

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

▼ DUERCLIENTSDK [SSH: 10.0.21.155]

> .vscode

> appresources

> build

> DCSApp

> doc_cn

> Framework

> resources

> resources32

> resources64

> Rk3308_buildout

> sdk

> third

> .gitignore

> build_third.sh

> build.sh

> CMakeLists.txt

> README.md

> release-notes

→ SampleApp资源目录

→ 相关交叉编译工具链配置

→ SampleApp源代码目录

→ DCS SDK资源目录

→ DCS SDK相关头文件和so库

→ DCS SDK第三方依赖库

Project structure of DCSApp:

- include
- src
 - Audio
 - Box86
 - Dlna
 - MediaPlayer
 - ActivityMonitorSingleton.cpp → 监控设备上的播放器状态 (并将状态信息存储在JSON文件中, 并通过一个后台线程定期更新状态)
 - ApplicationManager.cpp → 用于管理应用程序, 比如各种状态改变时的事件处理。比如消息发送完成后, 更新状态, 闹钟状态改变时执行的操作。
 - CMakeLists.txt → 读取和管理配置信息的类
 - Configuration.cpp → 初始化整个应用程序并创建各种组件
 - DCSApplication.cpp → 2, M
 - DCSKeyHandler.cpp → 按键事件处理器, 通过处理不同的事件来执行相应的操作, 如音量调整、播放暂停等。
 - DeviceIoWrapper.cpp → M → 设备 I/O 接口的封装, 实现了各种设备IO事件的处理
 - DlnaDmrOutputFfmpeg.cpp → 实现基于DLNA的音频播放功能 DLNA (Digital Living Network Alliance)
 - DuerLinkMtkInstance.cpp → M → 网络配置
 - DuerLinkWrapper.cpp → M → 启动DuerLink相关实例
 - get_event.cpp → M → 定义按键事件类型, 不断地读取设备上的事件, 并根据事件的类型执行相应的处理逻辑
 - main.cpp → M → 入口
 - RecordAudioManager.cpp → M → 音频录制类
 - SoundController.cpp
 - SystemUpdateRevWrapper.cpp
 - ThreadPoolExecutor.cpp
 - VoiceAndTouchWakeUpObserver.cpp

Project structure of DCSApp (continued):

- src
 - Audio
 - Box86
 - Dlna
 - MediaPlayer
 - Impl → 相关的播放器实例
 - AlertsPlayerProxy.cpp
 - BluetoothPlayerProxy.cpp
 - LocalPlayerProxy.cpp
 - LocalTtsProxy.cpp
 - MediaPlayerProxy.cpp
 - OffsetManager.cpp
 - TtsPlayerProxy.cpp
 - Proxy

Project structure of DCSApp (continued):

- include
- src
 - Audio
 - Box86
 - Dlna
 - MediaPlayer
 - Impl → 相关播放器的实际实现
 - AlsaController.cpp
 - AudioDecoder.cpp
 - AutoLock.cpp
 - BluetoothPlayerImpl.cpp
 - DlnaPlayerImpl.cpp
 - LocalTtsPlayerImpl.cpp
 - Mp3FilePlayerImpl.cpp → M
 - Mp3TtsPlayerImpl.cpp
 - Mp3UrlPlayerImpl.cpp
 - PcmBufPool.cpp
 - PcmFilePlayerImpl.cpp
 - PcmResampler.cpp
 - PcmTtsPlayerImpl.cpp
 - PcmUrlPlayer.cpp
 - PthreadLock.cpp
 - StreamPool.cpp

