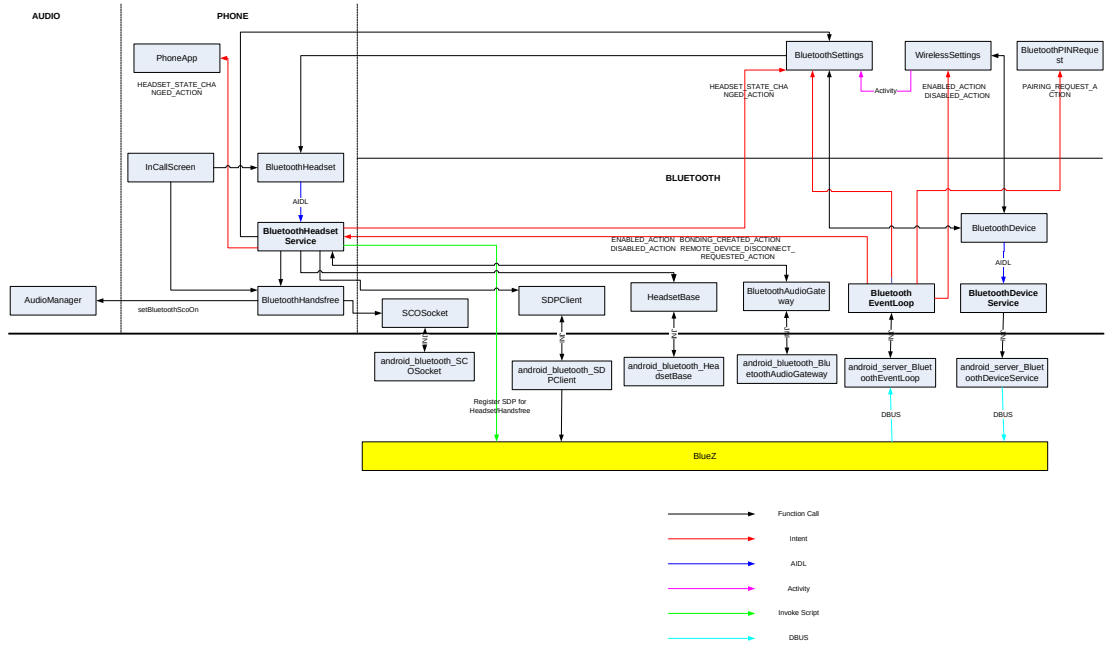
蓝牙模块



在 SystemServer 启动的时候，会生成一个 BluetoothDeviceService 的实例，

```
// Skip Bluetooth if we have an emulator kernel
// TODO: Use a more reliable check to see if this product should
// support Bluetooth - see bug 988521
if (SystemProperties.get("ro.kernel.qemu").equals("1")) {
    Log.i(TAG, "Registering null Bluetooth Service (emulator)");
    ServiceManager.addService(Context.BLUETOOTH_SERVICE, null);
} else if (factoryTest == SystemServer.FACTORY_TEST_LOW_LEVEL) {
    Log.i(TAG, "Registering null Bluetooth Service (factory test)");
    ServiceManager.addService(Context.BLUETOOTH_SERVICE, null);
} else {
    Log.i(TAG, "Starting Bluetooth Service.");
    bluetooth = new BluetoothDeviceService(context);
    bluetooth.init();
    ServiceManager.addService(Context.BLUETOOTH_SERVICE, bluetooth);

    int bluetoothOn = Settings.System.getInt(mContentResolver,

Settings.System.BLUETOOTH_ON, 0);
    if (bluetoothOn > 0) {
        bluetooth.enable(null);
    }
}
```

BluetoothDeviceService 会生成一个 BluetoothEventLoop 实例，它们两者均通过 DBUS 来和 BlueZ 通信。BluetoothDeviceService 是通过 DBUS 向 BlueZ 发送命令，而命令的返回结果则

是由 BlueZ 通过 DBUS 传回给 BluetoothEventLoop 的（具体交互请参见 BlueZ 的 dbus_api.txt ）,BlueZ 也会通过 DBUS 向 BluetoothEventLoop 发送一些事件通知。BluetoothEventLoop 和外部的接口是通过预先定义的 Intent，
初始的时候蓝牙是没有使能的，要通过 BluetoothSettings 或者 WirelessSettings 来打开蓝牙设备，然后通过 BluetoothSettings 去查找附近的其他蓝牙设备，找到后可以建立 RFCOMM 连接和配对。

