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

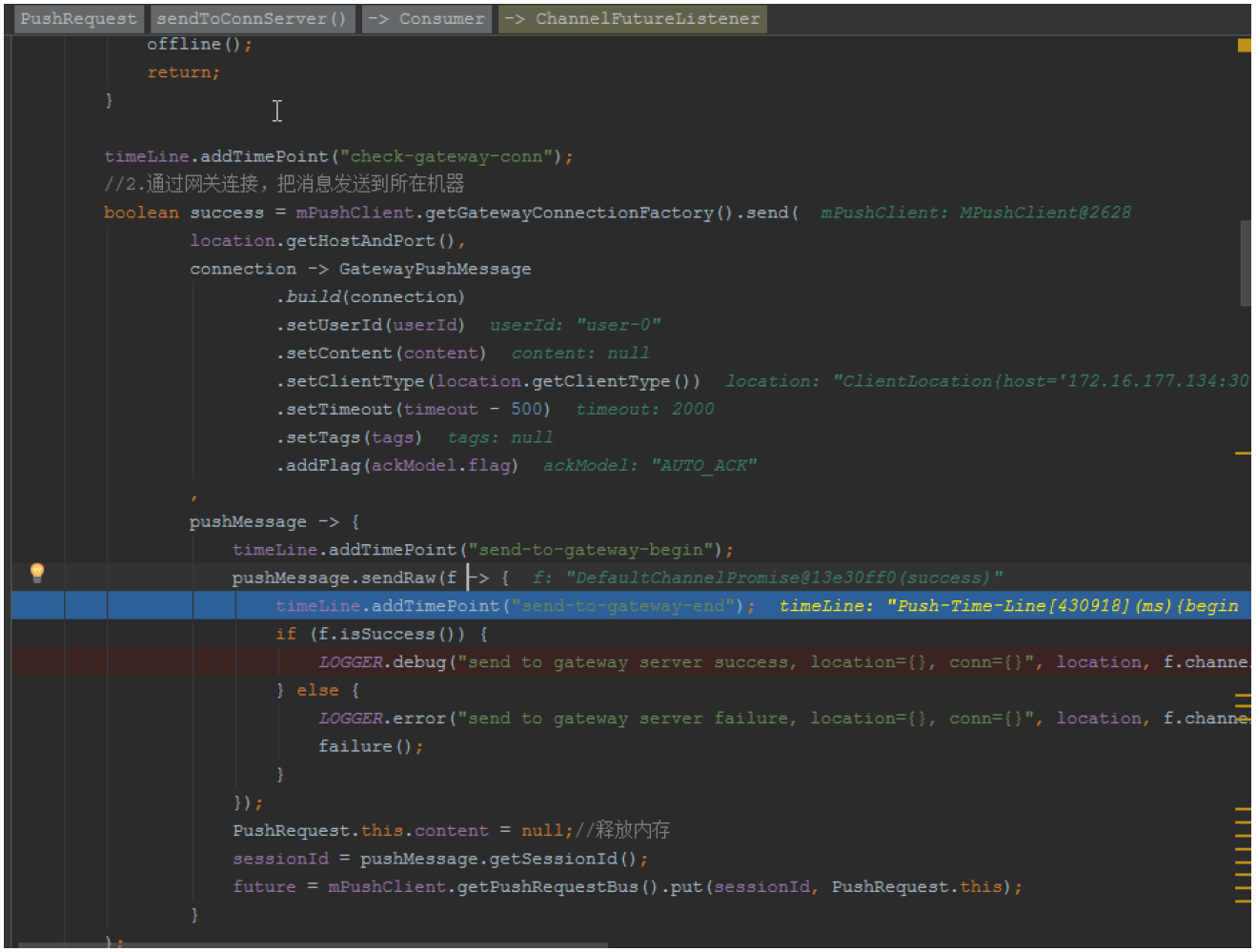
消息正常发送

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

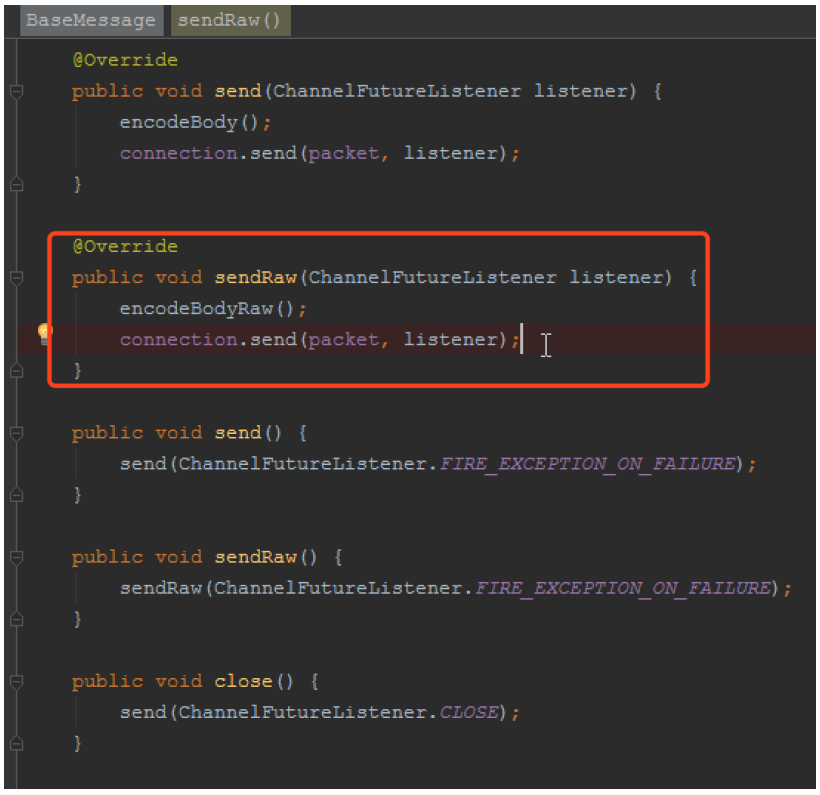
会先执行后面的语句：PushRequest.this.content = null;//释放内存

