HarmonyOS 驱动加载过程分析

1、HarmonyOS 驱动概述

HarmonyOS 驱动框架采用 C 语言面向对象编程模型构建，通过平台解耦、内核解耦，来达到兼容不同内核，统一平台底座的目的，从而帮助开发者实现驱动的“一次开发、多系统部署”。

为了达成这个目标， HarmonyOS 驱动框架提供了：

1. 操作系统适配层（OSAL，operating system abstraction layer）：提供内核操作相关接口进行统一封装，屏蔽不同系统操作接口。

2.平台驱动接口：提供 Board 部分驱动（例如：I2C/SPI/UART 总线等平台资源）支持，同时对 Board 硬件操作接口进行统一的适配抽象，开发者只需开发新硬件抽象接口，即可获得新增 Board 部分驱动支持。

3.驱动模型：面向器件驱动，提供常见的驱动抽象模型，主要达成两个目的：

1）提供标准化的器件驱动，开发者无需独立开发，通过配置即可完成驱动的部署。

2）提供驱动模型抽象，屏蔽驱动与不同系统组件间的交互，使得驱动更具备通用性。

为了进一步简化 HarmonyOS 驱动开发，HarmonyOS 驱动框架支持多种驱动加载方式：

1.支持驱动动态加载和静态加载，解除驱动代码和框架间的直接代码依赖，使得驱动程序可以独立编译和部署；

2.支持按需动态加载方式，避免设备驱动全量加载，可有效降低系统资源的占用。

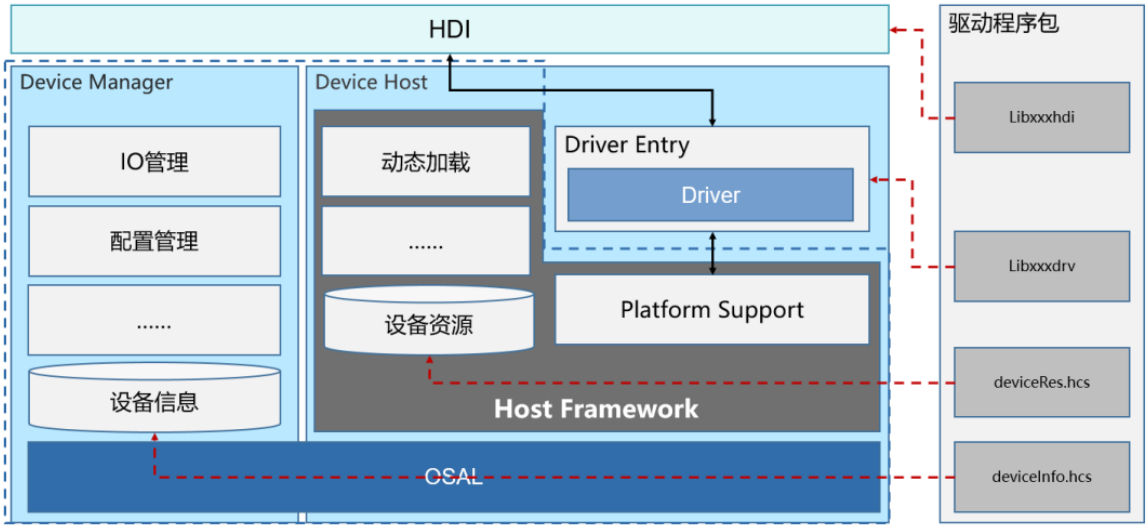
本文主要分析 HarmonyOS 驱动加载过程，在正式介绍之前，首先了解 HarmonyOS 驱动架构的组成、工作原理和机制，从而了解驱动加载的细节。

●官网相关介绍：

https://device.harmonyos.com/cn/docs/develop/drive/oem_drive_hdfdev-0000001051715456

2、HarmonyOS 驱动架构介绍

2.1 HarmonyOS 驱动架构组成



来源：HarmonyOS 开发者微信号 https://mp.weixin.qq.com/s/crH9YJvG1WwTWrQ-jn__nA

图 1 HarmonyOS 驱动架构

HarmonyOS 驱动架构主要由 HDF 驱动框架、驱动程序、驱动配置文件和驱动接口四个部分组成。

1）HDF 驱动框架提供统一的硬件资源管理，驱动加载管理以及设备节点管理等功能。

驱动框架采用的是主从模式设计，由 Device Manager 和 Device Host 组成。

Device Manager 提供了统一的驱动管理，Device Manager 启动时根据 Device Information 提供驱动设备信息加载相应的驱动 Device Host，并控制 Host 完成驱动的加载。

Device Host 提供驱动运行的环境，同时预置 Host Framework 与 Device Manager 进行协同，完成驱动加载和调用。根据业务的需求 Device Host 可以有多个实例。

说明：

- Device Host 顾名思义就是驱动宿主，提供驱动运行的环境。
- 当驱动部署在用户态时，Device Host 可以由独立的进程进行承载。
- 当驱动在部署在内核态时，Device Host 仅表示逻辑隔离。
- Device Host 的划分原则：Device Host 属于一类设备聚合，如 Camera、Audio、Display 等。
- 驱动程序是部署在一个 Device Host 还是部署在不同的 Device Host，主要考虑驱动程序之间是否存在业务耦合性，如果两个驱动程序之间存在依赖，可以考虑将这部分驱动程序

