HUAWEI

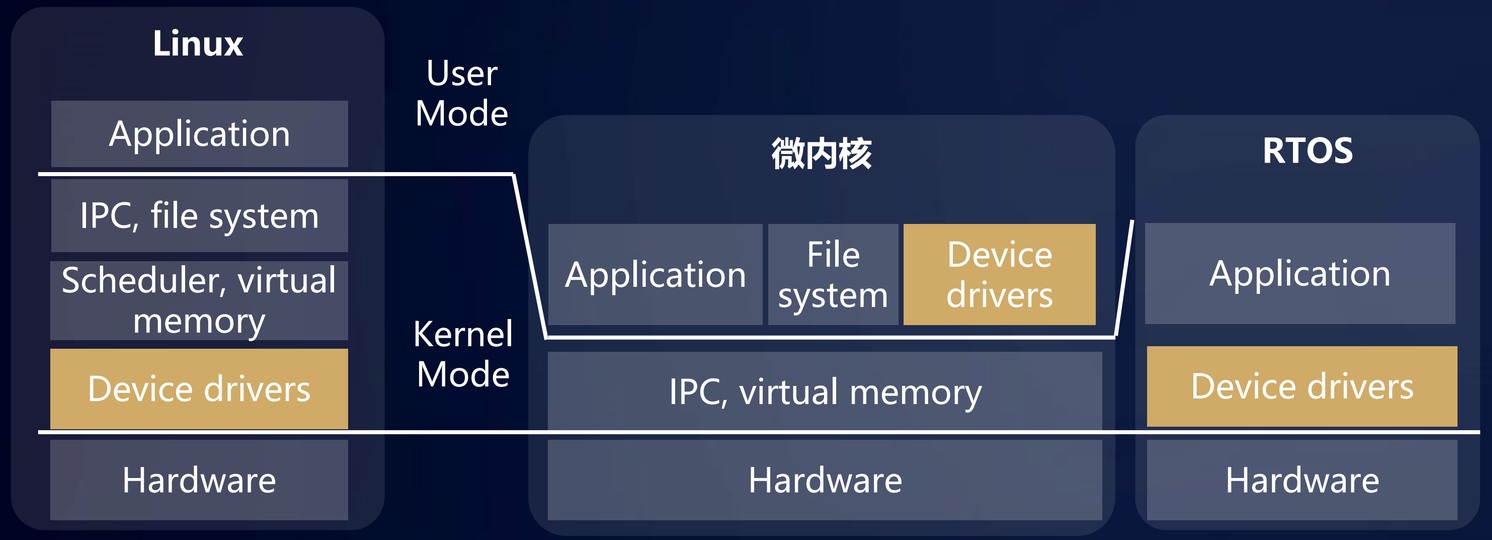
< HDC.Together >

华 为 开 发 者 大 会 2 0 2 0

开发一个HarmonyOS的驱动程序

IOT产业快速发展，给驱动软件带来新的挑战

现状：IOT产品及器件品类增多，驱动软件面临着在不同操作系统间的迁移和维护



期望有个友好的驱动平台，降低驱动开发、迁移和维护成本

驱动开发对平台的诉求：

- 规范化的驱动接口
- 弹性化的驱动框架
- 组件化的驱动模型
- 统一化的配置界面
- 兼容原有驱动生态，并能承载定制化的驱动能力

从业务上看

- OEM厂商需要投入大量的精力适配和维护驱动代码。
- 驱动接口无统一规范，驱动软件迁移成本高。

技术上看

- 与内核强耦合，能力并不均衡，驱动软件迁移难。
- 受License污染，部分驱动核心软件下沉到芯片，或迁移到用户态隔离保护，架构不友好。

提供规范化的、有上述特征型驱动平台，让开发者专注于硬件交互开发

HarmonyOS驱动架构的设计目标

聚焦效率提升，满足不同大小设备的诉求

- 缩减驱动开发周期，丰富平台驱动能力
- 降低驱动集成难度，缩短产品化落地周期

统一驱动平台

归一化平台底座

兼容不同内核，如Linux、LiteOS等。通过平台、系统接口解耦，构建驱动平台底座。

弹性化框架

同时支持百K级~G级容量的1+8+N设备能力，如手机、手环等，不同容量的设备驱动代码同源。

组件化驱动模型

驱动模型组件化分层，合理抽象驱动模型，具备独立构建驱动、简化三方驱动开发难度。

统一配置界面

支持硬件资源和配置抽象，屏蔽硬件差异，支撑开发者开发一致驱动逻辑代码，提升开发及迁移效率。

HarmonyOS驱动平台架构总体构建策略，归一化平台底座

构建统一驱动平台、与内核，芯片平台解耦的驱动框架，规范硬件驱动接口

面向驱动使用者

通过规范化的设备接口（HDI）标准，提供丰富、稳定的接口。

面向驱动开发者

开发环境

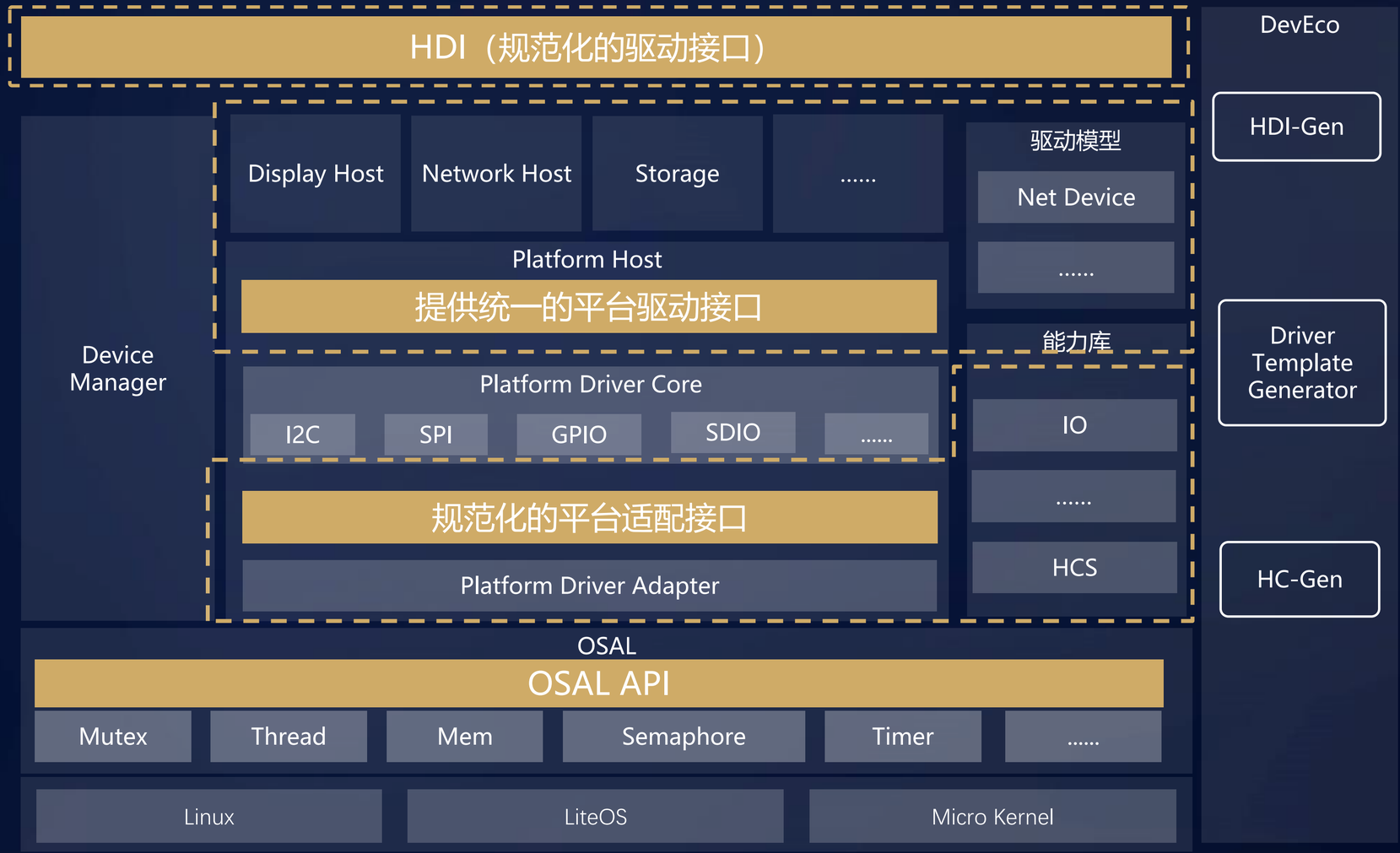
- 内核解耦的、跨平台的驱动开发环境，降低驱动的迁移和维护成本。

驱动模型

- 框架模型屏蔽驱动与操作系统交互的实现。
- 开发者专注硬件交互能力，降低驱动开发难度。

OEM厂商

- 最小粒度的接口适配，支撑平台驱动。
- 统一的配置界面，友好的配置管理能力，使开发效率更高。



构建弹性化的架构能力，可支撑百K级~G级容量的设备部署

采用C面向对象编程方式构建：模块的接口和实现分离，架构功能委派对象管理器管理，实现架构功能的弹性部署

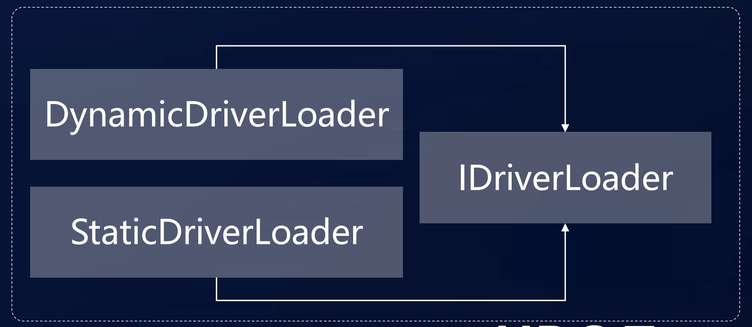
主流程一致确保架构的同源

架构对象ID	功能描述	配置
DEVMGR_SERVICE	设备管理器	必选
DEVSVC_MANAGER	设备IO服务管理	可配置
DEVHOST_SERVICE	宿主服务服务化运行	可配置
DRIVER_LOADER	驱动动态加载	可配置
DRIVER_INSTALLER	支持驱动安装	可配置
.....	.....	可配置

