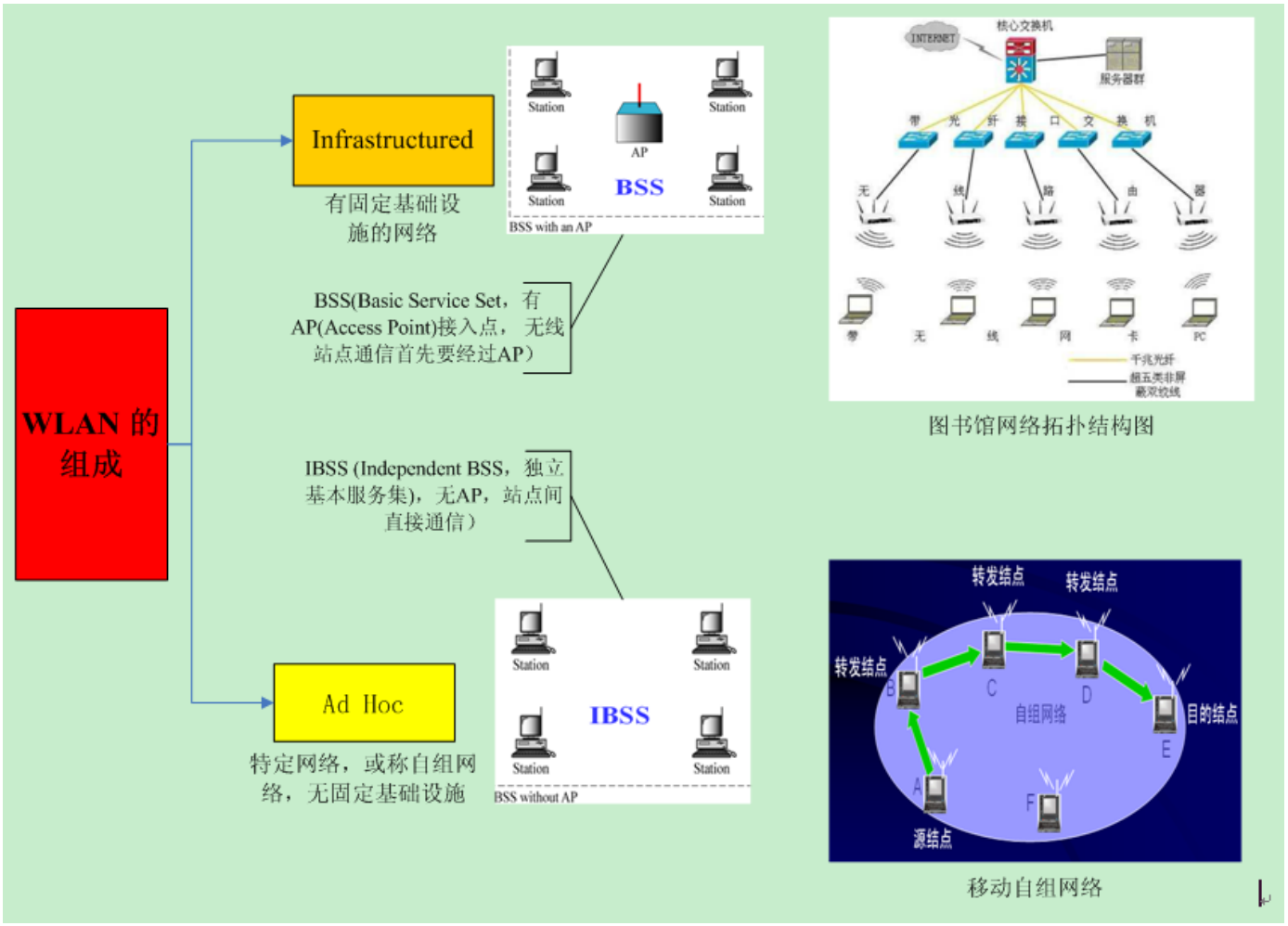
和菜鸟一起学android4.0.3源码之wifi的简单分析

2013-03-26 14:05 7311人阅读 评论(24) 收藏 举报

分类： Android源码学习之路（13）

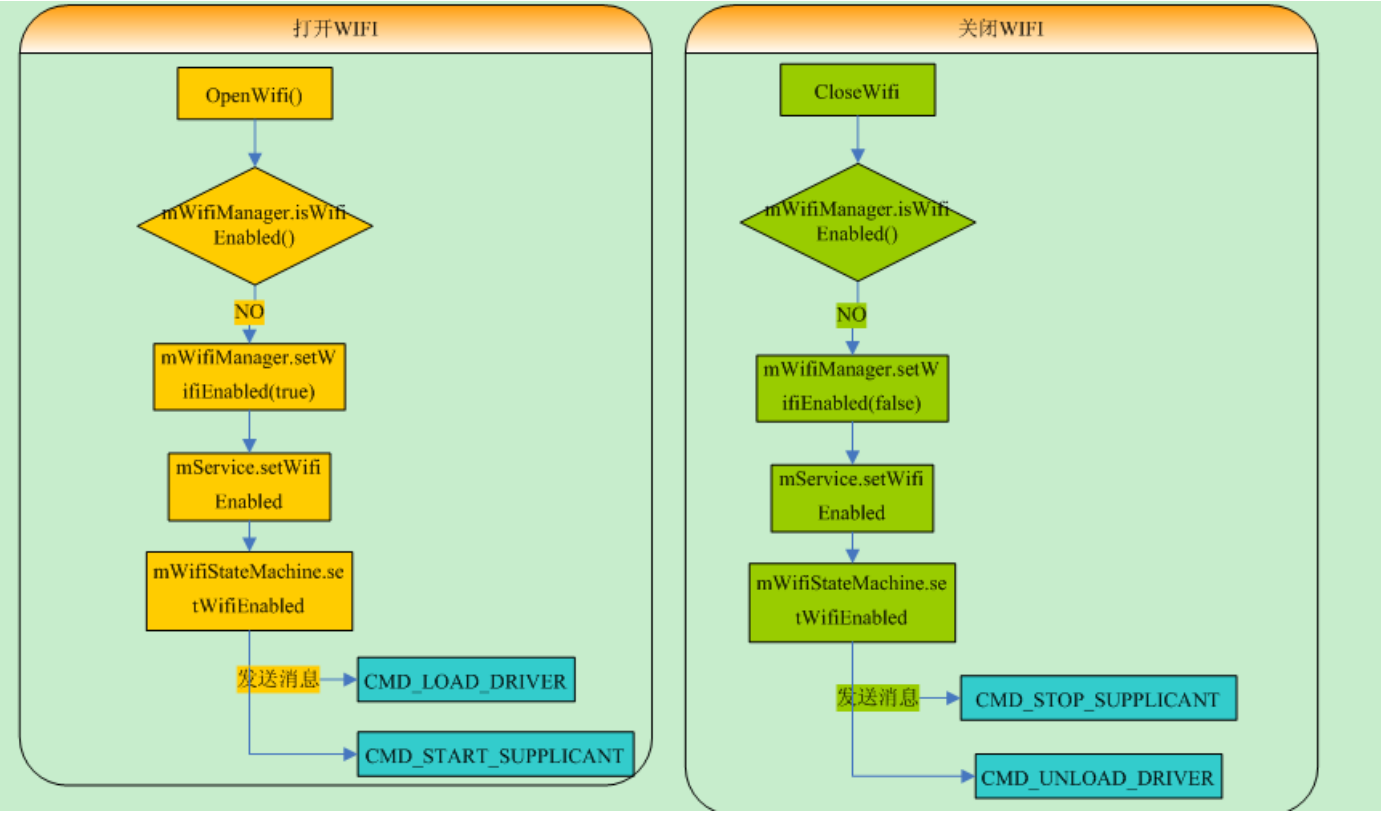
关于**wlan**的组成

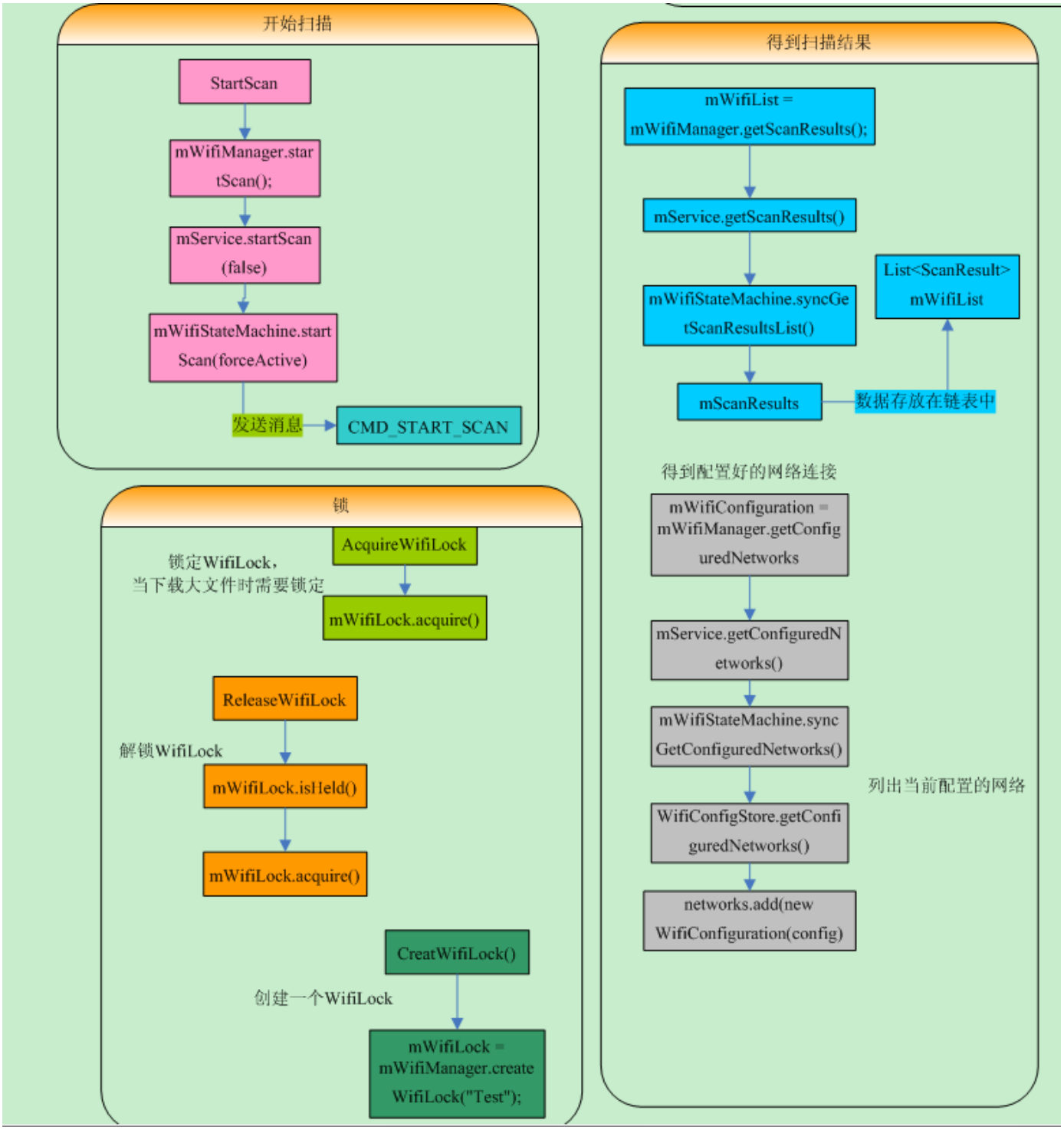


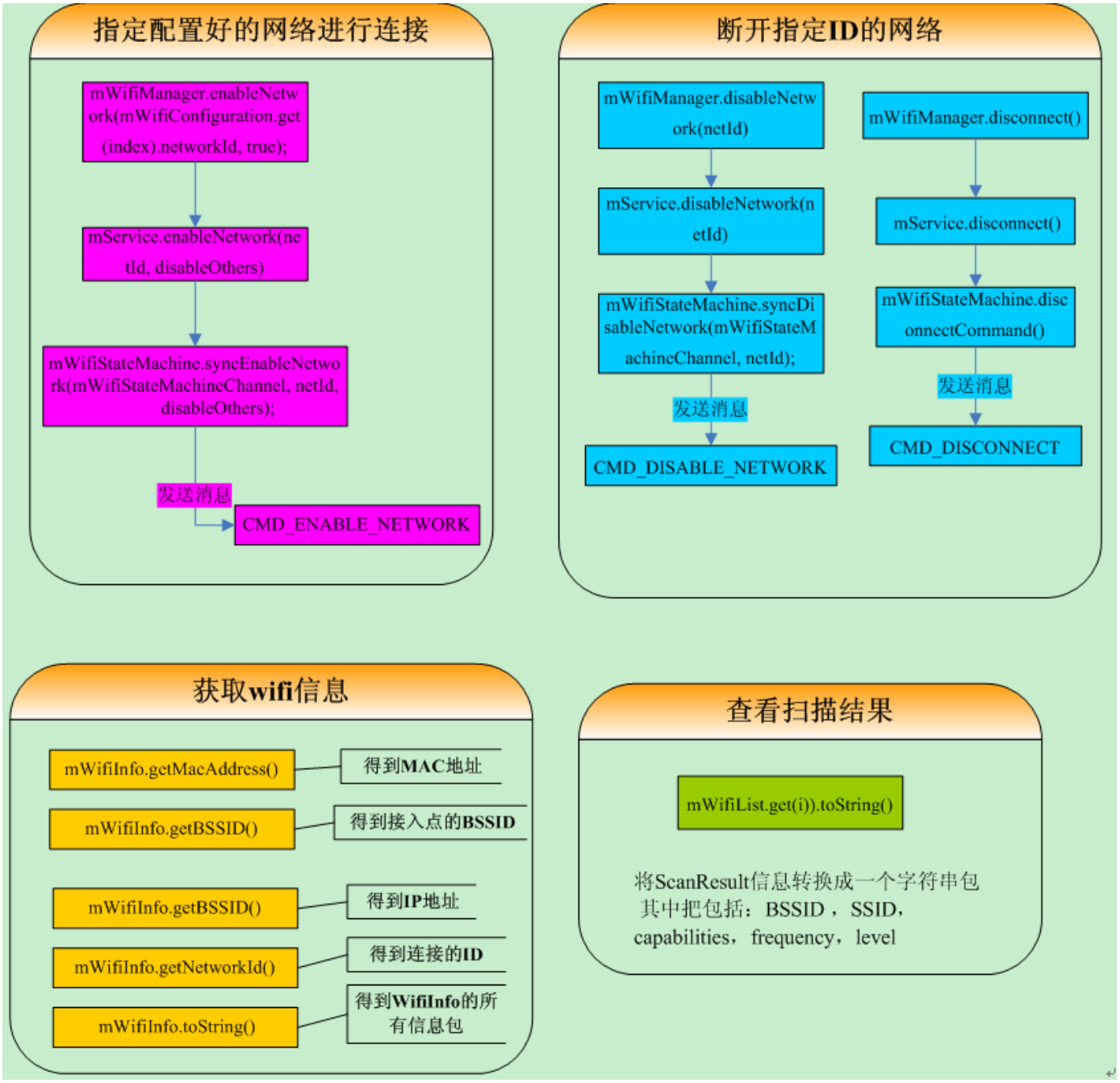
关于**wifi**应用层的接口的调用

首先从上层**Android** wifi的应用开始，首先会根据android的wifimanager的类，实例化一个mwifimanager的对象，这个对象处理了所有wifi需要处理的任务，接着比如说打开wifi，那么就会调用 mWifiManager.isWifiEnabled()；判断wifi是否已经打开，如果没有打开，那么就会调用mWifiManager.setWifiEnabled(true);来打开wifi了。这里会调用到wifiservice的方法，就是设置wifi使能，mService.setWifiEnabled。然后该函数会继续调用wifi的状态机中的设置使能的方法mWifiStateMachine.setWifiEnabled，而在wifi状态机中，这个方法主要是往这个状态机中发了两条消息，CMD_LOAD_DRIVER，和CMD_START_SUPPLICANT。看其意思就可以知道是加载驱动和启动wpa_supplicant了。剩下就是wifi状态机里做的事情了，这个下面分析。还有其他的比如关闭wifi啊，扫描

