###e####

Version: ##0.1

联通彩e接口开发

前段时间开发联通彩e接口，期间遇到很多问题，在朋友的帮助和自己的摸索中总算完成了接口的开发。 在sp联盟论坛上也见到许多同行各种各样的问题，因此将开发细节整理成文，希望能给与我当初一样 困扰的人以帮助。 第一次接触彩e,感觉有点无从下手，接口指南几百页之多，我在开发的时候不知道联通提供测试的接入 平台以供调试，而是对着接口规范编写代码，然后模拟接口规则生成数据，这些都是在单元测试中完成 的，到接入uni-wise测试环境时，问题多多。

目录

概述

本文以java语言为例，讲述彩e接口开发的点滴。参考的接口指南为《中国联通增值业务综合管理及接入平台SP接口规范v1.2》， 文中代码均经过测试，且与uni-wise平台能正常运行。彩e与sp接口包括:sso接口、预定接口、取消接口、彩e push接口，取 消push接口，查询push接口、wap push接口。文中除了wap push接口，其余的将会一一介绍。 彩e接口的开发其实就是sp与联通uni-wise平台之间的通信，uni-wise平台是以web方式工作，因此与sp的交互大部分通 过http+xml协议传输。 笔者在开发的过程中也曾用C#写过彩e的部分接口代码，如有此需求，我也将整理成文。

第 1 章 SSO接口

SSO 是 Single sign on的缩写，即单点登录，彩e接口中实现的功能是，用户在uni-wise平台或sp平台只需登录一次，即可 访问相关资源。通过cookies机制实现。

1.1. 传输安全

出于安全考虑，网络的传输中经常对传输数据做加密和编码处理，彩e接口开发中的一个关键点也是对加密解密的代码编写。 其中涉及以下几种：

1、md5加密，该加密算法是单向加密，即加密的数据不能再通过解密还原。相关类包含在java.security.MessageDigest包中。

2、3-DES加密，该加密算法是可逆的，解密方可以通过与加密方约定的密钥匙进行解密。相关类包含在javax.crypto.*包中。

3、base64编码，是用于传输8bit字节代码最常用的编码方式。相关类在sun.misc.BASE64Decoder 和sun.misc.BASE64Encoder 中。

4、URLEncoder编码，是一种字符编码，保证被传送的参数由遵循规范的文本组成。相关类在java.net.URLEncoder包中。

1.2. 生成请求票根

当用户从SP平台向uni-wise发起登录请求时，SP平台需要生成一个合法的票根，以http协议传输给uni-wise平台。 生成请求票根的规则是：SPTicketRequestValue = URLEncoding{UNICODE(SPCode + “$”)+ Base64 [Encrypt （UNICODE(Seed + “$”)+ Digest)]}

1、生成Seed: returnUrl + "$" + timeStamp;returnUrl为登录成功后接收uni-wise的响应票根链接。timeStamp为生成的 时间戳。

例 1.1. TimeStamp实现代码

```
public String getTimeStamp()
{
  Calendar cal=Calendar.getInstance();
  SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMddHHmmss.SSS");
  String timeStamp=formatter.format(cal.getTime());
  return timeStamp;
}
```

2、生成Digest :Base64{Hash[UNICODE(SPCode +"$"+ Seed + "$" + SPKey)]},其中Hash算法采用md5

例 1.2. Digest的实现代码

```
public String getDigest(String strSrc)
{
        //String strSrc = spCode + "$" + getSeed() + "$" + spKey;
        BASE64Encoder base64en = new BASE64Encoder();
        String digest="";
        try
        {
          byte[] srcMD5 = md5Encrypt(strSrc);
          digest = base64en.encode(srcMD5);          (1)
        }
        catch(Exception e){
          e.printStackTrace();
        }
        return digest;
  }

private byte[] md5Encrypt(String strSrc)
{
        byte[] returnByte = null;
        try
        {
                MessageDigest md5 = MessageDigest.getInstance("MD5"); (2)
                returnByte = md5.digest(strSrc.getBytes("GBK"));
        }
        catch(Exception e)
        {
                e.printStackTrace();
        }
        return returnByte;
}
```

