rk3308学习

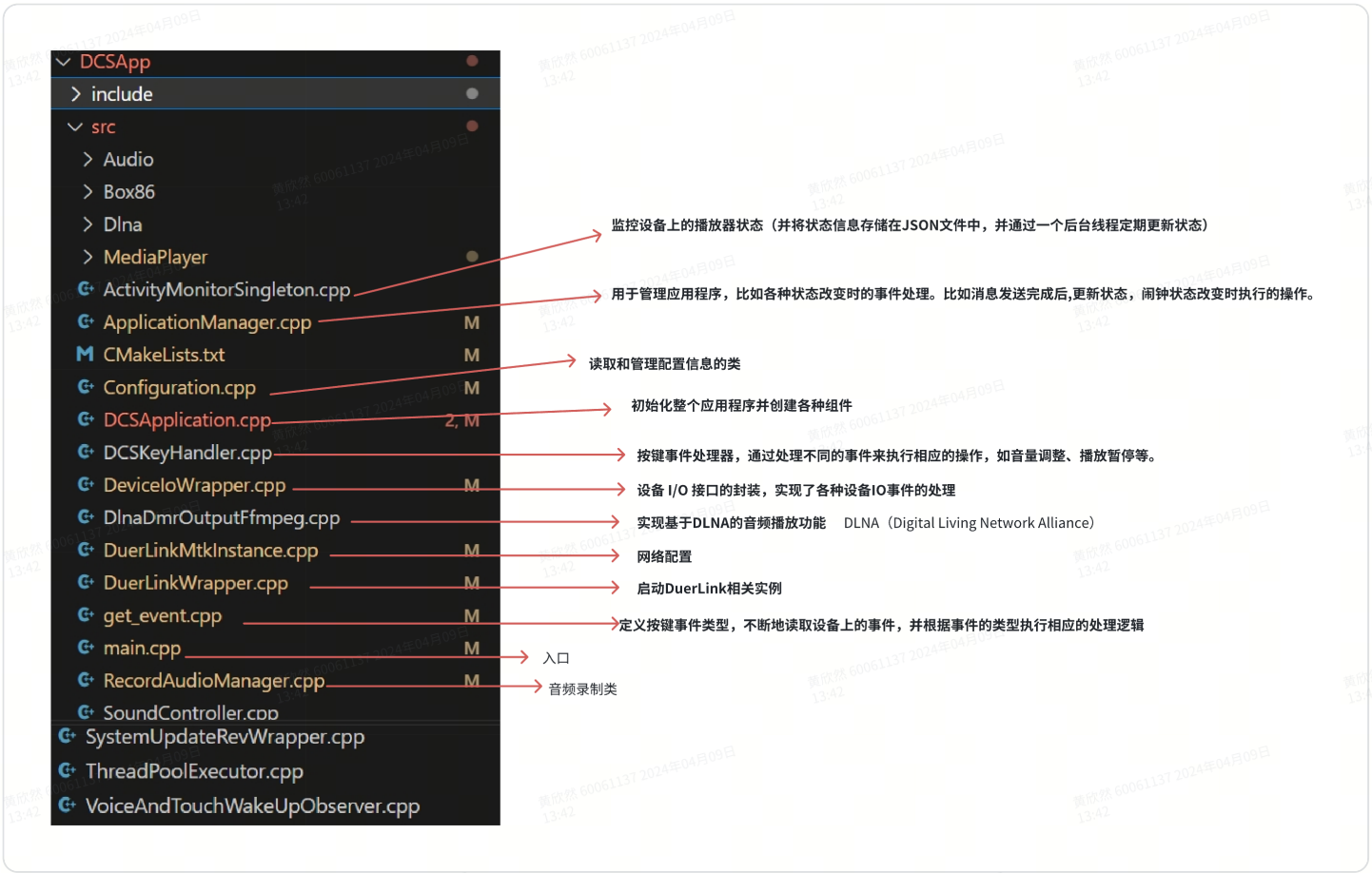
syslog

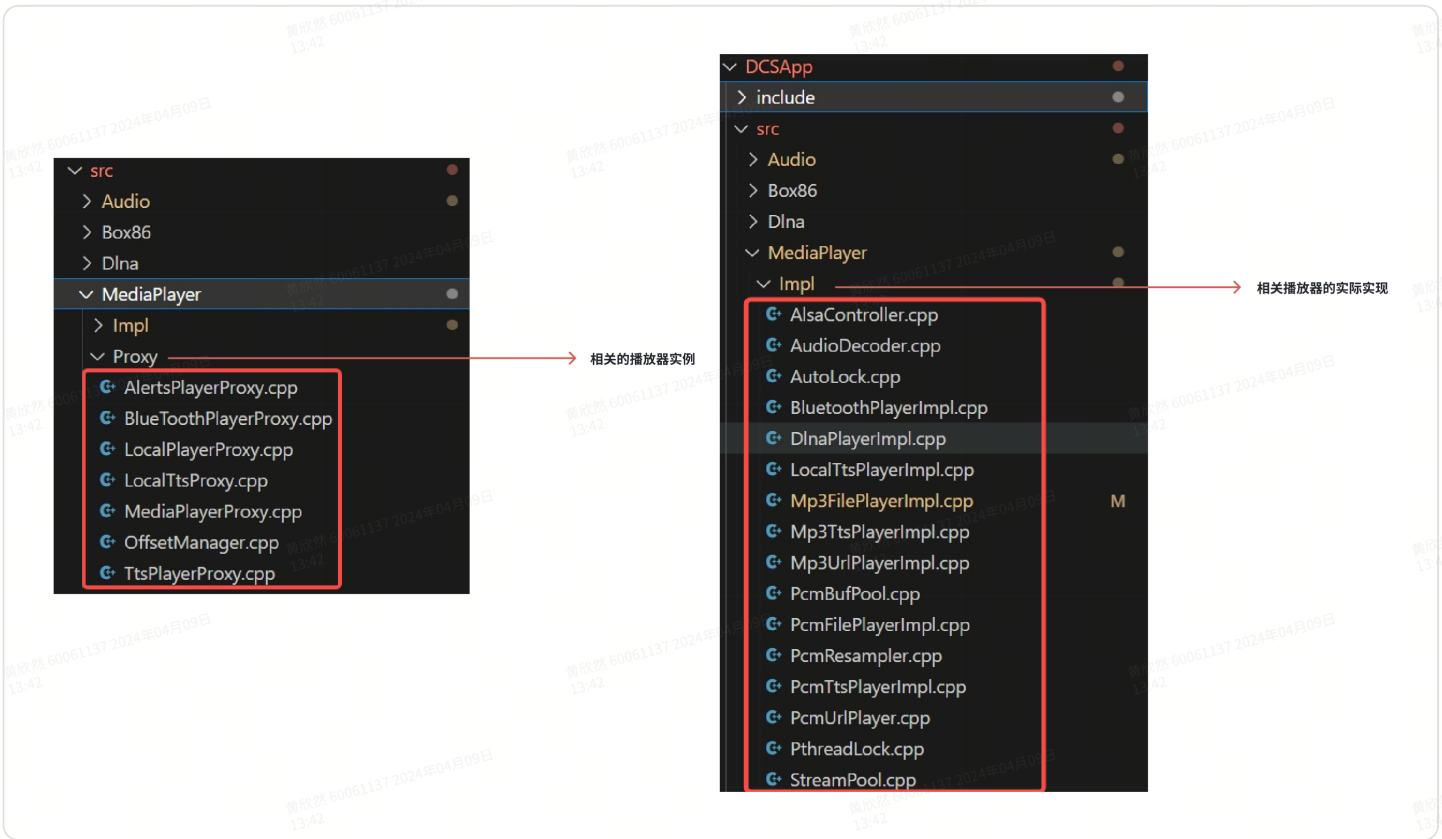
openlog("key_code_read",LOG_PID | LOG_PERROR | LOG_CONS, 0);

syslog(LOG_INFO,"start record test\n");

