**[蓝牙HDP协议源码解析](http://blog.csdn.net/u012439416/article/details/54348235)**

**1,概述**

**1.1 HDP协议**

HDP协议: Health Device Profile健康设备协议

使用场景:支持各种蓝牙健康设备和手机进行通信

市场产品:广泛应用于各种智能穿戴设备,比如蓝牙健康手环,蓝牙血压计,蓝牙温度计,蓝牙电子称等各种健康设备。

**1.2 代码路径**

客户端: frameworks\base\core\[**Java**](http://lib.csdn.net/base/javase)\[**Android**](http://lib.csdn.net/base/android)\bluetooth

  BluetoothHealth.java            hdp协议客户端

  BluetoothHealthAppConfiguration.java  设备连接通信的相关信息

  BluetoothHealthCallback.java     用于第三方app注册的回调方法

服务端: packages\apps\Bluetooth\src\com\android\bluetooth\ hdp

HealthService.java                 hdp协议的服务端

这4个类不是很难,主要看逻辑以及接口

**2,接口**

接口如下



首先要清楚一个概念,手机和健康设备(比如蓝牙血压计)连接,通常是将健康设备的数据发送给手机,一般来说,健康设备是数据的输出端,手机是数据的输入端,而开发也是针对手机端的开发。

看这些接口,居然没有Connect接口,这不科学啊！

connectChannelToSource手机主动连接健康设备时的接口。相当于其他协议的Connect接口。

connectChannelToSink蓝牙设备主动连接手机的接口。

registerSinkAppConfiguration注册方法,主要监听蓝牙信道的状态。最终是调用registerAppConfiguration接口来完成。

其他的接口从名字就知道什么意思,就不多论述了。

**3,开发步骤**

在官方文档中有一个建立通信的流程：

1、调用getProfileProxy(Context,BluetoothProfile.ServiceListener, int)来获取代理对象的连接。

2、创建BluetoothHealthCallback回调，调用registerSinkAppConfiguration方法注册一个sink端的应用程序配置。

3、将手机与健康设备配对，这一步在手机的设置中就可以完成。

4、使用connectChannelToSource方法来建立一个与健康设备的通信channel。有的设备会自动建立通信，不需要在代码中调用这个方法。第二步中的回调会指示channel的状态变化。

5、用ParcelFileDescriptor来读取健康设备传来的数据，并根据IEEE 11073-\*\*\*\*协议来解析数据。

6、通信结束后，关闭通信channel，取消第二步中的注册。

蓝牙的基本操作(打开,配对,连接等)就不论述了,主要说明手机端和蓝牙血压计如何利用IEEE 11073协议进行通信。

**3.1 获取客户端代理对象**

一般在oncreate方法中,直接调用getProfileProxy方法,这个没什么好说的。

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

1. BluetoothAdapter.getDefaultAdapter().getProfileProxy(getApplicationContext(),
2. mProfileServiceListener,BluetoothProfile.HEALTH);

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. private BluetoothProfile.ServiceListener mProfileServiceListener = new BluetoothProfile.ServiceListener() {
3. @Override
4. public void onServiceDisconnected(int profile) {
5. if (profile == BluetoothProfile.HEALTH){
6. mBluetoothHealth = null;
7. }
8. }
10. @SuppressLint("NewApi")
11. @Override
12. public void onServiceConnected(int profile, BluetoothProfile proxy) {
13. if (profile == BluetoothProfile.HEALTH) {
14. mBluetoothHealth = (BluetoothHealth) proxy;
15. }
16. }
17. };

一般经过这个步骤,客户端的BluetoothHealth对象已经和服务端的HealthService对象绑定了。

**3.2 注册BluetoothHealthAppConfiguration**

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. private static final String TAG = "BluetoothHealth"; // 这只是个标志,可以任意
2. private static final int dataType = 0x1007; //IEEE 11073中规定的血压数据类型
3. private BluetoothHealthAppConfiguration mHealthAppConfig;
4. private int mChannelId;

7. mBluetoothHealth.registerSinkAppConfiguration(TAG, dataType, mHealthCallback);

