OpenHarmony

开放原子开源基金会 OpenHarmony开发者大会 2023

HDF使能设备驱动开发更高效容易

赵文华

华为终端OS平台资深专家

OpenHarmony

目录 Contents

OpenHarmony

目录 Contents

HDF（Hardware Driver Foundation）驱动框架

设计目标

提供与芯片平台、内核解耦的驱动开发/运行时环境并为系统提供标准化硬件抽象接口，实现驱动软件在不同设备中部署，简化硬件接入开发步骤。

弹性化框架

通过对象管理器，通过多态加载不同容量设备实现方式，实现弹性伸缩部署

组件化设备模型

提供设备功能模型抽象，可动态拆解，满足不同容量设备部署

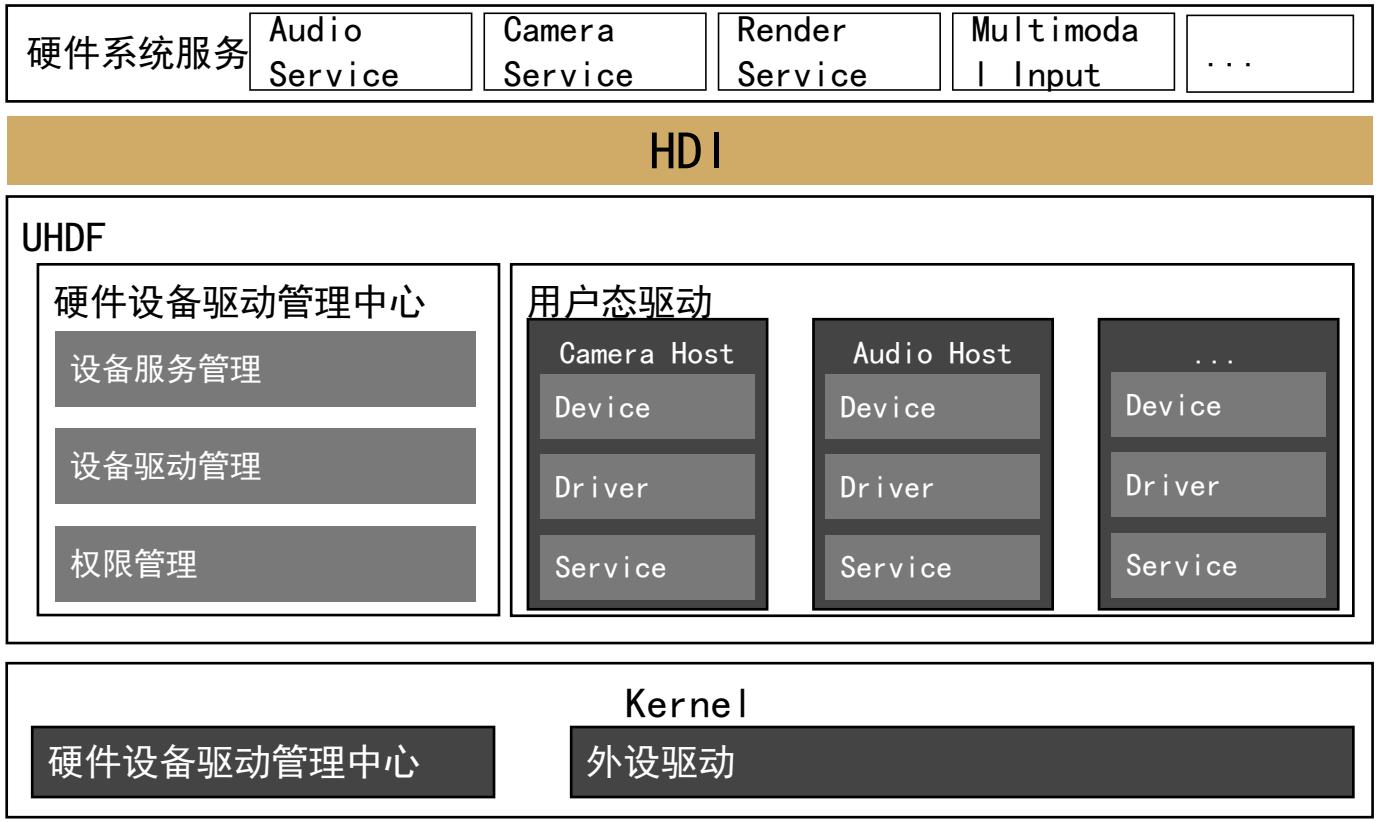
归一化平台底座

提供规范化的内核、硬件IO适配接口，屏蔽不同内核，芯片平台实现差异

统一配置界面

HCS驱动配置解决方案面向不同容量设备，提供统一的配置界面

