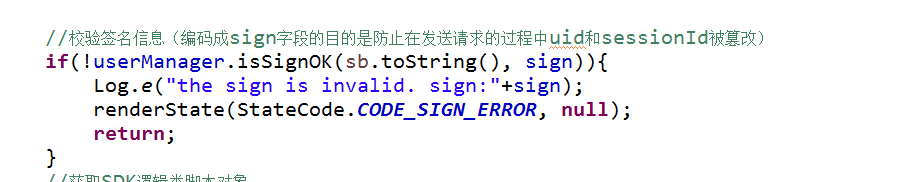
**U8Server之登录细节**

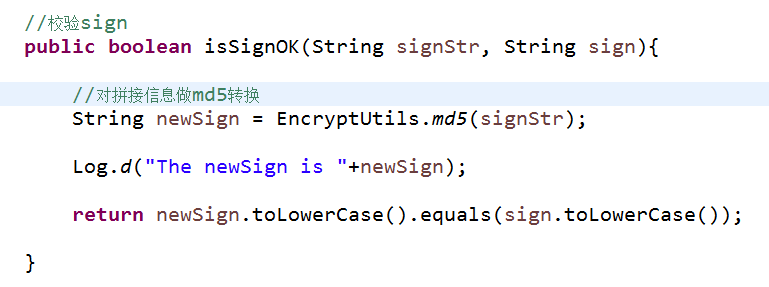
**1.从U8Server方面来讲，首先是游戏客户端拿着当前登录用户的信息和在U8Server哪儿注册的渠道（channelId）以及游戏ID（appid）等信息通过http访问U8Server来进行登录认证，登录方法由UserAction下的getLoginToken()方法开始**