(1) 用base64编码
(2) 指定加密方式为md5

3、生成密钥匙，用联通提供的key，进行md5加密。

例 1.3. 得到3-DES的密钥匙

```
private byte[] getEnKey(String spKey)
{
        byte[] desKey=null;
        try
        {
          byte[] desKey1 = md5Encrypt(spKey);
          desKey = new byte[24];
          int i = 0;
          while (i < desKey1.length && i < 24) {
                desKey[i] = desKey1[i];
                i++;
          }
          if (i < 24) {          (1)
                desKey[i] = 0;
                i++;
          }
```

```
        }
        catch(Exception e){
          e.printStackTrace();
        }
        return desKey;
}
```

(1) 根据接口规范，密钥匙为24个字节，md5加密出来的是16个字节，因此后面补8个字节的0

4、生成SPTicketRequestValue，URLEncoding{UNICODE(SPCode +“$”)+ Base64 [Encrypt (UNICODE(Seed + “$”)+ Digest)]}， Encrypt算法采用3-DES加密，用md5加密的key作为密钥匙。

例 1.4. 3-DES加密的实现代码

```
public String getSPTicketRequestValue()
{
        String SPTicketRequestValue="";
        try{
          byte[] src = (getSeed() + "$" + getDigest() ).getBytes("UTF-16LE");
          byte[] enKey = getEnKey(spKey);
          byte[] encryptedData = Encrypt(src,enKey);                                    (1)
          String base64Encrypt = filter(base64en.encode(encryptedData));    (2)
          String requestValue=spCode + "$" + base64Encrypt;
          SPTicketRequestValue=URLEncoder.encode(requestValue);
        }
        catch(Exception e){
          e.printStackTrace();
        }
        return SPTicketRequestValue;
}

 public byte[] Encrypt(byte[] src,byte[] enKey)
{
        byte[] encryptedData = null;
        try
        {
                DESedeKeySpec dks = new DESedeKeySpec(enKey);                           (3)
                SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DESede"); (4)
                SecretKey key = keyFactory.generateSecret(dks);                         (5)
                Cipher cipher = Cipher.getInstance("DESede");                           (6)
                cipher.init(Cipher.ENCRYPT_MODE, key);                                  (7)
                encryptedData = cipher.doFinal(src);                                    (8)
        }
        catch(Exception e)
        {
                e.printStackTrace();
        }
        return encryptedData;
}

 private String filter(String str)
{
        String output = null;
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < str.length(); i++)
```

```
        {
                int asc = str.charAt(i);
                if(asc != 10 && asc != 13)
                        sb.append(str.subSequence(i, i + 1));
        }
        output = new String(sb);
        return output;
}
```

(1) 调用3-DES的加密函数。
(2) base64编码3-DES的数据时，得到的字符串有换行符号，一定要去掉，否则uni-wise平台解析票根不会成功， 提示“sp验证失败”。在开发的过程中，因为这个问题让我束手无策，一个朋友告诉我可以问联通要一段加密后 的文字，然后去和自己生成的字符串比较，这是个不错的调试方法。我最后比较发现我生成的字符串唯一不同的 是多了换行。我用c#语言也写了票根请求程序，没有发现这个问题。
(3) 从原始密匙数据创建DESedeKeySpec对象。
(4) 创建一个密匙工厂，然后用它把DESKeySpec转换成一个SecretKey对象。
(5) 根据密匙工厂，得到一个密钥匙实例。
(6) 创建Cipher对象。
(7) 初始化Cipher对象（带入密钥匙）。
(8) 执行加密操作。

为方便调试给出一段加密后的字符串（用自己的代码加密同样的字符串后得到的结果和给出的结果进行比较）

```
content = key = 1234;
result = base64 (3des(contentbyte,keybyte));
result : "25Pxmw/+/qKg2arQpLdvqQ=="
```

1.3. 解析响应票根

uni-wise平台收到请求票根后会进行解析票根，然后用户登录后uni-wise又会将登录信息放在响应票根中，传回给请求票根者。（传回的地址是请求票根中的”returnUrl “）。传输的协议也是http+xml.

