手机接收的PDU串的分析(包含7-bit和UCS2解码，超长短信解释)

网络中这方面的资源还挺多的，特别是发短信的源码。利用AT(attention)命令接口控制SIM卡的活动，虽然我们也许不会去写嵌入式系统，但仍然建议基于串口/USB口通过Modem/手机编写AT高级语言编程的朋友们先找本较新版本的《AT Commands Interface》手册读读。对AT命令接口有个认识之后，再去开发你的解决方案。另外，通过串口连接Modem，还需要对串口的基础知识有一些了解，比如基本的常识：端口名称(COM1,COM2...)、波特率(正确的波特率才能与设备正常通信，在终端看到正确的字符编码，否则会是乱码)、校验位(这里通常为0)、数据位(这里通常为8)、停止位(这里通常为1)、读写超时(比如100ms)、串口缓冲区、握手协议(Handshake，通常不设定，由设备自己控制)、RTS是否启用(这里应该启用)、DTR是否启用等等。

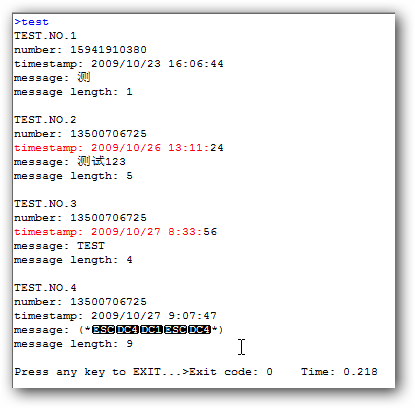
.NET Fromework 2.0之后，我们有两种方法解决与串口的通信问题，第一种方式，利用.NET框架中提供的SerialPort类，它可以很好的通过事件的方式监听串口数据，对缓冲区控制也不错，但是要注意，接收事件是在另一个线程进行的，你可能需要维护发送与接收串口数据的同步。第二种方式，使用网络上流传较多的JustinIO类，很简单的类，实现了读写串口的API方法(Read和Write)，比较简单易用，但是功能有限，也许需要二次开发。还要明白串口的收发数据的长度需要从缓冲区里读写，也没有读完了或者写完了的说法，只是有数据就提供，需要由自己编写代码来判断数据的长短，是否读取完整。

关于AT命令接口的问题，你需要了解，每条AT命令以CR结束(我也查到过CRLF结束的)，命令的返回信息则是被CRLF首尾括起的字符串。什么时候返回什么样的数据，成功(如<CRLF>OK<CRLF>)或者失败(如<CRLF>ERROR<CRLF>但不仅限于它)的提示都需要有一些了解。对得到的回应数据进行分析取舍，来电时或者来短信时，会自动返回的数据。推荐编程之前使用PComm Terminal软件(在国内的大型软件下载站内能够搜索下载到)，向Modem测试AT指令。

当我们收到短信时，可能会收到这样的数据：  
  
+CMTI: "SM",1  
  
+CMTI: "SM",2

然后通过AT+CMGR=1或者AT+CMGR＝2读取SIM卡中的短信内容，如果设置了PDU格式，那么会得到PDU数据。如果在之前设定过AT+CNMI=2,1,0,0,0，那么短信息会跳过SIM卡的存储，直接显示出收到的数据。将短信模式设置为PDU模式时(at+cmgf=0)，在PDU解码时需要注意编码方式，通常有汉字信息的采用UCS2编码(最多70字符)，纯英文信息采用7-bit编码(有7位编码算法，可以将ASCII的数字、大小写字母转换成该编码，但是标点符号就无能为力了，最多160字符)，图像和铃声采用8-bit编码(最多140字节)。在7位编码的时候建议用文本模式(at+cmgf=1)去读取短信，就不必进行7-bit解码，而且能够显示正确的标点。  
下面是设备收到PDU信息时得到的PDU字串的解码分析，仅供参考：