dueros

目录





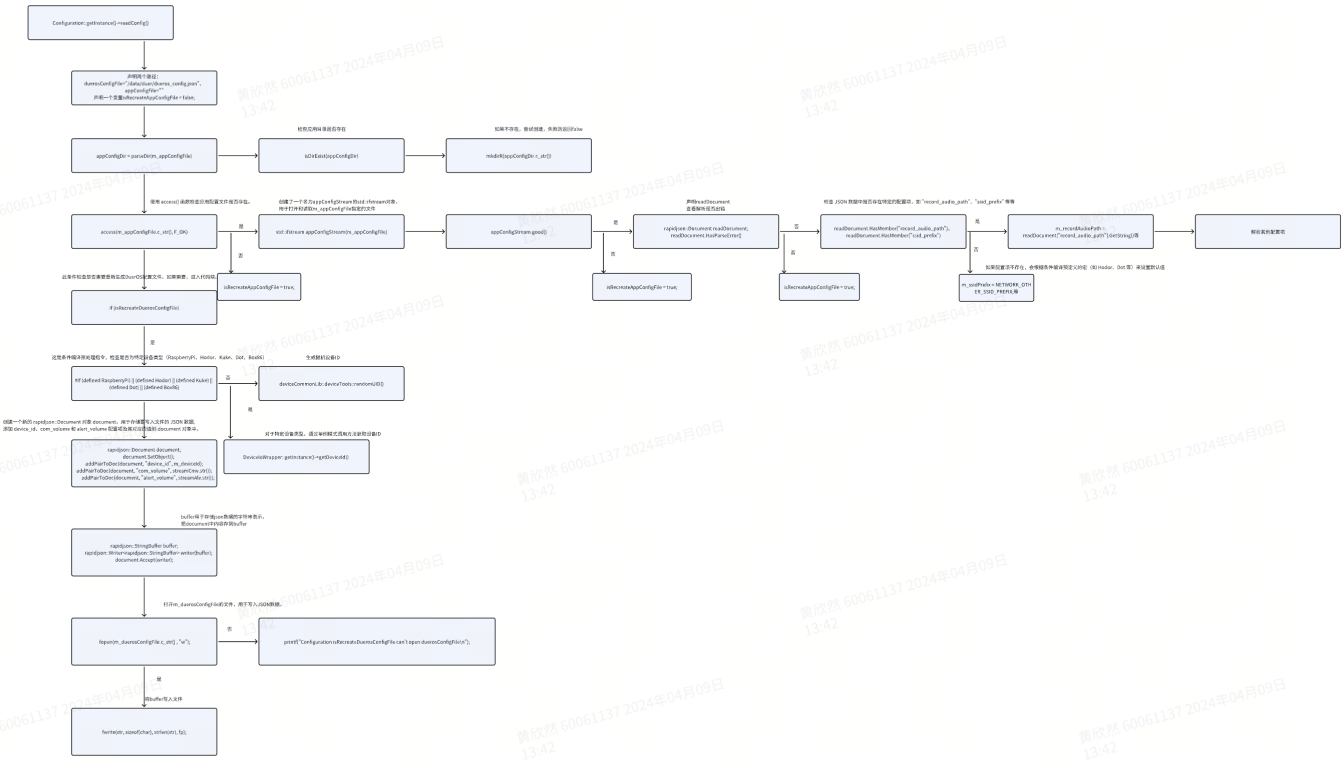
播放器实例

读取SampleApp配置文件

具体在：Configuration.h、Configuration.cpp

调用示例：

```
//读取sampleApp配置文件，返回bool类型值
duerOSDcsApp::application::Configuration::getInstance()->readConfig()
```