对象配置表

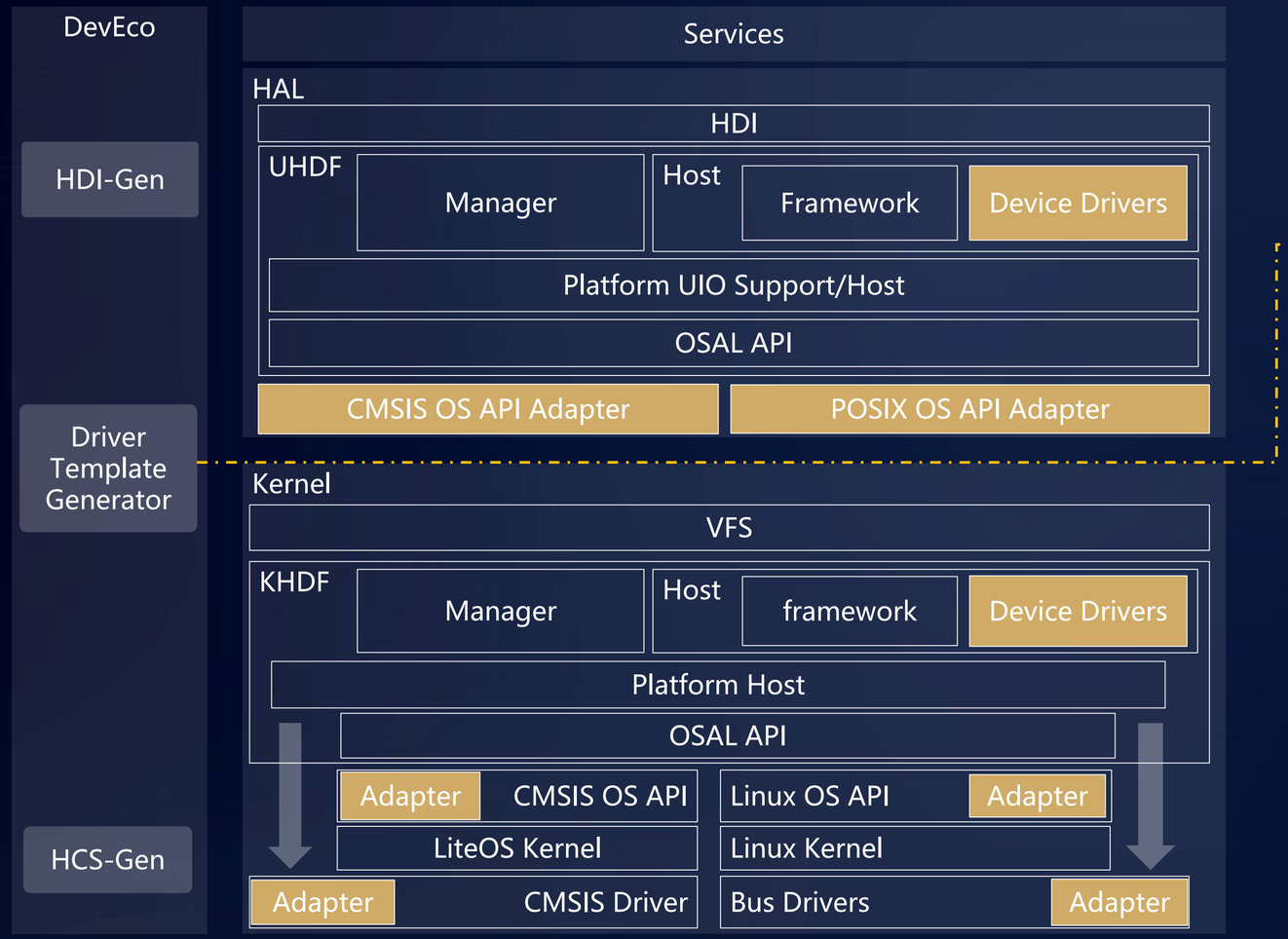
以驱动加载器为例：

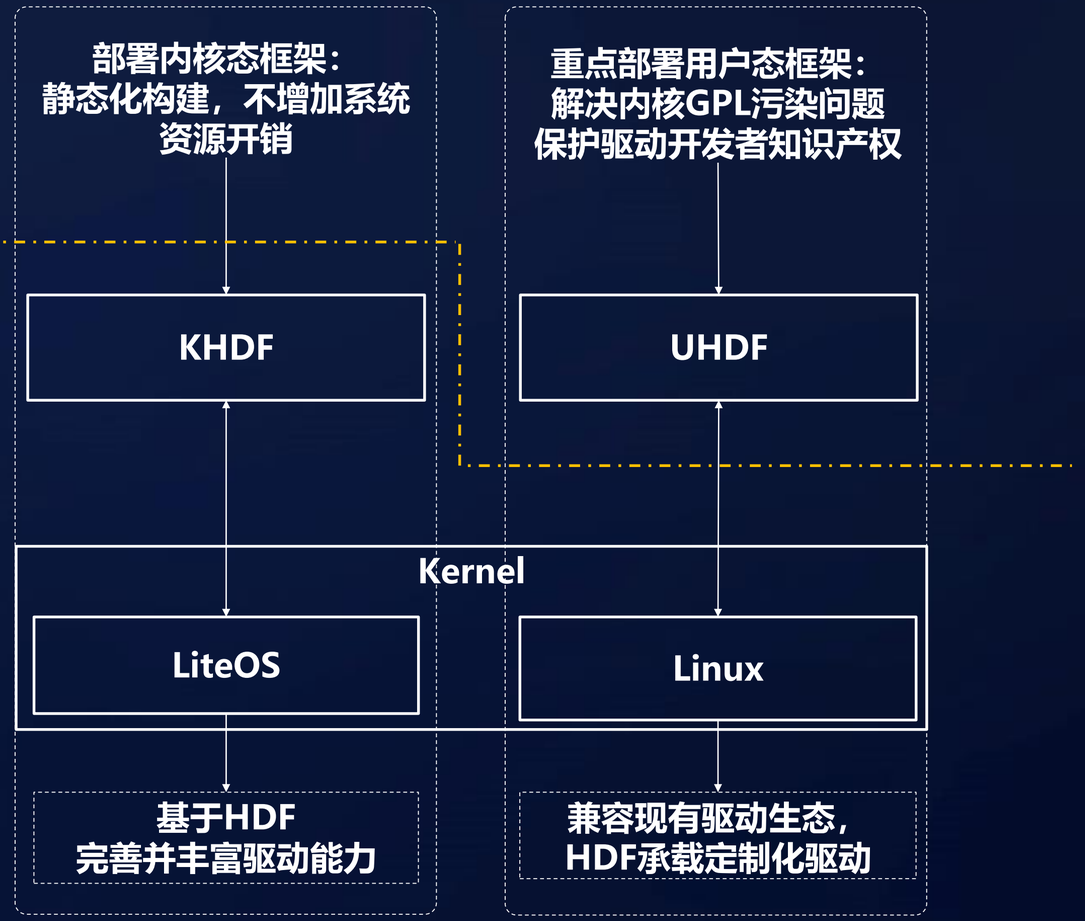
在不改变外部代码逻辑的前提下，通过对象配置指定动态加载和静态加载。



通过编译开关控制部署，按需裁剪增加框架能力，支撑不同大小级别的设备。

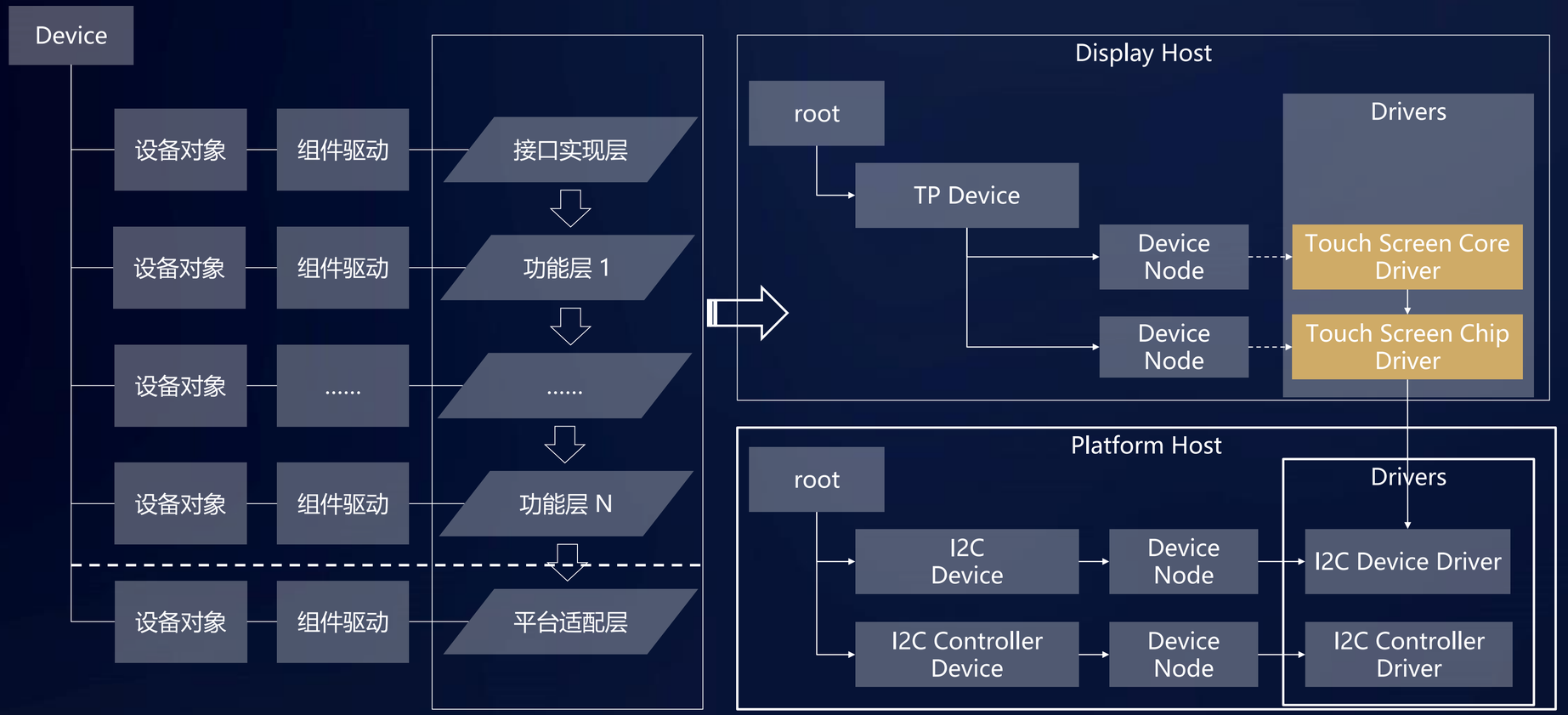
弹性部署策略：支持内核态和用户态驱动部署，满足不同形态的设备诉求





支持组件化的驱动模型，为开发者提供更精细化的驱动管理，让驱动开发和部署更灵活

抽屉式替换：驱动开发者在接口不变的前提下，可以对任意驱动组件替代，更易于维护和部署。
驱动分层：把系统相关和稳定的能力抽出来，合理规范成标准化的组件，并向上提供统一的接口。