创建超时任务（见"通信模型与超时控制"文章）

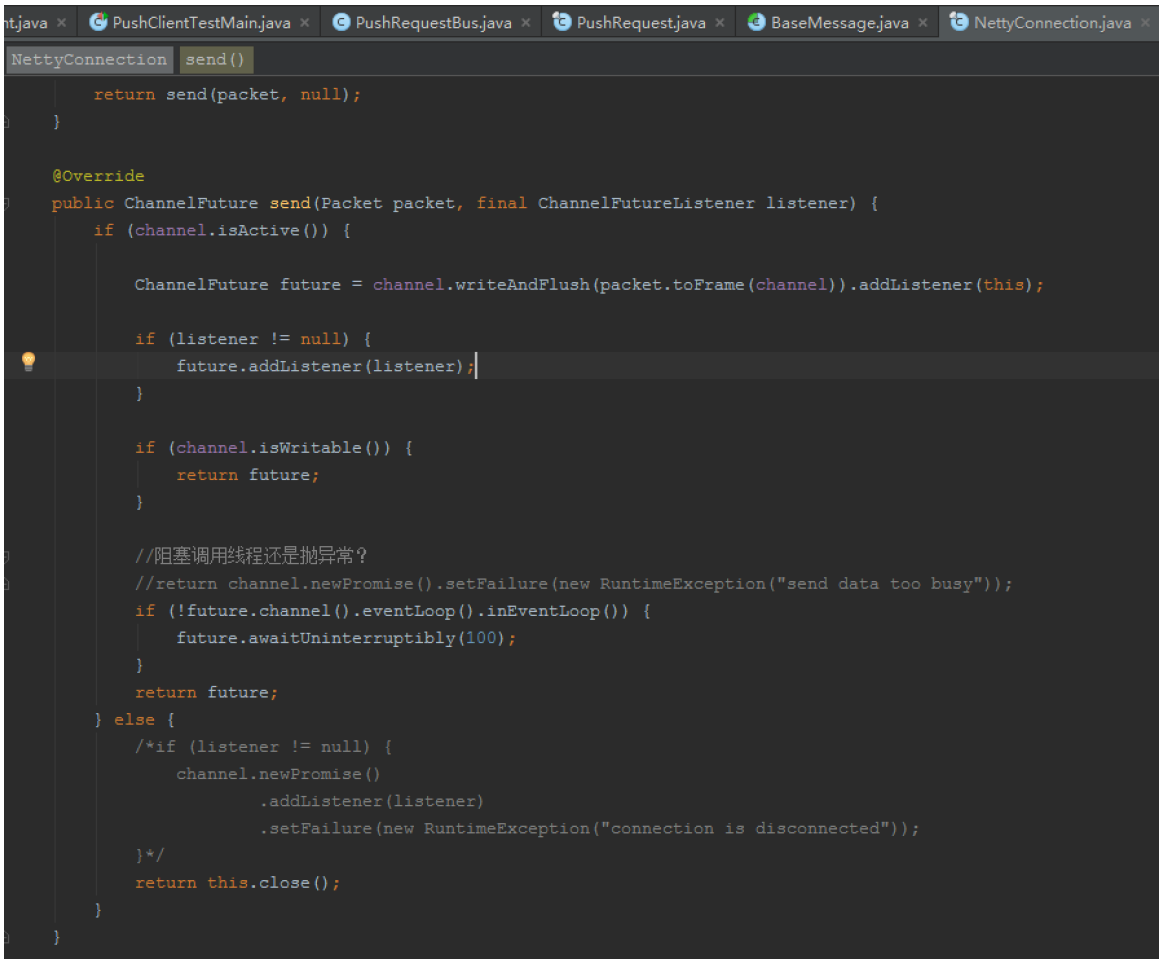
PushRequest.java



BaseMessage.java



NettyConnetion.java



消息发送超时

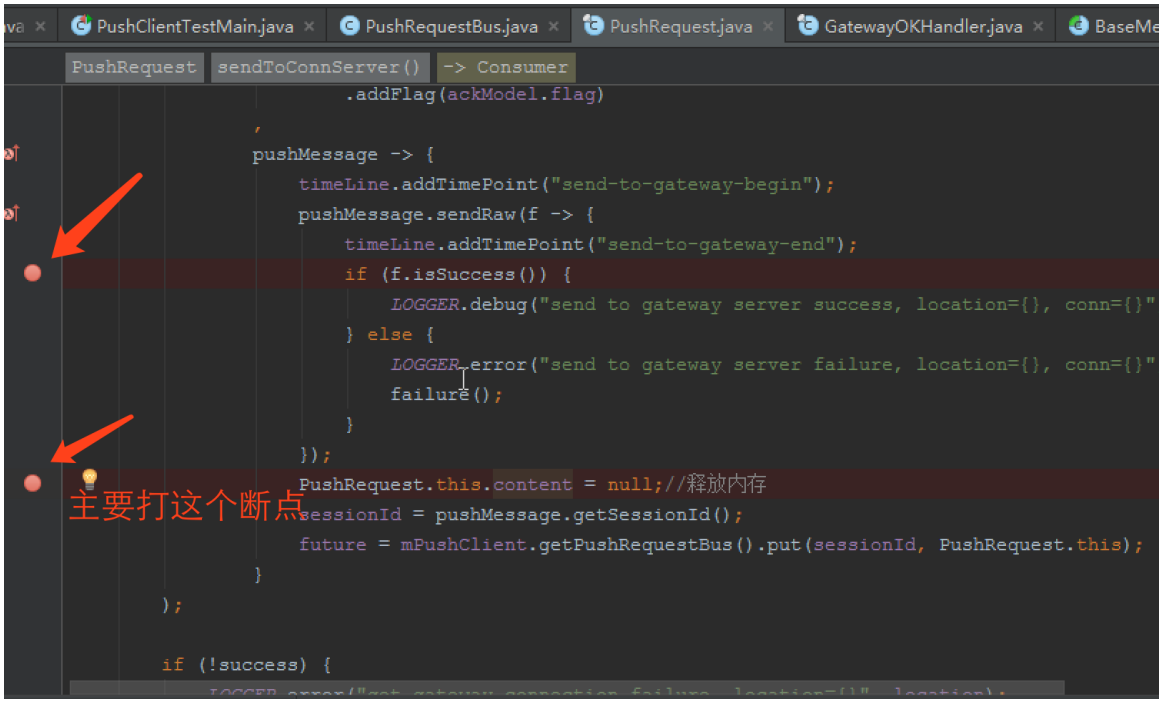