语音播放器实例

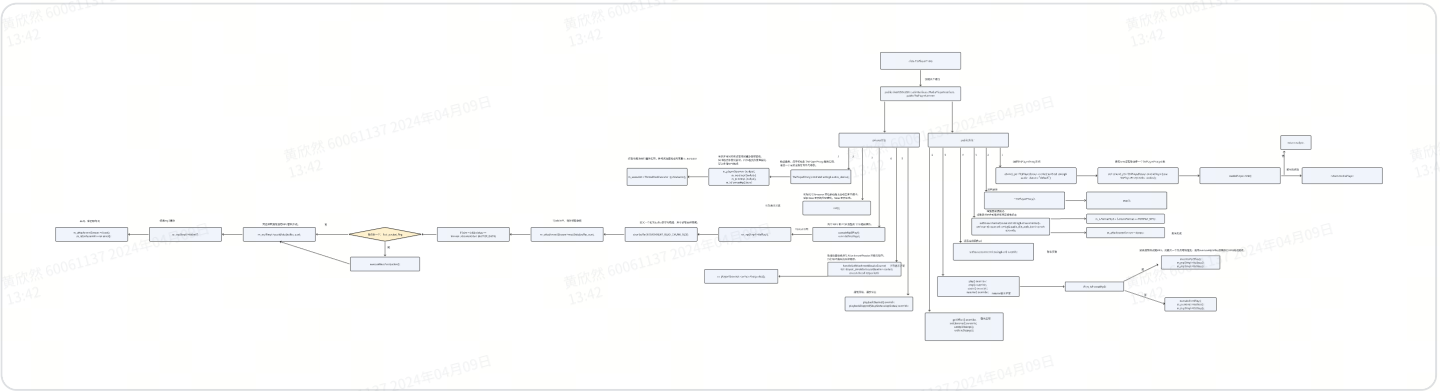
具体在：TtsPlayerProxy.h、TtsPlayerProxy.cpp

调用示例：

```
//创建语音播放器实例，传入参数为音频设备结点
auto speakMediaPlayer = mediaPlayer::TtsPlayerProxy::create(configuration->getTtsPlaybackDevice());
if (!speakMediaPlayer) {
```

```
4           APP_ERROR("Failed to create media player for speech!");
5           return false;
6   }
```

TtsPlayerProxy类的实现

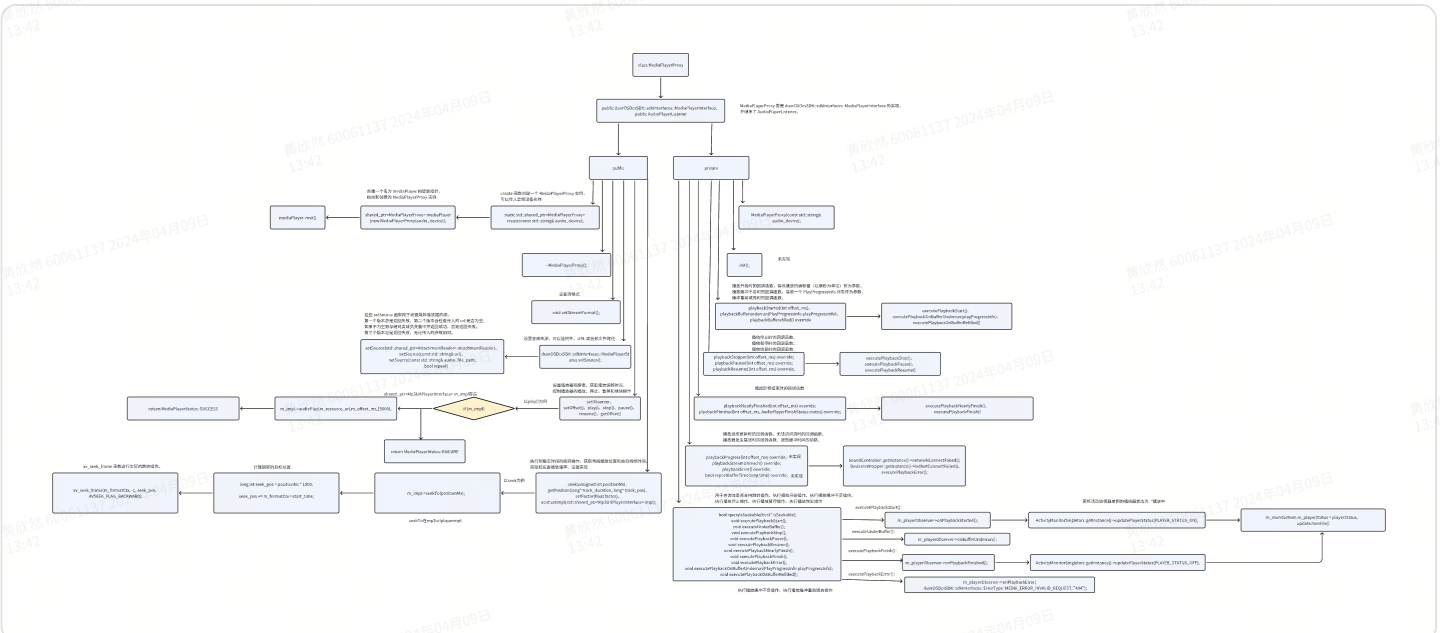


音乐播放器实例

具体在：MediaPlayerProxy.h、MediaPlayerProxy.cpp

调用示例：

```
1  auto audioMediaPlayer = mediaPlayer::MediaPlayerProxy::create(configuration-
   >getMusicPlaybackDevice());
2      if (!audioMediaPlayer) {
3          APP_ERROR("Failed to create media player for content!");
4          return false;
5      }
```



