|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |  |
| 内核扩展组件和外设 | | |
|  |  | |
|  |  | |
| 华为网格系统---方案4-032.png | | | | |
|  | 华为技术有限公司 | | 附件1-16K |  |

|  |
| --- |
| 版权所有 © 华为技术有限公司2025。 保留一切权利。  本材料所载内容受著作权法的保护，著作权由华为公司或其许可人拥有，但注明引用其他方的内容除外。未经华为公司或其许可人事先书面许可，任何人不得将本材料中的任何内容以任何方式进行复制、经销、翻印、播放、以超级链路连接或传送、存储于信息检索系统或者其他任何商业目的的使用。  商标声明  附件3-版权声明页图、附件3-版权声明页图、附件3-版权声明页图、华为，以上为华为公司的商标（非详尽清单），未经华为公司书面事先明示许可，任何第三方不得以任何形式使用。  注意  华为会不定期对本文档的内容进行更新。  本文档仅作为使用指导，文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |
| --- |
| 华为技术有限公司 |

目 录

[1 内核扩展组件和外设 4](#_Toc1536)

[1.1 课程提要 4](#_Toc20967)

[1.1.1 课程目标 4](#_Toc24161)

[1.1.2 主要内容 4](#_Toc20371)

[1.1.3 重难点分析 4](#_Toc17038)

[1.2 扩展组件 5](#_Toc13979)

[1.2.1 C++支持 5](#_Toc20840)

[1.2.2 CPU占用率 6](#_Toc16980)

[1.2.3 动态加载 8](#_Toc30980)

[1.2.4 文件系统 13](#_Toc4174)

[1.3 外设 18](#_Toc20818)

[1.3.1 WLAN 18](#_Toc3787)

[1.3.2 Touchscreen 22](#_Toc21825)

# 内核扩展组件和外设

## 课程提要

### 课程目标

介绍OpenHarmony的内核扩展组件以及外设。

### 主要内容

1. 1.1章节将主要介绍OpenHarmony扩展组件，并将分别从基本概念、接口参数、开发案例讲解C++支持、CPU占用率、动态加载和文件系统。
2. 1.2章节将介绍OpenHarmony支持的外设，包括：WLAN和Touchscreen。

### 重难点分析

1. 开发者需要了解CPU占率用的概念和计算。
2. 开发者需要了解动态加载中的ELF方案。

## 扩展组件

### C++支持

#### 基本概念

C++作为目前使用最广泛的编程语言之一，支持类、封装、重载等特性，是在C语言基础上开发的一种面向对象的编程语言。

#### 运行机制

C++代码的识别主要由编译器支持，系统主要对全局对象进行构造函数调用，进行初始化操作。

#### 开发指导

1. 接口说明

表 1-1 C++支持接口

|  |  |  |
| --- | --- | --- |
| 功能分类 | 接口名 | 描述 |
| 使用C++特性的前置条件 | LOS\_CppSystemInit | C++构造函数初始化 |

1. 开发流程

使用C++特性之前，需要调用函数LOS\_CppSystemInit，实现C++构造函数初始化，其中被初始化的构造函数存在init\_array这个段中，段区间通过变量\_\_init\_array\_start\_\_、\_\_init\_array\_end\_\_传递。

表 1-2 参数说明

|  |  |
| --- | --- |
| 参数 | 参数说明 |
| \_\_init\_array\_start\_\_ | init\_array段起始位置 |
| \_\_init\_array\_end\_\_ | init\_array段结束位置 |

 说明： 调用该函数时，一定要在c++业务前。另外部分与系统资源强相关的类或接口，如std::thread，std::mutex等，在三方编译器使用的c库非musl c时，存在兼容性问题，不建议使用。

### CPU占用率

#### 基本概念

CPU（中央处理器，Central Processing Unit）占用率分为系统CPU占用率和任务CPU占用率。

系统CPU占用率（CPU Percent）是指周期时间内系统的CPU占用率，用于表示系统一段时间内的闲忙程度，也表示CPU的负载情况。系统CPU占用率的有效表示范围为0～100，其精度（可通过配置调整）为百分比。100表示系统满负荷运转。

任务CPU占用率指单个任务的CPU占用率，用于表示单个任务在一段时间内的闲忙程度。任务CPU占用率的有效表示范围为0～100，其精度（可通过配置调整）为百分比。100表示在一段时间内系统一直在运行该任务。

