1、发送握手消息 HandshakeMessage

2、接收握手成功消息 HandshakeOkMessage

发送握手消息

与MPUSH服务端建立连接时，发送握手消息

```
//ConnClientChannelHandler.java

// 创建conn对象
private final Connection connection = new NettyConnection();
//建立连接事件方法
@Override
public void channelActive(ChannelHandlerContext ctx) throws Exception {
 int clientNum = STATISTICS.clientNum.incrementAndGet();
 LOGGER.info("client connect channel={}, clientNum={}", ctx.channel(), clientNum);
  for (int i = 0; i < 3; i++) {
  if (clientConfig != null) break;
  clientConfig = ctx.channel().attr(CONFIG_KEY).getAndSet(null);
  if (clientConfig == null) TimeUnit.SECONDS.sleep(1);
  }
  if (clientConfig == null) {
  throw new NullPointerException("client config is null, channel=" + ctx.channel());
  }
  connection.init(ctx.channel(), true);//初始化sessionContext、设置RSA加密
  if (perfTest) {
  handshake(); //握手
  } else {
  tryFastConnect();
  }
 }
 //发送握手消息
 private void handshake() {
  HandshakeMessage message = new HandshakeMessage(connection);
  message.clientKey = clientConfig.getClientKey();
  message.iv = clientConfig.getIv();
  message.clientVersion = clientConfig.getClientVersion();
  message.deviceId = clientConfig.getDeviceId();
  message.osName = clientConfig.getOsName();
```

```java
    message.osVersion = clientConfig.getOsVersion();
    message.timestamp = System.currentTimeMillis();
    message.send();
    LOGGER.debug("send handshake message={}", message);
  }
```

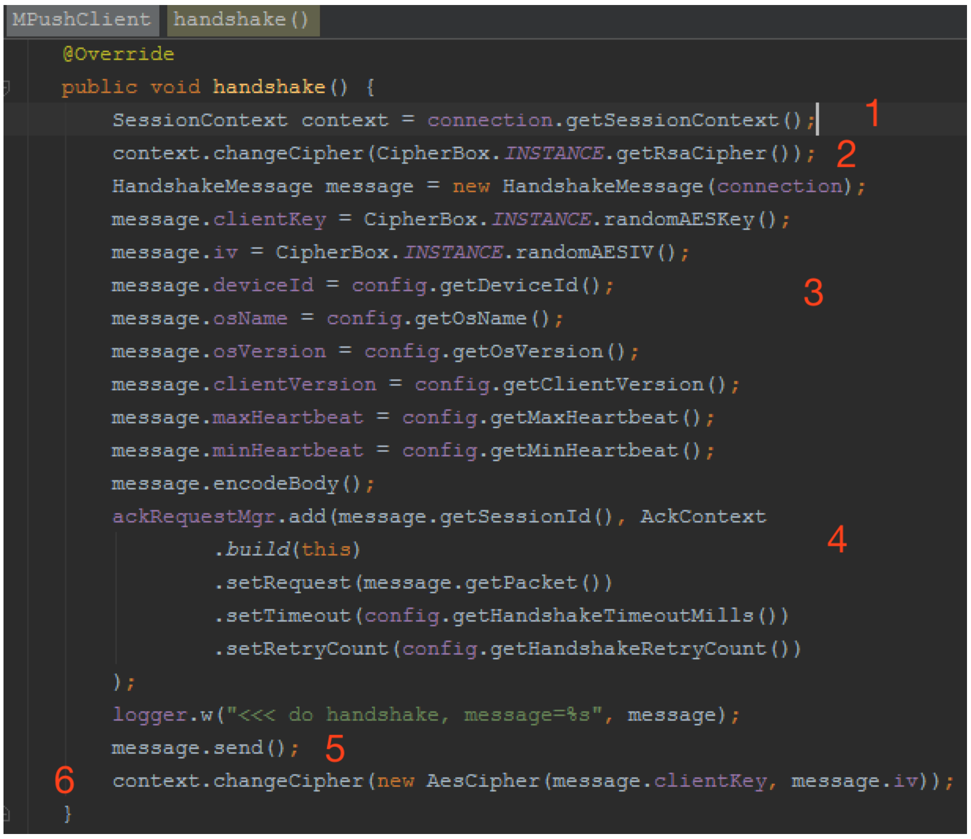
上面发送握手消息前，需要设置RSA加密

```java
//NettyConnection#init()

@Override
public void init(Channel channel, boolean security) {
  this.channel = channel;
  this.context = new SessionContext();
  this.lastReadTime = System.currentTimeMillis();
  this.status = STATUS_CONNECTED;
  if (security) {
    //设置RSA加密
    this.context.changeCipher(RsaCipherFactory.create());
  }
}
```

