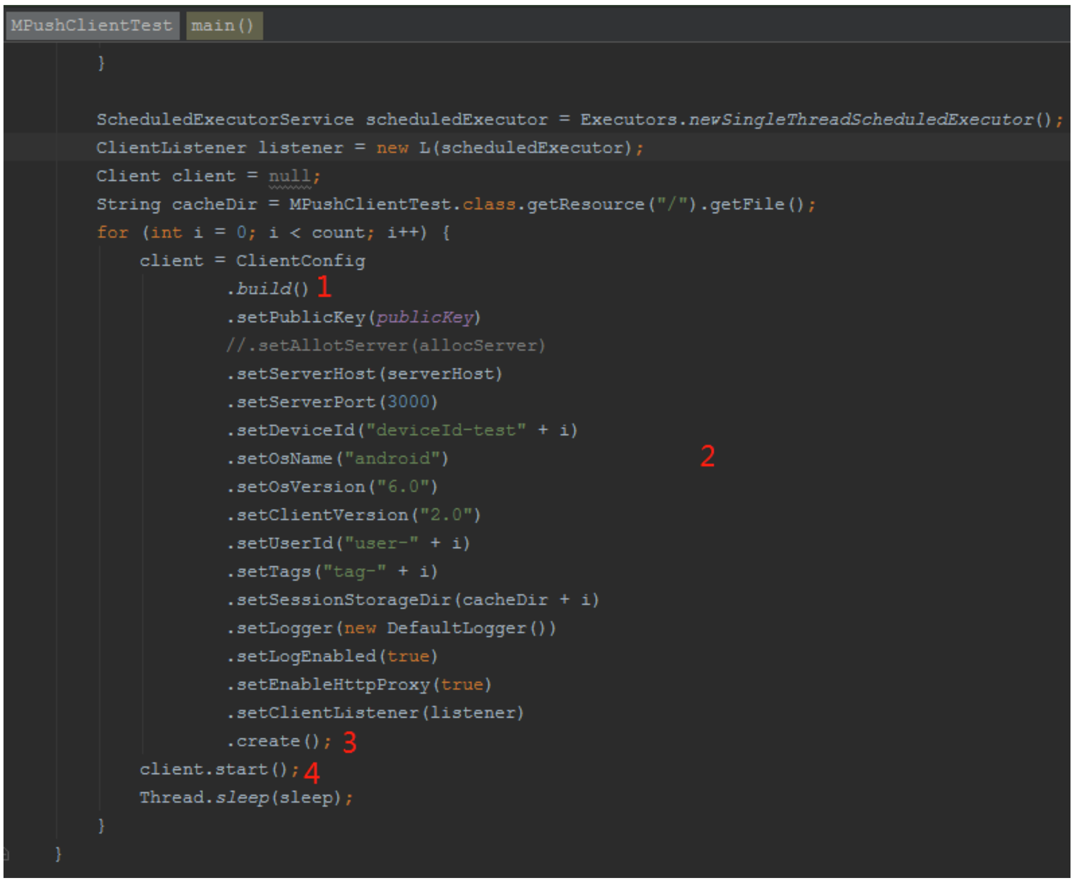
MPushClientTest.java



```
        }

        ScheduledExecutorService scheduledExecutor = Executors.newSingleThreadScheduledExecutor();
        ClientListener listener = new L(scheduledExecutor);
        Client client = null;
        String cacheDir = MPushClientTest.class.getResource("/").getFile();
        for (int i = 0; i < count; i++) {
            client = ClientConfig
                    .build() 1
                    .setPublicKey(publicKey)
                    //.setAllotServer(allocServer)
                    .setServerHost(serverHost)
                    .setServerPort(3000)
                    .setDeviceId("deviceId-test" + i)
                    .setOsName("android")                  2
                    .setOsVersion("6.0")
                    .setClientVersion("2.0")
                    .setUserId("user-" + i)
                    .setTags("tag-" + i)
                    .setSessionStorageDir(cacheDir + i)
                    .setLogger(new DefaultLogger())
                    .setLogEnabled(true)
                    .setEnableHttpProxy(true)
                    .setClientListener(listener)
                    .create(); 3
            client.start(); 4
            Thread.sleep(sleep);
        }
    }
```