部署在统一 Host。

2）驱动程序实现驱动的具体功能，每个驱动由一个或者多个驱动程序组成，每个驱动程序都对应着一个 Driver Entry。Driver Entry 主要完成驱动的初始化和驱动接口绑定功能。

3）驱动配置文件.hcs 主要由设备信息（Device Information）和设备资源（Device Resource）组成。

Device Information 完成设备信息的配置，如配置接口发布策略，驱动加载的方式等。

Device Resource 完成设备资源的配置，如 GPIO 管脚、寄存器等资源信息的配置。

4）驱动接口 HDI(Hardware Driver interface)提供标准化的接口定义和实现，驱动框架提供 IO Service 和 IO Dispatcher 机制，使得不同部署形态下驱动接口趋于形式一致。

当驱动部署在 RTOS（Real-Time Operating System）轻量化操作系统时，驱动接口和驱动程序之间采用的是 Function Call 方式调用，因此驱动接口仅提供定义，驱动接口实现由驱动程序提供。

2.2 HDF 驱动框架工作原理

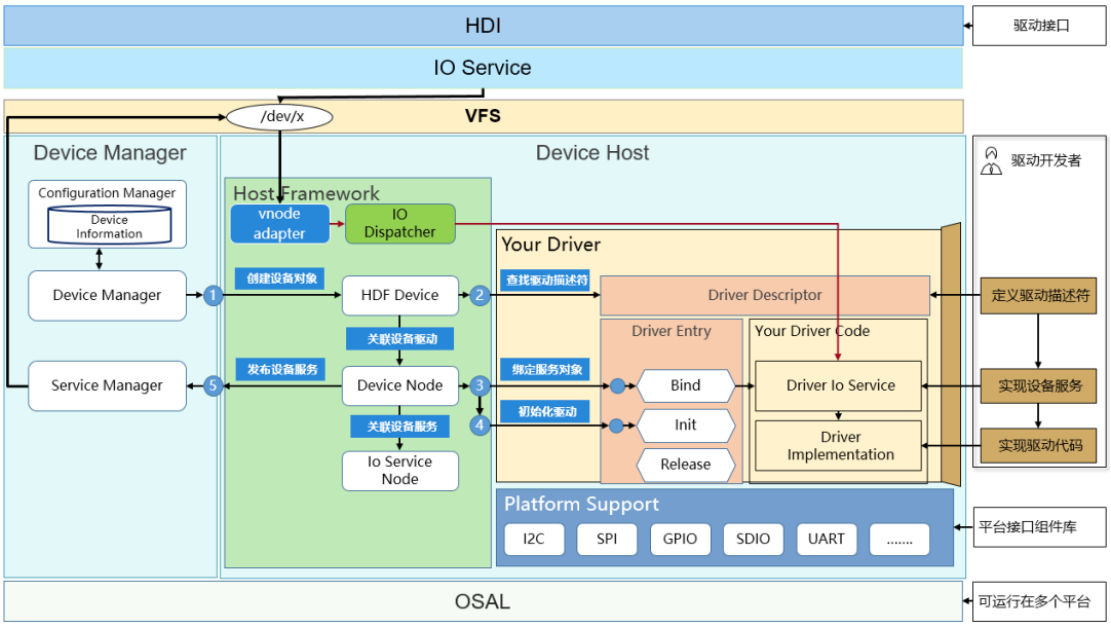


图 2 HDF 驱动框架工作原理

Device Manager 提供了统一的驱动加载管理机制和驱动接口发布机制。

当 Device Host 环境加载完成时，Device Manager 根据 Device Information 信息，请求 Host 加载相应的驱动程序，Device Host 在收到请求时，进行以下操作：

1）根据请求加载设备信息，查找并加载指定路径下驱动镜像或从指定段地址(section)查找驱动程序入口；

2）查找驱动设备描述符，匹配对应的设备驱动；

3）当驱动匹配成功时，加载指定驱动程序镜像；

4）Host Framework 在驱动镜像加载成功后，调用驱动程序（Driver Entry）的绑定接口和初始化接口，实现与驱动程序的服务对象绑定，同时初始化设备驱动程序；

5）当 Device Information 的配置中的服务策略要求对外暴露驱动接口时，驱动框架就将驱动程序的服务对象添加到对外发布的服务对象列表中，外部客户端程序就可以通过此列表