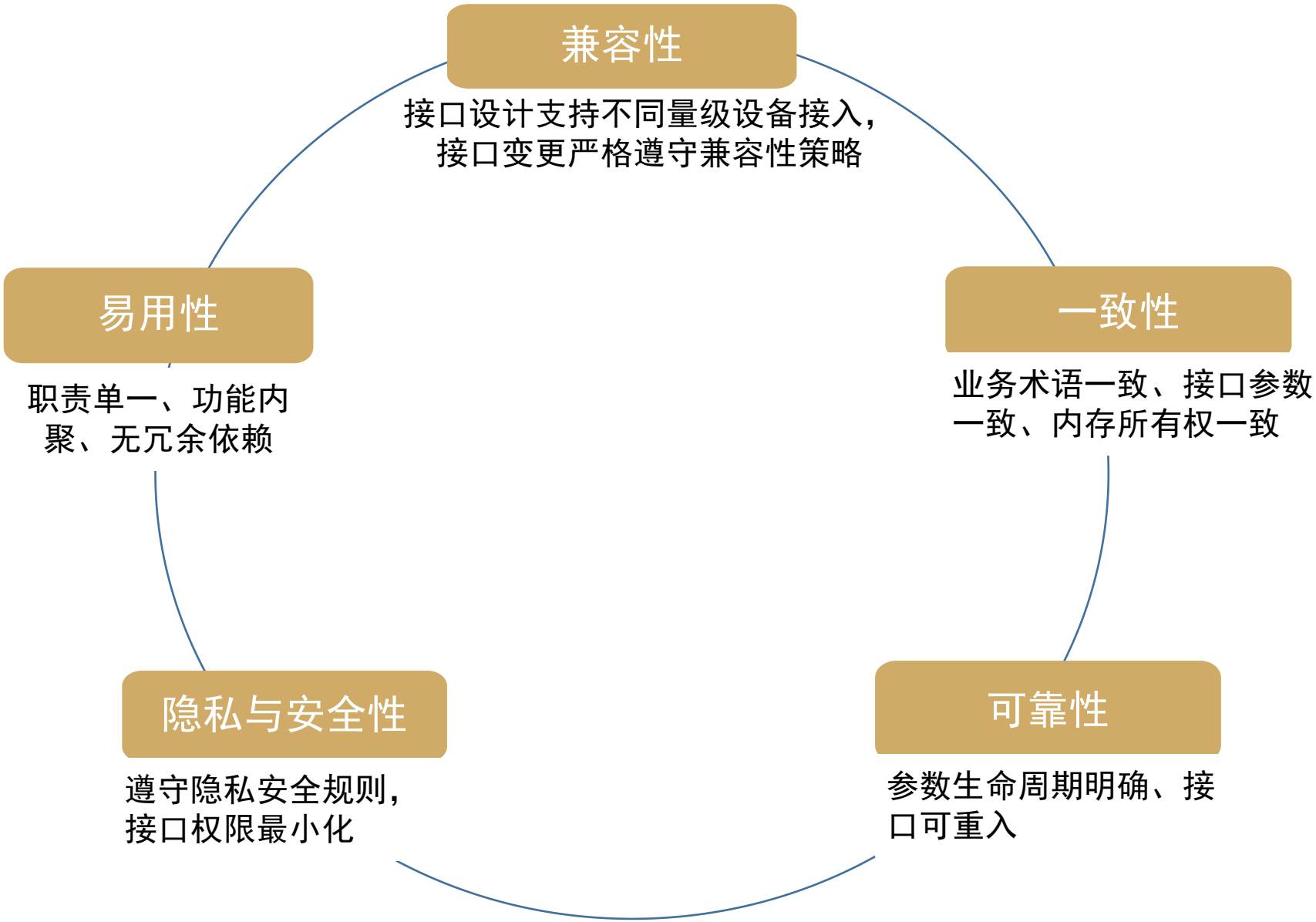
什么是HDI接口



HDI(Hardware Device Interface)
即硬件设备接口，为系统提供统一、标准的硬件设备抽象，实现系统服务与硬件解耦。

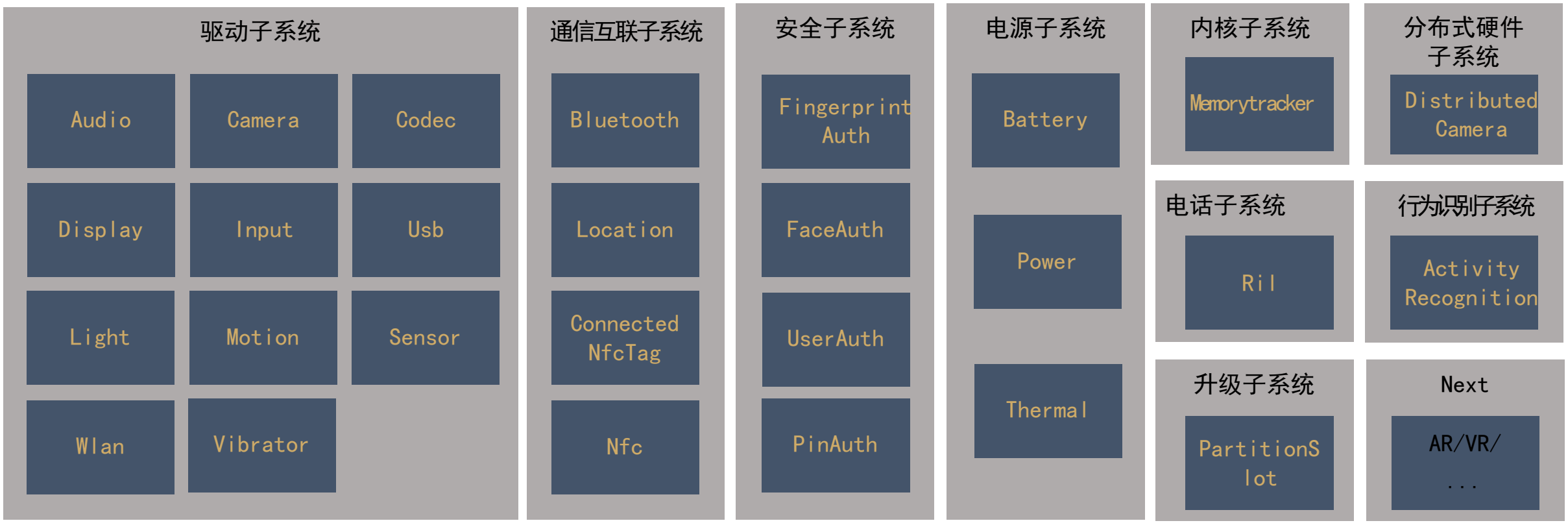
- 标准化硬件接口
- 基于接口描述的语言无关化接口定义(IDL)
- 支持C/C++客户端服务端
- 支持IPC模式与直通模式兼容
- 支持Mini、Small、Standard多种系统部署
- 自动生成IPC过程代码，开发效率大幅提升

HDI接口设计原则


兼容性
接口设计支持不同量级设备接入，
接口变更严格遵守兼容性策略
一致性
业务术语一致、接口参数
一致、内存所有权一致
可靠性
参数生命周期明确、接
口可重入
隐私与安全性
遵守隐私安全规则，
接口权限最小化
易用性
职责单一、功能内
聚、无冗余依赖

