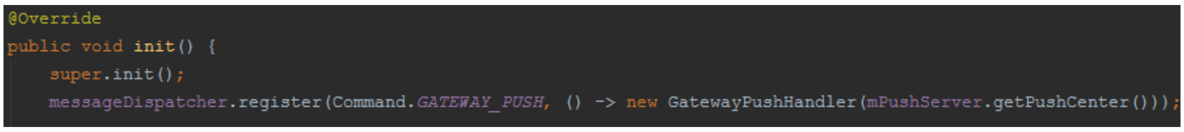
GatewayServer接收到Pushclient消息之后，根据消息类型推送到手机客户端，分2种：
1、广播 BroadcastPushTask
2、单任务 SingleUserPushTask

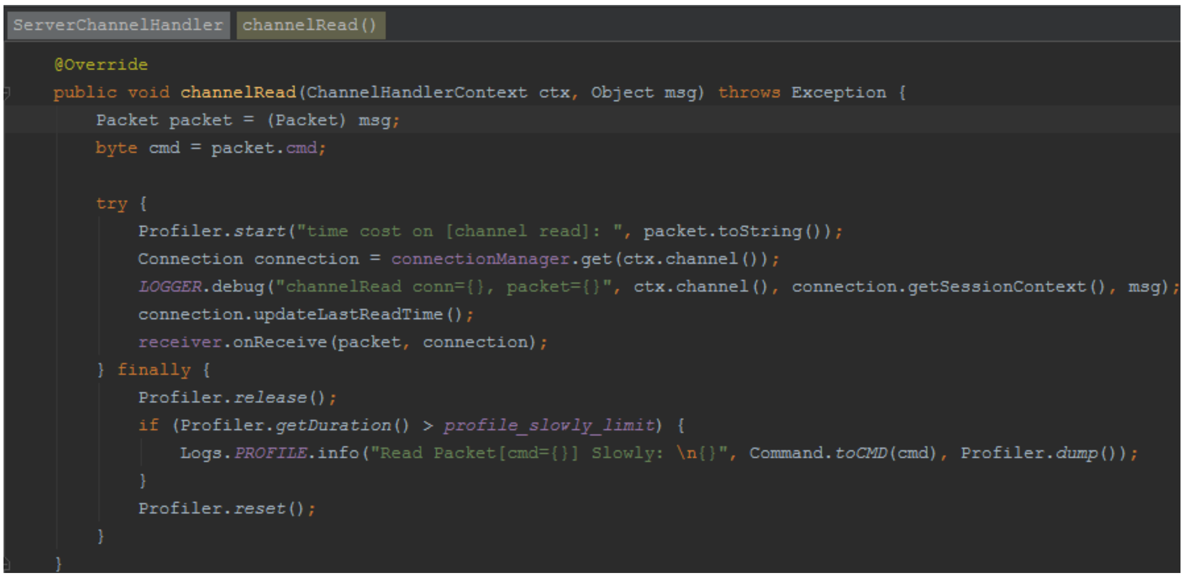
网关服务GatewayServer启动，并注册了netty handler类ServerChannelHandler，用于处理各种事件；
注册到MessageDispatcher中的GatewayPushHandler，用于处理GatewayClient来的消息；


```java
@Override
public void init() {
    super.init();
    messageDispatcher.register(Command.GATEWAY_PUSH, () -> new GatewayPushHandler(mPushServer.getPushCenter()));
```

消息解码

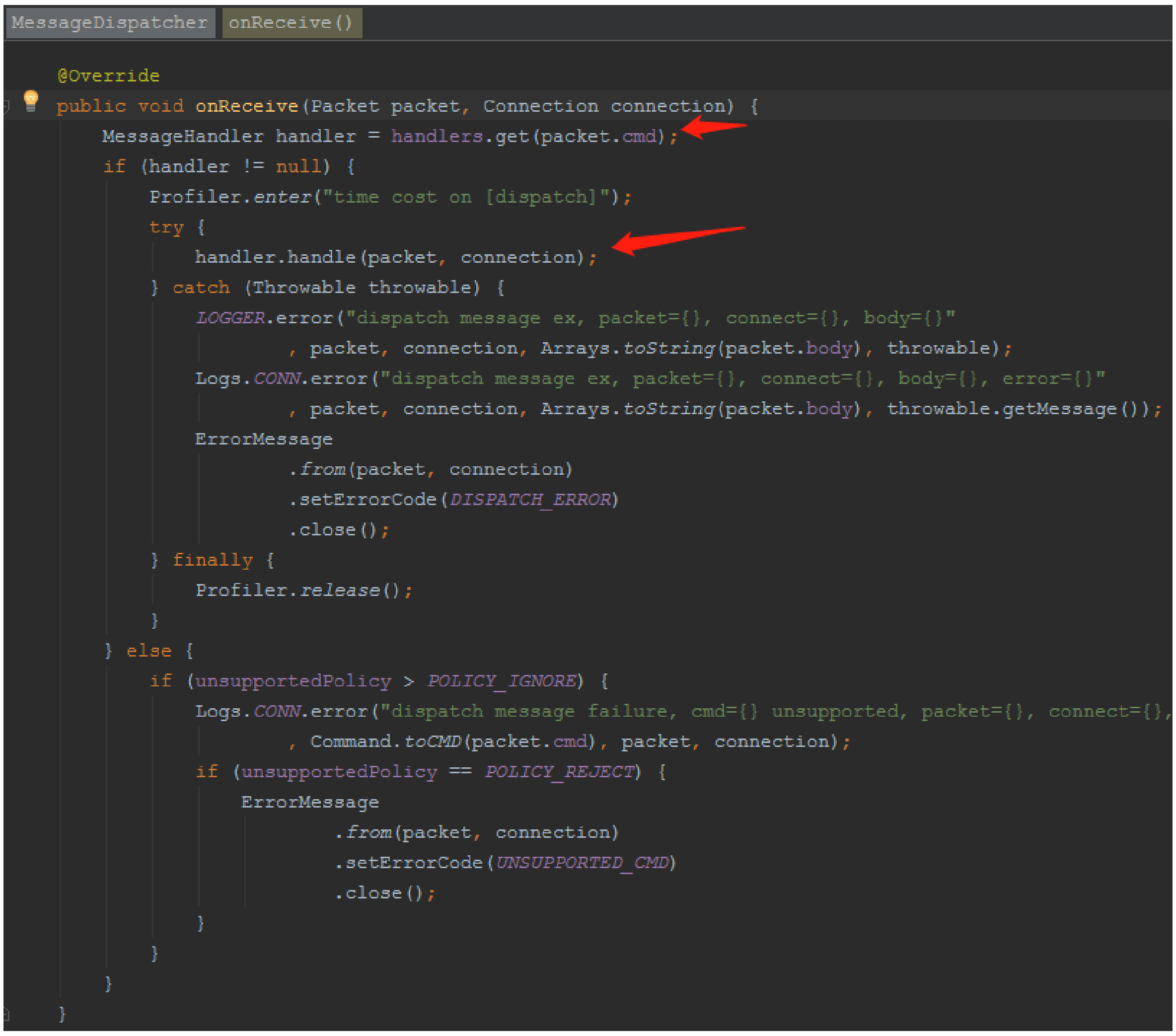
网关服务接收到消息，首先要进行decode解码；

消息处理

接收到GatewayClient来的消息：


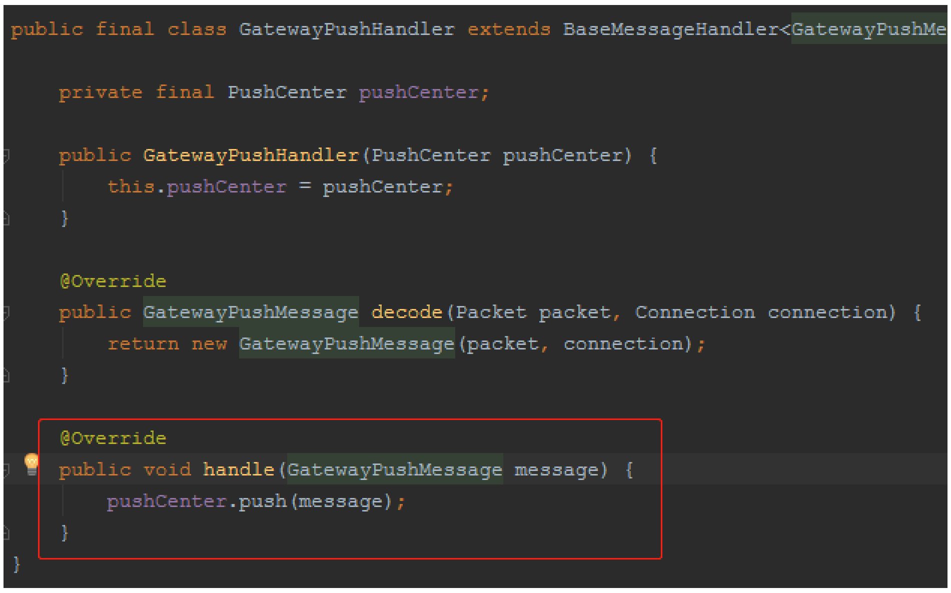
```java
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        Packet packet = (Packet) msg;
        byte cmd = packet.cmd;

        try {
            Profiler.start("time cost on [channel read]: ", packet.toString());
            Connection connection = connectionManager.get(ctx.channel());
            LOGGER.debug("channelRead conn={}, packet={}", ctx.channel(), connection.getSessionContext(), msg);
            connection.updateLastReadTime();
            receiver.onReceive(packet, connection);
        } finally {
            Profiler.release();
            if (Profiler.getDuration() > profile_slowly_limit) {
                Logs.PROFILE.info("Read Packet[cmd={}] Slowly: \n{}", Command.toCMD(cmd), Profiler.dump());
            }
            Profiler.reset();
        }
    }
```