1、初始化 ClientConfig 实例

2、设置ClientConfig 的各种参数

publicKey 公钥

allotServer 负载均衡服务的地址

serverHost 推送服务mpush的主机IP(如果allotServer为空，则会采用此值直连)

serverPort   推送服务mpush的主机PORT(如果allotServer为空，则会采用此值直连)

deviceId 设备ID

osName 操作系统名称

osVersion 操作系统版本

clientVersion 客户端版本

userId 用户ID

tags 用户标签

sessionStorageDir 会话存储目录

logger 日志实现

logEnabled 启用日志

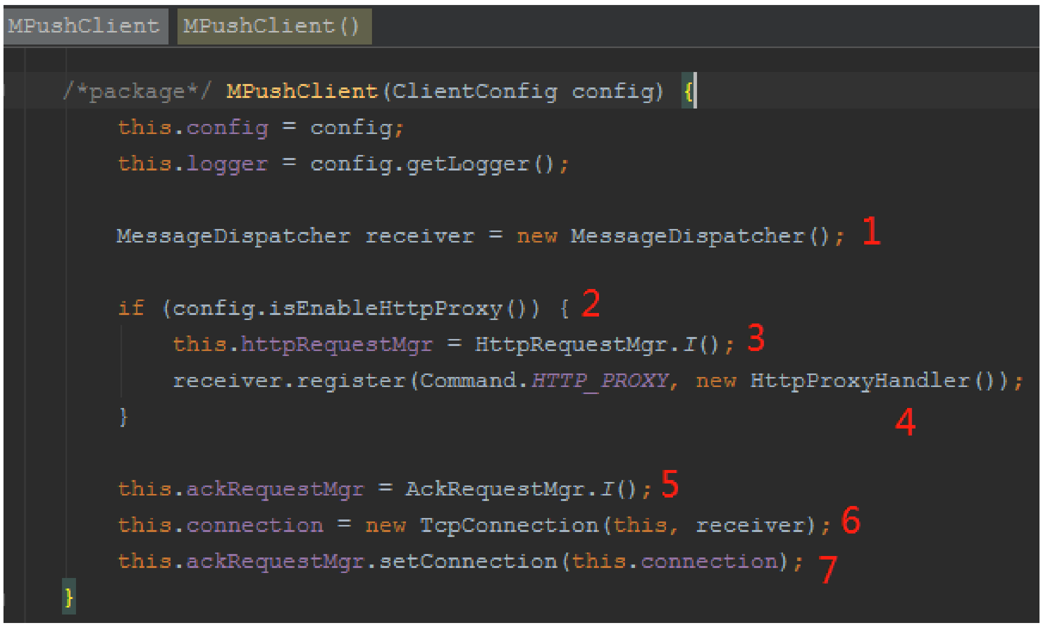
enableHttpProxy 启用HTTP代理

clientListenner 客户端监听器

3、初始化 MPushClient 实例

4、开始与推送服务端建立连接

初始化 MPushClient 实例

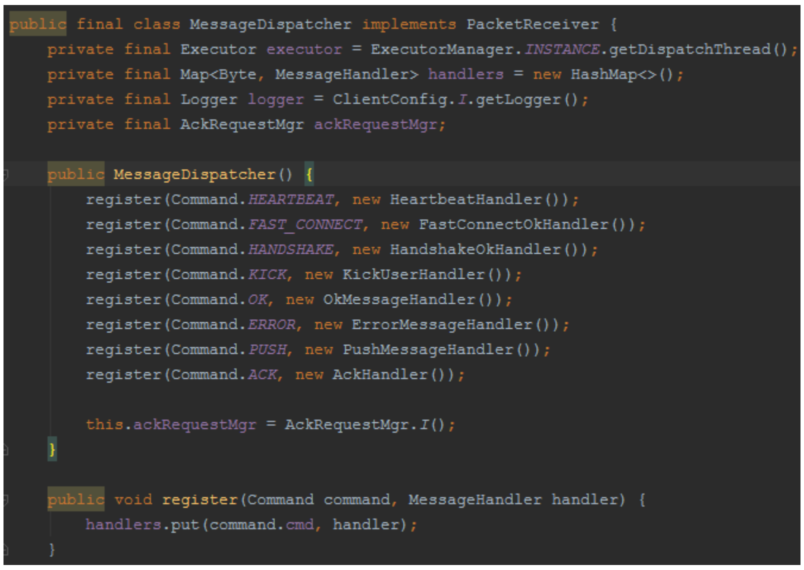


```
/*package*/ MPushClient(ClientConfig config) {
    this.config = config;
    this.logger = config.getLogger();

    MessageDispatcher receiver = new MessageDispatcher(); 1

    if (config.isEnableHttpProxy()) { 2
        this.httpRequestMgr = HttpRequestMgr.I(); 3
        receiver.register(Command.HTTP_PROXY, new HttpProxyHandler());
    }                                                                4

    this.ackRequestMgr = AckRequestMgr.I(); 5
    this.connection = new TcpConnection(this, receiver); 6
    this.ackRequestMgr.setConnection(this.connection); 7
}
```

1、初始化消息转发处理器MessageDispatcher

注册各种事件处理器，包括心跳、快连、握手、踢人、成功、错误、推送、确认；

获取AckRequestMgr对象；



```
public final class MessageDispatcher implements PacketReceiver {
    private final Executor executor = ExecutorManager.INSTANCE.getDispatchThread();
    private final Map<Byte, MessageHandler> handlers = new HashMap<>();