1、UrlEncoding的解码（笔者在测试中发现在uni-wise测试平台首先需要对所得的票根响应串进行urlEncoding解码,这也是接口 遵照接口手册进行的，但是在uni-wise正式平台下得到的响应票根已经是对urlEncoding解码的）。

```
import java.net.URLDecoder;
URLDecoder.decode(strEncoding);
```

2、分割响应票根，得到加密的字符串。分割很简单，以"$"为分割符，得到"$"后面的字符串即可。

3、对加密字符串进行base64解码。

```
import sun.misc.BASE64Decoder;
```

```
BASE64Decoder base64Decode = new BASE64Decoder();
base64Decode.decodeBuffer(strEnBase64);
```

4、对base64解码的值进行3-DES解密（密钥匙等同于加密的密钥匙）。

```
public String deCrypt(byte[] debase64)
{
        String strDe = null;
        Cipher cipher = null;
        try
        {
        cipher=Cipher.getInstance("DESede");
        byte[] key = getEnKey(spKey);                                    (1)
        DESedeKeySpec dks = new DESedeKeySpec(key);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DESede");
        SecretKey sKey = keyFactory.generateSecret(dks);
        cipher.init(Cipher.DECRYPT_MODE, sKey);                  (2)
        byte ciphertext[] = cipher.doFinal(debase64);
        strDe = new String(ciphertext,"UTF-16LE");
        }
        catch(Exception ex)
        {
         strDe = "";
         ex.printStackTrace();
        }
        return strDe;
}
```

(1) 得到密钥匙，具体实现请参考请求票根部分

(2) 加密、解密其实就是指定这个类型 Cipher.ENCRYPT_MODE 或 Cipher.DECRYPT_MODE

5、通过"$"分割解密的字符串，然后取得对应的信息，如MDN,UserId等。如果SP需要记录用户信息，应该在解析票根后 持久存储。然后根据实际需要将用户信息加入cookies.以实现SSO。

第 2 章 预定接口

2.1. 发起预定请求

用户可以从SP平台和uni-wise平台发起预定请求，当从SP发起预定请求时，SP平台需要 构建预定请求的数据,以htpp协议传输给uni-wise.请求数据的构建有两种方式一种是 service方式，一种是product方式.根据实际情况构建请求参数。 http://接入平台的URL?SPCode=A&ServiceCode=B&ReturnURL=C （service方式） http://接入平台的URL?SPCode=A&ProductCode=B&ReturnURL=C （product方式）

2.2. 解析uni-wise预定请求

所有的预定操作都是链接到uni-wise预定平台，然后用户在此发生预定关系，uni-wise构建 预定信息通知SP平台，根据SP的响应结果，决定预定动作成功与否。uni-wise要将预定信息 通知SP,因此需要预先知道SP接收预定的链接，这是在申请彩e业务时完成。

例 2.1. 解析预定请求

```
InputStream subscriptionRequestStream = request.getInputStream();  (1)
PraseXml(subscriptionRequestStream);                                (2)
```

(1) uni-wise将预定信息以输出流传输到SP平台，SP用HttpServletRequest得到输入流。
(2) ParseXml为解析xml的函数。与解析普通的xml一样，因此在此不再描述其实现方法。

2.3. 验证请求

得到预定请求信息后，应该对其数据进行验证，根据验证的结果返回相应的信息给uni-wise平台。

```
if (mdn == null || mdn.trim().equals(""))
{
        errorCode = "16842754";
        errorInfo = "cannot find MDN";
}

if ( spCode == null || spCode.trim().equals(""))
{
        errorCode = "16973826";
        errorInfo = "cannot find SPCode";
}

if (productCode == null || productCode.trim().equals(""))
{
        errorCode = "17104898";
        errorInfo = "cannot find ProductCode";
}
if (transactionID == null || transactionID.trim().equals(""))
```

```
{
        errorCode = "17170434";
        errorInfo = "cannot find TransactionID";
}
```