**2.U8Server就拿着这些信息到他的数据库（或缓存）中认证这些信息，若信息有误则响应个状态码（游戏为空）**

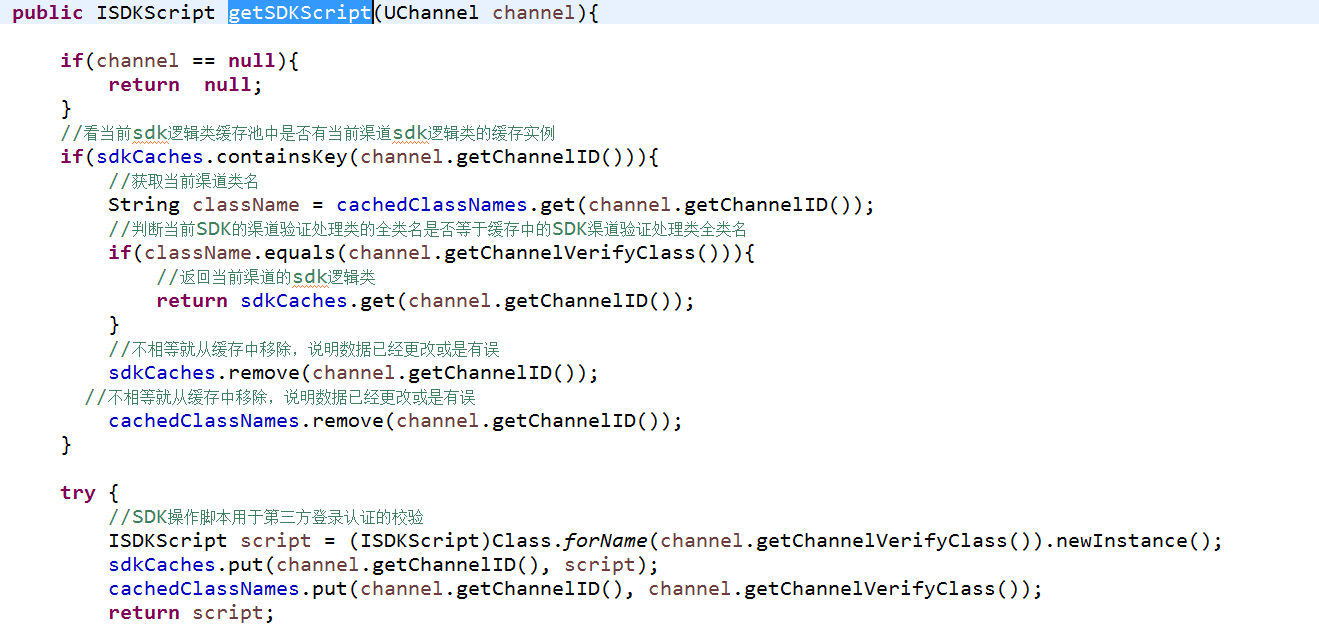


**3.将接收到的信息md5加密后签名信息进行比较（编码成sign字段的目的是防止在发送请求的过程中uid和sessionId被篡改）**

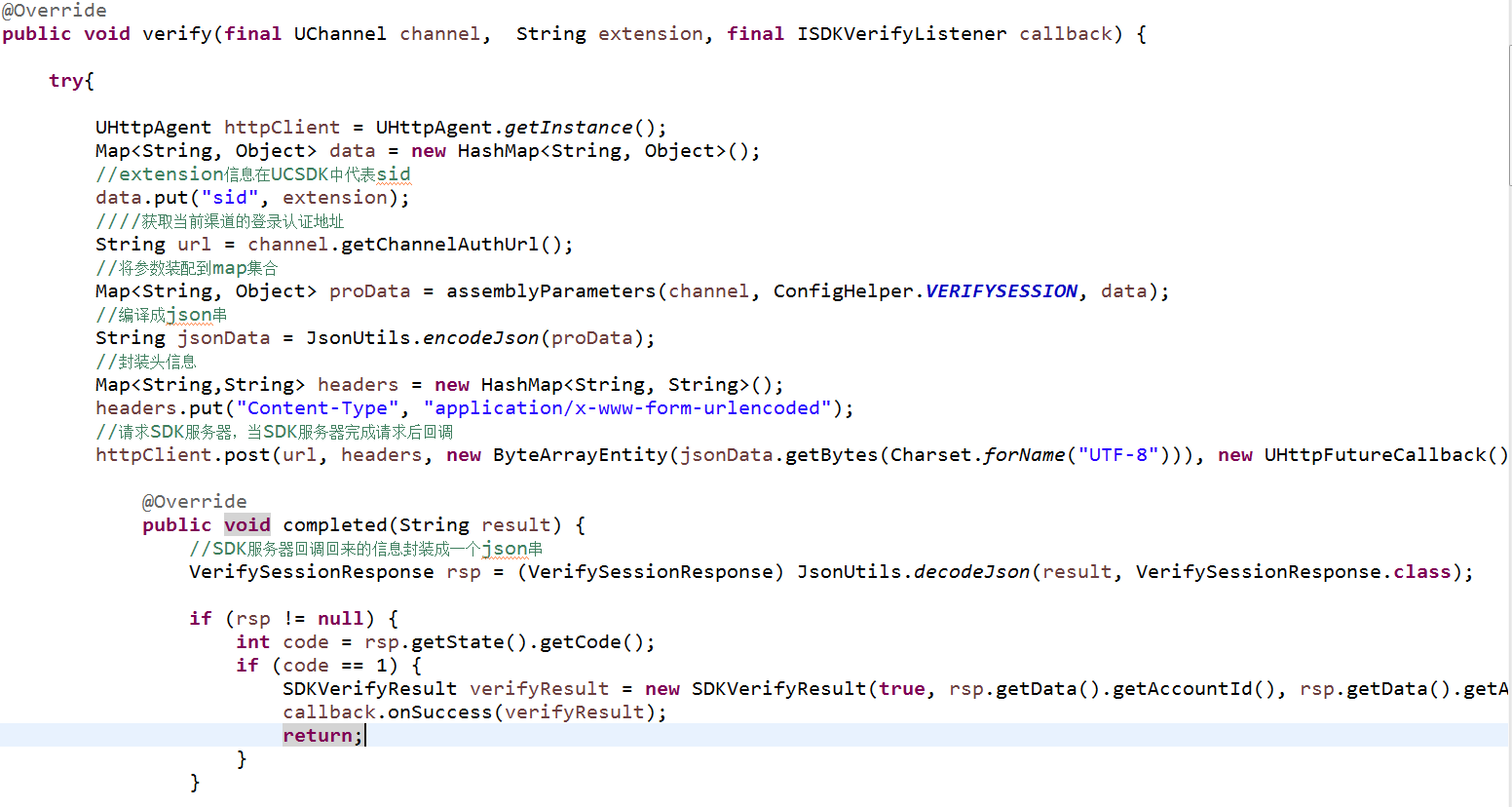




**4.获取SDK逻辑类脚本对象（每个渠道SDK都有一个实现了ISDKScript接口的SDK逻辑处理类，登录认证和获取订单号接口中，通过反射的方式来实例化对应渠道的类，可能会导致一定的性能瓶颈。所以，这里我们增加一个缓存，第一次实例化之后，将对应渠道的处理类，缓存起来。后面使用的时候，直接从缓存中取）**