HDI接口支持现状

- **涵盖27个模块，800+个HDI接口**



通过提供丰富的标准化设备接口，让硬件适配更加便捷，使能更多硬件设备快速接入。

HDI接口开发旅程

接口定义 → 编译接口 → 实现接口 → 编译实现 → 声明服务 → 执行与调试

IFoo.idl

```
package
ohos.hdi.foo.v1_0;

interface IFoo {
    Bar([in] int val);
}
```

BUILD.gn

```
hdi("battery") {
   module_name =
"foo_interface_service"
   sources = [
     "IFoo.idl"
   ]
   language = "cpp"
   subsystem_name = "hdf"
   part_name =
"drivers_interface_battery"
 }
```

libfoo_proxy_1.0.z.so

libfoo_stub_1.0.z.so

foo_driver.cpp
foo_service.cpp

```
class FooService :
public IFoo {
public:
    FooService();
    virtual
~FooService();
    Bar(int val);
};
```

BUILD.gn

```
ohos_shared_library
(   "libfoo_service_1.0")
{
  sources =
[ "foo_service.cpp" ]
  public_deps =
[ "//drivers/interface/foo/
v1_0:foo_idl_headers" ]
  subsystem_name = "hdf"
  part_name =
"drivers_peripheral_foo"
}
```

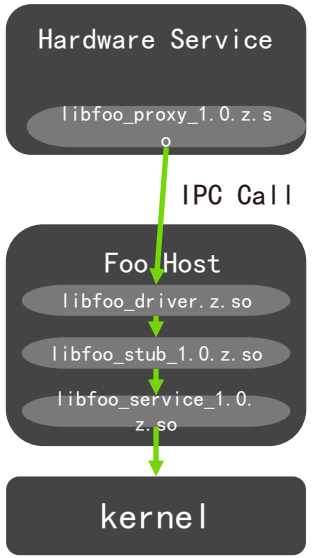
libfoo_driver.z.so

libfoo_service_1.0.z.so

udhf_config

```
device0 :: deviceNode {
    policy = 2;
    priority = 100;
    preload = 2;
    moduleName =

"libfoo_driver.z.so";
    serviceName =
          "foo_service";
 }
```



HDI接口开发旅程-接口定义

编写IDL文件

IFoo.idl

```
package ohos.hdi.foo.v1_0;

import ohos.hdi.foo.v1_0.IFooCallback;
import ohos.hdi.foo.v1_0.Types;

interface IFoo {
    Ping([in] String sendMsg,
         [out] String recvMsg);
    GetData([out] struct FooInfo info);
    SetCallback([in] IFooCallback cb);
}
```

Types.idl

```
package ohos.hdi.foo.v1_0;

enum FooType {
    FOO_TYPE_ONE = 1,
    FOO_TYPE_TWO = 2,
};
struct FooInfo {
    unsigned int id;
    String name;
    enum FooType type;
};
```

IFooCallback.idl

```
package ohos.hdi.foo.v1_0;

[callback] interface IFooCallback {
    PushData([in] String message);
}
```

也可以通过idl-gen工具
将头文件转换为IDL文件

ifoo.h

```
#include <string>

#include "v1_0/ifoo_callback.h"

class IFoo {
public:
    virtual int Ping(std::string sendMsg,
                     std::string &recvMsg) = 0;

    virtual int GetData(struct FooInfo &info) = 0;

    virtual int SetCallback(IFooCallback cbObj) = 0;
};
```

HDI设计规范：https://gitee.com/openharmony/docs/blob/master/zh-cn/design/hdi-design-specifications.md

HDI接口开发旅程-接口编译

BUILD.gn

```
import(“//drivers/hdf_core/adapter/uhdf2/hdi.gni”)    # 导入idl编译模板
hdi(“foo”) {                                          # 目标名称，生成so规则:
libfoo_[proxy/stub]_1.0.z.so
    module_name = "foo"
    sources = [                                       # 参与编译的idl文件
        "IFoo.idl",                                   # 接口idl
        "IFooCallback.idl",                           # 用于回调的idl
        "Types.idl",                                  # 自定义类型idl
    ]
    language = "cpp"                                  # 控制idl生成c或c++代码 可选择`c`或`cpp`
    subsystem_name = “hdf”                            # 子系统名称
    part_name = "drivers_interface_foo"               # 部件名
}
```

idl描述生成代码目录结构

```
out/xxx/gen/drivers/interface/foo/v1_0
├──── ifoo_callback.h
├──── ifoo_interfaces.h
├──── foo_driver.cpp
├──── foo_proxy.cpp
├──── foo_proxy.h
├──── foo_stub.cpp
├──── foo_stub.h
├──── foo_types.cpp
├──── foo_types.h
└──── ...
```

HDI接口开发旅程 -接口实现与服务声明

接口实现目录结构

```
drivers/peripheral/foo
├── BUILD.gn
├── bundle.json
└── hdi_service
    ├── BUILD.gn
    ├── include
    │   └── foo_service.h
    └── src
        ├── foo_driver.cpp
        └── foo_service.cpp
```

实现接口　　参考实现

idl描述生成代码

```
out/xxx/gen/drivers/interface/foo/v1_0
├── foo_driver.cpp
├── ifoo_callback.h
├── ifoo.h
├── └── foo_types.h
└── ...
```

foo_driver.cpp

```
static int32_t HdfBatteryInterfaceDriverBind(struct HdfDeviceObject *deviceObject)
{
    auto *hdfBatteryInterfaceHost = new (std::nothrow) HdfBatteryInterfaceHost;
    if (hdfBatteryInterfaceHost == nullptr) {
        BATTERY_HILOGE(COMP_HDI, "%{public}s: failed to create HdfBatteryInterfaceHost object", __func__);
        return HDF_FAILURE;
    }
```