啊，连接网络啊，断开网络啊等等可以看下面的简单的流程图。具体的实现可以跟踪代码，就不一一介绍了。

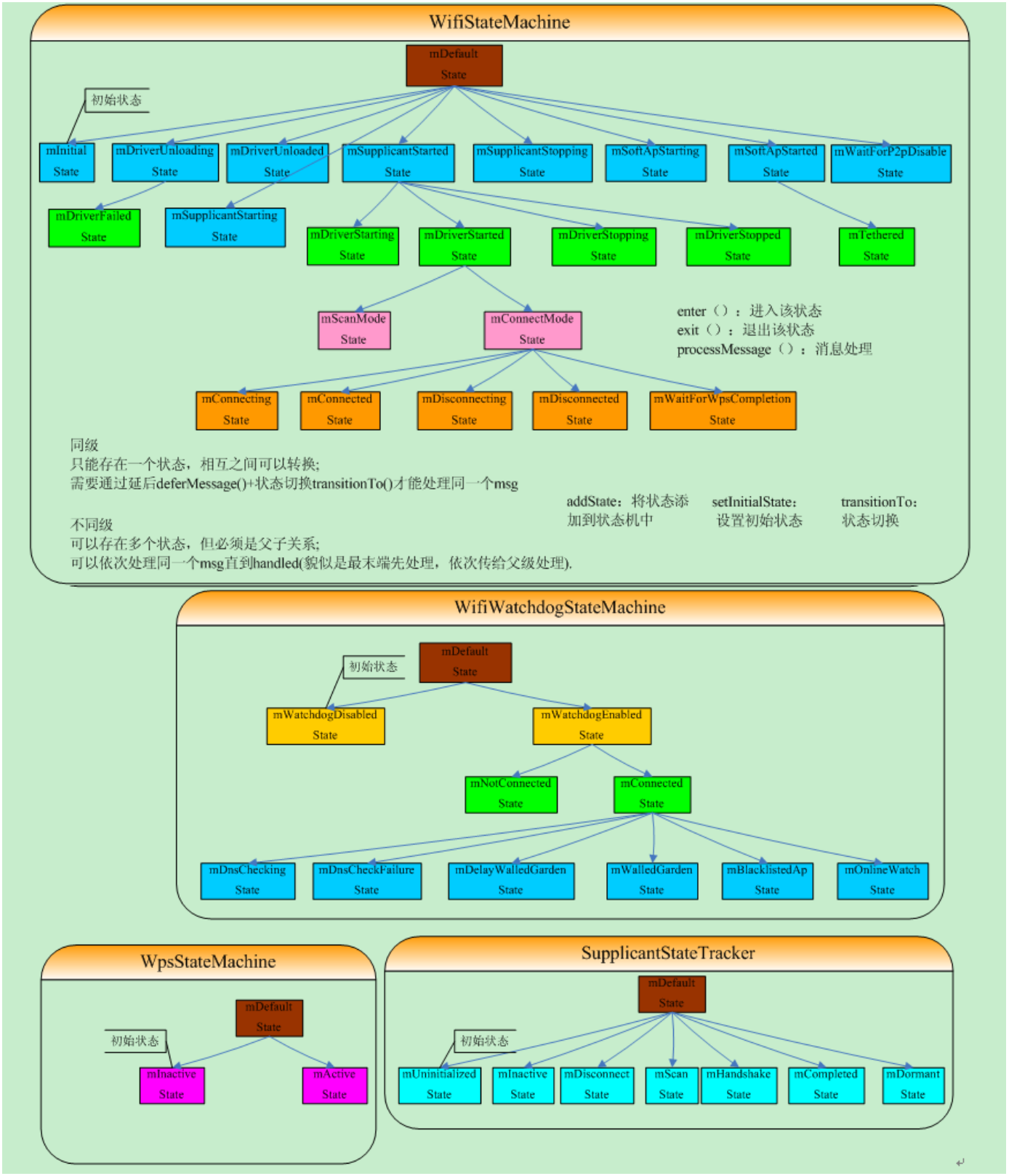



开始扫描
StartScan
mWifiManager.startScan();
mService.startScan(false)
mWifiStateMachine.startScan(forceActive)
发送消息
CMD_START_SCAN
得到扫描结果
mWifiList = mWifiManager.getScanResults();
mService.getScanResults()
mWifiStateMachine.syncGetScanResultsList()
List<ScanResult> mWifiList
mScanResults
数据存放在链表中
得到配置好的网络连接
mWifiConfiguration = mWifiManager.getConfiguredNetworks
mService.getConfiguredNetworks()
mWifiStateMachine.syncGetConfiguredNetworks()
列出当前配置的网络
WifiConfigStore.getConfiguredNetworks()
networks.add(new WifiConfiguration(config)
锁
AcquireWifiLock
锁定WifiLock，
当下载大文件时需要锁定
mWifiLock.acquire()
ReleaseWifiLock
解锁WifiLock
mWifiLock.isHeld()
mWifiLock.acquire()
CreatWifiLock()
创建一个WifiLock
mWifiLock = mWifiManager.createWifiLock("Test");