当你调用 auto audioMediaPlayer = mediaPlayer::MediaPlayerProxy::create(configuration->getMusicPlaybackDevice()); 时，以下是调用流程的大致过程：

通过命名空间 mediaPlayer 中的 MediaPlayerProxy 类，使用 create 函数创建了一个名为 audioMediaPlayer 的智能指针。在 create 函数内部，创建了一个 MediaPlayerProxy 实例，并传入了配置中的音乐播放设备参数。然后调用了实例的 init 函数。

init 函数目前只是简单地返回 true，并没有实际的初始化逻辑。

返回这个已初始化的 MediaPlayerProxy 实例作为智能指针。

此后，可以使用 audioMediaPlayer 智能指针来操作媒体播放器，调用其成员函数，例如 play、pause、stop 等来控制媒体播放的行为。根据不同的调用，会触发对应的函数执行，例如在播放开始时会调用 executePlaybackStart 函数，停止播放时会调用 executePlaybackStop 函数，等等。

闹钟闹铃播放器实例

调用示例：

```
1 //闹钟闹铃播放器实例
2 auto alertsMediaPlayer = mediaPlayer::AlertsPlayerProxy::create(configuration->getAlertPlaybackDevice());
```

它通过创建 AlertsPlayerProxy 的实例来代表警报播放功能。
这个类提供了用于初始化、设置音频源、播放、暂停、恢复、停止等功能的方法。
它使用一个名为 Mp3FilePlayerInterface 的接口实现来实际处理音频播放。
类中的方法通过调用内部的 Mp3FilePlayerInterface 实例来实现音频播放操作。
MediaPlayerObserverInterface 注册播放器观察者。

提示音播放器实例

调用示例：

```
1 //提示音播放器实例
2 auto localMediaPlayer = mediaPlayer::LocalPlayerProxy::create(configuration->getInfoPlaybackDevice());
```

本地语音合成播放器节点

调用示例：

```
1 //本地语音合成播放器节点
2 auto localTtsPlayer = mediaPlayer::LocalTtsProxy::create(configuration-
  >getNattsPlaybackDevice());
```

蓝牙播放器

在BlueToothPlayerProxy.h和BlueToothPlayerProxy.cpp

调用示例：

```
1 //蓝牙播放器
2 auto blueToothPlayer = mediaPlayer::BlueToothPlayerProxy::create();
```

1、静态成员函数create

```
std::shared_ptr<BlueToothPlayerProxy> BlueToothPlayerProxy::create() {
    APP_INFO("BlueToothPlayerProxy createCalled");
    std::shared_ptr<BlueToothPlayerProxy> bluetoothPlayer(new BlueToothPlayerProxy());

    if (!bluetoothPlayer) {
        APP_ERROR("BlueToothPlayerProxy Create Failed");
        return nullptr;
    } else {
        return bluetoothPlayer;
    }
}
```

它执行以下步骤：
记录日志信息"BlueToothPlayerProxy createCalled"
使用 new 操作符创建 BlueToothPlayerProxy 实例。
检查实例是否创建成功。如果失败，记录错误日志并返回空指针。
如果成功，返回包装在 std::shared_ptr 中的实例指针。

2、类的构造函数

```
BlueToothPlayerProxy::BlueToothPlayerProxy() : m_localMediaPlayerPlayActivity {STOPPED},
                                               m_bluetoothImpl {nullptr},
m_dlnaImpl {nullptr} {
    APP_ERROR("BlueToothPlayerProxy Constructor");
}
```

执行以下操作：
记录一个错误日志。
初始化成员变量 m_localMediaPlayerPlayActivity 为 STOPPED。
初始化指向蓝牙和 DLNA 实现的指针为 nullptr。

执行代码后可看到打印：

```
:1680000 ms> BlueToothPlayerProxy createCalled
:1680000 ms> BlueToothPlayerProxy Constructor
```

3、一系列用于控制媒体播放的函数：代码中有一系列用于播放(play)、暂停(stop)、恢复(resume)、停止(pause)等操作的函数，这些函数根据不同的播放器名称（蓝牙或 DLNA）来执行相应的操作。

