1. app进程

1.1使用WifiManager

WifiManager工具类：
android包中自带有WifiManager工具类，专门用于wifi管理：

```
import android.net.wifi.WifiManager;
```

取得WifiManager对象：

```
WifiManager mWifiManager = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
```

打开WIFI：

```
if (!mWifiManager.isWifiEnabled()) {
    mWifiManager.setWifiEnabled(true);
}
```

创建一个WIFI信息：

```
public WifiConfiguration CreateWifiInfo(String SSID, String Password,
            int Type) {
        WifiConfiguration config = new WifiConfiguration();
        config.allowedAuthAlgorithms.clear();
        config.allowedGroupCiphers.clear();
        config.allowedKeyManagement.clear();
        config.allowedPairwiseCiphers.clear();
        config.allowedProtocols.clear();
        config.SSID = "\"" + SSID + "\"";

        WifiConfiguration tempConfig = this.IsExsits(SSID);
```

```
        if (tempConfig != null) {
            mWifiManager.removeNetwork(tempConfig.networkId);
        }

        if (Type == 1) // WIFICIPHER_NOPASS
        {
            config.wepKeys[0] = "";
            config.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
            config.wepTxKeyIndex = 0;
        }
        if (Type == 2) // WIFICIPHER_WEP
        {
            config.hiddenSSID = true;
            config.wepKeys[0] = "\"" + Password + "\"";
            config.allowedAuthAlgorithms
                    .set(WifiConfiguration.AuthAlgorithm.SHARED);
            config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
            config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
            config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP40);
            config.allowedGroupCiphers
                    .set(WifiConfiguration.GroupCipher.WEP104);
            config.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
            config.wepTxKeyIndex = 0;
        }
        if (Type == 3) // WIFICIPHER_WPA
        {
            config.preSharedKey = "\"" + Password + "\"";
            config.hiddenSSID = true;
            config.allowedAuthAlgorithms
                    .set(WifiConfiguration.AuthAlgorithm.OPEN);
            config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
            config.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
            config.allowedPairwiseCiphers
```

```
                    .set(WifiConfiguration.PairwiseCipher.TKIP);
            // config.allowedProtocols.set(WifiConfiguration.Protocol.WPA);
            config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
            config.allowedPairwiseCiphers
                    .set(WifiConfiguration.PairwiseCipher.CCMP);
            config.status = WifiConfiguration.Status.ENABLED;
        }
        return config;
    }
```

这里只介绍第三个参数：Type。从代码中可以看出，Type有三个值，分别为1，2，3。WIFI热点是有加密的，加密方式包括：不加密，WEP，WPA三种，1、2、3就分别对应这三种加密方式，方法返回一个WIFI热点信息。

添加一个网络并连接：

```
public void addNetwork(WifiConfiguration wcg) {
        int wcgID = mWifiManager.addNetwork(wcg);
        boolean b = mWifiManager.enableNetwork(wcgID, true);
        System.out.println("a--" + wcgID);
        System.out.println("b--" + b);
    }
```

具体的使用方式如下：

```
mWifiAdmin.addNetwork(mWifiAdmin.CreateWifiInfo(SSID, password, 3));
```

判断WIFI是否连接成功：

```
public int isWifiContected(Context context) {
        ConnectivityManager connectivityManager = (ConnectivityManager) context
                .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo wifiNetworkInfo = connectivityManager
                .getNetworkInfo(ConnectivityManager.TYPE_WIFI);

        Log.v(TAG, "isConnectedOrConnecting = " + wifiNetworkInfo.isConnectedOrConnecting());
        Log.d(TAG, "wifiNetworkInfo.getDetailedState() = " + wifiNetworkInfo.getDetailedState());
        if (wifiNetworkInfo.getDetailedState() == DetailedState.OBTAINING_IPADDR
```

```
                || wifiNetworkInfo.getDetailedState() == DetailedState.CONNECTING) {
            return WIFI_CONNECTING;
        } else if (wifiNetworkInfo.getDetailedState() == DetailedState.CONNECTED) {
            return WIFI_CONNECTED;
        } else {
            Log.d(TAG, "getDetailedState() == " + wifiNetworkInfo.getDetailedState());
            return WIFI_CONNECT_FAILED;
        }
    }
```

关闭WIFI连接：

```
if (mWifiManager.isWifiEnabled()) {
    mWifiManager.setWifiEnabled(false);
}
```

开始WIFI扫描

```
mWifiManager.startScan()
```

获取WIFI扫描结果

```
mWifiManager.getWifiState()
```

获取当前已连接的wifi信息

```
WifiInfo info = mWifiManager.getConnectionInfo();
```

几乎和WifiConfiguration一样的，但是如果当前没有连接wifi的话，就会返回Null，它包括了SSID、networkId、BSSID的，切记一个问题：它的SSID是带双引号的，这点和ScanResult对象不一样

在高版本的安卓，普通APP无权限通过

```
mWifiManager.setWifiEnabled(true/false)
```

来开启关闭wifi

也无权限在wifi开启的状态通过如下连接网络

```
WifiConfiguration config
config.xxx=
int networkid = mWifiManager.addNetwork(config);
mWifiManager.enableNetwork(networkid, true);

普通APP应该在通过使用intent打开Settings.Panel.ACTION_WIFI activity,
Intent panelIntent = new Intent(android.provider.Settings.Panel.ACTION_WIFI);
startActivityForResult(panelIntent,1);
参考https://developer.android.google.cn/about/versions/10/features#settings-panels
然后待用户在Settings.Panel.ACTION_WIFI activity中来打开关闭连接wifi
```

Activity Service BroadcastReceiver

一个BroadcastReceiver 可注册多个intent action，action对应的每种intent 里面具有不同的extra data

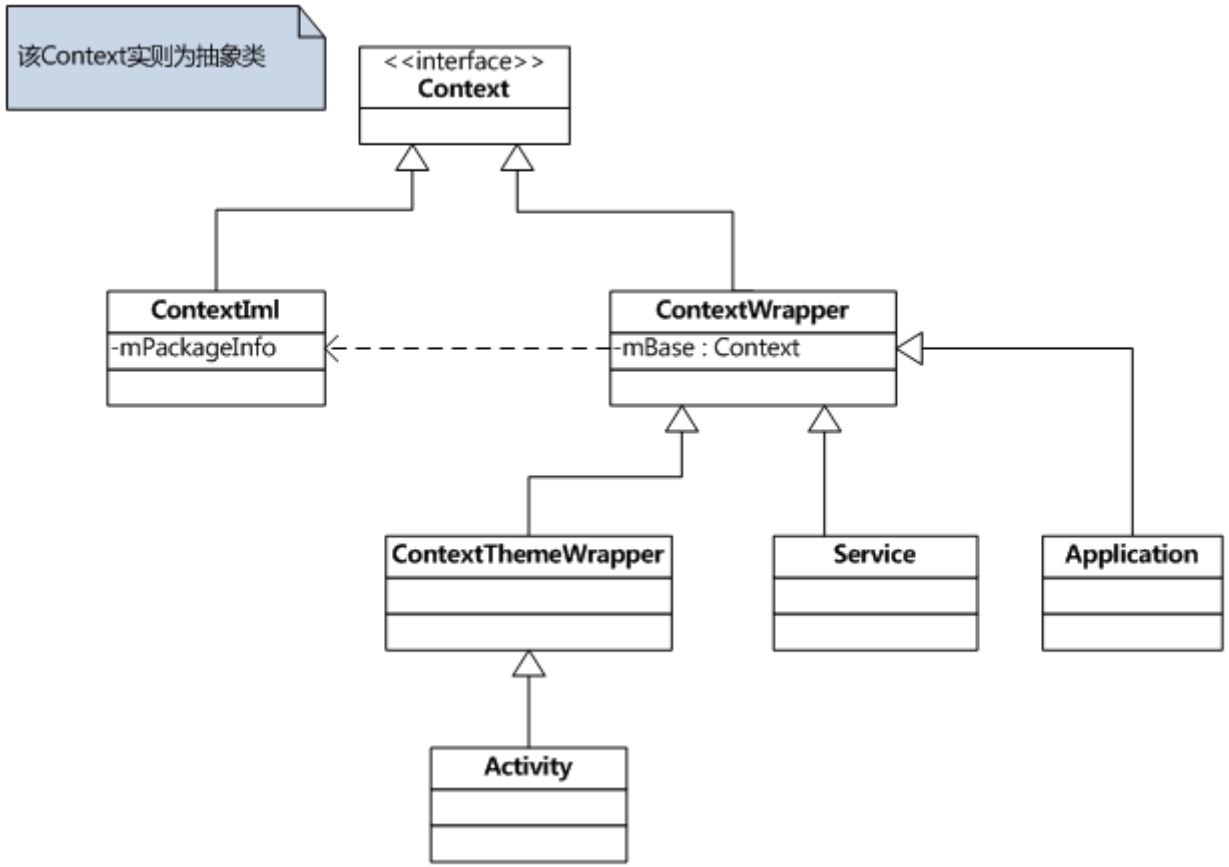
WifiManager.WIFI_STATE_CHANGED_ACTION

WifiManager.NETWORK_STATE_CHANGED_ACTION

WifiManager.SUPPLICANT_STATE_CHANGED_ACTION

ConnectivityManager.CONNECTIVITY_ACTION

2.获取IWifiManager的AIDL proxy端接口



APP 进程创建完成后进程中有一个Application对象，若干个Activity和Service对象。他们都直接或间接派生自ContextWrapper类。

ContextWrapper类和ContextIml类都实现了Context接口。在ContextWrapper对象(Application对象、Activity和Service对象都是ContextWrapper对象)中有一个ContextIml对象。调用ContextWrapper对象Context接口中的各种方法时，这些方法内部就调用ContextWrapper对象中的ContextIml对象相应的Context接口。

ContextWrapper对象与ContextIml对象是一对一还是多对一还是多对多还需后续看代码。目前还不确定

ContextIml类中使用SystemServiceRegistry类，SystemServiceRegistry类在静态初始化块中调用了WifiFrameworkInitializer.registerServiceWrappers():

frameworks/base/core/java/android/app/SystemServiceRegistry.java

```
public final class SystemServiceRegistry {
...
static {
...
WifiFrameworkInitializer.registerServiceWrappers();
...
}
...
}
```

在WifiFrameworkInitializer.registerServiceWrappers方法中调用SystemServiceRegistry.registerContextAwareService将new出来的各种WifiXXXManager对象注册到SystemServiceRegistry类的哈希表中，new各种WifiXXXManager对象时需要传入一个对应的IWifiXXXManager对象，即binder通信的 proxy 对象：

frameworks/base/wifi/java/android/net/wifi/WifiFrameworkInitializer.java

```
71      public static void registerServiceWrappers() {
72          SystemServiceRegistry.registerContextAwareService(
73                  Context.WIFI_SERVICE,
74                  WifiManager.class,
75                  (context, serviceBinder) -> {
76                      IWifiManager service = IWifiManager.Stub.asInterface(serviceBinder);
77                      return new WifiManager(context, service, getInstanceLooper());
78                  }
79          );
80          SystemServiceRegistry.registerStaticService(
81                  Context.WIFI_P2P_SERVICE,
82                  WifiP2pManager.class,
83                  serviceBinder -> {
84                      IWifiP2pManager service = IWifiP2pManager.Stub.asInterface(serviceBinder);
85                      return new WifiP2pManager(service);
86                  }
87          );
88          SystemServiceRegistry.registerContextAwareService(
89                  Context.WIFI_AWARE_SERVICE,
90                  WifiAwareManager.class,
91                  (context, serviceBinder) -> {
92                      IWifiAwareManager service = IWifiAwareManager.Stub.asInterface(serviceBinder);
93                      return new WifiAwareManager(context, service);
94                  }
95          );
96          SystemServiceRegistry.registerContextAwareService(
97                  Context.WIFI_SCANNING_SERVICE,
98                  WifiScanner.class,
99                  (context, serviceBinder) -> {
100                     IWifiScanner service = IWifiScanner.Stub.asInterface(serviceBinder);
101                     return new WifiScanner(context, service, getInstanceLooper());
102                 }
103         );
104         SystemServiceRegistry.registerContextAwareService(
105                 Context.WIFI_RTT_RANGING_SERVICE,
106                 WifiRttManager.class,
107                 (context, serviceBinder) -> {
108                     IWifiRttManager service = IWifiRttManager.Stub.asInterface(serviceBinder);
109                     return new WifiRttManager(context, service);
110                 }
```

```
111             );
112             SystemServiceRegistry.registerContextAwareService(
113                     Context.WIFI_RTT_SERVICE,
```

ContextWrapper对象的getSystemService方法直接调用其内部ContextIml对象的getSystemService方法，参考

frameworks/base/core/java/android/app/ContextImpl.java

```
1901    @Override
1902    public Object getSystemService(String name) {
1903        if (vmIncorrectContextUseEnabled()) {
1904            // We may override this API from outer context.
1905            final boolean isUiContext = isUiContext() || isOuterUiContext();
1906            // Check incorrect Context usage.
1907            if (isUiComponent(name) && !isUiContext) {
1908                final String errorMessage = "Tried to access visual service "
1909                        + SystemServiceRegistry.getSystemServiceClassName(name)
1910                        + " from a non-visual Context:" + getOuterContext();
1911                final String message = "Visual services, such as WindowManager, WallpaperService "
1912                        + "or LayoutInflater should be accessed from Activity or other visual "
1913                        + "Context. Use an Activity or a Context created with "
1914                        + "Context#createWindowContext(int, Bundle), which are adjusted to "
1915                        + "the configuration and visual bounds of an area on screen.";
1916                final Exception exception = new IllegalAccessException(errorMessage);
1917                StrictMode.onIncorrectContextUsed(message, exception);
1918                Log.e(TAG, errorMessage + " " + message, exception);
1919            }
1920        }
1921        return SystemServiceRegistry.getSystemService(this, name);
1922    }
```

ContextIml对象的getSystemService中调用了SystemServiceRegistry类的getSystemService方法将先前注册到哈希表中的WifiXXXManager对象取出，之后使用WifiXXXManager对象就能进行wifi相关的操作。

取出的WifiXXXManager对象是其内部IWifiXXXManager类的wrapper，调用WifiXXXManager的各种方法最终都会调用到其内部IWifiXXXManager类的方法，而IWifiXXXManager类是AIDL的proxy端，最终通过binder通信调到framework中具体负责实现的服务。

```
72          SystemServiceRegistry.registerContextAwareService(
73                  Context.WIFI_SERVICE,
74                  WifiManager.class,
75                  (context, serviceBinder) -> {
76                      IWifiManager service = IWifiManager.Stub.asInterface(serviceBinder);
77                      return new WifiManager(context, service, getInstanceLooper());
78                  }
79          );
80          SystemServiceRegistry.registerStaticService(
81                  Context.WIFI_P2P_SERVICE,
82                  WifiP2pManager.class,
83                  serviceBinder -> {
84                      IWifiP2pManager service = IWifiP2pManager.Stub.asInterface(serviceBinder);
85                      return new WifiP2pManager(service);
86                  }
87          );
88          SystemServiceRegistry.registerContextAwareService(
89                  Context.WIFI_AWARE_SERVICE,
90                  WifiAwareManager.class,
91                  (context, serviceBinder) -> {
92                      IWifiAwareManager service = IWifiAwareManager.Stub.asInterface(serviceBinder);
```

```
93                      return new WifiAwareManager(context, service);
94                  }
95          );
96          SystemServiceRegistry.registerContextAwareService(
97                  Context.WIFI_SCANNING_SERVICE,
98                  WifiScanner.class,
99                  (context, serviceBinder) -> {
100                     IWifiScanner service = IWifiScanner.Stub.asInterface(serviceBinder);
101                     return new WifiScanner(context, service, getInstanceLooper());
102                 }
103         );
104         SystemServiceRegistry.registerContextAwareService(
105                 Context.WIFI_RTT_RANGING_SERVICE,
106                 WifiRttManager.class,
107                 (context, serviceBinder) -> {
108                     IWifiRttManager service = IWifiRttManager.Stub.asInterface(serviceBinder);
109                     return new WifiRttManager(context, service);
110                 }
111         );
112         SystemServiceRegistry.registerContextAwareService(
113                 Context.WIFI_RTT_SERVICE,
114                 RttManager.class,
115                 context -> {
116                     WifiRttManager wifiRttManager = context.getSystemService(WifiRttManager.class);
117                     return new RttManager(context, wifiRttManager);
118                 }
119         );
```

RttManager使用wifiRttManager，wifiRttManager使用IWifiRttManager，IWifiRttManager是binder proxy端

```
wifi涉及的主要AIDL接口
IWifiManager
IWifiP2pManager
IWifiAwareManager
IWifiRttManager
```

2. framework层服务进程

WifiService服务注册

system_server调用startOtherServices创建AIDL服务时会创建WifiService服务对象并注册到systemManager。

```
private void startOtherServices(@NonNull TimingsTraceAndSlog t) {
......
248      private static final String WIFI_SERVICE_CLASS =
249              "com.android.server.wifi.WifiService";
......
1182             if (!isWatch) {
1183                 t.traceBegin("StartConsumerIrService");
1184                 consumerIr = new ConsumerIrService(context);
1185                 ServiceManager.addService(Context.CONSUMER_IR_SERVICE, consumerIr);
1186                 t.traceEnd();
1187             }
......
1535             if (context.getPackageManager().hasSystemFeature(
1536                     PackageManager.FEATURE_WIFI)) {
1537                 // Wifi Service must be started first for wifi-related services.
1538                 t.traceBegin("StartWifi");
1539                 mSystemServiceManager.startService(WIFI_SERVICE_CLASS);
1540                 t.traceEnd();
```

```
1541                t.traceBegin("StartWifiScanning");
1542                mSystemServiceManager.startService(
1543                        "com.android.server.wifi.scanner.WifiScanningService");
1544                t.traceEnd();
1545            }
......
}
```

***WifiService*对象在构造时首先分别*new*出来*WifiContext*、*WifiInjector*、*WifiAsyncChannel*对象，然后把三个对象作为参数*new* 了一个*WifiServiceImpl*对象**，其实现如下：

frameworks/opt/net/wifi/service/java/com/android/server/wifi/WifiService.java

```
34  public final class WifiService extends SystemService {
35
36      private static final String TAG = "WifiService";
37      // Notification channels used by the wifi service.
38      public static final String NOTIFICATION_NETWORK_STATUS = "NETWORK_STATUS";
39      public static final String NOTIFICATION_NETWORK_ALERTS = "NETWORK_ALERTS";
40      public static final String NOTIFICATION_NETWORK_AVAILABLE = "NETWORK_AVAILABLE";
41
42      private final WifiServiceImpl mImpl;
43      private final WifiContext mWifiContext;
44
45      public WifiService(Context contextBase) {
46          super(contextBase);
47          mWifiContext = new WifiContext(contextBase);
48          WifiInjector injector = new WifiInjector(mWifiContext);
49          WifiAsyncChannel channel =  new WifiAsyncChannel(TAG);
50          mImpl = new WifiServiceImpl(mWifiContext, injector, channel);
51      }
```

```
        ...
}
```

WifiInjector对象构造

new 一个SupplicantStaIfaceHal对象，然后将其作为参数new一个WifiNative对象。

```
public class WifiInjector {

99      private final WifiNative mWifiNative;

103      private final SupplicantStaIfaceHal mSupplicantStaIfaceHal;

172      public WifiInjector(WifiContext context) {

232          mSupplicantStaIfaceHal = new SupplicantStaIfaceHal(
233                  mContext, mWifiMonitor, mFrameworkFacade, wifiHandler, mClock, mWifiMetrics);
234          mHostapdHal = new HostapdHal(mContext, wifiHandler);
235          mWifiCondManager = (WifiNl80211Manager) mContext.getSystemService(
236                  Context.WIFI_NL80211_SERVICE);
237          mWifiNative = new WifiNative(
238                  mWifiVendorHal, mSupplicantStaIfaceHal, mHostapdHal, mWifiCondManager,
239                  mWifiMonitor, mPropertyService, mWifiMetrics,
240                  wifiHandler, new Random(), this);

389      }

}
```

WifiNative对象

构造时将传入的SupplicantStaIfaceHal对象赋给内部成员变量mSupplicantStaIfaceHal。

xref: /w600/src/mt8185_sdk/frameworks/opt/net/wifi/service/java/com/android/server/wifi/WifiNative.java

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download Se

```
94      public WifiNative(WifiVendorHal vendorHal,
95                        SupplicantStaIfaceHal staIfaceHal, HostapdHal hostapdHal,
96                        WifiNl80211Manager condManager, WifiMonitor wifiMonitor,
97                        PropertyService propertyService, WifiMetrics wifiMetrics,
98                        Handler handler, Random random,
99                        WifiInjector wifiInjector) {
100         mWifiVendorHal = vendorHal;
101         mSupplicantStaIfaceHal = staIfaceHal;
102         mHostapdHal = hostapdHal;
103         mWifiCondManager = condManager;
104         mWifiMonitor = wifiMonitor;
105         mPropertyService = propertyService;
106         mWifiMetrics = wifiMetrics;
107         mHandler = handler;
108         mRandom = random;
109         mWifiInjector = wifiInjector;
110     }
```

WifiNative的setupInterfaceForClientInScanMode方法

frameworks/opt/net/wifi/service/java/com/android/server/wifi/WifiNative.java

```
1086        /**
1087         * Setup an interface for client mode (for scan) operations.
1088         * M: We change AOSP flow and start supplicant under scan mode
1089         * for information sync from supplicant HAL
1090         *
1091         * This method configures an interface in STA mode in the native daemons
1092         * (wificond, vendor HAL).
1093         *
1094         * @param interfaceCallback Associated callback for notifying status changes for the iface.
1095         * @return Returns the name of the allocated interface, will be null on failure.
1096         */
1097        public String setupInterfaceForClientInScanMode(
1098                @NonNull InterfaceCallback interfaceCallback) {
1099            synchronized (mLock) {
1100                if (!startHal()) {
1101                    Log.e(TAG, "Failed to start Hal");
1102                    mWifiMetrics.incrementNumSetupClientInterfaceFailureDueToHal();
1103                    return null;
1104                }
1105                if (!startSupplicant()) {
1106                    Log.e(TAG, "Failed to start supplicant");
1107                    mWifiMetrics.incrementNumSetupClientInterfaceFailureDueToSupplicant();
1108                    return null;
1109                }
1110                Iface iface = mIfaceMgr.allocateIface(Iface.IFACE_TYPE_STA_FOR_SCAN);
1111                if (iface == null) {
1112                    Log.e(TAG, "Failed to allocate new STA iface");
1113                    return null;
1114                }
1115                iface.externalListener = interfaceCallback;
1116                iface.name = createStaIface(iface);
1117                if (TextUtils.isEmpty(iface.name)) {
1118                    Log.e(TAG, "Failed to create iface in vendor HAL");
1119                    mIfaceMgr.removeIface(iface.id);
1120                    mWifiMetrics.incrementNumSetupClientInterfaceFailureDueToHal();
1121                    return null;
1122                }
1123                if (!mWifiCondManager.setupInterfaceForClientMode(iface.name, Runnable::run,
1124                        new NormalScanEventCallback(iface.name),
1125                        new PnoScanEventCallback(iface.name))) {
```

```
1126                    Log.e(TAG, "Failed to setup iface in wificond=" + iface.name);
1127                    teardownInterface(iface.name);
1128                    mWifiMetrics.incrementNumSetupClientInterfaceFailureDueToWificond();
1129                    return null;
1130                }
1131                if (!mSupplicantStaIfaceHal.setupIface(iface.name)) {
1132                    Log.e(TAG, "Failed to setup iface in supplicant on " + iface);
1133                    teardownInterface(iface.name);
1134                    mWifiMetrics.incrementNumSetupClientInterfaceFailureDueToSupplicant();
1135                    return null;
1136                }
1137                iface.networkObserver = new NetworkObserverInternal(iface.id);
1138                if (!registerNetworkObserver(iface.networkObserver)) {
1139                    Log.e(TAG, "Failed to register network observer for iface=" + iface.name);
1140                    teardownInterface(iface.name);
1141                    return null;
1142                }
1143                mWifiMonitor.startMonitoring(iface.name);
1144                // Just to avoid any race conditions with interface state change callbacks,
1145                // update the interface state before we exit.
1146                onInterfaceStateChanged(iface, isInterfaceUp(iface.name));
1147                Log.i(TAG, "Successfully setup " + iface);
1148
1149                iface.featureSet = getSupportedFeatureSetInternal(iface.name);
1150                return iface.name;
1151            }
1152        }
```

WifiNative的startHal方法

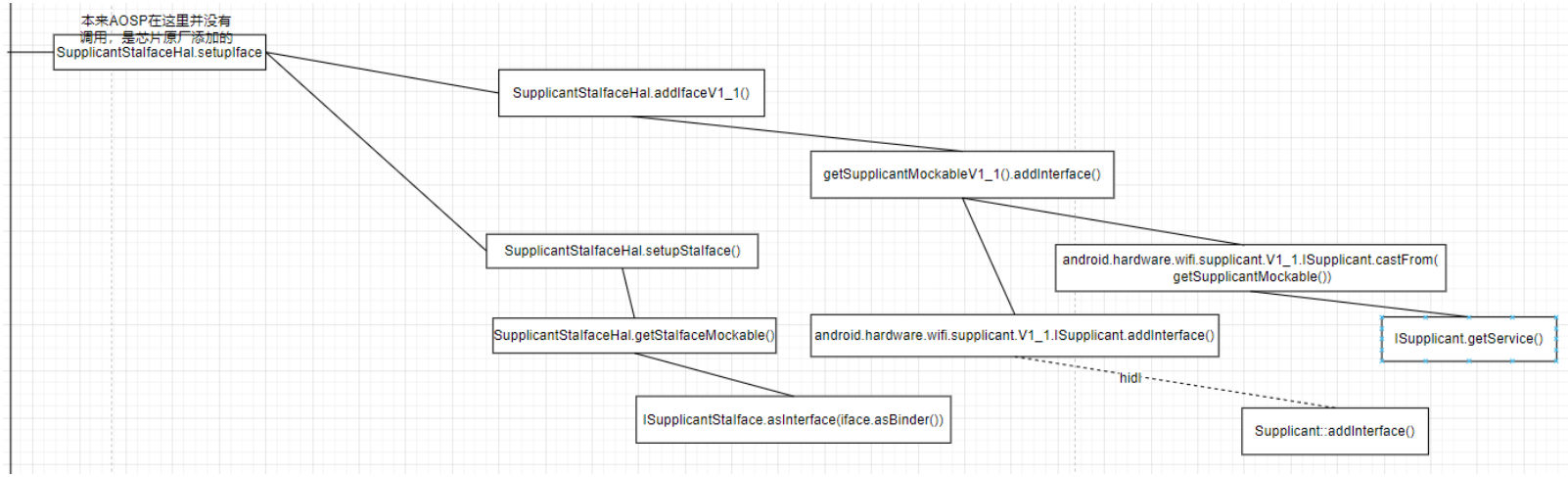
WifiNative的startSupplicant方法

SupplicantStaIfaceHal的setupIface方法

frameworks/opt/net/wifi/service/java/com/android/server/wifi/SupplicantStaIfaceHal.java

```
412     /**
413      * Setup a STA interface for the specified iface name.
414      *
415      * @param ifaceName Name of the interface.
416      * @return true on success, false otherwise.
417      */
418     public boolean setupIface(@NonNull String ifaceName) {
419         final String methodStr = "setupIface";
420         if (checkSupplicantStaIfaceAndLogFailure(ifaceName, methodStr) != null) return false;
421         ISupplicantIface ifaceHwBinder;
422
423         if (isV1_1()) {
424             ifaceHwBinder = addIfaceV1_1(ifaceName);
425         } else {
426             ifaceHwBinder = getIfaceV1_0(ifaceName);
427         }
428         if (ifaceHwBinder == null) {
429             Log.e(TAG, "setupIface got null iface");
430             return false;
431         }
432
433         try {
434             ISupplicantStaIface iface = setupStaIface(ifaceName, ifaceHwBinder);
435             mISupplicantStaIfaces.put(ifaceName, iface);
436             com.mediatek.server.wifi.MtkWapi.setupMtkIface(ifaceName);
437             com.mediatek.server.wifi.MtkSupplicantStaIfaceHal.setupMtkIface(ifaceName);
438         } catch (RemoteException e) {
439             loge("setup StaIface failed: " + e.toString());
440             return false;
441         }
442
443         return true;
444     }
```