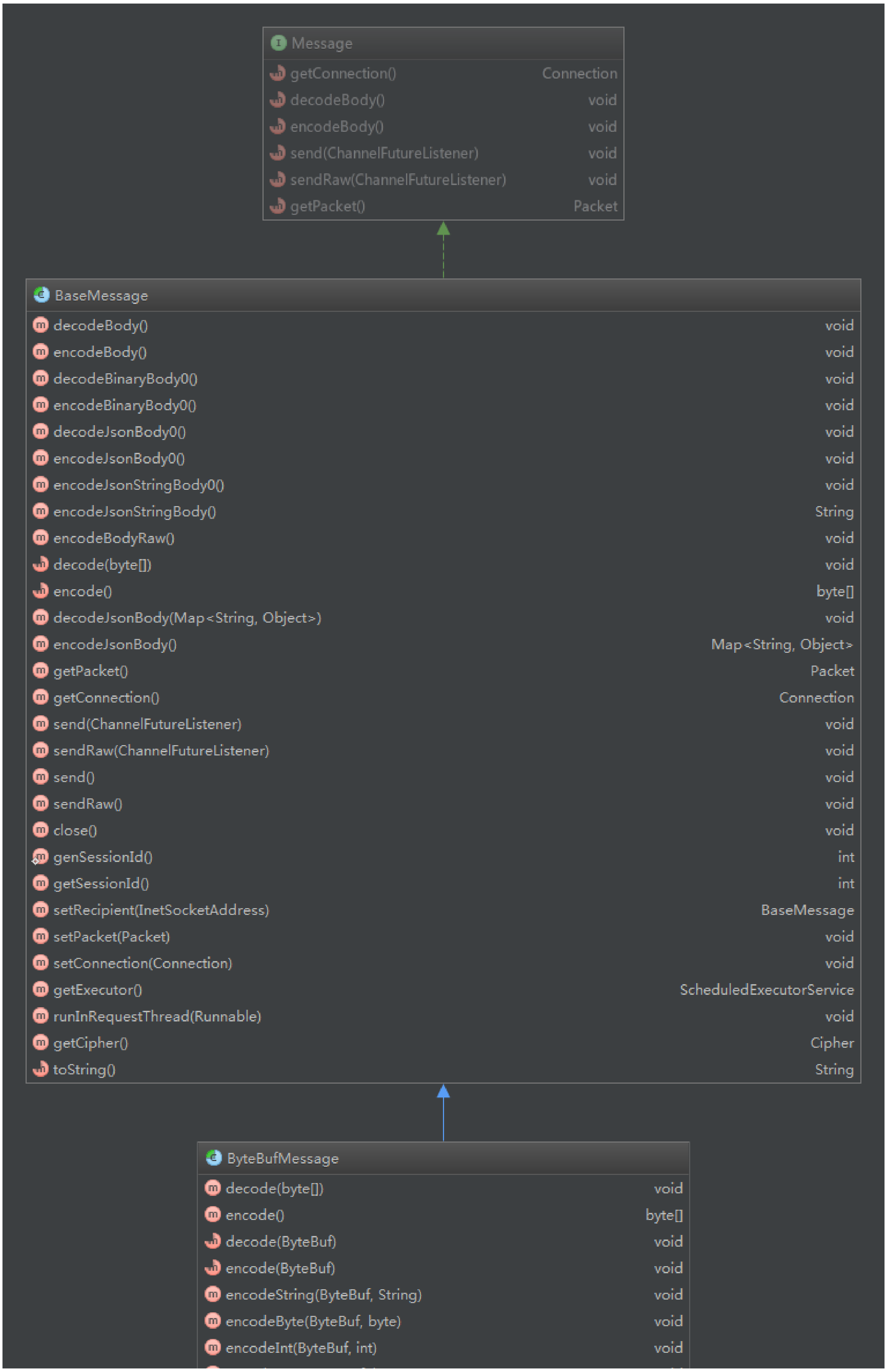
消息的加解密流程，参考《深度进阶-加解密》章节

握手消息加密流程，参考mpush-client-java工程里对发送握手消息的处理《4 握手.note》



```java
@Override
public void handshake() {
    SessionContext context = connection.getSessionContext();
    context.changeCipher(CipherBox.INSTANCE.getRsaCipher());
    HandshakeMessage message = new HandshakeMessage(connection);
    message.clientKey = CipherBox.INSTANCE.randomAESKey();
    message.iv = CipherBox.INSTANCE.randomAESIV();
    message.deviceId = config.getDeviceId();
    message.osName = config.getOsName();
    message.osVersion = config.getOsVersion();
    message.clientVersion = config.getClientVersion();
    message.maxHeartbeat = config.getMaxHeartbeat();
