channelClass")
    ));
}
```

4. ** ChannelPipeline**

1. ChannelPipeline **NIO** ** ChannelHandler**
2. **ChannelPipeline** **ChannelPipeline** **ChannelHandler** **ChannelHandler**

1.
2.
3.
4.
5.
6.
7.
8.

5. **ChannelPipeline** **ChannelHandler**

1. **ChannelHandler** **Netty** **ChannelHandler** TSL/SSL Netty **ChannelHandler** ChannelHandler

1. ByteToMessageCodec
2. LengthFieldBasedFrameDecoder
3. Handler LoggingHandler
4. SSL Handler SslHandler
5. Handler IdleStateHandler
6. Handler ChannelTrafficShapingHandler
7. Base64 Base64Decoder Base64Encoder

2. ChannelHandler

```
ServerBootstrap serverBootstrap = new ServerBootstrap();
serverBootstrap.childHandler(new ChannelInitializer<SocketChannel>() {
    @Override
    protected void initChannel(SocketChannel ch) throws Exception {
        ChannelPipeline channelPipeline = ch.pipeline();
```

```
        channelPipeline.addLast(new LoggingHandler());
    }
});
```

6.

    1. **ServerSocketChannel Selector**

```
public ChannelFuture bind(SocketAddress localAddress) {
    validate();
    return doBind(ObjectUtil.checkNotNull(localAddress, "localAddress"));
}
```

7. **Selector**

    1. **Reactor NioEventLoop Selector Channel**

8. **ChannelHandler**

    1. **Channel Reactor NioEventLoop ChannelPipeline ChannelHandler**

9. **ChannelHandler ChannelHandler**

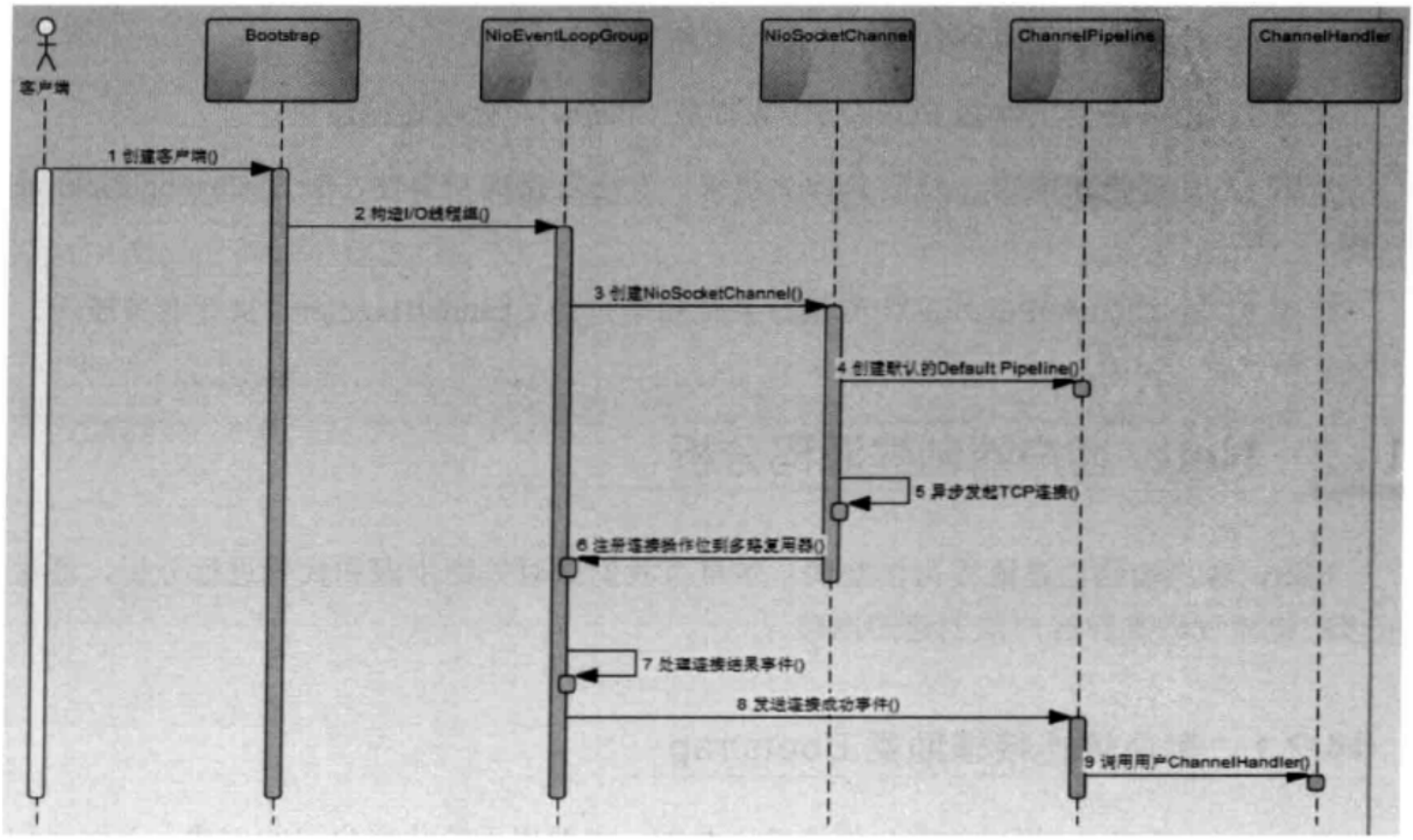
    1. **Netty ChannelHandler ChannelHandler ChannelPipeline ChannelHandler**

---

1. Netty 2

3.1

1



1. **Bootstrap API**
2. **I/O Reactor NioEventLoopGroup I/O CPU 2**
3. **Bootstrap ChannelFactory Channel NioSocketChannel JDK NIO SocketChannel**
4. **ChannelPipeline**
5. **TCP**
    1. **NioSocketChannel**
    2.
6.
7. **I/O Channel**
8. **Future ChannelPipeline**
9. **ChannelPipeline ChannelHandler**

1. Netty 2

TCP

1

1. **TCP**

2. **TCP**

    1. **Socket**