2.4. 响应预定请求

SP接收到uni-wise预定请求信息后进行验证核对，然后响应请求，传输协议为http+xml。 uni-wise得到响应后执行预定动作，返回给用户预定相关信息。SP处理请求结果有成功 和失败两种情况。

例 2.2. 请求成功的响应格式

```
StringBuffer sb = new StringBuffer();
sb.append("<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n");
sb.append("<U-MAX>\n");
sb.append("<PreSubscriptionNotify>\n");
sb.append("<MDN>" + getMdn() + "</MDN>\n");
sb.append("<SPCode>" + getSpCode() + "</SPCode>\n");
sb.append("<ProductCode>" + getProductCode() + "</ProductCode>\n");
sb.append("<TransactionID>" + getTransactionID() + "</TransactionID>\n");
sb.append("</PreSubscriptionNotify>\n");
sb.append("</U-MAX>");
String strResponse = new String(sb);
```

例 2.3. 请求失败的响应格式

```
StringBuffer sb = new StringBuffer();
sb.append("<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n");
sb.append("<U-MAX>\n");
sb.append("<ValidError>\n");
sb.append("<ValidErrorCode>");
sb.append(getErrorCode());
sb.append("</ValidErrorCode>");
sb.append("<ValidErrorInfo>");
sb.append(getErrorInfo());
sb.append("</ValidErrorInfo>");
sb.append("</ValidError>\n");
sb.append("</U-MAX>");
String strResponse = new String(sb);
```

第 3 章 预定取消接口

3.1. 发起取消请求

用户可以从SP平台和uni-wise平台发起取消请求，当从SP发起取消请求时，SP平台需要 构建取消请求的数据,以htpp协议传输给uni-wise.请求数据的构建有两种方式一种是 service方式，一种是product方式.规则同预定请求一致。

3.2. 解析uni-wise取消请求

所有的取消操作也是定向到uni-wise平台，然后用户在此执行取消动作，uni-wise接收 到用户取消动作后将取消信息通知SP平台，根据SP的响应结果，决定取消动作成功与否。 SP的取消接收链接，也是在申请彩e业务时完成。实现代码与解析预定请求代码一致，在此 不再重复。

3.3. 验证请求

请参考预定请求章节的相应内容。

3.4. 响应请求

例 3.1. 请求处理成功的响应

```
StringBuffer sb = new StringBuffer();
sb.append("<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n");
sb.append("<U-MAX>\n");
sb.append("<SubscriptionCancel>\n");
sb.append("<MDN>" + getMdn() + "</MDN>\n");
sb.append("<SPCode>" + getSpCode() + "</SPCode>\n");
sb.append("<ProductCode>" + getProductCode() + "</ProductCode>\n");
sb.append("<TransactionID>" + getTransactionID() + "</TransactionID>\n");
sb.append("</SubscriptionCancel>\n");
sb.append("</U-MAX>");
String strResponse = new String(sb);
```

例 3.2. 请求处理失败的响应

```
StringBuffer sb = new StringBuffer();
sb.append("<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n");
sb.append("<U-MAX>\n");
sb.append("<ValidError>\n");
sb.append("<ValidErrorCode>");
sb.append(getErrorCode());
```

```
sb.append("</ValidErrorCode>");
sb.append("<ValidErrorInfo>");
sb.append(getErrorInfo());
sb.append("</ValidErrorInfo>");
sb.append("</ValidError>\n");
sb.append("</U-MAX>");
String strResponse = new String(sb);
```

第 4 章 web push接口

发送彩e，先提交到uni-wise平台的push接口，然后再由uni-wise push到IMAP平台，如图：

图 4.1.