1、先打断点

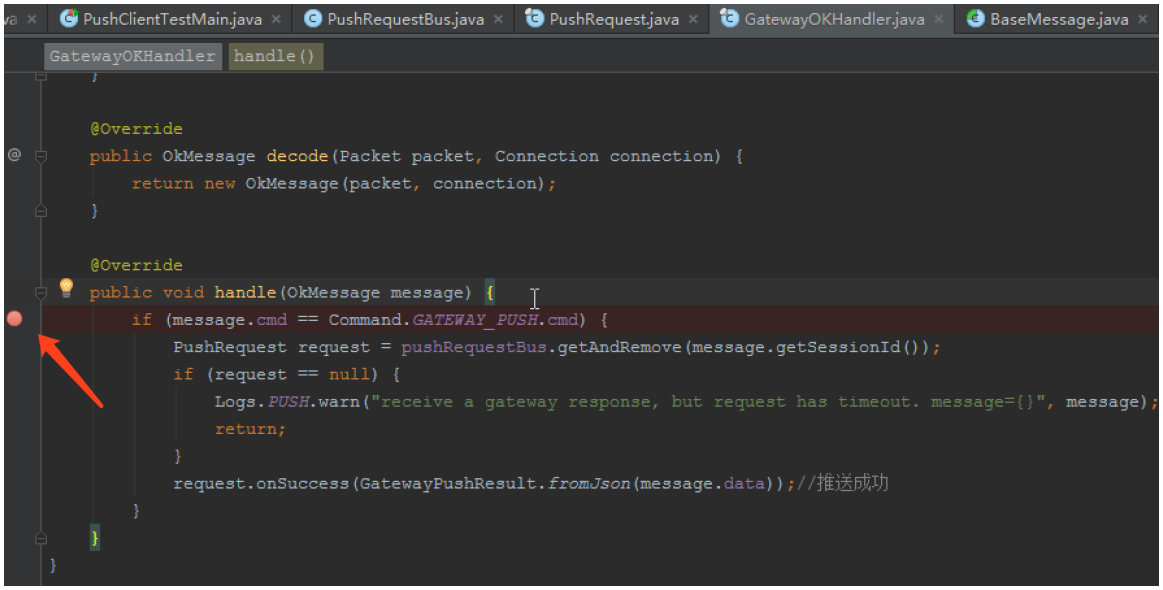
2、运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息

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)