1. using System;
2. using System.Text;
3. using System.Globalization;
5. class Test
6. {
7. static void Main()
8. {
9. string[] pdus =
10. {
11. "0891683108401105F0240D91685149910183F0000890013261604423026D4B",
12. "0891683108401105F0040D91683105706027F50008900162311142230A6D4B8BD5003100320033",
13. "0891683108401105F0040D91683105706027F500009001728033652304D4E2940A",
14. "0891683108401105F0040D91683105706027F50000900172907074230928D58612D9505429",
15. };
16. for (int i = 0; i < pdus.Length; i++)
17. {
18. string pdu = pdus[i];
19. string number, message;
20. DateTime timestamp;
21. ParseReceivedSms(pdu, out number, out timestamp, out message);
22. Console.WriteLine("TEST.NO.{0}", i + 1);
23. Console.WriteLine("number: {0}\ntimestamp: {1}\nmessage: {2}", number, timestamp, message);
24. Console.WriteLine("message length: " + message.Length + "\n");
25. }
27. Console.Write("Press any key to EXIT...");
28. Console.ReadKey(true);
29. }
31. public static void ParseReceivedSms(string pduEncoded, out string number, out DateTime timestamp, out string message)
32. {
33. //分析电话号码
34. char[] numberChars = new char[12];
35. for (int i = 26, j = 0; i < 26 + 12; i += 2, j += 2)
36. {
37. numberChars[j] = pduEncoded[i + 1];
38. numberChars[j + 1] = pduEncoded[i];
39. }
40. //numberChars[11]='\0';
41. number = new string(numberChars, 0, 11);
43. //分析接收时间
44. string timestampString = string.Format("{0}{1}/{2}{3}/{4}{5} {6}{7}:{8}{9}:{10}{11}",
45. pduEncoded[43], pduEncoded[42],
46. pduEncoded[45], pduEncoded[44],
47. pduEncoded[47], pduEncoded[46],
48. pduEncoded[49], pduEncoded[48],
49. pduEncoded[51], pduEncoded[50],
50. pduEncoded[53], pduEncoded[52]
51. );
52. timestamp = Convert.ToDateTime(timestampString);
54. //分析短信
55. //int msgCountByByte=(pduEncoded.Length-58)/2;
56. int msgCountByByte = Byte.Parse(pduEncoded.Substring(56, 2), NumberStyles.HexNumber);
57. if (msgCountByByte <= 0)
58. {
59. message = string.Empty;
60. }
61. else
62. {
63. byte[] bytes = new byte[msgCountByByte];
64. int msgType = Byte.Parse(pduEncoded.Substring(40, 2), NumberStyles.HexNumber); //取编码方式
65. switch (msgType)
66. {
67. case 0: //bit7
68. message = Gsm7bitDecoding(pduEncoded.Substring(58));
69. break;
70. case 8: //UCS2
71. for (int i = 58, j = 0; j < msgCountByByte; i += 2, j++)
72. {
73. bytes[j] = Byte.Parse(pduEncoded.Substring(i, 2), NumberStyles.HexNumber);
74. }
75. message = Encoding.BigEndianUnicode.GetString(bytes);
76. break;
77. case 4: //bit8
78. default:
79. for (int i = 58, j = 0; j < msgCountByByte; i += 2, j++)
80. {
81. bytes[j] = Byte.Parse(pduEncoded.Substring(i, 2), NumberStyles.HexNumber);
82. }
83. message = Encoding.ASCII.GetString(bytes);
84. break;
85. }
86. }
87. }
89. public static string Gsm7bitEncoding(string text)
90. {
91. byte[] textBytes = Encoding.ASCII.GetBytes(text);
92. //int gsm7bitEncodingLength = textBytes.Length \* 7;
93. //gsm7bitEncodingLength = gsm7bitEncodingLength % 8 != 0 ? gsm7bitEncodingLength / 8 + 1 : gsm7bitEncodingLength / 8;
94. int textBytesLength = textBytes.Length;
95. int gsm7bitEncodingLength = (int)Math.Ceiling((textBytesLength \* 7) / 8.0);
96. byte[] gsm7bitEncodingBytes = new byte[gsm7bitEncodingLength];
97. int a, b, k;
98. for (a = 0, b = 0; b < textBytesLength; a++, b++)
99. {
100. k = b % 8;
101. if (k == 7)
102. {
103. a--;
104. if (b < (textBytesLength - 1))
105. {
106. gsm7bitEncodingBytes[a] |= (byte)((textBytes[b] & 0x7f) << 1);
107. }
108. }
109. else
110. {
111. if (b < (textBytesLength - 1))
112. {
113. gsm7bitEncodingBytes[a] = (byte)(((textBytes[b] & 0x7f) >> k) | ((textBytes[b + 1] & 0x7f) << (7 - k)));
114. }
115. else
116. {
117. gsm7bitEncodingBytes[a] = (byte)(((textBytes[b] & 0x7f) >> k));
118. }
119. }
120. }
121. return BytesToHexString(gsm7bitEncodingBytes);
122. }
124. public static string Gsm7bitDecoding(string textEncoded)
125. {
126. byte[] src = HexStringToBytes(textEncoded);
127. if (src.Length == 0)
128. {
129. return string.Empty;
130. }
131. int srcLength = src.Length;
132. int dstLength = srcLength \* 8 / 7;
133. byte[] dst = new byte[dstLength];
134. int a, b, k;
135. for (a = 0, b = 0; b < srcLength ; a++, b++)
136. {
137. k = a % 8;
138. if (a > 0)
139. {
140. dst[a] = (byte)(((src[b] << k) & 0x7f) | (src[b - 1] >> 8 - k));
141. }
142. else
143. {
144. dst[a] = (byte)(src[b] & 0x7f);
145. }
146. if (k == 7 && a > 0)
147. {
148. dst[++a] = (byte)(src[b] & 0x7f);
149. }
150. }
151. return Encoding.ASCII.GetString(dst);
152. }
154. static string BytesToHexString(byte[] data)
155. {
156. StringBuilder sb = new StringBuilder();
157. for (int i = 0; i < data.Length; i++)
158. {
159. sb.Append(data[i].ToString("X2"));
160. }
161. return sb.ToString();
162. }
164. static byte[] HexStringToBytes(string hexString)
165. {
166. int hexStringLength = hexString.Length;
167. if (hexStringLength < 2 || hexStringLength % 2 != 0)
168. {
169. return new byte[0];
170. }
171. byte[] data = new byte[hexStringLength / 2];
172. for (int i = 0, j = 0; i < hexStringLength; i += 2, j++)
173. {
174. data[j] = Byte.Parse(hexString.Substring(i, 2), NumberStyles.HexNumber);
175. }
176. return data;
177. }
178. }