        1. TCP **TCP**

        2. **TCP Socket** SO_SNDBUF SO_RCVBUF **TCP**
           TCP **buffer** **buffer**
            1. SO_SNDBUF
                1. **send()** **Socket** **send()**

                2. send() **buffer** **Socket**
                   buffer
            2. SO_RCVBUF
                1. **Socket** read()
                   buffer
                2. **read()** **Socket** **Socket**
                   TCP **TCP** **TCP**

        3. **TCP** **SO_RCVBUF**

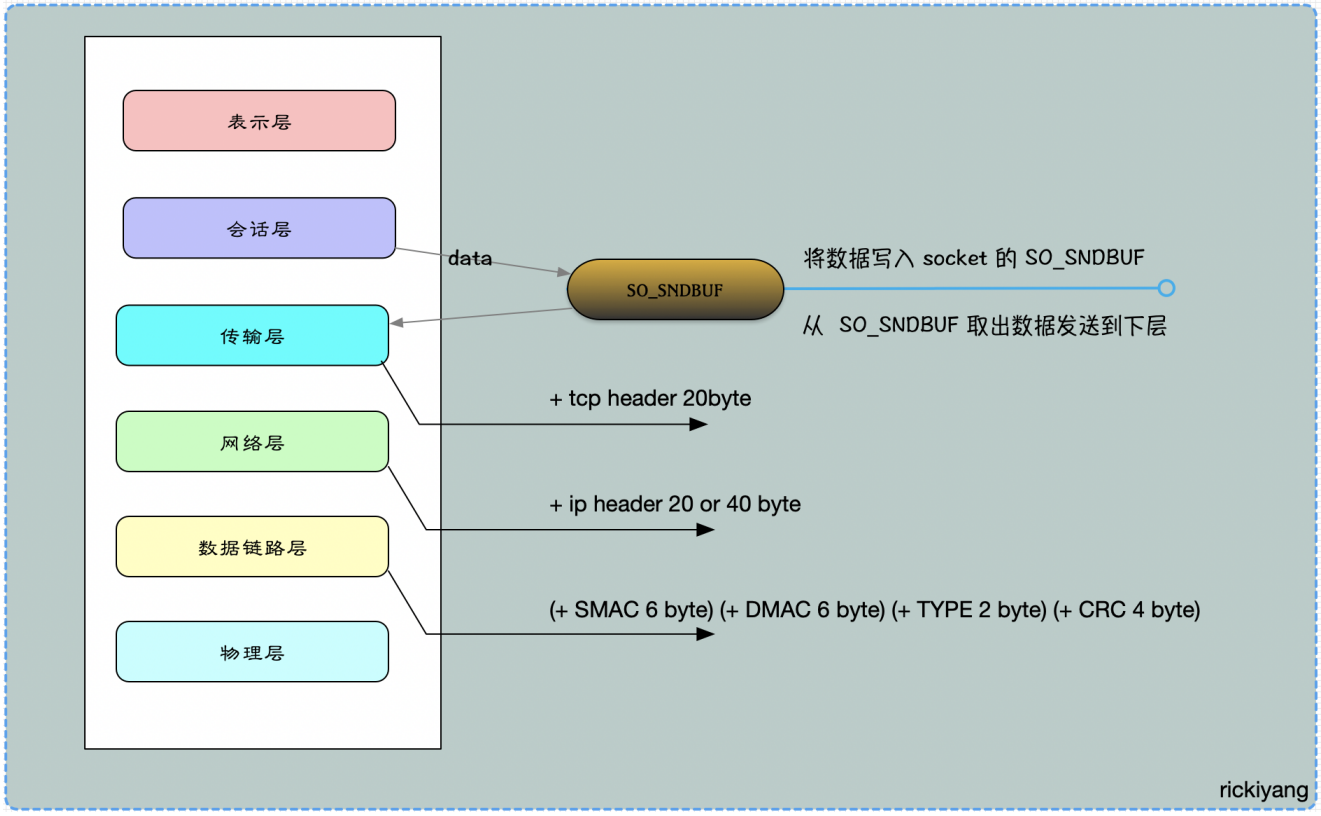
        4. **SO_RCVBUF**
        5. **SO_RCVBUF** **Socket** **SO_RCVBUF**
           ack
           ack
        6. **SO_RCVBUF**
           SO_RCVBUF **ack**

    2. ****MSS/MTU*** *

        1. MTU Maximum Transmission Unit

        2. MSS Maximum Segment SIze **TCP** **data**

3.



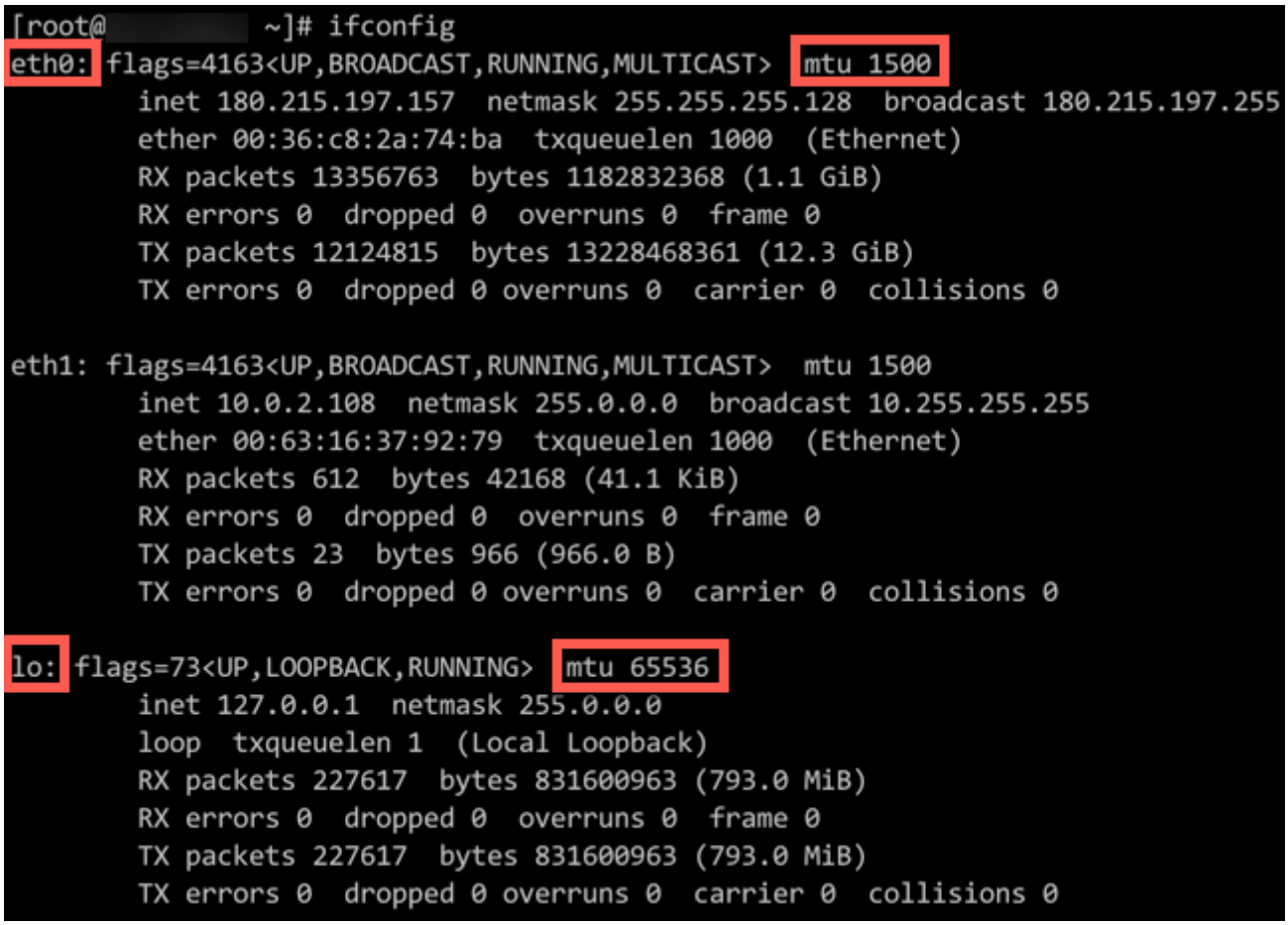
1. `data` **Socket** `SO_SNDBUF` `SO_SNDBUF`
2. `data` **TCP header** 20
