基于乐鑫芯片的smartComfig连接协议Android端具体技术实现

本文是基于乐鑫芯片的smartConfig协议的具体实现，为嵌入式端实现提供具体方案。连接协议主要包括三个部分，前导识别，数据传输，验证连接。

- 前导识别

前导是为了让wifi设备在一定时间内寻找ssid和切换通道。准备接收数据。本协议的前导识别包发送格式如下：

515个1	514个1	513个1	512个1

。

。

。

515个1	514个1	513个1	512个1

。

。

515个1	514个1	513个1	512个1

每组4个包，每个包内容分别为515个1，514个1，1513个1，1512个1，如此往复直至20s后停止。

- 数据传输

数据传输主要用到的几个字段如下：

apSsid，apBssid，apPwd，inetAddress，isSsidHiden。其中apBssid为路由器的mac地址，inetAddress
为当前发送终端的ip地址，isSsidHiden表示当前路由器是否是可见的。下面以一个具体的实例来详细说明。

apSsid = chenjie ， apBssid =4c:32:75:8d:00:d5 apPwd = 1234567890 inetAddress =/192.168.2.7 isSsidHiden=false；

- 数据准备

根据如上源数据，获取如下字段，apPwdLen , apssidCrc, apBssidCrc,apSsidLen,ipAddrChars,_totalLen，具体实现如下，Crc后缀字段均是对应字段CRC算法加密后的结果，备用字段方便下面方法处理

```
char apPwdLen = (char) ByteUtil.getBytesByString(apPassword).length;

CRC8 crc = new CRC8();
crc.update(ByteUtil.getBytesByString(apSsid));
char apSsidCrc = (char) crc.getValue();


crc.reset();
crc.update(EspNetUtil.parseBssid2bytes(apBssid));//
char apBssidCrc = (char) crc.getValue();


char apSsidLen = (char) ByteUtil.getBytesByString(apSsid).length;


String ipAddrStrs[] = ipAddress.getHostAddress().split("\\.");
int ipLen = ipAddrStrs.length;
char ipAddrChars[] = new char[ipLen];
// only support ipv4 at the moment
for (int i = 0; i < ipLen; ++i) {
ipAddrChars[i] = (char) Integer.parseInt(ipAddrStrs[i]);
}
```

```
char _totalLen = (char) (EXTRA_HEAD_LEN + ipLen + apPwdLen + apSsidLen);
```

对应测试实例输出结果：

字段	字符结果	Dec（十进制）
apPwdLen	'\n'	10
apSsidCrc	'6'	54
apBssidCrc	'<'	60
apSsidLen	'\u0007'	7
ipAddrChars	',' ¨ ','\u0002','\u0007'	192,168,2,7
_totalLen	'\u001A'	26
totalLen	'\u0013'	19

- 数据加密

数据准备完毕，接下来创建一个一维数组，mDataCodes。我们暂时先不去纠结DataCode类做了什么，构建的mDataCodes数组中，第一个值用来保存总长度，第二个值表示apPwdLen的长度，第三个值表示apSsidCrc，第四个值表示apBssidCrc，按照规律上面预先准备的值每一个都转变成一个DataCode对象存放在mDataCodes数组中，如此往下。。。第四个值设置为null，不明所以，暂不去管，EXTRA_HEAD_LEN
常量为5，说明从第五个值开始保存的是ipAddrChars，如下可知，第EXTRA_HEAD_LEN + ipLen 开始保存pwdChars，如果isSsidHiden为true，后面从EXTRA_HEAD_LEN + ipLen + apPwdLen开始还会保存ssidChars。在赋值的同时，我们也发现每次的赋值都会伴有一次异或运算，totalXor初始值为0，最后的结果将保存到第四个值中，根据其他协议的经验猜测此字段属于校验字段。

```
mDataCodes = new DataCode[totalLen];
mDataCodes[0] = new DataCode(_totalLen, 0);
totalXor ^= _totalLen;
mDataCodes[1] = new DataCode(apPwdLen, 1);
totalXor ^= apPwdLen;
mDataCodes[2] = new DataCode(apSsidCrc, 2);
totalXor ^= apSsidCrc;
mDataCodes[3] = new DataCode(apBssidCrc, 3);
totalXor ^= apBssidCrc;
```

