

## 章节

- 消息正常发送
- 消息发送超时
- 接收消息，并回调
- 用户离线
- 发送失败(发送时、netty处理返回时)
- 发送广播消息

准备工作：

- 1、运行com.mpush.test.sever.ServerTestMain.java启动长链接服务
- 2、运行com.mpush.test.client.ConnClientTestMain.java 模拟一个客户端
- 3、运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息

## 消息正常发送

pushMessage.sendRaw()方法体中的f为异步操作完成时调用；

会先执行后面的语句：PushRequest.this.content = null;//释放内存

创建超时任务（见"通信模型与超时控制"文章）

PushRequest.java

```

PushRequest sendToConnServer() -> Consumer -> ChannelFutureListener
    offline();
    return;
}

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( mPushClient: MPushClient@2628
    location.getHostAndPort(),
    connection -> GatewayPushMessage
        .build(connection)
        .setUserId(userId) userId: "user-0"
        .setContent(content) content: null
        .setClientType(location.getClientType()) location: "ClientLocation{host='172.16.177.134:30
        .setTimeout(timeout - 500) timeout: 2000
        .setTags(tags) tags: null
        .addFlag(ackModel.flag) ackModel: "AUTO_ACK"
    ,
    pushMessage -> {
        timeline.addTimePoint("send-to-gateway-begin");
        pushMessage.sendRaw(f -> { f: "DefaultChannelPromise@13e30ff0(success)"
        timeline.addTimePoint("send-to-gateway-end"); timeline: "Push-Time-Line[430918](ms){begin
        if (f.isSuccess()) {
            LOGGER.debug("send to gateway server success, location={}, conn={}", location, f.channe
        } else {
            LOGGER.error("send to gateway server failure, location={}, conn={}", location, f.channe
            failure();
        }
    });
    PushRequest.this.content = null; //释放内存
    sessionId = pushMessage.getSessionId();
    future = mPushClient.getPushRequestBus().put(sessionId, PushRequest.this);
}
}

```

## BaseMessage.java

```

BaseMessage sendRaw()

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

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

public void send() {
    send(ChannelFutureListener.FIRE_EXCEPTION_ON_FAILURE);
}

public void sendRaw() {
    sendRaw(ChannelFutureListener.FIRE_EXCEPTION_ON_FAILURE);
}

public void close() {
    send(ChannelFutureListener.CLOSE);
}