foo_service.cpp

```
namespace OHOS {
namespace HDI {
namespace Foo {
namespace V1_0 {

int32_t FooService::Ping(const std::string &sendMsg, const std::s
{
    recvMsg = sendMsg;
    return HDF_SUCCESS;
}

int32_t FooService::GetData(const FooInfo &info)
{
    info.id = 1;
    info.name = "foo";
    info.type = FOO_TYPE_ONE;
    return HDF_SUCCESS;
}

sptr<IFooCallback> g_cbObj;
int32_t FooService::SetCallback(const sptr<IFooCallback> &cbObj)
{
    g_cbObj = cbObj;
    return HDF_SUCCESS;
}

// 非常重要，必须要实现FooService的实例化接口
extern "C" IFoo *FooImplGetInstance(void)
{
    // 也可以在这里做一些其他初始化工作
    return new (std::nothrow) FooService();
}
```

vendor/xxx/hdf_config/uhdf/device_info.hcs

```
foo_host :: host {
    hostName = "foo_host";
    priority = 50;
    foo_device :: device {
        device0 :: deviceNode {
            policy = 2;
            priority = 100;
            moduleName = "libfoo_driver.z.so";
            serviceName = "foo_service";
        }
    }
}
```

HDI接口开发旅程 - 接口调用

通过接口静态方法获取
对接口实例，调用相关功能接口

foo_test.cpp

```cpp
#include <v1_0/ifoo_interface.h>

void WorkFunc(void) {
    // 获取该服务客户端实例
    sptr<IFoo> foo = Hdi::Foo::V1_0::IFoo::Get();
    if (foo == nullptr) {
        // hdi service not exist, handle error
    }

    foo->Bar(); // do interface call
}
```

通过Interface部件依赖proxy库

Build.gn

```
ohos_shared_library("xxx") {
  sources = [
    ...
  ]

  external_deps = [
    "drivers_interface_battery:
     libbattery_proxy_1.1",
  ]
  part_name = ""
}
```

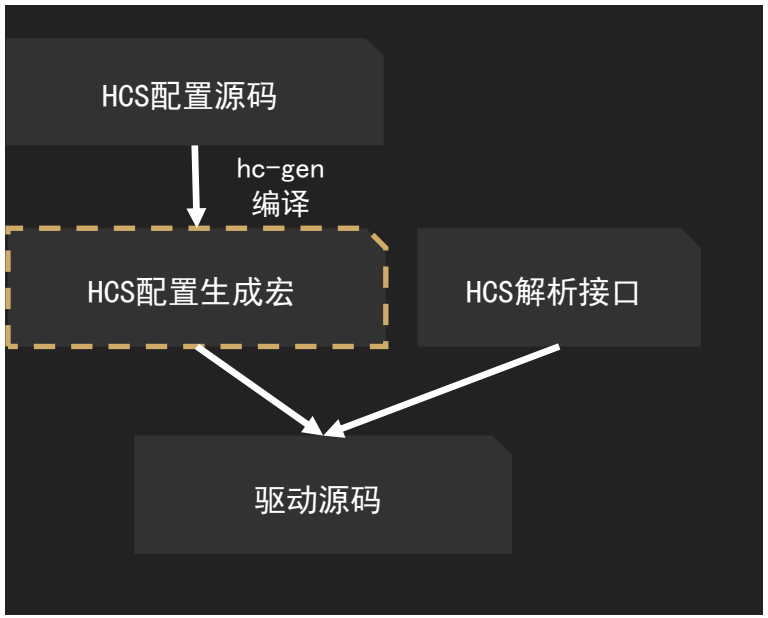
注意：为实现组件解耦和兼容性保证，接口调用者不可直接依赖服务实现库

OpenHarmony

目录 Contents

HCS宏式解析

为Mini系统提供极小资源占用的HCS配置管理方案



config.hcs

```
root {
    module = "test";
    foo {
        term1 = 1;
        term2 = 0x2;
        term3 = 0b11;
    }
}
```

hc-gen

config_gen.h

```
// hcs node macros: root
#define root_exists 1
#define root_nodeName "root"
#define root_foreach_child(func) \
    func(root_foo);

#define root_foreach_child_vargs(fu
    func(root_foo, __VA_ARGS__);

#define root_module_exists 1
#define root_module "test"

// hcs node macros: root/foo
#define root_foo_exists 1
#define root_foo_nodeName "foo"
#define root_foo_term1_exists 1
#define root_foo_term1 1
#define root_foo_term2_exists 1
#define root_foo_term2 2
#define root_foo_term3_exists 1
#define root_foo_term3 3

#endif // HCS_CONFIG_CASE_HEADER_H
```

include

driver.c

```
#include <hcs_macro.h>
#include "config_gen.h"

#define FOO_NODE HCS_NODE(HCS_ROOT, foo);
void ParseDriverConfig()
{
    int term1Val = HCS_PROP(FOO_NODE, term1);
    int term2Val = HCS_PROP(FOO_NODE, term2);
    int term3Val = HCS_PROP(FOO_NODE, term3);
    const char *moduleName = HCS_PROP(HCS_ROOT
}
```