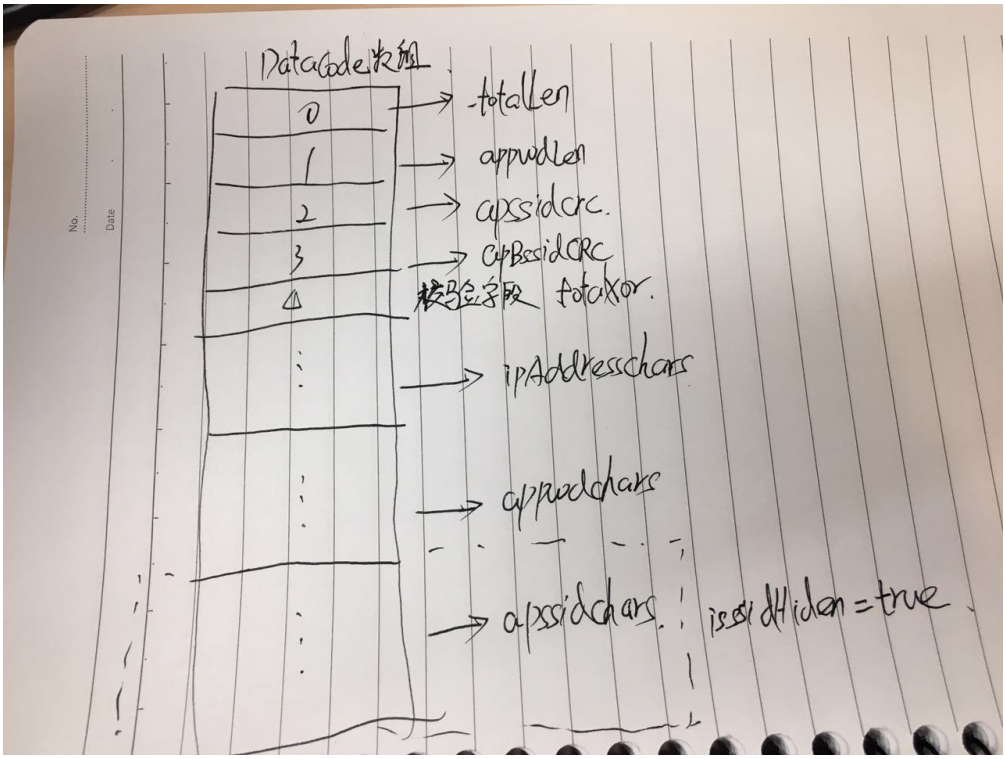
```
mDataCodes[4] = null;
for (int i = 0; i < ipLen; ++i) {
mDataCodes[i + EXTRA_HEAD_LEN] = new DataCode(ipAddrChars[i], i + EXTRA_HEAD_LEN);
 totalXor ^= ipAddrChars[i];
}

byte[] apPwdBytes = ByteUtil.getBytesByString(apPassword);
char[] apPwdChars = new char[apPwdBytes.length];
for (int i = 0;i < apPwdBytes.length; i++) {
apPwdChars[i] = ByteUtil.convertByte2Uint8(apPwdBytes[i]);
}
for (int i = 0; i < apPwdChars.length; i++) {
mDataCodes[i + EXTRA_HEAD_LEN + ipLen] = new DataCode(
apPwdChars[i], i + EXTRA_HEAD_LEN + ipLen);
 totalXor ^= apPwdChars[i];
}

byte[] apSsidBytes = ByteUtil.getBytesByString(apSsid);
char[] apSsidChars = new char[apSsidBytes.length];
// totalXor will xor apSsidChars no matter whether the ssid is hidden
for (int i = 0; i < apSsidBytes.length; i++) {
apSsidChars[i] = ByteUtil.convertByte2Uint8(apSsidBytes[i]);
 totalXor ^= apSsidChars[i];
}
if (isSsidHiden) {
for (int i = 0; i < apSsidChars.length; i++) {
mDataCodes[i + EXTRA_HEAD_LEN + ipLen + apPwdLen] = new DataCode(
apSsidChars[i], i + EXTRA_HEAD_LEN + ipLen + apPwdLen);
 }
}

// set total xor last
mDataCodes[4] = new DataCode(totalXor, 4);
```