    private final Logger logger = ClientConfig.I.getLogger();
    private final AckRequestMgr ackRequestMgr;

    public MessageDispatcher() {
        register(Command.HEARTBEAT, new HeartbeatHandler());
        register(Command.FAST_CONNECT, new FastConnectOkHandler());
        register(Command.HANDSHAKE, new HandshakeOkHandler());
        register(Command.KICK, new KickUserHandler());
        register(Command.OK, new OkMessageHandler());
        register(Command.ERROR, new ErrorMessageHandler());
        register(Command.PUSH, new PushMessageHandler());
        register(Command.ACK, new AckHandler());

        this.ackRequestMgr = AckRequestMgr.I();
    }

    public void register(Command command, MessageHandler handler) {
        handlers.put(command.cmd, handler);
    }
```

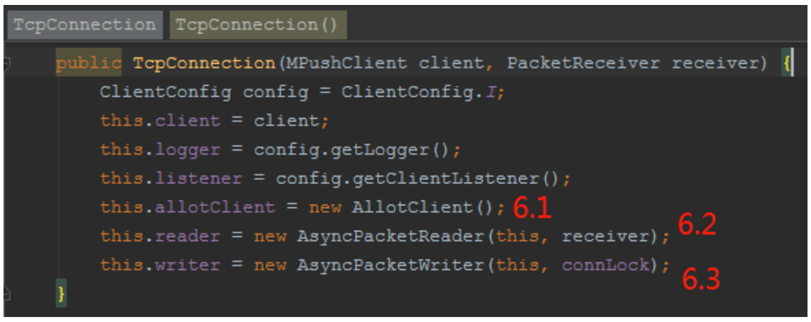
2、如果启用了HTTP代理

3、初始化HTTP请求处理器HttpRequestMgr

4、注册HTTP代理处理类HttpProxyHandler 给消息转发处理器MessageDispatcher

5、初始化ACK请求处理器AckRequestMgr

6、初始化连接管理TcpConnection



```
public TcpConnection(MPushClient client, PacketReceiver receiver) {
    ClientConfig config = ClientConfig.I;
    this.client = client;
    this.logger = config.getLogger();
    this.listener = config.getClientListener();
    this.allotClient = new AllotClient(); 6.1
    this.reader = new AsyncPacketReader(this, receiver); 6.2
    this.writer = new AsyncPacketWriter(this, connLock); 6.3
}
```