    message.minHeartbeat = config.getMinHeartbeat();
    message.encodeBody();
    ackRequestMgr.add(message.getSessionId(), AckContext
            .build(this)
            .setRequest(message.getPacket())
            .setTimeout(config.getHandshakeTimeoutMills())
            .setRetryCount(config.getHandshakeRetryCount())
    );
    logger.w("<<< do handshake, message=%s", message);
    message.send();
    context.changeCipher(new AesCipher(message.clientKey, message.iv));
}
```

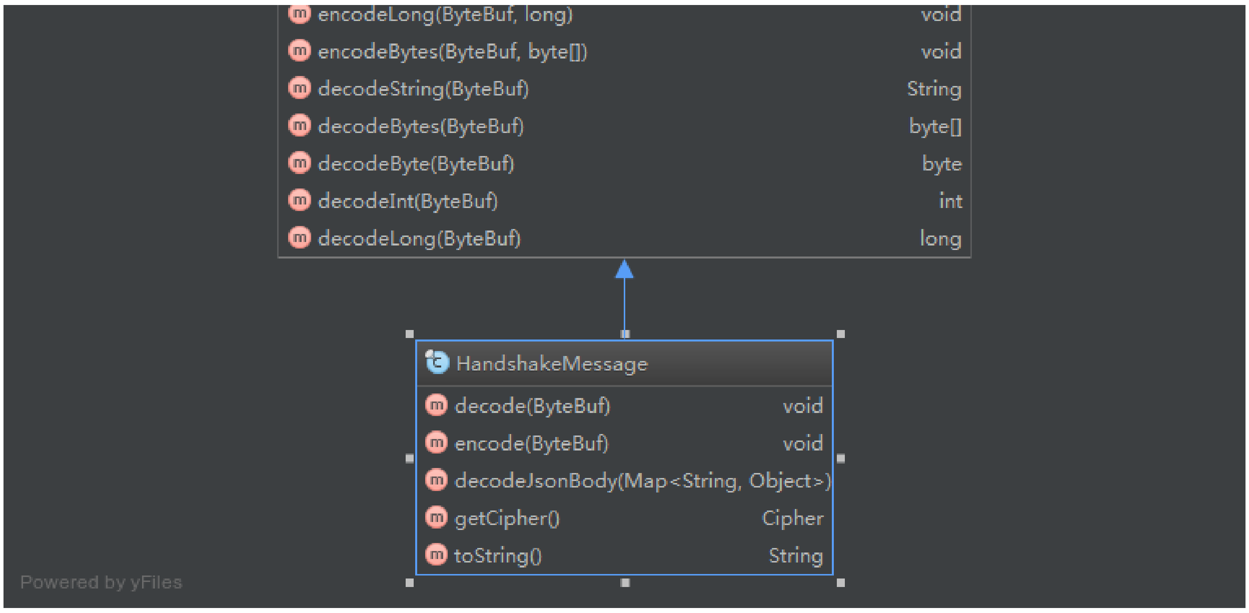
HandshakeMessage的继承关系



Message	
getConnection()	Connection
decodeBody()	void
encodeBody()	void
send(ChannelFutureListener)	void
sendRaw(ChannelFutureListener)	void
getPacket()	Packet