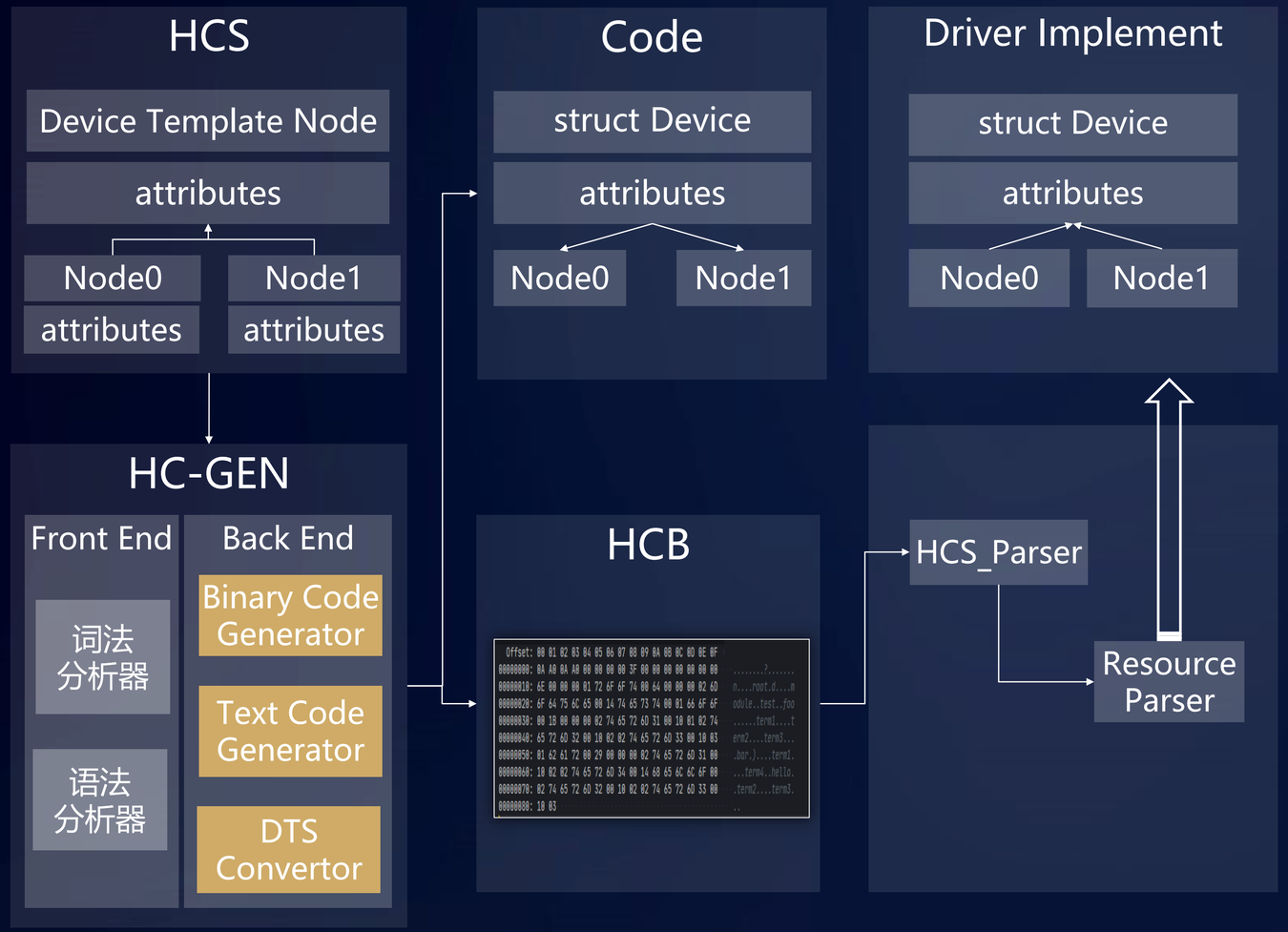
组件化驱动：

支持微驱动模型，驱动开发者可以将驱动功能分层，独立开发部署。

动态挂接

微驱动模型按需求启动，节省运行态内存空间。

提供统一的配置界面，支撑开发者高效的开发驱动软件



结构	树状结构
语法	节点删除、覆盖 同时支持节点复制、 模板和引用修改等特性
输出	二进制
	代码：.c和.h文件
数据类型	支持Bool型\8位\16位\32位\64位\字符串（提供给code转换使用）
方法	未来HCS支持方法配置