10. private final BluetoothHealthCallback mHealthCallback = new BluetoothHealthCallback() {
12. public void onHealthAppConfigurationStatusChange(BluetoothHealthAppConfiguration
13. config, int status) {
14. if (status == BluetoothHealth.APP\_CONFIG\_REGISTRATION\_FAILURE) {
15. mHealthAppConfig = null; // 注册失败
16. } else if (status == BluetoothHealth.APP\_CONFIG\_REGISTRATION\_SUCCESS) {
17. mHealthAppConfig = config; // 注册成功
18. } else if (status == BluetoothHealth.APP\_CONFIG\_UNREGISTRATION\_FAILURE ||
19. status == BluetoothHealth.APP\_CONFIG\_UNREGISTRATION\_SUCCESS) {
20. // 取消注册的广播,开发很少用到
21. }
22. }

25. public void onHealthChannelStateChange(BluetoothHealthAppConfiguration config,
26. BluetoothDevice device, int prevState, int newState, ParcelFileDescriptor fd,
27. int channelId) {
28. if (prevState == BluetoothHealth.STATE\_CHANNEL\_DISCONNECTED &&
29. newState == BluetoothHealth.STATE\_CHANNEL\_CONNECTED) {
30. if (config.equals(mHealthAppConfig)) {
31. mChannelId = channelId; // 连接成功
32. (new ReadThread(fd)).start();// 开始读取数据
33. } else {
34. // 连接失败  一般下面的几个连接状态开发中也很少用到,可以输出log
35. }
36. } else if (prevState == BluetoothHealth.STATE\_CHANNEL\_CONNECTING &&
37. newState == BluetoothHealth.STATE\_CHANNEL\_DISCONNECTED) {
38. •••
39. } else if (newState == BluetoothHealth.STATE\_CHANNEL\_DISCONNECTED) {
40. •••
41. if (config.equals(mHealthAppConfig)) {
42. •••
43. } else {
44. •••
45. }
46. }
47. }
48. };

**3.3 建立channel**

在设置中配对连接这个就不论述了,主要论述一下Pan协议的连接。

在手机端,调用connectChannelToSource接口就可以了,

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. mBluetoothHealth.connectChannelToSource(device,mHealthAppConfig);

在蓝牙血压计端,调用的是 connectChannelToSink接口

**3.4 读取解析数据**

在第二步中注册后, 在onHealthChannelStateChange方法中监听了channel的变化,一旦channel建立成功,则利用回调的ParcelFileDescriptor对象另开一个读取数据的线程,读取channel中的数据。

下面是手机端获取血压计的参数:收缩压,舒张压,心率以及测量时间。

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. int count = 0; // 交互通信的次数
2. byte invoke[] = new byte[] { (byte) 0x00, (byte) 0x00 };
4. private class ReadThread extends Thread {
5. private ParcelFileDescriptor mFd;
7. public ReadThread(ParcelFileDescriptor fd) {
8. super();
9. mFd = fd;
10. }

13. @Override
14. public void run() {
15. FileInputStream fis = new FileInputStream(mFd.getFileDescriptor());
16. byte data[] = new byte[300];
17. try {
18. while(fis.read(data) **>** -1) {
19. if (data[0] != (byte) 0x00) {
20. String test = byte2hex(data);
21. if(data[0] == (byte) 0xE2){ // 请求连接的信号
22. count = 1;
23. (new WriteThread(mFd)).start();
24. try {
25. sleep(100);
26. } catch (InterruptedException e) {
27. e.printStackTrace();
28. }
29. count = 2;
30. (new WriteThread(mFd)).start();
31. }else if (data[0] == (byte)0xE7){
32. if (data[18] == (byte) 0x0d && data[19] == (byte) 0x1d) {
33. count = 3;
34. invoke = new byte[] { data[6], data[7] };
35. (new WriteThread(mFd)).start();
36. int length = data[21];
37. int number\_of\_data\_packets = data[22+5];
38. int packet\_start = 30;
39. final int SYS\_DIA\_MAP\_DATA = 1;
40. final int PULSE\_DATA = 2;
41. final int ERROR\_CODE\_DATA = 3;
42. for (int i = 0; i **<** **number\_of\_data\_packets**; i++)
43. {
44. int obj\_handle = data[packet\_start+1];
45. switch (obj\_handle)
46. {
47. case SYS\_DIA\_MAP\_DATA:
48. int sys = byteToUnsignedInt(data[packet\_start+9]); // 收缩压
49. int dia = byteToUnsignedInt(data[packet\_start+11]);// 舒张压
50. int map = byteToUnsignedInt(data[packet\_start+13]);
51. break;
52. case PULSE\_DATA:
53. int pulse = byteToUnsignedInt(data[packet\_start+5]); // 心率
54. String month = byteToString(data[packet\_start + 8]);// 以下是测量时间
55. String day = byteToString(data[packet\_start + 9]);
56. String year = byteToString(data[packet\_start + 6]) +
57. byteToString(data[packet\_start + 7]);
58. String hour = byteToString(data[packet\_start + 10]);
59. String minute = byteToString(data[packet\_start + 11]);
60. String date = year + "." + month + "." + day + " " + hour + ":" + minute;
61. break;
62. case ERROR\_CODE\_DATA:
63. break;
64. }
65. packet\_start += 4 + data[packet\_start+3]; //4 = ignore beginning four bytes
66. }
67. }else{
68. count = 2;
69. }
70. }else if (data[0] == (byte) 0xE4) { // 结束
71. count = 4;
72. (new WriteThread(mFd)).start();
73. }
74. //zero out the data
75. for (int i = 0; i **<** **data.length**; i++){
76. data[i] = (byte) 0x00;
77. }
78. }
79. }
80. } catch(IOException ioe) {}
81. if (mFd != null) {
82. try {
83. mFd.close();
84. } catch (IOException e) { /\* Do nothing. \*/ }
85. }
86. }
87. }

