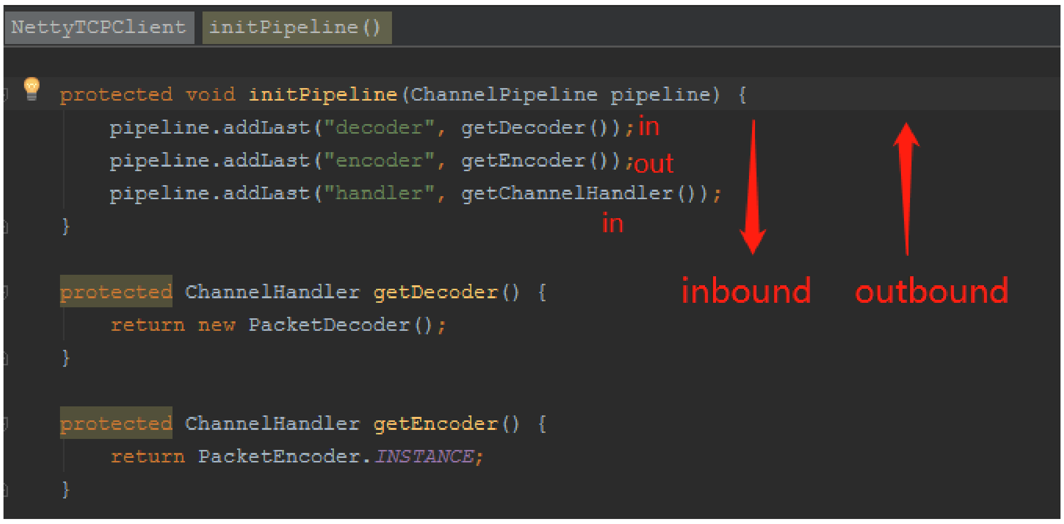
channelPipeline



```
protected void initPipeline(ChannelPipeline pipeline) {
    pipeline.addLast("decoder", getDecoder());
    pipeline.addLast("encoder", getEncoder());
    pipeline.addLast("handler", getChannelHandler());
}

protected ChannelHandler getDecoder() {
    return new PacketDecoder();
}

protected ChannelHandler getEncoder() {
    return PacketEncoder.INSTANCE;
}
```

协议格式

协议说明

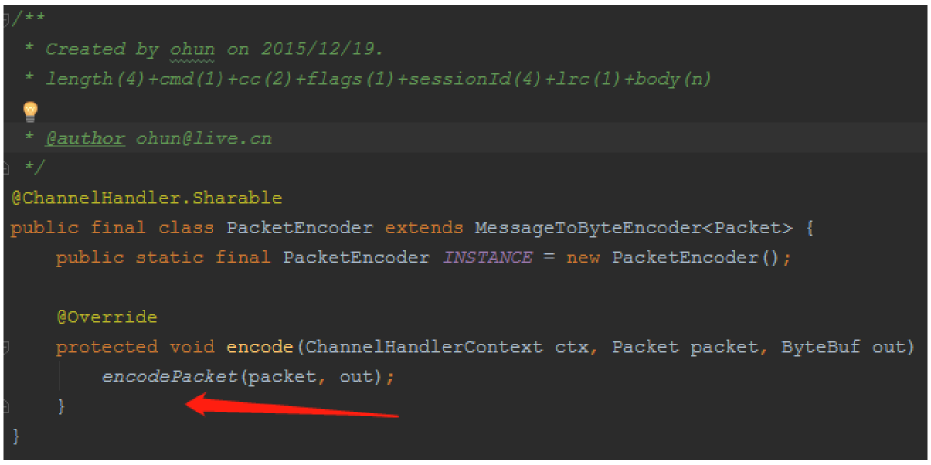
- mpush使用的为自定义私有协议，定长Header + body, 其中header部分固定13个字节。
- 心跳固定为一个字节，值为 -33。

Header 说明

名称	类型	长度	说明
length	int	4	表示body的长度
cmd	byte	1	表示消息协议类型
checkcode	short	2	是根据body生成的一个校验码
flags	byte	1	表示当前包启用的特性，比如是否启用加密，是否启用压缩
sessionId	int	4	消息会话标识用于消息响应
lrc	byte	1	纵向冗余校验，用于校验header

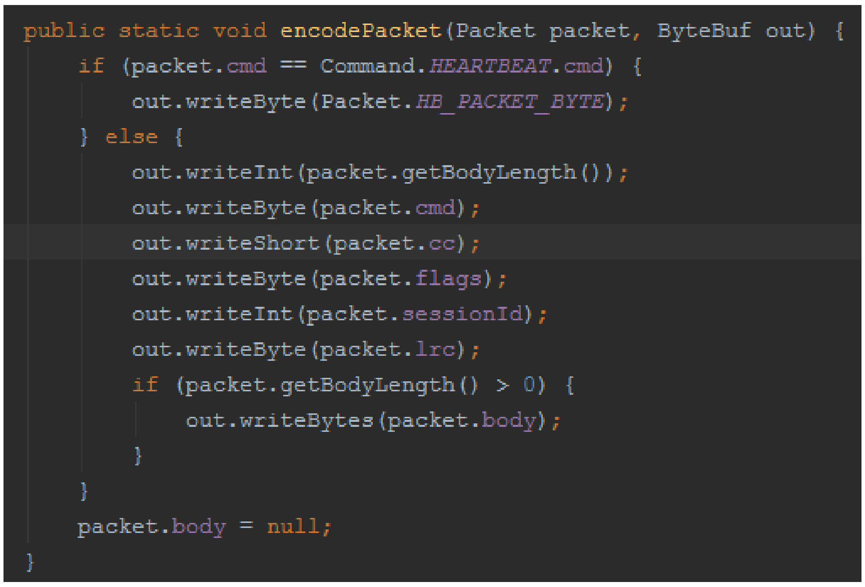
编码解码

编码器PacketEncoder



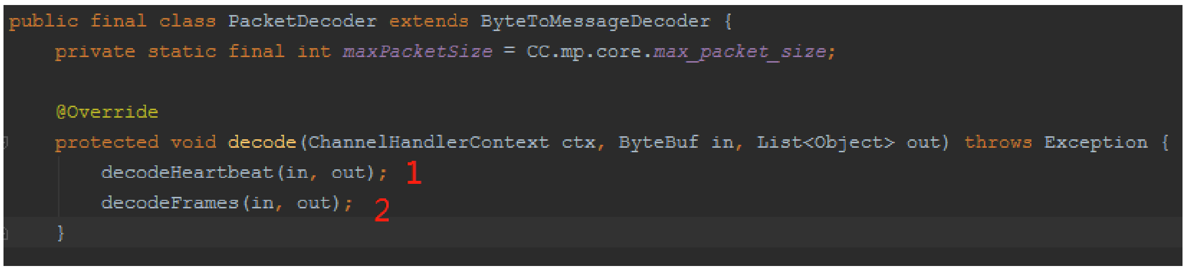
```java
/**
 * Created by ohun on 2015/12/19.
 * length(4)+cmd(1)+cc(2)+flags(1)+sessionId(4)+lrc(1)+body(n)
 *
 * @author ohun@live.cn
 */
@ChannelHandler.Sharable
public final class PacketEncoder extends MessageToByteEncoder<Packet> {
    public static final PacketEncoder INSTANCE = new PacketEncoder();

    @Override
    protected void encode(ChannelHandlerContext ctx, Packet packet, ByteBuf out)
        encodePacket(packet, out);
    }
}
```

MessageToByteEncoder#write方法中会调用子类的encode方法；



```java
public static void encodePacket(Packet packet, ByteBuf out) {
    if (packet.cmd == Command.HEARTBEAT.cmd) {
        out.writeByte(Packet.HB_PACKET_BYTE);
    } else {
        out.writeInt(packet.getBodyLength());
        out.writeByte(packet.cmd);
        out.writeShort(packet.cc);
        out.writeByte(packet.flags);
        out.writeInt(packet.sessionId);
        out.writeByte(packet.lrc);
        if (packet.getBodyLength() > 0) {
            out.writeBytes(packet.body);
        }
    }
    packet.body = null;
}
```

解码器PacketDecoder

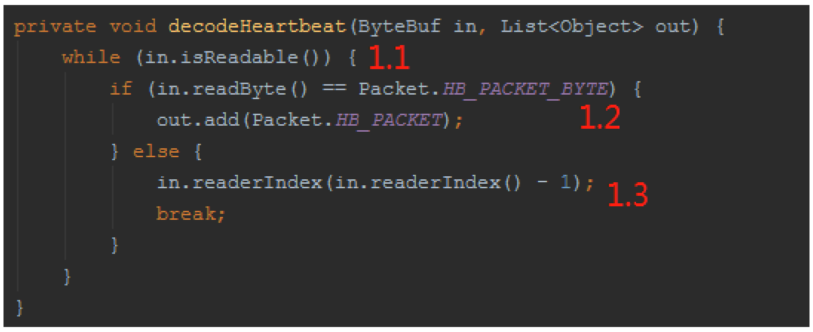


```java
public final class PacketDecoder extends ByteToMessageDecoder {
    private static final int maxPacketSize = CC.mp.core.max_packet_size;

    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws Exception {
        decodeHeartbeat(in, out); 1
        decodeFrames(in, out); 2
    }
```