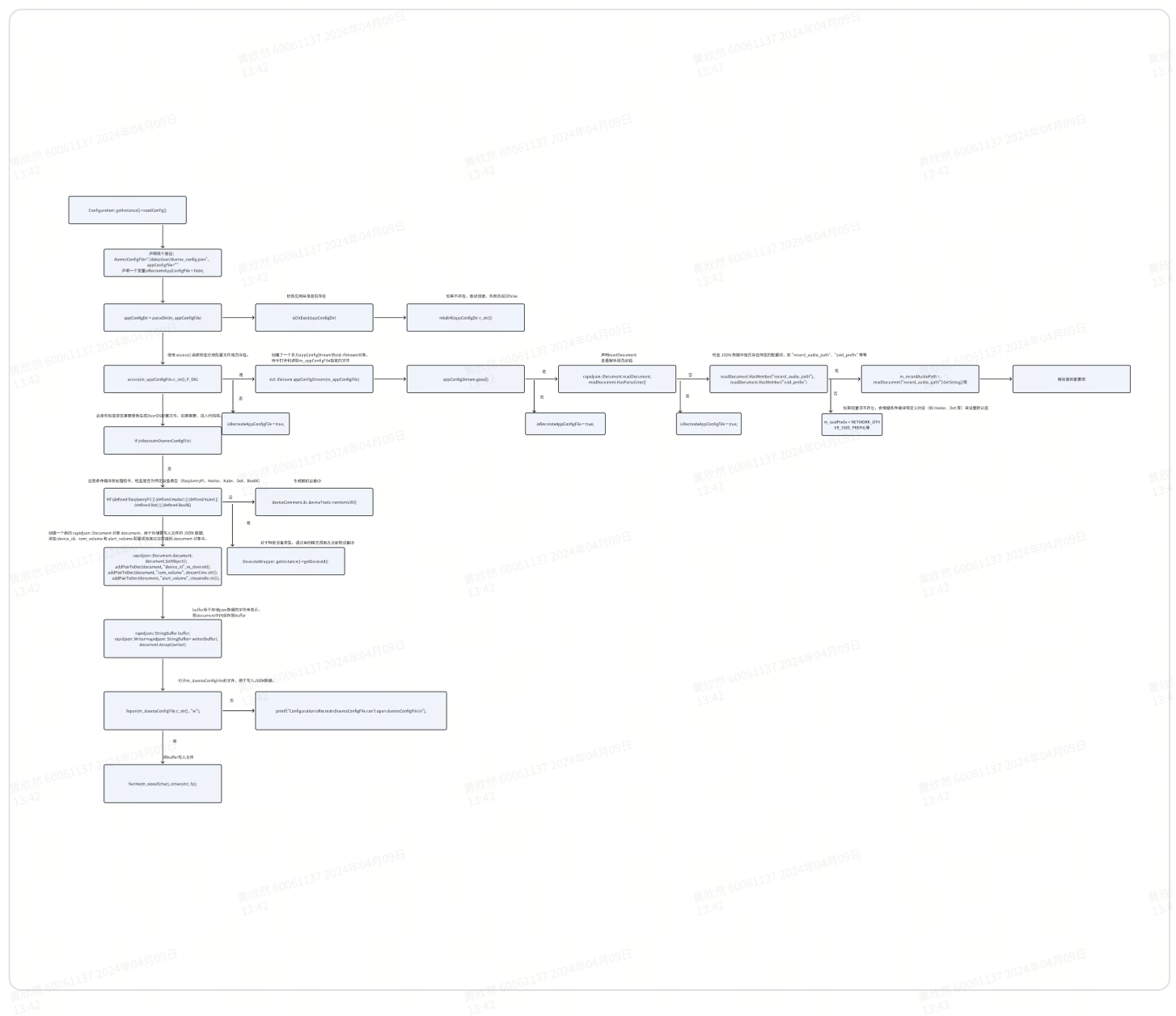
播放器实例

读取SampleApp配置文件

具体在: Configuration.h、Configuration.cpp

调用示例：

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



语音播放器实例

具体在：TtsPlayerProxy.h、TtsPlayerProxy.cpp

调用示例：

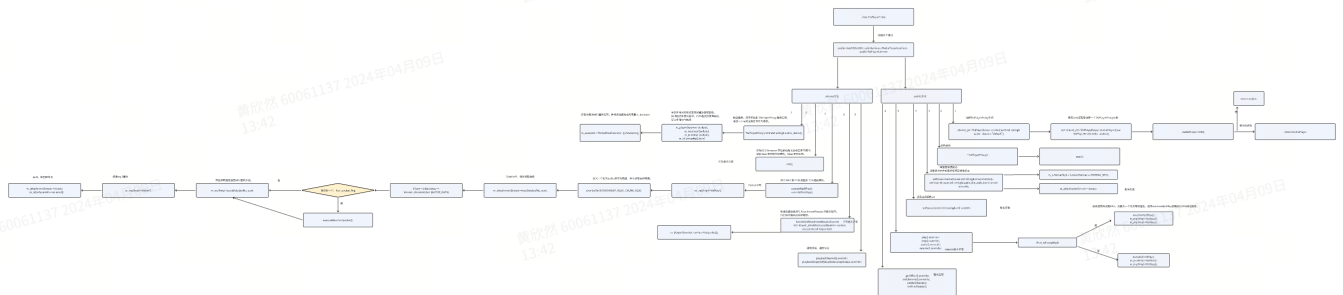
- ```
1 //创建语音播放器实例，传入参数为音频设备结点