SupplicantStaIfaceHal的setupIface方法



WifiServiceImpl对象

frameworks/opt/net/wifi/service/java/com/android/server/wifi/WifiServiceImpl.java

```
public class WifiServiceImpl extends BaseWifiService {
...
294      public WifiServiceImpl(Context context, WifiInjector wifiInjector, AsyncChannel asyncChannel) {
295          mContext = context;
296          mWifiInjector = wifiInjector;
297          mClock = wifiInjector.getClock();
298
299          mFacade = mWifiInjector.getFrameworkFacade();
300          mWifiMetrics = mWifiInjector.getWifiMetrics();
301          mWifiTrafficPoller = mWifiInjector.getWifiTrafficPoller();
302          mUserManager = mWifiInjector.getUserManager();
```

```
303         mCountryCode = mWifiInjector.getWifiCountryCode();
304         mClientModeImpl = mWifiInjector.getClientModeImpl();
305         mActiveModeWarden = mWifiInjector.getActiveModeWarden();
306         mScanRequestProxy = mWifiInjector.getScanRequestProxy();
307         mSettingsStore = mWifiInjector.getWifiSettingsStore();
308         mPowerManager = mContext.getSystemService(PowerManager.class);
309         mAppOps = (AppOpsManager) mContext.getSystemService(Context.APP_OPS_SERVICE);
310         mWifiLockManager = mWifiInjector.getWifiLockManager();
311         mWifiMulticastLockManager = mWifiInjector.getWifiMulticastLockManager();
312         mClientModeImplHandler = new ClientModeImplHandler(TAG,
313                 mWifiInjector.getAsyncChannelHandlerThread().getLooper(), asyncChannel);
314         mWifiBackupRestore = mWifiInjector.getWifiBackupRestore();
315         mSoftApBackupRestore = mWifiInjector.getSoftApBackupRestore();
316         mWifiApConfigStore = mWifiInjector.getWifiApConfigStore();
317         mWifiPermissionsUtil = mWifiInjector.getWifiPermissionsUtil();
318         mLog = mWifiInjector.makeLog(TAG);
319         mFrameworkFacade = wifiInjector.getFrameworkFacade();
320         mTetheredSoftApTracker = new TetheredSoftApTracker();
321         mActiveModeWarden.registerSoftApCallback(mTetheredSoftApTracker);
322         mLohsSoftApTracker = new LohsSoftApTracker();
323         mActiveModeWarden.registerLohsCallback(mLohsSoftApTracker);
324         mWifiNetworkSuggestionsManager = mWifiInjector.getWifiNetworkSuggestionsManager();
325         mDppManager = mWifiInjector.getDppManager();
326         mWifiThreadRunner = mWifiInjector.getWifiThreadRunner();
327         mWifiConfigManager = mWifiInjector.getWifiConfigManager();
328         mPasspointManager = mWifiInjector.getPasspointManager();
329         mWifiScoreCard = mWifiInjector.getWifiScoreCard();
330         mMemoryStoreImpl = new MemoryStoreImpl(mContext, mWifiInjector,
331                 mWifiScoreCard,  mWifiInjector.getWifiHealthMonitor());
332     }
...
}
```

frameworks/opt/net/wifi/service/java/com/android/server/wifi/BaseWifiService.java

```
public class BaseWifiService extends IWifiManager.Stub {

74      public long getSupportedFeatures() {
75          throw new UnsupportedOperationException();
76      }
...
687      public void stopReconnectAndScan(int index, int period) {
688          throw new UnsupportedOperationException();
689      }
690
}
```

WifiService 中new出来的 WifiServiceImpl 对象是IWifiManager AIDL接口的实现端 。WifiServiceImpl派生自BaseWifiService，BaseWifiService派生自IWifiManager.Stub，在BaseWifiService类中只负责调用各个接口后抛出异常，WifiServiceImpl继承并重写这些方法后才有真正的实现。

从WifiServiceImpl的构造函数来看WifiServiceImpl其实最终通过构造时传进来的WifiInjector干活。

frameworks/opt/net/wifi/service/java/com/android/server/wifi/WifiInjector.java

```
75  public class WifiInjector {
...
```

```
172     public WifiInjector(WifiContext context) {
...
232         mSupplicantStaIfaceHal = new SupplicantStaIfaceHal(
233                 mContext, mWifiMonitor, mFrameworkFacade, wifiHandler, mClock, mWifiMetrics);
234         mHostapdHal = new HostapdHal(mContext, wifiHandler);
235         mWifiCondManager = (WifiNl80211Manager) mContext.getSystemService(
236                 Context.WIFI_NL80211_SERVICE);
237         mWifiNative = new WifiNative(
238                 mWifiVendorHal, mSupplicantStaIfaceHal, mHostapdHal, mWifiCondManager,
239                 mWifiMonitor, mPropertyService, mWifiMetrics,
240                 wifiHandler, new Random(), this);
...
287         mWifiConnectivityHelper = new WifiConnectivityHelper(mWifiNative);
...


}
```

在WifiInjector的构造函数中new 了SupplicantStaIfaceHal对象和HostapdHal对象，并将其作为构造参数来new WifiNative对象，然后使用WifiNative对象作为构造参数new出其他各个wfi管理对象：WifiConnectivityHelper、WifiNetworkSelector、PasspointManager、SarManager、WifiDiagnostics、LinkProbeManager、MboOceController、WifiHealthMonitor、ClientModeImpl、ActiveModeWarden、WifiLockManager、DppManager，之后WifiInjector对象的很多方法在WifiServiceImpl中被调用时就在方法中使用各种wifi管理对象，wifi管理对象最终还是通过WifiNative对象使用来使用SupplicantStaIfaceHal对象来跟底层wpa_supplicant交互。

在WifiNative使用SupplicantStaIfaceHal对象将wifi设置成连接模式时调用到了mSupplicantStaIfaceHal的setupIface方法:

frameworks/opt/net/wifi/service/java/com/android/server/wifi/WifiNative.java

```
public class WifiNative {
...
1097      public String setupInterfaceForClientInScanMode(
...
1131              if (!mSupplicantStaIfaceHal.setupIface(iface.name)) {
1132                  Log.e(TAG, "Failed to setup iface in supplicant on " + iface);
1133                  teardownInterface(iface.name);
1134                  mWifiMetrics.incrementNumSetupClientInterfaceFailureDueToSupplicant();
1135                  return null;
1136              }
...
1152      }
...
```

mSupplicantStaIfaceHal的setupIface方法调用addIfaceV1_1(ifaceName)，mSupplicantStaIfaceHal的addIfaceV1_1方法调用getSupplicantMockableV1_1().addInterface()，getSupplicantMockableV1_1调用getSupplicantMockable()，getSupplicantMockable方法调用了ISupplicant.getService()。

frameworks/opt/net/wifi/service/java/com/android/server/wifi/SupplicantStaIfaceHal.java

```
public class SupplicantStaIfaceHal {
...
392      private ISupplicantStaIface setupStaIface(@NonNull String ifaceName,
393              @NonNull ISupplicantIface ifaceHwBinder) throws RemoteException {
394          /* Prepare base type for later cast. */
395          ISupplicantStaIface iface = getStaIfaceMockable(ifaceHwBinder);
396
397          /* try newer version first. */
```

```
398         if (trySetupStaIfaceV1_1(ifaceName, iface)) {
399             logd("Newer HAL is found, skip V1_0 remaining init flow.");
400             return iface;
401         }
402
403         SupplicantStaIfaceHalCallback callback = new SupplicantStaIfaceHalCallback(ifaceName);
404         if (!registerCallback(iface, callback)) {
405             throw new RemoteException("Init StaIface V1_0 failed.");
406         }
407         /* keep this in a store to avoid recycling by garbage collector. */
408         mISupplicantStaIfaceCallbacks.put(ifaceName, callback);
409         return iface;
410     }



418     public boolean setupIface(@NonNull String ifaceName) {
419         final String methodStr = "setupIface";
420         if (checkSupplicantStaIfaceAndLogFailure(ifaceName, methodStr) != null) return false;
421         ISupplicantIface ifaceHwBinder;
422
423         if (isV1_1()) {
424             ifaceHwBinder = addIfaceV1_1(ifaceName);
425         } else {
426             ifaceHwBinder = getIfaceV1_0(ifaceName);
427         }
428         if (ifaceHwBinder == null) {
429             Log.e(TAG, "setupIface got null iface");
430             return false;
431         }
432
433         try {
434             ISupplicantStaIface iface = setupStaIface(ifaceName, ifaceHwBinder);
```

```
435             mISupplicantStaIfaces.put(ifaceName, iface);
436             com.mediatek.server.wifi.MtkWapi.setupMtkIface(ifaceName);
437             com.mediatek.server.wifi.MtkSupplicantStaIfaceHal.setupMtkIface(ifaceName);
438         } catch (RemoteException e) {
439             loge("setup StaIface failed: " + e.toString());
440             return false;
441         }
442 
443         return true;
444     }


508     private ISupplicantIface addIfaceV1_1(@NonNull String ifaceName) {
509         synchronized (mLock) {
510             ISupplicant.IfaceInfo ifaceInfo = new ISupplicant.IfaceInfo();
511             ifaceInfo.name = ifaceName;
512             ifaceInfo.type = IfaceType.STA;
513             Mutable<ISupplicantIface> supplicantIface = new Mutable<>();
514             try {
515                 getSupplicantMockableV1_1().addInterface(ifaceInfo,
516                         (SupplicantStatus status, ISupplicantIface iface) -> {
517                             if (status.code != SupplicantStatusCode.SUCCESS
518                                     && status.code != SupplicantStatusCode.FAILURE_IFACE_EXISTS) {
519                                 Log.e(TAG, "Failed to create ISupplicantIface " + status.code);
520                                 return;
521                             }
522                             supplicantIface.value = iface;
523                         });
524             } catch (RemoteException e) {
525                 Log.e(TAG, "ISupplicant.addInterface exception: " + e);
526                 handleRemoteException(e, "addInterface");
527                 return null;
528             } catch (NoSuchElementException e) {
```

```
529                 Log.e(TAG, "ISupplicant.addInterface exception: " + e);
530                 handleNoSuchElementException(e, "addInterface");
531                 return null;
532             }
533             return supplicantIface.value;
534         }
535     }



761     protected ISupplicant getSupplicantMockable() throws RemoteException, NoSuchElementException {
762         synchronized (mLock) {
763             ISupplicant iSupplicant = ISupplicant.getService();
764             if (iSupplicant == null) {
765                 throw new NoSuchElementException("Cannot get root service.");
766             }
767             return iSupplicant;
768         }
769     }
770
771     protected android.hardware.wifi.supplicant.V1_1.ISupplicant getSupplicantMockableV1_1()
772             throws RemoteException, NoSuchElementException {
773         synchronized (mLock) {
774             android.hardware.wifi.supplicant.V1_1.ISupplicant iSupplicantDerived =
775                     android.hardware.wifi.supplicant.V1_1.ISupplicant.castFrom(
776                             getSupplicantMockable());
777             if (iSupplicantDerived == null) {
778                 throw new NoSuchElementException("Cannot cast to V1.1 service.");
779             }
780             return iSupplicantDerived;
781         }
782     }
```

```
797      protected ISupplicantStaIface getStaIfaceMockable(ISupplicantIface iface) {
798          synchronized (mLock) {
799              return ISupplicantStaIface.asInterface(iface.asBinder());
800          }
801      }
}
```

getSupplicantMockableV1_1()返回的是一个ISupplicant.Stub.Proxy对象，使用该hidl代理对象的addInterface方法相当于调用到了wpa_supplicant进程中Supplicant对象的addInterface方法。具体实现参考external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/supplicant.cpp：

```
171  Return<void> Supplicant::addInterface(
172      const IfaceInfo& iface_info, addInterface_cb _hidl_cb)
173  {
174     return validateAndCall(
175         this, SupplicantStatusCode::FAILURE_IFACE_INVALID,
176         &Supplicant::addInterfaceInternal, _hidl_cb, iface_info);
177  }
....
256  std::pair<SupplicantStatus, sp<ISupplicantIface>>
257  Supplicant::addInterfaceInternal(const IfaceInfo& iface_info)
258  {
....
307     iface_params.ifname = iface_info.name.c_str();
308     struct wpa_supplicant* wpa_s =
309         wpa_supplicant_add_iface(wpa_global_, &iface_params, NULL);
310     if (!wpa_s) {
311         return {{SupplicantStatusCode::FAILURE_UNKNOWN, ""}, {}};
312     }
```

```
313     // The supplicant core creates a corresponding hidl object via
314     // HidlManager when |wpa_supplicant_add_iface| is called.
315     return getInterfaceInternal(iface_info);
316 }
```

addInterface方法最终调用到了addInterfaceInternal方法，addInterfaceInternal方法调用wpa_supplicant_add_iface函数。wpa_supplicant_add_iface函数在ctrl_interface指定的目录下创建了一个unix socket套接字文件，并间接调用到HidlManager::registerInterface

external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/hidl_manager.cpp

```
440 int HidlManager::registerInterface(struct wpa_supplicant *wpa_s)...
460         if (addHidlObjectToMap<StaIface>(
461             wpa_s->ifname,
462             new StaIface(wpa_s->global, wpa_s->ifname),
463             sta_iface_object_map_)) {
464             wpa_printf(
465                 MSG_ERROR,
466                 "Failed to register STA interface with HIDL "
467                 "control: %s",
468                 wpa_s->ifname);
469             return 1;
470         }...}
```

HidlManager::registerInterface方法通过调用addHidlObjectToMap将new出来的一个StaIface对象放入Map中。

这一系列搞完后返回到addInterfaceInternal的return处调用getInterfaceInternal，getInterfaceInternal间接调用到HidlManager::getStaIfaceHidlObjectByIfname()从map中将StaIface对象取回并返回给JAVA。JAVA端调用getSupplicantMockableV1_1().addInterface()其实拿到的是StaIface对象的代理对象。

StaIface类派生自ISupplicantStaIface类，

external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/sta_iface.h

```
class StaIface : public V1_3::ISupplicantStaIface
```

ISupplicantStaIface类派生自ISupplicantIface类，hardware/interfaces/wifi/supplicant/1.0/ISupplicantStaIface.hal

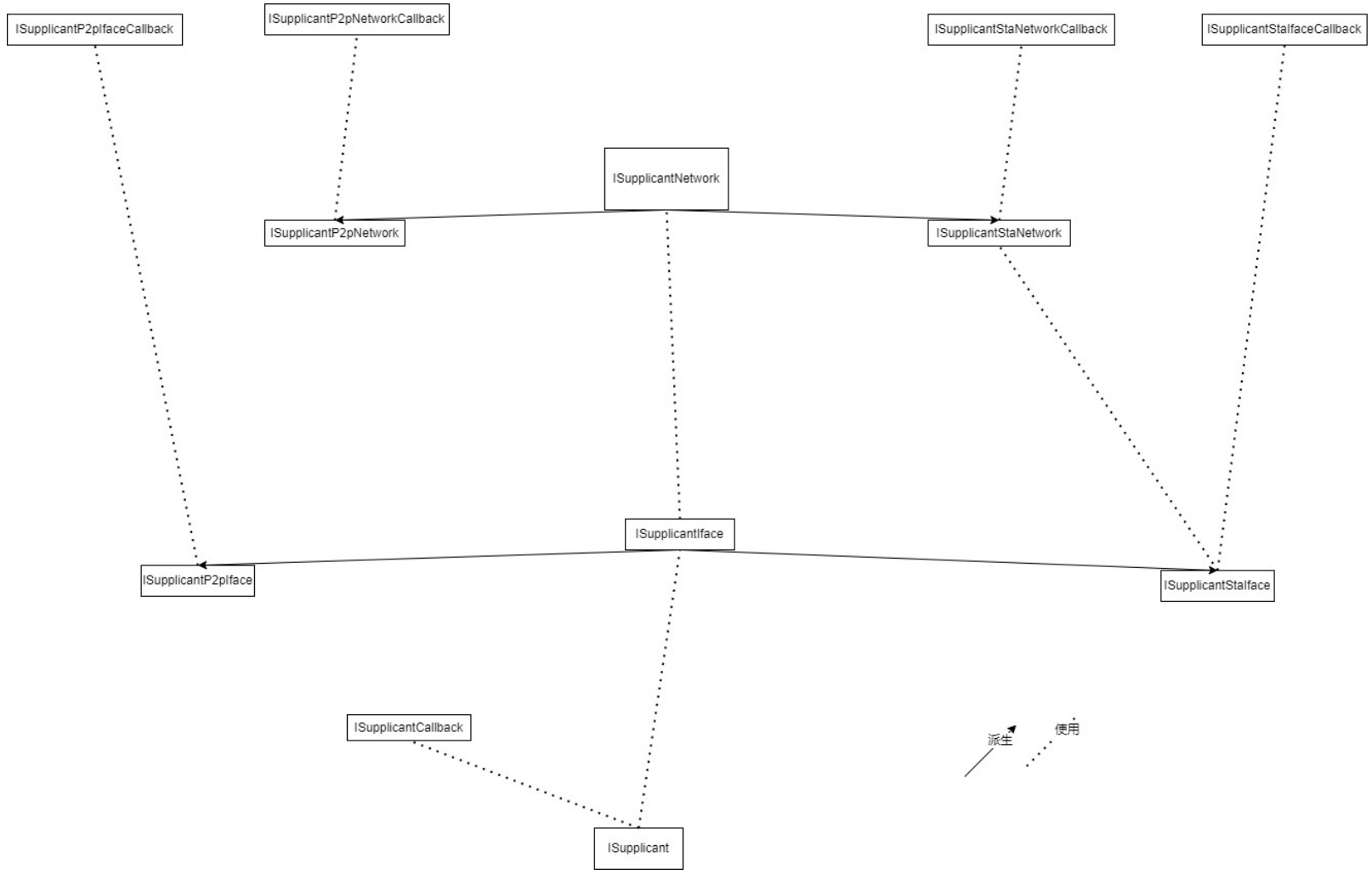
```
interface ISupplicantStaIface extends ISupplicantIface
```

在SupplicantStaIfaceHal类的的setupIface方法中调完addIfaceV1_1，addIfaceV1_1首先通过hwServiceManager拿到ISupplicant代理对象，再通过ISupplicant代理对象调用addInterface拿到ISupplicantIface的代理对象（其实是ISupplicantStaIface的代理对象）后返回，setupIface方法接着再调用setupStaIface方法，setupStaIface方法调用getStaIfaceMockable将ISupplicantIface的代理对象转换成ISupplicantStaIface的代理对象，即wpa_supplicant中sta_iface对象的代理对象。

3 hal文件中interface的关系

3.1 android.hardware.wifi@1.x-service

3.2 wpa_supplicant


ISupplicantP2pIfaceCallback
ISupplicantP2pNetworkCallback
ISupplicantStaNetworkCallback
ISupplicantStaIfaceCallback
ISupplicantNetwork
ISupplicantP2pNetwork
ISupplicantStaNetwork
ISupplicantIface
ISupplicantP2pIface
ISupplicantStaIface
ISupplicantCallback
ISupplicant
派生
使用

4.hal层的wifi服务

5. hal层wpa_supplicant服务

5.1 基本命令行参数及配置文件

```
1|console:/ #  cat /vendor/etc/init/android.hardware.wifi.supplicant-service.rc
service wpa_supplicant /vendor/bin/hw/wpa_supplicant \
    -O/data/vendor/wifi/wpa/sockets -dd \
    -g@android:wpa_wlan0
    #   we will start as root and wpa_supplicant will switch to user wifi
    #   after setting up the capabilities required for WEXT
    #   user wifi
    #   group wifi inet keystore
    interface android.hardware.wifi.supplicant@1.0::ISupplicant default
    interface android.hardware.wifi.supplicant@1.1::ISupplicant default
    interface android.hardware.wifi.supplicant@1.2::ISupplicant default
    interface android.hardware.wifi.supplicant@1.3::ISupplicant default
    interface vendor.mediatek.hardware.wifi.supplicant@2.0::ISupplicant default
    class main
```

```
    socket wpa_wlan0 dgram 660 wifi wifi
    disabled
    oneshot
```

```
console:/ #  cat /vendor/etc/wifi/wpa_supplicant.conf
ctrl_interface=wlan0
update_config=1
manufacturer=MediaTek Inc.
device_name=Wireless Client
model_name=MTK Wireless Model
model_number=1.0
serial_number=2.0
device_type=10-0050F204-5
os_version=01020300
config_methods=display push_button keypad
p2p_no_group_iface=1
driver_param=use_p2p_group_interface=1
hs20=1
pmf=1
wowlan_disconnect_on_deinit=1
```

```
console:/ # /vendor/bin/hw/wpa_supplicant -h
wpa_supplicant v2.10-devel-11
```

```
Copyright (c) 2003-2019, Jouni Malinen <j@w1.fi> and contributors

This software may be distributed under the terms of the BSD license.
See README for more details.

This product includes software developed by the OpenSSL Project
for use in the OpenSSL Toolkit (http://www.openssl.org/)

usage:
  wpa_supplicant [-BddhKLqqtvW] [-P<pid file>] [-g<global ctrl>] \
        [-G<group>] \
        -i<ifname> -c<config file> [-C<ctrl>] [-D<driver>] [-p<driver_param>] \
        [-b<br_ifname>] [-e<entropy file>] \
        [-o<override driver>] [-O<override ctrl>] \
        [-N -i<ifname> -c<conf> [-C<ctrl>] [-D<driver>] \
        [-m<P2P Device config file>] \
        [-p<driver_param>] [-b<br_ifname>] [-I<config file>] ...]

drivers:
  nl80211 = Linux nl80211/cfg80211
options:
  -b = optional bridge interface name
  -B = run daemon in the background
  -c = Configuration file
  -C = ctrl_interface parameter (only used if -c is not)
  -d = increase debugging verbosity (-dd even more)
  -D = driver name (can be multiple drivers: nl80211,wext)
  -e = entropy file
  -g = global ctrl_interface
  -G = global ctrl_interface group
  -h = show this help text
  -i = interface name
  -I = additional configuration file
```

```
  -K = include keys (passwords, etc.) in debug output
  -L = show license (BSD)
  -m = Configuration file for the P2P Device interface
  -N = start describing new interface
  -o = override driver parameter for new interfaces
  -O = override ctrl_interface parameter for new interfaces
  -p = driver parameters
  -P = PID file
  -q = decrease debugging verbosity (-qq even less)
  -t = include timestamp in debug messages
  -v = show version
  -W = wait for a control interface monitor before starting
example:
  wpa_supplicant -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf
```

ls /data/vendor/wifi/wpa/sockets/

setprop wifi.interface wlan0 指定默认套接字/网卡接口，在-i参数未指定时生效

直接使用-g参数指定/dev/socket/wpa_wlan0 套接字文件路径及文件名，即socket的sun_path，sun_path = ctrl_interface + ifname

```
#file /dev/socket/wpa_wlan0
/dev/socket/wpa_wlan0: no such file or directory

#wpa_supplicant -g@android:wpa_wlan0

#file /dev/socket/wpa_wlan0
/dev/socket/wpa_wlan0: socket

#wpa_cli  -g@android:wpa_wlan0
```

使用ctrl_interface属性指定socket套接字所在目录，使用ifname属性指定socket套接字文件名

```
//通过-C参数指定socket套接字所在目录的路径（使用-C了就不能使用-c指定配置文件），-i参数指定该目录下的socket套接字文件名
/vendor/bin/hw/wpa_supplicant -C /data/vendor/wifi/wpa/sockets -i wlan0

//通过-c指定wpa_supplicant.conf配置文件，配置文件中有ctrl_interface=/data/vendor/wifi/wpa/sockets  ifname=wlan0
wpa_supplicant -c /data/vendor/wifi/wpa/wpa_supplicant.conf

//使用-c指定wpa_supplicant.conf配置文件，配置文件中有ctrl_interface=/data/vendor/wifi/wpa/sockets，使用-i参数指定
ifname套接字文件名
wpa_supplicant -c /data/vendor/wifi/wpa/wpa_supplicant.conf -i wlan0


wpa_cli  -p /data/vendor/wifi/wpa/sockets  -i wlan0



注意，-O选项可以覆盖-C指定的ctrl_interface或在-c指定的conf文件中的ctrl_interface，即可以通过-O指定ctrl_interface
```

```
wpa_cli   <-xx   <yyy>>   zzz
wpa_cli 将除-xx及-xx选项所需的yyy之外的剩余参数zzz都会通过unix套接字原封不动地通过ctrl_interface指定的目录下指定的unix套接字文件发给wpa_supplicant, wpa_supplicant监听这些套件字文件并做后续相应事件处理。
```

装载wifi驱动后得到的wlan0网卡要经过如下步骤后才能正常使用：

insmod wifi-driver.ko #执行成功后就能看到网卡名wlan0

ip link set dev wlan0 up #或 ifconfig wlan0 up

wpa_cli add_inface wlan0 #成功后ctrl_interface指定的目录下就有wlan0（与网卡同名）套接字文件了

wpa_cli -i wlan0 scan

wpa_cli -i wlan0 scan_result

wpa_cli -i wlan0 add_network #在wlan0接口中添加网络编号,得到networkID

wpa_cli -i wlan0 set_network networkID xxxxx #wifi热点名ssid、key_mgmt加密方式wpa/psk认证等，相当于配置文件中network={ssid=xxx psk=xxx key_mgnt=xxx ...}字段

wpa_cli -i wlan0 select_network networkID

wpa_cli -i wlan0 enable_network enable_network networkID

busybox udhcpc -i wlan0 -q #使用hdcp协议让DNS服务器分配ip，dhclient -i wlan0 功能与其类似

ip addr add XXX.XXX.XXX.XXX/24 dev wlan0 #将wlan0网卡的ip设置成DNS服务器分配的ip，ifconfig也行

#查询当前局域网的网关

#添加路由表

ip route <add/del/change/show>

ip route add default via 192.168.1.2 dev eth0

ip route add 192.168.10.0/24 via 192.168.5.100 dev eth0

ip route add 192.168.42.0/24   dev eth0

ip route add 192.168.43.0/24   dev wlan0

ip route add < default 或者 0.0.0.0/0 > via 192.168.43.1 dev wlan0

ip route add 192.22.31.22/32 via 192.168.43.1 dev eth0

ip route add 192.22.33.0/24 via 192.168.43.1 dev eth0

ip route del < ip/mask >

ip route change

配置DNS

```
184     struct wpa_interface *ifaces, *iface;
185     int iface_count, exitcode = -1;
186     struct wpa_params params;
187     struct wpa_global *global;

iface->confname =""/.../wpa_supplicnat.conf"
iface->ctrl_interface=
iface->ifname =


global->params.ctrl_interface = "@android:wpa_wlan0"
global->params.override_ctrl_interface="/data/vendor/wifi/wpa/sockets"
```

```
 echo ctrl_interface=/data/vendor/wifi/wpa/sockets > /data/meta_wpa_supplicant.conf
/vendor/bin/hw/wpa_supplicant -iwlan0 -Dnl80211 -c /data/meta_wpa_supplicant.conf &

    wpa_cli -i wlan0 add_network //增加网络号
    wpa_cli -i wlan0 set_network 1 ssid '"ZTE123"' //设置TP-LINK_5G    TP-LINK_E41CB0
    wpa_cli -i wlan0 set_network 1 key_mgmt "WPA-PSK" //设置加密方式
    wpa_cli -i wlan0 set_network 1 psk '"88888888"' //设置密码
    wpa_cli -i wlan0 enable_network 1 //使能该网络号
    wpa_cli -i wlan0 status //查看状态
    wpa_cli -i wlan0 add_network //获得新的网络编号

    wpa_cli -i wlan0 set_network 1 ssid '"ZTE123"'
    wpa_cli -i wlan0 set_network 1 key_mgmt "WPA-PSK"
    wpa_cli -i wlan0 set_network 1 psk '"88888888"'
    wpa_cli -i wlan0 enable_network 1

    wpa_cli -i wlan0 status
```

```
    wpa_cli -i wlan0 list_networks
    wpa_cli -i wlan0 select_network <网络编号>
    wpa_cli -i wlan0 add_network
    wpa_cli -i wlan0 remove_network <网络编号>
    wpa_cli -i wlan0 disable_network <网络编号>
    wpa_cli -i wlan0 enable_network <网络编号>
    wpa_cli -i wlan0 save_config
    wpa_cli -i wlan0 disconnect
    wpa_cli -i wlan0 reconnect


    std::string sUsername = "6418000265";
    std::string sPassword = "yzf2019**";
    std::string sSSID     = "zte_wlanap_uds_test118";
    int iNetworkID = 0;
    std::string sKeymgmt = "WPA-EAP";   // WPA2-GTC
    std::string sEapType = "PEAP_90";   // ZTE无线特别定义的
    std::string sCACert = "/usr/local/wpa_server/cert_ca.crt";
    std::string sClientCert = "/usr/local/wpa_server/cert_server.crt";
    std::string sPhase1 = "peaplabel=0";
    std::string sPhase2 = "auth=MSCHAPV2";
    std::string sPriority = "10";




/vendor/bin/hw/wpa_supplicant -C /data/vendor/wifi/wpa/sockets -iwlan0 -Dnl80211 &
    wpa_cli -i wlan0 add_network
    对应StaIface::addNetwork
```

```
wpa_cli -i wlan0 set_network 0 ssid          '"zte_wlanap_sn_CS_h3c"'
对应StaNetwork::setSsid

wpa_cli -i wlan0 set_network 0 key_mgmt        WPA-EAP
对应StaNetwork::setKeyMgmt    StaNetwork::setKeyMgmt_1_2   StaNetwork::setKeyMgmt_1_3

wpa_cli -i wlan0 set_network 0 eap            PEAP_90
对应StaNetwork::setEapMethod


identity为二维码qrKey
wpa_cli -i wlan0 set_network 0 identity       '"   "'
对应StaNetwork::setEapIdentity

wpa_cli -i wlan0 set_network 0 password      '"   "'
对应StaNetwork::setEapPassword

wpa_cli -i wlan0 set_network 0 ca_cert        '"/usr/local/wpa_server/cert_ca.crt"'
对应StaNetwork::setEapCACert

wpa_cli -i wlan0 set_network 0 client_cert     '"/usr/local/wpa_server/cert_server.crt"'
对应StaNetwork::setEapClientCert

wpa_cli -i wlan0 set_network 0 phase1          '"peaplabel=0"'
不完全对应StaNetwork::enableTlsSuiteBEapPhase1Param，都是设置的wpa_ssid->eap.phase1但值不同

wpa_cli -i wlan0 set_network 0 phase2          '"auth=MSCHAPV2"'
对应StaNetwork::setEapPhase2Method

wpa_cli -i wlan0 set_network 0 priority        10
没有对应的hidl接口

wpa_cli -i wlan0 select_network 0
```

```
    对应StaNetwork::select

    wpa_cli -i wlan0 enable_network 0
    对应StaNetwork::enable

    wpa_cli -i wlan0 save_config
    sta没有对应的hidl接口，但p2p对应P2pIface::saveConfig
```