简单的数据队列如下：



字段	字符结果	Dec（十进制）	Hex(十六进制)	高位（high）	低位（low）	CrcHigh	CrcLow	**mSeqHeaderindex**

_totalLen	'\u001A'	26	1A	1	10	0	11	0
apPwdLen	'\n'	10	0A	0	10	11	9	1
apSsidCrc	'6'	54	36	3	6	3	11	2
apBssidCrc	'<'	60	3c	3	12	8	2	3
totalXor	'\u0010'	16	10	1	0	8	13	4
ipAddrChars	',' ¨ ','\u0002','\u0007'	192,168,2,7	c0,a8,2,7	12,10,0,0	0,8,2,7	8，4，1，10	11，5，2，12	5，6，7，8
apPwdChars	'1','2','3','4','5','6','7','8','9','0'	49,50,51,52...57,48	31,32,33,...39,30	3,3,3,3,3,3,3,3,3,3	1,2,3,4,5,6,7,8,9,0	7,12,5,11,2,9,0,12,5,0	5,2,8,5,2,8,2,6,12,12	9，10,11,12,13,14,15,16,17,

但这并不是最终结果，我们需要把dataCode对象数组转换成字符数组才能够发包，再看getU8s，该方法中首先获取了getBytes()方法，这里用到了我们刚刚搁置不管的那个类DataCode，这个类中的getBytes方法返回了一个长度为6位的数组，该方法将之前传递的字符和index按照一定规则转换为6位数组。具体规则如下：

0	0x00
1	`combine2bytesToOne(mCrcHigh,mDataHigh)`
2	`0x01`
3	`(byte) index`
4	`0x00`
5	`combine2bytesToOne(mCrcLow, mDataLow)`

```
public class DataCode implements ICodeData{

public static final int DATA_CODE_LEN = 6;

private static final int INDEX_MAX = 127;

private final byte mSeqHeader;
private final byte mDataHigh;
private final byte mDataLow;
 // the crc here means the crc of the data and sequence header be transformed
// it is calculated by index and data to be transformed
 private final byte mCrcHigh;
private final byte mCrcLow;

 /**
* Constructor of DataCode
* @param u8 the character to be transformed
* @param index the index of the char
*/
 public DataCode(char u8, int index) {
if (index > INDEX_MAX) {
throw new RuntimeException("index > INDEX_MAX");
 }
byte[] dataBytes = ByteUtil.splitUint8To2bytes(u8);
 mDataHigh = dataBytes[0];
 mDataLow = dataBytes[1];
 CRC8 crc8 = new CRC8();
 crc8.update(ByteUtil.convertUint8toByte(u8));
 crc8.update(index);
byte[] crcBytes = ByteUtil.splitUint8To2bytes((char) crc8.getValue());
 mCrcHigh = crcBytes[0];
 mCrcLow = crcBytes[1];
 mSeqHeader = (byte) index;
 }

@Override
 public byte[] getBytes() {
byte[] dataBytes = new byte[DATA_CODE_LEN];
 dataBytes[0] = 0x00;
 dataBytes[1] = ByteUtil.combine2bytesToOne(mCrcHigh,mDataHigh);
 dataBytes[2] = 0x01;
 dataBytes[3] = mSeqHeader;
 dataBytes[4] = 0x00;
 dataBytes[5] = ByteUtil.combine2bytesToOne(mCrcLow, mDataLow);
return dataBytes;
 }
}

public static byte combine2bytesToOne(byte high, byte low) {
if (high < 0 || high > 0xf || low < 0 || low > 0xf) {
throw new RuntimeException("Out of Boundary");
 }
return (byte) (high << 4 | low);
}
```