关于**wifi**的**framework**层状态机

Wifi framework层的主要的状态机可以参见下面的图了。



这里主要还是介绍下wifistatemachine的一些主要状态机的实现吧。

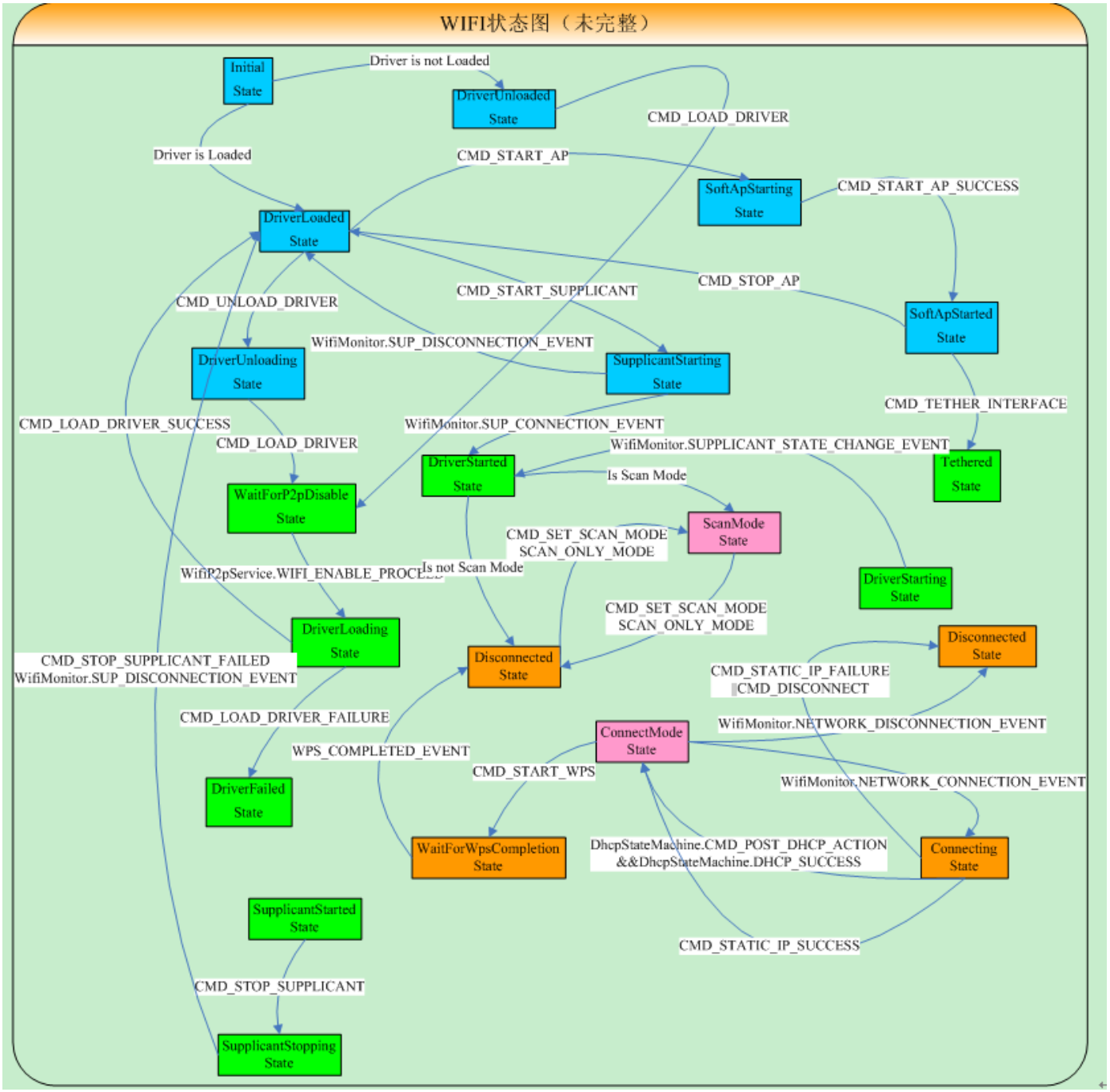
初始状态是initial，所以一开始执行initial的enter()，由于我们平台中，我移植的时候是wifi的驱动编译进内核的，所以WifiNative.isDriverLoaded()函数必须是true的。接着状态就转换为transitionTo(mDriverLoadedState)。

驱动加载的状态了。在这里的processMessage就会处理消息了。结合上面APP在开启wifi的时候发送的消息，CMD_START_SUPPLICANT，所以这里会处理开启wpa_supplicant。接着会下载firmware mNwService.wifiFirmwareReload(mInterfaceName,"STA");因为我们wifi驱动的firmware是驱动内部实现的，所以这里就不管他了。然后在WifiNative.startSupplicant()；这里调用了

hal层的，开启wpa_supplicant，具体下面分析。然后是mWifiMonitor.startMonitoring();开启一个monitor，主要还是处理wpa_supplicant往上报的事件的。接着，状态又转换了，就是
transitionTo(mSupplicantStartingState);
SupplicantStartingState状态了，这里如果wpa_supplicant启动成功的话，monitor那里上报一个WifiMonitor.SUP_CONNECTION_EVENT，接着就转换状态了，也就是transitionTo(mDriverStartedState);具体如下：