```

## NettyConnetion.java

```
NettyConnection send()
    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);

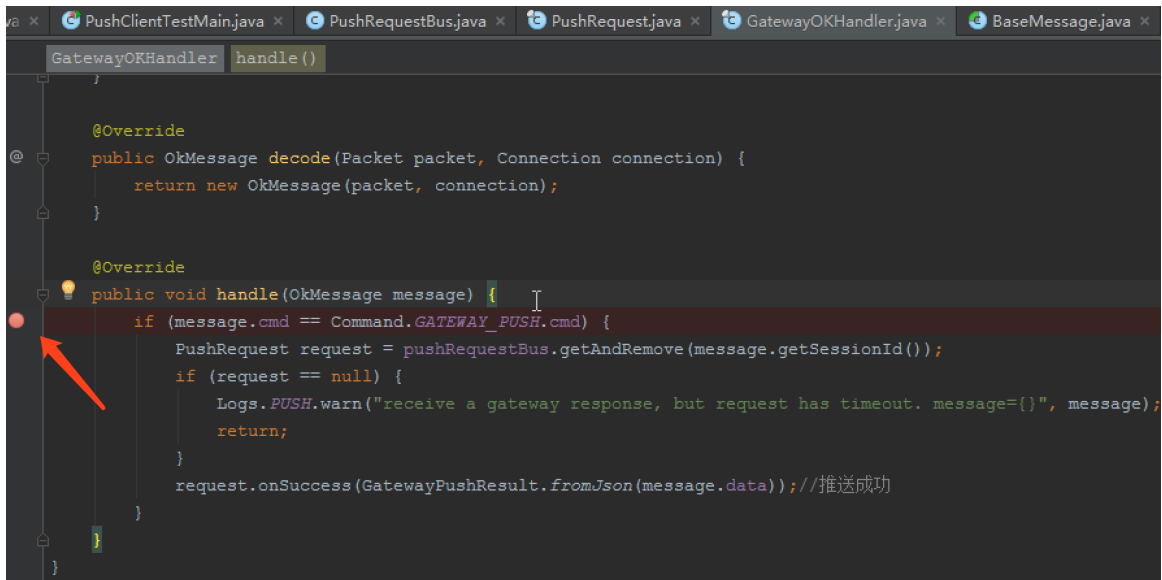
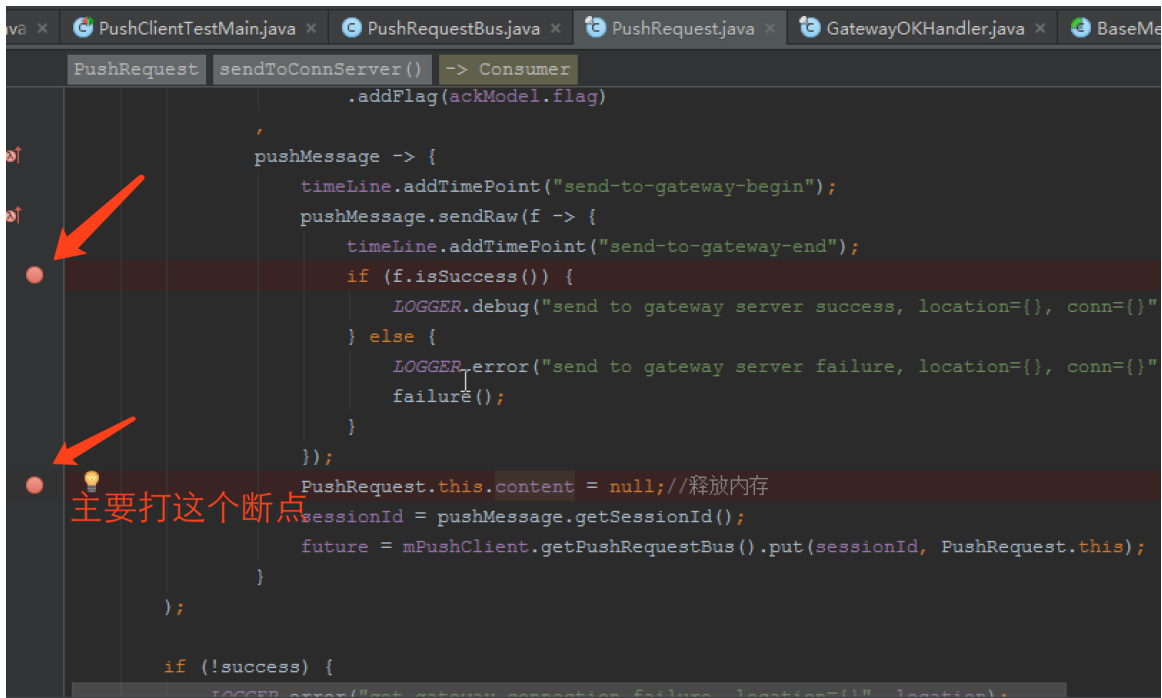
        if (listener != null) {
            future.addListener(listener);
        }

        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();
    }
}
```

## 消息发送超时

- 1、先打断点
- 2、运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息
- 3、然后程序会运行在这条语句：PushRequest.this.content = null;//释放内存
- 4、断点下一步执行完这条语句 future = mPushClient...，然后等待5S以上
- 5、点击最左边的绿按钮，即跳转到下一个断点处：handle(okMessage message)



## 接收消息

接收netty返回消息，由handle(OkMessage message)方法处理;

异步回调（见"通信模型与超时控制"文章）;

```
andler.java × GatewayPushResult.java × BaseMessageHandler.java × MessageDispatcher.java × GatewayClientChannelHandler.java ×
GatewayClientChannelHandler channelRead()
 * Created by ohun on 2015/12/19.
 *
 * @author ohun@live.cn
 */
@ChannelHandler.Sharable
public final class GatewayClientChannelHandler extends ChannelInboundHandlerAdapter {

    private static final Logger LOGGER = LoggerFactory.getLogger(GatewayClientChannelHandler.class);

    private final ConnectionManager connectionManager;

    private final PacketReceiver receiver;

    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()));
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
        Connection connection = connectionManager.get(ctx.channel());
        Logs.CONN.error("client caught ex, conn={}", connection);
        LOGGER.error("caught an ex, channel={}, conn={}", ctx.channel(), connection, cause);
        ctx.close();
    }
}
```

andler.java × GatewayPushResult.java × BaseMessageHandler.java × MessageDispatcher.java × Gateway