直接用CSC编译，编码的分析看代码最后的注释部分，测试结果如下：  
TEST.NO.1number: 15941910380timestamp: 2009-10-23 16:06:44message: 测message length: 1TEST.NO.2number: 13500706725timestamp: 2009-10-26 13:11:24message: 测试123message length: 5TEST.NO.3number: 13500706725timestamp: 2009-10-27 8:33:56message: TESTmessage length: 4TEST.NO.4number: 13500706725timestamp: 2009-10-27 9:07:47message: (\* \*)message length: 9Press any key to EXIT...

为了便于对齐，接收PDU串分析注释的等宽字体如下(您可能需要Firefox浏览器查看)：+CMGL: 1,1,,220891683108401105F0240D91685149910183F0000890013261604423026D4B+CMGL: 3,1,,300891683108401105F0040D91683105706027F50008900162311142230A6D4B8BD5003100320033+CMGL: 6,1,,240891683108401105F0040D91683105706027F500009001728033652304D4E2940A+CMGL: 7,1,,280891683108401105F0040D91683105706027F50000900172907074230928D58612D9505429OK手机接收的PDU串分析0891683108401105F0240D91685149910183F0000890013261604423026D4B08 +8613800411500F 0D +8615941910380F 09102316064432 6D4B 91 24 91 00 02 0808 - 短信中心号码/地址长度(指后面有8位字节)91 - 国际格式号码(在前面加"+")+8613800411500F - 短信中心号码/地址：1380041150024 - SMS\_DELIVER的第一个8位: 参考http://www.dreamfabric.com/sms/submit\_fo.html0D - 发送方号码/地址长度(指后面的号码是13位，与前面的短信中心号码表示方法不同)91 - 国际格式号码(在前面加"+")+8615941910380F - 发送方号码/地址：1594191038000 - 协议标识TP-PID：普通GSM，点到点方式08 - 编码方式TP-DCS：三种：00表示7-bit编码(英文)、04表示8-bit编码(图片和铃声)、08表示UCS2编码(汉字)（注意：见17楼回复）09102316064423 - 时间戳TP-SCTS：09/10/23 16:06:44 32(+32时区)02 - 短信内容字节长度(此处是16进制数，若是7-bit表示解码后的字节长度)6D4B - 短信内容(0x6D4B表示汉字“测”)