BaseMessage	
decodeBody()	void
encodeBody()	void
decodeBinaryBody0()	void
encodeBinaryBody0()	void
decodeJsonBody0()	void
encodeJsonBody0()	void
encodeJsonStringBody0()	void
encodeJsonStringBody()	String
encodeBodyRaw()	void
decode(byte[])	void
encode()	byte[]
decodeJsonBody(Map<String, Object>)	void
encodeJsonBody()	Map<String, Object>
getPacket()	Packet
getConnection()	Connection
send(ChannelFutureListener)	void
sendRaw(ChannelFutureListener)	void
send()	void
sendRaw()	void
close()	void
genSessionId()	int
getSessionId()	int
setRecipient(InetSocketAddress)	BaseMessage
setPacket(Packet)	void
setConnection(Connection)	void
getExecutor()	ScheduledExecutorService
runInRequestThread(Runnable)	void
getCipher()	Cipher
toString()	String

ByteBufMessage	
decode(byte[])	void
encode()	byte[]
decode(ByteBuf)	void
encode(ByteBuf)	void
encodeString(ByteBuf, String)	void
encodeByte(ByteBuf, byte)	void
encodeInt(ByteBuf, int)	void



```java
package com.mpush.common.message;

import com.mpush.api.connection.Cipher;
import com.mpush.api.connection.Connection;
import com.mpush.api.protocol.Packet;
import com.mpush.api.spi.core.RsaCipherFactory;
import io.netty.buffer.ByteBuf;

import java.util.Arrays;
import java.util.Map;

import static com.mpush.api.protocol.Command.HANDSHAKE;

/**
 * Created by ohun on 2015/12/24.
 *
 * @author ohun@live.cn
 */
public final class HandshakeMessage extends ByteBufMessage {
 public String deviceId;
 public String osName;
 public String osVersion;
 public String clientVersion;
 public byte[] iv;
 public byte[] clientKey;
 public int minHeartbeat;
 public int maxHeartbeat;
 public long timestamp;

 public HandshakeMessage(Connection connection) {
 super(new Packet(HANDSHAKE, genSessionId()), connection);
 }

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

 @Override
 public void decode(ByteBuf body) {
 deviceId = decodeString(body);
 osName = decodeString(body);
 osVersion = decodeString(body);
 clientVersion = decodeString(body);
 iv = decodeBytes(body);
 clientKey = decodeBytes(body);
 minHeartbeat = decodeInt(body);
 maxHeartbeat = decodeInt(body);
 timestamp = decodeLong(body);
 }

 public void encode(ByteBuf body) {
 encodeString(body, deviceId);
 encodeString(body, osName);
 encodeString(body, osVersion);
 encodeString(body, clientVersion);
 encodeBytes(body, iv);
 encodeBytes(body, clientKey);
 encodeInt(body, minHeartbeat);
 encodeInt(body, maxHeartbeat);
 encodeLong(body, timestamp);
 }

 @Override
 public void decodeJsonBody(Map<String, Object> body) {
 deviceId = (String) body.get("deviceId");
 osName = (String) body.get("osName");
 osVersion = (String) body.get("osVersion");
 clientVersion = (String) body.get("clientVersion");
 }

 @Override
 protected Cipher getCipher() {
 return RsaCipherFactory.create();
 }

 @Override
 public String toString() {
 return "HandshakeMessage{" +
 "clientKey=" + Arrays.toString(clientKey) +
 ", deviceId='" + deviceId + '\'' +
 ", osName='" + osName + '\'' +
 ", osVersion='" + osVersion + '\'' +
 ", clientVersion='" + clientVersion + '\'' +
 ", iv=" + Arrays.toString(iv) +
 ", minHeartbeat=" + minHeartbeat +
 ", maxHeartbeat=" + maxHeartbeat +
 ", timestamp=" + timestamp +
 ", packet=" + packet +
 '}';
 }
 }
```