用户通过系统级的CPU占用率，判断当前系统负载是否超出设计规格。

通过系统中各个任务的CPU占用情况，判断各个任务的CPU占用率是否符合设计的预期。

运行机制

OpenHarmony LiteOS-M的CPUP（CPU Percent，系统CPU占用率）采用任务级记录的方式，在任务切换时，记录任务启动时间，任务切出或者退出时间，每次当任务退出时，系统会累加整个任务的占用时间。

可以在target\_config.h的中对该功能进行选配。

1. OpenHarmony LiteOS-M提供以下两种CPU占用率的信息查询：

* 系统CPU占用率。
* 任务CPU占用率。

1. CPU占用率的计算方法：

系统CPU占用率=系统中除idle任务外其他任务运行总时间/系统运行总时间

任务CPU占用率=任务运行总时间/系统运行总时间

#### 开发指导

1. 接口说明

表 1-3 功能列表

|  |  |  |
| --- | --- | --- |
| 功能分类 | 接口名 | 描述 |
| 获取系统CPU占用率 | LOS\_SysCpuUsage | 获取当前系统CPU占用率 |
| LOS\_HistorySysCpuUsage | 获取系统历史CPU占用率 |
| 获取任务CPU占用率 | LOS\_TaskCpuUsage | 获取指定任务CPU占用率 |
| LOS\_HistoryTaskCpuUsage | 获取指定任务历史CPU占用率 |
| LOS\_AllCpuUsage | 获取所有任务CPU占用率 |
| 输出任务CPU占用率 | LOS\_CpupUsageMonitor | 输出任务历史CPU占用率 |

1. 开发流程

CPU占用率的典型开发流程：

* 调用获取系统CPU使用率函数LOS\_SysCpuUsage。
* 调用获取系统历史CPU使用率函数LOS\_HistorySysCpuUsage。
* 调用获取指定任务CPU使用率函数LOS\_TaskCpuUsage。
* 若任务已创建，则关中断，正常获取，恢复中断；
* 若任务未创建，则返回错误码；
* 调用获取指定任务历史CPU使用率函数LOS\_HistoryTaskCpuUsage。
* 若任务已创建，则关中断，根据不同模式正常获取，恢复中断；
* 若任务未创建，则返回错误码；
* 调用获取所有任务CPU使用率函数LOS\_AllCpuUsage。
* 若CPUP已初始化，则关中断，根据不同模式正常获取，恢复中断；
* 若CPUP未初始化或有非法入参，则返回错误码；

### 动态加载

#### 基本概念

在硬件资源有限的小设备中，需要通过算法的动态部署能力来解决无法同时部署多种算法的问题。以开发者易用为主要考虑因素，同时考虑到多平台的通用性，LiteOS-M选择业界标准的ELF方案，方便拓展算法生态。LiteOS-M提供类似于dlopen、dlsym等接口，APP通过动态加载模块提供的接口可以加载、卸载相应算法库。如图1所示，APP需要通过三方算法库所需接口获取对应信息输出，三方算法库又依赖内核提供的基本接口，如malloc等。APP加载所需接口，并对相关的未定义符号完成重定位后，APP即可调用该接口完成功能调用。目前动态加载组件只支持arm架构。

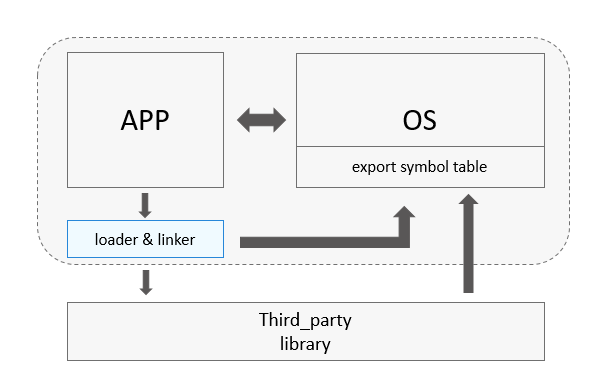


图1-1 LiteOS-M内核动态加载架构图

运行机制

1. 符号表导出

共享库调用内核接口需要内核主动暴露动态库所需的接口，如图2所示，该机制将符号信息编译到指定段中，调用SYM\_EXPORT宏即可完成对指定符号的信息导出。符号信息通过结构体SymInfo描述，成员包括符号名和符号地址信息，宏SYM\_EXPORT通过\_\_attribute\_\_编译属性将符号信息导入.sym.\*段中。

typedef struct {

CHAR \*name;

UINTPTR addr;