ByteToMessageDecoder#channelRead方法中会调用子类的decode方法；

1、心跳消息解码

2、消息解码

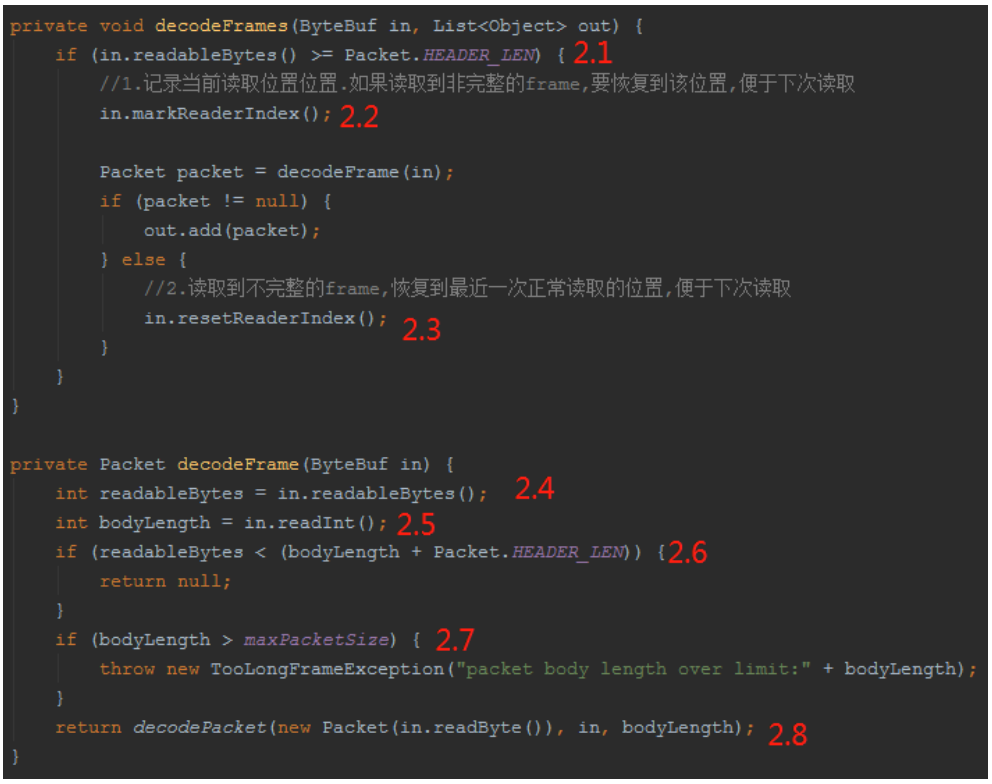


```java
private void decodeHeartbeat(ByteBuf in, List<Object> out) {
    while (in.isReadable()) { 1.1
        if (in.readByte() == Packet.HB_PACKET_BYTE) {
            out.add(Packet.HB_PACKET);          1.2
        } else {
            in.readerIndex(in.readerIndex() - 1); 1.3
            break;
        }
    }
}
```

1.1 是否有可读的数据

1.2 如果读取的byte数据是心跳，则加入List<object>

1.3 如果未读到心跳包，则readerIndex回到原来的位置(因为in.readByte()执行完index就增加了)



```
private void decodeFrames(ByteBuf in, List<Object> out) {
    if (in.readableBytes() >= Packet.HEADER_LEN) { 2.1
        //1.记录当前读取位置位置.如果读取到非完整的frame,要恢复到该位置,便于下次读取
        in.markReaderIndex(); 2.2

        Packet packet = decodeFrame(in);
        if (packet != null) {
            out.add(packet);
        } else {
            //2.读取到不完整的frame,恢复到最近一次正常读取的位置,便于下次读取
            in.resetReaderIndex();  2.3
        }
    }
}

private Packet decodeFrame(ByteBuf in) {
    int readableBytes = in.readableBytes();  2.4
    int bodyLength = in.readInt(); 2.5
    if (readableBytes < (bodyLength + Packet.HEADER_LEN)) {2.6
        return null;
    }
    if (bodyLength > maxPacketSize) { 2.7
        throw new TooLongFrameException("packet body length over limit:" + bodyLength);
    }
    return decodePacket(new Packet(in.readByte()), in, bodyLength); 2.8
}
```

2.1 如果可读字节长度大于头长度(13)，表示可以继续读取

2.2 先标记读取的位置

2.3 读的数据不完整，回到上面标记的位置

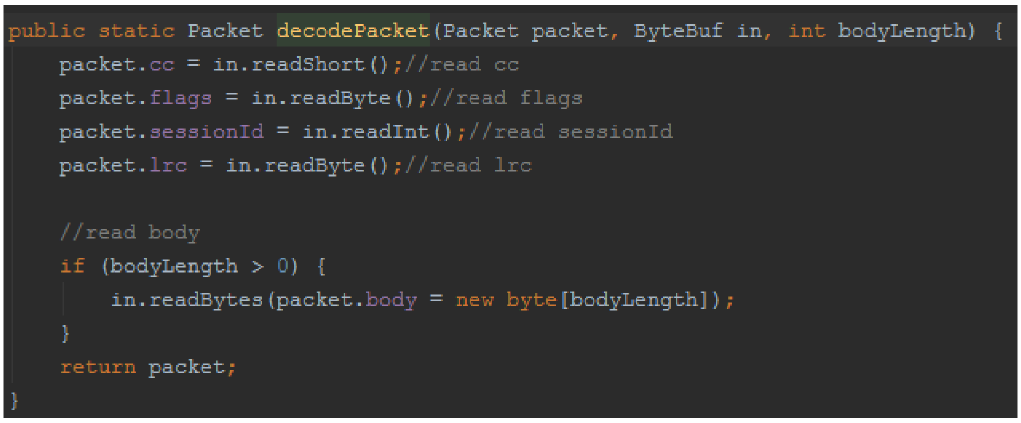
2.4 可读的字节长度

2.5 读取length字段

2.6 当ByteBuf没达到长度时，返回null

2.7 如果body长度超过设置的最大长度限制，则抛出异常

2.8 继续解码读取cmd、cc、flags、sesssionid、lrc、body



```
public static Packet decodePacket(Packet packet, ByteBuf in, int bodyLength) {
    packet.cc = in.readShort();//read cc
    packet.flags = in.readByte();//read flags
    packet.sessionId = in.readInt();//read sessionId
    packet.lrc = in.readByte();//read lrc

    //read body
    if (bodyLength > 0) {
        in.readBytes(packet.body = new byte[bodyLength]);
    }
    return packet;
}
```