极致性能：

- 资源占用相比二进制解析降低50%
- 解析在编译阶段完成，运行效率提升70%

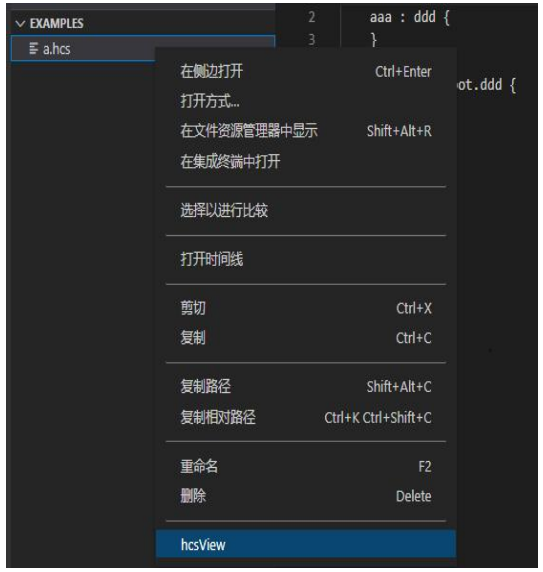
开发友好：

- 提供了适用于小型设备驱动软硬件解耦方案
- 功能完备，支持节点和属性的查询、遍历属、性存在性判断等操作

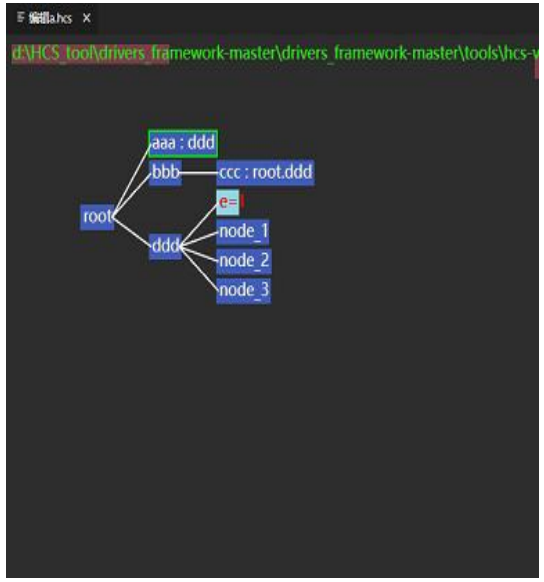
HCS可视化编辑器

安装

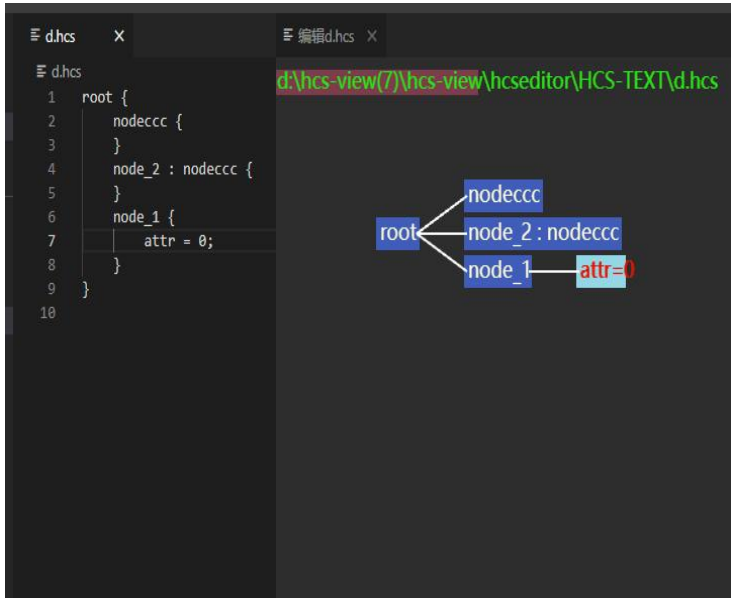
导入



编辑



导出

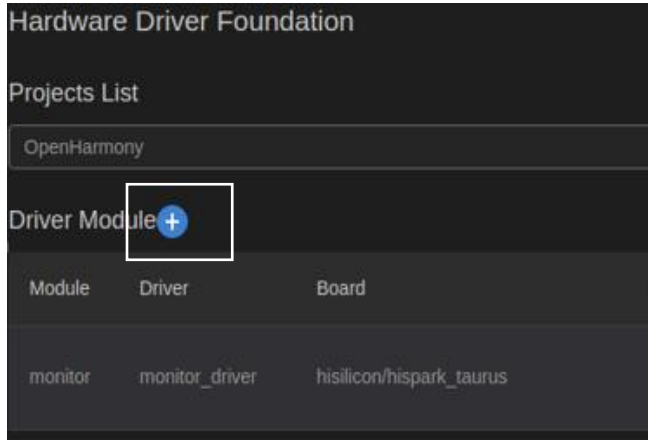


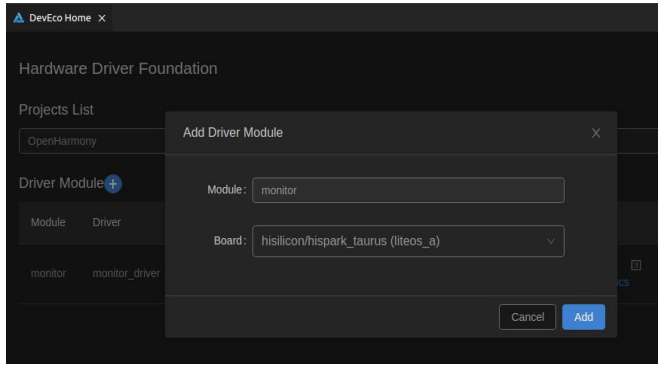
- GUI界面可视化编辑，显示直观，操作友好
- 错误实时提示，降低HCS配置方法学习成本
- 完整语法检查，HCS文件0错误