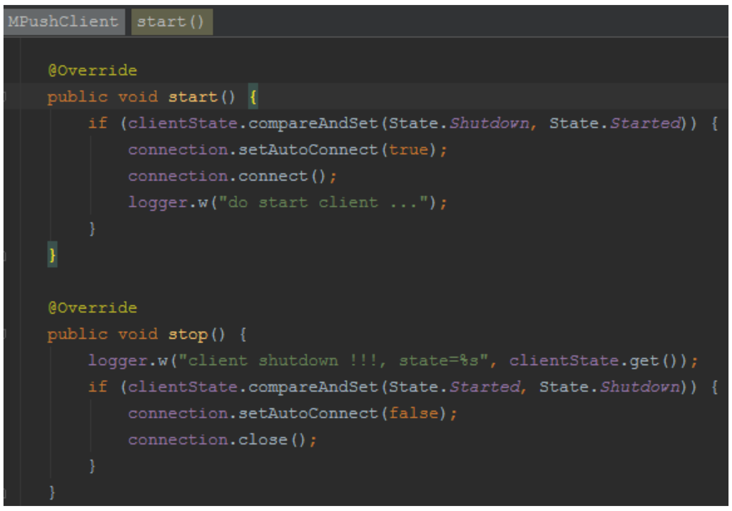
6.1 初始化负载均衡服务AllotClient

6.2 读数据，读来自推送服务端的消息

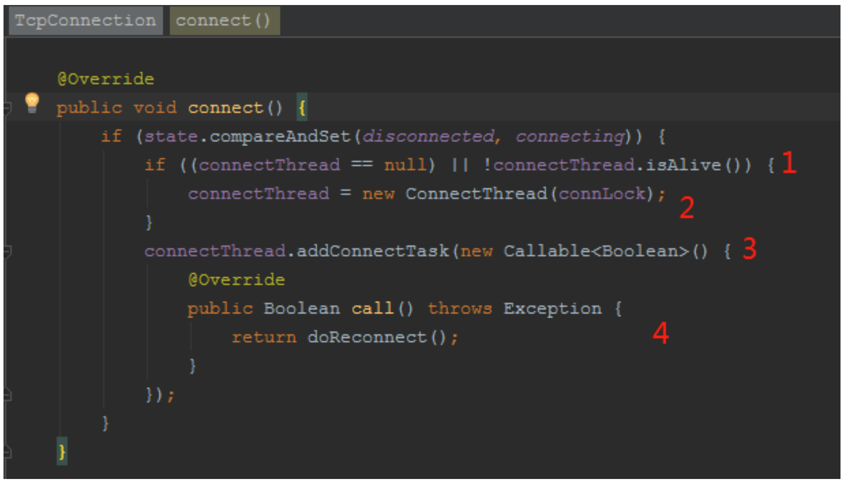
6.3 写数据，写消息给推送服务端

7、设置TcpConnection对象给ACK请求处理器AckRequestMgr

开始与推送服务端建立连接



```java
@Override
public void start() {
    if (clientState.compareAndSet(State.Shutdown, State.Started)) {
        connection.setAutoConnect(true);
        connection.connect();
        logger.w("do start client ...");
    }
}

@Override
public void stop() {
    logger.w("client shutdown !!!, state=%s", clientState.get());
    if (clientState.compareAndSet(State.Started, State.Shutdown)) {
        connection.setAutoConnect(false);
        connection.close();
    }
}
```