2 auto speakMediaPlayer = mediaPlayer::TtsPlayerProxy::create(configuration-
 >getTtsPlaybackDevice());
3 if (!speakMediaPlayer) {
```

```

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

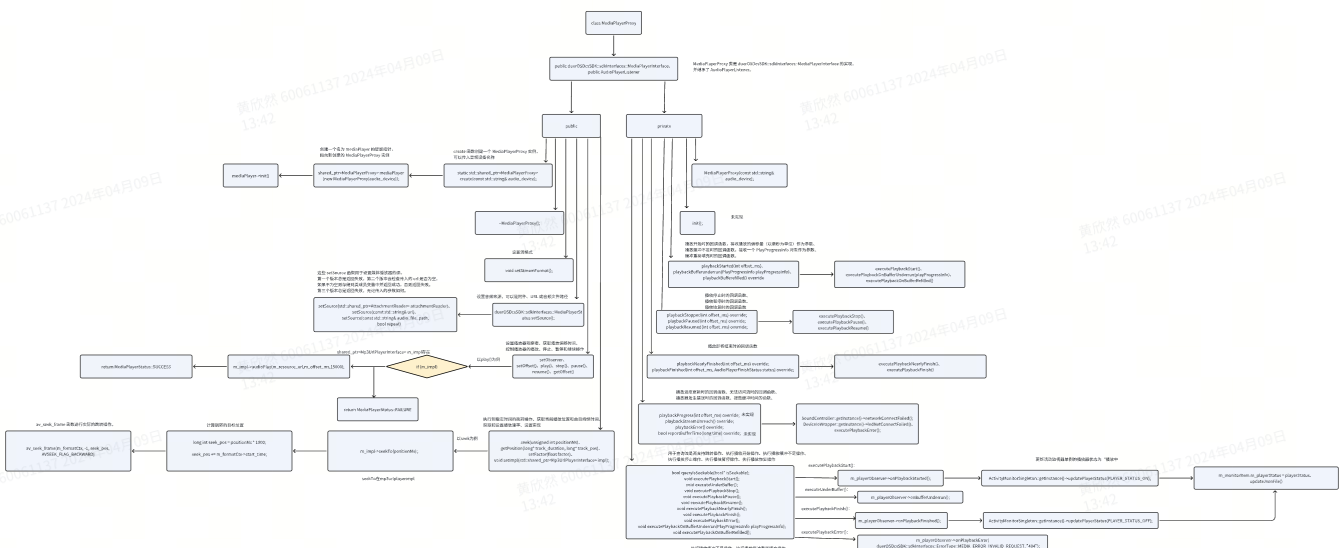
具体在：MediaPlayerProxy.h、MediaPlayerProxy.cpp

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
 }
}