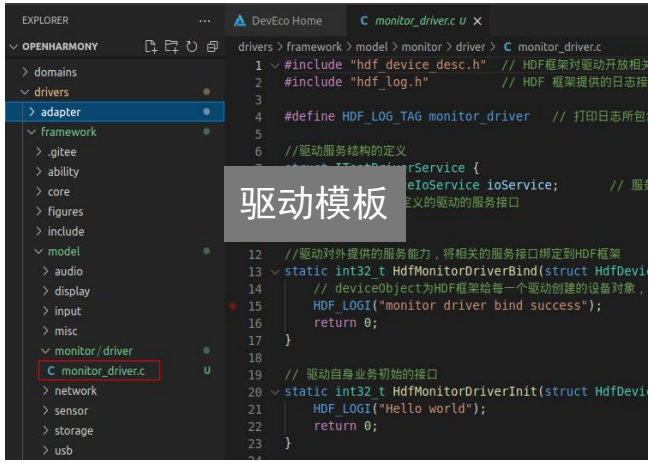
HDF设备驱动开发模板代码生成

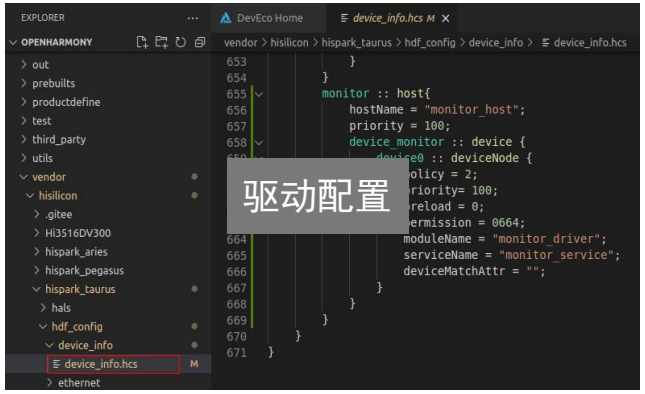
模板化生成工具：支持一键生成驱动开发所需的代码模板、驱动配置和编译配置

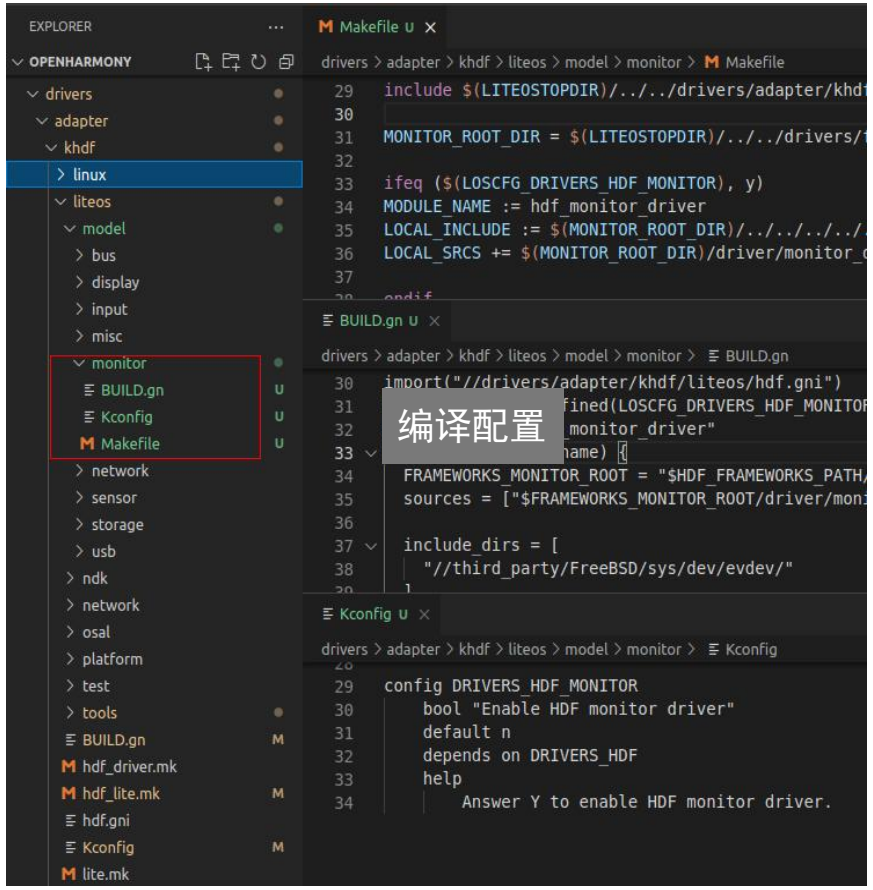
工具源码：https://gitee.com/openharmony/drivers_hdf_core/tree/master/framework/tools/hdf_dev_eco_tool