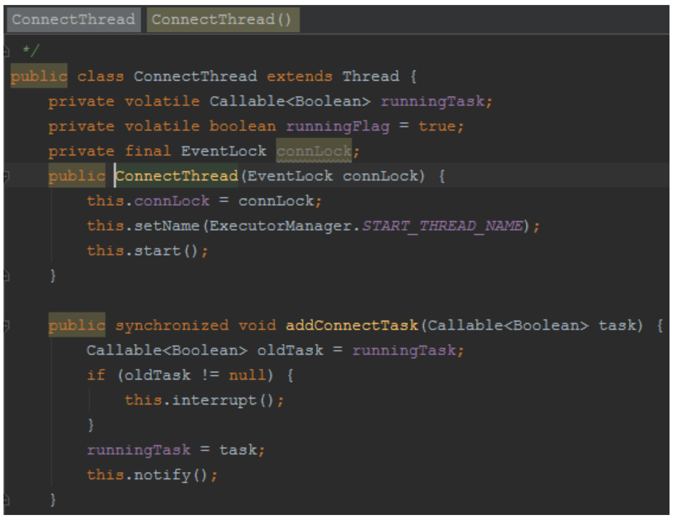


```java
@Override
public void connect() {
    if (state.compareAndSet(disconnected, connecting)) {
        if ((connectThread == null) || !connectThread.isAlive()) {
            connectThread = new ConnectThread(connLock);
        }
        connectThread.addConnectTask(new Callable<Boolean>() {
            @Override
            public Boolean call() throws Exception {
                return doReconnect();
            }
        });
    }
}
```

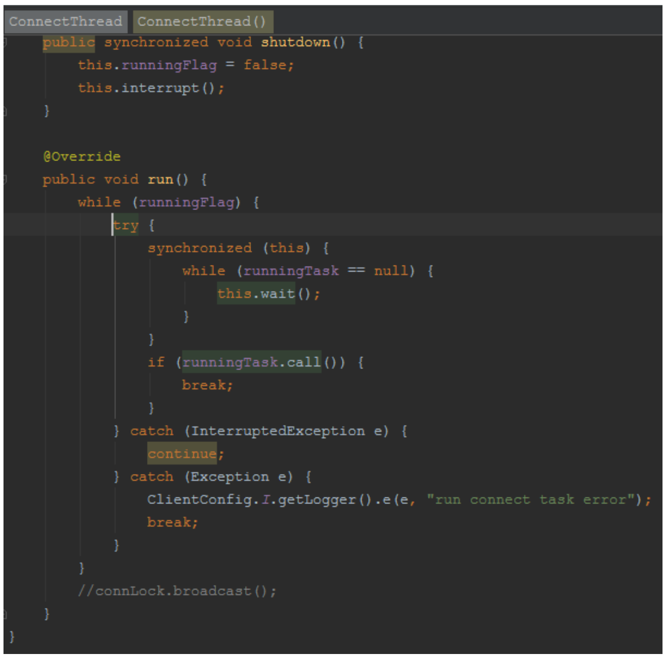
1、如果当前连接线程为空，或者不是存活的

2、创建一个连接线程ConnectThread对象

启动线程，将会调用run()方法，如果当前连接任务为空，则进入wait()阻塞状态；



```java
 */
public class ConnectThread extends Thread {
    private volatile Callable<Boolean> runningTask;
    private volatile boolean runningFlag = true;
    private final EventLock connLock;
    public ConnectThread(EventLock connLock) {
        this.connLock = connLock;
        this.setName(ExecutorManager.START_THREAD_NAME);
        this.start();
    }

    public synchronized void addConnectTask(Callable<Boolean> task) {
        Callable<Boolean> oldTask = runningTask;
        if (oldTask != null) {
            this.interrupt();
        }
        runningTask = task;
        this.notify();
    }
```