1、通过全局控unix套接字连接wlan0接口

wpa_cli -g /dev/socket/wpa_wlan0 -i wlan0

wpa_cli -g@android:wpa_wlan0 -i wlan0

2、通过某个路径下的wlan0 unix套接字连接wlan0接口

wpa_cli -p /data/vendor/wifi/wpa/sockets -i wlan0

5.2 wpa_supplicant代码分析

全局控制unix套接字接口

/dev/socket/wpa_wlan0

可以通过该unix套接字发送INTERFACE_ADD命令来在ctrl_interface下创建ifname，即在/data/vendor/wifi/wpa/sockets/目录下创建wlan0 unix套接字文件，如果只是通过start wpa_supplicant来启动wpa_supplicant服务，在/data/vendor/wifi/wpa/sockets/下没有任何文件，如果通过UI打开wifi，来启动wpa_supplicant服务，在wpa_supplicant服务启动后发送INTERFACE_ADD在/data/vendor/wifi/wpa/sockets/下创建wlan0套接字接口文件。该接口主要用于增加或删除网络接口。

单一网卡(例如wlan0)的unix套接字控制接口

/data/vendor/wifi/wpa/sockets/wlan0

向该套接字发送

eap_peer_peap_register函数调用eap_peer_method_alloc函数分配一个struct eap_method结构体，并经过初始化后调用eap_peer_method_register函数将struct eap_method结构体组成一个链表， 静态全局指针eap_methods 指向这个链表。

android.hardware.wifi.supplicant-service.rc中：

```
service wpa_supplicant /vendor/bin/hw/wpa_supplicant \
    -O/data/vendor/wifi/wpa/sockets -dd \
    -g@android:wpa_wlan0
    #   we will start as root and wpa_supplicant will switch to user wifi
    #   after setting up the capabilities required for WEXT
    #   user wifi
    #   group wifi inet keystore
    interface android.hardware.wifi.supplicant@1.0::ISupplicant default
    interface android.hardware.wifi.supplicant@1.1::ISupplicant default
    interface android.hardware.wifi.supplicant@1.2::ISupplicant default
    interface android.hardware.wifi.supplicant@1.3::ISupplicant default
    interface vendor.mediatek.hardware.wifi.supplicant@2.0::ISupplicant default
    class main
    socket wpa_wlan0 dgram 660 wifi wifi
```

```
    disabled
    oneshot
```

init进程在fork-exec wpa_supplicant进程之前会在/dev/socket目录下创建wpa_wlan0 unix套接字文件，然后把套接字fd放到一个ANDROID_SOCKET_wpa_wlan0 环境变量中，fork-exec出的wpa_supplicant进程将继承fd和套接字。

wpa_supplicant_global_ctrl_iface_init函数中调用wpas_global_ctrl_iface_open_sock函数，wpas_global_ctrl_iface_open_sock函数调用android_get_control_socket函数。android_get_control_socket函数首先调用__android_get_control_from_env，使用-g命令行参数传进来的套接字名组合成ANDROID_SOCKET_wpa_wlan0字符串，再使用该字符串调用getenv从环境变量中拿到fd。拿到fd后在android_get_control_socket函数中调用getsockname使用fd拿到套接字的sun_path，再查看与"/dev/socket"前缀和 -g传进来的"wpa_wlan0"套接字文件名是否匹配，如果成功返回到wpas_global_ctrl_iface_open_sock函数中将fd赋给priv->socks后再使用fcntl将套接字设为非阻塞模式，然后调用eloop_register_read_sock将套接字加入epoll事件队列，后续执行eloop_run时进行事件监听。当触发fd的可读事件时调用eloop_register_read_sock注册的wpa_supplicant_global_ctrl_iface_receive函数。

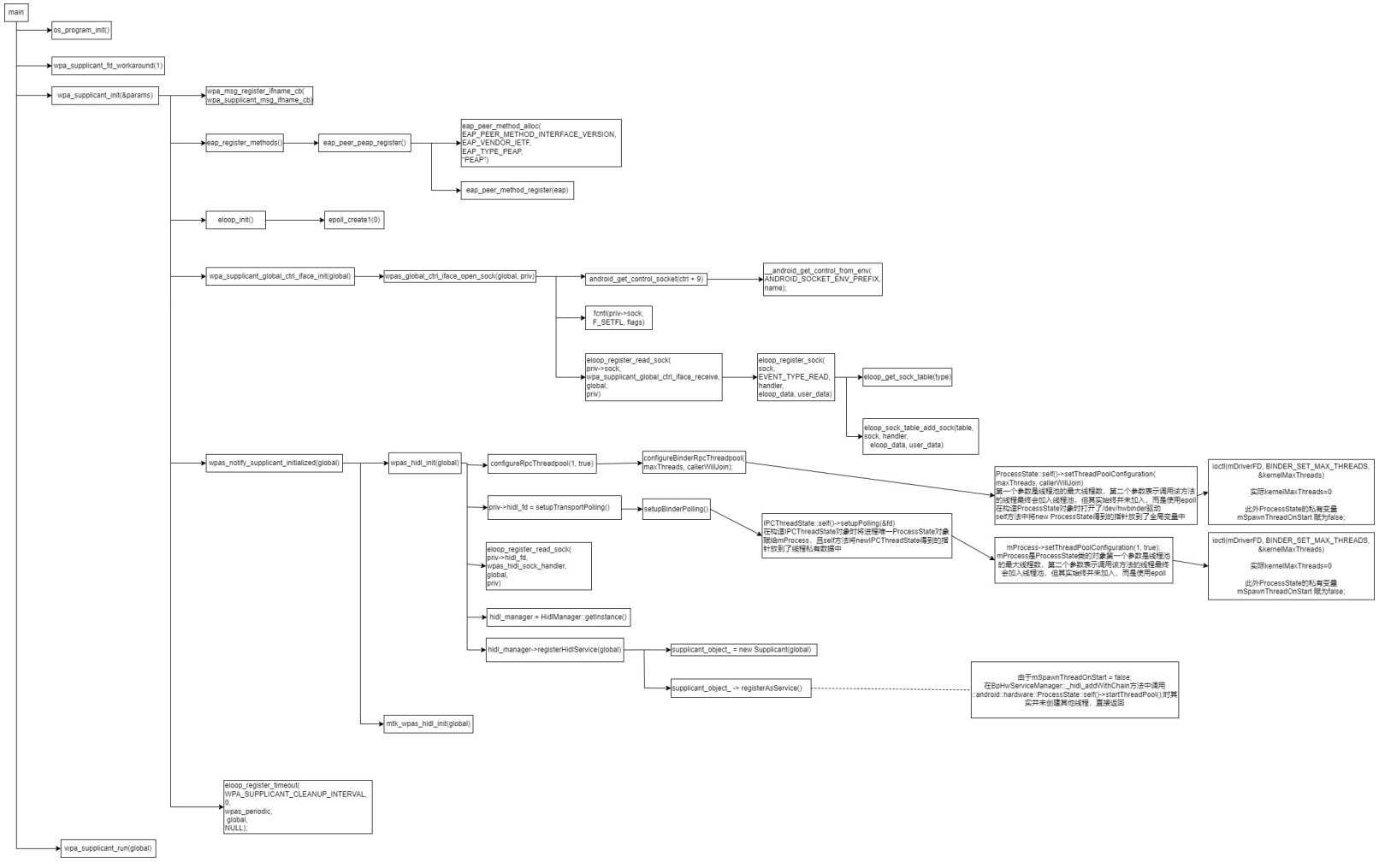
将套接字fd从ANDROID_SOCKET_wpa_wlan0 环境变量中取出，然后与-g命令行参数传进来的套接字名

tr.cook是BnHwSupplicant的智能指针，

```
main函数中的局部变量
struct wpa_supplicant *wpa_s
struct wpa_interface  *ifaces
struct wpa_global     *global;
```

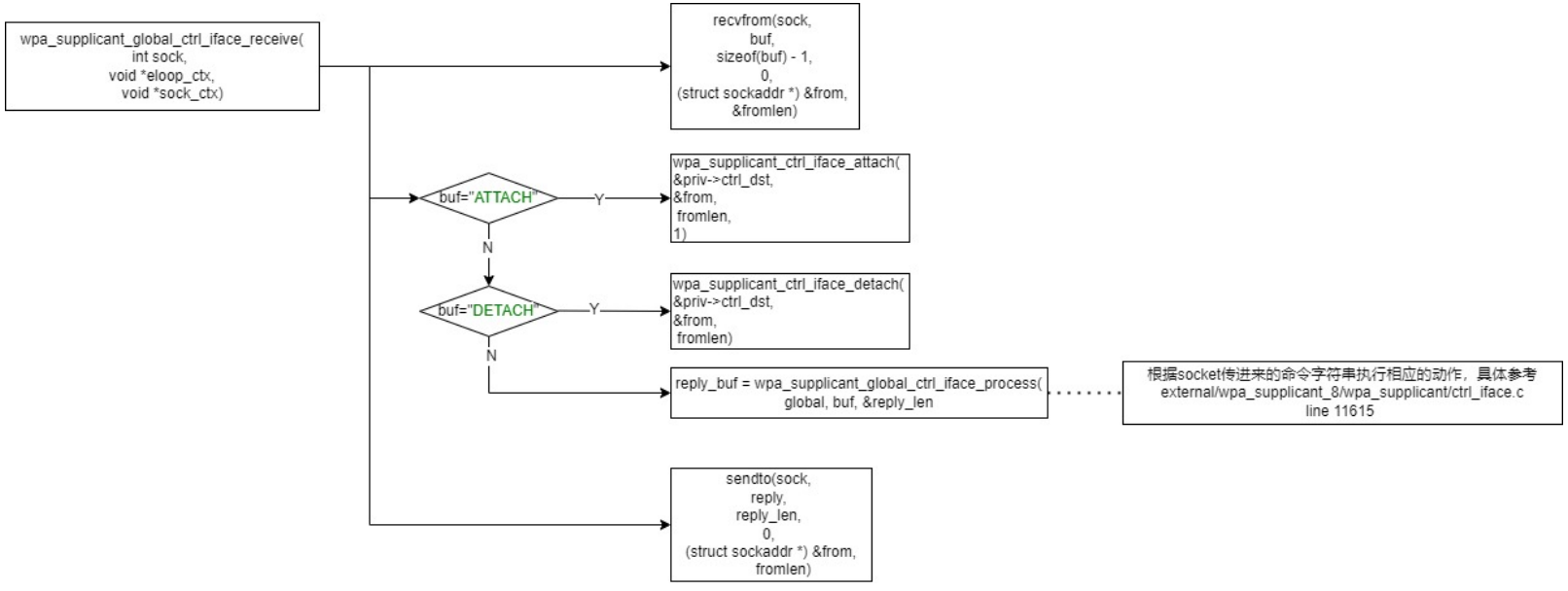
```
static struct eap_method *eap_methods;
external/wpa_supplicant_8/src/eap_peer/eap_methods.c
```

5.2.1 初始化


main
os_program_init()
wpa_supplicant_fd_workaround(1)
wpa_supplicant_init(&params)
wpa_msg_register_ifname_cb(
wpa_supplicant_msg_ifname_cb)
eap_register_methods()
eap_peer_peap_register()
eap_peer_method_alloc(
EAP_PEER_METHOD_INTERFACE_VERSION,
EAP_VENDOR_IETF,
EAP_TYPE_PEAP,
"PEAP")
eap_peer_method_register(eap)
eloop_init()
epoll_create1(0)
wpa_supplicant_global_ctrl_iface_init(global)
wpas_global_ctrl_iface_open_sock(global, priv)
android_get_control_socket(ctrl + 9)
__android_get_control_from_env(
ANDROID_SOCKET_ENV_PREFIX,
name);
fcntl(priv->sock,
F_SETFL, flags)
eloop_register_read_sock(
priv->sock,
wpa_supplicant_global_ctrl_iface_receive,
global,
priv)
eloop_register_sock(
sock,
EVENT_TYPE_READ,
handler,
eloop_data, user_data)
eloop_get_sock_table(type)
eloop_sock_table_add_sock(table,
sock, handler,
eloop_data, user_data)
wpas_notify_supplicant_initialized(global)
wpas_hidl_init(global)
configureRpcThreadpool(1, true)
configureBinderRpcThreadpool(
maxThreads, callerWillJoin);
ProcessState::self()->setThreadPoolConfiguration(
maxThreads, callerWillJoin)
第一个参数是线程池的最大线程数，第二个参数表示调用该方法的线程最终会加入线程池，但其实始终并未加入，而是使用epoll
在构造ProcessState对象时打开了/dev/hwbinder驱动
self方法中将new ProcessState得到的指针放到了全局变量中
ioctl(mDriverFD, BINDER_SET_MAX_THREADS,
&kernelMaxThreads)
实际kernelMaxThreads=0
此外ProcessState的私有变量
mSpawnThreadOnStart 赋为false;
priv->hidl_fd = setupTransportPolling()
setupBinderPolling()
IPCThreadState::self()->setupPolling(&fd)
在构造IPCThreadState对象时将进程唯一ProcessState对象赋值mProcess，且self方法将newIPCThreadState得到的指针放到了线程私有数据中
mProcess->setThreadPoolConfiguration(1, true);
mProcess是ProcessState类的对象第一个参数是线程池的最大线程数，第二个参数表示调用该方法的线程最终会加入线程池，但其实始终并未加入，而是使用epoll
ioctl(mDriverFD, BINDER_SET_MAX_THREADS,
&kernelMaxThreads)
实际kernelMaxThreads=0
此外ProcessState的私有变量
mSpawnThreadOnStart 赋为false;
eloop_register_read_sock(
priv->hidl_fd,
wpas_hidl_sock_handler,
global,
priv)
hidl_manager = HidlManager::getInstance()
hidl_manager->registerHidlService(global)
supplicant_object_ = new Supplicant(global)
supplicant_object_ -> registerAsService()
由于mSpawnThreadOnStart = false;
在BpHwServiceManager::_hidl_addWithChain方法中调用
::android::hardware::ProcessState::self()->startThreadPool()时其实并未创建其他线程，直接返回
mtk_wpas_hidl_init(global)
eloop_register_timeout(
WPA_SUPPLICANT_CLEANUP_INTERVAL,
0,
wpas_periodic,
global,
NULL);
wpa_supplicant_run(global)

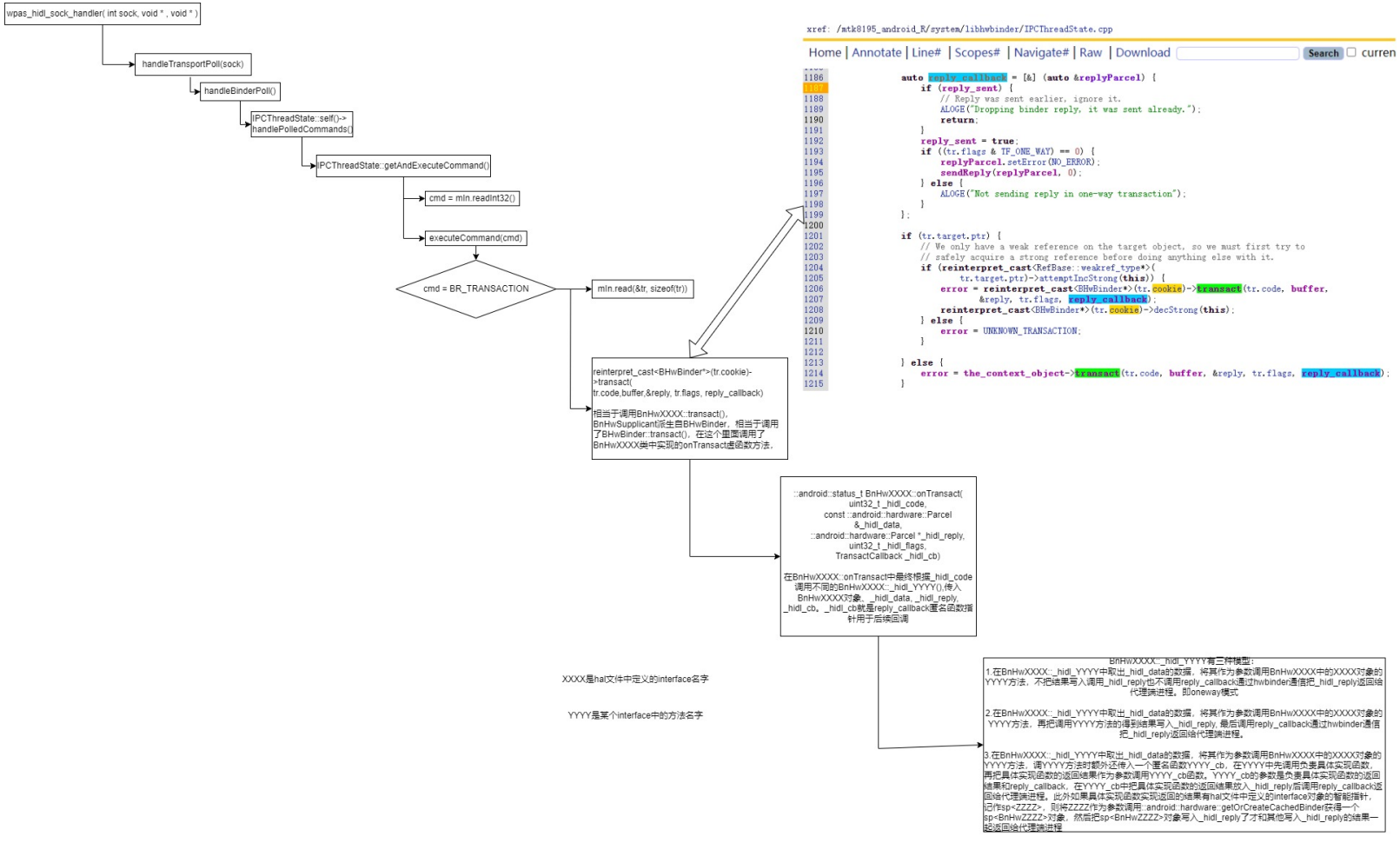
5.2.2 全局控制unix套接字事件处理

wpa_supplicant_global_ctrl_iface_receive



5.2.3 hwbinder通信事件处理

5.2.3.1 wpas_hidl_sock_handler


wpas_hidl_sock_handler(int sock, void * , void *)
handleTransportPoll(sock)
handleBinderPoll()
IPCThreadState::self()->handlePolledCommands()
IPCThreadState::getAndExecuteCommand()
cmd = mln.readInt32()
executeCommand(cmd)
cmd = BR_TRANSACTION
mln.read(&tr, sizeof(tr))
reinterpret_cast<BHwBinder*>(tr.cookie)->transact(tr.code,buffer,&reply, tr.flags, reply_callback)
相当于调用BnHwXXXX::transact(),BnHwSupplicant派生自BHwBinder，相当于调用了BHwBinder::transact()，在这个里面调用了BnHwXXXX类中实现的onTransact虚函数方法，
::android::status_t BnHwXXXX::onTransact(uint32_t _hidl_code, const ::android::hardware::Parcel &_hidl_data, ::android::hardware::Parcel *_hidl_reply, uint32_t _hidl_flags, TransactCallback _hidl_cb)
在BnHwXXXX::onTransact中最终根据_hidl_code调用不同的BnHwXXXX::_hidl_YYYY(),传入BnHwXXXX对象，_hidl_data, _hidl_reply, _hidl_cb，_hidl_cb就是reply_callback匿名函数指针用于后续回调
XXXX是hal文件中定义的interface名字
YYYY是某个interface中的方法名字
BnHwXXXX::_hidl_YYYY有三种模型：
1.在BnHwXXXX::_hidl_YYYY中取出_hidl_data的数据，将其作为参数调用BnHwXXXX中的XXXX对象的YYYY方法，不把结果写入调用_hidl_reply也不调用reply_callback通过hwbinder通信把_hidl_reply返回给代理端进程。即oneway模式
2.在BnHwXXXX::_hidl_YYYY中取出_hidl_data的数据，将其作为参数调用BnHwXXXX中的XXXX对象的YYYY方法，再把调用YYYY方法的得到结果写入_hidl_reply, 最后调用reply_callback通过hwbinder通信把_hidl_reply返回给代理端进程。
3.在BnHwXXXX::_hidl_YYYY中取出_hidl_data的数据，将其作为参数调用BnHwXXXX中的XXXX对象的YYYY方法，调YYYY方法时额外还传入一个匿名函数YYYY_cb，在YYYY中先调用负责具体实现函数，再把具体实现函数的返回结果作为参数调用YYYY_cb函数。YYYY_cb的参数是负责具体实现函数的返回结果和reply_callback，在YYYY_cb中把具体实现函数的返回结果放入_hidl_reply后调用reply_callback返回给代理端进程。此外如果具体实现函数实现返回的结果有hal文件中定义的interface对象的智能指针，记作sp<ZZZZ>，则将ZZZZ作为参数调用::android::hardware::getOrCreateCachedBinder获得一个sp<BnHwZZZZ>对象，然后把sp<BnHwZZZZ>对象写入_hidl_reply了才和其他写入_hidl_reply的结果一起返回给代理端进程
xref: /mtk8195_android_R/system/libhwbinder/IPCThreadState.cpp
Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download
Search
curren
1186 auto reply_callback = [&] (auto &replyParcel) {
1187 if (reply_sent) {
1188 // Reply was sent earlier, ignore it.
1189 ALOGE("Dropping binder reply, it was sent already.");
1190 return;
1191 }
1192 reply_sent = true;
1193 if ((tr.flags & TF_ONE_WAY) == 0) {
1194 replyParcel.setError(NO_ERROR);
1195 sendReply(replyParcel, 0);
1196 } else {
1197 ALOGE("Not sending reply in one-way transaction");
1198 }
1199 };
1200
1201 if (tr.target.ptr) {
1202 // We only have a weak reference on the target object, so we must first try to
1203 // safely acquire a strong reference before doing anything else with it.
1204 if (reinterpret_cast<RefBase::weakref_type*>(
1205 tr.target.ptr)->attemptIncStrong(this)) {
1206 error = reinterpret_cast<BHwBinder*>(tr.cookie)->transact(tr.code, buffer,
1207 &reply, tr.flags, reply_callback);
1208 reinterpret_cast<BHwBinder*>(tr.cookie)->decStrong(this);
1209 } else {
1210 error = UNKNOWN_TRANSACTION;
1211 }
1212
1213 } else {
1214 error = the_context_object->transact(tr.code, buffer, &reply, tr.flags, reply_callback);
1215 }

5.2.3.2 oneway方法接口

hal文件 hardware/interfaces/wifi/supplicant/1.1/ISupplicant.hal

```
28 interface ISupplicant extends @1.0::ISupplicant {
...
58     /**
59      * Terminate the service.
60      * This must de-register the service and clear all state. If this HAL
61      * supports the lazy HAL protocol, then this may trigger daemon to exit and
62      * wait to be restarted.
63      */
64     oneway terminate();
65 };
```

代理端

out/soong/.intermediates/hardware/interfaces/wifi/supplicant/1.1/android.hardware.wifi.supplicant@1.1_genc++/gen/android/hardware/wifi/supplicant/1.1/SupplicantAll.cpp

```
::android::hardware::Return<void> BpHwSupplicant::terminate(){
    ::android::hardware::Return<void>  _hidl_out =
::android::hardware::wifi::supplicant::V1_1::BpHwSupplicant::_hidl_terminate(this, this);

    return _hidl_out;
}
```

```
::android::hardware::Return<void> BpHwSupplicant::_hidl_terminate(::android::hardware::IInterface *_hidl_this,
::android::hardware::details::HidlInstrumentor *_hidl_this_instrumentor) {
    #ifdef __ANDROID_DEBUGGABLE__
    bool mEnableInstrumentation = _hidl_this_instrumentor->isInstrumentationEnabled();
    const auto &mInstrumentationCallbacks = _hidl_this_instrumentor->getInstrumentationCallbacks();
    #else
    (void) _hidl_this_instrumentor;
    #endif // __ANDROID_DEBUGGABLE__
    ::android::ScopedTrace PASTE(___tracer, __LINE__) (ATRACE_TAG_HAL,
"HIDL::ISupplicant::terminate::client");
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::CLIENT_API_ENTRY, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "terminate", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    ::android::hardware::Parcel _hidl_data;
    ::android::hardware::Parcel _hidl_reply;
    ::android::status_t _hidl_err;
    ::android::status_t _hidl_transact_err;
    ::android::hardware::Status _hidl_status;

    _hidl_err = _hidl_data.writeInterfaceToken(BpHwSupplicant::descriptor);
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    _hidl_transact_err = ::android::hardware::IInterface::asBinder(_hidl_this)->transact(11 /* terminate */,
_hidl_data, &_hidl_reply, 1u /* oneway */);
    if (_hidl_transact_err != ::android::OK)
    {
```

```
        _hidl_err = _hidl_transact_err;
        goto _hidl_error;
    }

    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::CLIENT_API_EXIT, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "terminate", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    return ::android::hardware::Return<void>();

_hidl_error:
    _hidl_status.setFromStatusT(_hidl_err);
    return ::android::hardware::Return<void>(_hidl_status);
}
```

```
代理端填充_hidl_data后调用
::android::hardware::IInterface::asBinder(_hidl_this)->transact(11 /* terminate */, _hidl_data, &_hidl_reply,
1u /* oneway */);
通过binder驱动发送出去之后并不读取 _hidl_reply直接返回
```

服务实现端

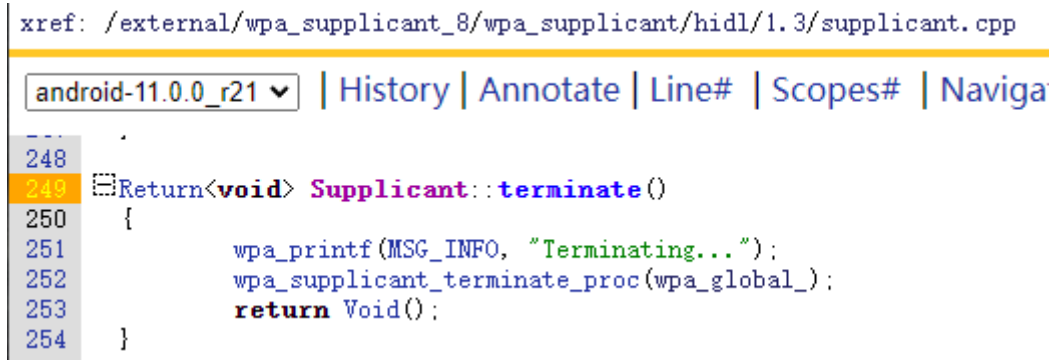
out/soong/.intermediates/hardware/interfaces/wifi/supplicant/1.1/android.hardware.wifi.supplicant@1.1_genc++/gen/android/hardware/wifi/supplicant/1.1/SupplicantAll.cpp

```
::android::status_t BnHwSupplicant::_hidl_terminate(
        ::android::hidl::base::V1_0::BnHwBase* _hidl_this,
        const ::android::hardware::Parcel &_hidl_data,
        ::android::hardware::Parcel *_hidl_reply,
        TransactCallback _hidl_cb) {
    #ifdef __ANDROID_DEBUGGABLE__
    bool mEnableInstrumentation = _hidl_this->isInstrumentationEnabled();
    const auto &mInstrumentationCallbacks = _hidl_this->getInstrumentationCallbacks();
    #endif // __ANDROID_DEBUGGABLE__

    ::android::status_t _hidl_err = ::android::OK;
    if (!_hidl_data.enforceInterface(BnHwSupplicant::Pure::descriptor)) {
        _hidl_err = ::android::BAD_TYPE;
        return _hidl_err;
    }

    atrace_begin(ATRACE_TAG_HAL, "HIDL::ISupplicant::terminate::server");
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::SERVER_API_ENTRY, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "terminate", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    ::android::hardware::Return<void> _hidl_ret = static_cast<ISupplicant*>(_hidl_this->getImpl().get())-
>terminate();

    (void) _hidl_cb;

    atrace_end(ATRACE_TAG_HAL);
```

```
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::SERVER_API_EXIT, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "terminate", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    _hidl_ret.assertOk();
    ::android::hardware::writeToParcel(::android::hardware::Status::ok(), _hidl_reply);

    return _hidl_err;
}
```

```
在服务端执行::android::hardware::Return<void> _hidl_ret = static_cast<ISupplicant*>(_hidl_this-
>getImpl().get())->terminate()调用具体实现的函数后 不需要填充_hidl_reply并通过调用_hidl_cb函数将_hidl_reply返回给代理端
::android::hardware::Return<void> _hidl_ret = static_cast<ISupplicant*>(_hidl_this->getImpl().get())-
>terminate();

    (void) _hidl_cb;
```

具体实现函数如下:



5.2.3.3 generates(单个参数且能映射成C++基本类型，包括void类型)

hal文件

hardware/interfaces/wifi/supplicant/1.0/ISupplicant.hal

```
27 interface ISupplicant {
...
33   enum DebugLevel : uint32_t {
34     EXCESSIVE = 0,
35     MSGDUMP = 1,
36     DEBUG = 2,
37     INFO = 3,
38     WARNING = 4,
39     ERROR = 5
40   };
...
129   getDebugLevel() generates (DebugLevel level);
...
159 };
```