以play()为例：

```
LocalMediaPlayerStatus BlueToothPlayerProxy::play(const LocalMediaPlayerName& name) {
    APP_INFO("BlueToothPlayerProxy play");
    m_localMediaPlayerPlayActivity = PLAYING;
    if (BLUETOOTH == name) {
        if (m_bluetoothImpl) {
            m_bluetoothImpl->play();
        }
    } else if (DLNA == name) {
        if (m_dlnaImpl) {
            m_dlnaImpl->play();
        }
    }
    return LocalMediaPlayerStatus::LOCAL_MEDIA_PLAYER_SUCCESS;
}
```

4、一系列用于处理 DLNA 播放器状态的函数：这些函数用于处理 DLNA 播放器的不同状态，例如开始播放、停止播放、暂停播放等。

```
void BlueToothPlayerProxy::dlnaStartPlay() {
    APP_INFO("BlueToothPlayerProxy dlnaStartPlay11111");
    if (m_playerObserver) {
        LocalMediaPlayerPlayInfo playInfo;
        playInfo.m_playerActivity = PLAYING;
        playInfo.m_status = FOREGROUND;
        playInfo.m_playerName = DLNA;
        playInfo.m_audioId = "";
        playInfo.m_title = "";
        playInfo.m_artist = "";
        playInfo.m_album = "";
        playInfo.m_year = "";
        playInfo.m_genre = "";
        m_playerObserver->setLocalMediaPlayerPlayInfo(playInfo);
    }

    if (m_dcsSdk) {
#ifdef KITTAI_KEY_WORD_DETECTOR
        m_dcsSdk->enterPlayMusicScene();
#endif
    }
}
```

5、一系列用于处理蓝牙播放器状态的函数：类似于上面的函数，这些函数用于处理蓝牙播放器的不同状态。

```
void BlueToothPlayerProxy::btStartPlay() {
    APP_INFO("BlueToothPlayerProxy btStartPlay");
    application::DeviceIoWrapper::deviceioWrapperBtSoundPlayFinished();
    if (m_playerObserver) {
        LocalMediaPlayerPlayInfo playInfo;
        playInfo.m_playerActivity = PLAYING;
        playInfo.m_status = FOREGROUND;
        playInfo.m_playerName = BLUETOOTH;
        playInfo.m_audioId = "";
        playInfo.m_title = "";
        playInfo.m_artist = "";
        playInfo.m_album = "";
        playInfo.m_year = "";
        playInfo.m_genre = "";
        m_playerObserver->setLocalMediaPlayerPlayInfo(playInfo);
    }

    if (m_dcsSdk) {
#ifdef KITTAI_KEY_WORD_DETECTOR
        m_dcsSdk->enterPlayMusicScene();
#endif
    }
}
```

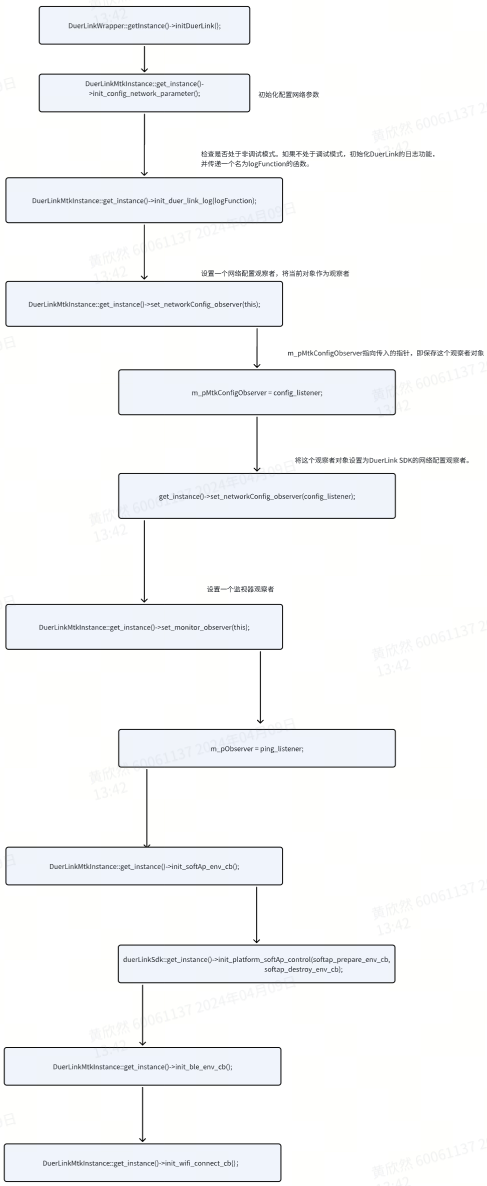
启动DuerLink相关实例

具体在：DuerLinkWrapper.h、DuerLinkWrapper.cpp

初始化Duerlink相关实例

调用示例：

```
1 //初始化DuerLink相关实例
2 DuerLinkWrapper::getInstance()->initDuerLink();
```