} SymInfo;

#define SYM\_EXPORT(func) \

const SymInfo sym\_##func \_\_attribute\_\_((section(".sym."#func))) = { \

.name = #func, \

.addr = (UINTPTR)func \

};

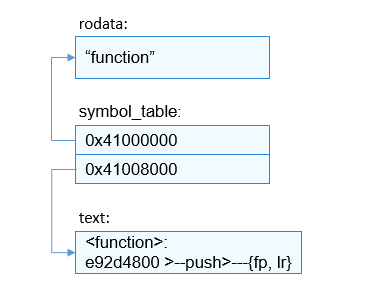


图 1-2 导出的符号表信息

1. ELF文件加载

加载过程中，根据ELF文件的句柄以及程序头表的段偏移可以得到需要加载到内存的LOAD段，一般有两个段，只读段及读写段，如下所示，可以用readelf -l查看ELF文件的LOAD段信息。如图3所示，根据相应的对齐属性申请物理内存，通过每个段的加载基址及偏移将代码段或数据段写入内存中。  
$ readelf -l lib.so

Elf file type is DYN (Shared object file)

Entry point 0x5b4

There are 4 program headers, starting at offset 52

Program Headers:

Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align

EXIDX 0x000760 0x00000760 0x00000760 0x00008 0x00008 R 0x4

LOAD 0x000000 0x00000000 0x00000000 0x0076c 0x0076c R E 0x10000

LOAD 0x00076c 0x0001076c 0x0001076c 0x0010c 0x00128 RW 0x10000

DYNAMIC 0x000774 0x00010774 0x00010774 0x000c8 0x000c8 RW 0x4

Section to Segment mapping:

Segment Sections...

00 .ARM.exidx

01 .hash .dynsym .dynstr .rel.dyn .rel.plt .init .plt .text .fini .ARM.exidx .eh\_frame

02 .init\_array .fini\_array .dynamic .got .data .bss

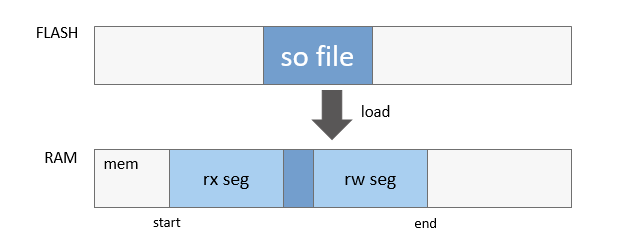
03 .dynamic

图1-3 ELF文件的加载过程

1. ELF文件链接

如图1-4所示，通过ELF文件的.dynamic段获取重定位表，遍历表中每一个需要重定位的条目，再根据需要重定位的符号名在共享库和内核提供的导出符号表中查找相应符号并更新相应的重定位信息。

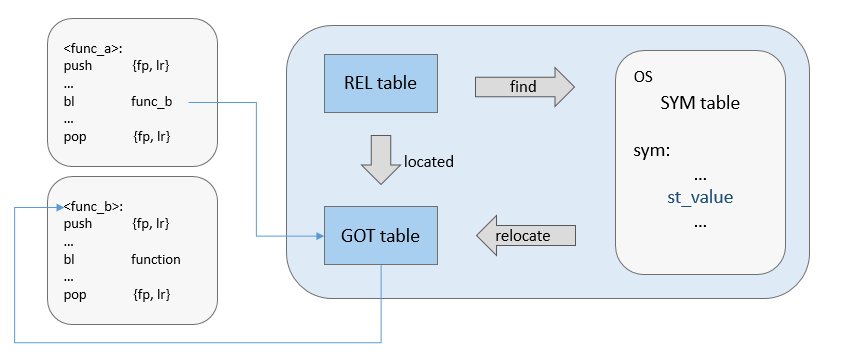


图1-4 ELF文件链接过程

#### 开发指导

1. 接口说明

表1-4 功能列表

|  |  |  |
| --- | --- | --- |
| 功能分类 | 接口名 | 描述 |
| 动态加载功能接口 | LOS\_DynlinkInit | 初始化动态链接器链表以及互斥锁 |
| LOS\_SoLoad | 加载指定路径的共享库 |
| LOS\_FindSym | 根据共享库句柄查找指定符号 |
| LOS\_SoUnload | 卸载共享库句柄 |