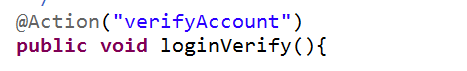
**5.调用脚本类对象的verify方法来完成封装信息发送到SDK服务器的操作（以UC为例）。**



**6.当SDK服务器发送成功消息返回U8时，U8会调用ISDKVerifyListener中的onsuccess方法来处理SDK服务器返回的信息创建token并将其发送给游戏客户端**



**7.上面协议返回客户端之后，开始连接登录游戏服。游戏服可以调用该协议进行再次登录认证。但是，该步骤是可选的。游戏服务器也可以自己验证token以及token的时效性，这样就不用来U8Server进行再次登录认证了服务器自己验证token，根据U8Server分配给每个游戏参数中的AppSecret，按照生成token的 规则，进行验证。同时，需要验证timestamp的时效性，如需要再来U8进行二次登录认证则访问UserAction中的loginVerify方法**



**U8Server支付细节**

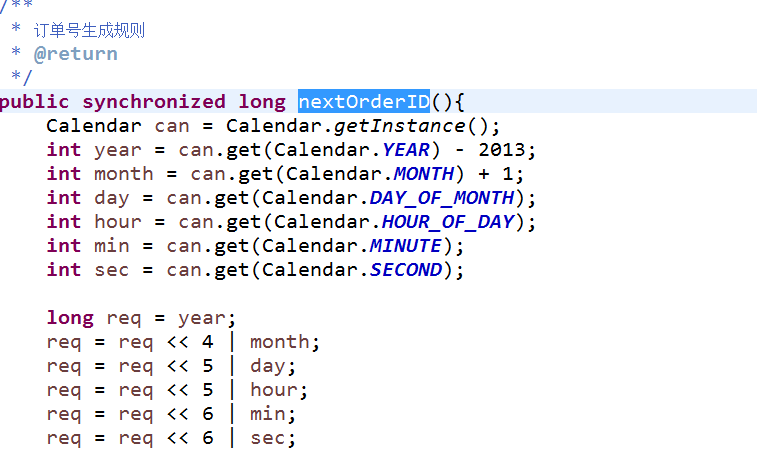
**1.首先玩家操作游戏客户端进行充值操作，客户端请求游戏服务器，游戏服务器收到信息即请求U8Server，在pay包下的PayAction下的getOrderID()方法。**

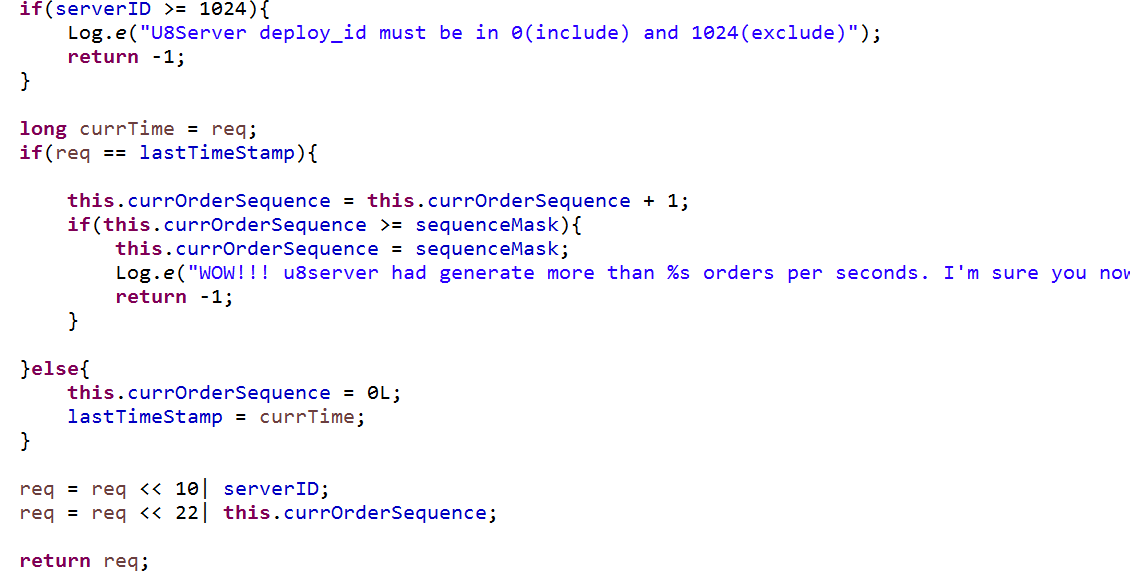
**2.U8验证请求信息，首先会根据请求信息中userID来获取玩家对象，判断玩家是否存在，不存在则不处理；然后判断金额是否大于0，不大于则不合理不处理；然后校对签名信息，签名信息核对不成功就不处理；验证当前渠道是否开通支付，没开通就不处理。**