调用MessageDispatcher#onReceive()



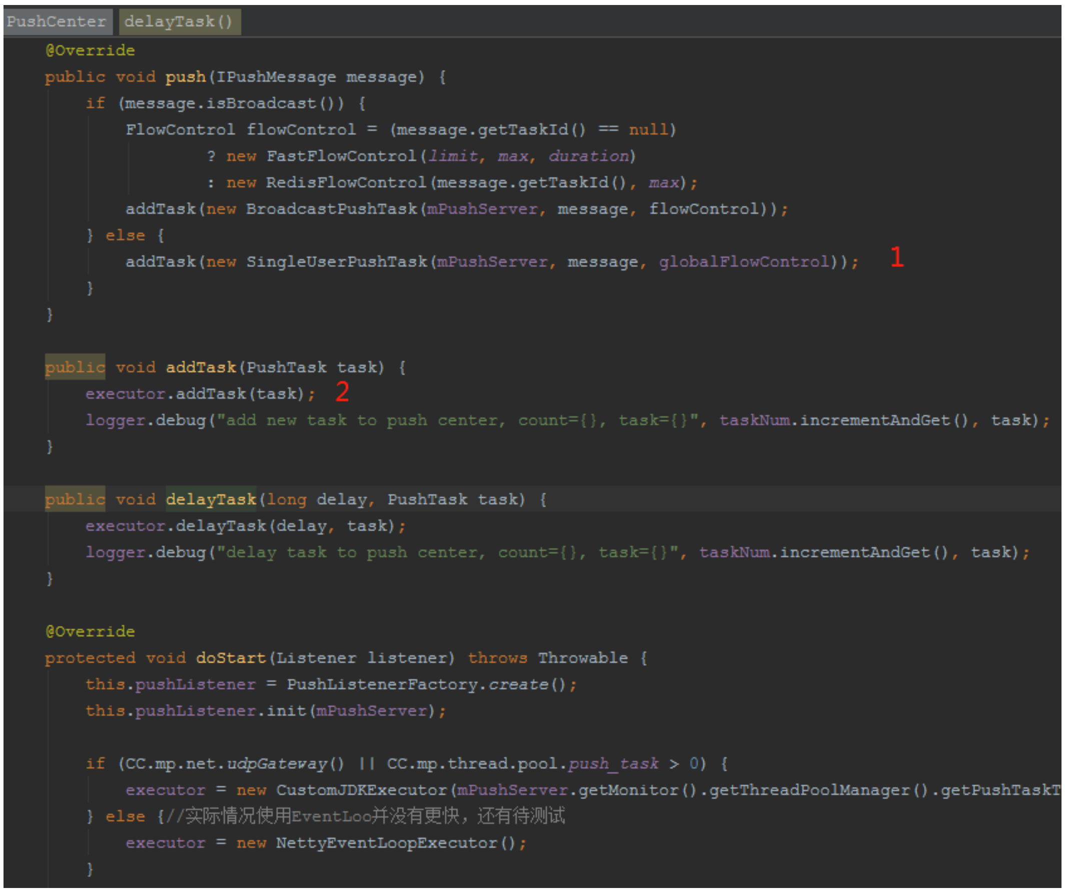
```
MessageDispatcher  onReceive()

    @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={}, connect={},
                        , Command.toCMD(packet.cmd), packet, connection);
                if (unsupportedPolicy == POLICY_REJECT) {
                    ErrorMessage
                            .from(packet, connection)
                            .setErrorCode(UNSUPPORTED_CMD)
                            .close();
                }
            }
        }
    }
```

根据请求包中的CMD，找到注册的GatewayPushHandler，然后调用GatewayPushHandler#handle();

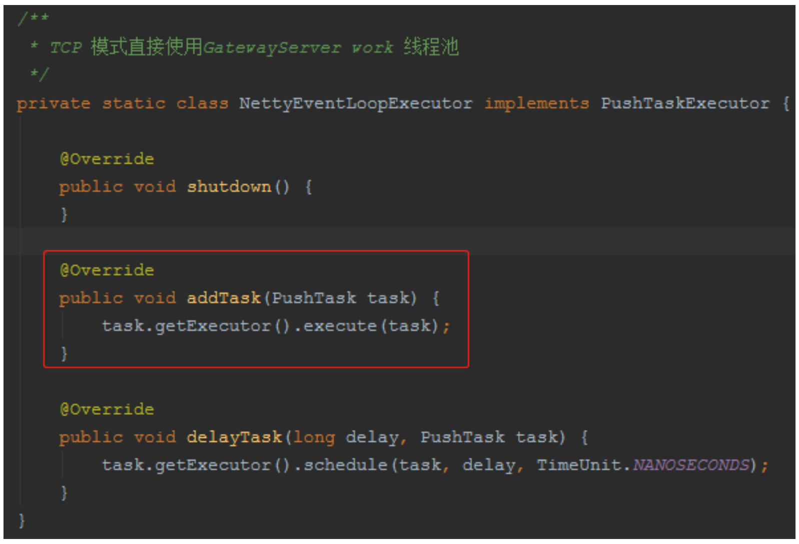


```
public final class GatewayPushHandler extends BaseMessageHandler<GatewayPushMe

    private final PushCenter pushCenter;

    public GatewayPushHandler(PushCenter pushCenter) {
        this.pushCenter = pushCenter;
    }

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

    @Override
    public void handle(GatewayPushMessage message) {
        pushCenter.push(message);
    }
}
```

调用PushCenter#push


```java
PushCenter  delayTask()
    @Override
    public void push(IPushMessage message) {
        if (message.isBroadcast()) {
            FlowControl flowControl = (message.getTaskId() == null)
                    ? new FastFlowControl(limit, max, duration)
                    : new RedisFlowControl(message.getTaskId(), max);
            addTask(new BroadcastPushTask(mPushServer, message, flowControl));
        } else {
            addTask(new SingleUserPushTask(mPushServer, message, globalFlowControl));   1
        }
    }

    public void addTask(PushTask task) {
        executor.addTask(task);  2
        logger.debug("add new task to push center, count={}, task={}", taskNum.incrementAndGet(), task);
    }

    public void delayTask(long delay, PushTask task) {
        executor.delayTask(delay, task);
        logger.debug("delay task to push center, count={}, task={}", taskNum.incrementAndGet(), task);
    }

    @Override
    protected void doStart(Listener listener) throws Throwable {
        this.pushListener = PushListenerFactory.create();
        this.pushListener.init(mPushServer);

        if (CC.mp.net.udpGateway() || CC.mp.thread.pool.push_task > 0) {
            executor = new CustomJDKExecutor(mPushServer.getMonitor().getThreadPoolManager().getPushTaskT
        } else {//实际情况使用EventLoo并没有更快，还有待测试
            executor = new NettyEventLoopExecutor();
        }
```
1、如果不是广播消息，新建一个SingleUserPushTask任务

2、用GatewayServer work 线程池执行任务(业务处理用IO线程执行，可以提高速度)


```java
/**
 * TCP 模式直接使用GatewayServer work 线程池
 */
private static class NettyEventLoopExecutor implements PushTaskExecutor {

    @Override
    public void shutdown() {
    }

    @Override
    public void addTask(PushTask task) {
        task.getExecutor().execute(task);
    }

    @Override
    public void delayTask(long delay, PushTask task) {
        task.getExecutor().schedule(task, delay, TimeUnit.NANOSECONDS);
    }
}
```
任务执行