sys, dia等数据就是最后的结果,我们就可以拿这些数据进行处理了。

通信是双方的,设备向手机发送信息,手机也可以回复设备的信息,所以还要另外一个写数据的线程,实现手机端和血压计之间的数据交换。

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. final byte data\_AR[] = new byte[] {(byte) 0xE3, (byte) 0x00,
2. (byte) 0x00, (byte) 0x2C, (byte) 0x00, (byte) 0x00, (byte) 0x50, (byte) 0x79,
3. (byte) 0x00, (byte) 0x26, (byte) 0x80, (byte) 0x00, (byte) 0x00, (byte) 0x00,
4. (byte) 0x80, (byte) 0x00, (byte) 0x80, (byte) 0x00, (byte) 0x00, (byte) 0x00,
5. (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x80, (byte) 0x00,
6. (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x08,
7. (byte) 0x3C, (byte) 0x5A, (byte) 0x37, (byte) 0xFF, (byte) 0xFE, (byte) 0x95,
8. (byte) 0xEE, (byte) 0xE3, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
9. (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00};
11. //Presentation APDU [0xE700]
12. final byte data\_DR[] = new byte[] { (byte) 0xE7, (byte) 0x00, (byte) 0x00, (byte) 0x12,
13. (byte) 0x00, (byte) 0x10, (byte) invoke[0], (byte) invoke[1],
14. (byte) 0x02, (byte) 0x01, (byte) 0x00, (byte) 0x0A, (byte) 0x00, (byte) 0x00,
15. (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x0D, (byte) 0x1D,
16. (byte) 0x00, (byte) 0x00 };
18. final byte get\_MDS[] = new byte[] { (byte) 0xE7, (byte) 0x00,
19. (byte) 0x00, (byte) 0x0E, (byte) 0x00, (byte) 0x0C, (byte) 0x00, (byte) 0x24,
20. (byte) 0x01, (byte) 0x03, (byte) 0x00, (byte) 0x06, (byte) 0x00, (byte) 0x00,
21. (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00 };
23. final byte data\_RR[] = new byte[] {(byte) 0xE5, (byte) 0x00,
24. (byte) 0x00, (byte) 0x02, (byte) 0x00, (byte) 0x00 };
26. final byte data\_RRQ[] = new byte[] {(byte) 0xE4, (byte) 0x00,
27. (byte) 0x00, (byte) 0x02, (byte) 0x00, (byte) 0x00 };
29. final byte data\_ABORT[] = new byte[] { (byte) 0xE6, (byte) 0x00,
30. (byte) 0x00, (byte) 0x02, (byte) 0x00, (byte) 0x00 };