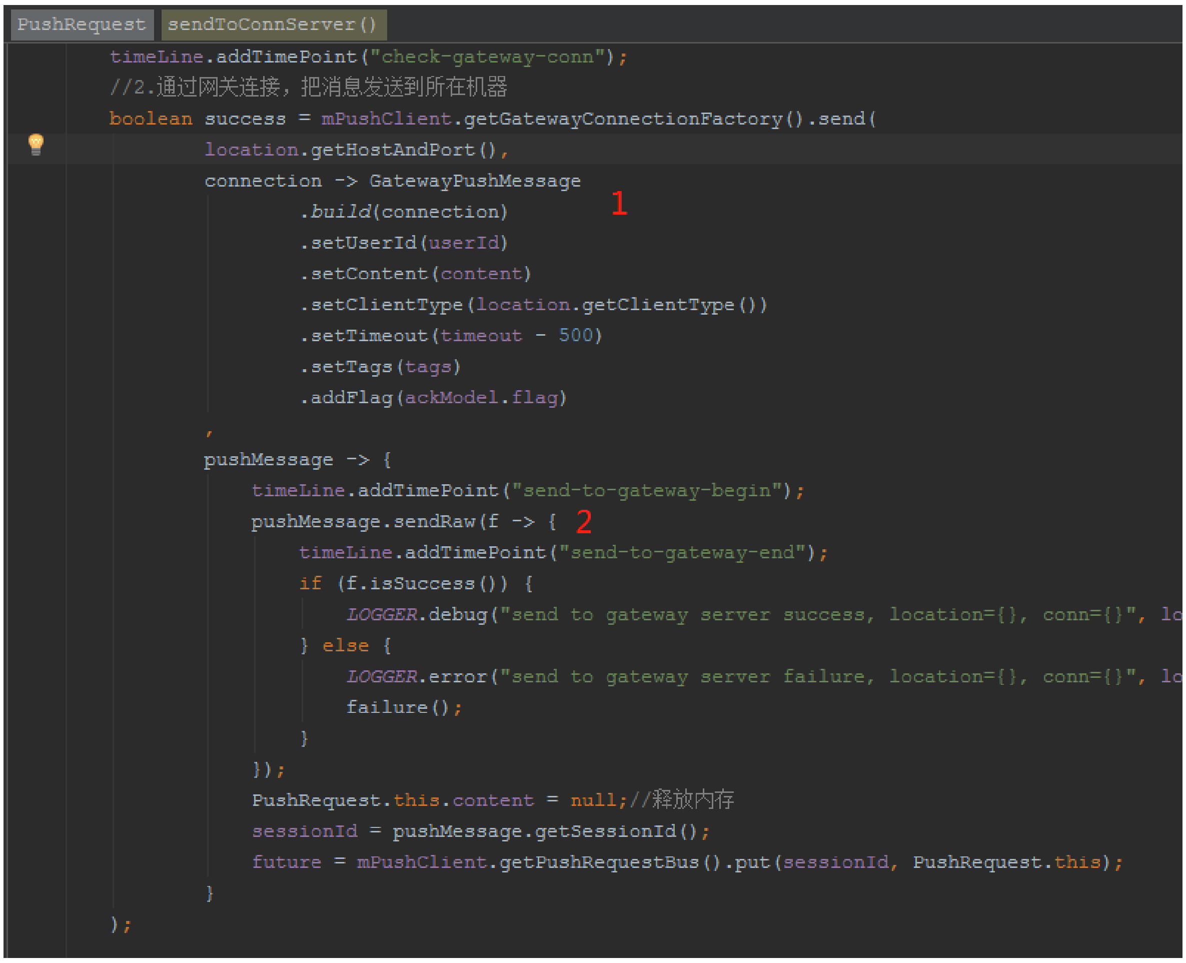
发送消息时编码

大体流程：先进行消息内容编码（body），channel.writerAndFlush()之后会经过PacketEncode编码器，进行header编码；

消息发送：

　　内容编码，body=userId+clientType+timeout+content+taskid+tags+condition

　　协议编码，packet= header+body，header=length+cmd+checkcode+flags+sesssionid+lrc

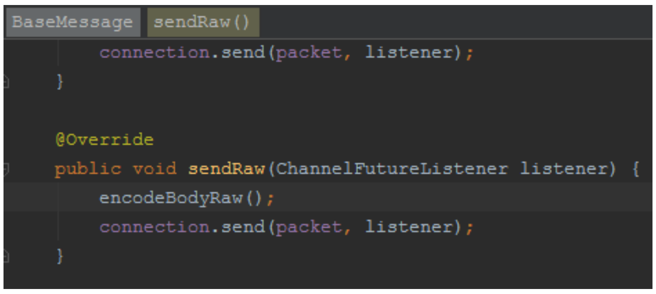


```
PushRequest  sendToConnServer()
        timeLine.addTimePoint("check-gateway-conn");
        //2.通过网关连接，把消息发送到所在机器
        boolean success = mPushClient.getGatewayConnectionFactory().send(
                location.getHostAndPort(),
                connection -> GatewayPushMessage
                        .build(connection)          1
                        .setUserId(userId)
                        .setContent(content)
                        .setClientType(location.getClientType())
                        .setTimeout(timeout - 500)
                        .setTags(tags)
                        .addFlag(ackModel.flag)
                ,
                pushMessage -> {
                    timeLine.addTimePoint("send-to-gateway-begin");
                    pushMessage.sendRaw(f -> {  2
                        timeLine.addTimePoint("send-to-gateway-end");
                        if (f.isSuccess()) {
                            LOGGER.debug("send to gateway server success, location={}, conn={}", lo
                        } else {
                            LOGGER.error("send to gateway server failure, location={}, conn={}", lo
                            failure();
                        }
                    });
                    PushRequest.this.content = null;//释放内存
                    sessionId = pushMessage.getSessionId();
                    future = mPushClient.getPushRequestBus().put(sessionId, PushRequest.this);
                }
        );
```

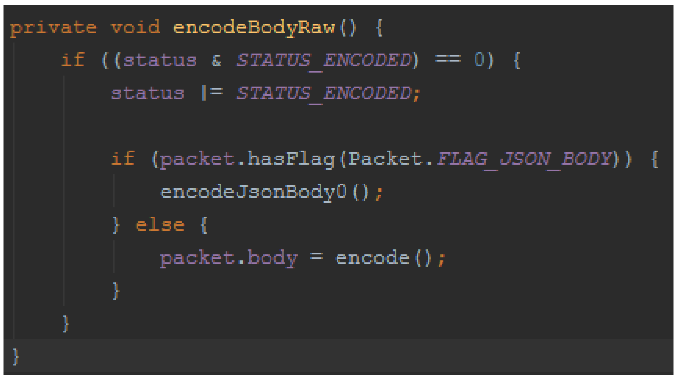
1、构建GatewayPushMessage实例，并把Packet、Connection传给父类

2、调用GatewayPushMessage->ByteBufMessage->BaseMessage#sendRaw方法

　　先把消息编码、然后再发送

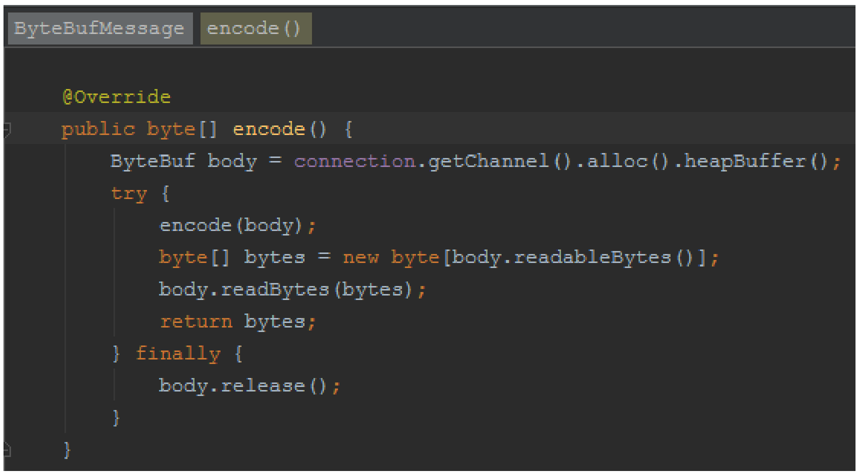


```
BaseMessage  sendRaw()
        connection.send(packet, listener);
    }

    @Override
    public void sendRaw(ChannelFutureListener listener) {
        encodeBodyRaw();
        connection.send(packet, listener);
    }
```



```java
private void encodeBodyRaw() {
    if ((status & STATUS_ENCODED) == 0) {
        status |= STATUS_ENCODED;

        if (packet.hasFlag(Packet.FLAG_JSON_BODY)) {
            encodeJsonBody0();
        } else {
            packet.body = encode();
        }
    }
}
```

调用子类ByteBufMessage#encode()方法；



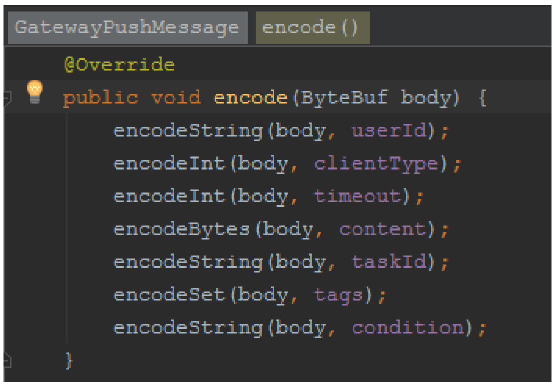
```java
ByteBufMessage  encode()

    @Override
    public byte[] encode() {
        ByteBuf body = connection.getChannel().alloc().heapBuffer();
        try {
            encode(body);
            byte[] bytes = new byte[body.readableBytes()];
            body.readBytes(bytes);
            return bytes;
        } finally {
            body.release();
        }
    }
```

1、在该channel上申请heapBuffer空间，用于子类写入消息内容(body)

2、继续调用子类GatewayPushMessage#encode()方法

将GatewayPushMessage#encode()写入ByteBuf中的内容转到byte[]数组中，并返回给父类BaseMessage#encodeBodyRaw()中:

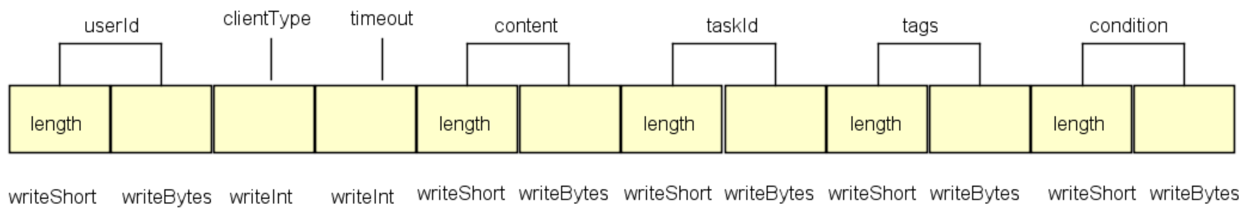
    packet.body=encode();