主要由如下两种配置组成

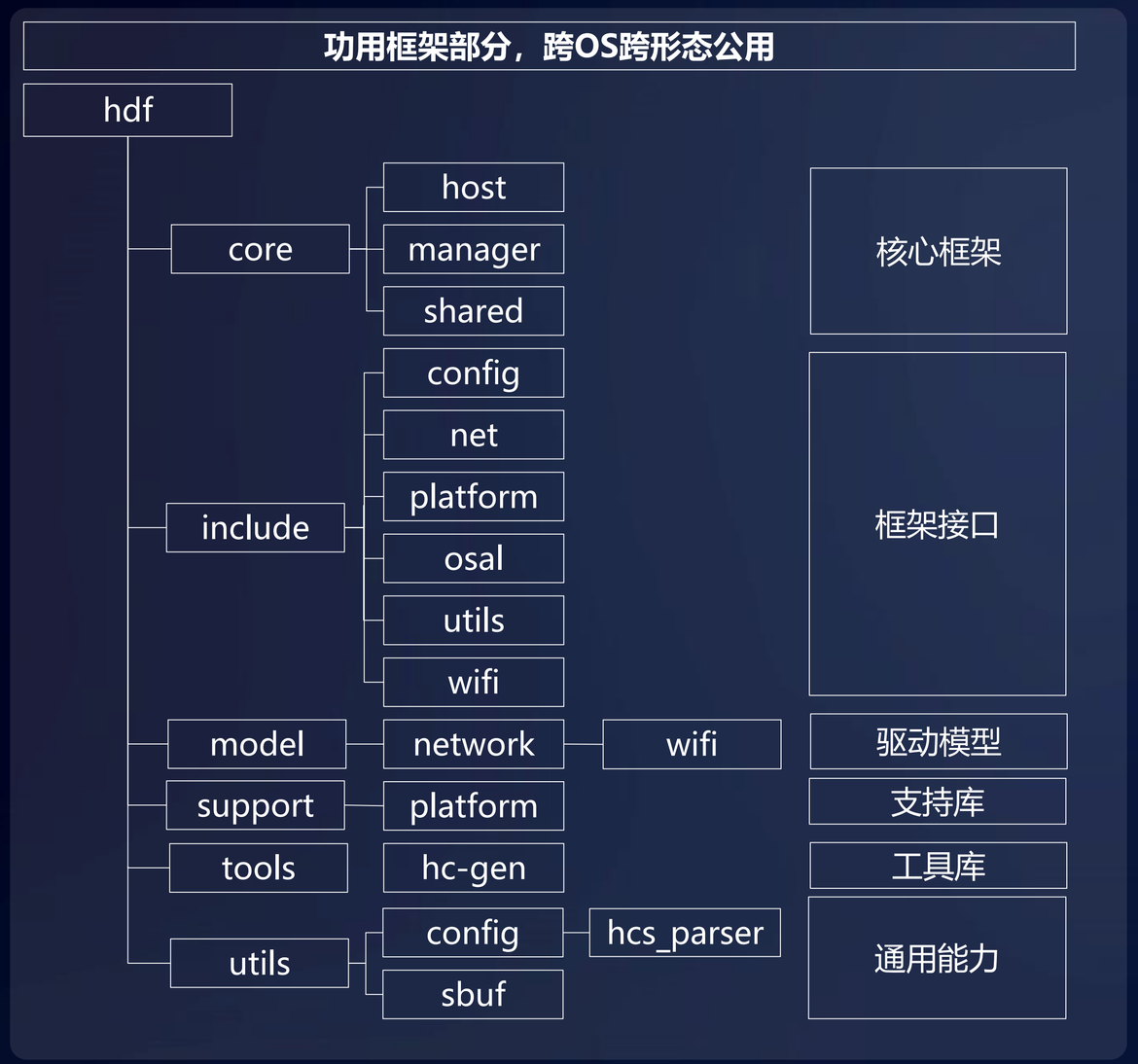
设备信息：

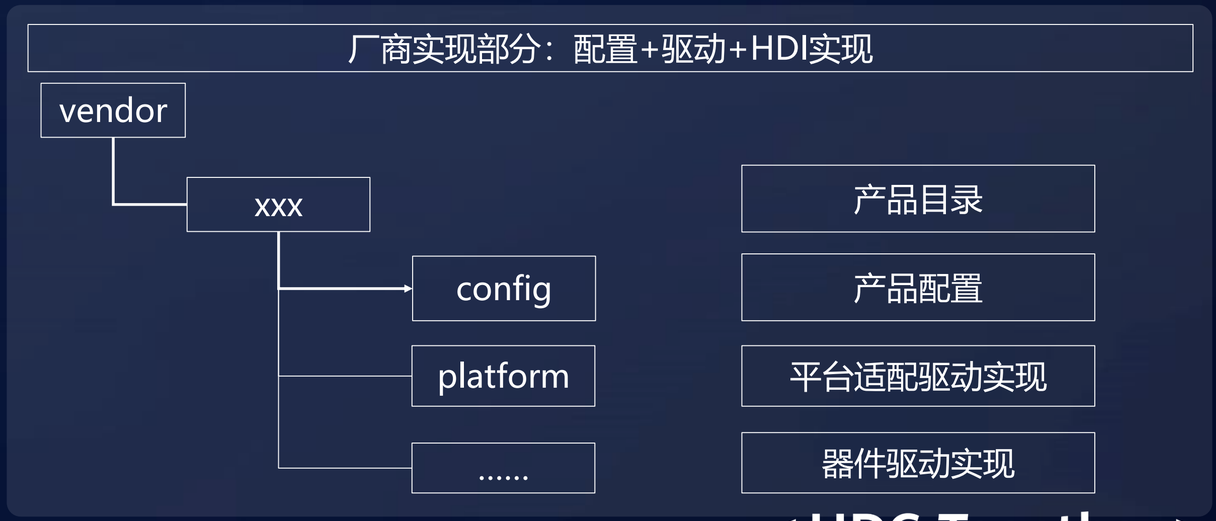
提供标准模板，可继承。

设备资源描述：

开发者可根据具体的设备驱动开发自行设计和定义。

HarmonyOS统一驱动平台目录结构



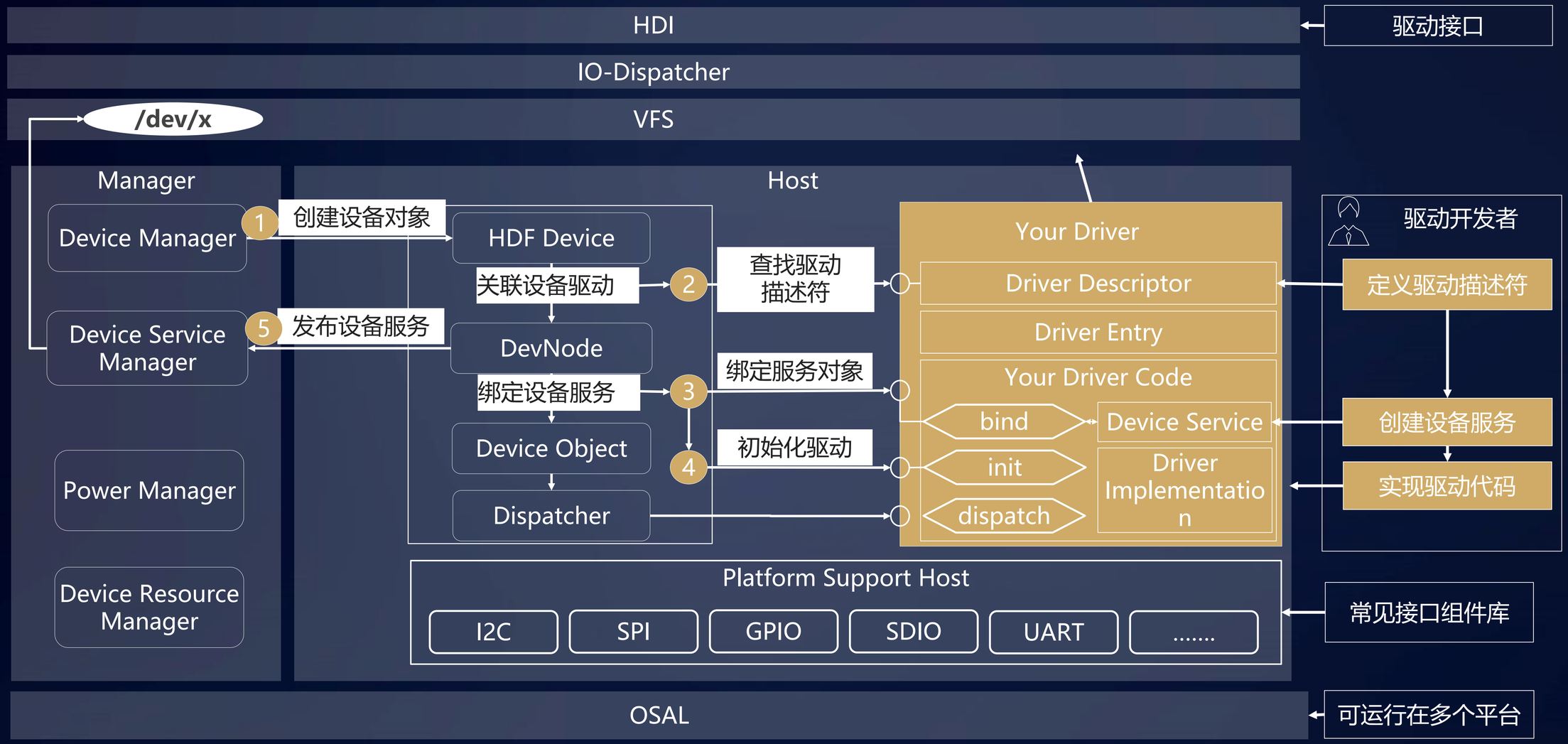


HUAWEI

HarmonyOS驱动开发示例

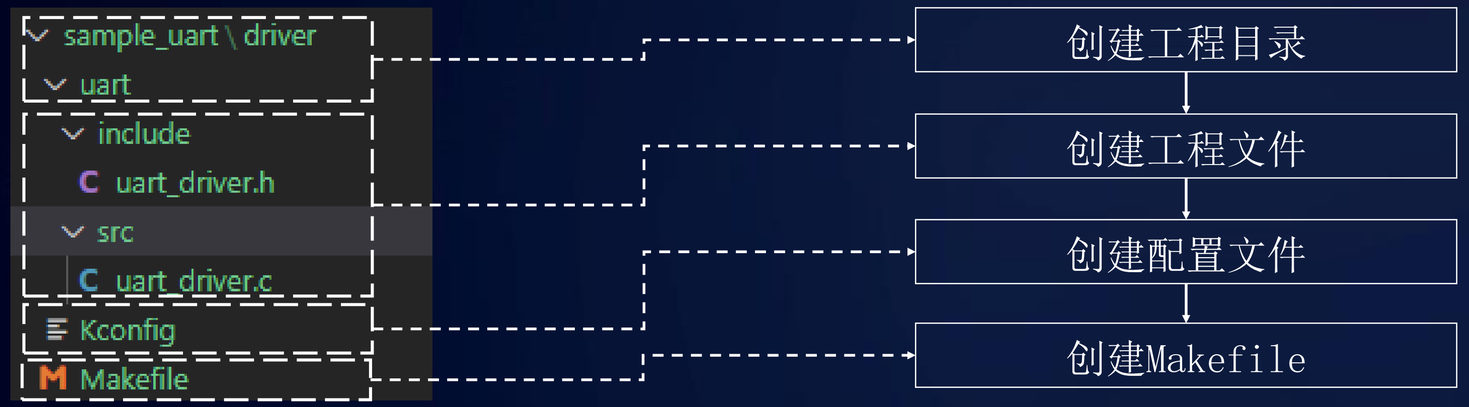
（UART驱动开发实例）

HarmonyOS统一驱动平台和驱动程序交互流程



UART驱动开发实例：创建驱动开发文件

1、添加驱动代码



驱动代码示例

```c
#include "uart_driver.h"
#include "hdf_log.h"

int32_t UartSampleDriverDispatch(
    struct HdfDeviceIoClient *client, int cmdId, struct HdfSBuf *data, struct HdfSBuf *reply)
{
    //TODO: Dispatch message form user
    return 0;
}

int32_t UartSampleDriverBind(struct HdfDeviceObject *deviceObject)
{
    //Bind device service to device object
    if (deviceObject == NULL) {
        return -1;
    }
    static struct UartDriverService service;
    service.uartService.Dispatch = UartSampleDriverDispatch;
    deviceObject->service = (struct IDeviceIoService *)(&service);
    return 0;
}

int32_t UartSampleDriverInit(struct HdfDeviceObject *deviceObject)
{
    //TODO: Initialize device
    return 0;
}

void UartSampleDriverRelease(struct HdfDeviceObject *deviceObject)
{
    //TODO: Release all driver resources
    return;
}

struct HdfDriverEntry uartSampleDriver = {
    .moduleVersion = 0x0001,
    .moduleName = "uart_sample",
    .Bind = UartSampleDriverBind,
    .Init = UartSampleDriverInit,
    .Release = UartSampleDriverRelease,
};

HDF_INIT(uartSampleDriver);
```

配置文件示例

```
config DRIVERS_HDF_SAMPLE_UART
    bool "Enable HDF sample_uart module"
    default n
    depends on DRIVERS_HDF
    help
        Answer Y to enable HDF sample_uart module.
```