1. 开发流程

* 利用arm-none-eabi-gcc交叉编译器编译共享库并制作FAT或LittleFS文件系统格式镜像烧写至flash中；
* 在target\_config.h文件中定义宏LOSCFG\_DYNLINK为1使能动态加载模块；
* 调用LOS\_SoLoad接口加载指定路径下的共享库；
* 调用LOS\_FindSym接口查找指定符号，获取符号地址；
* 调用LOS\_SoUnload卸载指定共享库句柄。

 说明：

1. 利用交叉编译器编译共享库所需要的编译选项参考ELF支持规格一节。
2. 制作文件系统镜像之前需要对特定单板适配FAT或LittleFS文件系统。
3. 共享库不依赖编译器中的libc库，不支持c++。
4. 共享库只能依赖内核提供的接口，不能依赖其他共享库。

### 文件系统

当前支持的文件系统有FATFS与LittleFS，支持的功能如下表所示：

表 1-5功能列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 功能分类 | 接口名 | 描述 | FATFS | LITTELFS |
| 文件操作 | open | 打开文件 | 支持 | 支持 |
| close | 关闭文件 | 支持 | 支持 |
| read | 读取文件内容 | 支持 | 支持 |
| write | 往文件写入内容 | 支持 | 支持 |
| lseek | 设置文件偏移位置 | 支持 | 支持 |
| unlink | 删除文件 | 支持 | 支持 |
| rename | 重命名文件 | 支持 | 支持 |
| fstat | 通过文件句柄获取文件信息 | 支持 | 支持 |
| stat | 通过文件路径名获取文件信息 | 支持 | 支持 |
| fsync | 文件内容刷入存储设备 | 支持 | 支持 |
| 目录操作 | mkdir | 创建目录 | 支持 | 支持 |
| opendir | 打开目录 | 支持 | 支持 |
| readdir | 读取目录项内容 | 支持 | 支持 |
| closedir | 关闭目录 | 支持 | 支持 |
| rmdir | 删除目录 | 支持 | 支持 |
| 分区操作 | mount | 分区挂载 | 支持 | 支持 |
| umount | 分区卸载 | 支持 | 支持 |
| umount2 | 分区卸载，可通过MNT\_FORCE参数进行强制卸载 | 支持 | 不支持 |
| statfs | 获取分区信息 | 支持 | 不支持 |

#### FAT

1. 基本概念

FAT文件系统是File Allocation Table（文件配置表）的简称，主要包括DBR区、FAT区、DATA区三个区域。其中，FAT区各个表项记录存储设备中对应簇的信息，包括簇是否被使用、文件下一个簇的编号、是否文件结尾等。FAT文件系统有FAT12、FAT16、FAT32等多种格式，其中，12、16、32表示对应格式中FAT表项的字节数。FAT文件系统支持多种介质，特别在可移动存储介质（U盘、SD卡、移动硬盘等）上广泛使用，使嵌入式设备和Windows、Linux等桌面系统保持很好的兼容性，方便用户管理操作文件。

OpenHarmony内核支持FAT12、FAT16与FAT32三种格式的FAT文件系统，具有代码量小、资源占用小、可裁切、支持多种物理介质等特性，并且与Windows、Linux等系统保持兼容，支持多设备、多分区识别等功能。OpenHarmony内核支持硬盘多分区，可以在主分区以及逻辑分区上创建FAT文件系统。

1. 开发指导

FAT文件系统的使用需要底层MMC相关驱动的支持。在一个带MMC存储设备的板子上运行FATFS，需要：

（1）适配板端EMMC驱动，实现disk\_status、disk\_initialize、disk\_read、disk\_write、disk\_ioctl接口；

（2）新增fs\_config.h文件，配置FS\_MAX\_SS（存储设备最大sector大小）、FF\_VOLUME\_STRS（分区名）等信息，例如：

#define FF\_VOLUME\_STRS "system", "inner", "update", "user"