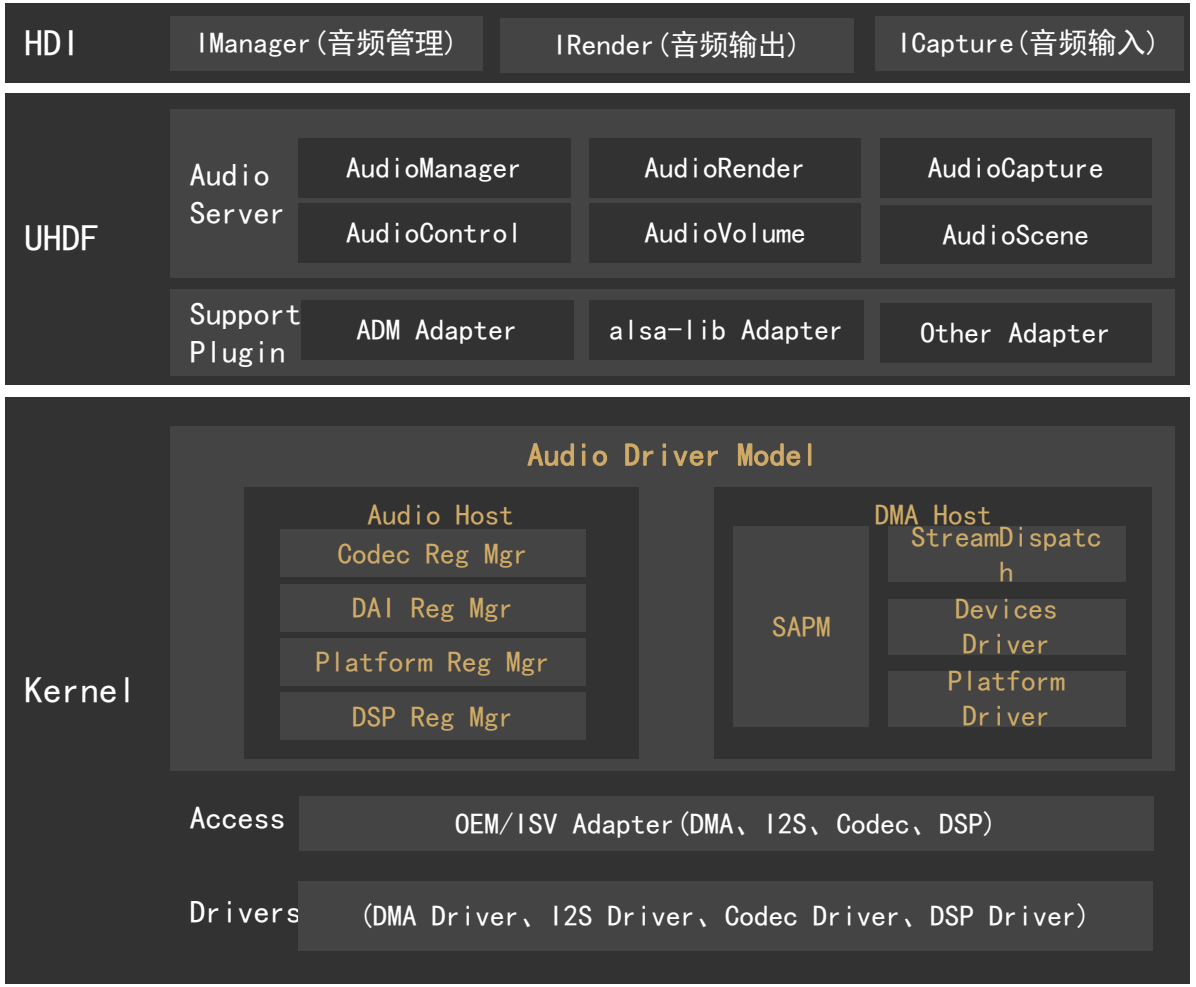




OpenHarmony

目录 Contents

新音频驱动模型ADM简介



ADM音频驱动模型(AudioDriverModel)关键能力特征 ：

灵活的轻量级架构

- 轻量级架构，资源占用少，同时支持小型和标准系统
- 灵活的可裁剪性，支持Liteos与Linux多内核支持

适配容易

- 提供HDF层通用实现，只需适配硬件驱动层即可让音频硬件快速接入系统

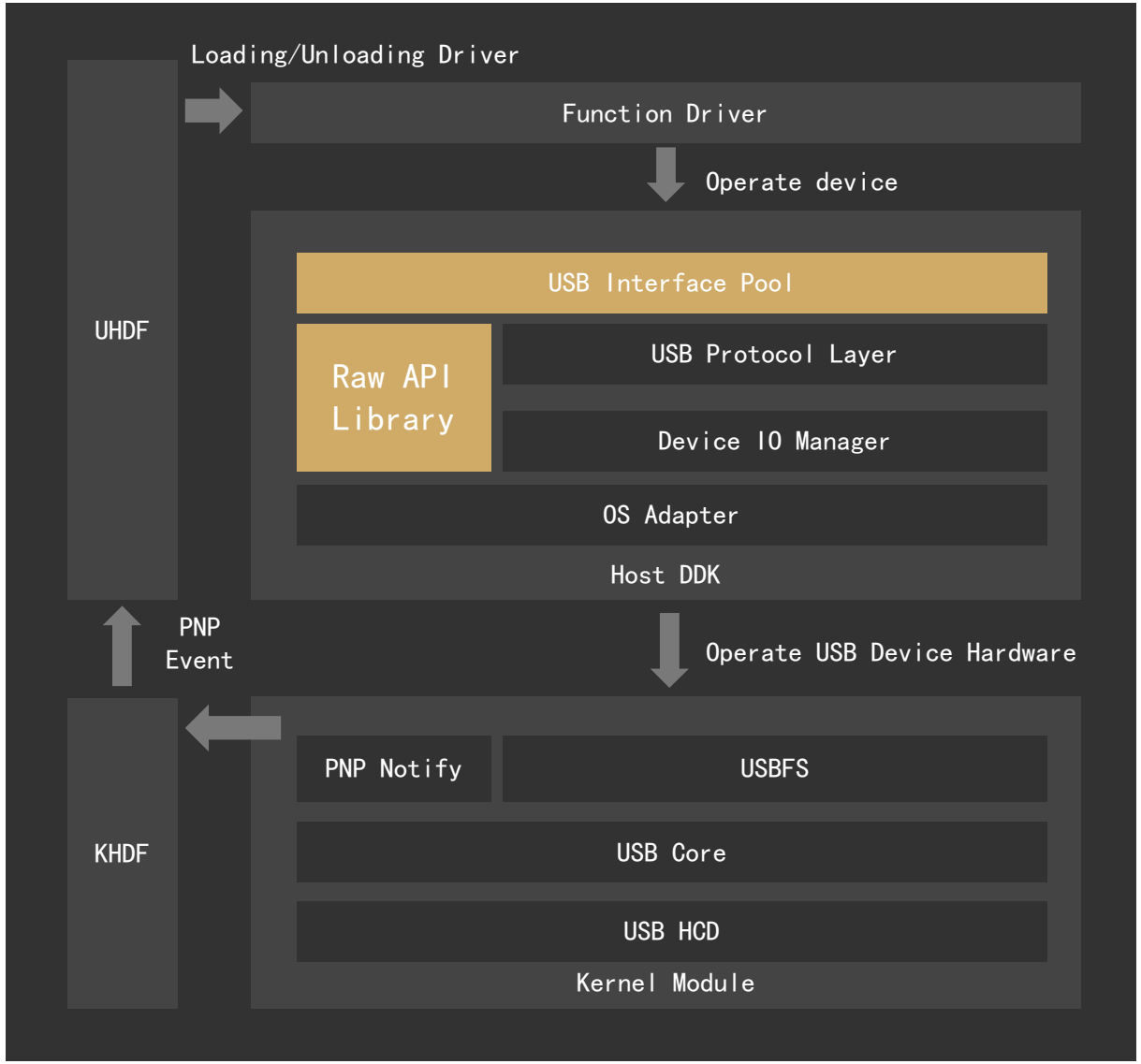
丰富的外设类型支持

- 支持audio codec、smartPA
- 支持speaker、mic、3.5mm耳机、USB type-c耳机

插件化动态音效框架

- 支持硬件音效与软件音效插件化集成

用户态USB驱动开发框架USB DDK简介



USB DDK(Driver Develpot Kit)核心特征：

支持用户态USB驱动开发

- 用户态USB DDK，支持便捷地开发安全稳定的用户态USB驱动

支持跨平台迁移

- 支持Liteos、Linux内核
- 支持小型、标准系统

支持多种USB设备模型

- 支持host模式与device模式
- 支持ACM、ECM 设备模型

支持USB PnP特性

- 支持设备热插拔等常见特性

Camera驱动模型简介

Camera驱动模型关键能力特征：

提供丰富的用户体验能力

- 预览：多路预览
- 拍照：拍照方向配置、拍照镜像、防抖模式
- 录像：录像帧率控制、录像抓拍
- 公共：地理位置信息

Camera能力增强

- 查询/设置3A参数
- 查询与设置JPEG质量
- 查询与设置分辨率
- 查询Camera Sensor捕获角度
- 指定角度对拍照数据做旋转

安全隐私能力

- 隐私相机保护
- 人脸识别能力支持（meta流）

适配容易

- Camera驱动模型提供通用框架实现，只需适配驱动层即可让Camera/ISP硬件快速接入系统

显示驱动模型简介

显示驱动模型核心特征：

基础显示能力
- 硬件合成
- GPU合成
- 2D gfx合成
- 显存分配、释放、映射

显示性能提升
- Layer SMQ，跨进程缓冲机制，提升显示多图层的合成性能

多屏支持
- 外接屏显示能力（如HDMI）
- 多屏显示

适配容易
- 显示驱动模型提供通用框架实现，只需适配驱动层即可让显示硬件快速接入系统
- 兼容FB显示架构

Sensor/马达/Light驱动模型功能简介

Sensor/马达/Light驱动模型核心特征：

提供丰富的用户体验能力

- Motion能力：系统定义融合感知能力，可以丰富用户体验
- 马达振幅与振频能力：为用户提供差异化的震感反馈；

基础能力增强

- 传感器数据上报周期
- 获取传感器精度、取值范围、厂商等设备信息能力

丰富的传感器类型支持：

- 九轴传感：加速计，陀螺仪，地磁
- 光感：环境光，接近光
- 体感：心率计

适配容易

Sensor/马达/Light驱动模型提供通用框架实现，只需适配驱动层即可让硬件快速接入系统

OpenHarmony

目录 Contents

HDF驱动社区化开发---开发板项目

OpenHarmony

共12+家共20+款芯片及对应开发板已进OpenHarmony主干，标准7款、小型2款、轻量11款；并通过兼容性认证；为南向开发者提供了丰富的参考样例，可访问下面链接获取详细信息：

https://gitee.com/openharmony/docs/blob/master/zh-cn/device-dev/dev-board-on-the-master.md

系统类别	开发板	芯片	代码仓
标准系统	润和HH-SCDAYU200	RK3568	device_soc_rockchip、device_board_hihope、vendor_hihope
	Hispark_Phoenix	Hi3751V351	device_soc_hisilicon、device_board_hisilicon、vendor_hisilicon
	Unionpi Tiger	Amlogic A311D	device_soc_amlogic、device_board_unionman、vendor_unionman