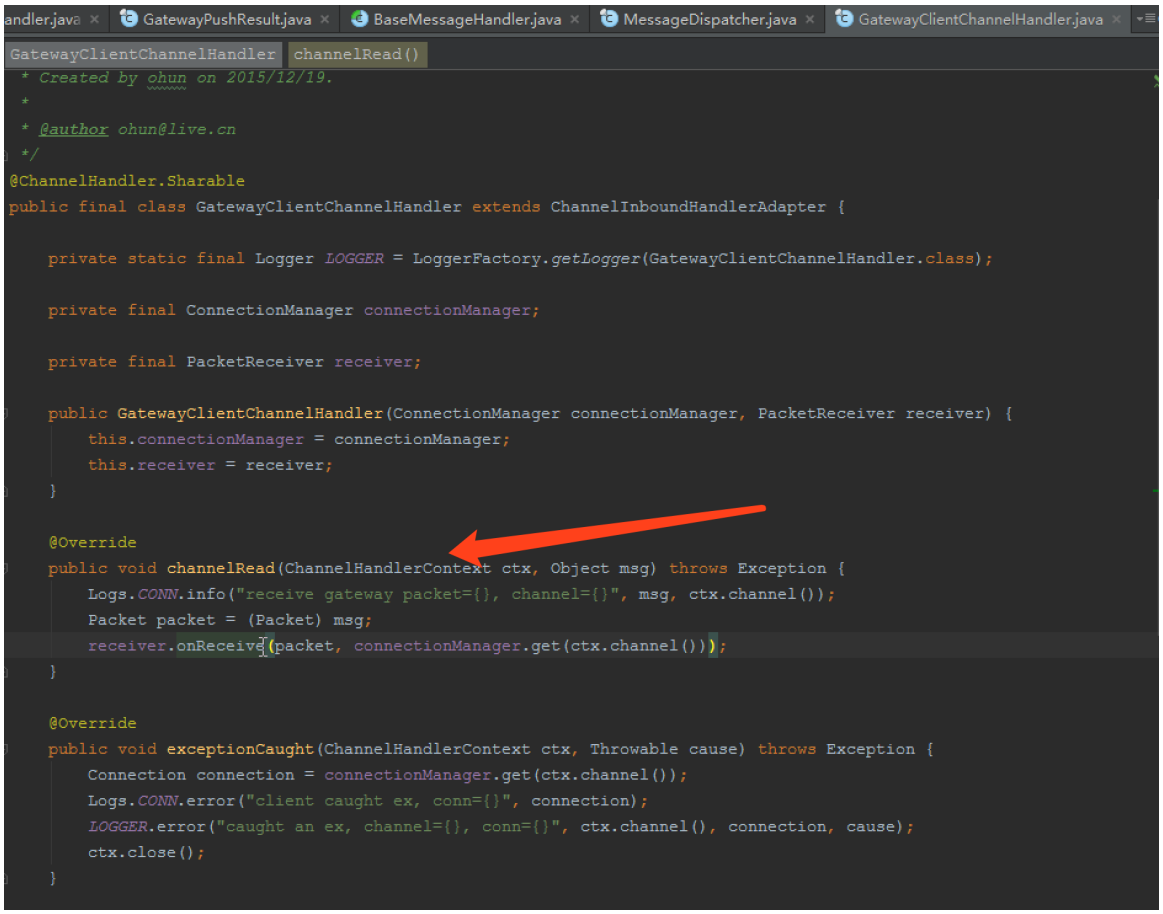


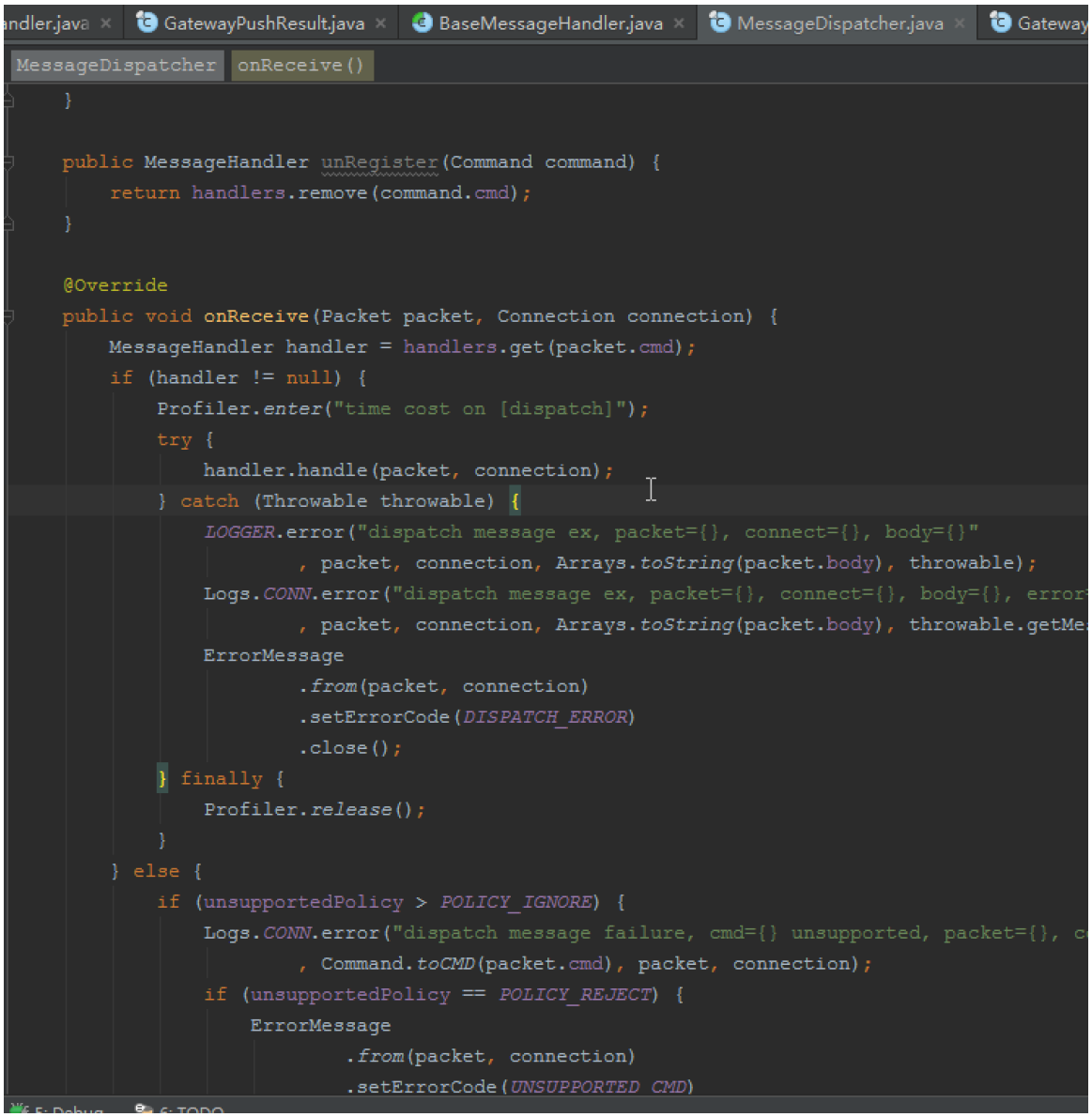


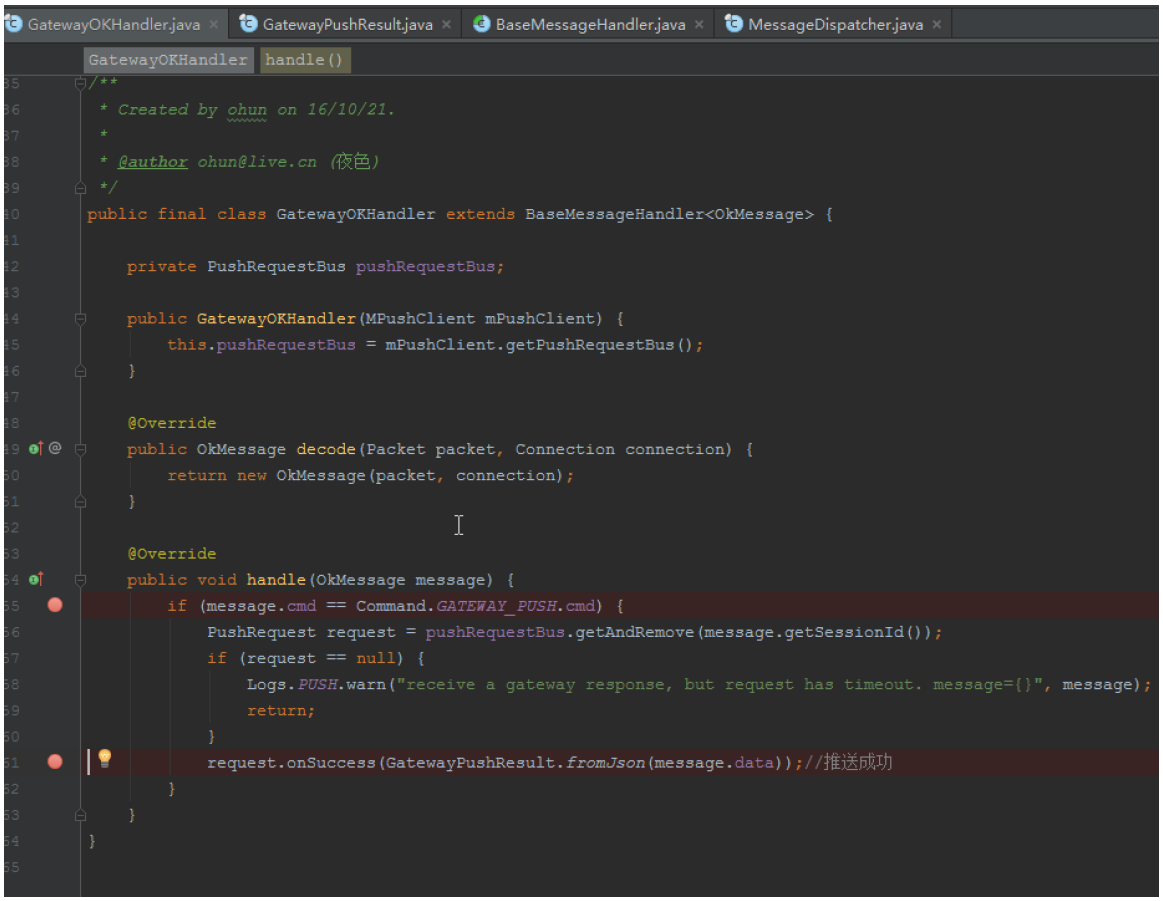
接收消息

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();
}


ndler.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.getMe
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)



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 | PushClientTestMain.java | PushRequestBus.java | PushRequest.java | PushContext.java

PushClient > send0()

```java
public final class PushClient extends BaseService implements PushSender {

    private MPushClient mPushClient;

    private PushRequestBus pushRequestBus;

    private CachedRemoteRouterManager cachedRemoteRouterManager;

    private GatewayConnectionFactory gatewayConnectionFactory;

    private FutureTask<PushResult> send0(PushContext ctx) {
        if (ctx.isBroadcast()) {
            return PushRequest.build(mPushClient, ctx).broadcast();
        } else {
            Set<RemoteRouter> remoteRouters = cachedRemoteRouterManager.lookupAll(ctx.getUserId());
            if (remoteRouters == null || remoteRouters.isEmpty()) {
                return PushRequest.build(mPushClient, ctx).onOffline();
            }
            FutureTask<PushResult> task = null;
            for (RemoteRouter remoteRouter : remoteRouters) {
                task = PushRequest.build(mPushClient, ctx).send(remoteRouter);
            }
            return task;
        }
    }
```

```java
private void offline() {
    mPushClient.getCachedRemoteRouterManager().invalidateLocalCache(userId);
    submit(Status.offline);
}
```