#define FS\_MAX\_SS 512

#define FAT\_MAX\_OPEN\_FILES 50

1. 开发流程

 说明：

* FATFS文件与目录操作：
* 单个文件大小不超过4G。
* 支持同时打开的文件数最大为FAT\_MAX\_OPEN\_FILES，文件夹数最大为FAT\_MAX\_OPEN\_DIRS。
* 暂不支持根目录管理，文件/目录名均以分区名开头，例如“user/testfile”就是在“user”分区下名为“testfile”的文件或目录。
* 若需要同时多次打开同一文件，必须全部使用只读方式（O\_RDONLY）。以可写方式（O\_RDWR、O\_WRONLY等）只能打开一次。
* 读写指针未分离，例如以O\_APPEND（追加写）方式打开文件后，读指针也在文件尾，从头读文件前需要用户手动置位。
* 暂不支持文件与目录的权限管理。
* stat及fstat接口暂不支持查询修改时间、创建时间和最后访问时间。微软FAT协议不支持1980年以前的时间。
* FATFS分区挂载与卸载：
* 支持以只读属性挂载分区。当mount函数的入参为MS\_RDONLY时，所有的带有写入的接口，如write、mkdir、unlink，以及非O\_RDONLY属性的open，将均被拒绝。
* mount支持通过MS\_REMOUNT标记修改已挂载分区的权限。
* 在umount操作前，需确保所有目录及文件全部关闭。
* umount2支持通过MNT\_FORCE参数强制关闭所有文件与文件夹并umount，但可能造成数据丢失，请谨慎使用。
* FATFS支持重新划分存储设备分区、格式化分区，对应接口为fatfs\_fdisk与fatfs\_format：
* 在fatfs\_format操作之前，若需要格式化的分区已挂载，需确保分区中的所有目录及文件全部关闭，并且分区umount。
* 在fatfs\_fdisk操作前，需要该设备中的所有分区均已umount。
* fatfs\_fdisk与fatfs\_format会造成设备数据丢失，请谨慎使用。

#### LittleFS

1. 基本概念

LittleFS是一个小型的的Flash文件系统，它结合日志结构（log-structured）文件系统和COW（copy-on-write）文件系统的思想，以日志结构存储元数据，以COW结构存储数据。这种特殊的存储方式，使LittleFS具有强大的掉电恢复能力（power-loss resilience)。分配COW数据块时LittleFS采用了名为统计损耗均衡的动态损耗均衡算法，使Flash设备的寿命得到有效保障。同时LittleFS针对资源紧缺的小型设备进行设计，具有极其有限的ROM和RAM占用，并且所有RAM的使用都通过一个可配置的固定大小缓冲区进行分配，不会随文件系统的扩大占据更多的系统资源。

当在一个资源非常紧缺的小型设备上，寻找一个具有掉电恢复能力并支持损耗均衡的Flash文件系统时，LittleFS是一个比较好的选择。

1. 开发指导

移植LittleFS到新硬件设备上，需要申明lfs\_config：

const struct lfs\_config cfg = {

// block device operations

.read = user\_provided\_block\_device\_read,

.prog = user\_provided\_block\_device\_prog,

.erase = user\_provided\_block\_device\_erase,

.sync = user\_provided\_block\_device\_sync,

// block device configuration

.read\_size = 16,

.prog\_size = 16,

.block\_size = 4096,

.block\_count = 128,

.cache\_size = 16,

.lookahead\_size = 16,

.block\_cycles = 500,

};

其中.read，.prog，.erase，.sync分别对应该硬件平台上的底层的读写\擦除\同步等接口。

read\_size 每次读取的字节数，可以比物理读单元大以改善性能，这个数值决定了读缓存的大小，但值太大会带来更多的内存消耗。

prog\_size 每次写入的字节数，可以比物理写单元大以改善性能，这个数值决定了写缓存的大小，必须是read\_size的整数倍，但值太大会带来更多的内存消耗。