	MILOS_Standard0	NXP i.MX8M Mini	device_soc_nxp、device_board_osware、vendor_osware
	扬帆开发板	RK3399	device_soc_rockchip、device_board_isoftstone、vendor_isoftstone
	致远开发板	T507	device_soc_allwinner、device_board_isoftstone、vendor_isoftstone
	khdvk_3566b	Rk3566	device_soc_rockchip、device_board_kaihong、vendor_kaihong
小型系统	Hispark_Taurus	Hi3516DV300	device_soc_hisilicon、device_board_hisilicon、vendor_hisilicon
	BearPi-HM Micro	STM32MP157A	device_soc_st、device_board_bearpi、vendor_bearpi
轻量系统	小熊派BearPi-HM Nano	Hi3861	device_soc_hisilicon、device_board_bearpi、vendor_bearpi
	Niobe407	STM32F407IGT6	device_soc_st、device_board_talkweb、vendor_talkweb
	B91 Generic Starter Kit	TLSR9x	device_soc_telink、device_board_telink、vendor_telink
	cst85_wblink	cst85f01	device_soc_chipsea、device_board_chipsea、vendor_chipsea
	Neptune100	W800	device_soc_winnermicro、device_board_hihope、vendor_hihope
	小凌派-RK2206	RK2206	device_soc_rockchip、device_board_lockzhiner、vendor_lockzhiner
	NiobeU4	ESP32U4WDH	device_soc_esp、device_board_openvalle、vendor_openvalley
	Multi-modal V200Z-R	BES2600	device_soc_bestechnic、device_board_fnlink、vendor_bestechnic
	朗国LANGO200	ASR582X	device_soc_asrmicro、device_board_lango、vendor_asrmicro
	汇顶GR5515-STARTER-KIT	GR5515	device_soc_goodix、device_board_goodix、vendor_goodix
	芯海CS1262	CS1262	vendor_hisilicon、drivers_peripheral

DriverSIG介绍

OpenHarmony

Driver SIG链接：

https://gitee.com/openharmony/community/blob/master/sig/sig-driver/sig_driver_cn.md

会议

- 会议时间：双周例会，周三下午16:00
- 会议申报：SIG-Driver会议议题收集
- 会议链接：腾讯会议或其他会议
- 会议通知: 请订阅邮件列表获取会议链接
- 会议纪要：查看往期会议纪要，请点此链接

联系方式(可选)

- 邮件列表：sig_driver@openharmony.io
- Zulip群组：https://zulip.openharmony.cn

对驱动框架有兴趣的开发者参与到驱动SIG：

- 可以参加SIG例会
- 可以查看驱动SIG会议纪要
- 代码贡献（比如bug解决），文档贡献；
- 社区论坛
- 申请驱动SIG社区共建项目

欢迎对驱动领域有兴趣的开发者申领共建项目或者bug修复，代码优化等工作；

OpenHarmony

THANK YOU

扫描二维码 关注官方公众号

【官网网址】www.openharmony.cn