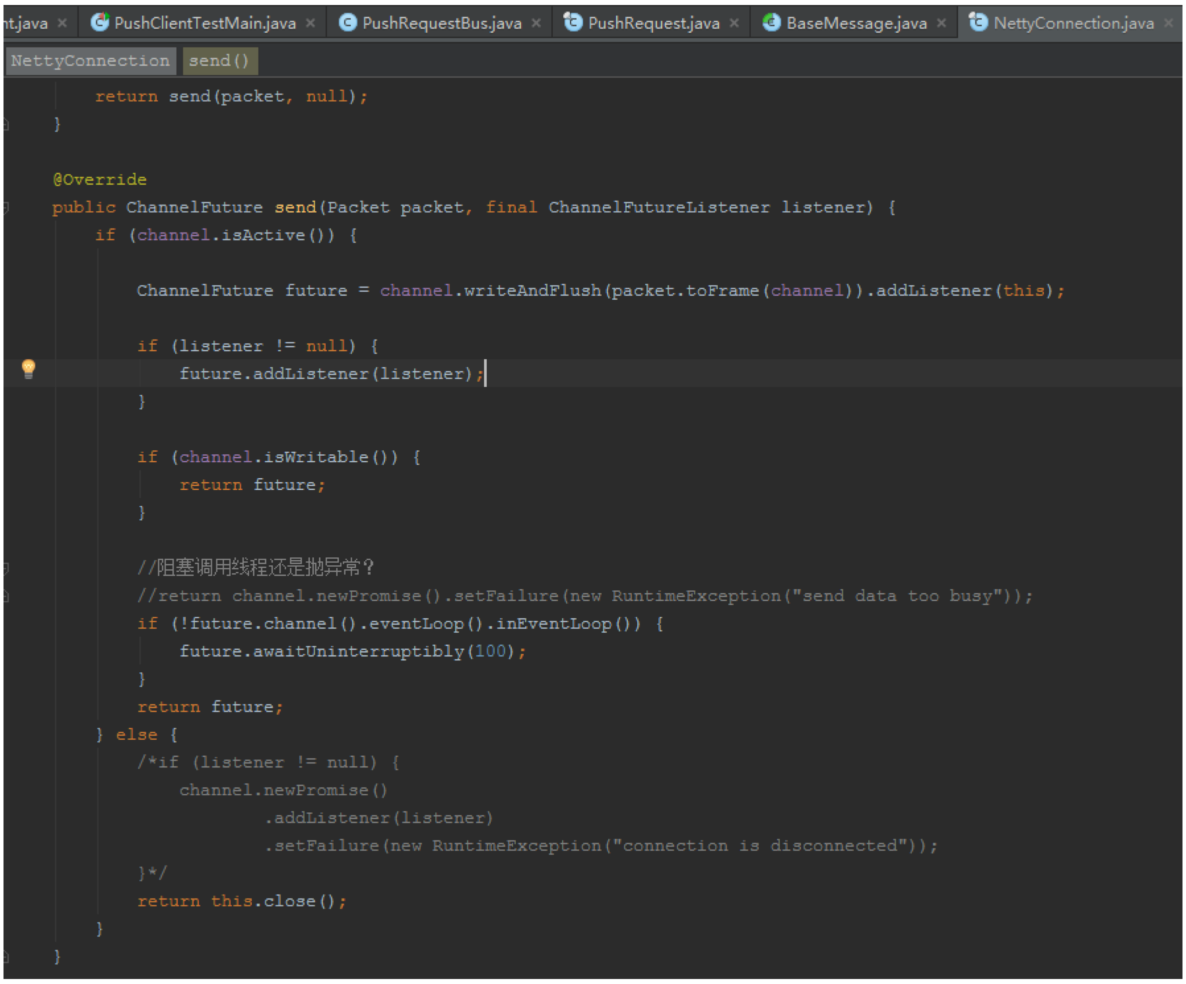
```java
GatewayPushMessage  encode()
    @Override
    public void encode(ByteBuf body) {
        encodeString(body, userId);
        encodeInt(body, clientType);
        encodeInt(body, timeout);
        encodeBytes(body, content);
        encodeString(body, taskId);
        encodeSet(body, tags);
        encodeString(body, condition);
    }
```

encodeString()、encodeInt()、encodeBytes()、encodeSet()都是其父类ByteBufMessage中的方法；

经过编码之后的字段，在buf中表现为：



至此，packet中的body已经全部封装准备完毕，调用下面的send()方法发送消息；
当调用writeAndFlush方法，packet对象会经过编码器PacketEncoder#encode方法，封装header头；
最终packet中的header+body全部封装完毕，然后发送出去；



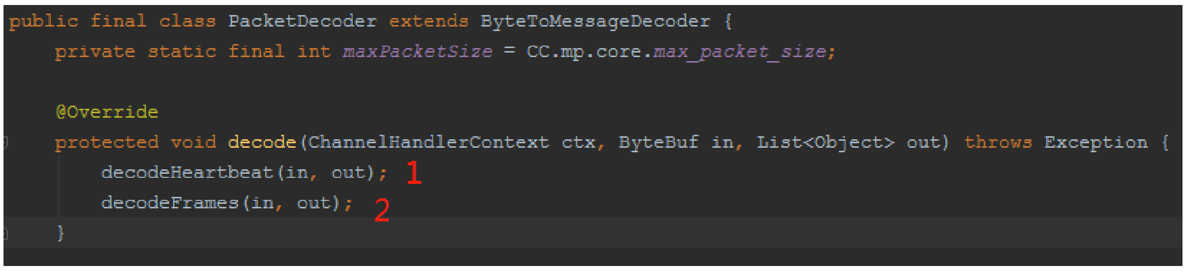
```java
        return send(packet, null);
    }

    @Override
    public ChannelFuture send(Packet packet, final ChannelFutureListener listener) {
        if (channel.isActive()) {

            ChannelFuture future = channel.writeAndFlush(packet.toFrame(channel)).addListener(this);

            if (listener != null) {
                future.addListener(listener);
            }

            if (channel.isWritable()) {
                return future;
            }

            //阻塞调用线程还是抛异常？
            //return channel.newPromise().setFailure(new RuntimeException("send data too busy"));
            if (!future.channel().eventLoop().inEventLoop()) {
                future.awaitUninterruptibly(100);
            }
            return future;
        } else {
            /*if (listener != null) {
                channel.newPromise()
                        .addListener(listener)
                        .setFailure(new RuntimeException("connection is disconnected"));
            }*/
            return this.close();
        }
    }
```

接收消息时解码

大体流程：接收到消息，首先会经过PacketDecode解码器，进行消息header解码；完了之后，会流转到handler处理器中的channelRead()方法；
消息接收：

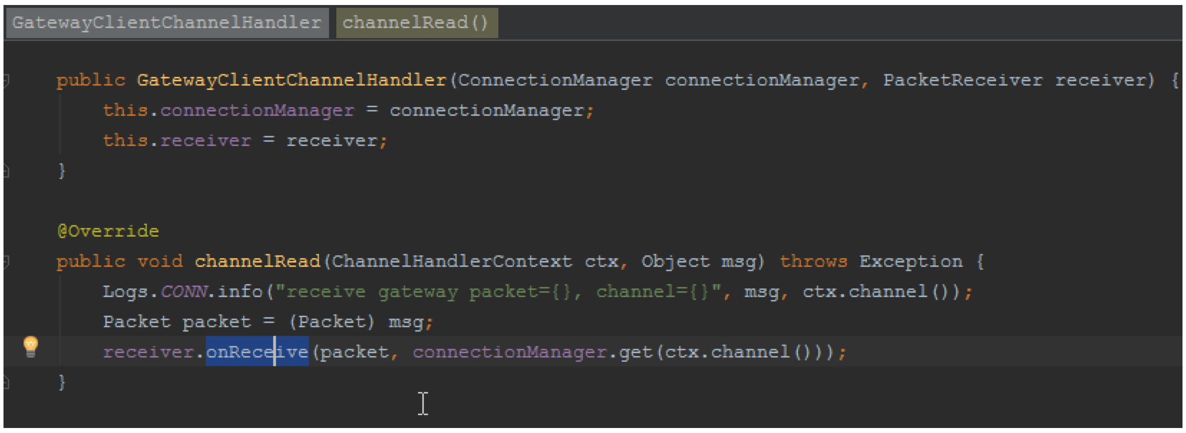
　　协议解码，解出header、body(此时还是byte)