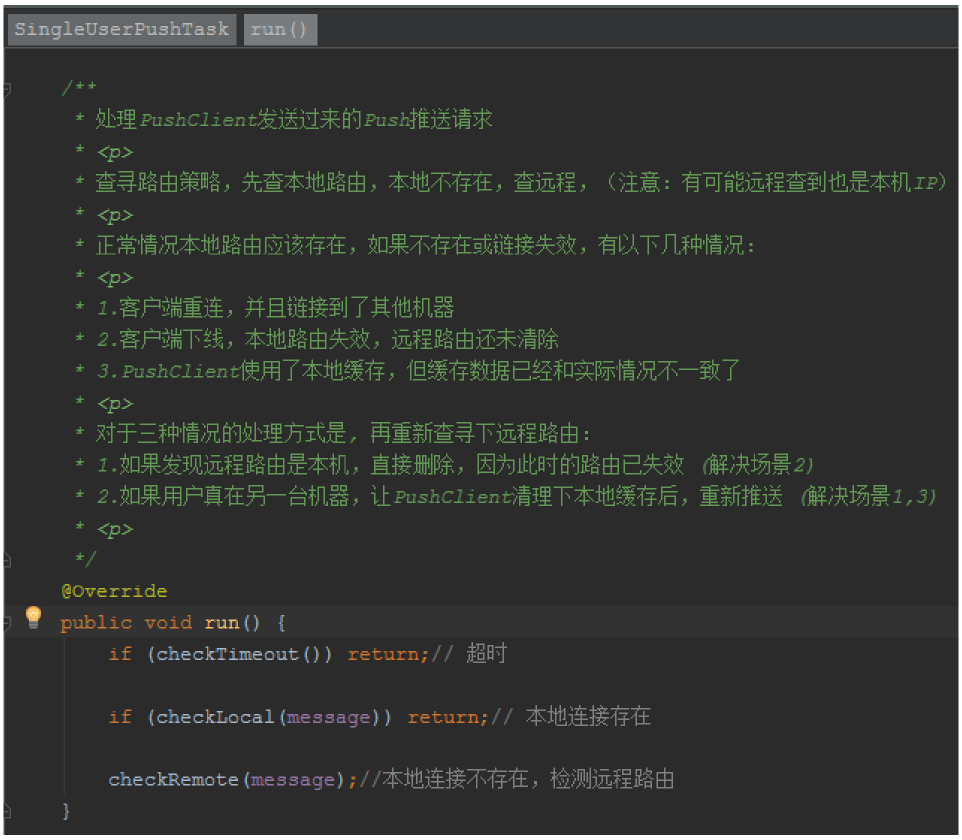
这里会有几种推送状态返回给pushclient（见GatewayPushListener）:

* success（包括2种，不需要ACK、需要ACK(与手机客户端完成ACK)）

* offline

* failure

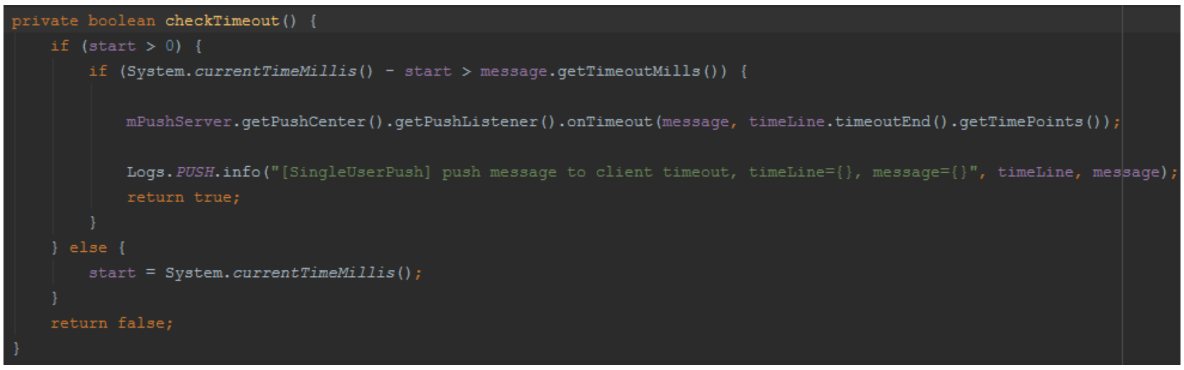
* ronter_change



```java
/**
 * 处理PushClient发送过来的Push推送请求
 * <p>
 * 查寻路由策略，先查本地路由，本地不存在，查远程，（注意：有可能远程查到也是本机IP）
 * <p>
 * 正常情况本地路由应该存在，如果不存在或链接失效，有以下几种情况：
 * <p>
 * 1.客户端重连，并且链接到了其他机器
 * 2.客户端下线，本地路由失效，远程路由还未清除
 * 3.PushClient使用了本地缓存，但缓存数据已经和实际情况不一致了
 * <p>
 * 对于三种情况的处理方式是，再重新查寻下远程路由：
 * 1.如果发现远程路由是本机，直接删除，因为此时的路由已失效 (解决场景2)
 * 2.如果用户真在另一台机器，让PushClient清理下本地缓存后，重新推送 (解决场景1,3)
 * <p>
 */
@Override
public void run() {
    if (checkTimeout()) return;// 超时

    if (checkLocal(message)) return;// 本地连接存在

    checkRemote(message);//本地连接不存在，检测远程路由
}
```

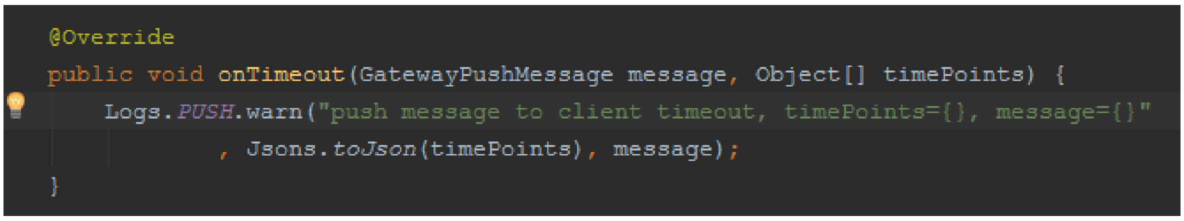
1、超时检测

如果超时，则打印日志；



```java
private boolean checkTimeout() {
    if (start > 0) {
        if (System.currentTimeMillis() - start > message.getTimeoutMills()) {

            mPushServer.getPushCenter().getPushListener().onTimeout(message, timeLine.timeoutEnd().getTimePoints());

            Logs.PUSH.info("[SingleUserPush] push message to client timeout, timeLine={}, message={}", timeLine, message);
            return true;
        }
    } else {
        start = System.currentTimeMillis();
    }
    return false;
}
```

GatewayPushListener#onTimeout



```java
@Override
public void onTimeout(GatewayPushMessage message, Object[] timePoints) {
    Logs.PUSH.warn("push message to client timeout, timePoints={}, message={}"
            , Jsons.toJson(timePoints), message);
}
```