Makefile示例

```
include $(LITEOSTOPDIR)/../../drivers/hdf/lite/lite.mk

MODULE_NAME := hdfsample_uart

ifeq ($(LOSCFG_DRIVERS_HDF_UART), y)
    LOCAL_INCLUDE += ./uart/include
    LOCAL_INCLUDE += ./uart/src
    LOCAL_SRCS += $(wildcard ./uart/src/*.c)
endif

include $(HDF_DRIVER)
```

2、创建设备信息文件device_info.hcs

```
root {
    device_info {
        platform :: host {
            hostName = "platform_host";
            priority = 50;
            device_uart :: device {
                device0 :: deviceNode {
                    policy = 1;
                    priority = 100;
                    preload = 0;
                    permission = 0666;
                    moduleName = "uart";
                    serviceName = "uartService";
                    deviceMatchAttr = "hisilicon_hi35xx_uart_1";
                }
            }
        }
    }
}
```

3、创建设备资源配置文件

```
root {
    platfrom {
        template uart_controller {
            match_attr = "";
            num = 0;
            baudrate = 115200;
            fifoRxEn = 1;
            fifoTxEn = 1;
            flags = 4;
            regPbase = 0x120a0000;
            interrupt = 38;
            iomemCount = 0x48;
        }
        controller_0x120a1000 :: uart_controller {
            num = 1;
            baudrate = 9600;
            regPbase = 0x120a1000;
            interrupt = 39;
            match_attr = "hisilicon_hi35xx_uart_1";
        }
    }
}
```

UART驱动开发实例：驱动模板函数说明

HUAWEI

```c
int32_t UartSampleDriverDispatch(
    struct HdfDeviceIoClient *client, int cmdId, struct HdfSBuf *data, struct HdfSBuf *reply)
{
    //TODO: Dispatch message form user
    return 0;
}
```

分发设备服务消息

- cmd Id：请求消息命令字
- Data：其他服务或者IO请求数据
- Reply：存储返回消息内容数据

```c
int32_t UartSampleDriverBind(struct HdfDeviceObject *deviceObject)
{
    //Bind device service to device object
    if (deviceObject == NULL) {
        return -1;
    }
    static struct UartDriverService service;
    service.uartService.Dispatch = UartSampleDriverDispatch;
    deviceObject->service = (struct IDeviceIoService *)(&service);
    return 0;
}
```

绑定设备服务

- 初始化设备服务对象
- 初始化设备资源对象

```c
int32_t UartSampleDriverInit(struct HdfDeviceObject *deviceObject)
{
    //TODO: Initialize device
    return 0;
}
```

驱动初始化函数，

- 探测并初始化驱动程序

```c
void UartSampleDriverRelease(struct HdfDeviceObject *deviceObject)
{
    //TODO: Release all driver resources
    return;
}
```

驱动资源释放函数

- 如已经绑定的设备服务对象

```c
struct HdfDriverEntry uartSampleDriver = {
    .moduleVersion = 0x0001,
    .moduleName = "uart_sample",
    .Bind = UartSampleDriverBind,
    .Init = UartSampleDriverInit,
    .Release = UartSampleDriverRelease,
};

HDF_INIT(uartSampleDriver);
```

定义驱动描述符

- 将驱动代码注册给驱动框架

UART驱动开发实例：创建驱动描述符

```
struct HdfDriverEntry uartSampleDriver = {

    .moduleVersion = 1,

    .moduleName = "uart_sample",

    .Bind = UartSampleDriverBind,

    .Init = UartSampleDriverInit,

    .Release = UartSampleDriverRelease,

};

HDF_INIT(uartSampleDriver);
```

设备模块名称：
与设备信息配置中模块名称保持一致。

指向设备绑定接口：
框架通过该接口绑定设备，并获取设备服务对象。

指向设备初始化接口：
框架通过该接口完成驱动的初始化。

指向设备资源释放接口：
框架在卸载驱动前，通过该接口通知驱动释放资源。

UART驱动开发实例：绑定驱动设备

HUAWEI

绑定设备的目的：根据设备服务策略发布设备/服务节点

```
static int32_t UartSampleDriverBind(struct HdfDeviceObject *device)
{
    static struct UartHost host = { 0 };
    struct UartDevice *uartDevice = NULL;
    if (device == NULL) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return  HDF_ERR_INVALID_OBJECT;
    }
    uartDevice = (struct UartDevice *)
                    OsalMemCalloc(sizeof(struct UartDevice));
    if (uartDevice == NULL) {
        HDF_LOGE("%s: OsalMemCalloc uartDevice error", __func__);
        return HDF_ERR_MALLOC_FAIL;
    }

    ret = UartDeviceGetResource(uartDevice, device->property);
    if (ret != HDF_SUCCESS) {
        (void)OsalMemFree(uartDevice);
        return HDF_FAILURE;
    }
    host->device = device;
    device->service = &(host.service);
    device->service.Dispatch = UartSampleDriverDispatch;

    host->priv = uartDevice;
    host->method = NULL;
    return HDF_SUCCESS;
}
```

获取驱动设备资源

绑定服务对象到框架

UART驱动开发实例：初始化驱动设备

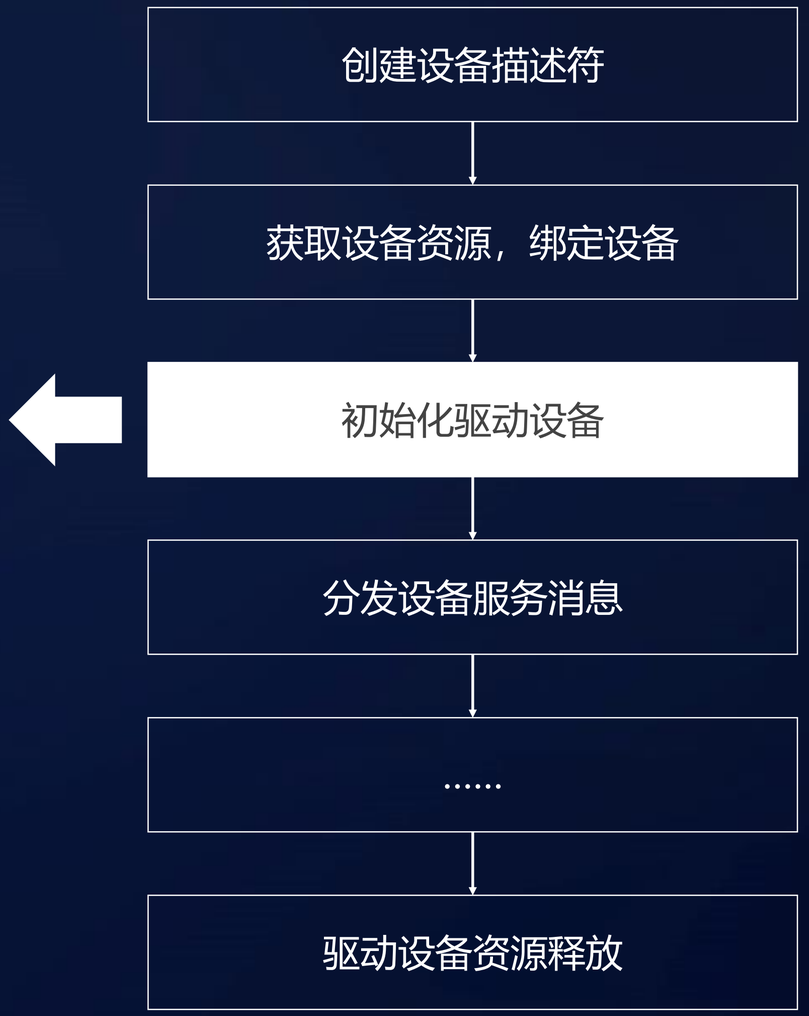
初始化硬件设备

```
static int32_t UartSampleDriverInit(struct HdfDeviceObject *device)
{
    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return HDF_ERR_INVALID_OBJECT;
    }

    struct UartHost *host = UartHostFromDevice(device);
    if (host == NULL) {
        HDF_LOGE("%s: host is null", __func__);
        return HDF_ERR_INVALID_OBJECT;
    }

    struct UartDevice *uartDevice =?struct UartDevice *)host->priv;
    if (uartDevice == NULL) {
        HDF_LOGE("%s: uartDevice is null", __func__);
        return HDF_ERR_MALLOC_FAIL;
    }
    return UartDeviceInit(uartDevice);
}
```