2009年11月3日补充：关于收到超长短信的问题上面代码中并没有处理超长短信，但是处理方法很简单，请阅读下面的内容。

一条短信的长度是有限的，不能超过140个Octs，因此，当我们发送超长短信的时候，需要被分割成若干条。那么，在接收PDU串的解码中，会占用掉前6个Octs用于记录超长短信的报文标志——所以你会发现，即使用手机发送超长短信的时候，分割后的汉字字符是按70-3＝67个来计算的，即每67个汉字会被分成一条消息(不必奇怪，手机提示比较人性化，第1页提示最多输入70个汉字，第2页提示最多输入64个汉字，以后每页提示最多输入67个汉字)。总之，最终要保证一条短信长度不超过140个Octs。

这6个Octs的超长短信报文标志会放在短信内容的最前6个Octs当中，也就是前面“手机接收的PDU串分析”中紧随“短信内容字节长度”之后。这6位代表的意义如下：(参见[这里](http://geocom.hhcc.net.cn/magz/wk0704/03.htm)的1.4超长短信)

假设短信内容前6位是：0500037E020105 03 02 00 7E 0105 - 协议长度（后面占5位）00 - 表示拆分短信03 - 拆分数据的长度（后面的3位）7E - 唯一标识（用于把多条短信合并）02 - 共被拆分2条短信01 - 序号，这是其中的第1条短信那么，第2条短信的头6位数据就应该是：0500037E02022010年3月8日补充：关于收到超长短信的问题2

感谢[565730166](http://hi.baidu.com/lfcomputer)的讨论！

超长短信(Long SMS)官方叫法是Concatenated SMS(CSMS)，大家可以到[维基百科](http://en.wikipedia.org/wiki/Concatenated_SMS)中查阅资料。上面关于超长短信的描述并不完整，实际上，针对存储地址类型不同，它可以分为两种类型：8位表示法和16位表示法。8位表示法，就是前面描述的办法。此次补充16位表示法。为了把CSMS表达清楚，下面引用了[国外一篇CSMS解释的文章](http://www.mediaburst.co.uk/blog/concatenated-sms/)中的图片。

简单描述一下：SMS短信的内容部分共140个字节长，CSMS把这140个字节分成两个部分，第一部分叫做UDH(User Data Header，用户数据头)，第二部分是短信内容的信息部分。现在关注的就是UDH怎么解释。UDH又分成三块（按图来的，似乎不太合理，大家明白了就好），第一块（UDHL），一个字节，表示其后UDH占用的字节长度；第二块，三个字节或四个字节，它们分别叫作IEI（Information Elements Identifier，信息元素标识位）、IEDL（Information Elements Data Length，信息元素数据长度）和IED（Information Elements Data，信息元素数据）的引用部分；第三块，IED的剩余部分，信息分隔数和当前信息编号组成。

IEI有两种标识，00和08，00是单字节(8位)信息地址引用号码表示方式，08是双字节(16信)信息地址引用号码表示方式。前面超长信息的解释，只解释的IEI为00的情况。所以，当采用08标识时，整个UDH部分会多出一个字节，用来表示16位的引用号码。假设短信内容UDH部分（此例为前7位）是：060804F42E020106 04 02 08 F42E 0106 - UDHL08 - IEI 表示用16位表示引用方式04 - IEDLF42E - 唯一标识（用于把多条短信合并）02 - 共被拆分2条短信01 - 序号，这是其中的第1条短信那么，第2条短信的UDH应该是：060804F42E0202

2009年12月14日补充：关于SMS字母表的问题SMS字母表与ASCII的字母表不同，因此，使用前面算法转换符号编码时并不会得到原始的输入符号。见讨论7和讨论34、36。请参考：[官方的GSM 03.38转换Unicode映射表格](http://www.unicode.org/Public/MAPPINGS/ETSI/GSM0338.TXT)