2、本地连接存在

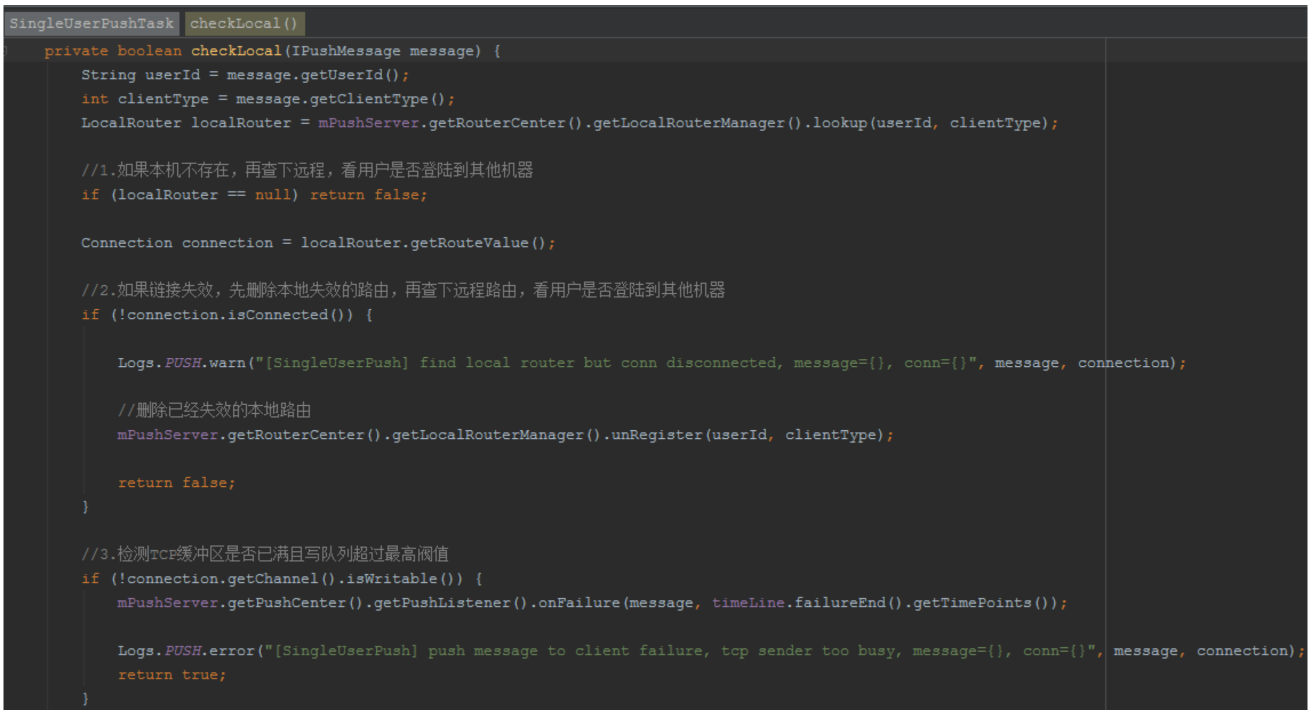
* 如果本地路由存在，且连接可用，则推送到手机客户端；

* 如果本地连接失效，删除路由；

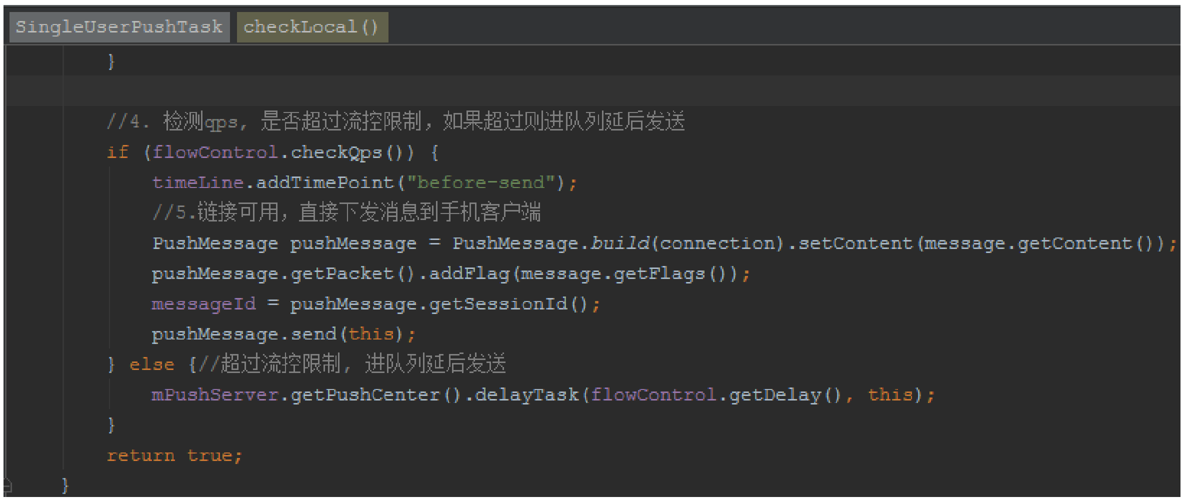
* 检测TCP缓冲区是否已满且写队列超过最高阀值（是否可写）

* 检测qps, 则推送到手机客户端

* 检测qps, 超过流控限制，如果超过则进队列延后发送



```java
private boolean checkLocal(IPushMessage message) {
    String userId = message.getUserId();
    int clientType = message.getClientType();
    LocalRouter localRouter = mPushServer.getRouterCenter().getLocalRouterManager().lookup(userId, clientType);

    //1.如果本机不存在，再查下远程，看用户是否登陆到其他机器
    if (localRouter == null) return false;

    Connection connection = localRouter.getRouteValue();

    //2.如果链接失效，先删除本地失效的路由，再查下远程路由，看用户是否登陆到其他机器
    if (!connection.isConnected()) {

        Logs.PUSH.warn("[SingleUserPush] find local router but conn disconnected, message={}, conn={}", message, connection);

        //删除已经失效的本地路由
        mPushServer.getRouterCenter().getLocalRouterManager().unRegister(userId, clientType);

        return false;
    }

    //3.检测TCP缓冲区是否已满且写队列超过最高阈值
    if (!connection.getChannel().isWritable()) {
        mPushServer.getPushCenter().getPushListener().onFailure(message, timeLine.failureEnd().getTimePoints());

        Logs.PUSH.error("[SingleUserPush] push message to client failure, tcp sender too busy, message={}, conn={}", message, connection);
        return true;
    }
```



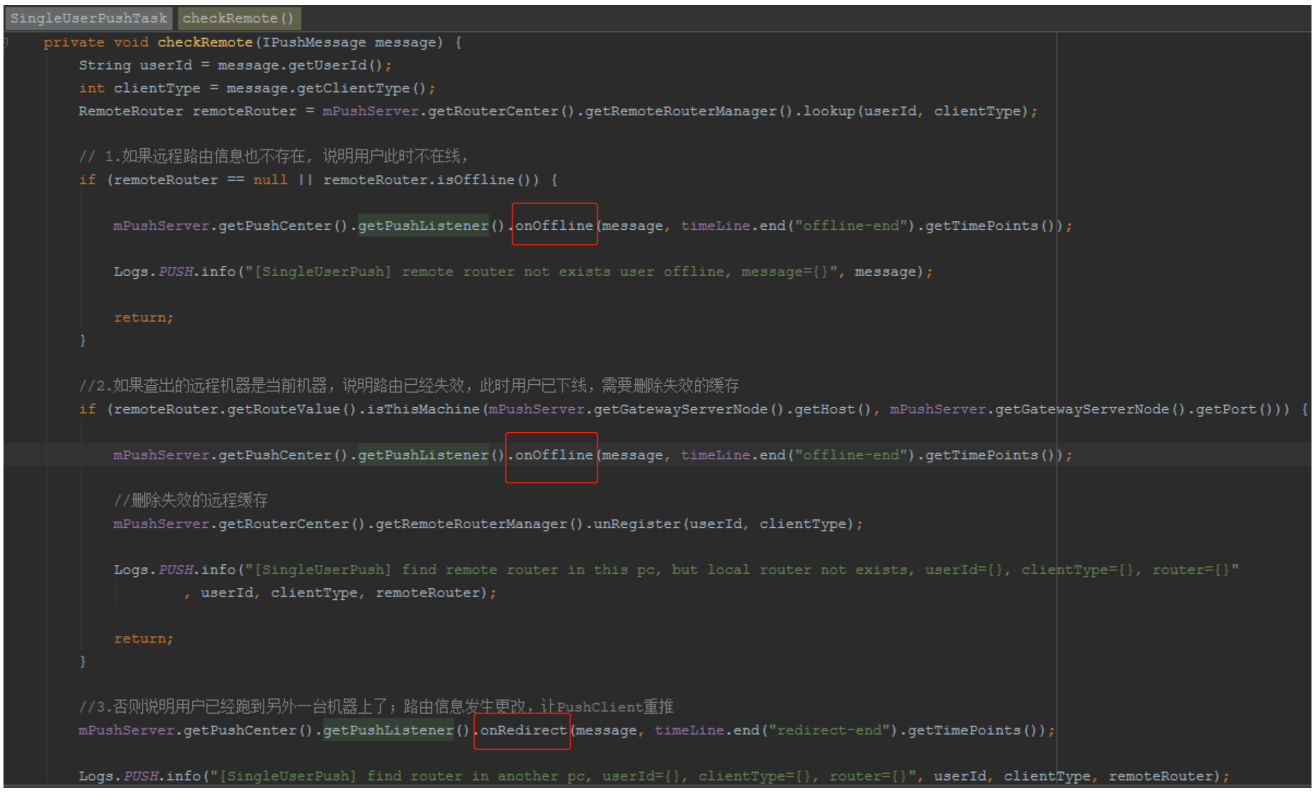
```java
    }

    //4. 检测qps, 是否超过流控限制，如果超过则进队列延后发送
    if (flowControl.checkQps()) {
        timeLine.addTimePoint("before-send");
        //5.链接可用，直接下发消息到手机客户端
        PushMessage pushMessage = PushMessage.build(connection).setContent(message.getContent());
        pushMessage.getPacket().addFlag(message.getFlags());
        messageId = pushMessage.getSessionId();
        pushMessage.send(this);
    } else {//超过流控限制, 进队列延后发送
        mPushServer.getPushCenter().delayTask(flowControl.getDelay(), this);
    }
    return true;
}
```