驱动设备初始化



UART驱动开发实例：分发设备服务消息

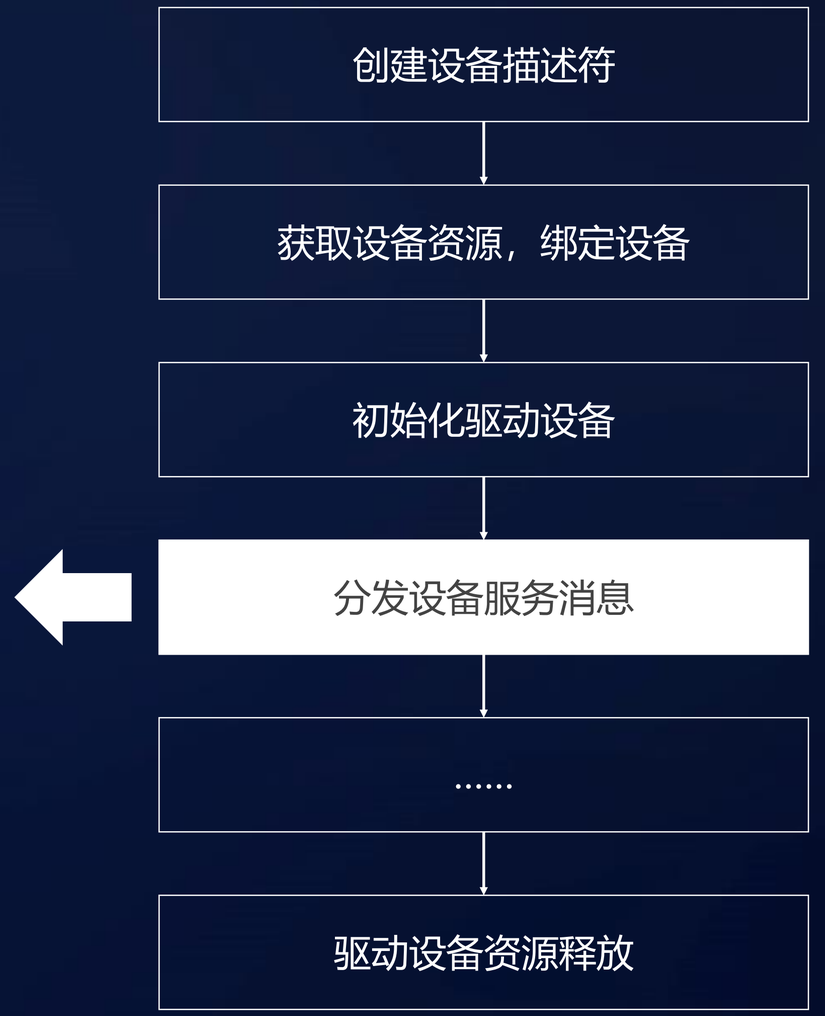
分发处理使用者下发的服务消息

```
int32_t UartSampleDriverDispatch(struct HdfDeviceIoClient *client, int cmdId,
                                 struct HdfSBuf *data, struct HdfSBuf *reply)
{
    int32_t result = HDF_FAILURE;
    if (client == NULL || client->device == NULL) {
        HDF_LOGE("%s: client or client->device is NULL", __func__);
        return result;
    }

    struct UartDevice *uartDevice = (struct UartDevice *)client->device->service;
    if (uartDevice == NULL) {
        HDF_LOGE("%s: uartDevice is NULL", __func__);
        return result;
    }

    switch (cmdId) {
        case UART_WRITE: {
            result = SampleDispatchWrite(uartDevice, data);
            break;
        }
        ……
        default:
            break;
    }
    return result;
}
```

基于消息ID的通信方式
提供序列化机制



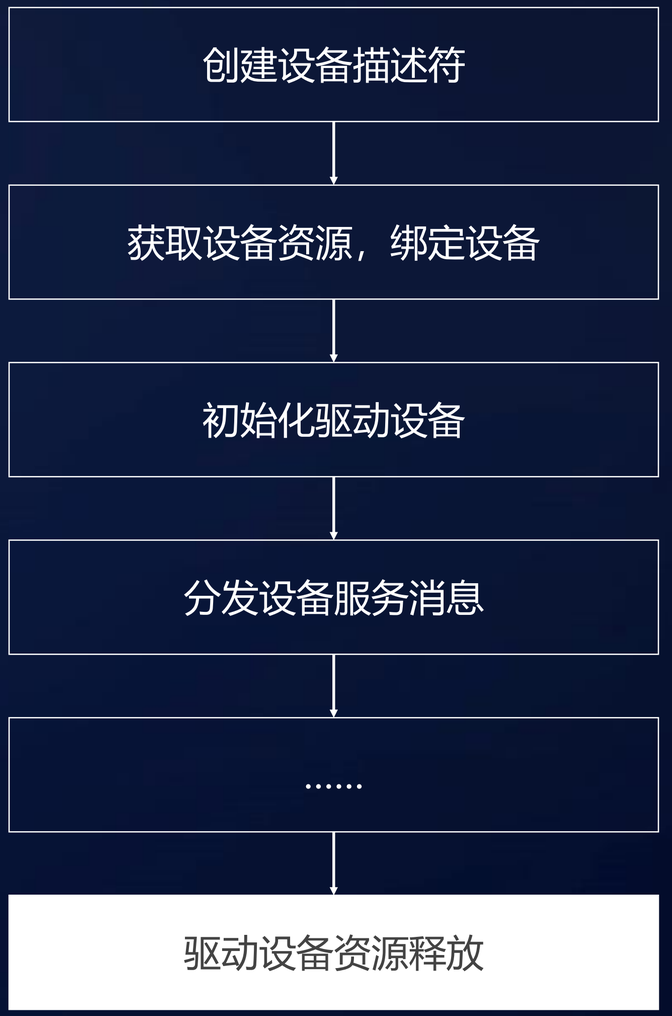
UART驱动开发实例：驱动资源释放

在驱动异常结束或者卸载时释放驱动相关资源。

```
static void UartSampleDriverRelease(struct HdfDeviceObject *device)
{
    struct UartHost *host = NULL;
    HDF_LOGI("Enter %s:", __func__);
    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return;
    }

    host = UartHostFromDevice(device);
    if (host == NULL) {
        HDF_LOGE("%s: host is NULL", __func__);
        return;
    }
    if (host->priv != NULL) {
        struct UartDevice *uartDevice =
            (struct UartDevice *)host->priv;
        UartDeviceDeinit(uartDevice);
        OsalMemFree(uartDevice);
        host->priv = NULL;
    }
}
```

资源释放操作



UART驱动开发实例：驱动设备信息描述

通过HCS构建：设备信息配置文件按照预框架定义的模板格式描述。

```
 template deviceNode {
    policy = 0;
    priority = 100;
    preload = 0;
    permission = 0664;
    moduleName = "";
    serviceName = "";
    deviceMatchAttr = "";
}
```

模板继承

```
device_uartpl011 :: device {
    device0 :: deviceNode {
        policy = 2;
        priority = 50;
        permission = 0644;
        moduleName = " uart_sample_module";
        serviceName = “uart_sample";
        deviceMatchAttr = "uart_sample";
    }
}
```