MessageDispatcher onReceive()

```

    }

    public MessageHandler unregister(Command command) {
        return handlers.remove(command.cmd);
    }

    @Override
    public void onReceive(Packet packet, Connection connection) {
        MessageHandler handler = handlers.get(packet.cmd);
        if (handler != null) {
            Profiler.enter("time cost on [dispatch]");
            try {
                handler.handle(packet, connection);
            } 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)
                    .setErrorCode(DISPATCH_ERROR)
                    .close();
            } finally {
                Profiler.release();
            }
        } else {
            if (unsupportedPolicy > POLICY_IGNORE) {
                Logs.CONN.error("dispatch message failure, cmd={} unsupported, packet={}, c"
                    , Command.toCMD(packet.cmd), packet, connection);
                if (unsupportedPolicy == POLICY_REJECT) {
                    ErrorMessage
                        .from(packet, connection)
                        .setErrorCode(UNSUPPORTED_CMD)

```

5: Debug 6: TO DO

```
GatewayOKHandler.java x GatewayPushResult.java x BaseMessageHandler.java x MessageDispatcher.java x
GatewayOKHandler handle()
35 /**
36  * Created by ohun on 16/10/21.
37  *
38  * @author ohun@live.cn (夜色)
39  */
40 public final class GatewayOKHandler extends BaseMessageHandler<OkMessage> {
41
42     private PushRequestBus pushRequestBus;
43
44     public GatewayOKHandler(MPushClient mPushClient) {
45         this.pushRequestBus = mPushClient.getPushRequestBus();
46     }
47
48     @Override
49     public OkMessage decode(Packet packet, Connection connection) {
50         return new OkMessage(packet, connection);
51     }
52
53     @Override
54     public void handle(OkMessage message) {
55         if (message.cmd == Command.GATEWAY_PUSH.cmd) {
56             PushRequest request = pushRequestBus.getAndRemove(message.getSessionId());
57             if (request == null) {
58                 Logs.PUSH.warn("receive a gateway response, but request has timeout. message={}", message);
59                 return;
60             }
61             request.onSuccess(GatewayPushResult.fromJson(message.data)); //推送成功
62         }
63     }
64 }
65
```

## 用户离线

测试用户离线：

- 1、关闭ServerTestMain.java服务、ConnClientTestMain.java服务
- 2、删除classes目录下的cache.dat文件
- 3、运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息

```
PushClient.java x PushClientTestMain.java x PushRequestBus.java x PushRequest.java x PushContext.java x
PushClient send0()
39 public final class PushClient extends BaseService implements PushSender {
40
41     private MPushClient mPushClient;
42
43     private PushRequestBus pushRequestBus;
44
45     private CachedRemoteRouterManager cachedRemoteRouterManager;
46
47     private GatewayConnectionFactory gatewayConnectionFactory;
48
49     private FutureTask<PushResult> send0(PushContext ctx) {
50         if (ctx.isBroadcast()) {
51             return PushRequest.build(mPushClient, ctx).broadcast();
52         } else {
53             Set<RemoteRouter> remoteRouters = cachedRemoteRouterManager.lookupAll(ctx.getUserId());
54             if (remoteRouters == null || remoteRouters.isEmpty()) {
55                 return PushRequest.build(mPushClient, ctx).onOffline();
56             }
57             FutureTask<PushResult> task = null;
58             for (RemoteRouter remoteRouter : remoteRouters) {
59                 task = PushRequest.build(mPushClient, ctx).send(remoteRouter);
60             }
61             return task;
62         }
63     }
}
```

```
private void offline() {
    mPushClient.getCachedRemoteRouterManager().invalidateLocalCache(userId);
    submit(Status.offline); 用户离线，清除本地的路由信息
}
```

```
private void submit(Status status) {
    if (this.status.compareAndSet(Status.init, status)) { //防止重复调用
        boolean isTimeoutEnd = status == Status.timeout; //任务是否超时结束

        if (future != null && !isTimeoutEnd) { //是超时结束任务不用再取消一次
            future.cancel(true); //取消超时任务
        }
        //此时future=null

        this.timeLine.end(); //结束时间流统计
        super.set(getResult()); //设置同步调用的返回结果

        if (callback != null) { //回调callback
            if (isTimeoutEnd) { //超时结束时，当前线程已经是线程池里的线程，直接调用callback
                callback.onResult(getResult());
            } else { //非超时结束时，当前线程为Netty线程池，要异步执行callback
                mPushClient.getPushRequestBus().asyncCall(this); //会执行run方法
            }
        }
    }
}

LOGGER.info("push request {} end, {}, {}, {}", status, userId, location, timeLine);
}
```



```

/**
 * run方法会有两个地方的线程调用
 * 1. 任务超时时会调用, 见 PushRequestBus.I.put(sessionId, PushRequest.this);
 * 2. 异步执行 callback 的时候, 见 PushRequestBus.I.asyncCall(this);
 */
@Override
public void run() {
    //判断任务是否超时, 如果超时了此时状态是 init, 否则应该是其他状态, 因为从submit方法过来的状态都不是init
    if (status.get() == Status.init) {
        timeout();
    } else {
        callback.onResult(getResult());
    }
}
}