4.1. 提交push

SP 提交信息到 uni-wise Push接口的数据分包头和包体，其中包体以MIME格式传输。

1、构建包头: SP的企业代码（不加密）+ SP的密钥（加密）+付费代码（加密）+条件类别（加密）+条件代码（加密） +发送方式（加密）+[发送开始时间（加密）+截止时间（加密）]+ 计费手机号码（加密）+ MIME包体 中的边界字段Boundary（加密）

```
import java.net.HttpURLConnection;
import java.net.URL;

private Url url = null;
HttpURLConnection conn = null;

//先建立URL长连接
public void connectUrl(String strUrl)
{
        try
        {
                url = new URL(strUrl);
                conn = (HttpURLConnection)url.openConnection();
                conn.setRequestMethod("post");
        }
```

```
        catch(Exception ex)
        {

        }
}

//设置包头
public void setHeader()
{
        conn.setRequestProperty("SPCode", spCode);
        conn.setHeader("EncryptSPKey",enKey);
        conn.setHeader("FeeCode",enFeeCode );
        conn.setHeader("ConditionType",enConditionType);
        conn.setHeader("ConditionCode", enConditionCode);
        conn.setHeader("SendType", enSendType);
        conn.setHeader("StartTime", enStartTime);
        conn.setHeader("EndTime",enEndTime);
        conn.setHeader("ThirdPartyPayPhone",enThirdPartyPayPhone);
        conn.setHeader("Boundary", enBoundary);
}
```

包头中除了spcode不用加密，其余的都遵行base64（3des(contentbyteplus,keybyte)）加密方式。加密的实现代码和前面章节描述的一致。

2、构建包体

MIME (Multipurpose Internet Mail Extensions,多目的Internet邮件扩展)是创建用于电子邮件交换， 网络文档，及企业网和Internet上的其他应用程序中的文件格式的规范.

例 4.1. 构建MIME格式的包体

```
private MimeMessage mime;
//构造MIME格式的包体
private void setMimeMessage()
{
try
{
mime.setFrom(new InternetAddress(strFrom));          (1)
mime.addRecipient(javax.mail.Message.RecipientType.TO, new InternetAddress(strTo)); (2)
mime.setSubject(strSubject,"UTF-8"); (3)
mime.setSentDate(new Date());
mime.setContent(getMimeMultipart());
}
catch(Exception ex)
{
}
}
private MimeMultipart getMimeMultipart()
{
MimeMultipart mimeMultipart = new MimeMultipart();
MimeBodyPart mimeBodyPart = new MimeBodyPart();
try
{
mimeBodyPart.setText((String)mimeBodyText.get(i),"UTF-8"); (4)
mimeMultipart.addBodyPart(mimeBodyPart);
```

```
Vector filePathes = getFilePaths();
for(int i = 0; i < filePathes.size(); i++)
{
String filePath = (String)filePathes.get(i);
javax.activation.DataSource datasource = new FileDataSource(filePath);
MimeBodyPart mimeFile = new MimeBodyPart();
mimeFile.setDataHandler(new DataHandler(datasource));
mimeFile.setFileName((new File(filePath)).getName());
mimeMultipart.addBodyPart(mimeFile);
}
}
catch(Exception e)
{

}
return mimeMultipart;
}
```

(1) 设置发送地址，在手机上显示发送方为该值
(2) 设置要发送到的手机号
(3) 手机上显示的标题值，经过笔者测试如果不指定编码为UTF-8，手机上显示为乱码（测试手机京瓷），不知道其他手机是否有这种情况。
(4) 手机上显示的正文，经过笔者测试如果不指定编码为UTF-8，手机上显示为乱码（测试手机京瓷）。

3、提交内容到push接口

```
public void write(String body)
{
        java.io.OutputStream outStream = conn.getOutputStream();                    (1)
        DataOutputStream dataOutStream = new DataOutputStream(outStream);
        dataOutStream.writeBytes(body);
        dataOutStream.flush();
        dataOutStream.close();
}
```

(1) 这里的conn对象是引用第一步中实例的URLConnection对象，已经处于open状态。

4、读取push响应信息

提交信息与返回信息是实时的，因此应该在提交后即实现读取操作。

```
public String responsePush()
{
        StringBuffer sb = null;
        try
        {
                sb = new StringBuffer("");
                BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
                for(String line = null; (line = rd.readLine()) != null;)
                        sb.append(line);
                rd.close();
        }
```

```
    catch(Exception ex)
    {

    }
    return new String(sb);
}
```

responsePush（）得到的是一个标准的xml字符串。格式请参考接口指南。code节点的值为0代表push成功。