　　内容解码，解出body中详细字段



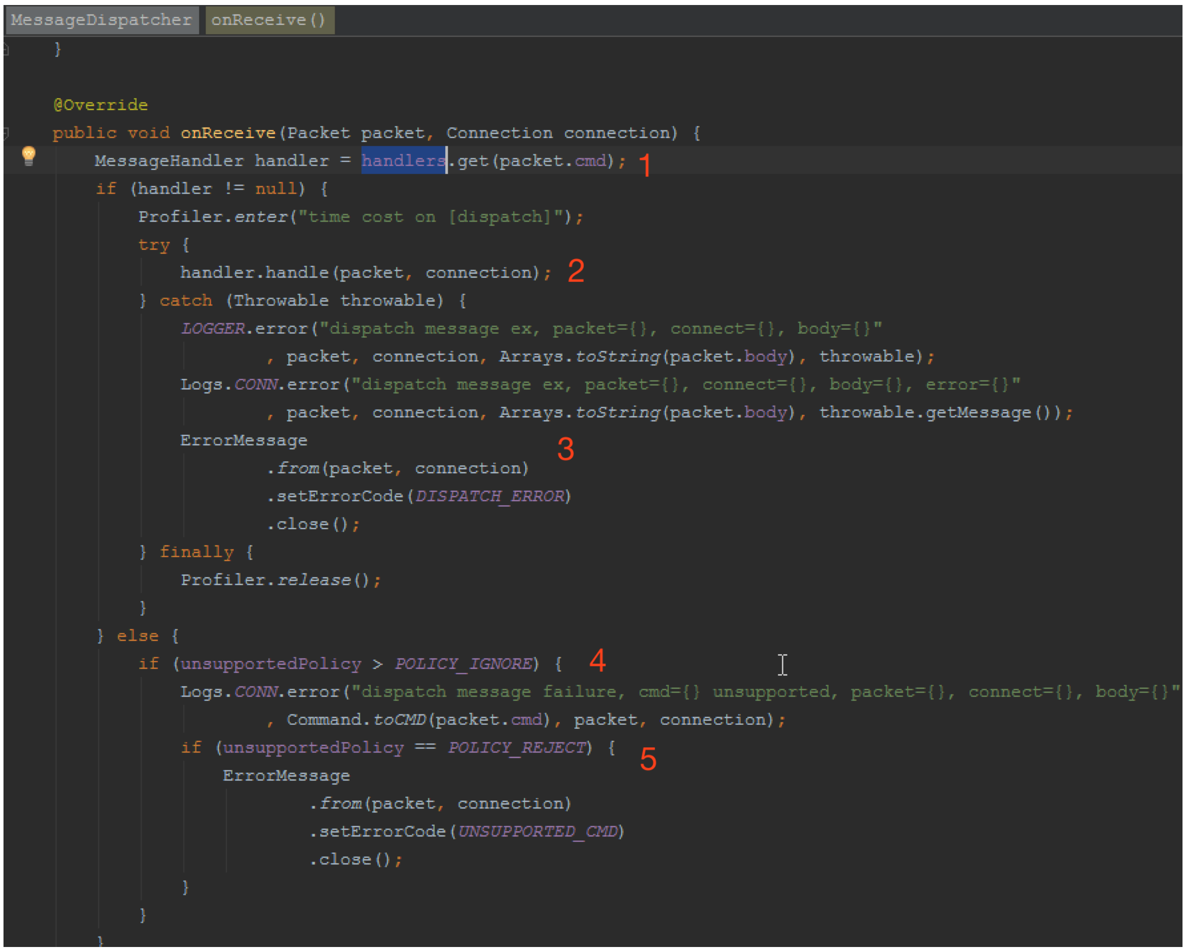
```java
public final class PacketDecoder extends ByteToMessageDecoder {
    private static final int maxPacketSize = CC.mp.core.max_packet_size;

    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws Exception {
        decodeHeartbeat(in, out); 1
        decodeFrames(in, out); 2
    }
```

消息首先经过PacketDecode解码器，进行消息header解码(这里心跳消息是单独分开的)，header解码之后封装成packet对象，添加到out集合中，用于传递给handler处理类GatewayClientChannelHandler#channelRead方法；


GatewayClientChannelHandler   channelRead()

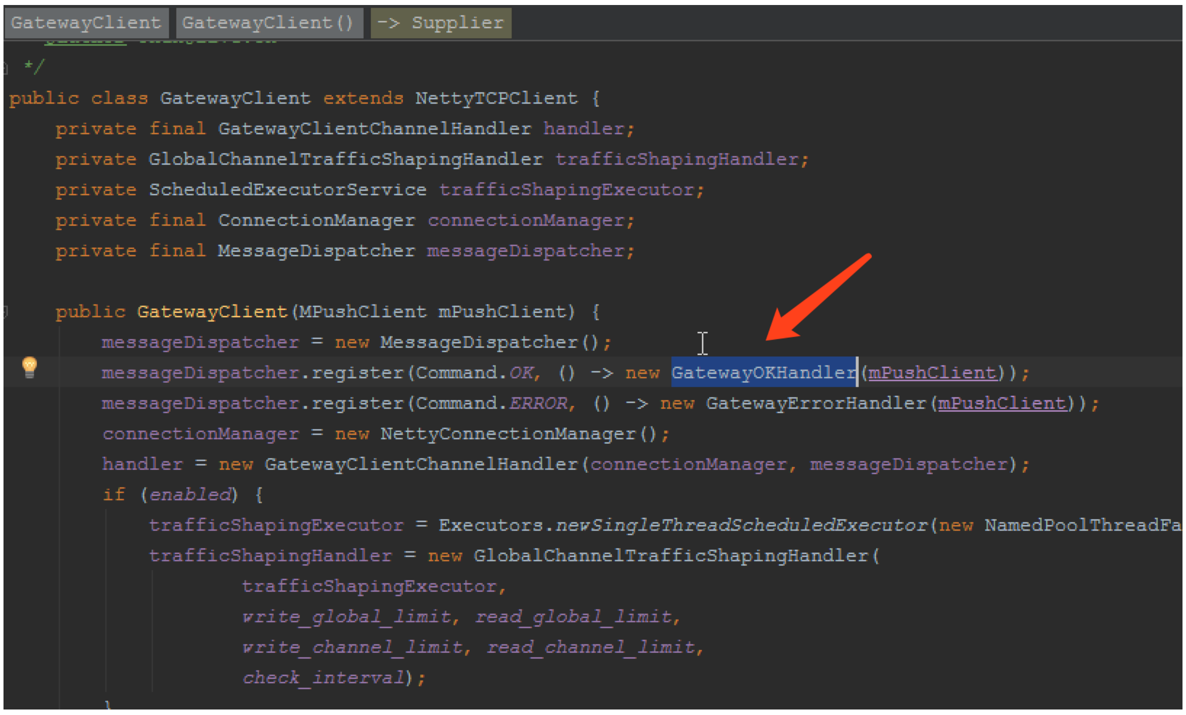
```java
    public GatewayClientChannelHandler(ConnectionManager connectionManager, PacketReceiver receiver) {
        this.connectionManager = connectionManager;
        this.receiver = receiver;
    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        Logs.CONN.info("receive gateway packet={}, channel={}", msg, ctx.channel());
        Packet packet = (Packet) msg;
        receiver.onReceive(packet, connectionManager.get(ctx.channel()));
    }
```

channelRead方法会调用MeassgeDispacher#onReceive()方法，找到相应的handler处理类进一步的对body内的字段进行解码；


MessageDispatcher   onReceive()

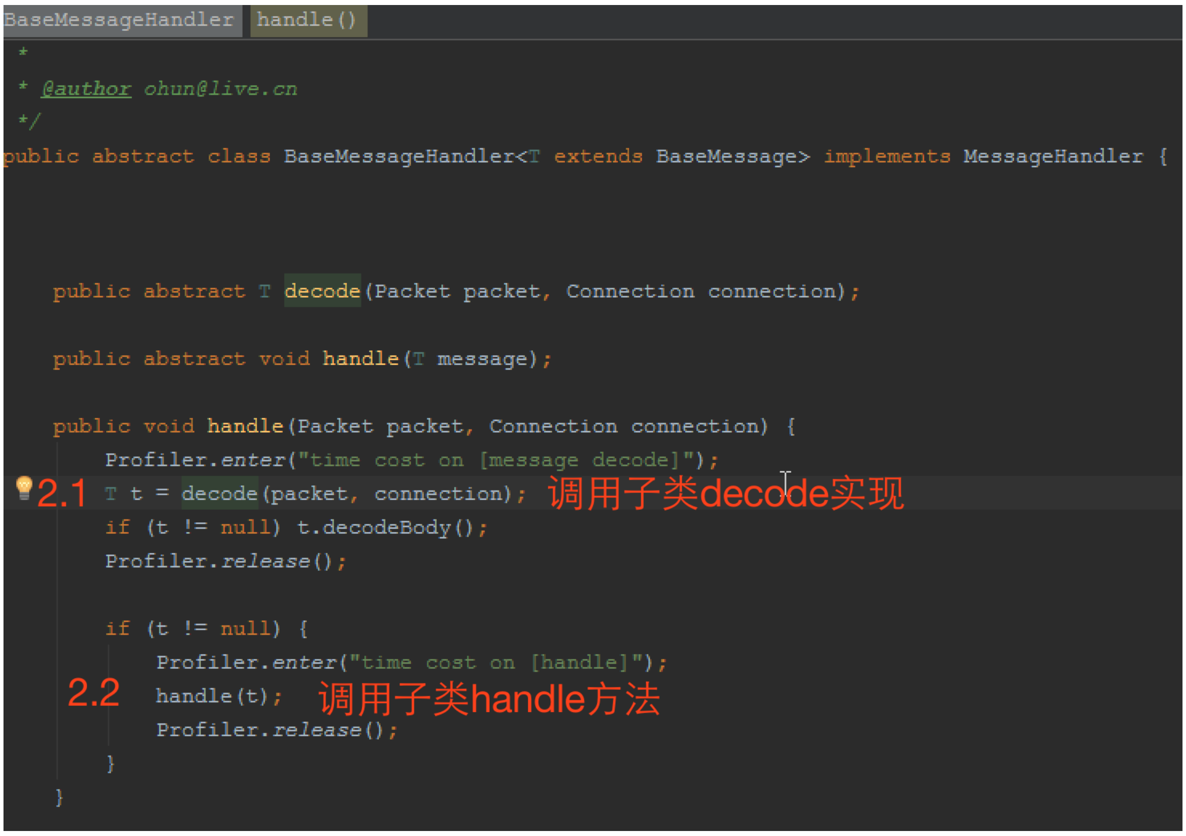
```java
    }

    @Override
    public void onReceive(Packet packet, Connection connection) {
        MessageHandler handler = handlers.get(packet.cmd); 1
        if (handler != null) {
            Profiler.enter("time cost on [dispatch]");
            try {
                handler.handle(packet, connection); 2
            } catch (Throwable throwable) {
                LOGGER.error("dispatch message ex, packet={}, connect={}, body={}"
                        , packet, connection, Arrays.toString(packet.body), throwable);
                Logs.CONN.error("dispatch message ex, packet={}, connect={}, body={}, error={}"
                        , packet, connection, Arrays.toString(packet.body), throwable.getMessage());
                ErrorMessage
                        .from(packet, connection)  3
                        .setErrorCode(DISPATCH_ERROR)
                        .close();
            } finally {
                Profiler.release();
            }
        } else {
            if (unsupportedPolicy > POLICY_IGNORE) { 4
                Logs.CONN.error("dispatch message failure, cmd={} unsupported, packet={}, connect={}, body={}"
                        , Command.toCMD(packet.cmd), packet, connection);
                if (unsupportedPolicy == POLICY_REJECT) { 5
                    ErrorMessage
                            .from(packet, connection)
                            .setErrorCode(UNSUPPORTED_CMD)
                            .close();
                }
            }
        }
```