代理端

out/soong/.intermediates/hardware/interfaces/wifi/supplicant/1.0/android.hardware.wifi.supplicant@1.0_genc++/gen/android/hardware/wifi/supplicant/1.0/SupplicantAll.cpp

```
::android::hardware::Return<::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel>
BpHwSupplicant::getDebugLevel(){
    ::android::hardware::Return<::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel>
_hidl_out = ::android::hardware::wifi::supplicant::V1_0::BpHwSupplicant::_hidl_getDebugLevel(this, this);

    return _hidl_out;
}



::android::hardware::Return<::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel>
BpHwSupplicant::_hidl_getDebugLevel(::android::hardware::IInterface *_hidl_this,
::android::hardware::details::HidlInstrumentor *_hidl_this_instrumentor) {
    #ifdef __ANDROID_DEBUGGABLE__
    bool mEnableInstrumentation = _hidl_this_instrumentor->isInstrumentationEnabled();
    const auto &mInstrumentationCallbacks = _hidl_this_instrumentor->getInstrumentationCallbacks();
    #else
    (void) _hidl_this_instrumentor;
    #endif // __ANDROID_DEBUGGABLE__
    ::android::ScopedTrace PASTE(___tracer, __LINE__) (ATRACE_TAG_HAL,
"HIDL::ISupplicant::getDebugLevel::client");
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::CLIENT_API_ENTRY, "android.hardware.wifi.supplicant", "1.0",
"ISupplicant", "getDebugLevel", &_hidl_args);
        }
```

```
    }
    #endif // __ANDROID_DEBUGGABLE__

    ::android::hardware::Parcel _hidl_data;
    ::android::hardware::Parcel _hidl_reply;
    ::android::status_t _hidl_err;
    ::android::status_t _hidl_transact_err;
    ::android::hardware::Status _hidl_status;

    ::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel _hidl_out_level;

    _hidl_err = _hidl_data.writeInterfaceToken(BpHwSupplicant::descriptor);
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    _hidl_transact_err = ::android::hardware::IInterface::asBinder(_hidl_this)->transact(5 /* getDebugLevel
*/, _hidl_data, &_hidl_reply, 0);
    if (_hidl_transact_err != ::android::OK)
    {
        _hidl_err = _hidl_transact_err;
        goto _hidl_error;
    }

    _hidl_err = ::android::hardware::readFromParcel(&_hidl_status, _hidl_reply);
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    if (!_hidl_status.isOk()) { return _hidl_status; }

    _hidl_err = _hidl_reply.readUint32((uint32_t *)&_hidl_out_level);
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
```

```
        _hidl_args.push_back((void *)&_hidl_out_level);
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::CLIENT_API_EXIT, "android.hardware.wifi.supplicant", "1.0",
"ISupplicant", "getDebugLevel", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    return ::android::hardware::Return<::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel>
(_hidl_out_level);

_hidl_error:
    _hidl_status.setFromStatusT(_hidl_err);
    return ::android::hardware::Return<::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel>
(_hidl_status);
}
```

代理端填充_hidl_data后调用

```
_hidl_transact_err = ::android::hardware::IInterface::asBinder(_hidl_this)->transact(5 /* getDebugLevel */,
_hidl_data, &_hidl_reply, 0);
```

将数据发送给服务端，之后再调用

```
 _hidl_err = ::android::hardware::readFromParcel(&_hidl_status, _hidl_reply);
```

阻塞等待服务端将_hidl_reply发回来，最后读取里面的值作为BpHwSupplicant::_hidl_getDebugLevel方法的返回值。

服务实现端

out/soong/.intermediates/hardware/interfaces/wifi/supplicant/1.0/android.hardware.wifi.supplicant@1.0_genc++/gen/android/hardware/wifi/supplicant/1.0/SupplicantAll.cpp

```
 ::android::status_t BnHwSupplicant::_hidl_getDebugLevel(
        ::android::hidl::base::V1_0::BnHwBase* _hidl_this,
        const ::android::hardware::Parcel &_hidl_data,
        ::android::hardware::Parcel *_hidl_reply,
        TransactCallback _hidl_cb) {
    #ifdef __ANDROID_DEBUGGABLE__
    bool mEnableInstrumentation = _hidl_this->isInstrumentationEnabled();
    const auto &mInstrumentationCallbacks = _hidl_this->getInstrumentationCallbacks();
    #endif // __ANDROID_DEBUGGABLE__

    ::android::status_t _hidl_err = ::android::OK;
    if (!_hidl_data.enforceInterface(BnHwSupplicant::Pure::descriptor)) {
        _hidl_err = ::android::BAD_TYPE;
        return _hidl_err;
    }

    atrace_begin(ATRACE_TAG_HAL, "HIDL::ISupplicant::getDebugLevel::server");
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::SERVER_API_ENTRY, "android.hardware.wifi.supplicant", "1.0",
"ISupplicant", "getDebugLevel", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    ::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel _hidl_out_level =
static_cast<ISupplicant*>(_hidl_this->getImpl().get())->getDebugLevel();

    ::android::hardware::writeToParcel(::android::hardware::Status::ok(), _hidl_reply);

    _hidl_err = _hidl_reply->writeUint32((uint32_t)_hidl_out_level);
```

```
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

_hidl_error:
    atrace_end(ATRACE_TAG_HAL);
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        _hidl_args.push_back((void *)&_hidl_out_level);
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::SERVER_API_EXIT, "android.hardware.wifi.supplicant", "1.0",
"ISupplicant", "getDebugLevel", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    if (_hidl_err != ::android::OK) { return _hidl_err; }
    _hidl_cb(*_hidl_reply);
    return _hidl_err;
}
```

调用具体实现函数

```
::android::hardware::wifi::supplicant::V1_0::ISupplicant::DebugLevel _hidl_out_level =
static_cast<ISupplicant*>(_hidl_this->getImpl().get())->getDebugLevel();
```

将得到的返回值_hidl_out_level写入_hidl_reply

```
_hidl_err = _hidl_reply->writeUint32((uint32_t)_hidl_out_level);
```

调用_hidl_cb函数把_hidl_reply发送给代理端。这里的_hidl_cb函数类型是TransactCallback，它指向调用BnHwSupplicant::_hidl_getDebugLevel()时传入的匿名函数，即reply_callback

xref: /system/libhwbinder/IPCThreadState.cpp

android-11.0.0_r21 | History | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
1185
1186            auto reply_callback = [&] (auto &replyParcel) {
1187                if (reply_sent) {
1188                    // Reply was sent earlier, ignore it.
1189                    ALOGE("Dropping binder reply, it was sent already.");
1190                    return;
1191                }
1192                reply_sent = true;
1193                if ((tr.flags & TF_ONE_WAY) == 0) {
1194                    replyParcel.setError(NO_ERROR);
1195                    sendReply(replyParcel, 0);
1196                } else {
1197                    ALOGE("Not sending reply in one-way transaction");
1198                }
1199            };
1200
1201            if (tr.target.ptr) {
1202                // We only have a weak reference on the target object, so we must first try to
1203                // safely acquire a strong reference before doing anything else with it.
1204                if (reinterpret_cast<RefBase::weakref_type*>(
1205                        tr.target.ptr)->attemptIncStrong(this)) {
1206                    error = reinterpret_cast<BHwBinder*>(tr.cookie)->transact(tr.code, buffer,
1207                            &reply, tr.flags, reply_callback);
1208                    reinterpret_cast<BHwBinder*>(tr.cookie)->decStrong(this);
1209                } else {
1210                    error = UNKNOWN_TRANSACTION;
1211                }
1212
1213            } else {
1214                error = the_context_object->transact(tr.code, buffer, &reply, tr.flags, reply_callback);
1215            }
```

具体实现函数如下：

xref: /external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/supplicant.cpp

android-11.0.0_r21 | History | Annotate | Line# | Scopes# | Navigate# | Raw | D

```
227
228 Return<ISupplicant::DebugLevel> Supplicant::getDebugLevel()
229 {
230         // TODO: Add SupplicantStatus in this method return for uniformity with
231         // the other methods in supplicant HIDL interface.
232         return (ISupplicant::DebugLevel)wpa_debug_level;
233 }
```

5.2.3.4 generates(多个参数 或 不能映射成C++基本类型的单个参数)

hal文件

hardware/interfaces/wifi/supplicant/1.1/ISupplicant.hal

```
28 interface ISupplicant extends @1.0::ISupplicant {
29     /**
30      * Registers a wireless interface in supplicant.
31      *
32      * @param ifaceInfo Combination of the interface type and name(e.g wlan0).
33      * @return status Status of the operation.
34      *         Possible status codes:
35      *         |SupplicantStatusCode.SUCCESS|,
36      *         |SupplicantStatusCode.FAILURE_ARGS_INVALID|,
37      *         |SupplicantStatusCode.FAILURE_UNKNOWN|,
38      *         |SupplicantStatusCode.FAILURE_IFACE_EXISTS|
39      * @return iface HIDL interface object representing the interface if
40      *         successful, null otherwise.
41      */
42     addInterface(IfaceInfo ifaceInfo)
43         generates (SupplicantStatus status, ISupplicantIface iface);
...
```

```
65 };
```

代理端

out/soong/.intermediates/hardware/interfaces/wifi/supplicant/1.1/android.hardware.wifi.supplicant@1.1_genc++/gen/android/hardware/wifi/supplicant/1.1/SupplicantAll.cpp

```
::android::hardware::Return<void> BpHwSupplicant::addInterface(const
::android::hardware::wifi::supplicant::V1_0::ISupplicant::IfaceInfo& ifaceInfo, addInterface_cb _hidl_cb){
    ::android::hardware::Return<void>  _hidl_out =
::android::hardware::wifi::supplicant::V1_1::BpHwSupplicant::_hidl_addInterface(this, this, ifaceInfo,
_hidl_cb);

    return _hidl_out;
}


::android::hardware::Return<void> BpHwSupplicant::_hidl_addInterface(::android::hardware::IInterface
*_hidl_this, ::android::hardware::details::HidlInstrumentor *_hidl_this_instrumentor, const
::android::hardware::wifi::supplicant::V1_0::ISupplicant::IfaceInfo& ifaceInfo, addInterface_cb _hidl_cb) {
    #ifdef __ANDROID_DEBUGGABLE__
    bool mEnableInstrumentation = _hidl_this_instrumentor->isInstrumentationEnabled();
    const auto &mInstrumentationCallbacks = _hidl_this_instrumentor->getInstrumentationCallbacks();
    #else
    (void) _hidl_this_instrumentor;
    #endif // __ANDROID_DEBUGGABLE__
    ::android::ScopedTrace PASTE(___tracer, __LINE__) (ATRACE_TAG_HAL,
"HIDL::ISupplicant::addInterface::client");
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
```

```
        _hidl_args.push_back((void *)&ifaceInfo);
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::CLIENT_API_ENTRY, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "addInterface", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    ::android::hardware::Parcel _hidl_data;
    ::android::hardware::Parcel _hidl_reply;
    ::android::status_t _hidl_err;
    ::android::status_t _hidl_transact_err;
    ::android::hardware::Status _hidl_status;

    _hidl_err = _hidl_data.writeInterfaceToken(BpHwSupplicant::descriptor);
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    size_t _hidl_ifaceInfo_parent;

    _hidl_err = _hidl_data.writeBuffer(&ifaceInfo, sizeof(ifaceInfo), &_hidl_ifaceInfo_parent);
    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    _hidl_err = writeEmbeddedToParcel(
            ifaceInfo,
            &_hidl_data,
            _hidl_ifaceInfo_parent,
            0 /* parentOffset */);

    if (_hidl_err != ::android::OK) { goto _hidl_error; }

    _hidl_transact_err = ::android::hardware::IInterface::asBinder(_hidl_this)->transact(9 /* addInterface */,
_hidl_data, &_hidl_reply, 0, [&] (::android::hardware::Parcel& _hidl_reply) {
        ::android::hardware::wifi::supplicant::V1_0::SupplicantStatus* _hidl_out_status;
```

```
        ::android::sp<::android::hardware::wifi::supplicant::V1_0::ISupplicantIface> _hidl_out_iface;


        _hidl_err = ::android::hardware::readFromParcel(&_hidl_status, _hidl_reply);
        if (_hidl_err != ::android::OK) { return; }

        if (!_hidl_status.isOk()) { return; }

        size_t _hidl__hidl_out_status_parent;

        _hidl_err = _hidl_reply.readBuffer(sizeof(*_hidl_out_status), &_hidl__hidl_out_status_parent,
const_cast<const void**>(reinterpret_cast<void **>(&_hidl_out_status)));
        if (_hidl_err != ::android::OK) { return; }

        _hidl_err = readEmbeddedFromParcel(
                const_cast<::android::hardware::wifi::supplicant::V1_0::SupplicantStatus &>
(*_hidl_out_status),
                _hidl_reply,
                _hidl__hidl_out_status_parent,
                0 /* parentOffset */);

        if (_hidl_err != ::android::OK) { return; }

        {
            ::android::sp<::android::hardware::IBinder> _hidl_binder;
            _hidl_err = _hidl_reply.readNullableStrongBinder(&_hidl_binder);
            if (_hidl_err != ::android::OK) { return; }

            _hidl_out_iface =
::android::hardware::fromBinder<::android::hardware::wifi::supplicant::V1_0::ISupplicantIface,::android::hardw
are::wifi::supplicant::V1_0::BpHwSupplicantIface,::android::hardware::wifi::supplicant::V1_0::BnHwSupplicantIf
ace>(_hidl_binder);
        }
```

```
        _hidl_cb(*_hidl_out_status, _hidl_out_iface);

        #ifdef __ANDROID_DEBUGGABLE__
        if (UNLIKELY(mEnableInstrumentation)) {
            std::vector<void *> _hidl_args;
            _hidl_args.push_back((void *)_hidl_out_status);
            _hidl_args.push_back((void *)&_hidl_out_iface);
            for (const auto &callback: mInstrumentationCallbacks) {
                callback(InstrumentationEvent::CLIENT_API_EXIT, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "addInterface", &_hidl_args);
            }
        }
        #endif // __ANDROID_DEBUGGABLE__

    });
    if (_hidl_transact_err != ::android::OK) {
        _hidl_err = _hidl_transact_err;
        goto _hidl_error;
    }

    if (!_hidl_status.isOk()) { return _hidl_status; }
    return ::android::hardware::Return<void>();

_hidl_error:
    _hidl_status.setFromStatusT(_hidl_err);
    return ::android::hardware::Return<void>(_hidl_status);
}
```

代理端在填充_hidl_data后调用_hidl_transact_err = ::android::hardware::IInterface::asBinder(_hidl_this)->transact(9 /* addInterface */, _hidl_data, &_hidl_reply, 0, [&] (::android::hardware::Parcel& _hidl_reply) { ... });额外传入了一个匿名函数指针记作A，_hidl_reply是A函数的参数，在A函数里面调用_hidl_err = ::android::hardware::readFromParcel(&_hidl_status, _hidl_reply);等待服务实现端将_hidl_reply发送过来，然后读取里面的数据尝试将其转换成SupplicantStatus类型的数据，再读取_hidl_reply的数据并尝试将其转换成BpHwSupplicantIface类型的数据，然后调用函数指针_hidl_cb(*_hidl_out_status, _hidl_out_iface)将二者返回。这里的_hidl_cb是addInterface_cb类型的函数指针，指向了调用BpHwSupplicant::_hidl_addInterface时传入的回调函数(记作B函数)。在B中可以将从服务实现端传入的_hidl_out_status, _hidl_out_iface参数进行保存。

服务实现端

out/soong/.intermediates/hardware/interfaces/wifi/supplicant/1.1/android.hardware.wifi.supplicant@1.1_genc++/gen/android/hardware/wifi/supplicant/1.1/SupplicantAll.cpp

```
::android::status_t BnHwSupplicant::_hidl_addInterface(
        ::android::hidl::base::V1_0::BnHwBase* _hidl_this,
        const ::android::hardware::Parcel &_hidl_data,
        ::android::hardware::Parcel *_hidl_reply,
        TransactCallback _hidl_cb) {
    #ifdef __ANDROID_DEBUGGABLE__
    bool mEnableInstrumentation = _hidl_this->isInstrumentationEnabled();
    const auto &mInstrumentationCallbacks = _hidl_this->getInstrumentationCallbacks();
    #endif // __ANDROID_DEBUGGABLE__

    ::android::status_t _hidl_err = ::android::OK;
    if (!_hidl_data.enforceInterface(BnHwSupplicant::Pure::descriptor)) {
        _hidl_err = ::android::BAD_TYPE;
        return _hidl_err;
    }
```

```
    ::android::hardware::wifi::supplicant::V1_0::ISupplicant::IfaceInfo* ifaceInfo;

    size_t _hidl_ifaceInfo_parent;

    _hidl_err = _hidl_data.readBuffer(sizeof(*ifaceInfo), &_hidl_ifaceInfo_parent,  const_cast<const void**>
(reinterpret_cast<void **>(&ifaceInfo)));
    if (_hidl_err != ::android::OK) { return _hidl_err; }

    _hidl_err = readEmbeddedFromParcel(
            const_cast<::android::hardware::wifi::supplicant::V1_0::ISupplicant::IfaceInfo &>(*ifaceInfo),
            _hidl_data,
            _hidl_ifaceInfo_parent,
            0 /* parentOffset */);

    if (_hidl_err != ::android::OK) { return _hidl_err; }

    atrace_begin(ATRACE_TAG_HAL, "HIDL::ISupplicant::addInterface::server");
    #ifdef __ANDROID_DEBUGGABLE__
    if (UNLIKELY(mEnableInstrumentation)) {
        std::vector<void *> _hidl_args;
        _hidl_args.push_back((void *)ifaceInfo);
        for (const auto &callback: mInstrumentationCallbacks) {
            callback(InstrumentationEvent::SERVER_API_ENTRY, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "addInterface", &_hidl_args);
        }
    }
    #endif // __ANDROID_DEBUGGABLE__

    bool _hidl_callbackCalled = false;

    ::android::hardware::Return<void> _hidl_ret = static_cast<ISupplicant*>(_hidl_this->getImpl().get())-
>addInterface(*ifaceInfo, [&](const auto &_hidl_out_status, const auto &_hidl_out_iface) {
```

```
        if (_hidl_callbackCalled) {
            LOG_ALWAYS_FATAL("addInterface: _hidl_cb called a second time, but must be called once.");
        }
        _hidl_callbackCalled = true;

        ::android::hardware::writeToParcel(::android::hardware::Status::ok(), _hidl_reply);

        size_t _hidl__hidl_out_status_parent;

        _hidl_err = _hidl_reply->writeBuffer(&_hidl_out_status, sizeof(_hidl_out_status),
&_hidl__hidl_out_status_parent);
        if (_hidl_err != ::android::OK) { goto _hidl_error; }

        _hidl_err = writeEmbeddedToParcel(
                _hidl_out_status,
                _hidl_reply,
                _hidl__hidl_out_status_parent,
                0 /* parentOffset */);

        if (_hidl_err != ::android::OK) { goto _hidl_error; }

        if (_hidl_out_iface == nullptr) {
            _hidl_err = _hidl_reply->writeStrongBinder(nullptr);
        } else {
            ::android::sp<::android::hardware::IBinder> _hidl_binder =
::android::hardware::getOrCreateCachedBinder(_hidl_out_iface.get());
            if (_hidl_binder.get() != nullptr) {
                _hidl_err = _hidl_reply->writeStrongBinder(_hidl_binder);
            } else {
                _hidl_err = ::android::UNKNOWN_ERROR;
            }
        }
        if (_hidl_err != ::android::OK) { goto _hidl_error; }
```

```
    _hidl_error:
        atrace_end(ATRACE_TAG_HAL);
        #ifdef __ANDROID_DEBUGGABLE__
        if (UNLIKELY(mEnableInstrumentation)) {
            std::vector<void *> _hidl_args;
            _hidl_args.push_back((void *)&_hidl_out_status);
            _hidl_args.push_back((void *)&_hidl_out_iface);
            for (const auto &callback: mInstrumentationCallbacks) {
                callback(InstrumentationEvent::SERVER_API_EXIT, "android.hardware.wifi.supplicant", "1.1",
"ISupplicant", "addInterface", &_hidl_args);
            }
        }
        #endif // __ANDROID_DEBUGGABLE__

        if (_hidl_err != ::android::OK) { return; }
        _hidl_cb(*_hidl_reply);
    });

    _hidl_ret.assertOk();
    if (!_hidl_callbackCalled) {
        LOG_ALWAYS_FATAL("addInterface: _hidl_cb not called, but must be called once.");
    }

    return _hidl_err;
}
```

服务端调用
::android::hardware::Return<void> _hidl_ret = static_cast<ISupplicant*>(_hidl_this->getImpl().get())->addInterface(*ifaceInfo, [&](const auto &_hidl_out_status, const auto &_hidl_out_iface) { ... });额外传入了一个匿名函数记作C。

xref: /external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/supplicant.cpp

android-11.0.0_r21 | History | Annotate | Line# | Scopes# | Navigate# | Raw

```
170
171 Return<void> Supplicant::addInterface(
172     const IfaceInfo& iface_info, addInterface_cb _hidl_cb)
173 {
174         return validateAndCall(
175             this, SupplicantStatusCode::FAILURE_IFACE_INVALID,
176             &Supplicant::addInterfaceInternal, _hidl_cb, iface_info);
177 }
```

```
256  std::pair<SupplicantStatus, sp<ISupplicantIface>>
257  Supplicant::addInterfaceInternal(const IfaceInfo& iface_info)
258  {
259          android::sp<ISupplicantIface> iface;
260
261          // Check if required |ifname| argument is empty.
262          if (iface_info.name.empty()) {
263                  return {{SupplicantStatusCode::FAILURE_ARGS_INVALID, ""}, {}};
264          }
265          // Try to get the wpa_supplicant record for this iface, return
266          // the iface object with the appropriate status code if it exists.
267          SupplicantStatus status;
268          std::tie(status, iface) = getInterfaceInternal(iface_info);
269          if (status.code == SupplicantStatusCode::SUCCESS) {
270                  return {{SupplicantStatusCode::FAILURE_IFACE_EXISTS, ""},
271                          iface};
272          }
273
274          struct wpa_interface iface_params = {};
275          iface_params.driver = kIfaceDriverName;
276          if (iface_info.type == IfaceType::P2P) {
277                  if (ensureConfigFileExists(
278                          kP2pIfaceConfPath, kOldP2pIfaceConfPath) != 0) {
279                          wpa_printf(
280                              MSG_ERROR, "Conf file does not exists: %s",
281                              kP2pIfaceConfPath);
282                          return {{SupplicantStatusCode::FAILURE_UNKNOWN,
283                                   "Conf file does not exist"},
284                                  {}};
285                  }
286                  iface_params.confname = kP2pIfaceConfPath;
287                  int ret = access(kP2pIfaceConfOverlayPath, R_OK);
288                  if (ret == 0) {
289                          iface_params.confanother = kP2pIfaceConfOverlayPath;
290                  }
291          } else {
292                  if (ensureConfigFileExists(
293                          kStaIfaceConfPath, kOldStaIfaceConfPath) != 0) {
294                          wpa_printf(
295                              MSG_ERROR, "Conf file does not exists: %s",
296                              kStaIfaceConfPath);
297                          return {{SupplicantStatusCode::FAILURE_UNKNOWN,
298                                   "Conf file does not exist"},
299                                  {}};
300                  }
301                  iface_params.confname = kStaIfaceConfPath;
302                  int ret = access(kStaIfaceConfOverlayPath, R_OK);
303                  if (ret == 0) {
304                          iface_params.confanother = kStaIfaceConfOverlayPath;
305                  }
306          }
307          iface_params.ifname = iface_info.name.c_str();
308          struct wpa_supplicant* wpa_s =
309              wpa_supplicant_add_iface(wpa_global_, &iface_params, NULL);
310          if (!wpa_s) {
```

```
311             return {{SupplicantStatusCode::FAILURE_UNKNOWN, ""}, {}};
312         }
313         // The supplicant core creates a corresponding hidl object via
314         // HidlManager when |wpa_supplicant_add_iface| is called.
315         return getInterfaceInternal(iface_info);
316     }
```

xref: /external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/hidl_return_util.h

android-11.0.0_r21 | History | Annotate | Line# | Scopes# | Navigate# | Raw | D

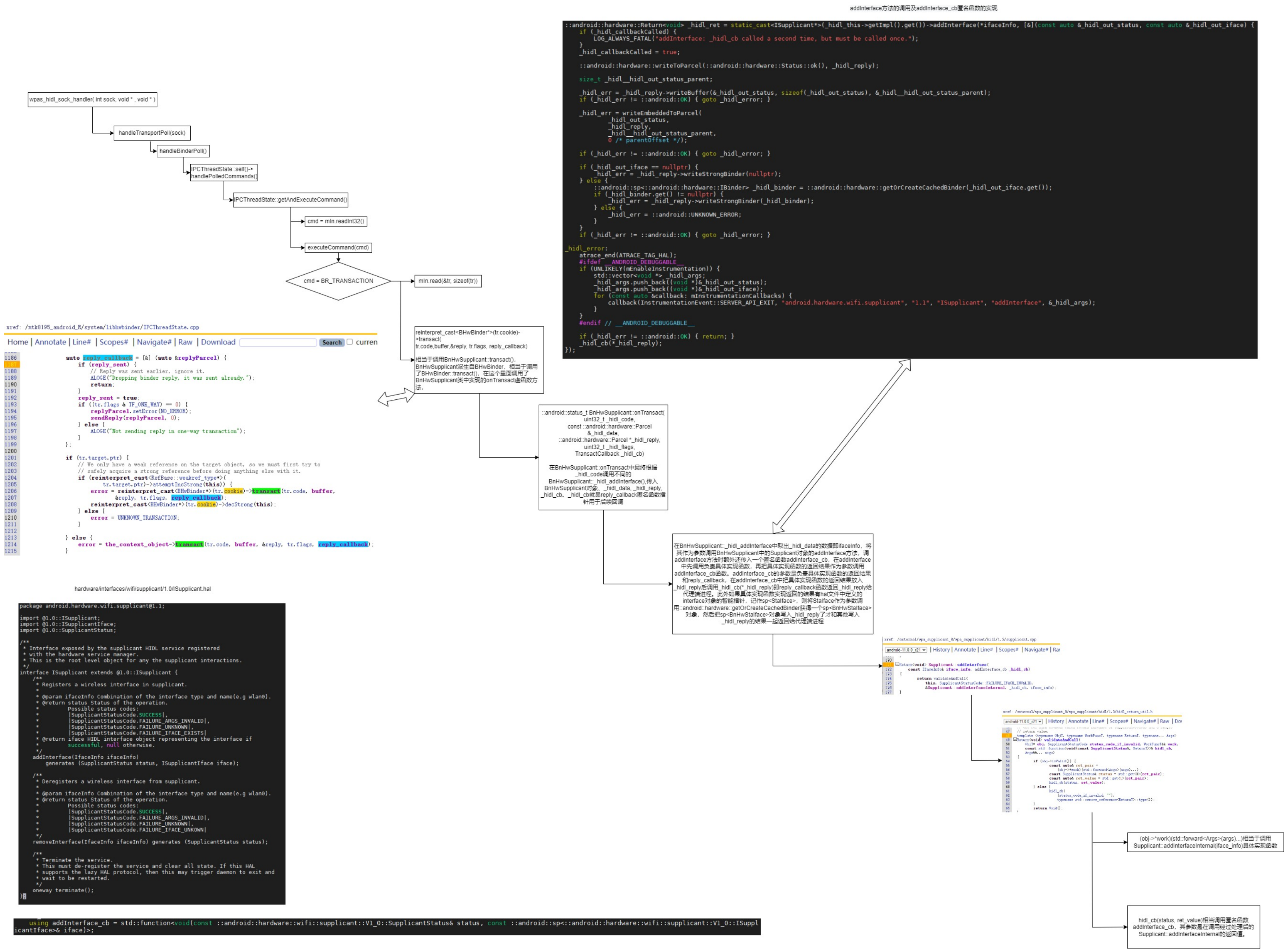
```
49  Return<void> validateAndCall(
50      ObjT* obj, SupplicantStatusCode status_code_if_invalid, WorkFuncT&& work,
51      const std::function<void(const SupplicantStatus&, ReturnT)>& hidl_cb,
52      Args&&... args)
53  {
54          if (obj->isValid()) {
55                  const auto& ret_pair =
56                      (obj->*work)(std::forward<Args>(args)...);
57                  const SupplicantStatus& status = std::get<0>(ret_pair);
58                  const auto& ret_value = std::get<1>(ret_pair);
59                  hidl_cb(status, ret_value);
60          } else {
61                  hidl_cb(
62                      {status_code_if_invalid, ""},
63                      typename std::remove_reference<ReturnT>::type());
64          }
65          return Void();
66  }
```

const auto& ret_pair = (obj->*work)(std::forward<Args>(args)...); 相当于调用了具体的实现函数Supplicant::addInterfaceInternal
接着从调用具体实现函数返回的ret_pair中取出SupplicantStatus类型的status对象和sp<ISupplicantIface>类型的ret_value对象(实际上ret_value指向的是一个ISupplicantStaIface对象，ISupplicantStaIface派生自ISupplicantIface)，然后将二者作为参数调用hidl_cb函数指针。hidl_cb其实指向了前面传入的匿名函数C，在匿名函数C中将传入的status写入_hidl_reply，然后使用传入的ret_value对象调用::android::sp<::android::hardware::IBinder> _hidl_binder = ::android::hardware::getOrCreateCachedBinder(_hidl_out_iface.get())得到一个sp<BnHwSupplicantIface>对象(实际指向的的是一个BnHwSupplicantStaIface对象)，然后将得到的sp<BnHwSupplicantIface>对象写入_hidl_reply，最后调用_hidl_cb(*_hidl_reply);将_hidl_reply返回给代理端，与前面一样，_hidl_cb的类型是TransactCallback，它指向调用BnHwSupplicant::_hidl_getDebugLevel()时传入的匿名函数，即在reply_callback：

xref: /system/libhwbinder/IPCThreadState.cpp

android-11.0.0_r21 | History | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
1185
1186            auto reply_callback = [&] (auto &replyParcel) {
1187                if (reply_sent) {
1188                    // Reply was sent earlier, ignore it.
1189                    ALOGE("Dropping binder reply, it was sent already.");
1190                    return;
1191                }
1192                reply_sent = true;
1193                if ((tr.flags & TF_ONE_WAY) == 0) {
1194                    replyParcel.setError(NO_ERROR);
1195                    sendReply(replyParcel, 0);
1196                } else {
1197                    ALOGE("Not sending reply in one-way transaction");
1198                }
1199            };
1200
1201            if (tr.target.ptr) {
1202                // We only have a weak reference on the target object, so we must first try to
1203                // safely acquire a strong reference before doing anything else with it.
1204                if (reinterpret_cast<RefBase::weakref_type*>(
1205                        tr.target.ptr)->attemptIncStrong(this)) {
1206                    error = reinterpret_cast<BHwBinder*>(tr.cookie)->transact(tr.code, buffer,
1207                            &reply, tr.flags, reply_callback);
1208                    reinterpret_cast<BHwBinder*>(tr.cookie)->decStrong(this);
1209                } else {
1210                    error = UNKNOWN_TRANSACTION;
1211                }
1212
1213            } else {
1214                error = the_context_object->transact(tr.code, buffer, &reply, tr.flags, reply_callback);
1215            }
```