接收握手成功消息

```
//ConnClientChannelHandler.java

@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exc
eption {
 connection.updateLastReadTime();
 if (msg instanceof Packet) {
 Packet packet = (Packet) msg;
 Command command = Command.toCMD(packet.cmd);
 if (command == Command.HANDSHAKE) {
  //统计连接数
  int connectedNum = STATISTICS.connectedNum.incrementAndGet();
  //设置AES加密
  connection.getSessionContext().changeCipher(new
AesCipher(clientConfig.getClientKey(), clientConfig.getIv()));
```

```java
    HandshakeOkMessage message = new HandshakeOkMessage(packet, connection);
    message.decodeBody();//解码body内容
    byte[] sessionKey = CipherBox.I.mixKey(clientConfig.getClientKey(), message.serverKey);
    connection.getSessionContext().changeCipher(new AesCipher(sessionKey, clientConfig.getIv()));
    connection.getSessionContext().setHeartbeat(message.heartbeat);
    //发送心跳
    startHeartBeat(message.heartbeat - 1000);
    LOGGER.info("handshake success, clientConfig={}, connectedNum={}", clientConfig, connectedNum);
    //绑定用户
    bindUser(clientConfig);
    if (!perfTest) {
    //保存session信息到redis，用于后续快速连接
    saveToRedisForFastConnection(clientConfig, message.sessionId, message.expireTime, sessionKey);
    }
    }
    }
  }
```

发送心跳消息：

```java
//ConnClientChannelHandler.java

private static final Timer HASHED_WHEEL_TIMER = new HashedWheelTimer(new NamedPoolThreadFactory(ThreadNames.T_CONN_TIMER));
//创建心跳任务
private void startHeartBeat(final int heartbeat) throws Exception {
  HASHED_WHEEL_TIMER.newTimeout(new TimerTask() {
  @Override
  public void run(Timeout timeout) throws Exception {
  //如果是连接状态，且健康检查成功，则继续心跳检测
    if (connection.isConnected() && healthCheck()) {
    HASHED_WHEEL_TIMER.newTimeout(this, heartbeat, TimeUnit.MILLISECONDS);
    }
    }
    }, heartbeat, TimeUnit.MILLISECONDS);
  }
  //健康检查
```

```java
//如果可读、可写，则是健康的;
//如果不可读，且超过2次计数，则断开连接;
//如果不可写，发送心跳包;
private int hbTimeoutTimes; //心跳超时次数
private boolean healthCheck() {
 //如果读取超时，累计超时次数
 if (connection.isReadTimeout()) {
 hbTimeoutTimes++;
 LOGGER.warn("heartbeat timeout times={}, client={}", hbTimeoutTimes, connection);
 } else {
 hbTimeoutTimes = 0;
 }
 //心跳超时次数超过2次，断开连接，返回健康检查失败
 if (hbTimeoutTimes >= 2) {
 LOGGER.warn("heartbeat timeout times={} over limit={}, client={}", hbTimeoutTimes, 2, connection);
 hbTimeoutTimes = 0;
 connection.close();
 return false;
 }
 //如果写超时(有可能网络不稳定或者网络断开)，发送心跳包
 if (connection.isWriteTimeout()) {
 LOGGER.info("send heartbeat ping...");
 connection.send(Packet.HB_PACKET); //发送心跳包
 }
 return true;
}
```