```



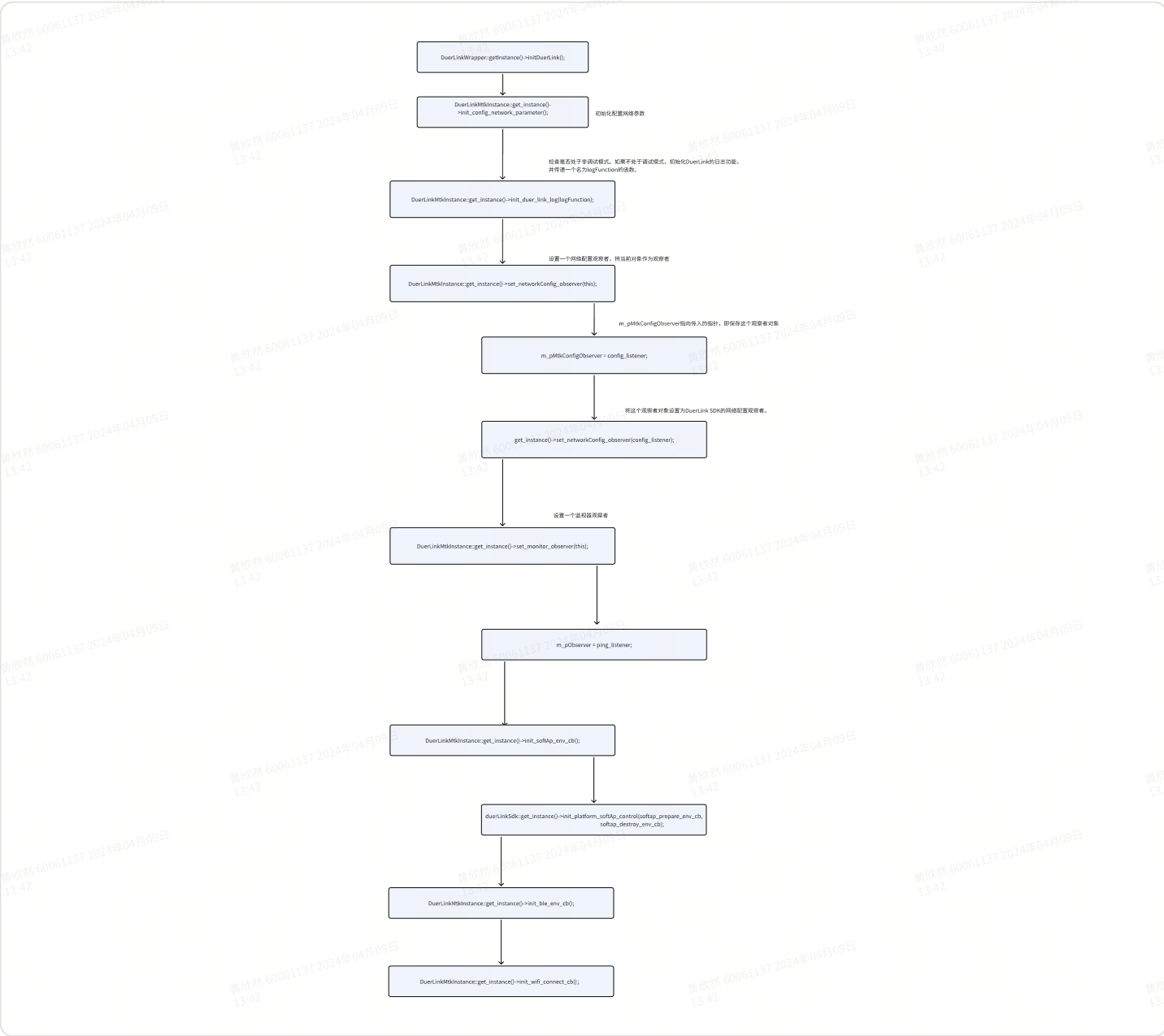
# 启动DuerLink相关实例

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();
```

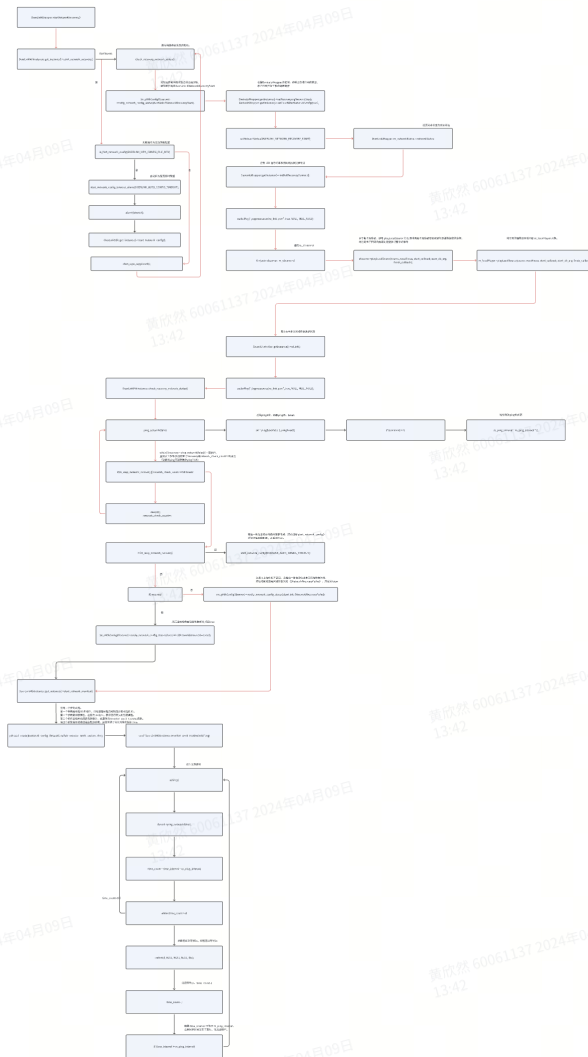


## 启动配网相关功能

调用示例：

```
1 //启动配网相关功能
```

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



## 配置wifi

- 1 连接上设备后
- 2 在/oem/userdata/cfg/wpa\_supplicant.conf
- 3 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：

- 1 //启动设备发现和DLP相关功能

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



## POCO

1 参考:

2 <http://www.stibel.icu/md/develop/library/library-poco.html>

## ESP-BLE-MESH 架构

1 参考:

2 [https://docs.espressif.com/projects/esp-idf/zh\\_CN/latest/esp32/api-guides/esp-ble-mesh/ble-mesh-architecture.html](https://docs.espressif.com/projects/esp-idf/zh_CN/latest/esp32/api-guides/esp-ble-mesh/ble-mesh-architecture.html)