addInterface方法的调用及addInterface_cb匿名函数的实现
wpas_hidl_sock_handler(int sock, void * , void *)
handleTransportPoll(sock)
handleBinderPoll()
IPCThreadState::self()->handlePolledCommands()
IPCThreadState::getAndExecuteCommand()
cmd = mIn.readInt32()
executeCommand(cmd)
cmd = BR_TRANSACTION
mIn.read(&tr, sizeof(tr))
reinterpret_cast<BHwBinder*>(tr.cookie)->transact(
tr.code,buffer,&reply, tr.flags, reply_callback)
相当于调用BnHwSupplicant::transact()，BnHwSupplicant派生自BHwBinder，相当于调用了BHwBinder::transact()，在这个里面调用了BnHwSupplicant类中实现的onTransact虚函数方法，
xref: /mtk8195_android_R/system/libhwbinder/IPCThreadState.cpp
::android::status_t BnHwSupplicant::onTransact(
uint32_t _hidl_code,
const ::android::hardware::Parcel
&_hidl_data,
::android::hardware::Parcel *_hidl_reply,
uint32_t _hidl_flags,
TransactCallback _hidl_cb)
在BnHwSupplicant::onTransact中最终根据_hidl_code调用不同的BnHwSupplicant::_hidl_addInterface(),传入BnHwSupplicant对象、_hidl_data, _hidl_reply, _hidl_cb。_hidl_cb就是reply_callback匿名函数指针用于后续回调
在BnHwSupplicant::_hidl_addInterface中取出_hidl_data的数据即ifaceInfo，将其作为参数调用BnHwSupplicant中的Supplicant对象的addInterface方法，调addInterface方法时额外还传入一个匿名函数addInterface_cb，在addInterface中先调用负责具体实现函数，再把具体实现函数的返回结果作为参数调用addInterface_cb函数。addInterface_cb的参数是负责具体实现函数的返回结果和reply_callback，在addInterface_cb中把具体实现函数的返回结果放入_hidl_reply后调用_hidl_cb(*_hidl_reply)即reply_callback函数返回_hidl_reply给代理端进程。此外如果具体实现函数实现返回的结果有hal文件中定义的interface对象的智能指针，记作sp<StaIface>，则将StaIface作为参数调用::android::hardware::getOrCreateCachedBinder获得一个sp<BnHwStaIface>对象，然后把sp<BnHwStaIface>对象写入_hidl_reply了才和其他写入_hidl_reply的结果一起返回给代理端进程
hardware/interfaces/wifi/supplicant/1.0/ISupplicant.hal
(obj->*work)(std::forward<Args>(args)...)相当于调用Supplicant::addInterfaceInternal(iface_info)具体实现函数
hidl_cb(status, ret_value)相当调用匿名函数addInterface_cb，其参数是在调用经过处理后的Supplicant::addInterfaceInternal的返回值。

5.2.4 部分关键的hidl接口具体实现函数

5.2.3.3 Supplicant::addInterfaceInternal

5.2.3.4 StaIface::addNetworkInternal

5.2.3.5 StaNetwork::setEapMethod

5.2.3.6 StaNetwork::setSsid

StaNetwork::setKeyMgmt

StaNetwork::setKeyMgmt_1_2

StaNetwork::setKeyMgmt_1_3

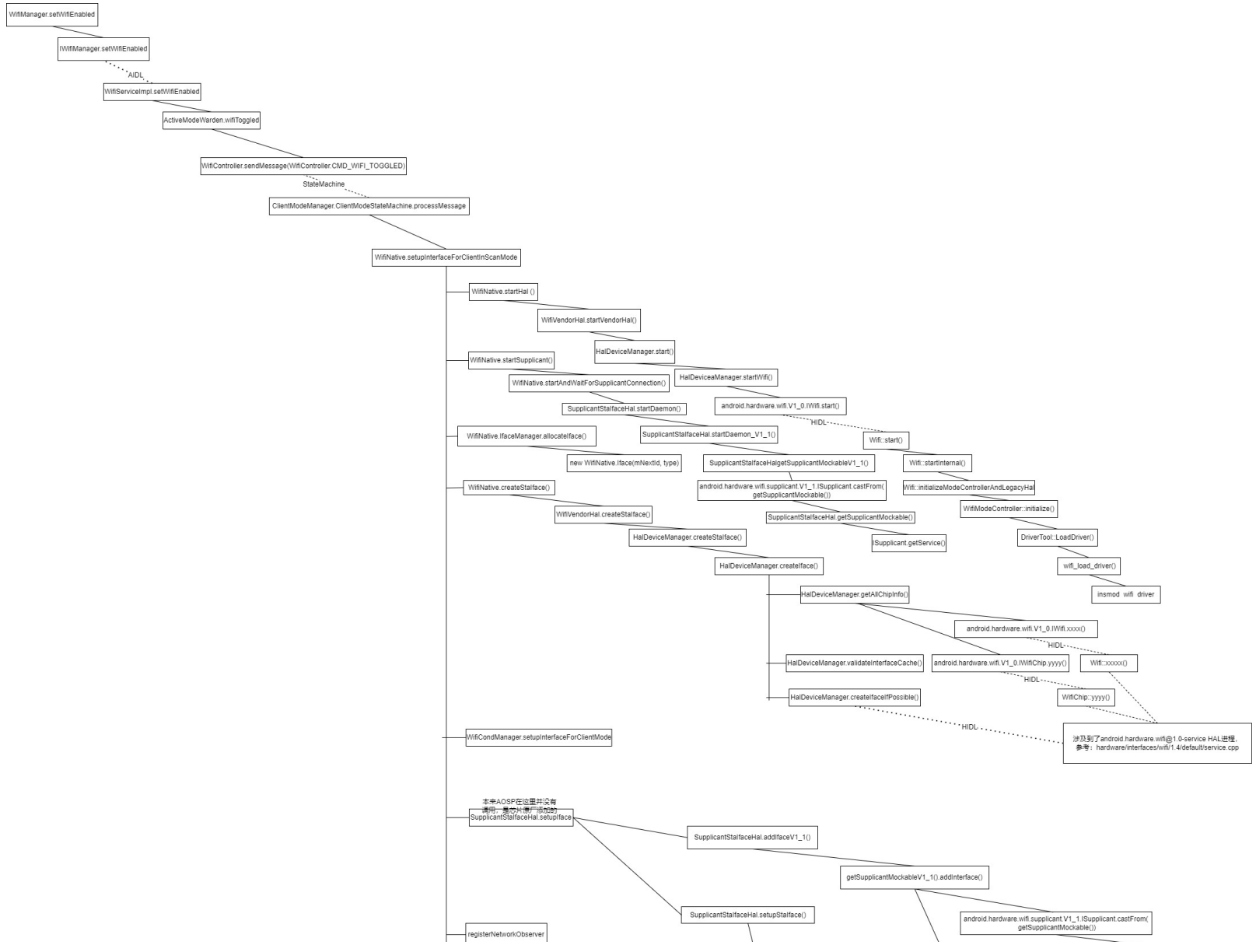
6.app-frameworks-hal主线调用关系

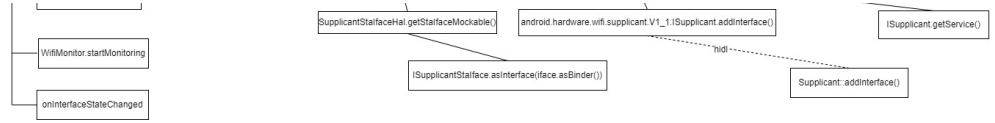
6.1 打开wifi

6.1.1 WifiManager.setWifiEnabled

WifiManager.getWifiState()

WifiManager.setWifiEnabled(true/false)


WifiManager.setWifiEnabled
IWifiManager.setWifiEnabled
AIDL
WifiServiceImpl.setWifiEnabled
ActiveModeWarden.wifiToggled
WifiController.sendMessage(WifiController.CMD_WIFI_TOGGLED)
StateMachine
ClientModeManager.ClientModeStateMachine.processMessage
WifiNative.setupInterfaceForClientInScanMode
WifiNative.startHal ()
WifiVendorHal.startVendorHal()
HalDeviceManager.start()
WifiNative.startSupplicant()
HalDeviceaManager.startWifi()
WifiNative.startAndWaitForSupplicantConnection()
SupplicantStaIfaceHal.startDaemon()
android.hardware.wifi.V1_0.IWifi.start()
HIDL
WifiNative.IfaceManager.allocateIface()
SupplicantStaIfaceHal.startDaemon_V1_1()
Wifi::start()
new WifiNative.Iface(mNextId, type)
SupplicantStaIfaceHalgetSupplicantMockableV1_1()
Wifi::startInternal()
WifiNative.createStaIface()
android.hardware.wifi.supplicant.V1_1.ISupplicant.castFrom(
getSupplicantMockable())
Wifi::initializeModeControllerAndLegacyHal
WifiVendorHal.createStaIface()
SupplicantStaIfaceHal.getSupplicantMockable()
WifiModeController::initialize()
HalDeviceManager.createStaIface()
ISupplicant.getService()
DriverTool::LoadDriver()
HalDeviceManager.createIface()
wifi_load_driver()
HalDeviceManager.getAllChipInfo()
insmod wifi driver
android.hardware.wifi.V1_0.IWifi.xxxx()
HIDL
HalDeviceManager.validateInterfaceCache()
android.hardware.wifi.V1_0.IWifiChip.yyyy()
Wifi::xxxxx()
HIDL
HalDeviceManager.createIfaceIfPossible()
WifiChip::yyyy()
HIDL
涉及到了android.hardware.wifi@1.0-service HAL进程，
参考：hardware/interfaces/wifi/1.4/default/service.cpp
WifiCondManager.setupInterfaceForClientMode
本来AOSP在这里并没有
调用，是芯片原厂添加的
SupplicantStaIfaceHal.setupIface
SupplicantStaIfaceHal.addIfaceV1_1()
getSupplicantMockableV1_1().addInterface()
SupplicantStaIfaceHal.setupStaIface()
android.hardware.wifi.supplicant.V1_1.ISupplicant.castFrom(
getSupplicantMockable())
registerNetworkObserver



6.1. wifi驱动装载

```
WIFI_MODULE

WIFI_DRIVER


WPA_SUPPLICANT_VERSION          := VER_0_8_X
BOARD_WPA_SUPPLICANT_DRIVER := NL80211
BOARD_HOSTAPD_DRIVER                := NL80211



BOARD_WPA_SUPPLICANT_PRIVATE_LIB
BOARD_HOSTAPD_PRIVATE_LIB
lib_driver_cmd_multi/lib_driver_cmd_qcom/lib_driver_cmd_mtk/lib_driver_cmd_bcmdhd_ampak/lib_driver_cmd_bcmdhd_
usi/lib_driver_cmd_nl80211/lib_driver_cmd_rtl




WIFI_DRIVER_MODULE_PATH
WIFI_DRIVER_MODULE_NAME
```

```
hardware/interfaces/wifi/1.x/default/Android.mk
android.hardware.wifi@1.0-service


libwifi-hal.so
frameworks/opt/net/wifi/libwifi_hal/wifi_hal_common.cpp
wifi_load_driver


libwifi-hal-common-ext.so
wifi_load_driver_ext
```

6.2 扫描AP

WifiManager.startScan()

WifiManager.getScanResults()

wpa_supplicant_req_scan

6.3 连接AP

6.3. framework配置相关参数，启动认证

WifiManager.addNetwork(WifiConfiguration config);//返回networkid

WifiManager.enableNetwork(networkid, true);

WifiManager.connect()

WifiManager.getConnectionInfo();

SupplicantStaIfaceHal对象的connectToNetwork方法定义在：
frameworks/opt/net/wifi/service/java/com/android/server/wifi/SupplicantStaIfaceHal.java

```java
    /**
     * Add the provided network configuration to wpa_supplicant and initiate connection to it.
     * This method does the following:
     * 1. If |config| is different to the current supplicant network, removes all supplicant
     * networks and saves |config|.
     * 2. Select the new network in wpa_supplicant.
     *
     * @param ifaceName Name of the interface.
     * @param config WifiConfiguration parameters for the provided network.
     * @return {@code true} if it succeeds, {@code false} otherwise
     */
    public boolean connectToNetwork(@NonNull String ifaceName, @NonNull WifiConfiguration config) {
        synchronized (mLock) {
            logd("connectToNetwork " + config.getKey());
            WifiConfiguration currentConfig = getCurrentNetworkLocalConfig(ifaceName);
            if (WifiConfigurationUtil.isSameNetwork(config, currentConfig)) {
                String networkSelectionBSSID = config.getNetworkSelectionStatus()
                        .getNetworkSelectionBSSID();
                String networkSelectionBSSIDCurrent =
                        currentConfig.getNetworkSelectionStatus().getNetworkSelectionBSSID();
                if (Objects.equals(networkSelectionBSSID, networkSelectionBSSIDCurrent)) {
                    logd("Network is already saved, will not trigger remove and add operation.");
                } else {
                    logd("Network is already saved, but need to update BSSID.");
                    if (!setCurrentNetworkBssid(
                            ifaceName,
                            config.getNetworkSelectionStatus().getNetworkSelectionBSSID())) {
                        loge("Failed to set current network BSSID.");
                        return false;
                    }
                    mCurrentNetworkLocalConfigs.put(ifaceName, new WifiConfiguration(config));
                }
            } else {
                mCurrentNetworkRemoteHandles.remove(ifaceName);
                mCurrentNetworkLocalConfigs.remove(ifaceName);
                if (!removeAllNetworks(ifaceName)) {
                    loge("Failed to remove existing networks");
                    return false;
                }
                Pair<SupplicantStaNetworkHal, WifiConfiguration> pair =
```

```
977                         addNetworkAndSaveConfig(ifaceName, config);
978                     if (pair == null) {
979                         loge("Failed to add/save network configuration: " + config.getKey());
980                         return false;
981                     }
982                     mCurrentNetworkRemoteHandles.put(ifaceName, pair.first);
983                     mCurrentNetworkLocalConfigs.put(ifaceName, pair.second);
984                 }
985                 SupplicantStaNetworkHal networkHandle =
986                         checkSupplicantStaNetworkAndLogFailure(ifaceName, "connectToNetwork");
987                 if (networkHandle == null) {
988                     loge("No valid remote network handle for network configuration: "
989                             + config.getKey());
990                     return false;
991                 }
992
993                 PmkCacheStoreData pmkData = mPmkCacheEntries.get(config.networkId);
994                 if (pmkData != null
995                         && !WifiConfigurationUtil.isConfigForPskNetwork(config)
996                         && pmkData.expirationTimeInSec > mClock.getElapsedSinceBootMillis() / 1000) {
997                     logi("Set PMK cache for config id " + config.networkId);
998                     if (networkHandle.setPmkCache(pmkData.data)) {
999                         mWifiMetrics.setConnectionPmkCache(true);
1000                    }
1001                }
1002
1003                if (!networkHandle.select()) {
1004                    loge("Failed to select network configuration: " + config.getKey());
1005                    return false;
1006                }
1007                return true;
1008            }
1009        }
```

line 977 调用了 SupplicantStaIfaceHal的addNetworkAndSaveConfig方法来添加并配置一个网络，addNetworkAndSaveConfig方法在定义如下：frameworks/opt/net/wifi/service/java/com/android/server/wifi/SupplicantStaIfaceHal.java

```
900     /**
901      * Add a network configuration to wpa_supplicant.
902      *
903      * @param config Config corresponding to the network.
904      * @return a Pair object including SupplicantStaNetworkHal and WifiConfiguration objects
905      * for the current network.
906      */
907     private Pair<SupplicantStaNetworkHal, WifiConfiguration>
908             addNetworkAndSaveConfig(@NonNull String ifaceName, WifiConfiguration config) {
909         synchronized (mLock) {
910             logi("addSupplicantStaNetwork via HIDL");
911             if (config == null) {
912                 loge("Cannot add NULL network!");
913                 return null;
914             }
915             SupplicantStaNetworkHal network = addNetwork(ifaceName);
916             if (network == null) {
917                 loge("Failed to add a network!");
918                 return null;
919             }
920             boolean saveSuccess = false;
921             try {
922                 saveSuccess = network.saveWifiConfiguration(config);
923             } catch (IllegalArgumentException e) {
924                 Log.e(TAG, "Exception while saving config params: " + config, e);
925             }
926             if (!saveSuccess) {
927                 loge("Failed to save variables for: " + config.getKey());
928                 if (!removeAllNetworks(ifaceName)) {
929                     loge("Failed to remove all networks on failure.");
930                 }
931                 return null;
932             }
933             return new Pair(network, new WifiConfiguration(config));
934         }
935     }
```

line 915调用SupplicantStaIfaceHal的addNetwork来在StaIface中添加一个network，line 922调用SupplicantStaNetworkHal的saveWifiConfiguration方法配置新增的网络接口。

```
284         /**
285          * Save an entire WifiConfiguration to wpa_supplicant via HIDL.
286          *
287          * @param config WifiConfiguration object to be saved.
288          * @return true if succeeds, false otherwise.
289          * @throws IllegalArgumentException on malformed configuration params.
290          */
291         public boolean saveWifiConfiguration(WifiConfiguration config) {
292             synchronized (mLock) {
293                 if (config == null) return false;
294                 /** SSID */
295                 if (config.SSID != null) {
296                     if (!setSsid(NativeUtil.decodeSsid(config.SSID))) {
297                         Log.e(TAG, "failed to set SSID: " + config.SSID);
298                         return false;
299                     }
300                 }
301                 /** BSSID */
302                 String bssidStr = config.getNetworkSelectionStatus().getNetworkSelectionBSSID();
303                 if (bssidStr != null) {
304                     byte[] bssid = NativeUtil.macAddressToByteArray(bssidStr);
305                     if (!setBssid(bssid)) {
306                         Log.e(TAG, "failed to set BSSID: " + bssidStr);
307                         return false;
308                     }
309                 }
310                 /** HiddenSSID */
311                 if (!setScanSsid(config.hiddenSSID)) {
312                     Log.e(TAG, config.SSID + ": failed to set hiddenSSID: " + config.hiddenSSID);
313                     return false;
314                 }
315
316                 /** RequirePMF */
317                 if (!setRequirePmf(config.requirePmf)) {
318                     Log.e(TAG, config.SSID + ": failed to set requirePMF: " + config.requirePmf);
319                     return false;
320                 }
321                 /** Key Management Scheme */
322                 if (config.allowedKeyManagement.cardinality() != 0) {
323                     // Add FT flags if supported.
324                     BitSet keyMgmtMask = addFastTransitionFlags(config.allowedKeyManagement);
325                     // Add SHA256 key management flags.
326                     keyMgmtMask = addSha256KeyMgmtFlags(keyMgmtMask);
```

```
327                 if (!setKeyMgmt(wifiConfigurationToSupplicantKeyMgmtMask(keyMgmtMask))) {
328                     Log.e(TAG, "failed to set Key Management");
329                     return false;
330                 }
331                 // Check and set SuiteB configurations.
332                 if (keyMgmtMask.get(WifiConfiguration.KeyMgmt.SUITE_B_192)
333                         && !saveSuiteBConfig(config)) {
334                     Log.e(TAG, "Failed to set Suite-B-192 configuration");
335                     return false;
336                 }
337             }
338             /** Security Protocol */
339             if (config.allowedProtocols.cardinality() != 0
340                     && !setProto(wifiConfigurationToSupplicantProtoMask(config.allowedProtocols))) {
341                 Log.e(TAG, "failed to set Security Protocol");
342                 return false;
343             }
344             /** Auth Algorithm */
345             if (config.allowedAuthAlgorithms.cardinality() != 0
346                     && !setAuthAlg(wifiConfigurationToSupplicantAuthAlgMask(
347                     config.allowedAuthAlgorithms))) {
348                 Log.e(TAG, "failed to set AuthAlgorithm");
349                 return false;
350             }
351             /** Group Cipher */
352             if (config.allowedGroupCiphers.cardinality() != 0
353                     && (!setGroupCipher(wifiConfigurationToSupplicantGroupCipherMask(
354                     config.allowedGroupCiphers)))) {
355                 Log.e(TAG, "failed to set Group Cipher");
356                 return false;
357             }
358             /** Pairwise Cipher*/
359             if (config.allowedPairwiseCiphers.cardinality() != 0
360                     && !setPairwiseCipher(wifiConfigurationToSupplicantPairwiseCipherMask(
361                     config.allowedPairwiseCiphers))) {
362                 Log.e(TAG, "failed to set PairwiseCipher");
363                 return false;
364             }
365             /** Pre Shared Key */
366             // For PSK, this can either be quoted ASCII passphrase or hex string for raw psk.
367             // For SAE, password must be a quoted ASCII string
368             if (config.preSharedKey != null) {
369                 if (config.allowedKeyManagement.get(WifiConfiguration.KeyMgmt.WAPI_PSK)) {
```

```
370                     if (!setPskPassphrase(config.preSharedKey)) {
371                         Log.e(TAG, "failed to set wapi psk passphrase");
372                         return false;
373                     }
374                 } else if (config.preSharedKey.startsWith("\"")) {
375                     if (config.allowedKeyManagement.get(WifiConfiguration.KeyMgmt.SAE)) {
376                         /* WPA3 case, field is SAE Password */
377                         if (!setSaePassword(
378                                 NativeUtil.removeEnclosingQuotes(config.preSharedKey))) {
379                             Log.e(TAG, "failed to set sae password");
380                             return false;
381                         }
382                     } else {
383                         if (!setPskPassphrase(
384                                 NativeUtil.removeEnclosingQuotes(config.preSharedKey))) {
385                             Log.e(TAG, "failed to set psk passphrase");
386                             return false;
387                         }
388                     }
389                 } else {
390                     if (config.allowedKeyManagement.get(WifiConfiguration.KeyMgmt.SAE)) {
391                         return false;
392                     }
393                     if (!setPsk(NativeUtil.hexStringToByteArray(config.preSharedKey))) {
394                         Log.e(TAG, "failed to set psk");
395                         return false;
396                     }
397                 }
398             }
399
400             ///M: [WAPI] If wapi type is auto selection, may need to update wapi alias list
401             if (WifiConfigurationUtil.isConfigForWapiCertNetwork(config)) {
402                 if (TextUtils.isEmpty(config.enterpriseConfig.getCaCertificateAlias())) {
403                     if (!MtkWapi.updateWapiCertSelList(config)) {
404                         Log.e(TAG, "failed to set wapi certificate selection list");
405                         return false;
406                     }
407                 } else {
408                     if (!MtkWapi.setWapiCertAlias(this, getSupplicantNetworkId(),
409                         config.enterpriseConfig.getCaCertificateAlias())) {
410                         Log.e(TAG, "failed to set alias: " +
411                               config.enterpriseConfig.getCaCertificateAlias());
412                         return false;
```

```
413                 }
414             }
415         }
416         /** Wep Keys */
417         boolean hasSetKey = false;
418         if (config.wepKeys != null) {
419             for (int i = 0; i < config.wepKeys.length; i++) {
420                 if (config.wepKeys[i] != null) {
421                     if (!setWepKey(
422                             i, NativeUtil.hexOrQuotedStringToBytes(config.wepKeys[i]))) {
423                         Log.e(TAG, "failed to set wep_key " + i);
424                         return false;
425                     }
426                     hasSetKey = true;
427                 }
428             }
429         }
430         /** Wep Tx Key Idx */
431         if (hasSetKey) {
432             if (!setWepTxKeyIdx(config.wepTxKeyIndex)) {
433                 Log.e(TAG, "failed to set wep_tx_keyidx: " + config.wepTxKeyIndex);
434                 return false;
435             }
436         }
437         /** metadata: FQDN + ConfigKey + CreatorUid */
438         final Map<String, String> metadata = new HashMap<String, String>();
439         if (config.isPasspoint()) {
440             metadata.put(ID_STRING_KEY_FQDN, config.FQDN);
441         }
442         metadata.put(ID_STRING_KEY_CONFIG_KEY, config.getKey());
443         metadata.put(ID_STRING_KEY_CREATOR_UID, Integer.toString(config.creatorUid));
444         if (!setIdStr(createNetworkExtra(metadata))) {
445             Log.e(TAG, "failed to set id string");
446             return false;
447         }
448         /** UpdateIdentifier */
449         if (config.updateIdentifier != null
450                 && !setUpdateIdentifier(Integer.parseInt(config.updateIdentifier))) {
451             Log.e(TAG, "failed to set update identifier");
452             return false;
453         }
454         // Finish here if no EAP config to set
455         if (config.enterpriseConfig != null
```

```
456                 && config.enterpriseConfig.getEapMethod() != WifiEnterpriseConfig.Eap.NONE) {
457             if (config.enterpriseConfig.getEapMethod() == WifiEnterpriseConfig.Eap.WAPI_CERT) {
458                 /** WAPI certificate suite name*/
459                 String param = config.enterpriseConfig
460                         .getFieldValue(WifiEnterpriseConfig.WAPI_CERT_SUITE_KEY);
461                 if (!TextUtils.isEmpty(param) && !setWapiCertSuite(param)) {
462                     Log.e(TAG, config.SSID + ": failed to set WAPI certificate suite: "
463                             + param);
464                     return false;
465                 }
466                 return true;
467             } else if (!saveWifiEnterpriseConfig(config.SSID, config.enterpriseConfig)) {
468                 return false;
469             }
470         }
471
472         // Now that the network is configured fully, start listening for callback events.
473         mISupplicantStaNetworkCallback =
474                 new SupplicantStaNetworkHalCallback(config.networkId, config.SSID);
475         if (!registerCallback(mISupplicantStaNetworkCallback)) {
476             Log.e(TAG, "Failed to register callback");
477             return false;
478         }
479         return true;
480     }
481 }
```