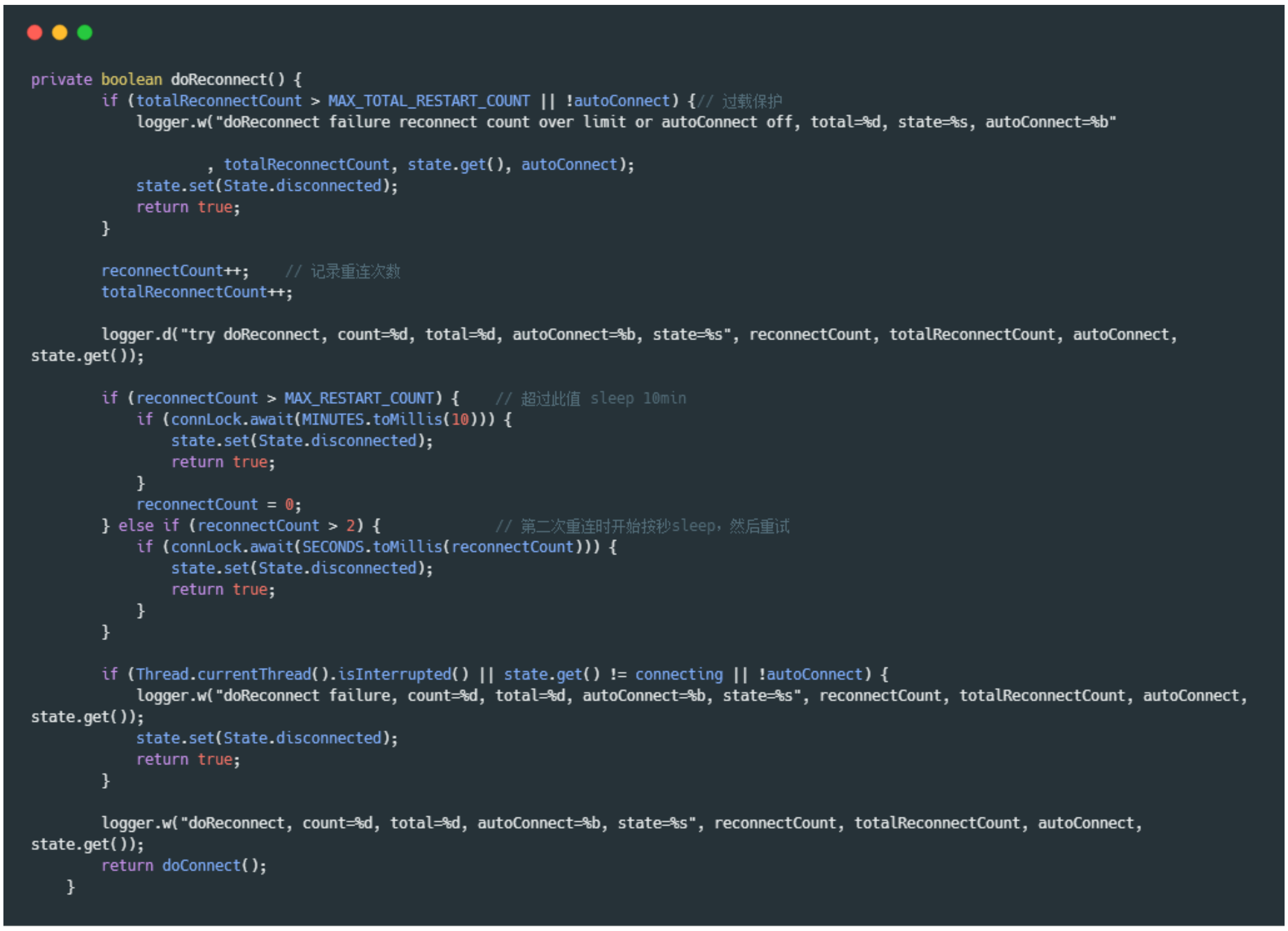
```java
    public synchronized void shutdown() {
        this.runningFlag = false;
        this.interrupt();
    }

    @Override
    public void run() {
        while (runningFlag) {
            try {
                synchronized (this) {
                    while (runningTask == null) {
                        this.wait();
                    }
                }
                if (runningTask.call()) {
                    break;
                }
            } catch (InterruptedException e) {
                continue;
            } catch (Exception e) {
                ClientConfig.I.getLogger().e(e, "run connect task error");
                break;
            }
        }
        //connLock.broadcast();
    }
}
```