block\_size 每个擦除块的字节数，可以比物理擦除单元大，但此数值应尽可能小因为每个文件至少会占用一个块。必须是prog\_size的整数倍。

block\_count 可以被擦除的块数量，这取决于块设备的容量及擦除块的大小。

## 外设

### WLAN

#### WLAN开发概述

1. 简介

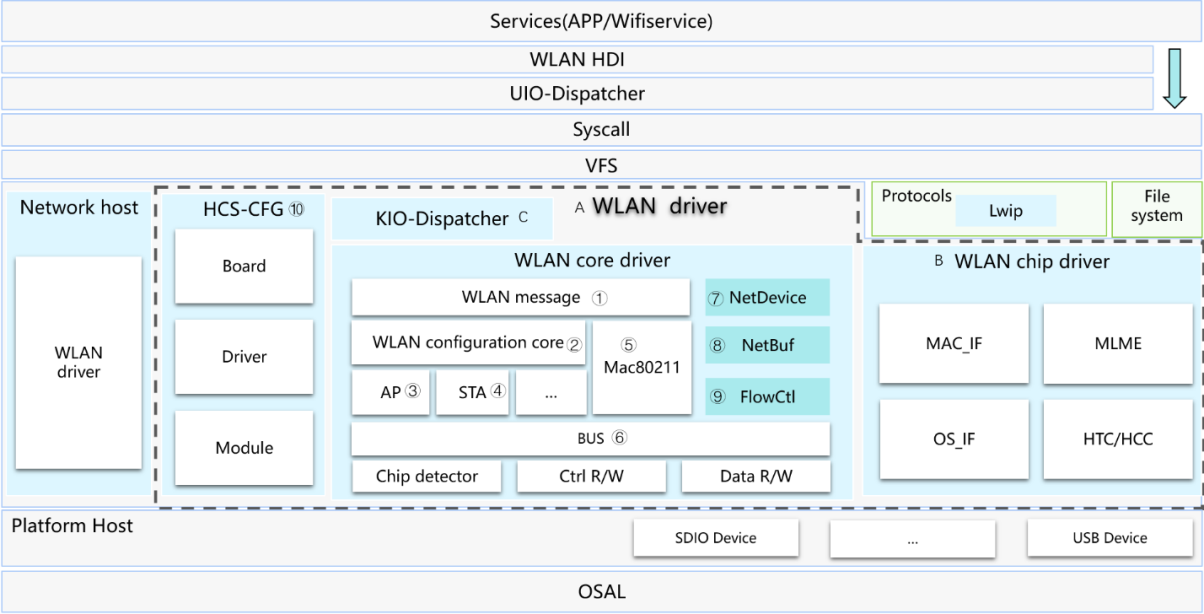
WLAN是基于统一驱动平台（以下简称HDF）开发的模块，该模块可实现跨操作系统迁移，自适应器件差异，模块化拼装编译等功能。各WLAN厂商驱动开发人员可根据WLAN模块提供的向下统一接口适配各自的驱动代码，HAL层开发人员可根据WLAN模块提供的向上统一接口获取如下能力：建立/关闭WLAN热点，扫描，关联WLAN热点等等。WLAN模块框架图如下：

图1-5 WLAN框架

（1）WLAN驱动接口架构

WLAN模块有三部分对外开放的API接口，如下图所示：

1）对HAL层提供的能力接口；

2）驱动直接调用WLAN模块能力接口；

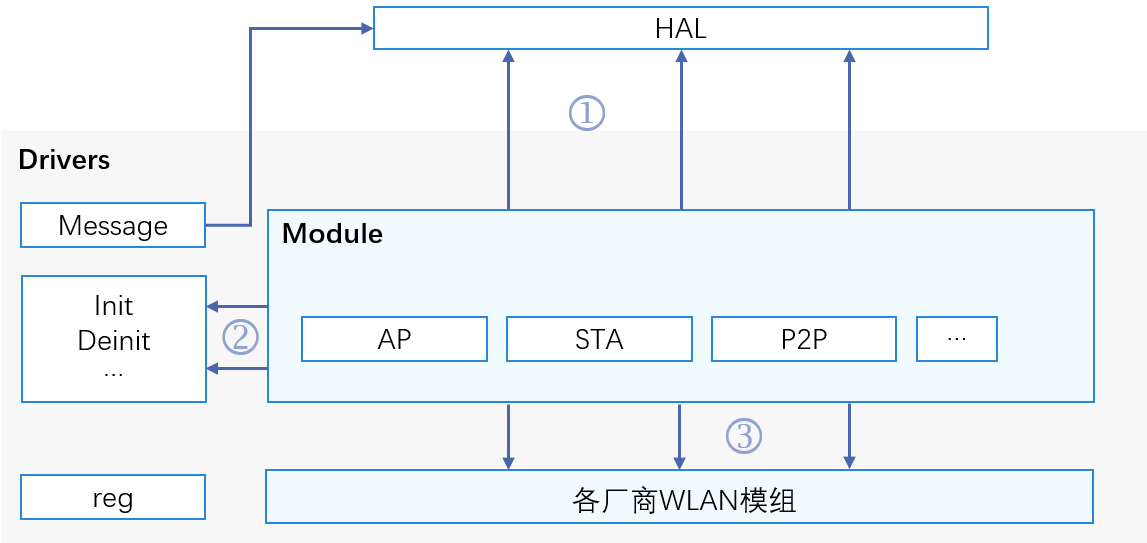
3）提供给各厂商实现的能力接口，本章节以下内容将从以初始化为例进行开发指导。

图1-6 WLAN模块开放能力分布图

1. 接口说明

WLAN驱动模块对HAL层开发人员提供的接口功能有：建立/关闭WLAN热点、扫描WLAN、开始关联、断开连接等等。

WLAN驱动模块提供了驱动开发人员可直接调用的能力接口，主要功能有：创建/释放WifiModule、关联/取消关联、申请/释放net buf、lwip的pbuf和netbuf的相互转换等等。