字段	字符结果	Dec（十进制）	Hex(十六进制)	高位（high）	低位（low）	CrcHigh	CrcLow	**mSeqHeaderindex**
_totalLen	'\u001A'	26	1A	1	10	0	11	0
apPwdLen	'\n'	10	0A	0	10	11	9	1
apSsidCrc	'6'	54	36	3	6	3	11	2
apBssidCrc	'<'	60	3c	3	12	8	2	3
totalXor	'\u0010'	16	10	1	0	8	13	4

ipAddrChars	',' ¨ ','\u0002','\u0007'	192,168,2,7	c0,a8,2,7	12,10,0,0	0,8,2,7	8,4,1,10	11,5,2,12	5,6,7,8
apPwdChars	'1','2','3','4','5','6','7','8','9','0'	49,50,51,52...57,48	31,32,33,...39,30	3,3,3,3,3,3,3,3,3,3	1,2,3,4,5,6,7,8,9,0	7,12,5,11,2,9,0,12,5,0	5,2,8,5,2,8,2,6,12,12	9,10,11,12,13,14,15,16,1

可以理解为DataCode这个类其实就是一个字符通过一定的规则转换为byte[]的类。到这里我们发现一个dataCode对象返回的bytes长度为6，每一个字符又都有规则的保存在mDataCodes数组中，故我们将mDataCodes数组整体转换为byte[]，这样getBytes()方法就不难理解了。getU8s方法这里又做了一次混码，将偶数位设为高位，奇数位设为低位，再combine2bytesToU16 转换成16进制再+EXTRA_LEN（等于40，define by the Esptouch protocol, all of the datum code should add 1 at last to prevent 0），这样就完成了整个的混码输出bytes。

```
DatumCode dc = new DatumCode(apSsid, apBssid, apPassword, inetAddress,
 isSsidHiden);
char[] dcU81 = dc.getU8s();
```

```
@Override
public char[] getU8s() {
byte[] dataBytes = getBytes();
int len = dataBytes.length / 2;
char[] dataU8s = new char[len];
byte high, low;
for (int i = 0; i < len; i++) {
high = dataBytes[i * 2];
 low = dataBytes[i * 2 + 1];
 dataU8s[i] = (char) (ByteUtil.combine2bytesToU16(high, low) + EXTRA_LEN);
 }
return dataU8s;
}
```

```
@Override
public byte[] getBytes() {
byte[] datumCode = new byte[mDataCodes.length * DataCode.DATA_CODE_LEN];
for (int i = 0; i < mDataCodes.length; i++) {
byte[] bytes =mDataCodes[i].getBytes();
 System.arraycopy(bytes, 0, datumCode, i
* DataCode.DATA_CODE_LEN, DataCode.DATA_CODE_LEN);
 }
return datumCode;
}
```

```
public static char combine2bytesToU16(byte high, byte low) {
char highU8 = convertByte2Uint8(high);
char lowU8 = convertByte2Uint8(low);
return (char) (highU8 << 8 | lowU8);
}
```

```
public static char convertByte2Uint8(byte b) {
 return (char) (b & 0xff);
}
```

最后一步，将如上表中的所有数据拷贝到一个byte数组中，合计共114个值，将位置为偶数位的作为high，将奇数位的作为low，再次

```
(char) (ByteUtil.combine2bytesToU16(high, low) + EXTRA_LEN);charcharbyte[]
```


```
public static byte[] genSpecBytes(char len) {
byte[] data = new byte[len];
for (int i = 0; i < len; i++) {
data[i] = '1';
 }
return data;
}
```

最终传值为一个一维含有57个值的二维数组

41个1	296个1	226个1	216个1	297个1	194个1	91个1	298个1	222个1	171个1	299个1	.....

- 数据传输

数据准备完毕，我们可以开始发送，该demo采用了while循环方式，使用时间差来中断，切换发送包，支持设置发送次数。此处对DatagramSocket的send方法做了封装，支持一个数据包发送多次的需求，demo中默认是3次，即每一个数据包都会发三次。

另外该demo中将targetHostName设置为动态循环获取，target hostname is : 234.1.1.1, 234.2.2.2, 234.3.3.3 to 234.100.100.100，234为保留组包ip网段。

```
int index = 0;
while (!mIsInterrupt) {
if (currentTime - lastTime >= mParameter.getTimeoutTotalCodeMillisecond()) {
if (__IEsptouchTask.DEBUG) {
log.d(TAG, "send gc code ");
 }
// send guide code
 while (!mIsInterrupt
 && System.currentTimeMillis() - currentTime < mParameter
 .getTimeoutGuideCodeMillisecond()) {
mSocketClient.sendData(gcBytes2,
 mParameter.getTargetHostname(),
 mParameter.getTargetPort(),
 mParameter.getIntervalGuideCodeMillisecond());
 // check whether the udp is send enough time
 if (System.currentTimeMillis() - startTime > mParameter.getWaitUdpSendingMillisecond()) {
break;
 }
}
lastTime = currentTime;
 } else {
mSocketClient.sendData(dcBytes2, index, ONE_DATA_LEN,
 mParameter.getTargetHostname(),
 mParameter.getTargetPort(),
 mParameter.getIntervalDataCodeMillisecond());
 index = (index + ONE_DATA_LEN) % dcBytes2.length;
 }
currentTime = System.currentTimeMillis();
 // check whether the udp is send enough time
 if (currentTime - startTime > mParameter.getWaitUdpSendingMillisecond()) {
break;
 }
}
```