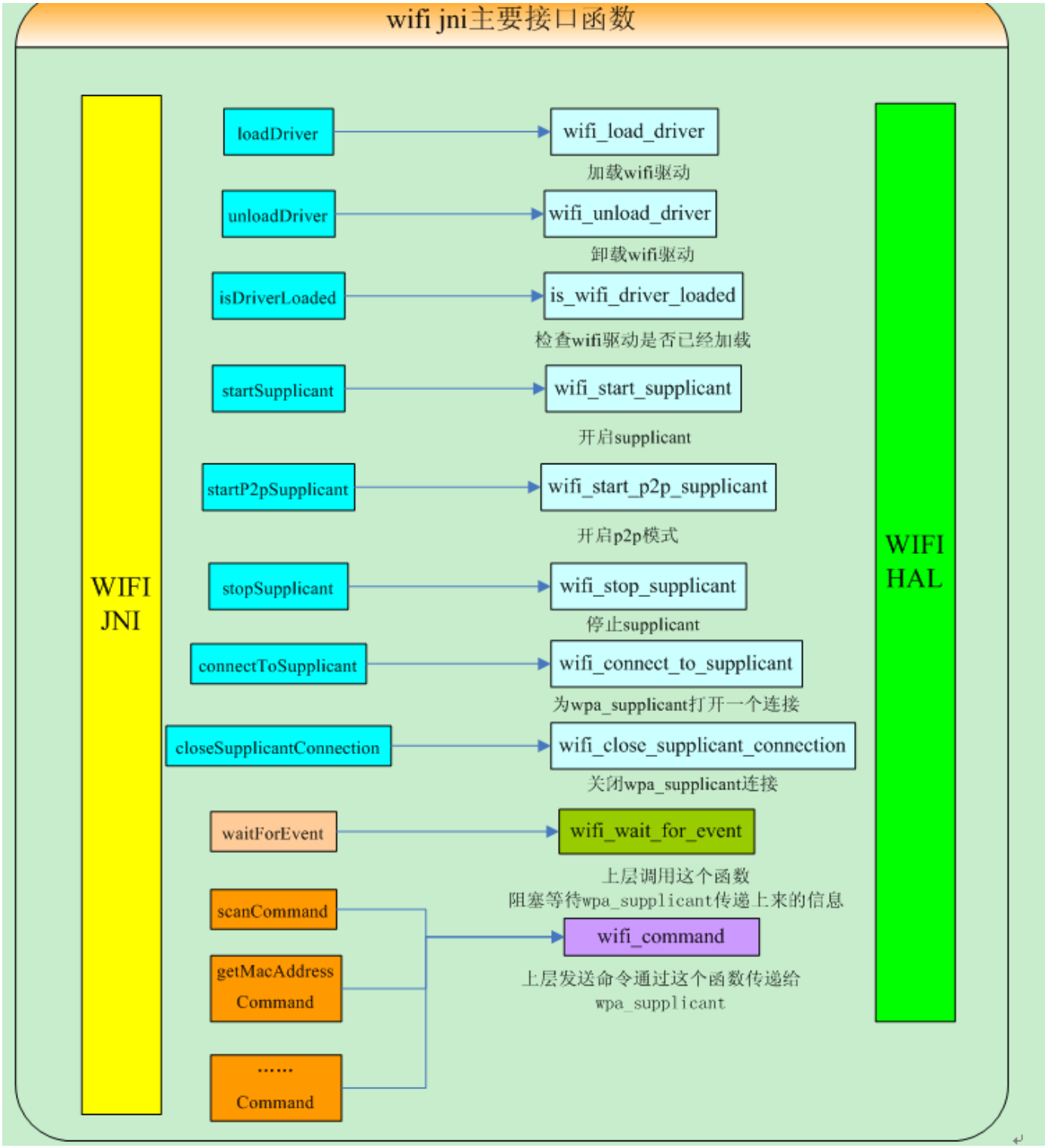
```
case WifiMonitor.SUP_CONNECTION_EVENT:
                if (DBG)log("Supplicant connection established");
                setWifiState(WIFI_STATE_ENABLED);
                mSupplicantRestartCount =0;
                /* Reset the supplicantstate to indicate the supplicant
                 * state is not known atthis time */
                mSupplicantStateTracker.sendMessage(CMD_RESET_SUPPLICANT_STATE);
                mWpsStateMachine.sendMessage(CMD_RESET_WPS_STATE);
                /* Initialize datastructures */
                mLastBssid = null;
                mLastNetworkId =WifiConfiguration.INVALID_NETWORK_ID;
                mLastSignalLevel = -1;

                mWifiInfo.setMacAddress(WifiNative.getMacAddressCommand());

                WifiConfigStore.initialize(mContext);

                sendSupplicantConnectionChangedBroadcast(true);
                transitionTo(mDriverStartedState);
```

这里在DriverStartedState状态下了。这里又做了很多处理：其中有一个CMD_START_SCAN，就是开始扫描。
对于状态机，下面画了大部分的状态图。