3. **TCP** **IP header** **IPv4** **IP header** **20** **IPv6** **IP header** **40**
4. **Datalink header** **CRC** **SMAC** Source MAC MAC **DMAC** Destination MAC MAC **Type** **SMAC + DMAC + Type + CRC** **18**
5.

4. MTU MSS

$$< br > MSS = MTU(1500) - IPHeader(20or40) - TCPHeader(20) < br >$$

5. `SO_SNDBUF` **MSS** **MSS** **TCP Header** **TCP**

6. Lookback **$MTU = 1500$**



2

1. eth0 **MTU 1500**.
2. lo **1500**

3. **Nagle**

1. TCP/IP data TCP Header + IP Header
ACK
2. TCP Nagle

3. Nagle **MSS**
ACK
4. Nagle
1. SO_SNDBUF **MSS**
2. SO_SNDBFU **FIN** **SO_SNDBUF**
3. **TCP_NODELAY = true** TCP_NODELAY **TCP**
Nagle **Server** **ACK** **ACK**
40ms **t** **ACK**
ACK **TCP** **40ms** TCP **40ms**
RTO
TCP_QUICKACK
4. **TCP_CORK** MSS

5. 200ms

5. TCP

1.
2.

6.

1.
2. **MTU**

3. 5 

1. **A B**
2. **A B** **A B**
3. **A B** **B-1** **A** **B**
4. **A** **A-1** **A**
5. **A** **A**

2

>

1.
2.
3. +

2.1

1.

1.