33. private class WriteThread extends Thread {
34. private ParcelFileDescriptor mFd;
36. public WriteThread(ParcelFileDescriptor fd) {
37. super();
38. mFd = fd;
39. }
41. @Override
42. public void run() {
43. FileOutputStream fos = new FileOutputStream(mFd.getFileDescriptor());
45. try {
46. Log.i(TAG, String.valueOf(count));
47. if (count == 1) {
48. fos.write(data\_AR);
49. }  else if (count == 2) {
50. fos.write(get\_MDS);
51. //fos.write(data\_ABORT);
52. }else if (count == 3) {
53. fos.write(data\_DR);
54. }else if (count == 4) {
55. fos.write(data\_RR);
56. }
57. } catch(IOException ioe) {}
58. }
59. }
61. public String byte2hex(byte[] b){
62. String hs = "";
63. String stmp = "";
64. for (int n = 0; n **<** **b.length**; n++){
65. stmp = (java.lang.Integer.toHexString(b[n] & 0XFF));
66. if (stmp.length() == 1) {
67. hs = hs + "0" + stmp;
68. }else {
69. hs = hs + stmp;
70. }
71. if (n **<** **b.length** - 1) {
72. hs = hs + "";
73. }
74. }
76. return hs;
77. }
79. public static int byteToUnsignedInt(byte b) {  // byte转int
80. return 0x00 **<<** **24** | b & 0xff;
81. }
83. public static String byteToString(byte b) { // byte转String
84. return String.valueOf(0x00 **<<** **24** | b & 0xff);
85. }

看这些读和写的代码有些头晕,幸好count记下了相互交流的次数,其实逻辑很简单.

1,建立channel时,血压计发送请求连接的信号,手机端回复同意连接

2,手机端发送血压计相关数据,血压计发送相关数据。

3,手机端确认接收数据。

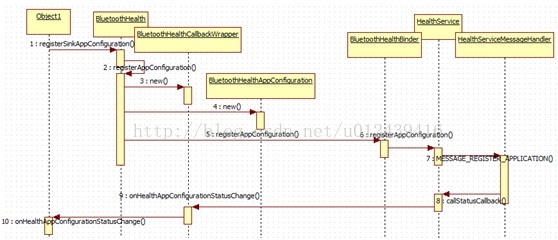
4,发送完数据之后,血压计请求断开连接,最后手机端确认断开连接。

至于发送的数据是什么意思,参考IEEE 11073中的内容,其中IEEE 11073－10407是专门针对血压计的协议。

到此为止,我们已经可以拿到最后的结果进行处理了,但是这些远远不够,说好的源码呢,这些背后的逻辑是什么呢?

**4,源码解析**

**4.1 注册流程分析**



主要分为2个步骤:

1,跨进程调用,从客户端调用到服务端对应的方法去注册

2,服务端将注册的结果回调给客户端, 然后我们根据注册做其他事情

其实,在这儿我觉得将注册的结果直接发送广播更好,方便直接.

首先看看registerSinkAppConfiguration方法,

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. public boolean registerSinkAppConfiguration(String name, int dataType,
2. BluetoothHealthCallback callback) {
3. if (!isEnabled() || name == null) return false;
5. if (VDBG) log("registerSinkApplication(" + name + ":" + dataType + ")");
6. return registerAppConfiguration(name, dataType, SINK\_ROLE,
7. CHANNEL\_TYPE\_ANY, callback);
8. }

这里有5个参数:

第一个参数:任意一个字符,只是标志作用

第二个参数: 数据类型,不同健康设备的类型不一样,详细查阅IEEE 11073协议

第三个参数:设备角色,有如下2个值,

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. public static final int SOURCE\_ROLE = 1 **<<** **0**; // 健康设备端注册
2. public static final int SINK\_ROLE = 1 **<<** **1**;   // 手机端注册

第四个参数: channel通道的类型,有3种类型可选

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. public static final int CHANNEL\_TYPE\_RELIABLE = 10;
2. public static final int CHANNEL\_TYPE\_STREAMING = 11;
3. public static final int CHANNEL\_TYPE\_ANY = 12; // 默认类型