**3.生成订单号以及订单对象**

**通过IDGenerator，这是U8中一个简单且唯一的ID生成器**





**首先取到当前时间戳，就是当前年，月，日，小时，分钟，秒。 我们只精确到秒。对于月，天，小时，分钟，秒几个值，他的表示需要的最大的位数是固定的，比如，月，一年最大12个月，所以，我们用4位即可表示（1<<4=16），同理，对于天，一个月最大31天，用5位即可表示；对于小时，一天最多24小时，用5位即可表示；对于分和秒，最大都是60，我们用6位即可表示。**

**所以月日时分秒，几个总占位为4+5+5+6+6=26位（总时间戳预留是32位，所以表示年份的位数为6位，所以上面说最多可以生成64年的订单号）**

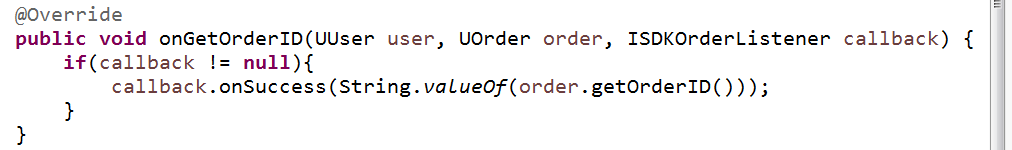
**时间戳之后， 再加上10位U8Server唯一实例ID，这个我们放在jdbc.properties中u8server.deploy\_id进行配置，每部署一个实例，这个ID不能重复，从0开始，最大1023。**