发送绑定用户消息：

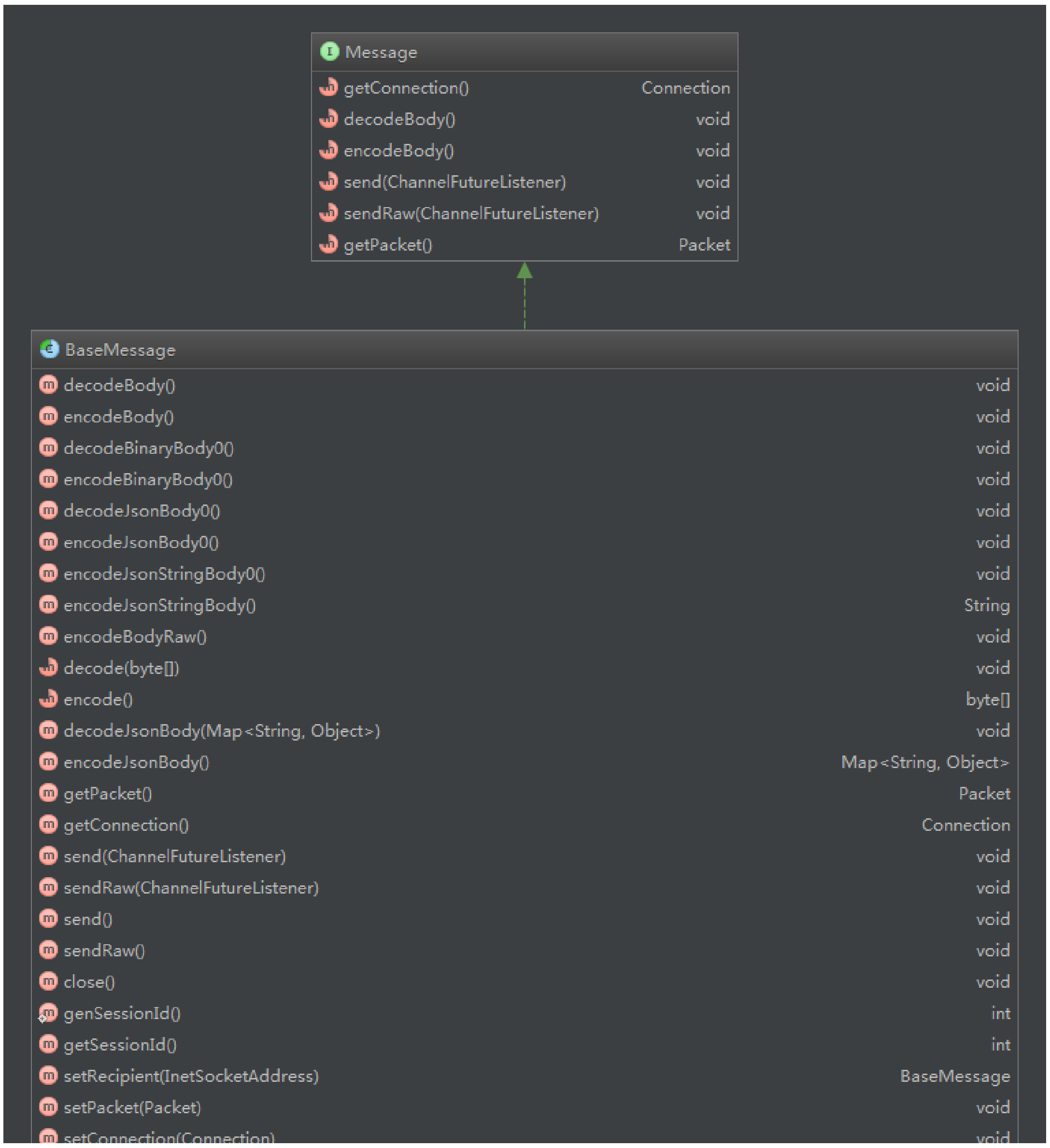
```java
//ConnClientChannelHandler.java
private void bindUser(ClientConfig client) {
 BindUserMessage message = new BindUserMessage(connection);
 message.userId = client.getUserId();
 message.tags = "test";
 message.send();
 //设置SessionContext上下文
 connection.getSessionContext().setUserId(client.getUserId());
 LOGGER.debug("send bind user message={}", message);
}
```