用户离线，清除本地的路由信息

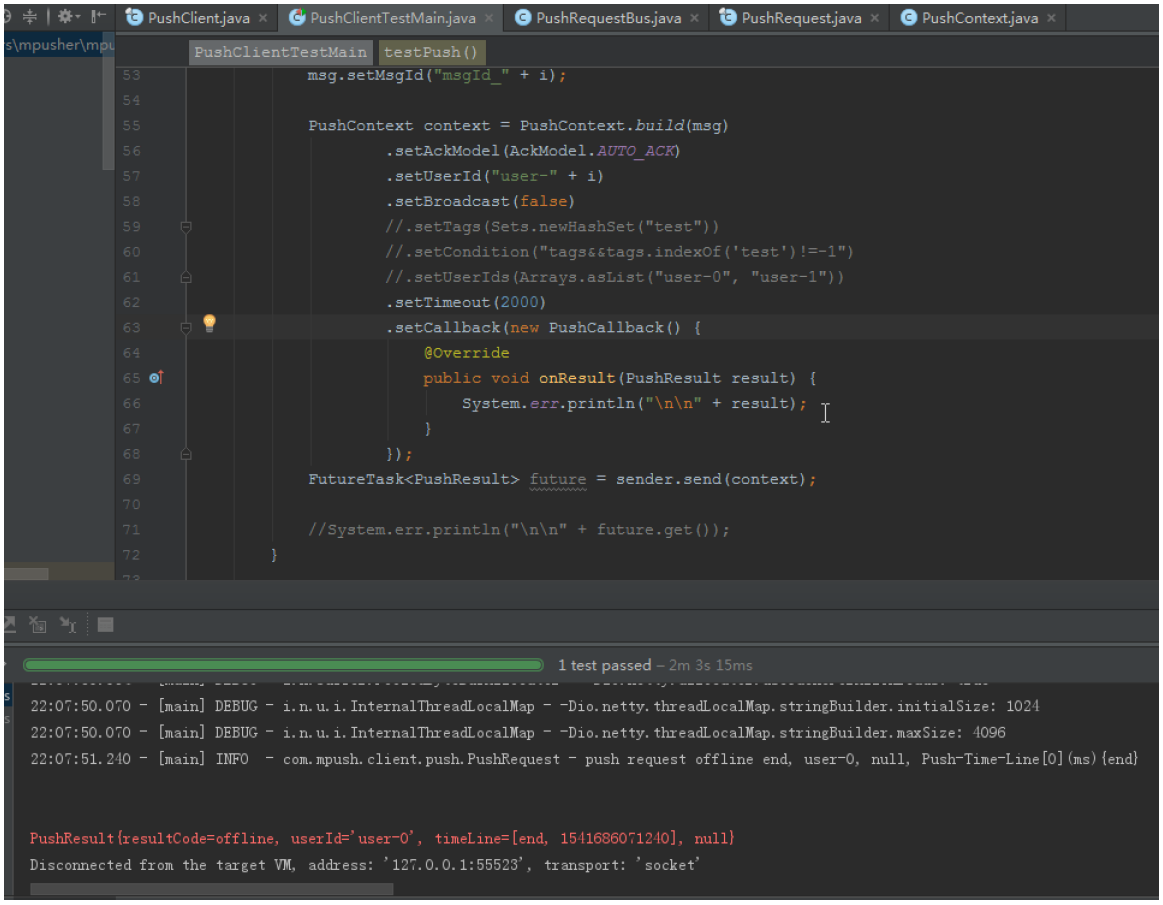
```java
private void submit(Status status) {
    if (this.status.compareAndSet(Status.init, status)) {//防止重复调用
        boolean isTimeoutEnd = status == Status.timeout;//任务是否超时结束

        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
        }

        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);
}
```