3、添加连接任务

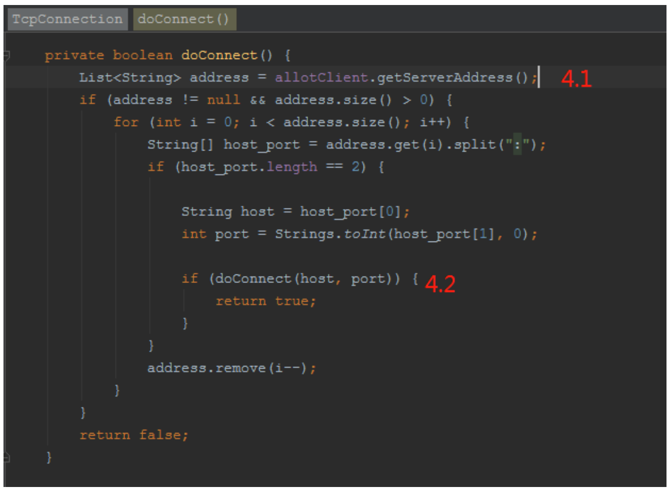
如果存在老的任务，则覆盖老的任务；

唤醒当前wait()状态的任务，调用任务的call()方法；

4、调用doReconnect()与推送服务端建立连接



```java
private boolean doReconnect() {
        if (totalReconnectCount > MAX_TOTAL_RESTART_COUNT || !autoConnect) {// 过载保护
            logger.w("doReconnect failure reconnect count over limit or autoConnect off, total=%d, state=%s, autoConnect=%b"

                    , totalReconnectCount, state.get(), autoConnect);
            state.set(State.disconnected);
            return true;
        }

        reconnectCount++;    // 记录重连次数
        totalReconnectCount++;

        logger.d("try doReconnect, count=%d, total=%d, autoConnect=%b, state=%s", reconnectCount, totalReconnectCount, autoConnect,
state.get());

        if (reconnectCount > MAX_RESTART_COUNT) {    // 超过此值 sleep 10min
            if (connLock.await(MINUTES.toMillis(10))) {
                state.set(State.disconnected);
                return true;
            }
            reconnectCount = 0;
        } else if (reconnectCount > 2) {             // 第二次重连时开始按秒sleep，然后重试
            if (connLock.await(SECONDS.toMillis(reconnectCount))) {
                state.set(State.disconnected);
                return true;
            }
        }

        if (Thread.currentThread().isInterrupted() || state.get() != connecting || !autoConnect) {
            logger.w("doReconnect failure, count=%d, total=%d, autoConnect=%b, state=%s", reconnectCount, totalReconnectCount, autoConnect,
state.get());
            state.set(State.disconnected);
            return true;
        }

        logger.w("doReconnect, count=%d, total=%d, autoConnect=%b, state=%s", reconnectCount, totalReconnectCount, autoConnect,
state.get());
        return doConnect();
    }
```

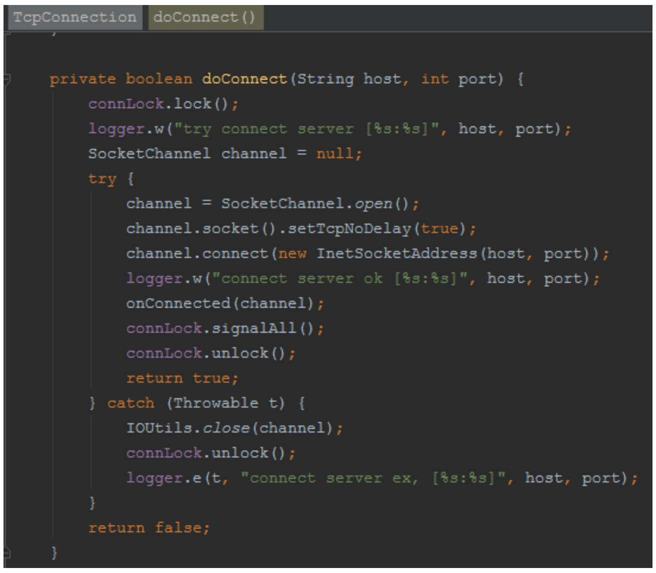


```java
private boolean doConnect() {
    List<String> address = allotClient.getServerAddress();
    if (address != null && address.size() > 0) {
        for (int i = 0; i < address.size(); i++) {
            String[] host_port = address.get(i).split(":");
            if (host_port.length == 2) {

                String host = host_port[0];
                int port = Strings.toInt(host_port[1], 0);

                if (doConnect(host, port)) {
                    return true;
                }
            }
            address.remove(i--);
        }
    }
    return false;
}
```