- 验证连接

和其他协议一样，在准备发包之前，也同时创建一个UDPSocketServer来准备接收验证包，不同之处在于接收包的规则，这里约定了mEsptouchResultTotalLen = 1 + 6 + 4;

（1代表验证值，6代表有6个字节表示bssid，4代表inetAddress）。作为有效包的总长度，当接收到的包的byte[]长度是11时，则认为此包是有效包，然后再获取当前有效包byte[]的第1个值，如果当前值与(byte) (apSsidAndPassword.length + 9); 相等，则表明验证通过。通过对byte数组的分析可获取到bssid和inetAddress，具体实现见下方代码：

1	2.....7	8.....11
(byte) (apSsidAndPassword.length + 9)	bssid	inetAddress

```
private void __listenAsyn(final int expectDataLen) {
new Thread() {
public void run() {
if (__IEsptouchTask.DEBUG) {
log.d(TAG, "__listenAsyn() start");
 }
long startTimestamp = System.currentTimeMillis();
byte[] apSsidAndPassword = ByteUtil.getBytesByString(mApSsid
 + mApPassword);
byte expectOneByte = (byte) (apSsidAndPassword.length + 9);
if (__IEsptouchTask.DEBUG) {
log.i(TAG, "expectOneByte: " + (0 + expectOneByte));
 }
byte receiveOneByte = -1;
byte[] receiveBytes = null;
while (mEsptouchResultList.size() < mParameter
 .getExpectTaskResultCount() && !mIsInterrupt) {
receiveBytes = mSocketServer
 .receiveSpecLenBytes(expectDataLen);
if (receiveBytes != null) {
receiveOneByte = receiveBytes[0];
 } else {
receiveOneByte = -1;
 }
if (receiveOneByte == expectOneByte) {
if (__IEsptouchTask.DEBUG) {
log.i(TAG, "receive correct broadcast");
 }
// change the socket's timeout
 long consume = System.currentTimeMillis()
- startTimestamp;
int timeout = (int) (mParameter
 .getWaitUdpTotalMillisecond() - consume);
if (timeout < 0) {
if (__IEsptouchTask.DEBUG) {
log.i(TAG, "esptouch timeout");
 }
break;
 } else {
if (__IEsptouchTask.DEBUG) {
log.i(TAG, "mSocketServer's new timeout is "
 + timeout + " milliseconds");
 }
mSocketServer.setSoTimeout(timeout);
if (__IEsptouchTask.DEBUG) {
log.i(TAG, "receive correct broadcast");
 }
if (receiveBytes != null) {
String bssid = ByteUtil.parseBssid(
receiveBytes,
 mParameter.getEsptouchResultOneLen(),
 mParameter.getEsptouchResultMacLen());
 InetAddress inetAddress = EspNetUtil
.parseInetAddr(
receiveBytes,
 mParameter.getEsptouchResultOneLen()
+ mParameter.getEsptouchResultMacLen(),
```

```
 mParameter.getEsptouchResultIpLen());
 __putEsptouchResult(true, bssid, inetAddress);
 }
}
} else {
if (__IEsptouchTask.DEBUG) {
log.i(TAG, "receive rubbish message, just ignore");
 }
}
}
mIsSuc = mEsptouchResultList.size() >= mParameter
 .getExpectTaskResultCount();
 __EsptouchTask.this.__interrupt();
if (__IEsptouchTask.DEBUG) {
log.d(TAG, "__listenAsyn() finish");
 }
```

```
}
}.start();
}
```