3、本地连接不存在，检测远程路由

* 如果远程路由信息也不存在, 说明用户此时不在线，发送offline消息到pushclient

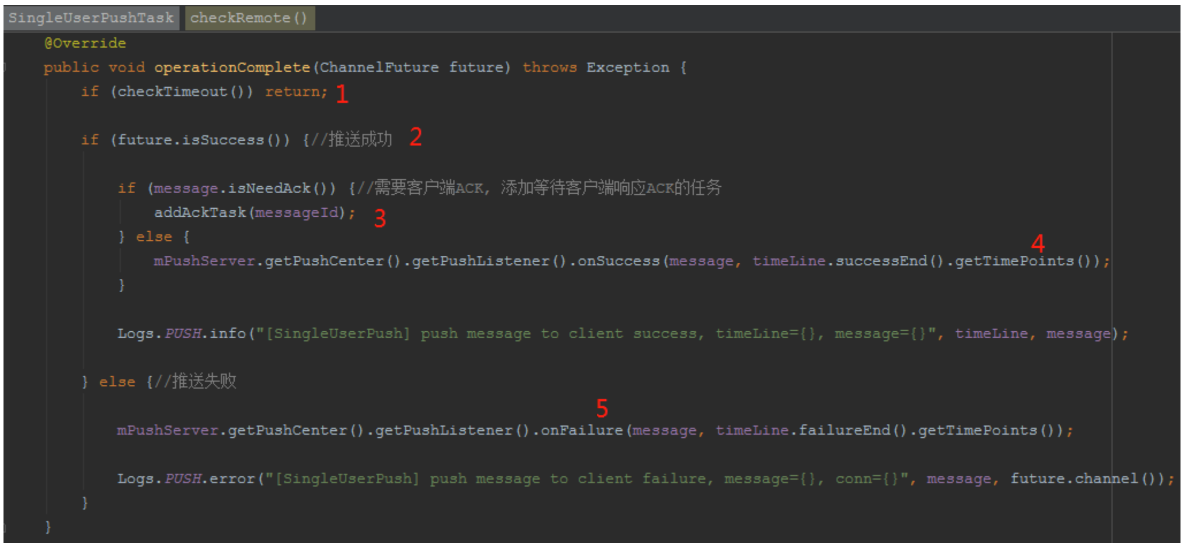
* 如果查出的远程机器是当前机器，说明路由已经失效，此时用户已下线，删除远程路由信息、发送offline消息到pushclient

* 否则说明用户已经跑到另外一台机器上了；路由信息发生更改，发送router_change消息让PushClient重推



```
SingleUserPushTask  checkRemote()
    private void checkRemote(IPushMessage message) {
        String userId = message.getUserId();
        int clientType = message.getClientType();
        RemoteRouter remoteRouter = mPushServer.getRouterCenter().getRemoteRouterManager().lookup(userId, clientType);

        // 1.如果远程路由信息也不存在，说明用户此时不在线，
        if (remoteRouter == null || remoteRouter.isOffline()) {

            mPushServer.getPushCenter().getPushListener().onOffline(message, timeLine.end("offline-end").getTimePoints());

            Logs.PUSH.info("[SingleUserPush] remote router not exists user offline, message={}", message);

            return;
        }

        //2.如果查出的远程机器是当前机器，说明路由已经失效，此时用户已下线，需要删除失效的缓存
        if (remoteRouter.getRouteValue().isThisMachine(mPushServer.getGatewayServerNode().getHost(), mPushServer.getGatewayServerNode().getPort())) {

            mPushServer.getPushCenter().getPushListener().onOffline(message, timeLine.end("offline-end").getTimePoints());

            //删除失效的远程缓存
            mPushServer.getRouterCenter().getRemoteRouterManager().unRegister(userId, clientType);

            Logs.PUSH.info("[SingleUserPush] find remote router in this pc, but local router not exists, userId={}, clientType={}, router={}"
                    , userId, clientType, remoteRouter);

            return;
        }

        //3.否则说明用户已经跑到另外一台机器上了；路由信息发生更改，让PushClient重推
        mPushServer.getPushCenter().getPushListener().onRedirect(message, timeLine.end("redirect-end").getTimePoints());

        Logs.PUSH.info("[SingleUserPush] find router in another pc, userId={}, clientType={}, router={}", userId, clientType, remoteRouter);
```

推送完成，ACK



```
SingleUserPushTask  checkRemote()
    @Override
    public void operationComplete(ChannelFuture future) throws Exception {
        if (checkTimeout()) return; 1

        if (future.isSuccess()) {//推送成功  2

            if (message.isNeedAck()) {//需要客户端ACK，添加等待客户端响应ACK的任务
                addAckTask(messageId); 3
            } else {                                                                          4
                mPushServer.getPushCenter().getPushListener().onSuccess(message, timeLine.successEnd().getTimePoints());
            }

            Logs.PUSH.info("[SingleUserPush] push message to client success, timeLine={}, message={}", timeLine, message);

        } else {//推送失败
                                                   5
            mPushServer.getPushCenter().getPushListener().onFailure(message, timeLine.failureEnd().getTimePoints());

            Logs.PUSH.error("[SingleUserPush] push message to client failure, message={}, conn={}", message, future.channel());
        }
    }
```

1、超时检测，看是不是整个过程执行下来已经超时

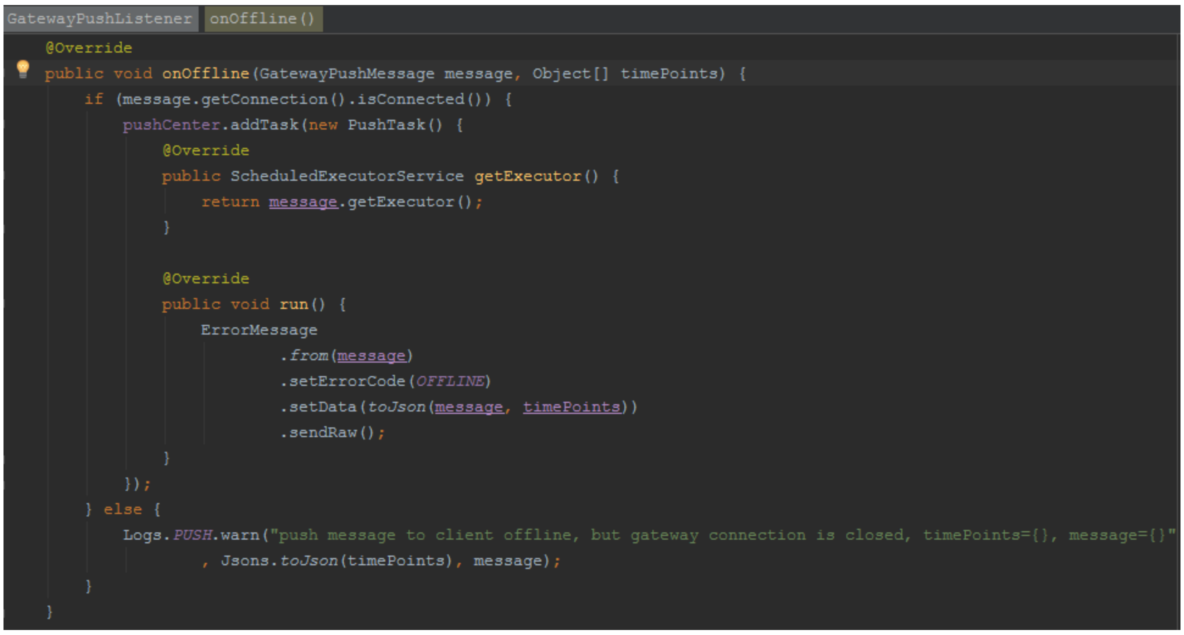
2、判断此次推送消息到手机客户端的Future任务是否成功

3、如果gateway server接收到的消息需要手机客户端ACK，则加入到ack任务队列中异步执行；

4、如果不需要手机端ACK，直接返回success给Pushclient

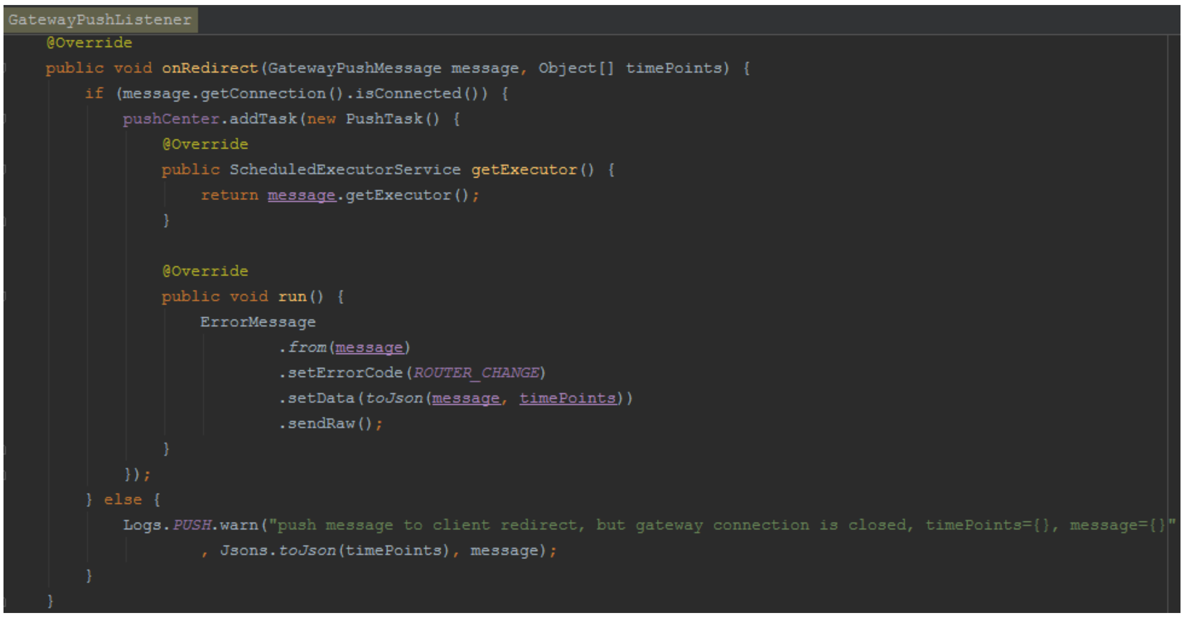
5、推送消息到手机客户端的Future任务执行失败，则返回failure给Pushclient