保存会话信息到redis：

```java
//ConnClientChannelHandler.java

//保存会话信息到redis，用于后续的快速连接
private void saveToRedisForFastConnection(ClientConfig client, String sessionId, Long expireTime, byte[] sessionKey) {
 Map<String, String> map = Maps.newHashMap();
 map.put("sessionId", sessionId);
 map.put("expireTime", expireTime + "");
 map.put("cipherStr", connection.getSessionContext().cipher.toString());
 String key = CacheKeys.getDeviceIdKey(client.getDeviceId());
  cacheManager.set(key, map, 60 * 5); //5分钟
}
```

HandshakeOkMessage 的继承关系

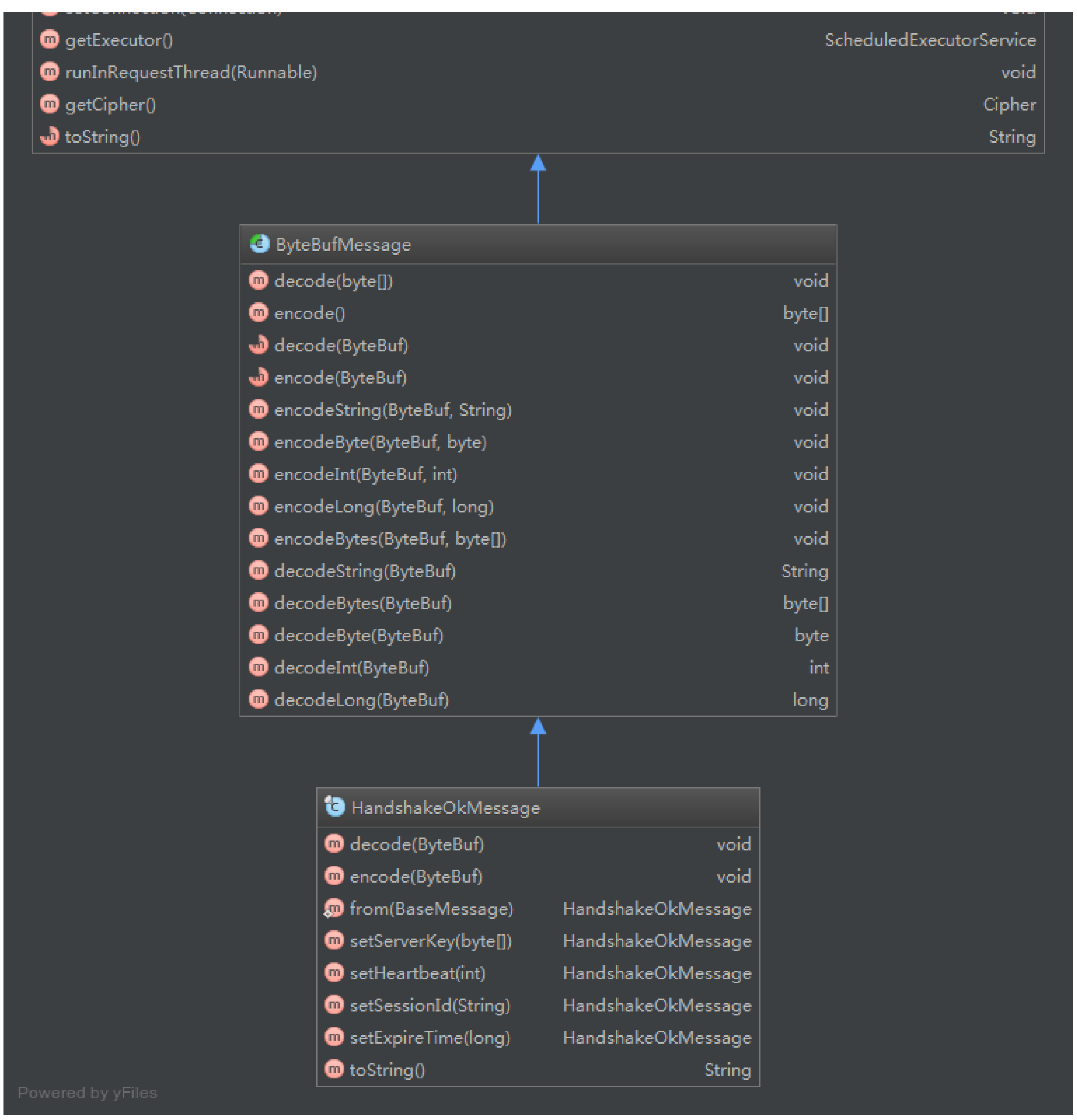


Message

getConnection()	Connection
decodeBody()	void
encodeBody()	void
send(ChannelFutureListener)	void
sendRaw(ChannelFutureListener)	void
getPacket()	Packet

BaseMessage

decodeBody()	void
encodeBody()	void
decodeBinaryBody0()	void
encodeBinaryBody0()	void
decodeJsonBody0()	void
encodeJsonBody0()	void
encodeJsonStringBody0()	void
encodeJsonStringBody()	String
encodeBodyRaw()	void
decode(byte[])	void
encode()	byte[]
decodeJsonBody(Map<String, Object>)	void
encodeJsonBody()	Map<String, Object>
getPacket()	Packet
getConnection()	Connection
send(ChannelFutureListener)	void
sendRaw(ChannelFutureListener)	void
send()	void
sendRaw()	void
close()	void
genSessionId()	int
getSessionId()	int
setRecipient(InetSocketAddress)	BaseMessage
setPacket(Packet)	void
setConnection(Connection)	void


getExecutor() ScheduledExecutorService
runInRequestThread(Runnable) void
getCipher() Cipher
toString() String
ByteBufMessage
decode(byte[]) void
encode() byte[]
decode(ByteBuf) void
encode(ByteBuf) void
encodeString(ByteBuf, String) void
encodeByte(ByteBuf, byte) void
encodeInt(ByteBuf, int) void
encodeLong(ByteBuf, long) void
encodeBytes(ByteBuf, byte[]) void
decodeString(ByteBuf) String