表1 提供了部分接口说明如下：

表 1-6 可直接调用的接口

|  |  |  |
| --- | --- | --- |
| **头文件** | **接口名称** | **功能描述** |
| wifi\_module.h | struct WifiModule *WifiModuleCreate(const struct HdfConfigWifiModuleConfig*config); | 基于HDF开发WLAN驱动时，创建一个WifiModule。 |
| void WifiModuleDelete(struct WifiModule *module);* | 基于HDF开发WLAN驱动时，删除并释放WifiModule所有数据。 |
| int32\_t DelFeature(struct WifiModule module, uint16\_t featureType); | 基于HDF开发WLAN驱动时，从WifiModule删除一个功能组件。 |
| int32\_t AddFeature(struct WifiModule *module, uint16\_t featureType, struct WifiFeature*featureData); | 基于HDF开发WLAN驱动时，注册一个功能组件到WifiModule。 |
| wifi\_mac80211\_ops.h | int32\_t (*startAp)(NetDevice*netDev); | 启动AP。 |
| int32\_t (*stopAp)(NetDevice*netDev); | 停止AP。 |
| int32\_t (*connect)(NetDevice*netDev, WifiConnectParams *param);* | 开始连接。 |
| int32\_t (disconnect)(NetDevice *netDev, uint16\_t reasonCode);* | 断开连接。 |
| hdf\_netbuf.h | static inline void NetBufQueueInit(struct NetBufQueue q); | 初始化net buffer队列。 |
| struct NetBuf *NetBufAlloc(uint32\_t size);* | 申请net buffer。 |
| void NetBufFree(struct NetBuf nb); | 释放net buffer。 |
| struct NetBuf *Pbuf2NetBuf(const struct NetDevice*netdev, struct pbuf *lwipBuf);* | lwip的pbuf转换为net buffer。 |
| struct pbuf NetBuf2Pbuf(const struct NetBuf \*nb); | net buffer转换为lwip的pbuf。 |

WLAN驱动模块提供了需驱动开发人员实现的能力接口，主要功能有：初始化/注销net device、打开/关闭net device、获取net device的状态等等。

表2 提供了部分接口说明如下：

表 1-7 需要开发人员实现的接口

|  |  |  |
| --- | --- | --- |
| **头文件** | **接口名称** | **功能描述** |
| net\_device.h | int32\_t (*init)(struct NetDevice*netDev); | 初始化net device。 |
| struct NetDevStats *(*getStats)(struct NetDevice *netDev);* | 获取net device的状态。 |
| int32\_t (setMacAddr)(struct NetDevice *netDev, void*addr); | 设置mac地址。 |
| void (*deInit)(struct NetDevice*netDev); | 注销net device。 |
| int32\_t (*open)(struct NetDevice*netDev); | 打开net device。 |
| int32\_t (*stop)(struct NetDevice*netDev); | 关闭net device。 |

### Touchscreen

#### Touchscreen开发概述

1. 简介

* Touchscreen驱动主要任务

Touchscreen驱动用于驱动触摸屏使其正常工作，该驱动主要完成如下工作：对Touchscreen Driver IC进行上电、配置硬件管脚并初始化其状态、注册中断、配置通信接口（I2C或SPI）、设定input相关配置、下载及更新固件等，从而保证Touchscreen Driver IC驱动触摸屏正常工作。

* Touchscreen驱动层次说明

本节主要介绍基于HDF驱动框架开发Touchscreen器件驱动，其整体的框架模型如下。

Touchscreen驱动向下基于HDF驱动框架、PLATFORM接口、OSAL接口开发，向上对接规范化的驱动接口HDI（OpenHarmony Driver Interface）层，通过HDI层对外提供硬件能力，即上层Service可以通过HDI接口层获取相应的驱动能力，进而操作控制Touchscreen器件。