SupplicantStaNetworkHal的saveWifiConfiguration方法根据WifiConfiguration调用一系列的hidl接口向wpa_supplicant发送wifi网络配置数据，在467行调用了SupplicantStaNetworkHal的saveWifiEnterpriseConfig方法来把输**企业级**认证所需数据从WifiConfiguration传到wpa_supplicant，其定义如下：

```
    /**
     * Save network variables from the provided WifiEnterpriseConfig object to wpa_supplicant.
     *
     * @param ssid      SSID of the network. (Used for logging purposes only)
     * @param eapConfig WifiEnterpriseConfig object to be saved.
     * @return true if succeeds, false otherwise.
     */
    private boolean saveWifiEnterpriseConfig(String ssid, WifiEnterpriseConfig eapConfig) {
        synchronized (mLock) {
            if (eapConfig == null) return false;
            /** EAP method */
            if (!setEapMethod(wifiConfigurationToSupplicantEapMethod(eapConfig.getEapMethod()))) {
                Log.e(TAG, ssid + ": failed to set eap method: " + eapConfig.getEapMethod());
                return false;
            }
            /** EAP Phase 2 method */
            if (!setEapPhase2Method(wifiConfigurationToSupplicantEapPhase2Method(
                    eapConfig.getPhase2Method()))) {
                Log.e(TAG, ssid + ": failed to set eap phase 2 method: "
                        + eapConfig.getPhase2Method());
                return false;
            }
            String eapParam = null;
            /** EAP Identity */
            eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.IDENTITY_KEY);
            if (!TextUtils.isEmpty(eapParam)
                    && !setEapIdentity(NativeUtil.stringToByteArrayList(eapParam))) {
                Log.e(TAG, ssid + ": failed to set eap identity: " + eapParam);
                return false;
            }
            /** EAP Anonymous Identity */
            eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.ANON_IDENTITY_KEY);
            if (!TextUtils.isEmpty(eapParam)
                    && !setEapAnonymousIdentity(NativeUtil.stringToByteArrayList(eapParam))) {
                Log.e(TAG, ssid + ": failed to set eap anonymous identity: " + eapParam);
                return false;
            }
            /** EAP Password */
            eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.PASSWORD_KEY);
            if (!TextUtils.isEmpty(eapParam)
                    && !setEapPassword(NativeUtil.stringToByteArrayList(eapParam))) {
                Log.e(TAG, ssid + ": failed to set eap password");
```

```
690                 return false;
691             }
692             /** EAP Client Cert */
693             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.CLIENT_CERT_KEY);
694             if (!TextUtils.isEmpty(eapParam) && !setEapClientCert(eapParam)) {
695                 Log.e(TAG, ssid + ": failed to set eap client cert: " + eapParam);
696                 return false;
697             }
698             /** EAP CA Cert */
699             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.CA_CERT_KEY);
700             if (!TextUtils.isEmpty(eapParam) && !setEapCACert(eapParam)) {
701                 Log.e(TAG, ssid + ": failed to set eap ca cert: " + eapParam);
702                 return false;
703             }
704             /** EAP Subject Match */
705             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.SUBJECT_MATCH_KEY);
706             if (!TextUtils.isEmpty(eapParam) && !setEapSubjectMatch(eapParam)) {
707                 Log.e(TAG, ssid + ": failed to set eap subject match: " + eapParam);
708                 return false;
709             }
710             /** EAP Engine ID */
711             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.ENGINE_ID_KEY);
712             if (!TextUtils.isEmpty(eapParam) && !setEapEngineID(eapParam)) {
713                 Log.e(TAG, ssid + ": failed to set eap engine id: " + eapParam);
714                 return false;
715             }
716             /** EAP Engine */
717             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.ENGINE_KEY);
718             if (!TextUtils.isEmpty(eapParam) && !setEapEngine(
719                     eapParam.equals(WifiEnterpriseConfig.ENGINE_ENABLE) ? true : false)) {
720                 Log.e(TAG, ssid + ": failed to set eap engine: " + eapParam);
721                 return false;
722             }
723             /** EAP Private Key */
724             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.PRIVATE_KEY_ID_KEY);
725             if (!TextUtils.isEmpty(eapParam) && !setEapPrivateKeyId(eapParam)) {
726                 Log.e(TAG, ssid + ": failed to set eap private key: " + eapParam);
727                 return false;
728             }
729             /** EAP Alt Subject Match */
730             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.ALTSUBJECT_MATCH_KEY);
731             if (!TextUtils.isEmpty(eapParam) && !setEapAltSubjectMatch(eapParam)) {
```

```
732                 Log.e(TAG, ssid + ": failed to set eap alt subject match: " + eapParam);
733                 return false;
734             }
735             /** EAP Domain Suffix Match */
736             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.DOM_SUFFIX_MATCH_KEY);
737             if (!TextUtils.isEmpty(eapParam) && !setEapDomainSuffixMatch(eapParam)) {
738                 Log.e(TAG, ssid + ": failed to set eap domain suffix match: " + eapParam);
739                 return false;
740             }
741             /** EAP CA Path*/
742             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.CA_PATH_KEY);
743             if (!TextUtils.isEmpty(eapParam) && !setEapCAPath(eapParam)) {
744                 Log.e(TAG, ssid + ": failed to set eap ca path: " + eapParam);
745                 return false;
746             }
747
748             /** EAP Proactive Key Caching */
749             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.OPP_KEY_CACHING);
750             if (!TextUtils.isEmpty(eapParam)
751                     && !setEapProactiveKeyCaching(eapParam.equals("1") ? true : false)) {
752                 Log.e(TAG, ssid + ": failed to set proactive key caching: " + eapParam);
753                 return false;
754             }
755
756             /**
757              * OCSP (Online Certificate Status Protocol)
758              * For older HAL compatibility, omit this step to avoid breaking
759              * connection flow.
760              */
761             if (getV1_3StaNetwork() != null && !setOcsp(eapConfig.getOcsp())) {
762                 Log.e(TAG, "failed to set ocsp");
763                 return false;
764             }
765             /** EAP ERP */
766             eapParam = eapConfig.getFieldValue(WifiEnterpriseConfig.EAP_ERP);
767             if (!TextUtils.isEmpty(eapParam) && eapParam.equals("1")) {
768                 if (!setEapErp(true)) {
769                     Log.e(TAG, ssid + ": failed to set eap erp");
770                     return false;
771                 }
772             }
773
```

```
774
775                 return true;
776             }
777         }
```

返回之后在SupplicantStaNetworkHal的saveWifiConfiguration方法中在line 475调用SupplicantStaNetworkHal的registerCallback方法注册回调对象，registerCallback方法调用StaNetwork的代理对象，真正实现在wpa_supplicant_8/wpa_supplicant/hidl/1.3/sta_network.cpp中的line 147行。向wpa_supplicant注册回调对象后当wpa_supplicant触发一些事件就会调用这些代理回调对象里的一些方法，回调对象具体实现在SupplicantStaNetworkHal的一个内部类

```
3585
3586    private class SupplicantStaNetworkHalCallback extends ISupplicantStaNetworkCallback.Stub {
3587        /**
3588         * Current configured network's framework network id.
3589         */
3590        private final int mFramewokNetworkId;
3591        /**
3592         * Current configured network's ssid.
3593         */
3594        private final String mSsid;
3595
3596        SupplicantStaNetworkHalCallback(int framewokNetworkId, String ssid) {
3597            mFramewokNetworkId = framewokNetworkId;
3598            mSsid = ssid;
3599        }
3600
3601        @Override
3602        public void onNetworkEapSimGsmAuthRequest(
3603                ISupplicantStaNetworkCallback.NetworkRequestEapSimGsmAuthParams params) {
3604            synchronized (mLock) {
3605                logCallback("onNetworkEapSimGsmAuthRequest");
3606                String[] data = new String[params.rands.size()];
3607                int i = 0;
3608                for (byte[] rand : params.rands) {
3609                    data[i++] = NativeUtil.hexStringFromByteArray(rand);
3610                }
3611                mWifiMonitor.broadcastNetworkGsmAuthRequestEvent(
3612                        mIfaceName, mFramewokNetworkId, mSsid, data);
3613            }
3614        }
3615
3616        @Override
3617        public void onNetworkEapSimUmtsAuthRequest(
3618                ISupplicantStaNetworkCallback.NetworkRequestEapSimUmtsAuthParams params) {
3619            synchronized (mLock) {
3620                logCallback("onNetworkEapSimUmtsAuthRequest");
3621                String randHex = NativeUtil.hexStringFromByteArray(params.rand);
```

```
3622                String autnHex = NativeUtil.hexStringFromByteArray(params.autn);
3623                String[] data = {randHex, autnHex};
3624                mWifiMonitor.broadcastNetworkUmtsAuthRequestEvent(
3625                        mIfaceName, mFrameworkNetworkId, mSsid, data);
3626            }
3627        }
3628
3629        @Override
3630        public void onNetworkEapIdentityRequest() {
3631            synchronized (mLock) {
3632                logCallback("onNetworkEapIdentityRequest");
3633                mWifiMonitor.broadcastNetworkIdentityRequestEvent(
3634                        mIfaceName, mFrameworkNetworkId, mSsid);
3635            }
3636        }
3637    }
```

最后返回到SupplicantStaIfaceHal的connectToNetwork在line 1003 调用SupplicantStaNetworkHal的select方法来选择一个网络并开始认证流程。

6.3. wpa select network

SupplicantStaIfaceHal通过调用StaNetwork代理端的select方法调到了wpa_supplicant中的C++具体实现的select方法：

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/sta_network.cpp

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
653
654 Return<void> StaNetwork::select(select_cb _hidl_cb)
655 {
656         return validateAndCall(
657             this, SupplicantStatusCode::FAILURE_NETWORK_INVALID,
658             &StaNetwork::selectInternal, _hidl_cb);
659 }
```

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/wpa_supplicant/hidl/1.3/sta_network.cpp

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
1857 SupplicantStatus StaNetwork::selectInternal()
1858 {
1859         struct wpa_ssid *wpa_ssid = retrieveNetworkPtr();
1860         if (wpa_ssid->disabled == 2) {
1861                 return {SupplicantStatusCode::FAILURE_UNKNOWN, ""};
1862         }
1863         struct wpa_supplicant *wpa_s = retrieveIfacePtr();
1864         wpa_s->scan_min_time.sec = 0;
1865         wpa_s->scan_min_time.usec = 0;
1866         wpa_supplicant_select_network(wpa_s, wpa_ssid);
1867         return {SupplicantStatusCode::SUCCESS, ""};
1868 }
```

external/wpa_supplicant_8/wpa_supplicant/wpa_supplicant.c中定义了wpa_supplicant_select_network函数，wpa_supplicant_select_network函数经过一番判断最终如果有需要则调用wpa_supplicant_req_scan函数，第二和第三个参数传入的都是0。wpa_supplicant_req_scan函数如下：

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/wpa_supplicant/scan.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download Search current directory

```
1364  /**
1365   * wpa_supplicant_req_scan - Schedule a scan for neighboring access points
1366   * @wpa_s: Pointer to wpa_supplicant data
1367   * @sec: Number of seconds after which to scan
1368   * @usec: Number of microseconds after which to scan
1369   *
1370   * This function is used to schedule a scan for neighboring access points after
1371   * the specified time.
1372   */
1373 void wpa_supplicant_req_scan(struct wpa_supplicant *wpa_s, int sec, int usec)
1374  {
1375          int res;
1376
1377          if (wpa_s->p2p_mgmt) {
1378                  wpa_dbg(wpa_s, MSG_DEBUG,
1379                          "Ignore scan request (%d.%06d sec) on p2p_mgmt interface",
1380                          sec, usec);
1381                  return;
1382          }
1383
1384          res = eloop_deplete_timeout(sec, usec, wpa_supplicant_scan, wpa_s,
1385                                      NULL);
1386          if (res == 1) {
1387                  wpa_dbg(wpa_s, MSG_DEBUG, "Rescheduling scan request: %d.%06d sec",
1388                          sec, usec);
1389          } else if (res == 0) {
1390                  wpa_dbg(wpa_s, MSG_DEBUG, "Ignore new scan request for %d.%06d sec since an earlier request is scheduled to trigger sooner",
1391                          sec, usec);
1392          } else {
1393                  wpa_dbg(wpa_s, MSG_DEBUG, "Setting scan request: %d.%06d sec",
1394                          sec, usec);
1395                  eloop_register_timeout(sec, usec, wpa_supplicant_scan, wpa_s, NULL);
1396          }
1397  }
```

wpa_supplicant_req_scan在epoll队列中注册了一个超时函数，其间隔为0，即待调用完wpa_supplicant_req_scan函数后回到epoll监听时wpa_supplicant_scan函数马上就能得到执行。

external/wpa_supplicant_8/wpa_supplicant/scan.c

```
855  static void wpa_supplicant_scan(void *eloop_ctx, void *timeout_ctx)
856  {
......
954     if (connect_without_scan) {
955         wpa_s->connect_without_scan = NULL;
956         if (ssid) {
957             wpa_printf(MSG_DEBUG, "Start a pre-selected network "
958                    "without scan step");
959             wpa_supplicant_associate(wpa_s, NULL, ssid);
960             return;
961         }
962     }
......
1307    ret = wpa_supplicant_trigger_scan(wpa_s, scan_params);
......
1338  }
```

如果是第一次选择连接某个网络则需要调用*wpa_supplicant_trigger_scan*先触发一次扫描，之后就不用触发扫描直接调用*wpa_supplicant_associate*进行认证关联操作。

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/wpa_supplicant/scan.c

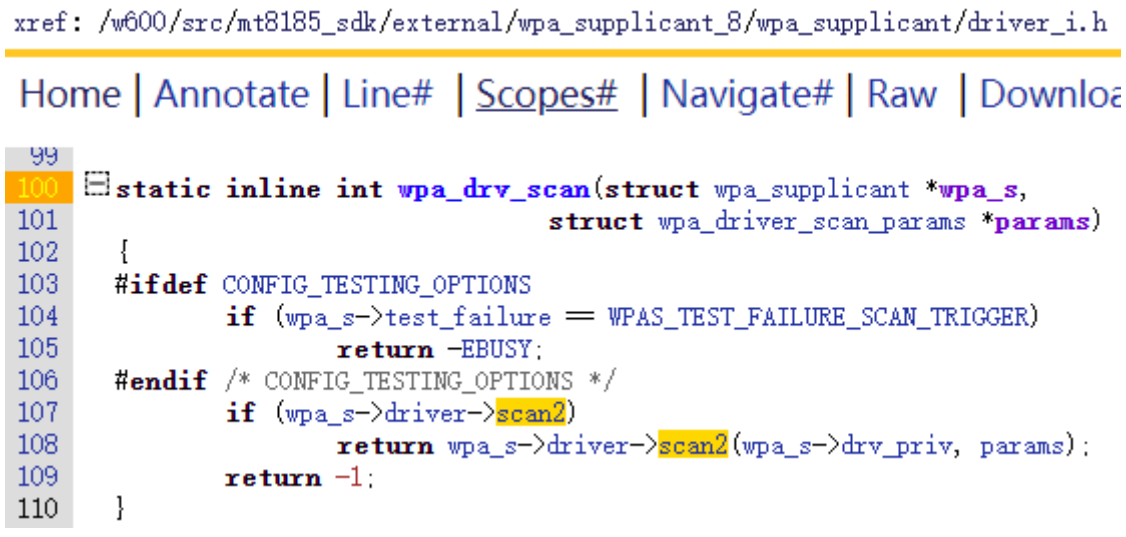
Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
271
272
273   /**
274    * wpa_supplicant_trigger_scan - Request driver to start a scan
275    * @wpa_s: Pointer to wpa_supplicant data
276    * @params: Scan parameters
277    * Returns: 0 on success, -1 on failure
278    */
279  int wpa_supplicant_trigger_scan(struct wpa_supplicant *wpa_s,
280                                  struct wpa_driver_scan_params *params)
281   {
282           struct wpa_driver_scan_params *ctx;
283
284           if (wpa_s->scan_work) {
285                   wpa_dbg(wpa_s, MSG_INFO, "Reject scan trigger since one is already pending");
286                   return -1;
287           }
288
289           ctx = wpa_scan_clone_params(params);
290           if (!ctx ||
291               radio_add_work(wpa_s, 0, "scan", 0, wpas_trigger_scan_cb, ctx) < 0)
292           {
293                   wpa_scan_free_params(ctx);
294                   wpa_msg(wpa_s, MSG_INFO, WPA_EVENT_SCAN_FAILED "ret=-1");
295                   return -1;
296           }
297
298           return 0;
299   }
```

wpas_trigger_scan_cb是一个回调函数

external/wpa_supplicant_8/wpa_supplicant/scan.c

```
182  static void wpas_trigger_scan_cb(struct wpa_radio_work *work, int deinit)
183  {
......
219      ret = wpa_drv_scan(wpa_s, params);
......
270  }
```



addInterface的驱动初始化时wpa_s->driver指向了如下结构体：

scan2是函数指针，指向driver_nl80211_scan2。

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/src/drivers/driver_nl80211.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
11486
11487   const struct wpa_driver_ops wpa_driver_nl80211_ops = {
11488           .name = "nl80211",
11489           .desc = "Linux nl80211/cfg80211",
11490           .get_bssid = wpa_driver_nl80211_get_bssid,
11491           .get_ssid = wpa_driver_nl80211_get_ssid,
11492           .set_key = driver_nl80211_set_key,
11493           .scan2 = driver_nl80211_scan2,
11494           .sched_scan = wpa_driver_nl80211_sched_scan,
11495           .stop_sched_scan = wpa_driver_nl80211_stop_sched_scan,
11496           .get_scan_results2 = wpa_driver_nl80211_get_scan_results,
11497           .abort_scan = wpa_driver_nl80211_abort_scan,
11498           .deauthenticate = driver_nl80211_deauthenticate,
11499           .authenticate = driver_nl80211_authenticate,
11500           .associate = wpa_driver_nl80211_associate,
11501           .global_init = nl80211_global_init,
11502           .global_deinit = nl80211_global_deinit,
11503           .init2 = wpa_driver_nl80211_init,
11504           .deinit = driver_nl80211_deinit,
11505           .get_capa = wpa_driver_nl80211_get_capa,
```

因此wpa_drv_scan最终调用到了driver_nl80211_scan2函数：

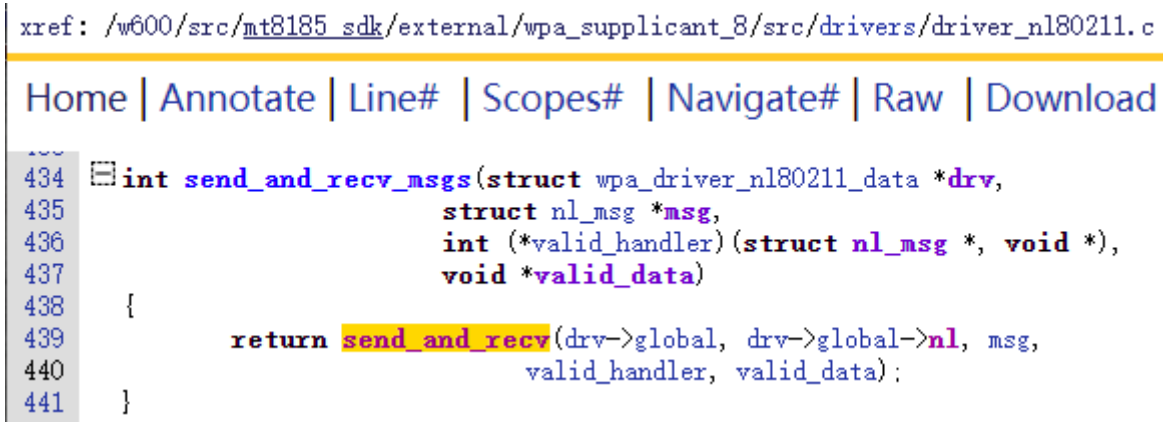
xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/src/drivers/driver_nl80211.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
8928
8929  static int driver_nl80211_scan2(void *priv,
8930                                  struct wpa_driver_scan_params *params)
8931  {
8932          struct i802_bss *bss = priv;
8933  #ifdef CONFIG_DRIVER_NL80211_QCA
8934          struct wpa_driver_nl80211_data *drv = bss->drv;
8935
8936          /*
8937           * Do a vendor specific scan if possible. If only_new_results is
8938           * set, do a normal scan since a kernel (cfg80211) BSS cache flush
8939           * cannot be achieved through a vendor scan. The below condition may
8940           * need to be modified if new scan flags are added in the future whose
8941           * functionality can only be achieved through a normal scan.
8942           */
8943          if (drv->scan_vendor_cmd_avail && !params->only_new_results)
8944                  return wpa_driver_nl80211_vendor_scan(bss, params);
8945  #endif /* CONFIG_DRIVER_NL80211_QCA */
8946          return wpa_driver_nl80211_scan(bss, params);
8947  }
```

external/wpa_supplicant_8/src/drivers/driver_nl80211_scan.c

```
323  int wpa_driver_nl80211_scan(struct i802_bss *bss,
324                 struct wpa_driver_scan_params *params)
325  {
......
371     ret = send_and_recv_msgs(drv, msg, NULL, NULL);
......
413     eloop_cancel_timeout(wpa_driver_nl80211_scan_timeout, drv, drv->ctx);
414     eloop_register_timeout(timeout, 0, wpa_driver_nl80211_scan_timeout,
415                    drv, drv->ctx);
......
421  }
```



驱动回复数据后回调函数调用到do_process_drv_event函数来处理：

external/wpa_supplicant_8/src/drivers/driver_nl80211_event.c

```
2577  static void do_process_drv_event(struct i802_bss *bss, int cmd,
2578                 struct nlattr **tb)
2579  {
......
2607    switch (cmd) {
......
2632    case NL80211_CMD_NEW_SCAN_RESULTS:
2633        wpa_dbg(drv->ctx, MSG_DEBUG,
2634            "nl80211: New scan results available");
2635        if (drv->last_scan_cmd != NL80211_CMD_VENDOR)
2636            drv->scan_state = SCAN_COMPLETED;
2637        drv->scan_complete_events = 1;
2638        if (drv->last_scan_cmd == NL80211_CMD_TRIGGER_SCAN) {
2639            eloop_cancel_timeout(wpa_driver_nl80211_scan_timeout,
2640                         drv, drv->ctx);
2641            drv->last_scan_cmd = 0;
2642        } else {
2643            external_scan_event = 1;
2644        }
2645        send_scan_event(drv, 0, tb, external_scan_event);
2646        break;
......
}
```

external/wpa_supplicant_8/src/drivers/driver_nl80211_event.c

```
1177  static void send_scan_event(struct wpa_driver_nl80211_data *drv, int aborted,
1178                 struct nlattr *tb[], int external_scan)
1179  {
......
1249    wpa_supplicant_event(drv->ctx, EVENT_SCAN_RESULTS, &event);
1250  }
```

external/wpa_supplicant_8/wpa_supplicant/events.c

```
4594  void wpa_supplicant_event(void *ctx, enum wpa_event_type event,
4595              union wpa_event_data *data)
4596  {
......
4719    case EVENT_SCAN_RESULTS:
......
4738        if (wpa_supplicant_event_scan_results(wpa_s, data))
4739            break; /* interface may have been removed */

......
5413  }
```

```
2305
2306  static int wpa_supplicant_event_scan_results(struct wpa_supplicant *wpa_s,
2307                                               union wpa_event_data *data)
2308  {
2309          struct wpa_supplicant *ifs;
2310          int res;
2311
2312          res = _wpa_supplicant_event_scan_results(wpa_s, data, 1, 0);
2313          if (res == 2) {
2314                  /*
2315                   * Interface may have been removed, so must not dereference
2316                   * wpa_s after this.
2317                   */
2318                  return 1;
2319          }
2320
2321          if (res < 0) {
2322                  /*
2323                   * If no scan results could be fetched, then no need to
2324                   * notify those interfaces that did not actually request
2325                   * this scan. Similarly, if scan results started a new operation on this
2326                   * interface, do not notify other interfaces to avoid concurrent
2327                   * operations during a connection attempt.
2328                   */
2329                  return 0;
2330          }
2331
2332          /*
2333           * Check other interfaces to see if they share the same radio. If
2334           * so, they get updated with this same scan info.
2335           */
2336          dl_list_for_each(ifs, &wpa_s->radio->ifaces, struct wpa_supplicant,
2337                           radio_list) {
```

```
2338                    if (ifs != wpa_s) {
2339                            wpa_printf(MSG_DEBUG, "%s: Updating scan results from "
2340                                    "sibling", ifs->ifname);
2341                            res = _wpa_supplicant_event_scan_results(ifs, data, 0,
2342                                                                     res > 0);
2343                            if (res < 0)
2344                                    return 0;
2345                    }
2346            }
2347
2348            return 0;
2349    }
```

external/wpa_supplicant_8/wpa_supplicant/events.c

```
1991  static int _wpa_supplicant_event_scan_results(struct wpa_supplicant *wpa_s,
1992                          union wpa_event_data *data,
1993                          int own_request, int update_only)
1994  {
......
2009    scan_res = wpa_supplicant_get_scan_results(wpa_s,
2010                           data ? &data->scan_info :
2011                           NULL, 1);
......
2145    return wpas_select_network_from_last_scan(wpa_s, 1, own_request);
......
2155  }
```

external/wpa_supplicant_8/wpa_supplicant/events.c

```
2158  static int wpas_select_network_from_last_scan(struct wpa_supplicant *wpa_s,
2159                                                int new_scan, int own_request)
2160  {
......
2179    selected = wpa_supplicant_pick_network(wpa_s, &ssid);
......
2189    if (selected) {
......
2207        if (wpa_supplicant_connect(wpa_s, selected, ssid) < 0) {
2208            wpa_dbg(wpa_s, MSG_DEBUG, "Connect failed");
2209            return -1;
2210        }
......
2219    } else {
......
2301    }
2302    return 0;
2303  }
```

external/wpa_supplicant_8/wpa_supplicant/events.c

```
1661  int wpa_supplicant_connect(struct wpa_supplicant *wpa_s,
1662                struct wpa_bss *selected,
1663                struct wpa_ssid *ssid)
1664  {
......
1714        wpa_supplicant_associate(wpa_s, selected, ssid);
......
1721  }
```

6.3. wpa athenticate&&associate

external/wpa_supplicant_8/wpa_supplicant/wpa_supplicant.c

```
2118  void wpa_supplicant_associate(struct wpa_supplicant *wpa_s,
2119                  struct wpa_bss *bss, struct wpa_ssid *ssid)
2120  {
......
2298    if (radio_add_work(wpa_s, bss ? bss->freq : 0, "connect", 1,
2299                wpas_start_assoc_cb, cwork) < 0) {
2300        os_free(cwork);
2301    }
2302  }
```

external/wpa_supplicant_8/wpa_supplicant/wpa_supplicant.c

```
3388  static void wpas_start_assoc_cb(struct wpa_radio_work *work, int deinit)
3389  {
......
3777    ret = wpa_drv_associate(wpa_s, &params);
......
3869  }
```

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/wpa_supplicant/driver_i.h

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
58
59  static inline int wpa_drv_associate(struct wpa_supplicant *wpa_s,
60                                      struct wpa_driver_associate_params *params)
61  {
62          if (wpa_s->driver->associate) {
63                  return wpa_s->driver->associate(wpa_s->drv_priv, params);
64          }
65          return -1;
66  }
```

xref: /w600/src/mt8185_sdk/external/wpa_supplicant_8/src/drivers/driver_nl80211.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
11487  const struct wpa_driver_ops wpa_driver_nl80211_ops = {
11488          .name = "nl80211",
11489          .desc = "Linux nl80211/cfg80211",
11490          .get_bssid = wpa_driver_nl80211_get_bssid,
11491          .get_ssid = wpa_driver_nl80211_get_ssid,
11492          .set_key = driver_nl80211_set_key,
11493          .scan2 = driver_nl80211_scan2,
11494          .sched_scan = wpa_driver_nl80211_sched_scan,
11495          .stop_sched_scan = wpa_driver_nl80211_stop_sched_scan,
11496          .get_scan_results2 = wpa_driver_nl80211_get_scan_results,
11497          .abort_scan = wpa_driver_nl80211_abort_scan,
11498          .deauthenticate = driver_nl80211_deauthenticate,
11499          .authenticate = driver_nl80211_authenticate,
11500          .associate = wpa_driver_nl80211_associate,
11501          .global_init = nl80211_global_init,
11502          .global_deinit = nl80211_global_deinit,
```

external/wpa_supplicant_8/src/drivers/driver_nl80211.c

```
6117  static int wpa_driver_nl80211_associate(
6118    void *priv, struct wpa_driver_associate_params *params)
6119  {
......
6153    msg = nl80211_drv_msg(drv, 0, NL80211_CMD_ASSOCIATE);
......
6181    ret = send_and_recv_msgs_owner(drv, msg,
6182                       get_connect_handle(drv->first_bss), 1,
6183                       NULL, NULL);
......
6198  }
```

```
443
444   /* Use this method to mark that it is necessary to own the connection/interface
445    * for this operation.
446    * handle may be set to NULL, to get the same behavior as send_and_recv_msgs().
447    * set_owner can be used to mark this socket for receiving control port frames.
448    */
449  static int send_and_recv_msgs_owner(struct wpa_driver_nl80211_data *drv,
450                                      struct nl_msg *msg,
451                                      struct nl_sock *handle, int set_owner,
452                                      int (*valid_handler)(struct nl_msg *,
453                                                           void *),
454                                      void *valid_data)
455   {
456           /* Control port over nl80211 needs the flags and attributes below.
457            *
458            * The Linux kernel has initial checks for them (in nl80211.c) like:
459            *     validate_pae_over_nl80211(...)
460            * or final checks like:
461            *     dev->ieee80211_ptr->conn_owner_nlportid != info->snd_portid
462            *
463            * Final operations (e.g., disassociate) don't need to set these
464            * attributes, but they have to be performed on the socket, which has
465            * the connection owner property set in the kernel.
466            */
467           if ((drv->capa.flags2 & WPA_DRIVER_FLAGS2_CONTROL_PORT_RX) &&
468               handle && set_owner &&
469               (nla_put_flag(msg, NL80211_ATTR_CONTROL_PORT_OVER_NL80211) ||
470                nla_put_flag(msg, NL80211_ATTR_SOCKET_OWNER) ||
471                nla_put_u16(msg, NL80211_ATTR_CONTROL_PORT_ETHERTYPE, ETH_P_PAE) ||
472                nla_put_flag(msg, NL80211_ATTR_CONTROL_PORT_NO_PREAUTH)))
473                   return -1;
474
475           return send_and_recv(drv->global, handle ? handle : drv->global->nl,
```

```
476                                     msg, valid_handler, valid_data);
477     }
```

驱动回复数据后调用回调函数do_process_drv_event函数来处理NL80211_CMD_CONNECT事件：

external/wpa_supplicant_8/src/drivers/driver_nl80211_event.c