```

## 返回结果

```

PushClientTestMain testPush()
53 msg.setMsgId("msgId_" + i);
54
55 PushContext context = PushContext.build(msg)
56     .setAckModel(AckModel.AUTO_ACK)
57     .setUserId("user-" + i)
58     .setBroadcast(false)
59     //.setTags(Sets.newHashSet("test"))
60     //.setCondition("tags&&tags.indexOf('test')!=-1")
61     //.setUserIds(Arrays.asList("user-0", "user-1"))
62     .setTimeout(2000)
63     .setCallback(new PushCallback() {
64         @Override
65         public void onResult(PushResult result) {
66             System.err.println("\n\n" + result);
67         }
68     });
69 FutureTask<PushResult> future = sender.send(context);
70
71 //System.err.println("\n\n" + future.get());
72
73

```

```

1 test passed - 2m 3s 15ms
22:07:50.070 - [main] DEBUG - i.n.u.i.InternalThreadLocalMap - -Dio.netty.threadLocalMap.stringBuilder.initialSize: 1024
22:07:50.070 - [main] DEBUG - i.n.u.i.InternalThreadLocalMap - -Dio.netty.threadLocalMap.stringBuilder.maxSize: 4096
22:07:51.240 - [main] INFO - com.mpush.client.push.PushRequest - push request offline end, user-0, null, Push-Time-Line[0](ms){end}

PushResult{resultCode=offline, userId='user-0', timeLine=[end, 1541686071240], null}
Disconnected from the target VM, address: '127.0.0.1:55523', transport: 'socket'

```

## 发送失败

消息发送失败, 有几种场景: 1、连接不上 2、发送时, 连接中途断开 3、netty处理失败  
场景1: (连接不上)

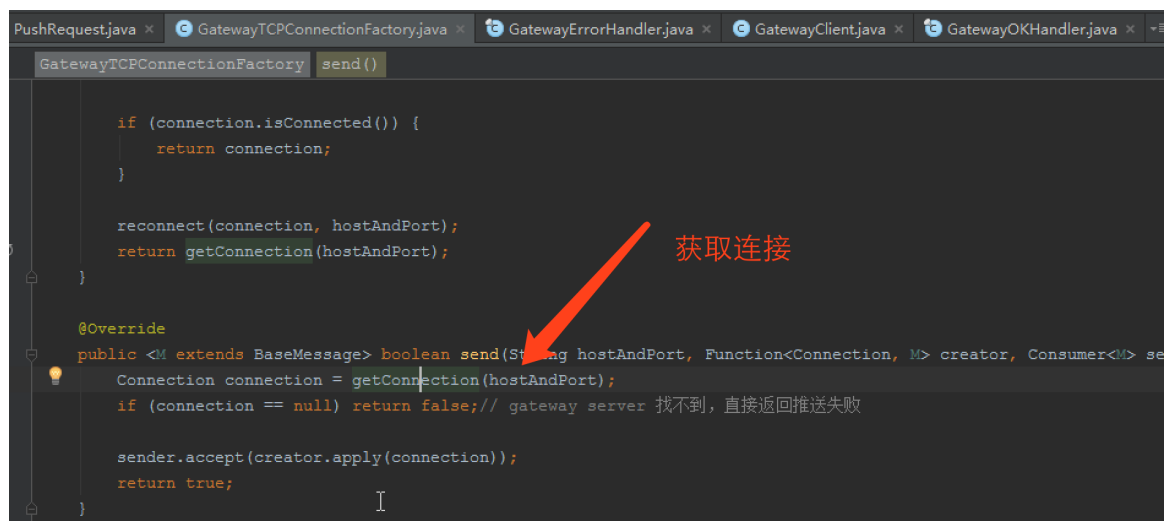
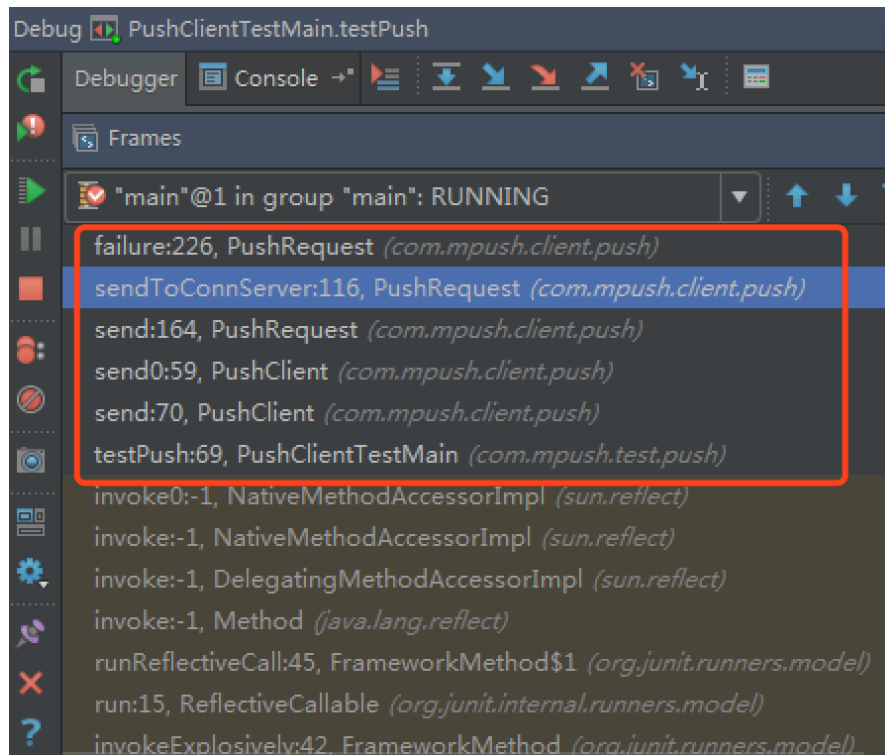
- 1、关闭ServerTestMain.java服务、ConnClientTestMain.java服务
  - 2、修改classes目录下的cache.dat文件中网关IP地址为一个错误地址(模拟"连接不上"场景)
- ```

{
    "mp:ur:user-0": {
        "1": {"port": "3001", "host": "172.16.177.129", "deviceId": "test-device-id-0", "osName": "android", "clientVersion": "1.0.0"}
    },
}