1、根据packet.cmd找到对应的handler，这里handlers集合里面的处理类是在GatewayClient中注册的



```
GatewayClient   GatewayClient()   -> Supplier
 */
public class GatewayClient extends NettyTCPClient {
    private final GatewayClientChannelHandler handler;
    private GlobalChannelTrafficShapingHandler trafficShapingHandler;
    private ScheduledExecutorService trafficShapingExecutor;
    private final ConnectionManager connectionManager;
    private final MessageDispatcher messageDispatcher;

    public GatewayClient(MPushClient mPushClient) {
        messageDispatcher = new MessageDispatcher();
        messageDispatcher.register(Command.OK, () -> new GatewayOKHandler(mPushClient));
        messageDispatcher.register(Command.ERROR, () -> new GatewayErrorHandler(mPushClient));
        connectionManager = new NettyConnectionManager();
        handler = new GatewayClientChannelHandler(connectionManager, messageDispatcher);
        if (enabled) {
            trafficShapingExecutor = Executors.newSingleThreadScheduledExecutor(new NamedPoolThreadFa
            trafficShapingHandler = new GlobalChannelTrafficShapingHandler(
                    trafficShapingExecutor,
                    write_global_limit, read_global_limit,
                    write_channel_limit, read_channel_limit,
                    check_interval);
```

2、调用GatewayOKHandler或者GatewayErrorHandler的父类中BaseMessageHandler#handler()方法

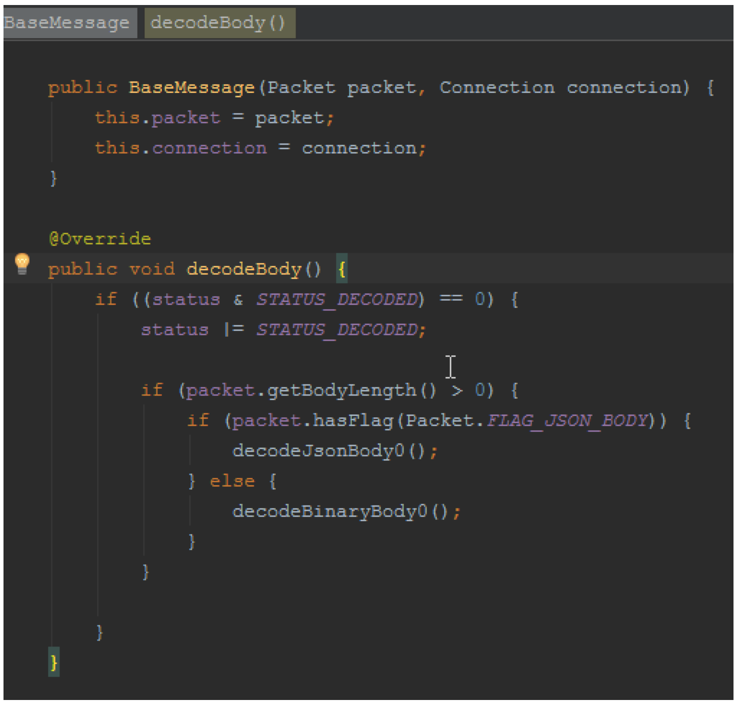


```
BaseMessageHandler   handle()
 *
 * @author ohun@live.cn
 */
public abstract class BaseMessageHandler<T extends BaseMessage> implements MessageHandler {


    public abstract T decode(Packet packet, Connection connection);

    public abstract void handle(T message);

    public void handle(Packet packet, Connection connection) {
        Profiler.enter("time cost on [message decode]");
2.1     T t = decode(packet, connection);  调用子类decode实现
        if (t != null) t.decodeBody();
        Profiler.release();

        if (t != null) {
            Profiler.enter("time cost on [handle]");
2.2         handle(t);  调用子类handle方法
            Profiler.release();
        }
    }
```

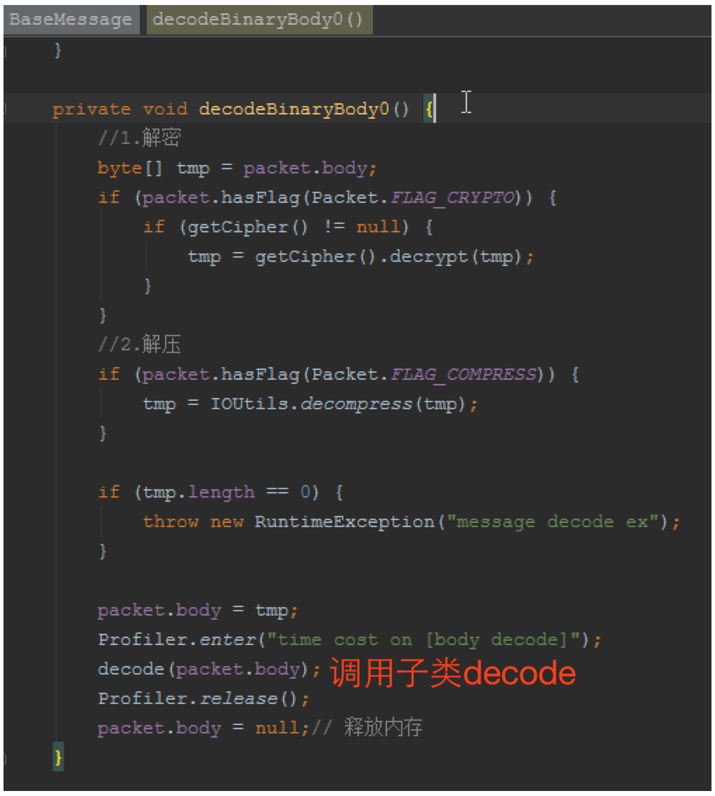
2.1 调用GatewayOKHandler或者GatewayErrorHandler的decode方法获得OKMessage或者ErrorMessage实例，然后再调用其decodeBody()方法解码；

OkMessage->ByteBufMessage->BaseMessage

ErrorMessage->ByteBufMessage->BaseMessage

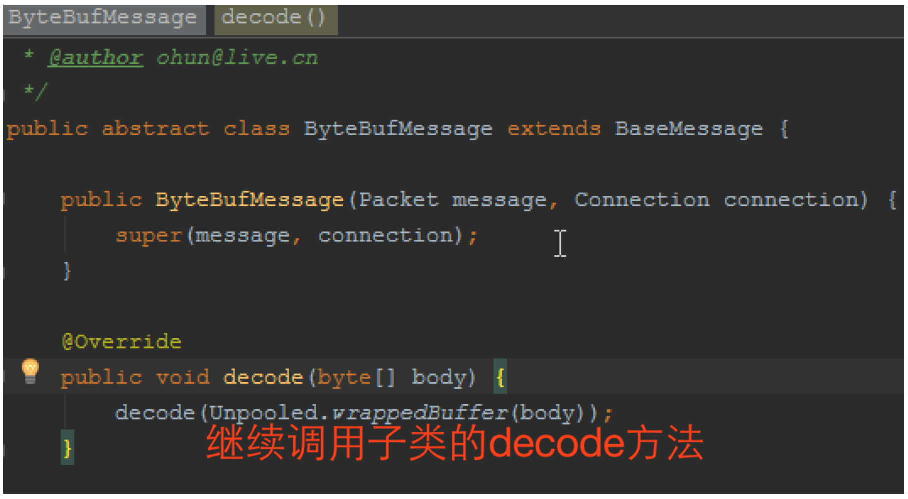


```
public BaseMessage(Packet packet, Connection connection) {
    this.packet = packet;
    this.connection = connection;
}

@Override
public void decodeBody() {
    if ((status & STATUS_DECODED) == 0) {
        status |= STATUS_DECODED;

        if (packet.getBodyLength() > 0) {
            if (packet.hasFlag(Packet.FLAG_JSON_BODY)) {
                decodeJsonBody0();
            } else {
                decodeBinaryBody0();
            }
        }

    }
}
```