驱动的设备信息使用HCS文件描述，节点属性含义如下：

policy：设备服务发布策略

- 0-不创建驱动service
- 1-对内核开放的service接口
- 2-对内核和用户态开放的service接口

priority 加载优先级

- 0~49-板级驱动
- 50~149-设备驱动
- 150~200-接口插拔类设备对应的安装驱动

preload：设备加载策略

- 0-默认加载
- 1-默认不加载，使用过程中按需加载

permission：vfs中创建的设备文件权限

moduleName：该设备驱动Module名称，用于和driver匹配

serviceName：创建的设备服务名称

deviceMatchAttr：匹配设备资源配置信息

UART驱动开发实例：驱动设备资源描述

开发者可以按需编写自己的配置节点，用于保存寄存器地址、硬件资源、产品相关属性等内容，在驱动加载过程中，通过配置解析的接口获取驱动设备资源描述信息。

```
uart_sample {
    num = 5;
    base = 0x120a0000;
    irqNum = 38;
    baudrate = 115200;
    uartClk = 24000000;
    wlen = 0x60;
    parity = 0;
    stopBit = 0;
    match_attr = "sample_uart_5";
}
```

约束：配置节点的格式与代码结构保持一致

```
struct UartResource {
    uint32_t num;      /* UART port num */
    uint32_t base;     /* UART PL011 base address */
    uint32_t irqNum;      /* UART PL011 IRQ num */
    uint32_t baudrate;    /* Default baudrate */
    uint32_t wlen;        /* Default word length */
    uint32_t parity;         /* Default parity */
    uint32_t stopBit;       /* Default stop bits */
    uint32_t uartClk;       /* UART clock */
    unsigned long physBase;
};
```

```
static uint32_t UartDeviceGetResource(
    struct UartDevice *device, const struct DeviceResourceNode *resourceNode)
{
    struct UartResource *resource = &device->resource;
    struct DeviceResourceIface *dri = NULL;
    dri = DeviceResourceGetIfaceInstance(HDF_CONFIG_SOURCE);
    if (dri == NULL || dri->GetUint32 == NULL) {
        HDF_LOGE("DeviceResourceIface is invalid");
        return HDF_FAILURE;
    }

    if (dri->GetUint32(resourceNode, "num", &resource->num, 0) != HDF_SUCCESS) {
        HDF_LOGE("uart config  read num fail");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "base", &resource->base, 0) != HDF_SUCCESS) {
        HDF_LOGE("uart config  read base fail");

    ……

    return HDF_SUCCESS;
}
```

UART驱动开发实例：配置编译&反编译

HC-Gen是将HCS配置源文件转化为二进制的工具，在开发者网站获取源码页面下载， 提供Windows和Linux两种版本 。

生成HCB配置文件方法

```
hc-gen -o [OutputFileName] [SourceFileName]
```

生成代码文件方法

```
hc-gen -t [OutputFileName] [SourceFileName]
```

为了验证HCB二进制编译的正确性或者debug，可以将hcb文件反编译为hcs

```
hc-gen -o [OutputFileName] -d [SourceFileName]
```

Makefile中集成了HC-Gen的编译过程，在系统构建时，HCS源文件编译产生配置二进制文件并打包到内核镜像中。二进制配置将在HDF框架启动时加载并通过解析接口读取使用。

UART驱动开发实例：Makefile 模板说明

HUAWEI

Makefile模板介绍。

```
HDF_ROOT_DIR = $(LITEOSTOPDIR)/../../drivers/hdf/lite
include $(HDF_ROOT_DIR)/lite.mk
MODULE_NAME :=
LOCAL_SRCS :=
LOCAL_INCLUDE :=
include $(HDF_DRIVER)
```

工具自动实例化目录配置

```
HDF_ROOT_DIR = $(LITEOSTOPDIR)/../../drivers/hdf/lite
include $(HDF_ROOT_DIR)/lite.mk

MODULE_NAME := sample_uart

LOCAL_SRCS :=   ./src/uart_pl011x.c  \
                ./src/uart_device.c \
                ./src/buf_fifo.c

LOCAL_INCLUDE := ./include

include $(HDF_DRIVER)
```

Makefile变量说明：
MODULE_NAME： 模块名
LOCAL_SRCS：源文件列表
LOCAL_CFLAGS：C文件编译选项，选填
LOCAL_INCLUDE：本模块的头文件目录，HDF框架提供的头文件已经默认导入

驱动静态库配置说明：
默认情况驱动目录下Makefile参与编译，同时生成的静态链接库被链接到系统镜像中。如果不需要链接对应生成的代码需要手动配置：
修改路径：驱动目录上一级的hdf_vendor.mk

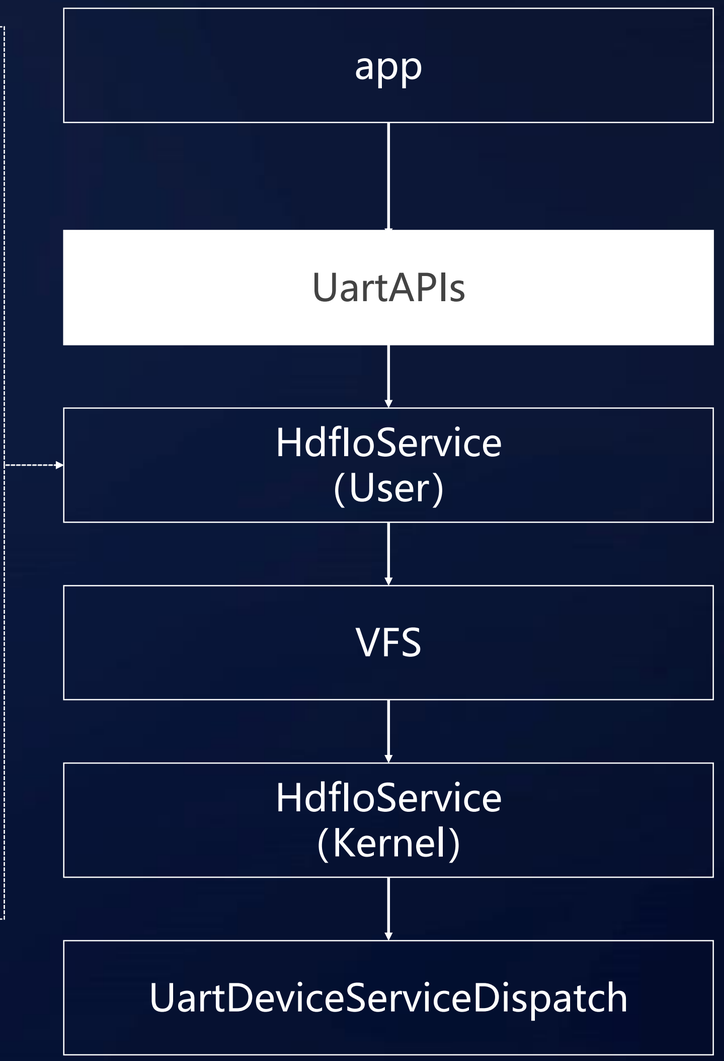
```
#LITEOS_BASELIB += -lsample_uart
#LIB_SUBDIRS    += $(VENDOR_HDF_DRIVERS_ROOT)/sample/uart
```

UART驱动开发实例：用户态使用驱动接口

```
struct HdfIoService *service = HdfIoServiceBind(serviceName, 0);
```

```
int32_t UartWrite(struct DevHandle *handle, uint8_t *data, uint32_t size)
{
    int ret;
    struct HdfIoService *service = NULL;
    if (handle == NULL || handle->object == NULL) {
        HDF_LOGE("handle or handle->object is NULL");
        return HDF_FAILURE;
    }
    struct HdfSBuf *sBuf = HdfSBufObtainDefaultSize();
    if (sBuf == NULL) {
        HDF_LOGE("Failed to obtain sBuf");
        return HDF_FAILURE;
    }

    if (!HdfSbufWriteBuffer(sBuf, data, size)) {
        HDF_LOGE("Failed to write sbuf");
        HdfSBufRecycle(sBuf);
        return HDF_FAILURE;
    }
    service = (struct HdfIoService *)handle->object;
    ret = service->dispatcher->Dispatch(&service->object, UART_WRITE, sBuf, NULL);
    if (ret != HDF_SUCCESS) {
        HDF_LOGE("Failed to send service call");
    }
    HdfSBufRecycle(sBuf);
    return ret;
}
```



欢迎关注HarmonyOS开发者微信公众号