关于offline，如果发送消息给pushclient的连接被关闭，也就无法告知其离线状态；



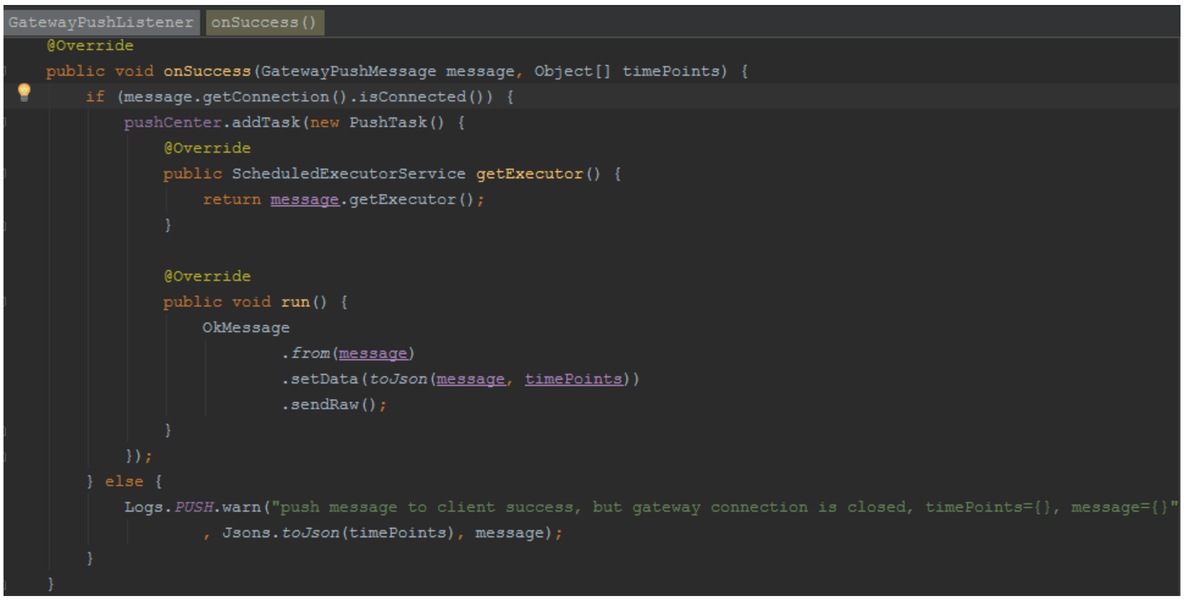
```java
@Override
public void onOffline(GatewayPushMessage message, Object[] timePoints) {
    if (message.getConnection().isConnected()) {
        pushCenter.addTask(new PushTask() {
            @Override
            public ScheduledExecutorService getExecutor() {
                return message.getExecutor();
            }

            @Override
            public void run() {
                ErrorMessage
                        .from(message)
                        .setErrorCode(OFFLINE)
                        .setData(toJson(message, timePoints))
                        .sendRaw();
            }
        });
    } else {
        Logs.PUSH.warn("push message to client offline, but gateway connection is closed, timePoints={}, message={}"
                , Jsons.toJson(timePoints), message);
    }
}
```

关于redirect，如果发送消息给pushclient的连接被关闭，也就无法告知其redirect状态；



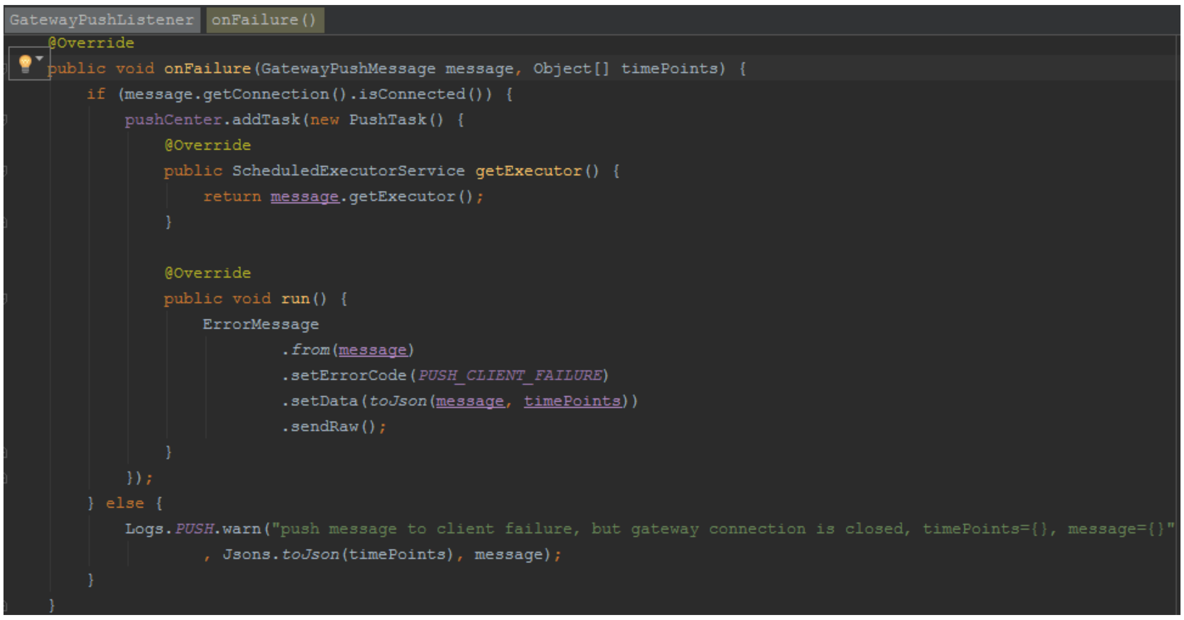
```java
@Override
public void onRedirect(GatewayPushMessage message, Object[] timePoints) {
    if (message.getConnection().isConnected()) {
        pushCenter.addTask(new PushTask() {
            @Override
            public ScheduledExecutorService getExecutor() {
                return message.getExecutor();
            }

            @Override
            public void run() {
                ErrorMessage
                        .from(message)
                        .setErrorCode(ROUTER_CHANGE)
                        .setData(toJson(message, timePoints))
                        .sendRaw();
            }
        });
    } else {
        Logs.PUSH.warn("push message to client redirect, but gateway connection is closed, timePoints={}, message={}"
                , Jsons.toJson(timePoints), message);
    }
}
```

关于success，如果发送消息给pushclient的连接被关闭，也就无法告知其success状态；



```java
@Override
public void onSuccess(GatewayPushMessage message, Object[] timePoints) {
    if (message.getConnection().isConnected()) {
        pushCenter.addTask(new PushTask() {
            @Override
            public ScheduledExecutorService getExecutor() {
                return message.getExecutor();
            }

            @Override
            public void run() {
                OkMessage
                        .from(message)
                        .setData(toJson(message, timePoints))
                        .sendRaw();
            }
        });
    } else {
        Logs.PUSH.warn("push message to client success, but gateway connection is closed, timePoints={}, message={}"
                , Jsons.toJson(timePoints), message);
    }
}
```