获取可用的MPUSH server列表,然后按顺序去尝试建立TCP链接,直到链接建立成功

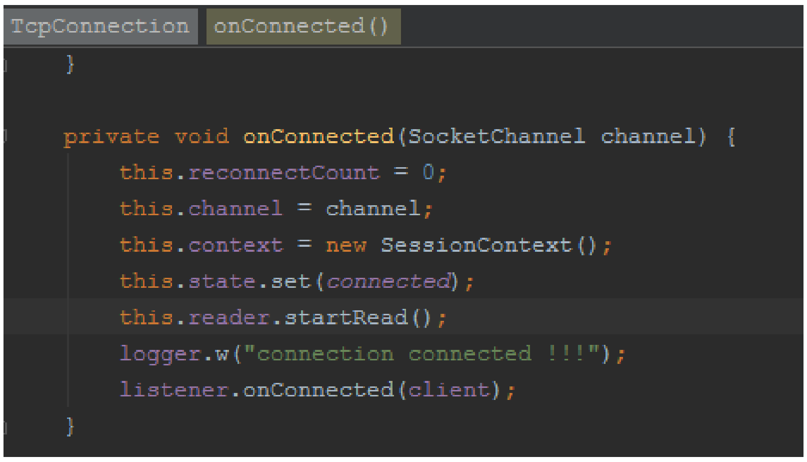
4.1 从负载均衡服务ALLO拿到推送服务地址列表

4.2 与推送服务端建立socket连接

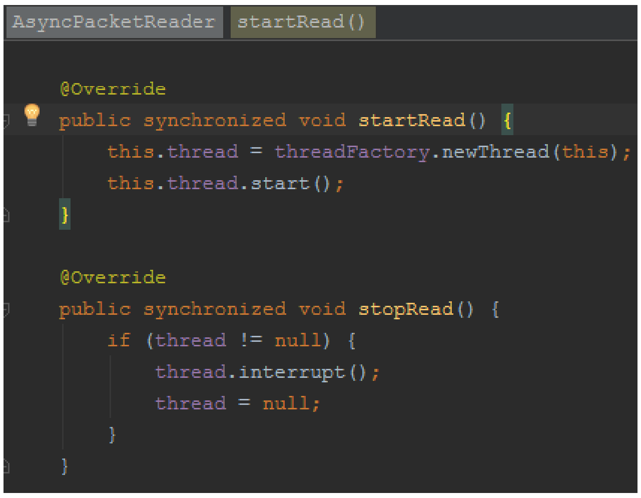


```java
    private boolean doConnect(String host, int port) {
        connLock.lock();
        logger.w("try connect server [%s:%s]", host, port);
        SocketChannel channel = null;
        try {
            channel = SocketChannel.open();
            channel.socket().setTcpNoDelay(true);
            channel.connect(new InetSocketAddress(host, port));
            logger.w("connect server ok [%s:%s]", host, port);
            onConnected(channel);
            connLock.signalAll();
            connLock.unlock();
            return true;
        } catch (Throwable t) {
            IOUtils.close(channel);
            connLock.unlock();
            logger.e(t, "connect server ex, [%s:%s]", host, port);
        }
        return false;
    }
```

开始读取数据，调用startRead()方法



```java
    }

    private void onConnected(SocketChannel channel) {
        this.reconnectCount = 0;
        this.channel = channel;
        this.context = new SessionContext();
        this.state.set(connected);
        this.reader.startRead();
        logger.w("connection connected !!!");
        listener.onConnected(client);
    }
```



```java
    @Override
    public synchronized void startRead() {
        this.thread = threadFactory.newThread(this);
        this.thread.start();
    }

    @Override
    public synchronized void stopRead() {
        if (thread != null) {
            thread.interrupt();
            thread = null;
        }
    }
```