decodeBytes(ByteBuf) byte[]
decodeByte(ByteBuf) byte
decodeInt(ByteBuf) int
decodeLong(ByteBuf) long
HandshakeOkMessage
decode(ByteBuf) void
encode(ByteBuf) void
from(BaseMessage) HandshakeOkMessage
setServerKey(byte[]) HandshakeOkMessage
setHeartbeat(int) HandshakeOkMessage
setSessionId(String) HandshakeOkMessage
setExpireTime(long) HandshakeOkMessage
toString() String
Powered by yFiles

```
public final class HandshakeOkMessage extends ByteBufMessage {
 public byte[] serverKey;
 public int heartbeat;
 public String sessionId;
 public long expireTime;

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

  @Override
  public void decode(ByteBuf body) {
  serverKey = decodeBytes(body);
  heartbeat = decodeInt(body);
  sessionId = decodeString(body);
  expireTime = decodeLong(body);
```

```java
}

@Override
public void encode(ByteBuf body) {
encodeBytes(body, serverKey);
encodeInt(body, heartbeat);
encodeString(body, sessionId);
encodeLong(body, expireTime);
}

public static HandshakeOkMessage from(BaseMessage src) {
return new HandshakeOkMessage(src.packet.response(HANDSHAKE), src.conne
ction);
}

public HandshakeOkMessage setServerKey(byte[] serverKey) {
this.serverKey = serverKey;
return this;
}

public HandshakeOkMessage setHeartbeat(int heartbeat) {
this.heartbeat = heartbeat;
return this;
}

public HandshakeOkMessage setSessionId(String sessionId) {
this.sessionId = sessionId;
return this;
}

public HandshakeOkMessage setExpireTime(long expireTime) {
this.expireTime = expireTime;
return this;
}

@Override
public String toString() {
return "HandshakeOkMessage{" +
"expireTime=" + expireTime +
", serverKey=" + Arrays.toString(serverKey) +
", heartbeat=" + heartbeat +
 ", sessionId='" + sessionId + '\'' +
 ", packet=" + packet +
 '}';
 }
}
```