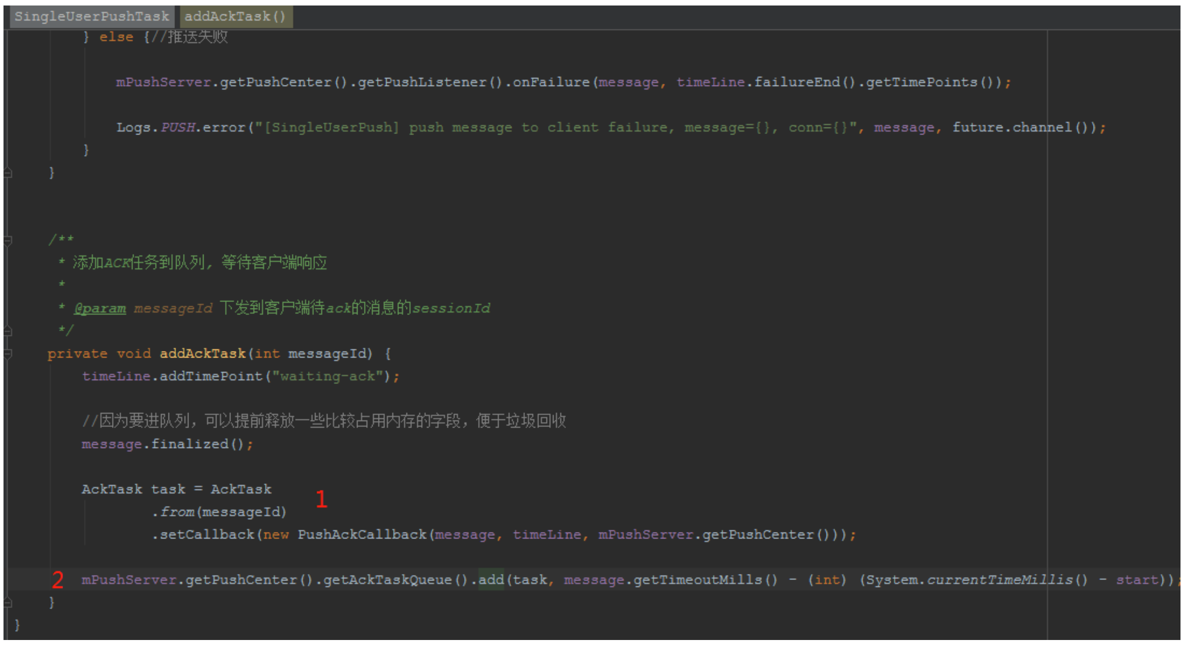
关于failure，如果发送消息给pushclient的连接被关闭，也就无法告知其failure状态；



GatewayPushListener　onFailure()

```java
@Override
public void onFailure(GatewayPushMessage message, Object[] timePoints) {
    if (message.getConnection().isConnected()) {
        pushCenter.addTask(new PushTask() {
            @Override
            public ScheduledExecutorService getExecutor() {
                return message.getExecutor();
            }

            @Override
            public void run() {
                ErrorMessage
                        .from(message)
                        .setErrorCode(PUSH_CLIENT_FAILURE)
                        .setData(toJson(message, timePoints))
                        .sendRaw();
            }
        });
    } else {
        Logs.PUSH.warn("push message to client failure, but gateway connection is closed, timePoints={}, message={}"
                , Jsons.toJson(timePoints), message);
    }
}
```

添加ACK任务

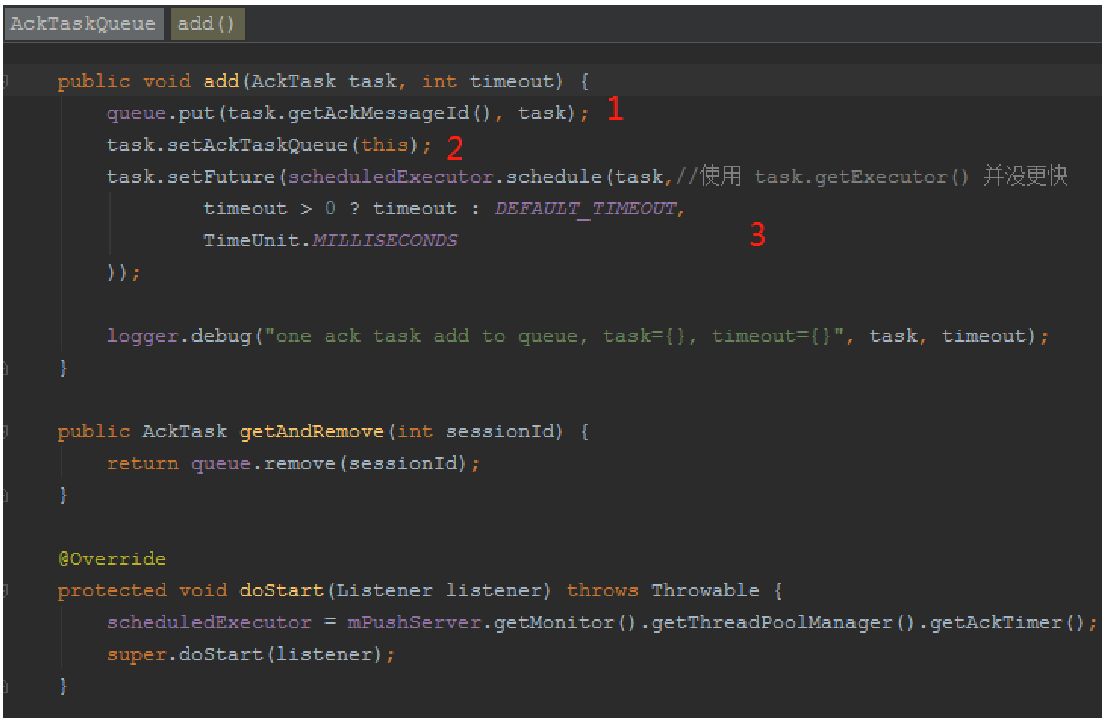


SingleUserPushTask　addAckTask()

```java
        } else {//推送失败

            mPushServer.getPushCenter().getPushListener().onFailure(message, timeLine.failureEnd().getTimePoints());

            Logs.PUSH.error("[SingleUserPush] push message to client failure, message={}, conn={}", message, future.channel());
        }
    }


    /**
     * 添加ACK任务到队列, 等待客户端响应
     *
     * @param messageId 下发到客户端待ack的消息的sessionId
     */
    private void addAckTask(int messageId) {
        timeLine.addTimePoint("waiting-ack");

        //因为要进队列，可以提前释放一些比较占用内存的字段，便于垃圾回收
        message.finalized();

        AckTask task = AckTask          1
                .from(messageId)
                .setCallback(new PushAckCallback(message, timeLine, mPushServer.getPushCenter()));

  2     mPushServer.getPushCenter().getAckTaskQueue().add(task, message.getTimeoutMills() - (int) (System.currentTimeMillis() - start))
    }
}
```

1、封装AckTask任务，设置回调

2、添加任务到AckTaskQueue中



```java
    public void add(AckTask task, int timeout) {
        queue.put(task.getAckMessageId(), task); 1
        task.setAckTaskQueue(this); 2
        task.setFuture(scheduledExecutor.schedule(task,//使用 task.getExecutor() 并没更快
                timeout > 0 ? timeout : DEFAULT_TIMEOUT,
                TimeUnit.MILLISECONDS                      3
        ));

        logger.debug("one ack task add to queue, task={}, timeout={}", task, timeout);
    }

    public AckTask getAndRemove(int sessionId) {
        return queue.remove(sessionId);
    }

    @Override
    protected void doStart(Listener listener) throws Throwable {
        scheduledExecutor = mPushServer.getMonitor().getThreadPoolManager().getAckTimer();
        super.doStart(listener);
    }
```