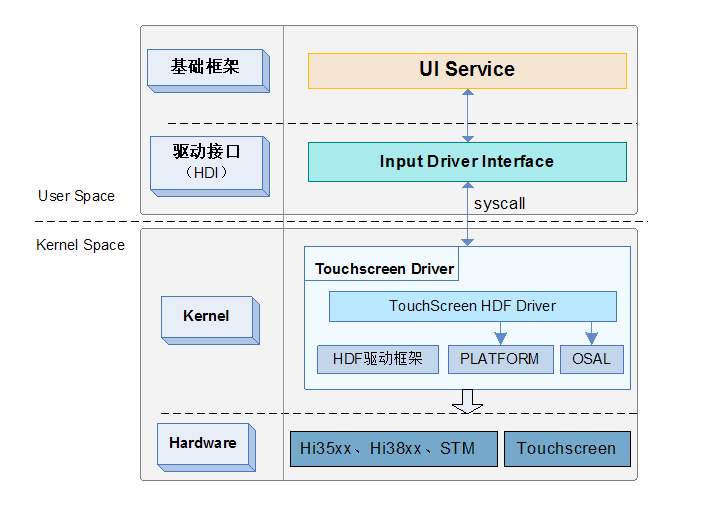


图 1-7 Touchscreen驱动软件层次

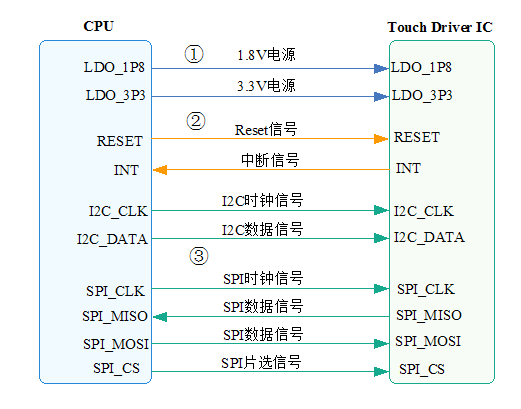
* 基于HDF驱动框架开发器件驱动的优势

在HDF（OpenHarmony Driver Foundation）[驱动管理框架](https://www.bookstack.cn/read/openharmony-1.0-zh-cn/driver-%E9%A9%B1%E5%8A%A8%E5%BC%80%E5%8F%91.md)的基础上，器件驱动调用OSAL和PLATFORM基础接口实现开发，包括bus通信接口、操作系统原生接口（memory、lock、thread、timer等）。由于OSAL和PLATFORM接口屏蔽了芯片平台差异，所以Touchscreen HDF Driver可以进行跨平台、跨OS执行，以便逐步实现驱动的一次开发，多端部署。

1. 接口说明

Touchscreen器件的硬件接口相对简单，根据PIN脚的属性，可以简单分为三类：

* 电源接口
* IO控制接口
* 通信接口

图 1-8 Touchscreen器件常用管脚

如上图所示的三类接口，分别做简要说明如下：

（1）电源接口

* + LDO\_1P8：1.8v数字电路
  + LDO\_3P3：3.3v模拟电路

通常情况下，Touchscreen Driver IC和LCD的Driver IC是相互分离的，这种情况下，Touchscreen Driver IC一般同时需要1.8v和3.3v两路供电。随着芯片演进，行业内已有Touchscreen Driver IC和LCD Driver IC 集成在一颗IC中的芯片案例，对Touchscreen而言，只需要关注1.8v供电即可，其内部需要的3.3v电源，会在Driver Chip IC内部从LCD的VSP电源（典型值5.5V）中分出来。

（2）IO控制接口

* + Reset：reset管脚，用于在系统休眠、唤醒时，由主机侧对Driver IC进行复位操作。
  + INT：中断管脚，需要在驱动初始化时，配置为输入上拉状态。在Driver IC检测到外部触摸信号后，通过操作中断管脚来通知Driver，Driver一般会在中断处理函数中，进行报点数据读取等操作。

（3）通信接口

* + I2C：由于Touchscreen的报点数据量相对较少，所以一般选用I2C方式传输数据。I2C的具体协议及对应操作接口，可以参考PLATFORM的[“I2C”使用指南](https://www.bookstack.cn/read/openharmony-1.0-zh-cn/driver-I2C%E4%BD%BF%E7%94%A8%E6%8C%87%E5%AF%BC.md)。
  + SPI：部分厂商，由于需要传递的数据不止报点坐标，而是需要获取基础容值，数据量较大，所以会选用SPI通信方式。SPI的具体协议及对应操作接口，可以参考PLATFORM的[“SPI” 使用指南](https://www.bookstack.cn/read/openharmony-1.0-zh-cn/driver-SPI%E4%BD%BF%E7%94%A8%E6%8C%87%E5%AF%BC.md)。