此时future=null

会执行异步调用，执行run()

```java
/**
 * 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



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）

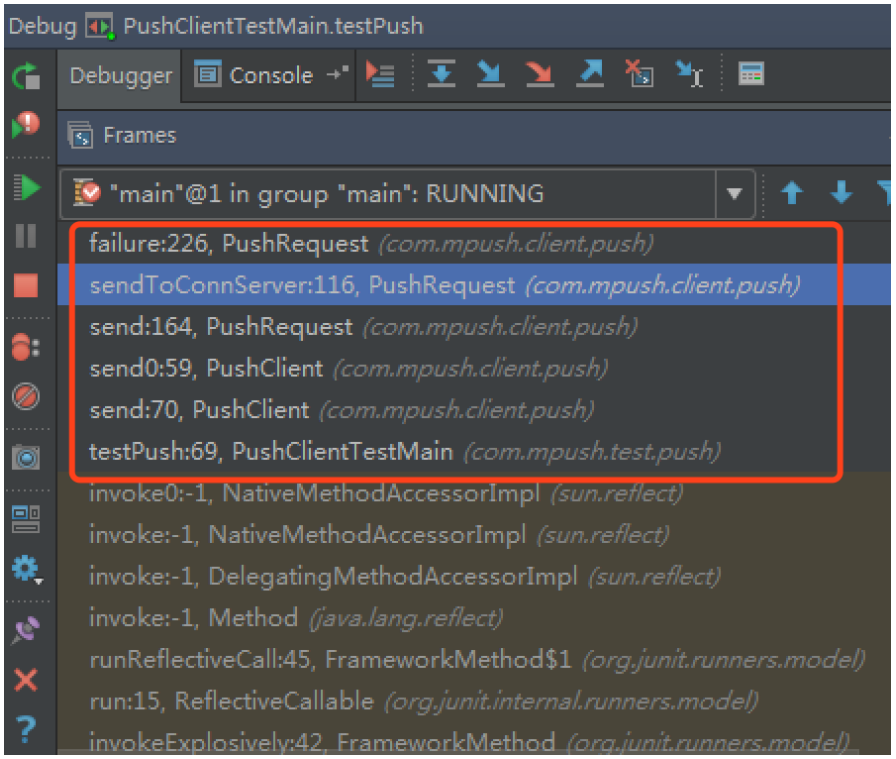
1、关闭ServerTestMain.java服务、ConnClientTestMain.java服务

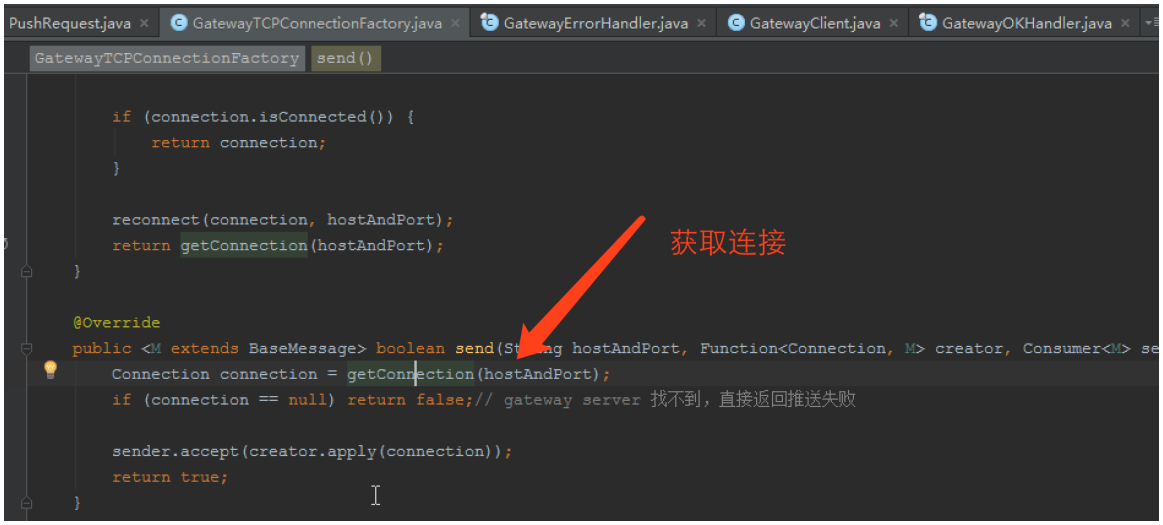
2、修改classes目录下的cache.dat文件中网关IP地址为一个错误地址(模拟"连接不上"场景)

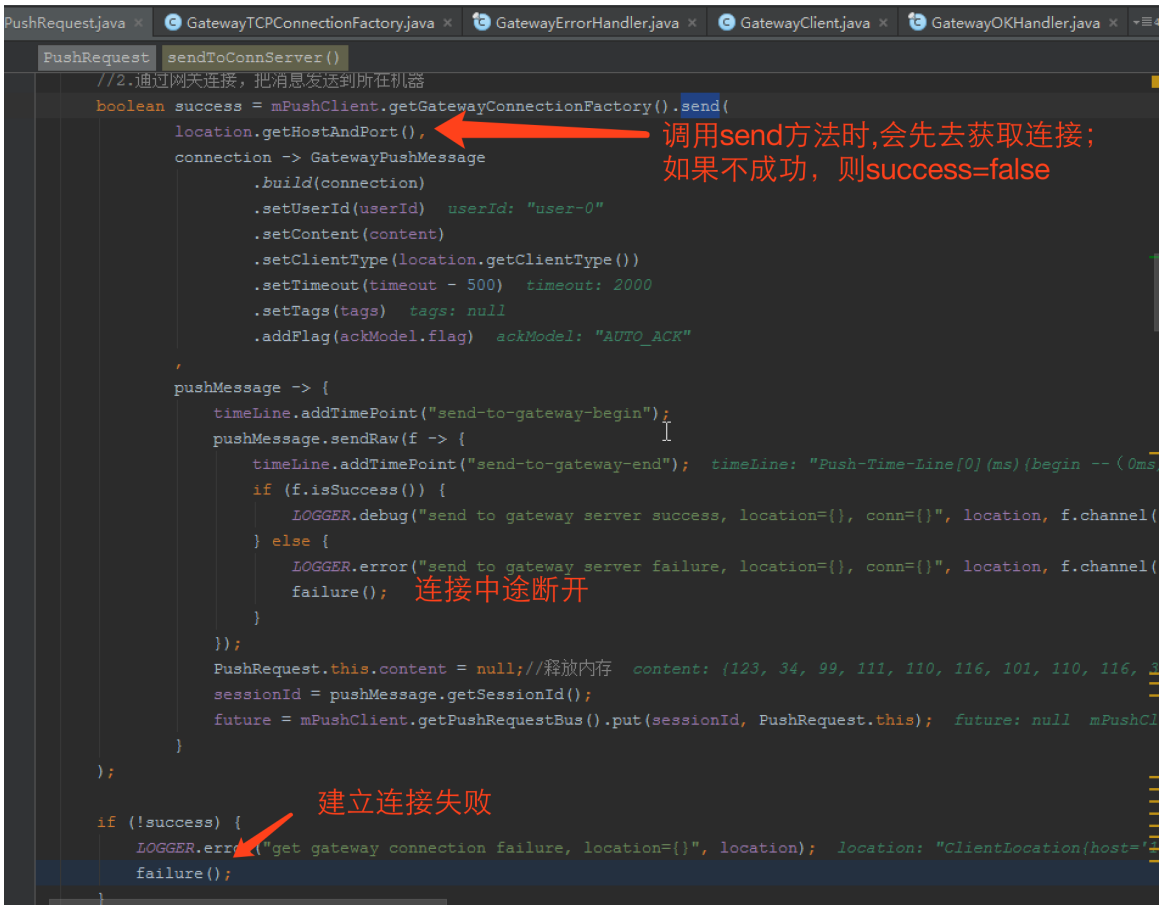
```
{
    "mp:ur:user-0": {