```

### 3、运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息

自下而上的调用栈



```
PushRequest.java x GatewayTCPConnectionFactory.java x GatewayErrorHandler.java x GatewayClient.java x GatewayOKHandler.java x
PushRequest sendToConnServer()
//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)
        .setUserId(userId) .userId: "user-0"
        .setContent(content)
        .setClientType(location.getClientType())
        .setTimeout(timeout - 500) .timeout: 2000
        .setTags(tags) .tags: null
        .addFlag(ackModel.flag) .ackModel: "AUTO_ACK"
);

pushMessage -> {
    timeLine.addTimePoint("send-to-gateway-begin");
    pushMessage.sendRaw(f -> {
        timeLine.addTimePoint("send-to-gateway-end"); timeLine: "Push-Time-Line[0] (ms) {begin -- (0ms
        if (f.isSuccess()) {
            LOGGER.debug("send to gateway server success, location={}, conn={}", location, f.channel(
        } else {
            LOGGER.error("send to gateway server failure, location={}, conn={}", location, f.channel(
                failure(); 连接中途断开
        }
    });
    PushRequest.this.content = null; //释放内存 content: {123, 34, 99, 111, 110, 116, 101, 110, 116, 3
    sessionId = pushMessage.getSessionId();
    future = mPushClient.getPushRequestBus().put(sessionId, PushRequest.this); future: null mPushCl
};

if (!success) {
    LOGGER.error("get gateway connection failure, location={}, location); location: "ClientLocation(host="
    failure();
}
```

场景2：（发送时，连接中途断开）

.....

- 1、运行com.mpush.test.sever.ServerTestMain.java启动长链接服务
- 2、在GatewayTCPConnectionFactory#send方法中的如下语句打上断点  
sender.accept(creator.apply(connection));
- 3、运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息
- 4、关闭ConnClientTestMain.java服务  
等步骤3，卡在断点处，然后再关闭服务；

场景3：（netty处理失败）

```
ConnectionFactory.java × GatewayErrorHandler.java × GatewayClient.java × GatewayOKHandler.java × GatewayClientChannelHandler.java ×
GatewayErrorHandler handle()

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

@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
            1 → request.onOffline();
        } else if (message.code == PUSH_CLIENT_FAILURE.errorCode) { //下发到客户端失败
            2 → request.onFailure();
        } else if (message.code == ROUTER_CHANGE.errorCode) { //用户路由信息更改
            3 → request.onRedirect();
        }
    }
}
}
```