蓝牙耳机

Android 实现了对 Headset 和 Handsfree 两种 profile 的支持。其实现核心是 BluetoothHeadsetService，在 PhoneApp 创建的时候会启动它。

```
if (getSystemService(Context.BLUETOOTH_SERVICE) != null) {
    mBtHandsfree = new BluetoothHandsfree(this, phone);
    startService(new Intent(this, BluetoothHeadsetService.class));
} else {
    // Device is not bluetooth capable
    mBtHandsfree = null;
}
```

BluetoothHeadsetService 通过接收 ENABLED_ACTION、BONDING_CREATED_ACTION 、DISABLED_ACTION 和 REMOTE_DEVICE_DISCONNECT_REQUESTEDACTION 来改变状态，它也会监听 Phone 的状态变化。

```
IntentFilter filter = new IntentFilter(BluetoothIntent.BONDING_CREATED_ACTION);

filter.addAction(BluetoothIntent.REMOTE_DEVICE_DISCONNECT_REQUESTED_ACTION);
filter.addAction(BluetoothIntent.ENABLED_ACTION);
filter.addAction(BluetoothIntent.DISABLED_ACTION);
registerReceiver(mBluetoothIntentReceiver, filter);

mPhone.registerForPhoneStateChanged(mStateChangeHandler,
PHONE_STATE_CHANGED, null);
```

BluetoothHeadsetService 收到 ENABLED_ACTION 时，会先向 BlueZ 注册 Headset 和 Handsfree 两种 profile（通过执行 sdptool 来实现的，均作为 Audio Gateway），然后让 BluetoothAudioGateway 接收 RFCOMM 连接，让 BluetoothHandsfree 接收 SCO 连接（这些操作都是为了让蓝牙耳机能主动连上 Android）。

```
if (action.equals(BluetoothIntent.ENABLED_ACTION)) {
    // SDP server may not be ready, so wait 3 seconds before
    // registering records.
    // TODO: Use a different mechanism to register SDP records,
    // that actually ACK's on success, so that we can retry rather
    // than hardcoding a 3 second guess.

    mHandler.sendMessageDelayed(mHandler.obtainMessage(REGISTER_SDP_
    RECORDS),3000);
    mAg.start(mIncomingConnectionHandler);
```

```
            mBtHandsfree.onBluetoothEnabled();
        }
```

BluetoothHeadsetService 收到 DISABLED_ACTION 时，会停止 BluetoothAudioGateway 和 BluetoothHandsfree。

```
        if (action.equals(BluetoothIntent.DISABLED_ACTION)) {
            mBtHandsfree.onBluetoothDisabled();
            mAg.stop();
        }
```

Android 跟蓝牙耳机建立连接有两种方式。

1. Android 主动跟蓝牙耳机连

BluetoothSettings 中和蓝牙耳机配对上之后，BluetoothHeadsetService 会收到 BONDING_CREATED_ACTION，这个时候 BluetoothHeadsetService 会主动去和蓝牙耳机建立 RFCOMM 连接。

```
        if (action.equals(BluetoothIntent.BONDING_CREATED_ACTION)) {
            if (mState == BluetoothHeadset.STATE_DISCONNECTED) {
                // Lets try and initiate an RFCOMM connection
                try {
                    mBinder.connectHeadset(address, null);
                } catch (RemoteException e) {}
            }
        }
```

RFCOMM 连接的真正实现是在 ConnectionThread 中，它分两步，第一步先通过 SDPClient 查询蓝牙设备时候支持 Headset 和 Handsfree profile。

```
        // 1) SDP query
        SDPClient client = SDPClient.getSDPClient(address);
        if (DBG) log("Connecting to SDP server (" + address + ")...");
        if (!client.connectSDPAsync()) {
            Log.e(TAG, "Failed to start SDP connection to " + address);
            mConnectingStatusHandler.obtainMessage(SDP_ERROR).sendToTarget();
            client.disconnectSDP();
            return;
        }
        if (isInterrupted()) {
            client.disconnectSDP();
            return;
        }
        if (!client.waitForSDPAsyncConnect(20000)) {    // 20 secs
            if (DBG) log("Failed to make SDP connection to " + address);
            mConnectingStatusHandler.obtainMessage(SDP_ERROR).sendToTarget();
            client.disconnectSDP();
            return;
        }
        if (DBG) log("SDP server connected (" + address + ")");
        int headsetChannel = client.isHeadset();
```

```
                if (DBG) log("headset channel = " + headsetChannel);
                int handsfreeChannel = client.isHandsfree();
                if (DBG) log("handsfree channel = " + handsfreeChannel);
                client.disconnectSDP();
```