```
2577  static void do_process_drv_event(struct i802_bss *bss, int cmd,
2578                 struct nlattr **tb)
2579  {
......
2607    switch (cmd) {
......
2685    case NL80211_CMD_CONNECT:
2686    case NL80211_CMD_ROAM:
2687        mlme_event_connect(drv, cmd,
2688                   tb[NL80211_ATTR_STATUS_CODE],
2689                   tb[NL80211_ATTR_MAC],
2690                   tb[NL80211_ATTR_REQ_IE],
2691                   tb[NL80211_ATTR_RESP_IE],
2692                   tb[NL80211_ATTR_TIMED_OUT],
2693                   tb[NL80211_ATTR_TIMEOUT_REASON],
2694  #ifdef CONFIG_MTK_COMMON
2695                   tb[NL80211_ATTR_PORT_AUTHORIZED], NULL, NULL,
2696  #else
2697                   NULL, NULL, NULL,
2698  #endif
2699                   tb[NL80211_ATTR_FILS_KEK],
2700                   NULL,
2701                   tb[NL80211_ATTR_FILS_ERP_NEXT_SEQ_NUM],
```

```
2702                    tb[NL80211_ATTR_PMK],
2703                    tb[NL80211_ATTR_PMKID]);
2704        break;
......
}
```

external/wpa_supplicant_8/src/drivers/driver_nl80211_event.c

```
213  static void mlme_event_assoc(struct wpa_driver_nl80211_data *drv,
214                  const u8 *frame, size_t len, struct nlattr *wmm,
215                  struct nlattr *req_ie)
216  {
......
290     wpa_supplicant_event(drv->ctx, EVENT_ASSOC, &event);
291  }
```

经过判断如果成功认证关联则调用wpa_supplicant_event_assoc进行后续处理

external/wpa_supplicant_8/wpa_supplicant/events.c

```
4594  void wpa_supplicant_event(void *ctx, enum wpa_event_type event,
4595              union wpa_event_data *data)
4596  {
......
4642    case EVENT_ASSOC:
......
4655        wpa_supplicant_event_assoc(wpa_s, data);
......
5413  }
```

external/wpa_supplicant_8/wpa_supplicant/events.c

```
2987  static void wpa_supplicant_event_assoc(struct wpa_supplicant *wpa_s,
2988                         union wpa_event_data *data)
2989  {


3273  }
```

6.3. wpa-enterprise EAPOL-EAP-PEAP_90-GTC-TLV认证流程

struct wpa_supplicant a

struct wpa_ssid b
struct eap_peer_config c

struct eap_sm d

struct eapol_sm e

a->current_ssid = b
b.eap = c

d->eapol_ctx = e
e->config = c

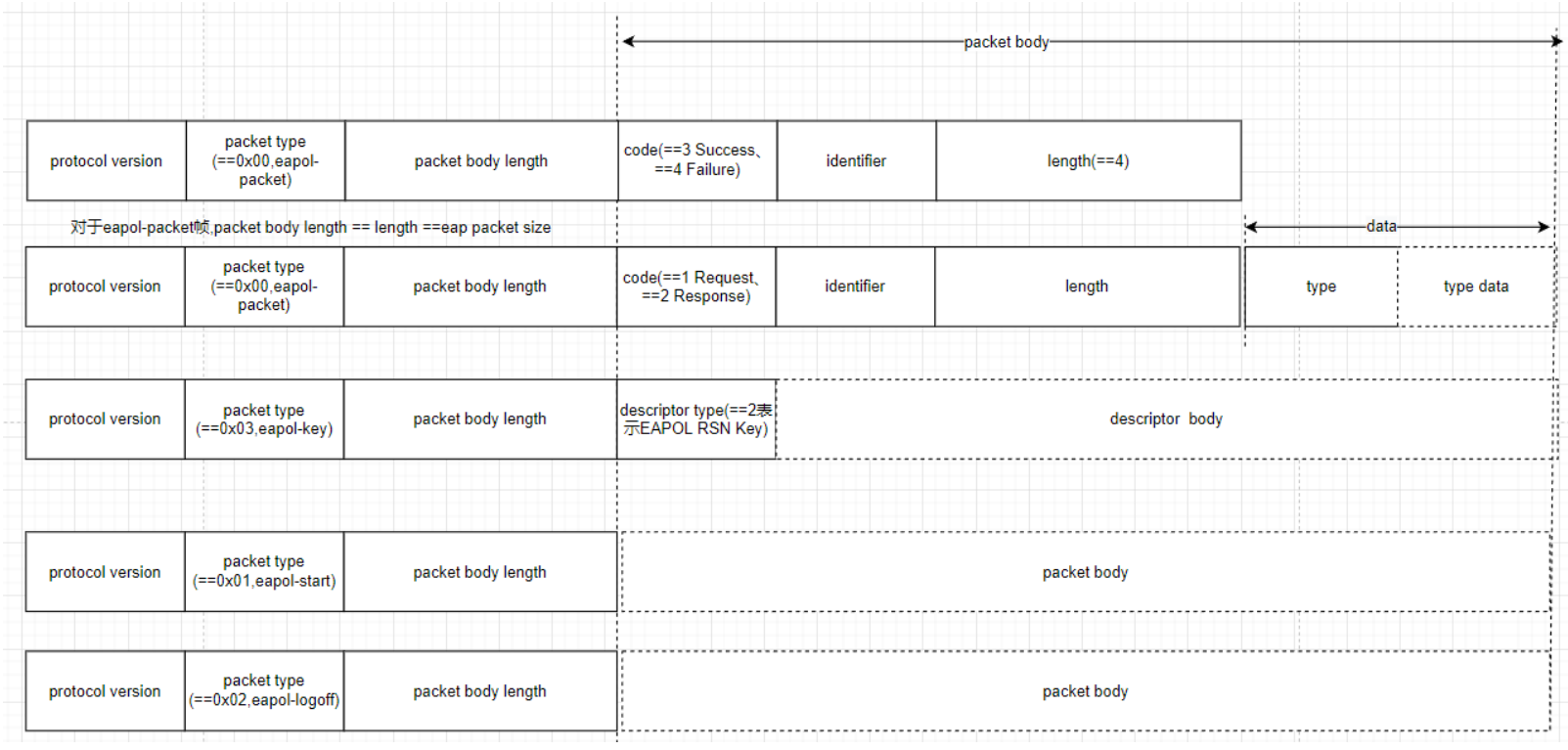
d->last_config = c

```
######TX SUPP_PAE -> CONNECTING   ,  packet type ==0x01, EAPOL-Start



######RX,packet type ==0x00 eapol-packet,   code=Request/Identity + null
######TX,packet type ==0x00 eapol-packet,   code=Response/Identity + qrKey


######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data =0x20( SSL: Received
packet - Flags 0x20)
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data= 明文  “客户”->“服务器”:
handshake/client hello


######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data = 明文 “服务器”->“客户”:
你好，我是服务器，这里是数字证书的内容(包含公钥)，请查收。
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data=0x00 “客户”->“服务器”:
OK，接下来向我证明你就是服务器


######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data = 密文   “服务器”->“客
户”：{一个随机字符串,即handshake/server hello}[私钥|RSA]
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data= 密文   “客户”->“服务
器”：{我们后面的通信过程，用对称加密来进行，这里是对称加密算法类型和及密钥，请查收}[公钥|RSA]


######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data = 密文   “服务器”->“客
户”：{OK，收到对称加密算法的类型和对应的密钥}[密钥|对称加密算法]
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data=0x00   “客户”->“服务
器”：OK，接下来开始第二阶段认证
```

```
######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data = 密文   “服务器”->“客户”：{code=Request/type=1(Identity)  + null }[密钥|对称加密算法]
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data= 密文   “客户”->“服务器”：{code=Response/type=1(Identity)  + identity}[密钥|对称加密算法]


######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data = 密文   “服务器”->“客户”：{code=Request/type=6(身份验证方法为GTC，Generic Token Card，通用令牌卡) + ? }[密钥|对称加密算法]
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data= 密文   “客户”->“服务器”：{code=Response/type=6(身份验证方法为GTC，Generic Token Card，通用令牌卡)  + identity + password}[密钥|对称加密算法]


######RX,packet type ==0x00 eapol-packet,   code=Request/type=0x5a(PEAP_90), type data = 密文   “服务器”->“客户”：{code=Request/type=33(EAP_TYPE_TLV) + T+L+pMessage }[密钥|对称加密算法]
######TX,packet type ==0x00 eapol-packet,   code=Response/type=0x5a(PEAP_90), type data=密文   “客户”->“服务器”：{code=Response/type=33(EAP_TYPE_TLV)  + 0x000000001(int,EAP_TLV_RESULT_SUCCESS)}[密钥|对称加密算法]


######RX,packet type ==0x00 eapol-packet,   code=Success


######RX,  packet type=0x03  EAPOL-Key, descriptor_type=2(表示EAPOL RSN Key), descriptor_body=ANonece, Individual
######TX,  packet type=0x03  EAPOL-Key, descriptor_type=2(表示EAPOL RSN Key), descriptor_body=SNonece, Individual，  MIC


######RX,  packet type=0x03  EAPOL-Key, descriptor_type=2(表示EAPOL RSN Key),descriptor_body=Install GTK,Individual,MIC,Encrypted(GTK,IGTK)
######TX,  packet type=0x03  EAPOL-Key, descriptor_type=2(表示EAPOL RSN Key),descriptor_body=Individual，MIC
```

http://www.youdzone.com/signature.html

https://blog.csdn.net/ly131420/article/details/38400583



6.3. wpa 4-way-handsheck && group handsheck

10 wifi驱动(rk平台)

platform_device/platform_drive

device_add

```
platform_device_register//1 (2) 3
mmc_host_add//1 2
device_create//1 2
mmc_add_card  //1 (2) 3


int device_add(struct device *dev) {

1、dev->parent   //parent必须不为空，用于在/sys/devices/${dev->parent}下面创建子设备

2、device_add_class_symlinks(dev);//dev->class，class可以为空，如果不为空则在/sys/class/${dev->class}下创建符号连接并指向在/sys/devices/${dev->parent}下创建的子设备。

3、bus_add_device(dev);//dev->bus，bus可以为空，如果不为空则在/sys/bus/${dev->bus}/devices/下创建符号连接并指向在/sys/devices/${dev->parent}下创建的子设备，并匹配dev->bus上的driver如果匹配则调用driver的probe。
```

```
}
```

10.1platform_device

根据/rk3569_r/kernel/drivers/mmc/host/Kconfig和/rk3569_r/kernel/drivers/mmc/host/Makefile

分别定义了如下config，并且对应在/rk3569_r/kernel/drivers/mmc/host/下要编译进内核的驱动模块文件有：

CONFIG_MMC_SDHCI=y
obj-y + = sdhci.o
CONFIG_MMC_SDHCI_PLTFM=y
obj-y + = sdhci-pltfm.o
CONFIG_MMC_SDHCI_OF_ARASAN=y
obj-y + = sdhci-of-arasan.o

CONFIG_MMC_SDHCI_OF_DWCMSHC=y
obj-y + = sdhci-of-dwcmshc.o

CONFIG_MMC_DW=y
obj-y + = dw_mmc.o

CONFIG_MMC_DW_PLTFM=y
obj-y + = dw_mmc-pltfm.o

CONFIG_MMC_DW_ROCKCHIP=y
obj-y + = dw_mmc-rockchip.o
obj-y + = rk_sdmmc_ops.o

CONFIG_MMC_CQHCI=y
obj-y + = cqhci.o

kernel/arch/arm64/boot/dts/rockchip/rk3568.dtsi

```
2122    sdmmc2: dwmmc@fe000000 {
2123        compatible = "rockchip,rk3568-dw-mshc",
2124                 "rockchip,rk3288-dw-mshc";
2125        reg = <0x0 0xfe000000 0x0 0x4000>;
2126        interrupts = <GIC_SPI 100 IRQ_TYPE_LEVEL_HIGH>;
2127        max-frequency = <150000000>;
2128        clocks = <&cru HCLK_SDMMC2>, <&cru CLK_SDMMC2>,
2129             <&cru SCLK_SDMMC2_DRV>, <&cru SCLK_SDMMC2_SAMPLE>;
2130        clock-names = "biu", "ciu", "ciu-drive", "ciu-sample";
2131        fifo-depth = <0x100>;
2132        resets = <&cru SRST_SDMMC2>;
2133        reset-names = "reset";
2134        status = "disabled";
2135    };
......
2420    sdmmc0: dwmmc@fe2b0000 {
2421        compatible = "rockchip,rk3568-dw-mshc",
2422                 "rockchip,rk3288-dw-mshc";
2423        reg = <0x0 0xfe2b0000 0x0 0x4000>;
2424        interrupts = <GIC_SPI 98 IRQ_TYPE_LEVEL_HIGH>;
2425        max-frequency = <150000000>;
2426        clocks = <&cru HCLK_SDMMC0>, <&cru CLK_SDMMC0>,
2427             <&cru SCLK_SDMMC0_DRV>, <&cru SCLK_SDMMC0_SAMPLE>;
2428        clock-names = "biu", "ciu", "ciu-drive", "ciu-sample";
2429        fifo-depth = <0x100>;
2430        resets = <&cru SRST_SDMMC0>;
2431        reset-names = "reset";
2432        status = "disabled";
2433    };
```

```
2434
2435    sdmmc1: dwmmc@fe2c0000 {
2436        compatible = "rockchip,rk3568-dw-mshc",
2437                 "rockchip,rk3288-dw-mshc";
2438        reg = <0x0 0xfe2c0000 0x0 0x4000>;
2439        interrupts = <GIC_SPI 99 IRQ_TYPE_LEVEL_HIGH>;
2440        max-frequency = <150000000>;
2441        clocks = <&cru HCLK_SDMMC1>, <&cru CLK_SDMMC1>,
2442             <&cru SCLK_SDMMC1_DRV>, <&cru SCLK_SDMMC1_SAMPLE>;
2443        clock-names = "biu", "ciu", "ciu-drive", "ciu-sample";
2444        fifo-depth = <0x100>;
2445        resets = <&cru SRST_SDMMC1>;
2446        reset-names = "reset";
2447        status = "disabled";
2448    };
```

设备树被解析后生成三个平台设备，对应

/sys/devices/platform/dwmmc@fe000000

/sys/devices/platform/dwmmc@fe2b0000

/sys/devices/platform/dwmmc@fe2c0000

三个平台设备compatible有一项为"rockchip,rk3288-dw-mshc"，与如下驱动匹配(调三次probe，每次调用时传入的struct platform_device *pdev指向不同的平台设备)：

kernel/drivers/mmc/host/dw_mmc-rockchip.c

```
392  static const struct of_device_id dw_mci_rockchip_match[] = {
393      { .compatible = "rockchip,rk2928-dw-mshc",
394          .data = &rk2928_drv_data },
395      { .compatible = "rockchip,rk3288-dw-mshc",
396          .data = &rk3288_drv_data },
```

```
397     {},
398  };
......
464  static struct platform_driver dw_mci_rockchip_pltfm_driver = {
465     .probe      = dw_mci_rockchip_probe,
466     .remove     = dw_mci_rockchip_remove,
467     .driver     = {
468         .name       = "dwmmc_rockchip",
469         .of_match_table = dw_mci_rockchip_match,
470         .pm     = &dw_mci_rockchip_dev_pm_ops,
471     },
472  };
473
474  module_platform_driver(dw_mci_rockchip_pltfm_driver);
......
```

在dw_mmc-rockchip驱动模块装载时调用到dw_mci_rockchip_probe

kernel/drivers/mmc/host/dw_mmc-rockchip.c

xref: /rk3569_r/kernel/drivers/mmc/host/dw_mmc-rockchip.c


```
399  MODULE_DEVICE_TABLE(of, dw_mci_rockchip_match);
400
401 static int dw_mci_rockchip_probe(struct platform_device *pdev)
402  {
403          const struct dw_mci_drv_data *drv_data;
404          const struct of_device_id *match;
405          int ret;
406          bool use_rpm = true;
407
408          if (!pdev->dev.of_node)
409                  return -ENODEV;
410
411          if (!device_property_read_bool(&pdev->dev, "non-removable") &&
412              !device_property_read_bool(&pdev->dev, "cd-gpios"))
413                  use_rpm = false;
414
415          match = of_match_node(dw_mci_rockchip_match, pdev->dev.of_node);
416          drv_data = match->data;
417
418          /*
419           * increase rpm usage count in order to make
420           * pm_runtime_force_resume calls rpm resume callback
421           */
422          pm_runtime_get_noresume(&pdev->dev);
423
424          if (use_rpm) {
425                  pm_runtime_set_active(&pdev->dev);
426                  pm_runtime_enable(&pdev->dev);
427                  pm_runtime_set_autosuspend_delay(&pdev->dev, 50);
428                  pm_runtime_use_autosuspend(&pdev->dev);
429          }
430
431          ret = dw_mci_pltfm_register(pdev, drv_data);
432          if (ret) {
433                  if (use_rpm) {
434                          pm_runtime_disable(&pdev->dev);
```

```
435                     pm_runtime_set_suspended(&pdev->dev);
436             }
437             pm_runtime_put_noidle(&pdev->dev);
438             return ret;
439     }
440
441     if (use_rpm)
442             pm_runtime_put_autosuspend(&pdev->dev);
443
444     return 0;
445 }
```

在dw_mci_rockchip_probe函数中调用到dw_mci_pltfm_register。dw_mci_pltfm_register是dw_mmc-pltfm驱动模块导出的函数符号，dw_mci_rockchip_probe和dw_mci_pltfm_register负责解析设备树中的部分信息并放入struct dw_mci结构体host。

xref: /rk3569_r/kernel/drivers/mmc/host/dw_mmc-pltfm.c


```
28
29  int dw_mci_pltfm_register(struct platform_device *pdev,
30                            const struct dw_mci_drv_data *drv_data)
31  {
32          struct dw_mci *host;
33          struct resource         *regs;
34
35          host = devm_kzalloc(&pdev->dev, sizeof(struct dw_mci), GFP_KERNEL);
36          if (!host)
37                  return -ENOMEM;
38
39          host->irq = platform_get_irq(pdev, 0);
40          if (host->irq < 0)
41                  return host->irq;
42
43          host->drv_data = drv_data;
44          host->dev = &pdev->dev;
45          host->irq_flags = 0;
46          host->pdata = pdev->dev.platform_data;
47
48          regs = platform_get_resource(pdev, IORESOURCE_MEM, 0);
49          host->regs = devm_ioremap_resource(&pdev->dev, regs);
50          if (IS_ERR(host->regs))
51                  return PTR_ERR(host->regs);
52
53          /* Get registers' physical base address */
54          host->phy_regs = regs->start;
55
56          platform_set_drvdata(pdev, host);
57          return dw_mci_probe(host);
58  }
59  EXPORT_SYMBOL_GPL(dw_mci_pltfm_register);
60
```

dw_mci_pltfm_register最后调用到了dw_mci_probe，dw_mci_probe是dw_mmc驱动模块导出的符号：

/rk3569_r/kernel/drivers/mmc/host/dw_mmc.c

```
3305  int dw_mci_probe(struct dw_mci *host)
3306  {
......
3502    /* We need at least one slot to succeed */
3503    ret = dw_mci_init_slot(host);
......
3528  }
3529  EXPORT_SYMBOL(dw_mci_probe);
```

dw_mci_probe调用到了dw_mci_init_slot。

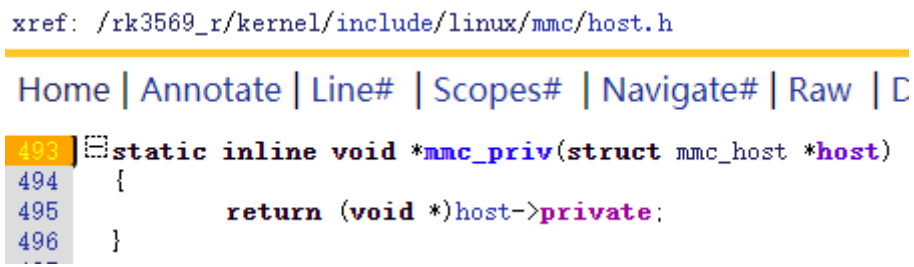
10.2 mmc_host

平台驱动把host子设备注册到平台设备之下，一个sdio/sd/mmc控制器对应一个host子设备。

/rk3569_r/kernel/drivers/mmc/host/dw_mmc.c

```
2902  static int dw_mci_init_slot(struct dw_mci *host)
2903  {
2904    struct mmc_host *mmc;
2905    struct dw_mci_slot *slot;
2906    int ret;
2907
```

```
2908    mmc = mmc_alloc_host(sizeof(struct dw_mci_slot), host->dev);
2909    if (!mmc)
2910        return -ENOMEM;
2911
2912    slot = mmc_priv(mmc);
2913    slot->id = 0;
2914    slot->sdio_id = host->sdio_id0 + slot->id;
2915    slot->mmc = mmc;
2916    slot->host = host;
2917    host->slot = slot;
2918
2919    mmc->ops = &dw_mci_ops;
......
2968    ret = mmc_add_host(mmc);
......
2981  }
```



dw_mci_init_slot调用mmc_alloc_host分配一个mmc_host结构体mmc，并把mmc->private赋给struct dw_mci_slot *slot后进行初始化，mmc->ops指向了一个mmc_host_ops结构体。mmc_host_ops结构体中存放着大量mmc子系统core与host直接的操作接口函数，这些函数具体实现是操作SD/SDIO/MMC控制器的寄存器。这里的控制器使用的是

```
Rockchip specific extensions for Synopsys DW Memory Card Interface
```

即该控制器由Synopsys设计，Rockchip对其进行了一些定制化。

xref: /rk3569_r/kernel/drivers/mmc/host/dw_mmc.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
1840
1841  static const struct mmc_host_ops dw_mci_ops = {
1842          .request                = dw_mci_request,
1843          .pre_req                = dw_mci_pre_req,
1844          .post_req               = dw_mci_post_req,
1845          .set_ios                = dw_mci_set_ios,
1846          .set_sdio_status        = dw_mci_set_sdio_status,
1847          .get_ro                 = dw_mci_get_ro,
1848          .get_cd                 = dw_mci_get_cd,
1849          .hw_reset               = dw_mci_hw_reset,
1850          .enable_sdio_irq        = dw_mci_enable_sdio_irq,
1851          .ack_sdio_irq           = dw_mci_ack_sdio_irq,
1852          .execute_tuning         = dw_mci_execute_tuning,
1853          .card_busy              = dw_mci_card_busy,
1854          .start_signal_voltage_switch = dw_mci_switch_voltage,
1855          .init_card              = dw_mci_init_card,
1856          .prepare_hs400_tuning   = dw_mci_prepare_hs400_tuning,
1857  };
1858
```

mmc_alloc_host在kernel/drivers/mmc/core/host.c定义：

```
382  /**
383   *      mmc_alloc_host - initialise the per-host structure.
384   *      @extra: sizeof private data structure
385   *      @dev: pointer to host device model structure
386   *
387   *      Initialise the per-host structure.
388   */
389  struct mmc_host *mmc_alloc_host(int extra, struct device *dev)
390  {
391          int err;
392          struct mmc_host *host;
393          int alias_id, min_idx, max_idx;
394
395          host = kzalloc(sizeof(struct mmc_host) + extra, GFP_KERNEL);
396          if (!host)
397                  return NULL;
398
399          /* scanning will be enabled when we're ready */
400          host->rescan_disable = 1;
401
402          alias_id = of_alias_get_id(dev->of_node, "mmc");
403          if (alias_id >= 0) {
404                  min_idx = alias_id;
405                  max_idx = alias_id + 1;
406          } else {
407                  min_idx = mmc_first_nonreserved_index();
408                  max_idx = 0;
409          }
410
411          err = ida_simple_get(&mmc_host_ida, min_idx, max_idx, GFP_KERNEL);
412          if (err < 0) {
413                  kfree(host);
414                  return NULL;
415          }
416
417          host->index = err;
418
419          dev_set_name(&host->class_dev, "mmc%d", host->index);
420
421          host->parent = dev;
```

```
422             host->class_dev.parent = dev;
423             host->class_dev.class = &mmc_host_class;
424             device_initialize(&host->class_dev);
425             device_enable_async_suspend(&host->class_dev);
426
427             if (mmc_gpio_alloc(host)) {
428                     put_device(&host->class_dev);
429                     return NULL;
430             }
431
432             spin_lock_init(&host->lock);
433             init_waitqueue_head(&host->wq);
434             INIT_DELAYED_WORK(&host->detect, mmc_rescan);
435             INIT_DELAYED_WORK(&host->sdio_irq_work, sdio_irq_work);
436             timer_setup(&host->retune_timer, mmc_retune_timer, 0);
437
438             /*
439              * By default, hosts do not support SGIO or large requests.
440              * They have to set these according to their abilities.
441              */
442             host->max_segs = 1;
443             host->max_seg_size = PAGE_SIZE;
444
445             host->max_req_size = PAGE_SIZE;
446             host->max_blk_size = 512;
447             host->max_blk_count = PAGE_SIZE / 512;
448
449             host->fixed_drv_type = -EINVAL;
450             host->ios.power_delay_ms = 10;
451
452             return host;
453     }
454
455     EXPORT_SYMBOL(mmc_alloc_host);
```

mmc_alloc_host函数关键部分在

```
422     host->class_dev.parent = dev;
423     host->class_dev.class = &mmc_host_class;
......
434     INIT_DELAYED_WORK(&host->detect, mmc_rescan);
435     INIT_DELAYED_WORK(&host->sdio_irq_work, sdio_irq_work);
```

dev和class_dev是struct device结构体

dev是前面dw_mmc-rockchip驱动匹配的那个平台设备的dev(即pdev->dev)，该设备由与驱动compatible匹配的设备树节点解析生成，在/sys/devices下对应节点/sys/devices/platform/dwmmc@xxxxxxxx，在/sys/class下没有对应节点，在/sys/bus下对应节点/sys/bus/platform/devices/dwmmc@xxxxxxxx。

mmc_host_class定义在kernel/drivers/mmc/core/host.c：

```
47  static struct class mmc_host_class = {
48          .name           = "mmc_host",
49          .dev_release    = mmc_host_classdev_release,
50  };
```

返回到dw_mci_init_slot函数中，后续调用在mmc_add_host函数，在mmc_add_host函数中调用device_add函数时就会在/sys/devices/platform/dwmmc@xxxxxxxx/下创建${class_dev.name}节点，对应dev的子设备。此外还会在/sys/class/下创建${mmc_host_class.name}节点。且没有在/sys/bus下创建节点，因为class_dev.bus没有指定。

```
460  /**
461   *      mmc_add_host - initialise host hardware
462   *      @host: mmc host
463   *
464   *      Register the host with the driver model. The host must be
465   *      prepared to start servicing requests before this function
466   *      completes.
467   */
468 int mmc_add_host(struct mmc_host *host)
469 {
470         int err;
471
472         WARN_ON((host->caps & MMC_CAP_SDIO_IRQ) &&
473                 !host->ops->enable_sdio_irq);
474
475         err = device_add(&host->class_dev);
476         if (err)
477                 return err;
478
479         led_trigger_register_simple(dev_name(&host->class_dev), &host->led);
480
481 #ifdef CONFIG_DEBUG_FS
482         mmc_add_host_debugfs(host);
483 #endif
484
485         mmc_start_host(host);
486         if (!(host->pm_flags & MMC_PM_IGNORE_PM_NOTIFY))
487                 mmc_register_pm_notifier(host);
488
489         if (host->restrict_caps & RESTRICT_CARD_TYPE_SDIO)
490                 primary_sdio_host = host;
491
492         return 0;
493 }
494
495 EXPORT_SYMBOL(mmc_add_host);
```

mmc_add_host函数调完device_add后再调mmc_start_host函数：

xref: /rk3569_r/kernel/drivers/mmc/core/core.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Dowr

```
2716
2717  void mmc_start_host(struct mmc_host *host)
2718  {
2719          host->f_init = max(freqs[0], host->f_min);
2720          host->rescan_disable = 0;
2721          host->ios.power_mode = MMC_POWER_UNDEFINED;
2722
2723          if (!(host->caps2 & MMC_CAP2_NO_PRESCAN_POWERUP)) {
2724                  mmc_claim_host(host);
2725                  mmc_power_up(host, host->ocr_avail);
2726                  mmc_release_host(host);
2727          }
2728
2729          mmc_gpiod_request_cd_irq(host);
2730          _mmc_detect_change(host, 0, false);
2731  }
```

mmc_start_host函数调用_mmc_detect_change函数：

xref: /rk3569_r/kernel/drivers/mmc/core/core.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
1785
1786  static void _mmc_detect_change(struct mmc_host *host, unsigned long delay,
1787                                 bool cd_irq)
1788  {
1789          /*
1790           * If the device is configured as wakeup, we prevent a new sleep for
1791           * 5 s to give provision for user space to consume the event.
1792           */
1793          if (cd_irq && !(host->caps & MMC_CAP_NEEDS_POLL) &&
1794                  device_can_wakeup(mmc_dev(host)))
1795                  pm_wakeup_event(mmc_dev(host), 5000);
1796
1797          host->detect_change = 1;
1798          mmc_schedule_delayed_work(&host->detect, delay);
1799  }
1800
```

_mmc_detect_change函数调用mmc_schedule_delayed_work函数:

xref: /rk3569_r/kernel/drivers/mmc/core/core.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
63
64  static int mmc_schedule_delayed_work(struct delayed_work *work,
65                                       unsigned long delay)
66  {
67          /*
68           * We use the system_freezable_wq, because of two reasons.
69           * First, it allows several works (not the same work item) to be
70           * executed simultaneously. Second, the queue becomes frozen when
71           * userspace becomes frozen during system PM.
72           */
73          return queue_delayed_work(system_freezable_wq, work, delay);
74  }
```

mmc_schedule_delayed_work函数调用queue_delayed_work函数将host->detect加入system_freezable_wq队列。host->detect先前在mmc_alloc_host函数中进行了初始化，指定处理函数为mmc_rescan：

kernel/drivers/mmc/core/host.c

```
434     INIT_DELAYED_WORK(&host->detect, mmc_rescan);
```