1、将任务缓存在MAP中

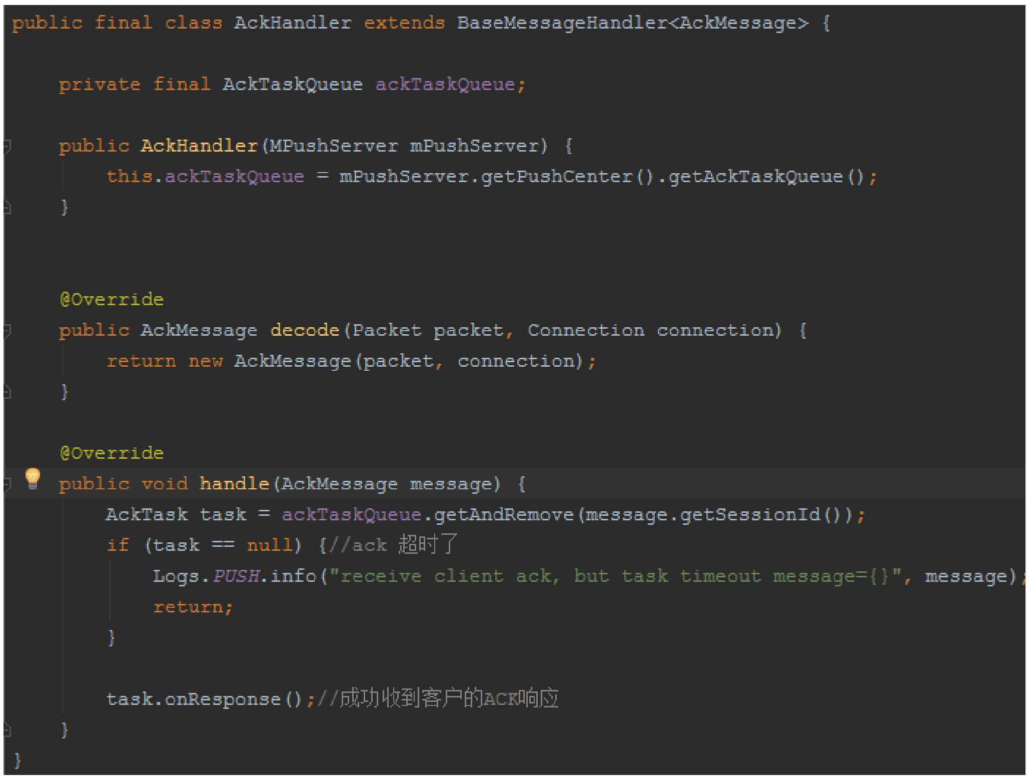
2、将AckTaskQueue实例对象设置到AckTask中

3、用ACK线程池，创建一个超时任务，并且设置到AckTask

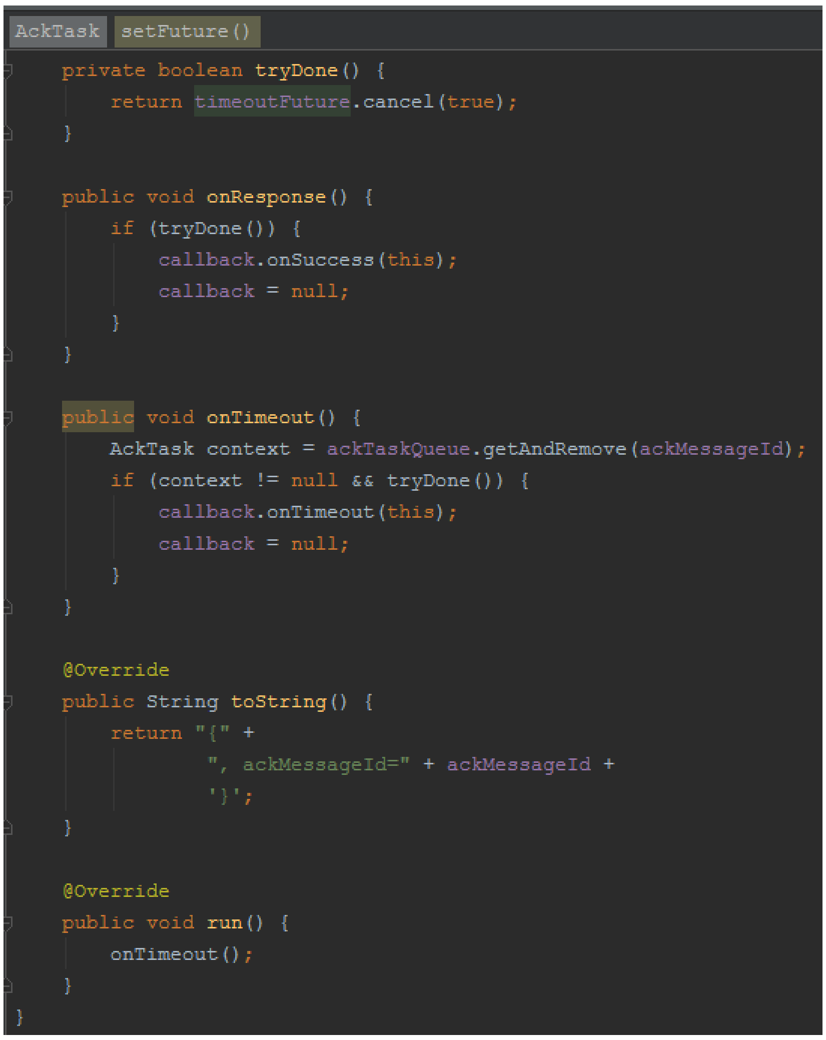
客户端返回ACK消息

手机客户端返回ACK消息给connection server（注意：不是Gateway server）;

ConnectionServer#init()中有指定ACK的handle类AckHandler



```java
public final class AckHandler extends BaseMessageHandler<AckMessage> {

    private final AckTaskQueue ackTaskQueue;

    public AckHandler(MPushServer mPushServer) {
        this.ackTaskQueue = mPushServer.getPushCenter().getAckTaskQueue();
    }


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

    @Override
    public void handle(AckMessage message) {
        AckTask task = ackTaskQueue.getAndRemove(message.getSessionId());
        if (task == null) {//ack 超时了
            Logs.PUSH.info("receive client ack, but task timeout message={}", message)
            return;
        }

        task.onResponse();//成功收到客户的ACK响应
    }
}
```

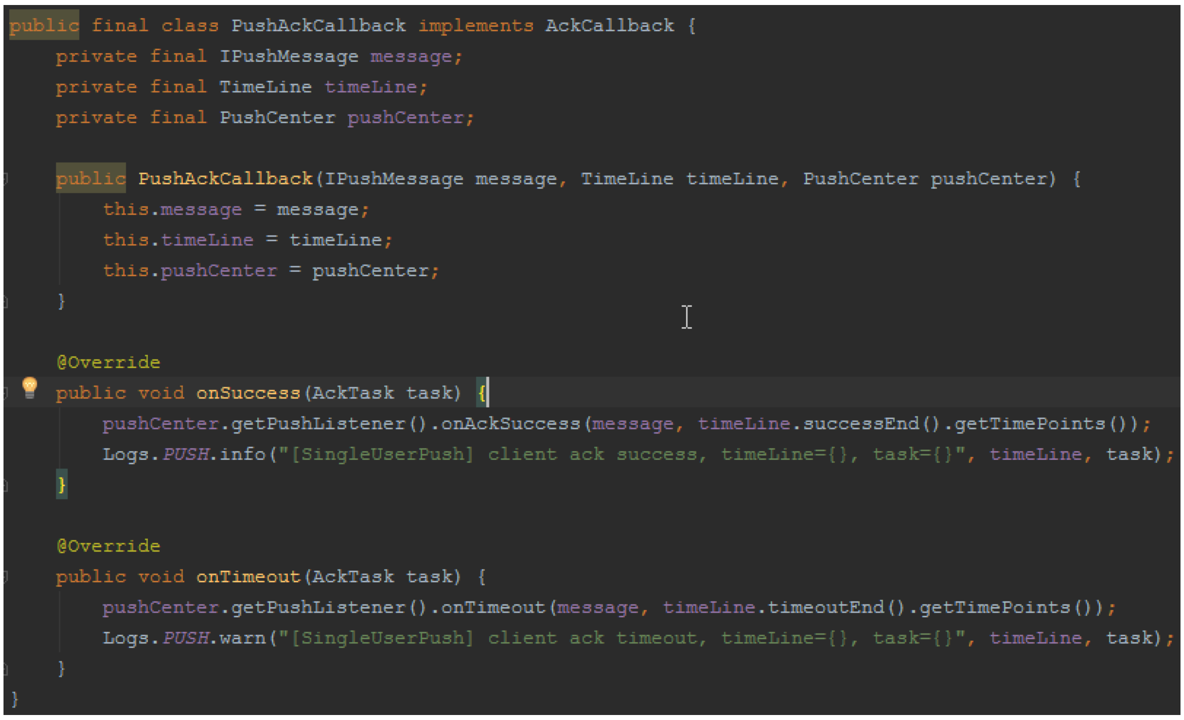


```
AckTask  setFuture()
    private boolean tryDone() {
        return timeoutFuture.cancel(true);
    }

    public void onResponse() {
        if (tryDone()) {
            callback.onSuccess(this);
            callback = null;
        }
    }

    public void onTimeout() {
        AckTask context = ackTaskQueue.getAndRemove(ackMessageId);
        if (context != null && tryDone()) {
            callback.onTimeout(this);
            callback = null;
        }
    }

    @Override
    public String toString() {
        return "{" +
                ", ackMessageId=" + ackMessageId +
                '}';
    }

    @Override
    public void run() {
        onTimeout();
    }
}
```

两个动作：

1、之前创建的ACK超时任务，超时后会调用AckTask#run()方法；

2、服务接收到ACK消息时，调用AckHandler#handle()方法，然后再调用AckTask#onResponse()方法；



```
public final class PushAckCallback implements AckCallback {
    private final IPushMessage message;
    private final TimeLine timeLine;
    private final PushCenter pushCenter;

    public PushAckCallback(IPushMessage message, TimeLine timeLine, PushCenter pushCenter) {
        this.message = message;
        this.timeLine = timeLine;
        this.pushCenter = pushCenter;
    }

    @Override
    public void onSuccess(AckTask task) {
        pushCenter.getPushListener().onAckSuccess(message, timeLine.successEnd().getTimePoints());
        Logs.PUSH.info("[SingleUserPush] client ack success, timeLine={}, task={}", timeLine, task);
    }

    @Override
    public void onTimeout(AckTask task) {
        pushCenter.getPushListener().onTimeout(message, timeLine.timeoutEnd().getTimePoints());
        Logs.PUSH.warn("[SingleUserPush] client ack timeout, timeLine={}, task={}", timeLine, task);
    }
}
```