第二步才是去真正建立 RFCOMM 连接。

```
                // 2) RFCOMM connect
                mHeadset = new HeadsetBase(mBluetooth, address, channel);
                if (isInterrupted()) {
                    return;
                }
                int result = mHeadset.waitForAsyncConnect(20000,    // 20 secs
                                                          mConnectedStatusHandler);
                if (DBG) log("Headset RFCOMM connection attempt took " +
                        (System.currentTimeMillis() - timestamp) + " ms");
                if (isInterrupted()) {
                    return;
                }
                if (result < 0) {
                    Log.e(TAG, "mHeadset.waitForAsyncConnect() error: " + result);

mConnectingStatusHandler.obtainMessage(RFCOMM_ERROR).sendToTarget();
                    return;
                } else if (result == 0) {
                    Log.e(TAG, "mHeadset.waitForAsyncConnect() error: " + result +
"(timeout)");

mConnectingStatusHandler.obtainMessage(RFCOMM_ERROR).sendToTarget();
                    return;
                } else {
                    if (DBG) log("mHeadset.waitForAsyncConnect() success");

mConnectingStatusHandler.obtainMessage(RFCOMM_CONNECTED).sendToTarget();
                }
```

当 RFCOMM 连接成功建立后，BluetoothHeadsetDevice 会收到 RFCOMM_CONNECTED 消息，它会调用 BluetoothHandsfree 来建立 SCO 连接，广播通知 Headset 状态变化的 Intent（PhoneApp 和 BluetoothSettings 会接收这个 Intent）。

```
                case RFCOMM_CONNECTED:
                    // success
                    if (DBG) log("Rfcomm connected");
                    if (mConnectThread != null) {
                        try {
                            mConnectThread.join();
                        } catch (InterruptedException e) {
                            Log.w(TAG, "Connect attempt cancelled, ignoring
```

```
RFCOMM_CONNECTED", e);
                        return;
                    }
                    mConnectThread = null;
                }
                setState(BluetoothHeadset.STATE_CONNECTED,
BluetoothHeadset.RESULT_SUCCESS);

                mBtHandsfree.connectHeadset(mHeadset, mHeadsetType);
                break;
```

BluetoothHandsfree 会先做一些初始化工作，比如根据是 Headset 还是 Handsfree 初始化不同的 ATParser，并且启动一个接收线程从已建立的 RFCOMM 上接收蓝牙耳机过来的控制命令（也就是 AT 命令），接着判断如果是在打电话过程中，才去建立 SCO 连接来打通数据通道。

```
/* package */ void connectHeadset(HeadsetBase headset, int headsetType) {
    mHeadset = headset;
    mHeadsetType = headsetType;
    if (mHeadsetType == TYPE_HEADSET) {
        initializeHeadsetAtParser();
    } else {
        initializeHandsfreeAtParser();
    }
    headset.startEventThread();
    configAudioParameters();

    if (inDebug()) {
        startDebug();
    }

    if (isIncallAudio()) {
        audioOn();
    }
}
```

建立 SCO 连接是通过 SCOSocket 实现的

```
/** Request to establish SCO (audio) connection to bluetooth
  * headset/handsfree, if one is connected. Does not block.
  * Returns false if the user has requested audio off, or if there
  * is some other immediate problem that will prevent BT audio.
  */
/* package */ synchronized boolean audioOn() {

    mOutgoingSco = createScoSocket();
    if (!mOutgoingSco.connect(mHeadset.getAddress())) {
        mOutgoingSco = null;
```

```
        }

        return true;
    }
```

当 SCO 连接成功建立后，BluetoothHandsfree 会收到 SCO_CONNECTED 消息，它就会去调用 AudioManager 的 setBluetoothScoOn 函数，从而通知音频系统有个蓝牙耳机可用了。到此，Android 完成了和蓝牙耳机的全部连接。

```
            case SCO_CONNECTED:
                if (msg.arg1 == ScoSocket.STATE_CONNECTED && isHeadsetConnected()
&&
                        mConnectedSco == null) {
                    if (DBG) log("Routing audio for outgoing SCO conection");
                    mConnectedSco = (ScoSocket)msg.obj;
                    mAudioManager.setBluetoothScoOn(true);
                } else if (msg.arg1 == ScoSocket.STATE_CONNECTED) {
                    if (DBG) log("Rejecting new connected outgoing SCO socket");
                    ((ScoSocket)msg.obj).close();
                    mOutgoingSco.close();
                }
                mOutgoingSco = null;
                break;
```