调用queue_delayed_work后，一方面一路返回到平台驱动程序的dw_mci_rockchip_probe执行完毕，另一方面负责处理system_freezable_wq队列中任务的内核kwoker线程就会取出host->detect，并将其作为参数调用里面的mmc_rescan函数。mmc_rescan如下：

/rk3569_r/kernel/drivers/mmc/core/core.c

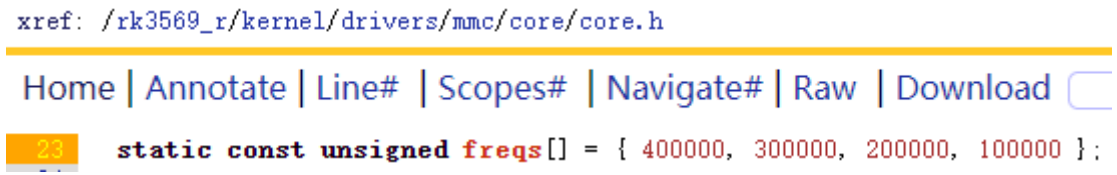
```
2645 void mmc_rescan(struct work_struct *work)
2646 {
2647         struct mmc_host *host =
2648                 container_of(work, struct mmc_host, detect.work);
2649         int i;
2650
2651         if (host->rescan_disable)
2652                 return;
2653
2654         /* If there is a non-removable card registered, only scan once */
2655         if (!mmc_card_is_removable(host) && host->rescan_entered)
2656                 return;
2657         host->rescan_entered = 1;
2658
2659         if (host->trigger_card_event && host->ops->card_event) {
2660                 mmc_claim_host(host);
2661                 host->ops->card_event(host);
2662                 mmc_release_host(host);
2663                 host->trigger_card_event = false;
2664         }
2665
2666         mmc_bus_get(host);
2667
2668         /*
2669          * if there is a _removable_ card registered, check whether it is
2670          * still present
2671          */
2672         if (host->bus_ops && !host->bus_dead && mmc_card_is_removable(host))
2673                 host->bus_ops->detect(host);
2674
2675         host->detect_change = 0;
2676
2677         /*
2678          * Let mmc_bus_put() free the bus/bus_ops if we've found that
2679          * the card is no longer present.
2680          */
2681         mmc_bus_put(host);
2682         mmc_bus_get(host);
2683
2684         /* if there still is a card present, stop here */
```

```
2685            if (host->bus_ops != NULL) {
2686                    mmc_bus_put(host);
2687                    goto out;
2688            }
2689
2690            /*
2691             * Only we can add a new handler, so it's safe to
2692             * release the lock here.
2693             */
2694            mmc_bus_put(host);
2695
2696            mmc_claim_host(host);
2697            if (mmc_card_is_removable(host) && host->ops->get_cd &&
2698                            host->ops->get_cd(host) == 0) {
2699                    mmc_power_off(host);
2700                    mmc_release_host(host);
2701                    goto out;
2702            }
2703
2704            for (i = 0; i < ARRAY_SIZE(freqs); i++) {
2705                    if (!mmc_rescan_try_freq(host, max(freqs[i], host->f_min)))
2706                            break;
2707                    if (freqs[i] <= host->f_min)
2708                            break;
2709            }
2710            mmc_release_host(host);
2711
2712     out:
2713            if (host->caps & MMC_CAP_NEEDS_POLL)
2714                    mmc_schedule_delayed_work(&host->detect, HZ);
2715    }
```

mmc_rescan涉及如下函数：

mmc_claim_host(host);//尝试获得host子设备的使用全权
mmc_release_host(host);//释放host子设备
host->ops->xxx(yyy);//调用host子设备中的函数指针来配置硬件控制器的寄存器以实现某些功能。函数指针指向厂家实现的函数。

在for循环中以四个不同的频率作为参数调用mmc_rescan_try_freq函数，依次尝试几个给定频率扫卡，直至检测到mmc card的存在。



经实测，在RK平台上由于host->f_min的限制mmc_rescan_try_freq执行了两次，传入的第二个参数(频率)分别为400000,300000

mmc_rescan_try_freq函数十分关键，首先调用mmc_power_up扫卡初始化，随后按顺序判断卡是sdio/sd/mmc那种，然后再调用对应的mmc_attach_xxx函数进行处理：

mmc_power_up　sdmmc控制器的对应管脚初始化

sdio_reset　　发送CMD52读取寄存器，修改

mmc_go_idle

mmc_send_if_cond

mmc_send_if_cond

/rk3569_r/kernel/drivers/mmc/core/core.c

```
2491  static int mmc_rescan_try_freq(struct mmc_host *host, unsigned freq)
2492  {
2493          host->f_init = freq;
2494
2495          pr_debug("%s: %s: trying to init card at %u Hz\n",
2496                  mmc_hostname(host), __func__, host->f_init);
2497
2498          mmc_power_up(host, host->ocr_avail);
2499
2500          /*
2501           * Some eMMCs (with VCCQ always on) may not be reset after power up, so
2502           * do a hardware reset if possible.
2503           */
2504  #ifndef CONFIG_ROCKCHIP_THUNDER_BOOT
2505          mmc_hw_reset_for_init(host);
2506  #endif
2507
2508  #ifdef CONFIG_SDIO_KEEPALIVE
2509          if (host->support_chip_alive) {
2510                  host->chip_alive = 1;
2511                  if (!mmc_attach_sdio(host)) {
2512                          return 0;
2513                  } else {
2514                          pr_err("%s: chip_alive attach sdio failed.\n", mmc_hostname(host));
2515                          host->chip_alive = 0;
2516                  }
2517          } else {
2518                  host->chip_alive = 0;
2519          }
2520  #endif
2521
2522          /*
2523           * sdio_reset sends CMD52 to reset card.  Since we do not know
2524           * if the card is being re-initialized, just send it.  CMD52
2525           * should be ignored by SD/eMMC cards.
2526           * Skip it if we already know that we do not support SDIO commands
2527           */
2528  #ifdef MMC_STANDARD_PROBE
2529          if (!(host->caps2 & MMC_CAP2_NO_SDIO))
2530                  sdio_reset(host);
```

```
2531
2532            mmc_go_idle(host);
2533
2534            if (!(host->caps2 & MMC_CAP2_NO_SD))
2535                    mmc_send_if_cond(host, host->ocr_avail);
2536
2537            /* Order's important: probe SDIO, then SD, then MMC */
2538            if (!(host->caps2 & MMC_CAP2_NO_SDIO))
2539                    if (!mmc_attach_sdio(host))
2540                            return 0;
2541
2542            if (!(host->caps2 & MMC_CAP2_NO_SD))
2543                    if (!mmc_attach_sd(host))
2544                            return 0;
2545
2546            if (!(host->caps2 & MMC_CAP2_NO_MMC))
2547                    if (!mmc_attach_mmc(host))
2548                            return 0;
2549    #else
2550    #ifdef CONFIG_SDIO_KEEPALIVE
2551            if ((!(host->chip_alive)) && (host->restrict_caps & RESTRICT_CARD_TYPE_SDIO))
2552                    sdio_reset(host);
2553    #else
2554            if (host->restrict_caps & RESTRICT_CARD_TYPE_SDIO)
2555                    sdio_reset(host);
2556    #endif
2557
2558            mmc_go_idle(host);
2559
2560            if (host->restrict_caps &
2561                (RESTRICT_CARD_TYPE_SDIO | RESTRICT_CARD_TYPE_SD))
2562                    mmc_send_if_cond(host, host->ocr_avail);
2563            /* Order's important: probe SDIO, then SD, then MMC */
2564            if ((host->restrict_caps & RESTRICT_CARD_TYPE_SDIO) &&
2565                !mmc_attach_sdio(host))
2566                    return 0;
2567            if ((host->restrict_caps & RESTRICT_CARD_TYPE_SD) &&
2568                !mmc_attach_sd(host))
2569                    return 0;
2570            if ((host->restrict_caps & RESTRICT_CARD_TYPE_EMMC) &&
```

```
2571                !mmc_attach_mmc(host))
2572                    return 0;
2573    #endif
2574            mmc_power_off(host);
2575            return -EIO;
2576    }
```

10.3 mmc_card&&sdio_func

mmc_attach_sdio依次调用：

```
mmc_attach_bus
mmc_sdio_init_card
sdio_init_func
mmc_add_card
sdio_add_func
```

/rk3569_r/kernel/drivers/mmc/core/sdio.c

```
1123  int mmc_attach_sdio(struct mmc_host *host)
1124  {
1125          int err, i, funcs;
1126          u32 ocr, rocr;
1127          struct mmc_card *card;
1128
1129          WARN_ON(!host->claimed);
1130
1131  #ifdef CONFIG_SDIO_KEEPALIVE
1132          if (!(host->chip_alive)) {
1133                  err = mmc_send_io_op_cond(host, 0, &ocr);
1134                  if (err) {
1135                          pr_err("%s mmc_send_io_op_cond err: %d\n", mmc_hostname(host), err);
1136                          return err;
1137                  }
1138          } else {
1139                  ocr = 0x20ffff00;
1140          }
1141  #else
1142          err = mmc_send_io_op_cond(host, 0, &ocr);
1143          if (err)
1144                  return err;
1145  #endif
1146
1147          mmc_attach_bus(host, &mmc_sdio_ops);
1148          if (host->ocr_avail_sdio)
1149                  host->ocr_avail = host->ocr_avail_sdio;
1150
1151
1152          rocr = mmc_select_voltage(host, ocr);
1153
1154          /*
1155           * Can we support the voltage(s) of the card(s)?
1156           */
1157          if (!rocr) {
1158                  err = -EINVAL;
1159                  goto err;
1160          }
1161
1162          /*
```

```
1163            * Detect and init the card.
1164            */
1165           err = mmc_sdio_init_card(host, rocr, NULL, 0);
1166           if (err)
1167                   goto err;
1168
1169           card = host->card;
1170
1171           /*
1172            * Enable runtime PM only if supported by host+card+board
1173            */
1174           if (host->caps & MMC_CAP_POWER_OFF_CARD) {
1175                   /*
1176                    * Do not allow runtime suspend until after SDIO function
1177                    * devices are added.
1178                    */
1179                   pm_runtime_get_noresume(&card->dev);
1180
1181                   /*
1182                    * Let runtime PM core know our card is active
1183                    */
1184                   err = pm_runtime_set_active(&card->dev);
1185                   if (err)
1186                           goto remove;
1187
1188                   /*
1189                    * Enable runtime PM for this card
1190                    */
1191                   pm_runtime_enable(&card->dev);
1192           }
1193
1194           /*
1195            * The number of functions on the card is encoded inside
1196            * the ocr.
1197            */
1198           funcs = (ocr & 0x70000000) >> 28;
1199           card->sdio_funcs = 0;
1200
1201           /*
1202            * Initialize (but don't add) all present functions.
```

```
1203                 */
1204                for (i = 0; i < funcs; i++, card->sdio_funcs++) {
1205                        err = sdio_init_func(host->card, i + 1);
1206                        if (err)
1207                                goto remove;
1208
1209                        /*
1210                         * Enable Runtime PM for this func (if supported)
1211                         */
1212                        if (host->caps & MMC_CAP_POWER_OFF_CARD)
1213                                pm_runtime_enable(&card->sdio_func[i]->dev);
1214                }
1215
1216        #ifdef CONFIG_SDIO_KEEPALIVE
1217                if (host->card->sdio_func[1])
1218                        host->card->sdio_func[1]->card_alive = host->chip_alive;
1219        #endif
1220
1221                /*
1222                 * First add the card to the driver model...
1223                 */
1224                mmc_release_host(host);
1225                err = mmc_add_card(host->card);
1226                if (err)
1227                        goto remove_added;
1228
1229                /*
1230                 * ...then the SDIO functions.
1231                 */
1232                for (i = 0;i < funcs;i++) {
1233                        err = sdio_add_func(host->card->sdio_func[i]);
1234                        if (err)
1235                                goto remove_added;
1236                }
1237
1238                if (host->caps & MMC_CAP_POWER_OFF_CARD)
1239                        pm_runtime_put(&card->dev);
1240
1241                mmc_claim_host(host);
1242                return 0;
1243
```

```
1244
1245    remove:
1246            mmc_release_host(host);
1247    remove_added:
1248            /*
1249             * The devices are being deleted so it is not necessary to disable
1250             * runtime PM. Similarly we also don't pm_runtime_put() the SDIO card
1251             * because it needs to be active to remove any function devices that
1252             * were probed, and after that it gets deleted.
1253             */
1254            mmc_sdio_remove(host);
1255            mmc_claim_host(host);
1256    err:
1257            mmc_detach_bus(host);
1258
1259            pr_err("%s: error %d whilst initialising SDIO card\n",
1260                    mmc_hostname(host), err);
1261
1262            return err;
1263    }
```

mmc_attach_bus(host, &mmc_sdio_ops);

负责注册回调函数集合到host->bus_ops

xref: /rk3569_r/kernel/drivers/mmc/core/sdio.c


```
1105
1106  static const struct mmc_bus_ops mmc_sdio_ops = {
1107          .remove = mmc_sdio_remove,
1108          .detect = mmc_sdio_detect,
1109          .pre_suspend = mmc_sdio_pre_suspend,
1110          .suspend = mmc_sdio_suspend,
1111          .resume = mmc_sdio_resume,
1112          .runtime_suspend = mmc_sdio_runtime_suspend,
1113          .runtime_resume = mmc_sdio_runtime_resume,
1114          .alive = mmc_sdio_alive,
1115          .hw_reset = mmc_sdio_hw_reset,
1116          .sw_reset = mmc_sdio_sw_reset,
1117  };
```

xref: /rk3569_r/kernel/drivers/mmc/core/core.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
1748    */
1749   void mmc_attach_bus(struct mmc_host *host, const struct mmc_bus_ops *ops)
1750    {
1751            unsigned long flags;
1752
1753            WARN_ON(!host->claimed);
1754
1755            spin_lock_irqsave(&host->lock, flags);
1756
1757            WARN_ON(host->bus_ops);
1758            WARN_ON(host->bus_refs);
1759
1760            host->bus_ops = ops;
1761            host->bus_refs = 1;
1762            host->bus_dead = 0;
1763
1764            spin_unlock_irqrestore(&host->lock, flags);
1765    }
```

mmc_sdio_init_card(host, rocr, NULL, 0);

创建一个card设备，指定父设备为host，指定card设备的bus是mmc_bus_type总线。

/rk3569_r/kernel/drivers/mmc/core/sdio.c

```
566  static int mmc_sdio_init_card(struct mmc_host *host, u32 ocr,
567                  struct mmc_card *oldcard, int powered_resume)
568  {
......
622      card = mmc_alloc_card(host, NULL);
......
838  }
```

xref: /rk3569_r/kernel/drivers/mmc/core/bus.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
263   */
264  struct mmc_card *mmc_alloc_card(struct mmc_host *host, struct device_type *type)
265  {
266          struct mmc_card *card;
267
268          card = kzalloc(sizeof(struct mmc_card), GFP_KERNEL);
269          if (!card)
270                  return ERR_PTR(-ENOMEM);
271
272          card->host = host;
273
274          device_initialize(&card->dev);
275
276          card->dev.parent = mmc_classdev(host);
277          card->dev.bus = &mmc_bus_type;
278          card->dev.release = mmc_release_card;
279          card->dev.type = type;
280
281          return card;
282  }
```

sdio_init_func(host->card, i + 1);

sdio_init_func函数很关键，它负责分配一个sdio_func结构体用于描述sdio设备，并对其进行初始化，主要函数如下：

```
sdio_alloc_func
sdio_read_fbr
sdio_read_func_cis
```

xref: /rk3569_r/kernel/drivers/mmc/core/sdio.c


```
64
65  static int sdio_init_func(struct mmc_card *card, unsigned int fn)
66  {
67          int ret;
68          struct sdio_func *func;
69
70          if (WARN_ON(fn > SDIO_MAX_FUNCS))
71                  return -EINVAL;
72
73          func = sdio_alloc_func(card);
74          if (IS_ERR(func))
75                  return PTR_ERR(func);
76
77          func->num = fn;
78
79          if (!(card->quirks & MMC_QUIRK_NONSTD_SDIO)) {
80                  ret = sdio_read_fbr(func);
81                  if (ret)
82                          goto fail;
83
84                  ret = sdio_read_func_cis(func);
85                  if (ret)
86                          goto fail;
87          } else {
88                  func->vendor = func->card->cis.vendor;
89                  func->device = func->card->cis.device;
90                  func->max_blksize = func->card->cis.blksize;
91          }
92
93          card->sdio_func[fn - 1] = func;
94
95          return 0;
96
97  fail:
98          /*
99           * It is okay to remove the function here even though we hold
```

```
100             * the host lock as we haven't registered the device yet.
101             */
102            sdio_remove_func(func);
103            return ret;
104    }
```

sdio_alloc_func负责分配一个sdio_func结构体并对func->dev.parent和func->dev.bus进行初始化

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
276  */
277 struct sdio_func *sdio_alloc_func(struct mmc_card *card)
278 {
279         struct sdio_func *func;
280
281         func = kzalloc(sizeof(struct sdio_func), GFP_KERNEL);
282         if (!func)
283                 return ERR_PTR(-ENOMEM);
284
285         /*
286          * allocate buffer separately to make sure it's properly aligned for
287          * DMA usage (incl. 64 bit DMA)
288          */
289         func->tmpbuf = kmalloc(4, GFP_KERNEL);
290         if (!func->tmpbuf) {
291                 kfree(func);
292                 return ERR_PTR(-ENOMEM);
293         }
294
295         func->card = card;
296
297         device_initialize(&func->dev);
298
299         func->dev.parent = &card->dev;
300         func->dev.bus = &sdio_bus_type;
301         func->dev.release = sdio_release_func;
302
303         return func;
304 }
```

sdio_read_fbr负责读卡并初始化func->class

xref: /rk3569_r/kernel/drivers/mmc/core/sdio.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
34
35  static int sdio_read_fbr(struct sdio_func *func)
36  {
37          int ret;
38          unsigned char data;
39
40          if (mmc_card_nonstd_func_interface(func->card)) {
41                  func->class = SDIO_CLASS_NONE;
42                  return 0;
43          }
44
45          ret = mmc_io_rw_direct(func->card, 0, 0,
46                  SDIO_FBR_BASE(func->num) + SDIO_FBR_STD_IF, 0, &data);
47          if (ret)
48                  goto out;
49
50          data &= 0x0f;
51
52          if (data == 0x0f) {
53                  ret = mmc_io_rw_direct(func->card, 0, 0,
54                          SDIO_FBR_BASE(func->num) + SDIO_FBR_STD_IF_EXT, 0, &data);
55                  if (ret)
56                          goto out;
57          }
58
59          func->class = data;
60
61  out:
62          return ret;
63  }
```

sdio_read_func_cis负责读卡并初始化func->vendor和func->device

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_cis.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Downl

```
381
382 int sdio_read_func_cis(struct sdio_func *func)
383 {
384         int ret;
385
386         ret = sdio_read_cis(func->card, func);
387         if (ret)
388                 return ret;
389
390         /*
391          * Since we've linked to tuples in the card structure,
392          * we must make sure we have a reference to it.
393          */
394         get_device(&func->card->dev);
395
396         /*
397          * Vendor/device id is optional for function CIS, so
398          * copy it from the card structure as needed.
399          */
400         if (func->vendor == 0) {
401                 func->vendor = func->card->cis.vendor;
402                 func->device = func->card->cis.device;
403         }
404
405         return 0;
406 }
```

mmc_add_card(host->card);

让前面指定的父设备、总线生效，即在/sys/devices/platform/dwmmc@fe2c0000/mmc_host/下创建对应card节点

/rk3569_r/kernel/drivers/mmc/core/bus.c

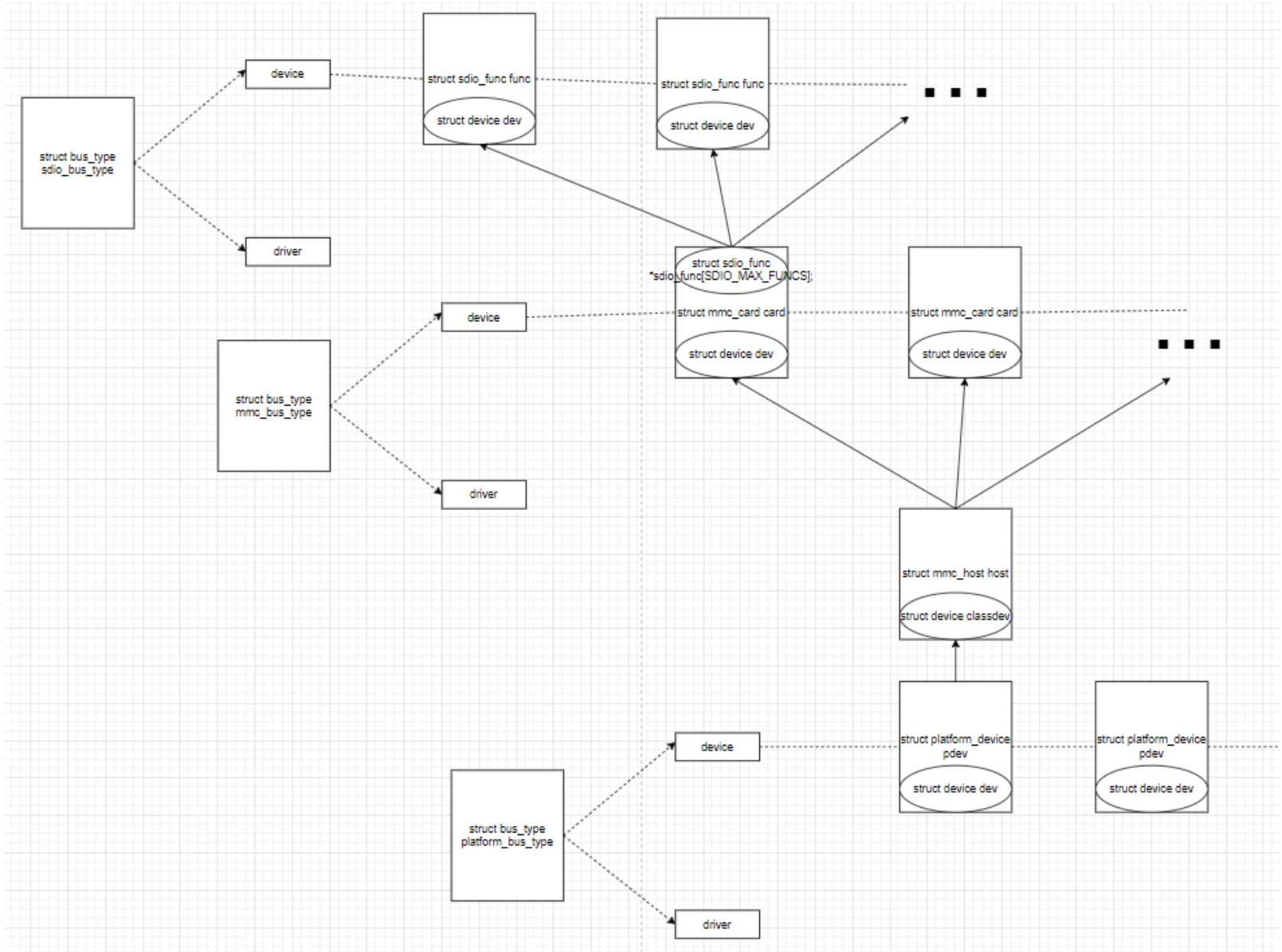
```
287  int mmc_add_card(struct mmc_card *card)
288  {
......
358      ret = device_add(&card->dev);
......
365  }
```

sdio_add_func(host->card->sdio_func[i]);

/rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

```
325  /*
326   * Register a new SDIO function with the driver model.
327   */
328 int sdio_add_func(struct sdio_func *func)
329  {
330          int ret;
331
332          dev_set_name(&func->dev, "%s:%d", mmc_card_id(func->card), func->num);
333
334          sdio_set_of_node(func);
335          sdio_acpi_set_handle(func);
336          device_enable_async_suspend(&func->dev);
337          ret = device_add(&func->dev);
338          if (ret == 0)
339                  sdio_func_set_present(func);
340
341          return ret;
342  }
```

10.3 关系拓扑图


struct bus_type
sdio_bus_type
device
driver
struct sdio_func func
struct device dev
struct sdio_func func
struct device dev
struct sdio_func
*sdio_func[SDIO_MAX_FUNCS];
struct mmc_card card
struct device dev
struct mmc_card card
struct device dev
struct bus_type
mmc_bus_type
device
driver
struct mmc_host host
struct device classdev
struct platform_device
pdev
struct device dev
struct platform_device
pdev
struct device dev
struct bus_type
platform_bus_type
device
driver

10.4 sdio bus

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

Home | Annotate | Line# | Scopes# | Navigate#

```
219
220  static struct bus_type sdio_bus_type = {
221          .name           = "sdio",
222          .dev_groups     = sdio_dev_groups,
223          .match          = sdio_bus_match,
224          .uevent         = sdio_bus_uevent,
225          .probe          = sdio_bus_probe,
226          .remove         = sdio_bus_remove,
227          .pm             = &sdio_bus_pm_ops,
228  };
```

sdio bus 注册:

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

Home | Annotate | Line# | Scopes# | Navigate

```
233  }
234
235  void sdio_unregister_bus(void)
236  {
237          bus_unregister(&sdio_bus_type);
238  }
```

xref: /rk3569_r/kernel/drivers/mmc/core/core.c


```
2827    {
2828            host->pm_notify.notifier_call = mmc_pm_notify;
2829            register_pm_notifier(&host->pm_notify);
2830    }
2831
2832    void mmc_unregister_pm_notifier(struct mmc_host *host)
2833    {
2834            unregister_pm_notifier(&host->pm_notify);
2835    }
2836    #endif
2837
2838    static int __init mmc_init(void)
2839    {
2840            int ret;
2841
2842            ret = mmc_register_bus();
2843            if (ret)
2844                    return ret;
2845
2846            ret = mmc_register_host_class();
2847            if (ret)
2848                    goto unregister_bus;
2849
2850            ret = sdio_register_bus();
2851            if (ret)
2852                    goto unregister_host_class;
2853
2854            return 0;
2855
2856    unregister_host_class:
2857            mmc_unregister_host_class();
2858    unregister_bus:
2859            mmc_unregister_bus();
2860            return ret;
2861    }
2862
```

```
2863  static void __exit mmc_exit(void)
2864  {
2865          sdio_unregister_bus();
2866          mmc_unregister_host_class();
2867          mmc_unregister_bus();
2868  }
2869
2870  subsys_initcall(mmc_init);
2871  module_exit(mmc_exit);
2872
2873  MODULE_LICENSE("GPL");
```

sdio bus 匹配sdio驱动和sdio设备:

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
 97
 98  static int sdio_bus_match(struct device *dev, struct device_driver *drv)
 99  {
100          struct sdio_func *func = dev_to_sdio_func(dev);
101          struct sdio_driver *sdrv = to_sdio_driver(drv);
102
103          if (sdio_match_device(func, sdrv))
104                  return 1;
105
106          return 0;
107  }
```

xref: /rk3569_r/kernel/include/linux/mmc/sdio_func.h

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
76  #define dev_to_sdio_func(d)  container_of(d, struct sdio_func, dev)
```

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
31
32  #define to_sdio_driver(d)    container_of(d, struct sdio_driver, drv)
```

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c


```
79
80  static const struct sdio_device_id *sdio_match_device(struct sdio_func *func,
81          struct sdio_driver *sdrv)
82  {
83          const struct sdio_device_id *ids;
84
85          ids = sdrv->id_table;
86
87          if (ids) {
88                  while (ids->class || ids->vendor || ids->device) {
89                          if (sdio_match_one(func, ids))
90                                  return ids;
91                          ids++;
92                  }
93          }
94
95          return NULL;
96  }
```

xref: /rk3569_r/kernel/drivers/mmc/core/sdio_bus.c

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
67
68  static const struct sdio_device_id *sdio_match_one(struct sdio_func *func,
69          const struct sdio_device_id *id)
70  {
71          if (id->class != (__u8)SDIO_ANY_ID && id->class != func->class)
72                  return NULL;
73          if (id->vendor != (__u16)SDIO_ANY_ID && id->vendor != func->vendor)
74                  return NULL;
75          if (id->device != (__u16)SDIO_ANY_ID && id->device != func->device)
76                  return NULL;
77          return id;
78  }
```

xref: /rk3569_r/kernel/include/linux/mod_devicetable.h

Home | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```
372 struct sdio_device_id {
373         __u8    class;                  /* Standard interface or SDIO_ANY_ID */
374         __u16   vendor;                 /* Vendor or SDIO_ANY_ID */
375         __u16   device;                 /* Device ID or SDIO_ANY_ID */
376         kernel_ulong_t driver_data;     /* Data private to the driver */
377 };
```

sdio驱动和sdio设备在sdio总线上匹配的规则是：

1、sdio驱动可以包含多个sdio_device_id结构体，只要任意一个sdio_device_id结构体能够与sdio设备的sdio_func结构体匹配则驱动与设备匹配。

2、sdio_device_id结构体与sdio_func结构体中的class、vendor、device都匹配才算匹配。

3、如果sdio_device_id结构体的class、vendor、device为SDIO_ANY_ID则无论sdio_func结构体对应的那项属性是什么，该项属性必然匹配，否则就需要sdio_device_id结构体与sdio_func结构体的class、vendor、device对应相等则匹配。

10.5 sdio驱动