第五个参数:开发者必须实现的BluetoothHealthCallback类以及2个方法。

然后看看MESSAGE\_REGISTER\_APPLICATION这个消息的处理,

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. BluetoothHealthAppConfiguration appConfig =
2. (BluetoothHealthAppConfiguration) msg.obj;
3. AppInfo appInfo = mApps.get(appConfig);
4. if (appInfo == null) break;
5. int halRole = convertRoleToHal(appConfig.getRole());
6. int halChannelType = convertChannelTypeToHal(appConfig.getChannelType());
7. int appId = registerHealthAppNative(appConfig.getDataType(), halRole,
8. appConfig.getName(), halChannelType);
9. if (appId == -1) { // 注册失败
10. callStatusCallback(appConfig,BluetoothHealth.APP\_CONFIG\_REGISTRATION\_FAILURE);
11. appInfo.cleanup();
12. mApps.remove(appConfig);
13. } else { // 注册成功
14. //link to death with a recipient object to implement binderDead()
15. appInfo.mRcpObj = new BluetoothHealthDeathRecipient(HealthService.this,appConfig);
16. IBinder binder = appInfo.mCallback.asBinder();
17. try {
18. binder.linkToDeath(appInfo.mRcpObj,0);
19. } catch (RemoteException e) {
20. Log.e(TAG,"LinktoDeath Exception:"+e);
21. }
22. appInfo.mAppId = appId;
23. callStatusCallback(appConfig, BluetoothHealth.APP\_CONFIG\_REGISTRATION\_SUCCESS);
24. }

linkToDeath这个方法是干嘛的,有什么作用呢?

Binder自然是指BluetoothHealth 的内部类BluetoothHealthCallbackWrapper对象,

appInfo.mRcpObj指BluetoothHealthDeathRecipient对象,

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. private static class BluetoothHealthDeathRecipient implements IBinder.DeathRecipient{
2. private BluetoothHealthAppConfiguration mConfig;
3. private HealthService mService;
5. public BluetoothHealthDeathRecipient(HealthService service,
6. BluetoothHealthAppConfiguration config) {
7. mService = service;
8. mConfig = config;
9. }
11. public void binderDied() {
12. if (DBG) Log.d(TAG,"Binder is dead.");
13. mService.unregisterAppConfiguration(mConfig);
14. }
16. public void cleanup(){
17. mService = null;
18. mConfig = null;
19. }
20. }

直接说了,一旦客户端的app突然崩溃了, BluetoothHealt对象还未来得及调用

unregisterAppConfiguration方法,服务端的注册还在,怎么办呢?通过linkToDeath方法,一旦客户端的app挂了,就会直接调用BluetoothHealthDeathRecipient的binderDied方法,这样,就可以取消服务端的注册。嗯,的确是一个很好的方法。进程调用以下语句可以杀死自己。

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. android.os.Process.killProcess(android.os.Process.myPid());

 注册函数说到底最后还是调用registerHealthAppNative方法完成,根据返回的结果调用callStatusCallback方法,最后调用开发者写的

onHealthAppConfigurationStatusChange方法,我还是觉得不如广播方便,广播多简单啊。

**3.2建立channel**

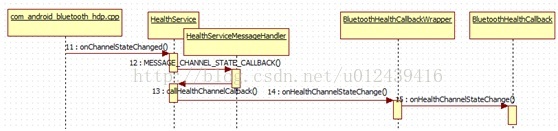
建立channel有2个方法,

1,手机端调用connectChannelToSource方法时会新建channel

2,健康设备调用connectChannelToSink方法时也会新建channel

connectChannelToSource和connectChannelToSink接口的调用流程完全和上小节中注册的流程一样,都会调用HealthService的connectChannel方法,最后会connectChannelNative方法完成最后的连接,根据连接返回的状态通过onHealthChannelStateChange方法监听。

下面主要论述底层连接的反馈过程,流程图如下:



直接看onChannelStateChanged方法,

**[html]** [view plain](http://blog.csdn.net/u012439416/article/details/54348235) [copy](http://blog.csdn.net/u012439416/article/details/54348235)

[在CODE上查看代码片](https://code.csdn.net/snippets/2118722)

1. private void onChannelStateChanged(int appId, byte[] addr, int cfgIndex,
2. int channelId, int state, FileDescriptor pfd) {
3. Message msg = mHandler.obtainMessage(MESSAGE\_CHANNEL\_STATE\_CALLBACK);
4. ChannelStateEvent channelStateEvent = new ChannelStateEvent(appId, addr, cfgIndex,
5. channelId, state, pfd);
6. msg.obj = channelStateEvent;
7. mHandler.sendMessage(msg);
8. }

重点在后面三个参数,分别是新建channel的Id,设备连接的状态以及用于通信的ParcelFileDescriptor对象,这三个主要的对象都从底层传上来了,其他的都是浮云的,按照流程走就可以了。

从onChannelStateChanged方法来看,还是不能发广播的,只能跨进程调用。

**5,小节**

这些都捋清楚了,如果是换一个健康设备,比如蓝牙体重秤呢,如何进行通信?

关键得看IEEE 11073协议了。