```
+----+------+------+---+----+

| AB | CDEF | GHIJ | K | LM |

+----+------+------+---+----+
```

2. 4 5 4

```
+------+------+------+------+

```

```
| ABCD | EFGH | IJKL | M000 |

+------+------+------+------+
```

2.

3. Netty **FixedLengthFrameDecoder**

2.2

1. \n AB CDEF GHIJ K LM

```
+------------------------+

| AB\nCDEF\nGHIJ\nK\nLM\n |

+------------------------+
```

2. base64 64
Redis

3. Netty **DelimiterBasedFrameDecoder** **LineBasedFrameDecoder**

2.3 +

1. +
$length$ **$length$**

```
+-----+-------+-------+----+-----+

| 2AB | 4CDEF | 4GHIJ | 1K | 2LM |

+-----+-------+-------+----+-----+
```

2. +

3. Netty **LengthFieldBasedFrameDecoder** **length**

1. 06
2. Netty 2
3. Netty

1

1. C++ struct Java
2. **IO** **IO**

   IO
3.
   1. RPC
   2.
   3. Redis

2

Kyro **Protobuf** **ProtoStuff** **Hession** JDK JSON XML

2.1 JDK

1. JDK **java.io.Serializable**

```
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Builder
@ToString
public class RpcRequest implements Serializable {
    private static final long serialVersionUID = 1905122041950251207L;
    private String requestId;
    private String interfaceName;
    private String methodName;
    private Object[] parameters;
    private Class<?>[] paramTypes;
    private RpcMessageTypeEnum rpcMessageTypeEnum;
}
```

2. **serialVersionUID** **serialVersionUID**
   serialVersionUID **InvalidClassException**

serialVersionUID

3.

1.

1.

2.

1.

2.2 Kyro

1. Kyro /

2. Kyro Twitter Groupon Yahoo Hive Storm

3.

```
/**
 * Kryo Kryo Java
 *
 * @author shuang.kou
 * @createTime 2020 05 13 19:29:00
 */
@Slf4j
public class KryoSerializer implements Serializer {

    /**
     * Kryo Kryo Input Output
     * ThreadLocal Kryo
     */
    private final ThreadLocal<Kryo> kryoThreadLocal = ThreadLocal.withInitial(() -> {
        Kryo kryo = new Kryo();
        kryo.register(RpcResponse.class);
        kryo.register(RpcRequest.class);
        kryo.setReferences(true); // true, , true
        kryo.setRegistrationRequired(false); // false, true
        return kryo;
    });

    @Override
    public byte[] serialize(Object obj) {
        try (ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
             Output output = new Output(byteArrayOutputStream)) {
```

```
            Kryo kryo = kryoThreadLocal.get();
            // Object->byte:                    byte
            kryo.writeObject(output, obj);
            kryoThreadLocal.remove();
            return output.toBytes();
        } catch (Exception e) {
            throw new SerializeException("              ");
        }
    }

    @Override
    public <T> T deserialize(byte[] bytes, Class<T> clazz) {
        try (ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(bytes);
             Input input = new Input(byteArrayInputStream)) {
            Kryo kryo = kryoThreadLocal.get();
            // byte->Object:    byte
            Object o = kryo.readObject(input, clazz);
            kryoThreadLocal.remove();
            return clazz.cast(o);
        } catch (Exception e) {
            throw new SerializeException("                ");
        }
    }

}
```

2.3 Protobuf

1. Protobuf Google

2. Protobuf **IDL** `proto` **IDL**

3. Protobuf **IDL** Protobuf

4. `proto`

```
// protobuf
syntax = "proto3";
// SearchRequest                                  Java  class  Go  struct
message Person {
  //string
  string name = 1;
```

```
    // int
    int32 age = 2;
}
```

2.4 ProtoStuff

1. ProtoStuff **Google Protobuf** ProtoStuff

2.5 Hession

1. Hession RPC
2. Dubbo RPC **Hession2** Dubbo Hession2

---

1. 03 RPC
2. PROTOSTUFF