**最后再加上22位序号。这个序号，每秒会进行重置。也就是说同一秒中， 单台U8Server实例最多可以生成4194304(1<<22)个订单，如果真的超出了，那么真的要恭喜你了。。。**

**这样最终就生成一个long类型的全局唯一的订单号了。**

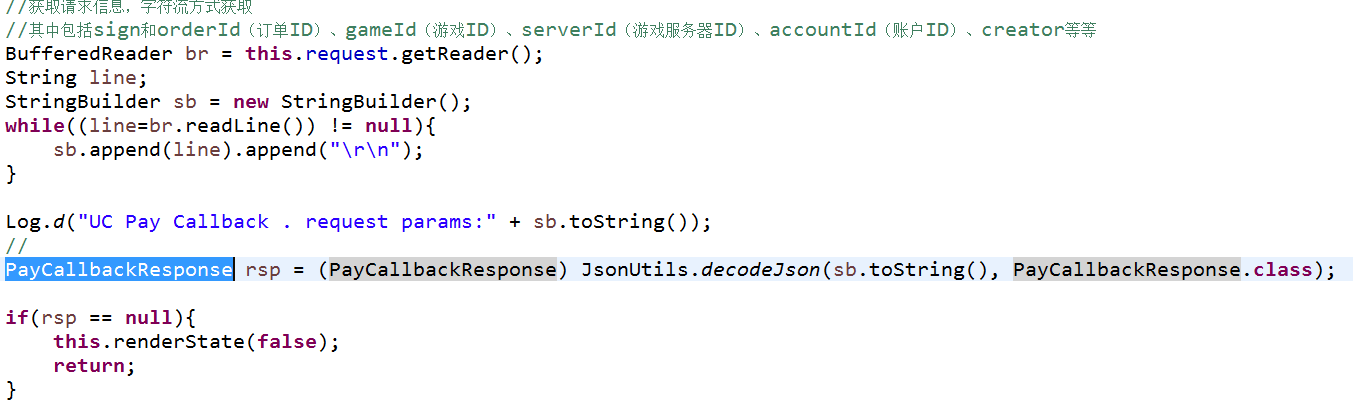
**4.生成当前渠道对应渠道SDK逻辑处理类来通知与封装信息**



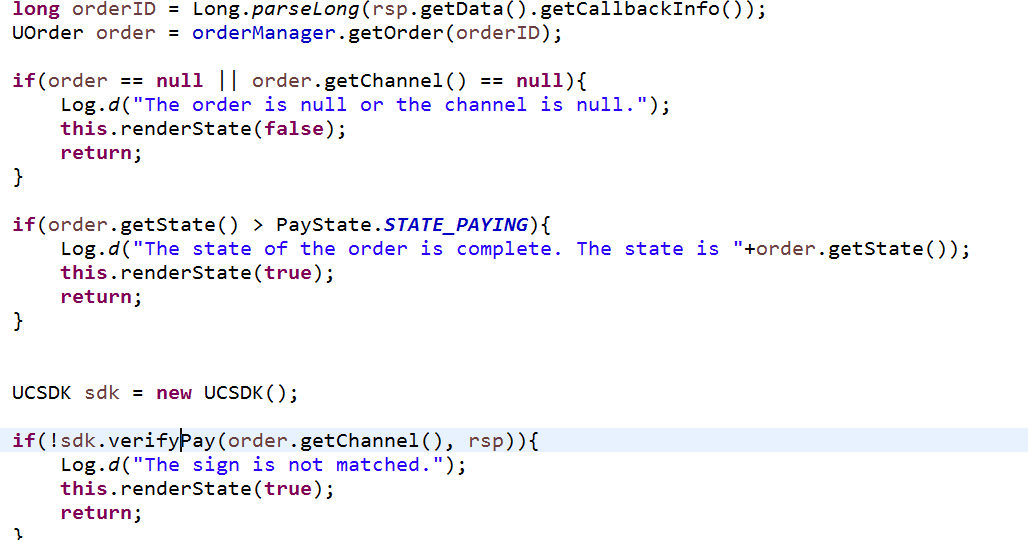


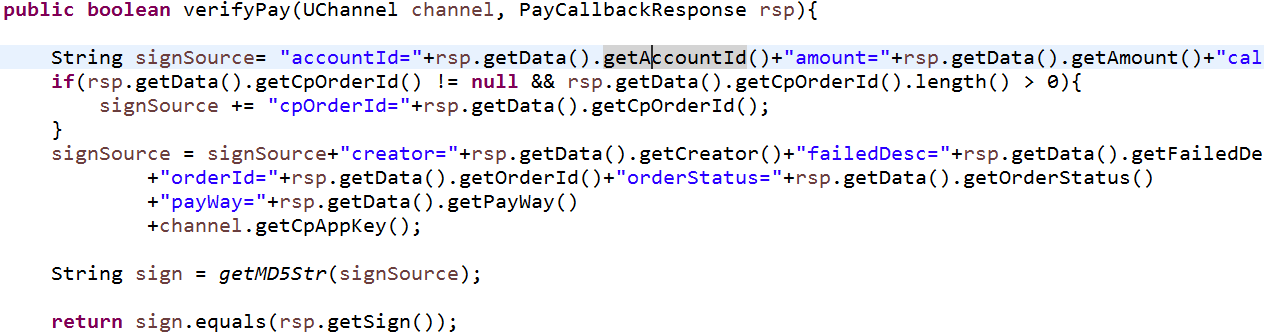
**5.订单号返回游戏服务器之后，等待渠道SDK来确认订单的完成程度，U8为每个渠道SDK都单独提供了一个支付回调处理接口，都在web/pay/sdk包下,以UC渠道为例，请求地址为http://localhost:8080/pay/uc/payCallback/渠道号**