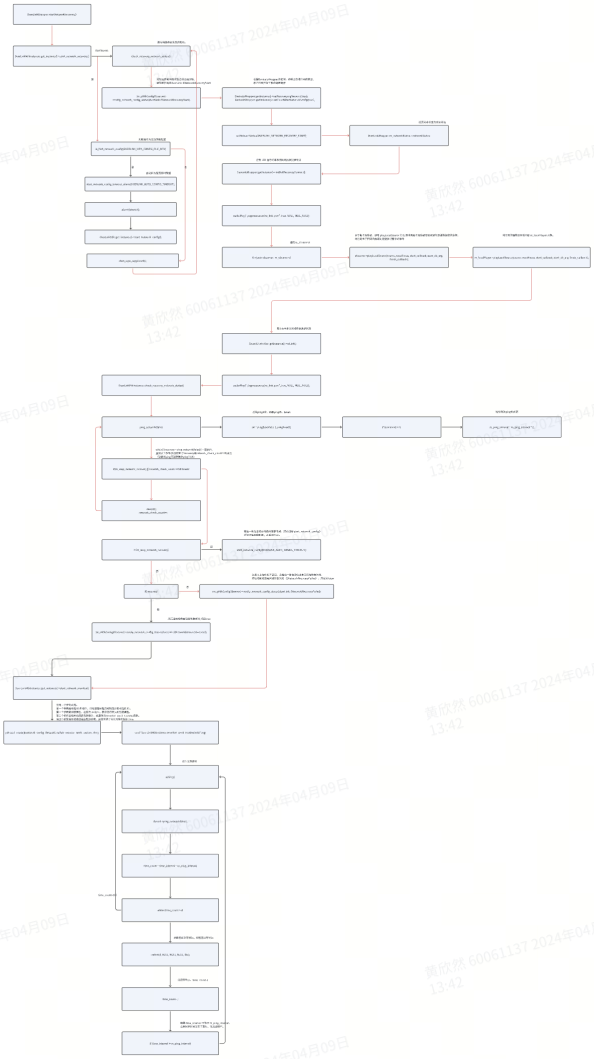


启动配网相关功能

调用示例：

```
1 //启动配网相关功能
```

```
DuerLinkWrapper::getInstance()->startDiscoverAndBound(m_dcsSdk->getClientId());
```



配置wifi

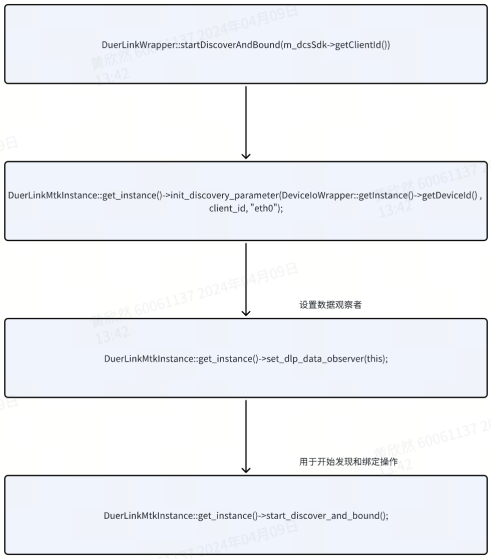
```
连接上设备后
在/oem/userdata/cfg/wpa_supplicant.conf
将network中的ssid和psk分别改成自己的wifi热点的名字和密码，注意热点要开4G的
```

启动设备发现和DLP相关功能

调用示例：

```
//启动设备发现和DLP相关功能
```

```
DuerLinkWrapper::getInstance()->startDiscoverAndBound(m_dcsSdk->getClientId());
```



POCO

```
参考:
http://www.stibel.icu/md/develop/library/library-poco.html
```

ESP-BLE-MESH 架构

```
参考:
https://docs.espressif.com/projects/esp-idf/zh_CN/latest/esp32/api-guides/esp-ble-mesh/ble-mesh-architecture.html
```