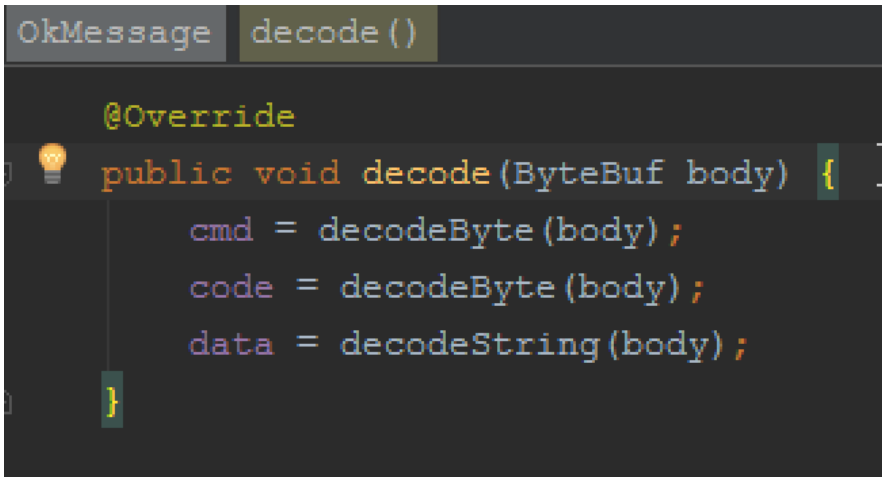
```
    }

    private void decodeBinaryBody0() {
        //1.解密
        byte[] tmp = packet.body;
        if (packet.hasFlag(Packet.FLAG_CRYPTO)) {
            if (getCipher() != null) {
                tmp = getCipher().decrypt(tmp);
            }
        }
        //2.解压
        if (packet.hasFlag(Packet.FLAG_COMPRESS)) {
            tmp = IOUtils.decompress(tmp);
        }

        if (tmp.length == 0) {
            throw new RuntimeException("message decode ex");
        }

        packet.body = tmp;
        Profiler.enter("time cost on [body decode]");
        decode(packet.body); 调用子类decode
        Profiler.release();
        packet.body = null;// 释放内存
    }
```

调用子类ByteBufMessage#decode()



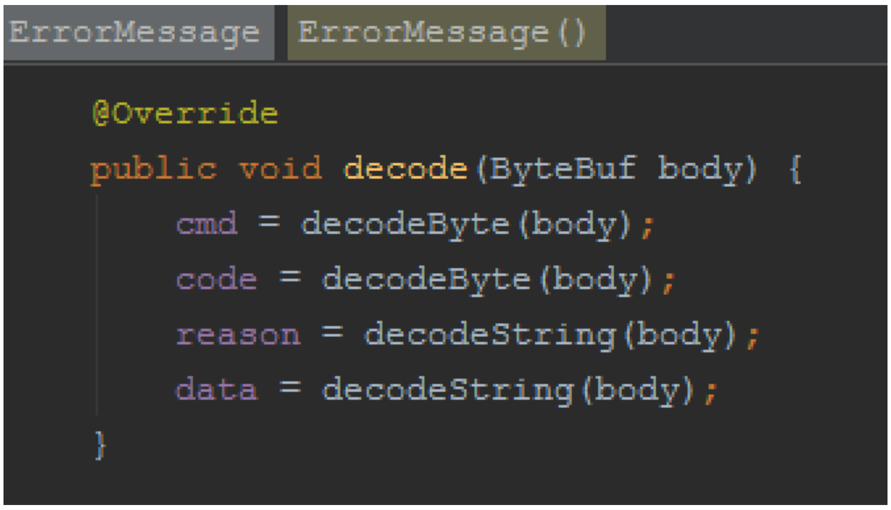
```
ByteBufMessage  decode()
 * @author ohun@live.cn
 */
public abstract class ByteBufMessage extends BaseMessage {

    public ByteBufMessage(Packet message, Connection connection) {
        super(message, connection);
    }

    @Override
    public void decode(byte[] body) {
        decode(Unpooled.wrappedBuffer(body));
    }
```

OKMessage#decode()



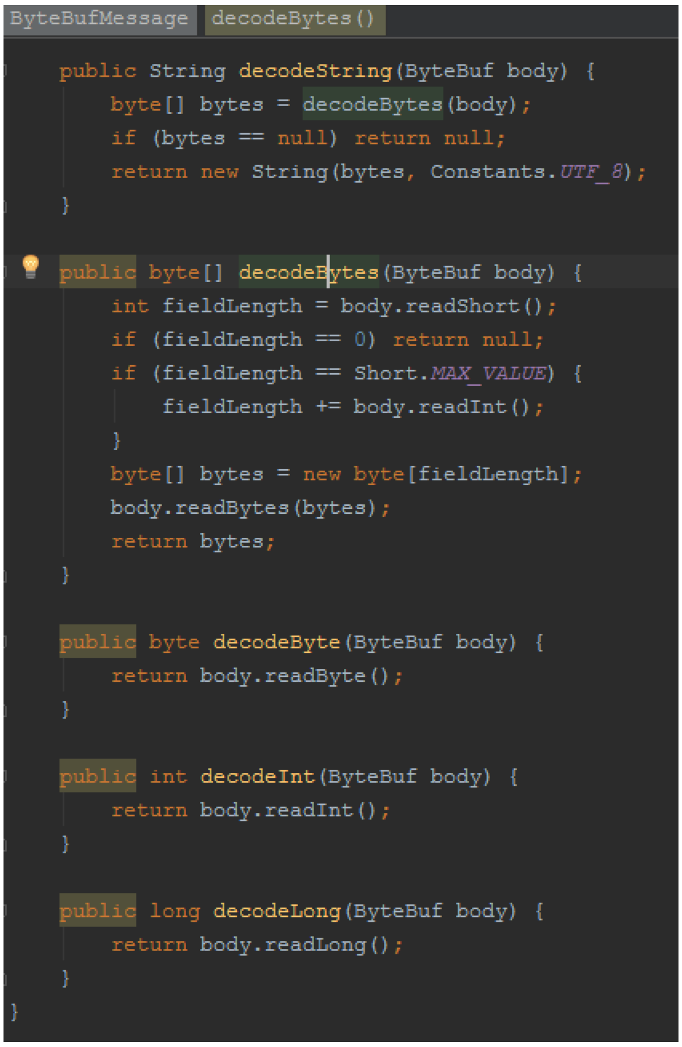
```
OkMessage  decode()

    @Override
    public void decode(ByteBuf body) {
        cmd = decodeByte(body);
        code = decodeByte(body);
        data = decodeString(body);
    }
```

ErrorMessage#decode()



```
ErrorMessage  ErrorMessage()

    @Override
    public void decode(ByteBuf body) {
        cmd = decodeByte(body);
        code = decodeByte(body);
        reason = decodeString(body);
        data = decodeString(body);
    }
```