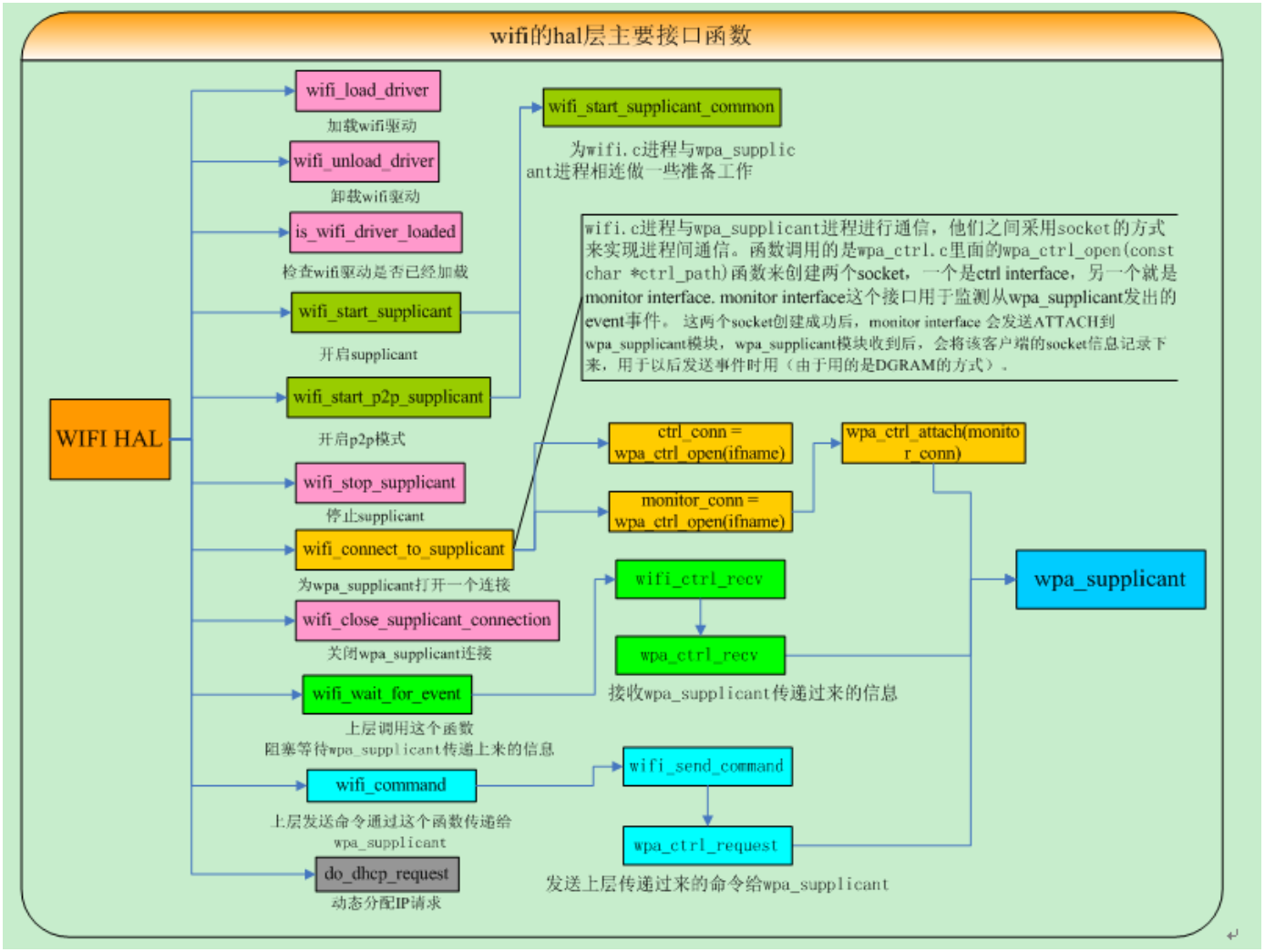
关于**wifi**的**jni**层

Jni层主要是封装了很多的接口，可以让上层的**Java**代码来调用hal层的c或者c++代码。这里他把命令和一些控制都封装好了。其主要接口可以看下图。

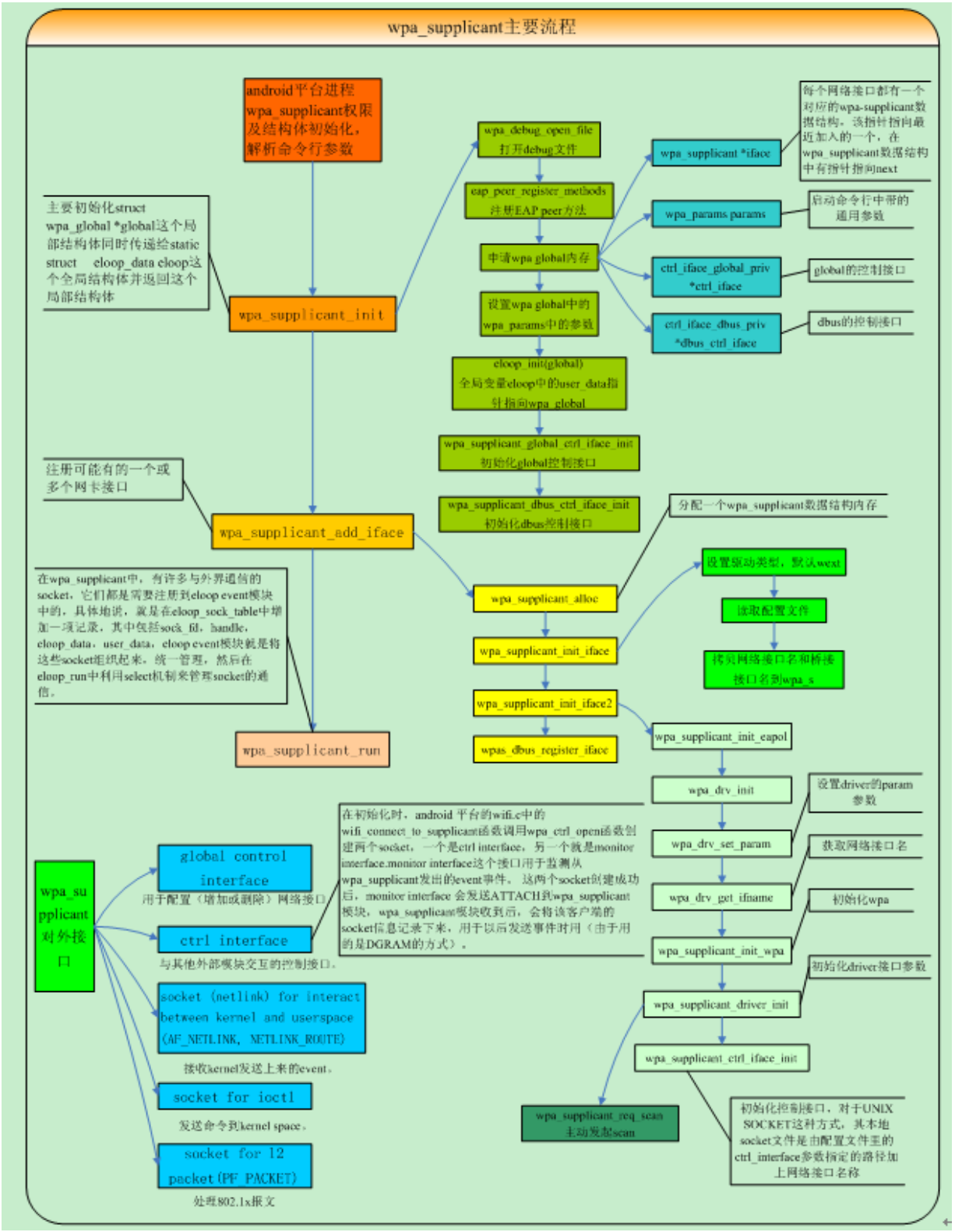


关于**wifi**的**hal**层

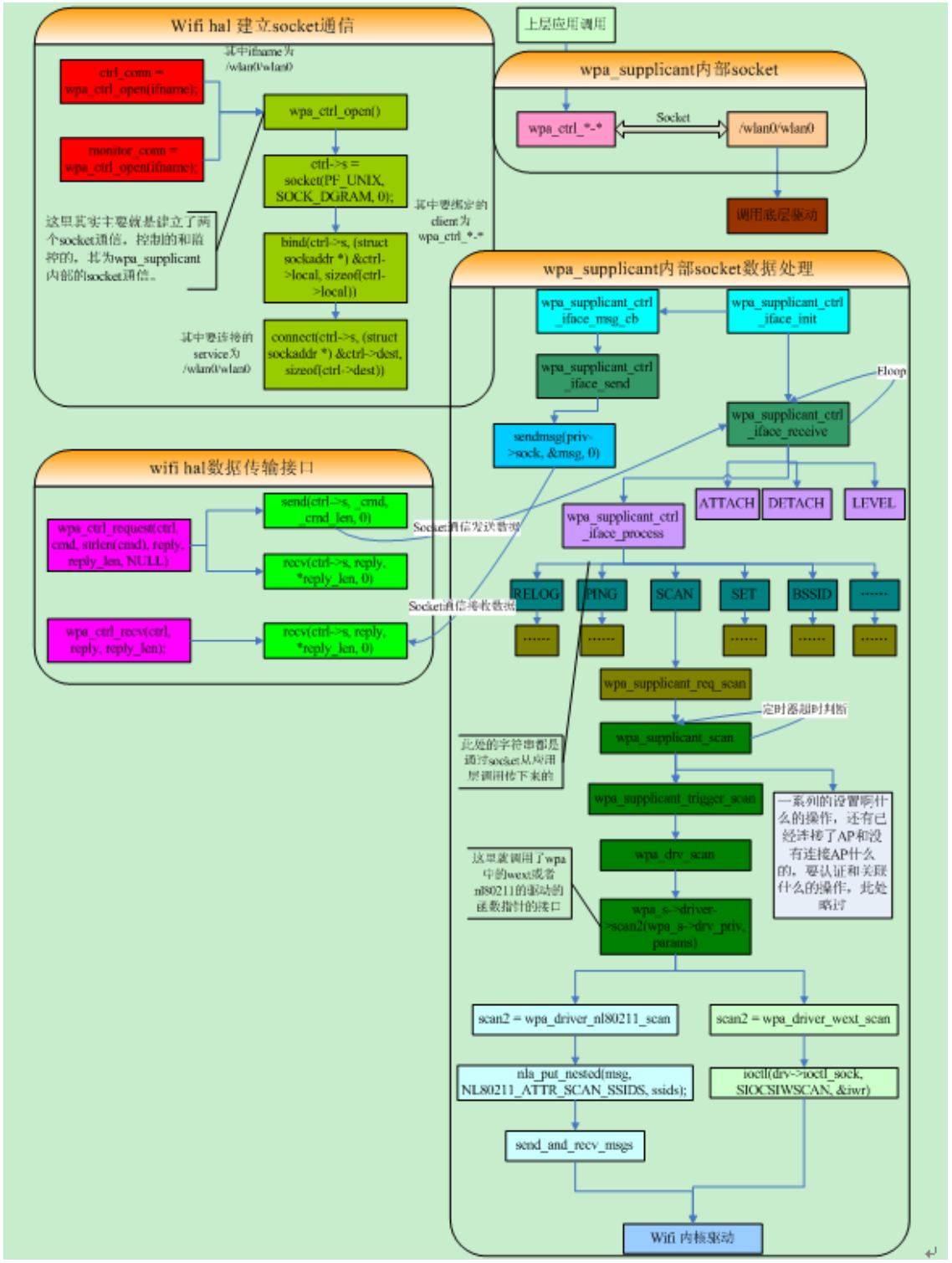
Wifi的hal层主要就是上层jni会调用到wpa_supplicant的，具体如下图所示。



关于**wpa_supplicant**的总工作流程



关于**wpa_supplicant**的**socket**与**hal**通信的流程



以上主要是以流程结合图形文字的方式简单的介绍了android的整个wifi的机制。具体可以结合代码分析。因为网上已经有很多的说明了，这里也不会太过分析。

关于**wifi**的**linux**驱动流程

可以参见《和菜鸟一起学linux之wifi学习记录》。这里已经对sdio wifi数据流程等做了简单的分析了。。

至此，对于android的整个wifi的机制，从宏观到微观做了简单的分析，相信基于这些，androidwifi的移植开发不再那么神秘而不可见了。