**6.当UC渠道回调U8的时候，U8首先会读取字符流来获取UC渠道SDK的请求信息，用一个StringBuilder对象存储，然后转化成支付回调的响应内容类。**

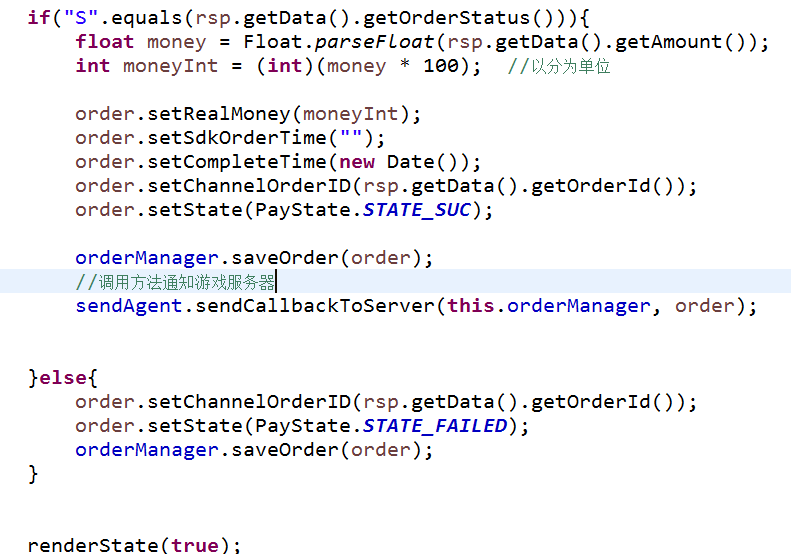


7.从支付回调响应内容类中获取订单ID，进而获取订单对象，拿着从订单中获取渠道信息对象，拿着渠道对象和支付回调响应内容对象的信息进行md5加密来对比sign（签名）





**8.获取订单完成状况，如果成功则将订单信息封装完善给游戏服务器发送过去，否则将U8中订单信息状态改为失败保存起来，处理完了返回给游戏服务器一个处理成功信号**



**9.通知游戏服务器具体流程如下，**

public boolean sendCallbackToServer(UOrderManager orderManager, UOrder order){

//获取游戏对象

UGame game = order.getGame();

if(game == null){

return false;

}

UUserManager userManager = (UUserManager)UApplicationContext.getBean("userManager");

//获取玩家对象

UUser user = userManager.getUser(order.getUserID());

if(user != null && (user.getFirstCharge() == null || user.getFirstCharge() == 0)){

//记录玩家首冲信息

user.setFirstCharge(1);

user.setFirstChargeTime(new Date());

userManager.saveUser(user);

}

//游戏下单的时候，可以携带notifyUrl过来，作为渠道支付回调时，通知到游戏服务器的地址，没有设置的话，默认走后台游戏管理中配置的固定通知回调地址

String callbackUrl = order.getNotifyUrl();

if(StringUtils.isEmpty(callbackUrl)){

callbackUrl = game.getPayCallback();

}

if(StringUtils.isEmpty(callbackUrl)){

Log.d("the order paycallback url is not configed. no in order. no in game.");

return false;

}

try{

//封装返回信息，包括订单信息，签名加密方式，状态信息，头信息等

//将订单对象封装成json

JSONObject data = new JSONObject();

data.put("productID", order.getProductID());

data.put("orderID", String.valueOf(order.getOrderID()));

data.put("userID", order.getUserID());

data.put("channelID", order.getChannelID());

data.put("gameID", order.getAppID());

data.put("serverID", order.getServerID());

data.put("money", order.getMoney());

data.put("currency", order.getCurrency());

data.put("extension", order.getExtension());

//如果需要将签名方式改为MD5，把下面两行SIGN\_RSA改为SIGN\_MD5

String sign = generateSign(order, SIGN\_MD5);

data.put("signType", SIGN\_MD5);

data.put("sign", sign);

JSONObject response = new JSONObject();

response.put("state", StateCode.CODE\_SUCCESS);

response.put("data", data);

Map<String, String> headers = new HashMap<String, String>();

headers.put("Content-Type", "text/html");

//返回订单是完成信息（玩家元宝，装备是否已经发放）

String serverRes = UHttpAgent.getInstance().post(callbackUrl, headers, new ByteArrayEntity(response.toString().getBytes(Charset.forName("UTF-8"))));

if(serverRes.equals("SUCCESS")){

Log.d("SendCallbackToServer success : orderId : " + order.getOrderID() + " callbackUrl : " + callbackUrl);

order.setState(PayState.STATE\_COMPLETE);

orderManager.saveOrder(order);

return true;

}else{

Log.d("SendCallbackToServer fail : orderId : " + order.getOrderID() + " callbackUrl : " + callbackUrl + " result : " + serverRes);

}

}catch (Exception e){

Log.d("SendCallbackToServer error : orderId : " + order.getOrderID() + " callbackUrl : " + callbackUrl);

Log.e(e.getMessage(), e);

e.printStackTrace();

}

//失败了，加入重发队列，尝试6次

OrderTaskManager.getInstance().addOrder(order);

return false;

}