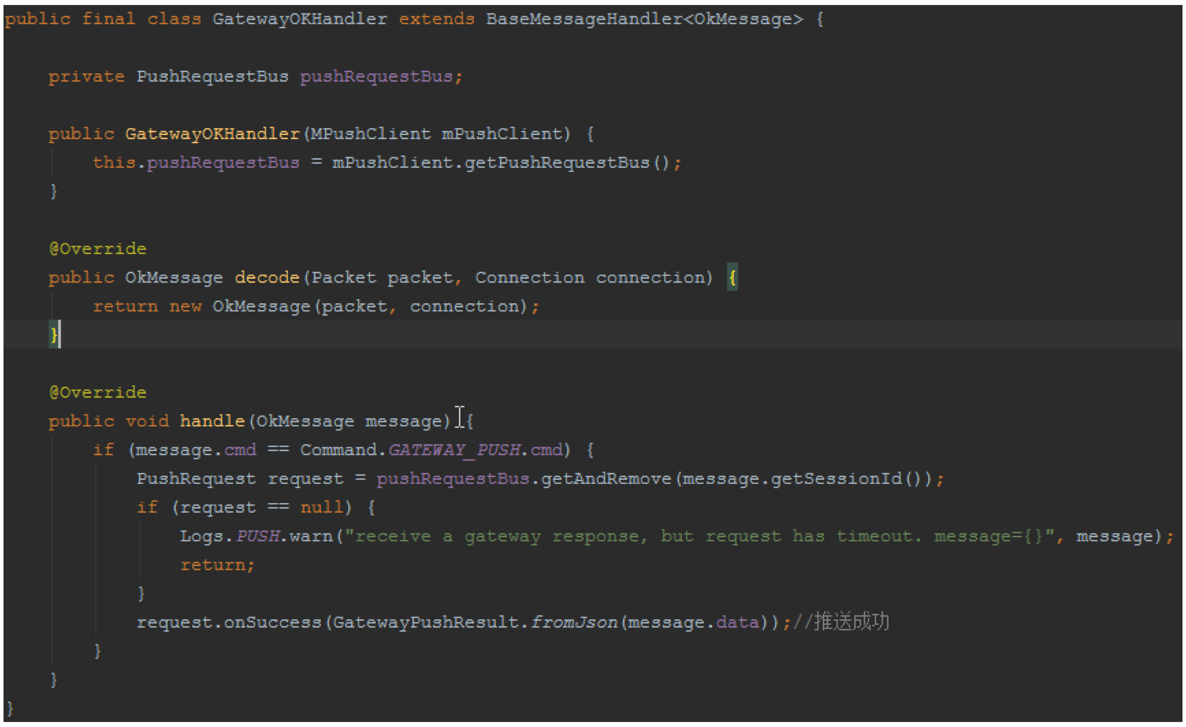
body中的cmd/code/data字段，由netty server端返回



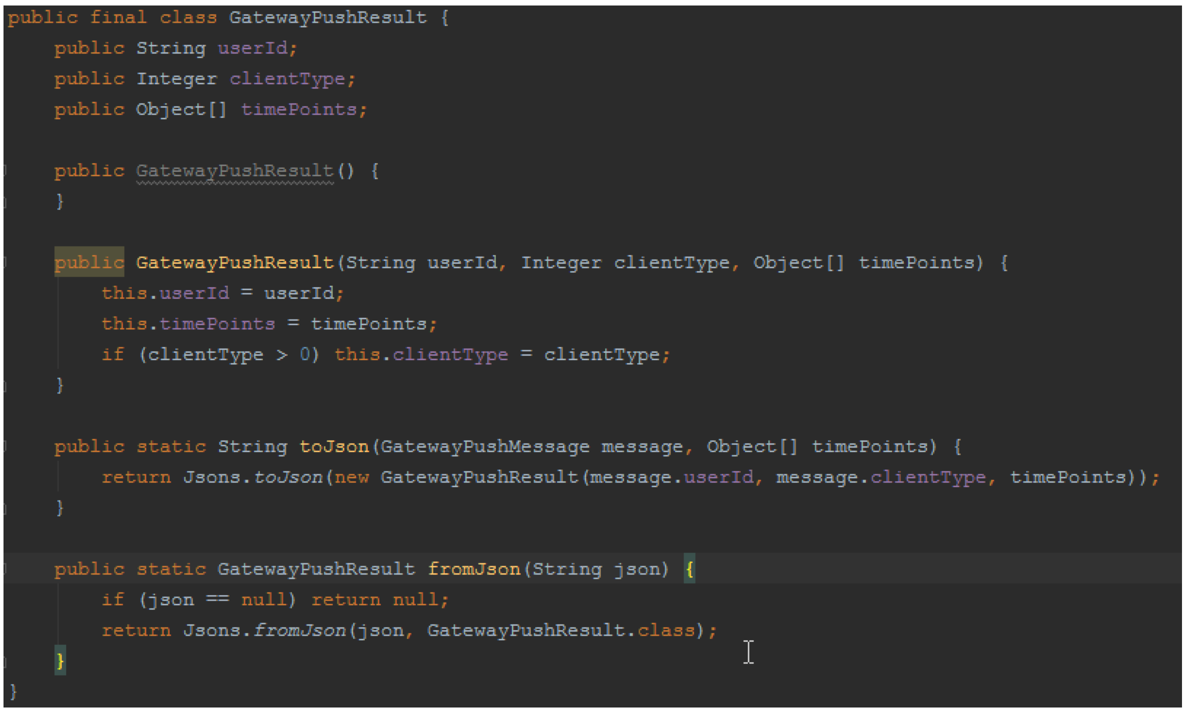
ByteBufMessage decodeBytes()

```java
    public String decodeString(ByteBuf body) {
        byte[] bytes = decodeBytes(body);
        if (bytes == null) return null;
        return new String(bytes, Constants.UTF_8);
    }

    public byte[] decodeBytes(ByteBuf body) {
        int fieldLength = body.readShort();
        if (fieldLength == 0) return null;
        if (fieldLength == Short.MAX_VALUE) {
            fieldLength += body.readInt();
        }
        byte[] bytes = new byte[fieldLength];
        body.readBytes(bytes);
        return bytes;
    }

    public byte decodeByte(ByteBuf body) {
        return body.readByte();
    }

    public int decodeInt(ByteBuf body) {
        return body.readInt();
    }

    public long decodeLong(ByteBuf body) {
        return body.readLong();
    }
}
```

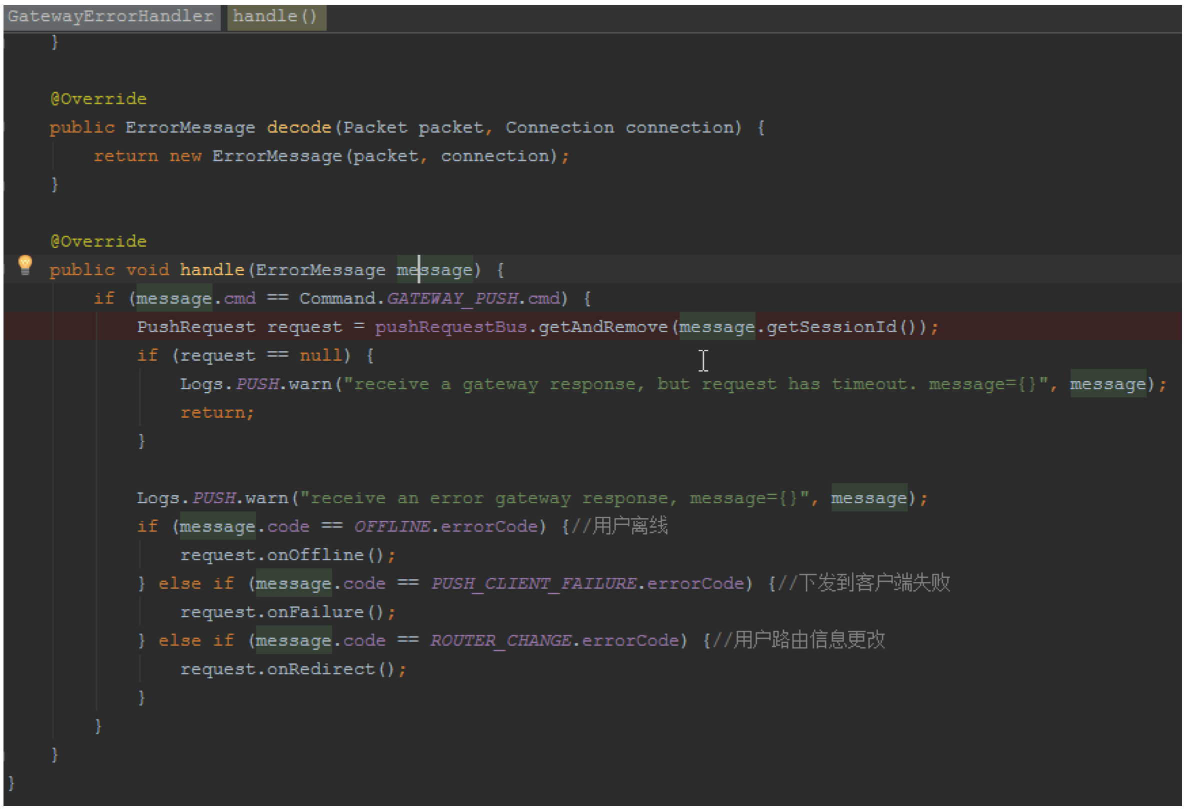
2.2 调用GatewayOKHandler或者GatewayErrorHandler的handle方法



```java
public final class GatewayOKHandler extends BaseMessageHandler<OkMessage> {

    private PushRequestBus pushRequestBus;

    public GatewayOKHandler(MPushClient mPushClient) {
        this.pushRequestBus = mPushClient.getPushRequestBus();
    }

    @Override
    public OkMessage decode(Packet packet, Connection connection) {
        return new OkMessage(packet, connection);
    }

    @Override
    public void handle(OkMessage message) {
        if (message.cmd == Command.GATEWAY_PUSH.cmd) {
            PushRequest request = pushRequestBus.getAndRemove(message.getSessionId());
            if (request == null) {
                Logs.PUSH.warn("receive a gateway response, but request has timeout. message={}", message);
                return;
            }
            request.onSuccess(GatewayPushResult.fromJson(message.data));//推送成功
        }
    }
}
```



```java
public final class GatewayPushResult {
    public String userId;
    public Integer clientType;
    public Object[] timePoints;

    public GatewayPushResult() {
    }

    public GatewayPushResult(String userId, Integer clientType, Object[] timePoints) {
        this.userId = userId;
        this.timePoints = timePoints;
        if (clientType > 0) this.clientType = clientType;
    }

    public static String toJson(GatewayPushMessage message, Object[] timePoints) {
        return Jsons.toJson(new GatewayPushResult(message.userId, message.clientType, timePoints));
    }

    public static GatewayPushResult fromJson(String json) {
        if (json == null) return null;
        return Jsons.fromJson(json, GatewayPushResult.class);
    }
}
```



GatewayErrorHandler handle()

```java
    }

    @Override
    public ErrorMessage decode(Packet packet, Connection connection) {
        return new ErrorMessage(packet, connection);
    }

    @Override
    public void handle(ErrorMessage message) {
        if (message.cmd == Command.GATEWAY_PUSH.cmd) {
            PushRequest request = pushRequestBus.getAndRemove(message.getSessionId());
            if (request == null) {
                Logs.PUSH.warn("receive a gateway response, but request has timeout. message={}", message);
                return;
            }

            Logs.PUSH.warn("receive an error gateway response, message={}", message);
            if (message.code == OFFLINE.errorCode) {//用户离线
                request.onOffline();
            } else if (message.code == PUSH_CLIENT_FAILURE.errorCode) {//下发到客户端失败
                request.onFailure();
            } else if (message.code == ROUTER_CHANGE.errorCode) {//用户路由信息更改
                request.onRedirect();
            }
        }
    }
}
```