2. 蓝牙耳机主动跟 Android 连

首先 BluetoothAudioGateway 会在一个线程中收到来自蓝牙耳机的 RFCOMM 连接，然后发送消息给 BluetoothHeadsetService。

```
                    mConnectingHeadsetRfcommChannel = -1;
                    mConnectingHandsfreeRfcommChannel = -1;
                    if
(waitForHandsfreeConnectNative(SELECT_WAIT_TIMEOUT) == false) {
                        if (mTimeoutRemainingMs > 0) {
                            try {
                                Log.i(tag, "select thread timed out, but " +
                                        mTimeoutRemainingMs + "ms of
waiting remain.");
                                Thread.sleep(mTimeoutRemainingMs);
                            } catch (InterruptedException e) {
                                Log.i(tag, "select thread was interrupted (2),
exiting");
                                mInterrupted = true;
                            }
                        }
                    }
```

BluetoothHeadsetService 会根据当前的状态来处理消息，分 3 种情况，第一是当前状态是非连接状态，会发送 RFCOMM_CONNECTED 消息，后续处理请参见前面的分析。

```
            case BluetoothHeadset.STATE_DISCONNECTED:
```

```
                    // headset connecting us, lets join
                    setState(BluetoothHeadset.STATE_CONNECTING);
                    mHeadsetAddress = info.mAddress;
                    mHeadset = new HeadsetBase(mBluetooth, mHeadsetAddress,
info.mSocketFd,
                                                        info.mRfcommChan,
mConnectedStatusHandler);
                    mHeadsetType = type;


mConnectingStatusHandler.obtainMessage(RFCOMM_CONNECTED).sendToTarget();

                    break;
```

如果当前是正在连接状态，则先停掉已经存在的 ConnectThread，并直接调用 BluetoothHandsfree 去建立 SCO 连接。

```
                case BluetoothHeadset.STATE_CONNECTING:
                    // If we are here, we are in danger of a race condition
                    // incoming rfcomm connection, but we are also attempting an
                    // outgoing connection. Lets try and interrupt the outgoing
                    // connection.
                    mConnectThread.interrupt();

                    // Now continue with new connection, including calling callback
                    mHeadset = new HeadsetBase(mBluetooth, mHeadsetAddress,
info.mSocketFd,
                                                        info.mRfcommChan,
mConnectedStatusHandler);
                    mHeadsetType = type;

                    setState(BluetoothHeadset.STATE_CONNECTED,
BluetoothHeadset.RESULT_SUCCESS);
                    mBtHandsfree.connectHeadset(mHeadset, mHeadsetType);

                    // Make sure that old outgoing connect thread is dead.
                    break;
```

如果当前是已连接的状态，这种情况是一种错误 case，所以直接断掉所有连接。

```
                case BluetoothHeadset.STATE_CONNECTED:
                    if (DBG) log("Already connected to " + mHeadsetAddress + ", disconnecting
" +
                            info.mAddress);
                    mBluetooth.disconnectRemoteDeviceAcl(info.mAddress);
                    break;
```

蓝牙耳机也可能会主动发起 SCO 连接，BluetoothHandsfree 会接收到一个 SCO_ACCEPTED 消息，它会去调用 AudioManager 的 setBluetoothScoOn 函数，从而通知音

频系统有个蓝牙耳机可用了。到此，蓝牙耳机完成了和 Android 的全部连接。

```
                case SCO_ACCEPTED:
                    if (msg.arg1 == ScoSocket.STATE_CONNECTED) {
                        if (isHeadsetConnected() && mAudioPossible && mConnectedSco ==
null) {
                            Log.i(TAG, "Routing audio for incoming SCO connection");
                            mConnectedSco = (ScoSocket)msg.obj;
                            mAudioManager.setBluetoothScoOn(true);
                        } else {
                            Log.i(TAG, "Rejecting incoming SCO connection");
                            ((ScoSocket)msg.obj).close();
                        }
                    } // else error trying to accept, try again
                    mIncomingSco = createScoSocket();
                    mIncomingSco.accept();
                    break;
```