        "1": "{\"port\":3001,\"host\":\"172.16.177.129\",\"deviceId\":\"test-device-id-0\",\"osName\":\"android\",\"clientVersion\":\"1.0.0\"}"
    },
}
```

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







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

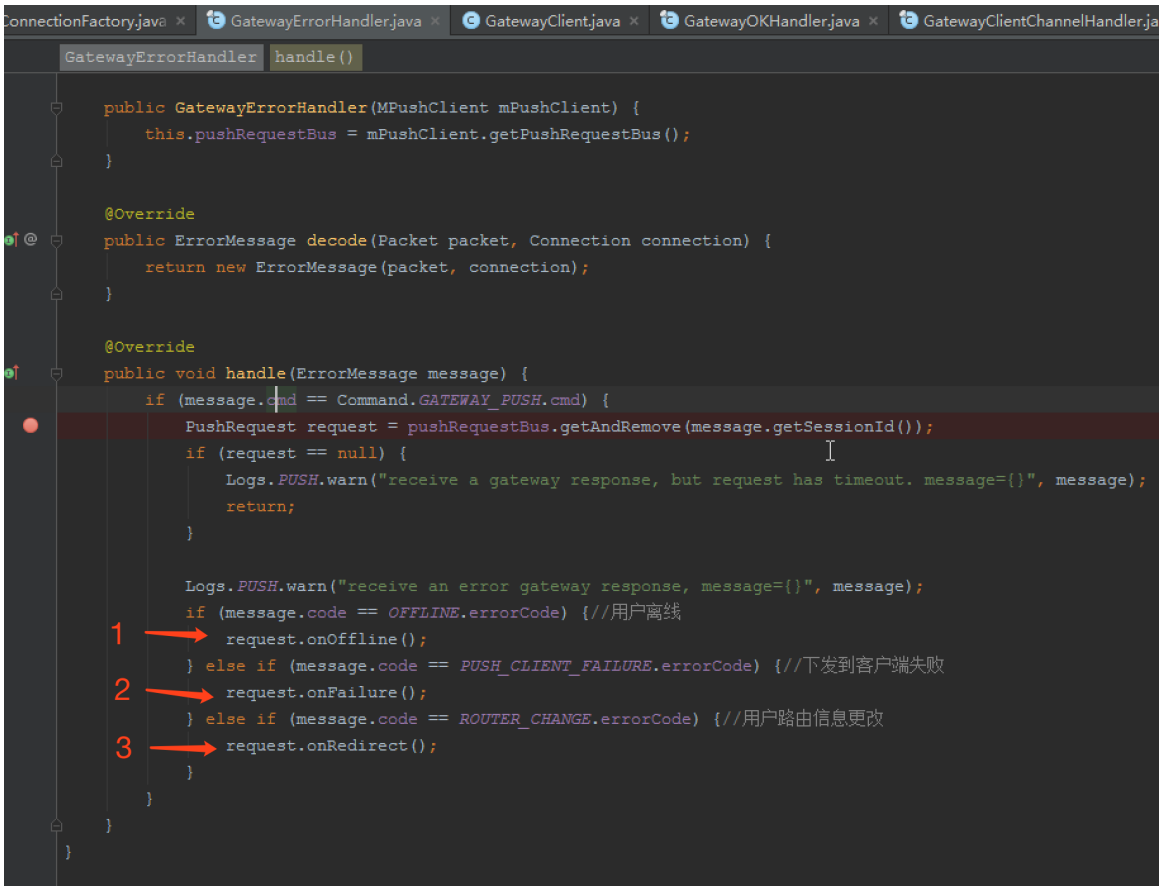
sender.accept(creator.apply(connection));

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.ja
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();
}
}
}
}