来查询并访问相应的服务接口。

2.3 驱动接口工作机制

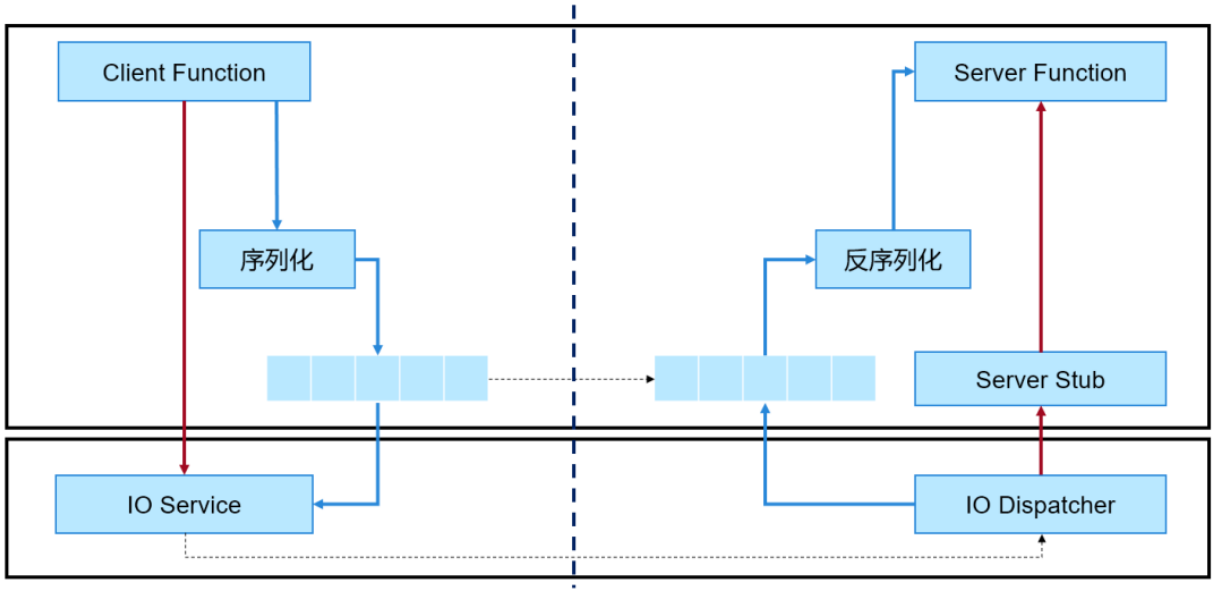


图 3 驱动接口工作机制

驱动接口主要存在以下几种实现：

•当驱动以内核组件部署时，客户端程序访问驱动程序需要通过 system call 方式调用，驱动接口通过 IO Service 请求将消息通过 system call 方式调用到内核，并将消息分发到 IO Dispatcher 处理。

•当驱动以用户态服务形式部署时，客户端进程访问驱动进程需要通过 IPC 方式通信，IO Service 完成 IPC 通信的客户端消息请求封装，IO Dispatcher 完成驱动服务端消息请求封装，客户端消息通过 IPC 通信到达服务端并分发给 IO Dispatcher 处理。

为了使客户端和服务端驱动调用方式基本一致，驱动框架提供 IO Service 和 IO Dispatcher 机制屏蔽了调用消息传递方式的差异。

驱动接口实现统一采用远程调用方式，客户端驱动接口函数将请求序列化成内存数据，通过

驱动框架提供的 IO Service 将消息发送到服务端处理，服务端在收到请求消息时通过 IO Dispatcher 机制将消息分发给消息处理函数处理，处理函数将反序列化内存数据解析成相应的请求。这样做的好处是，开发者只需重点关注接口的定义，无需过多关注如何实现不同平台上接口适配。

3、驱动加载过程分析

HarmonyOS 驱动根据部署的不同方式，存在两种驱动加载方式：

•动态加载方式：采用传统的 so（共享库） 加载方式，驱动程序通过指定 Symbol 找到驱动函数入口进行加载。

•静态加载方式：采用将驱动程序通过 Scatter 编译方式，编译到指定的 Section，再通过访问指定 Section 对应的地址，找到驱动函数入口进行加载。

下面结合一个 Sample 示例代码，讲解驱动加载过程，重点分析静态加载方式下内核态驱动加载过程。

3.1 实现驱动程序初始化接口

在 HDF 驱动框架中，HdfDriverEntry 对象被用来描述一个驱动实现。

```
struct HdfDriverEntry {
    int32_t moduleVersion;
    const char *moduleName;
```

```
    int32_t (*Bind)(struct HdfDeviceObject *deviceObject);
    int32_t (*Init)(struct HdfDeviceObject *deviceObject);
    void (*Release)(struct HdfDeviceObject *deviceObject);
};
```

编写一个简单的驱动，首先需要实现驱动程序(Driver Entry)入口中的三个主要接口：

•Bind 接口：

实现驱动接口实例化绑定，如果需要发布驱动接口，会在驱动加载过程中被调用，实例化该接口的驱动服务并和 DeviceObject 绑定。

•Init 接口：

实现驱动的初始化，返回错误将中止驱动加载流程。

•Release 接口：

实现驱动的卸载，在该接口中释放驱动实例的软硬件资源。

```
int SampleDriverBind(struct HdfDeviceObject *deviceObject)
{
    HDF_LOGE("SampleDriverBind enter!");
    static struct IDeviceIoService testService = {
        .Dispatch = SampleServiceDispatch,
        .Open = NULL,
        .Release = NULL,
    };
    deviceObject->service = &testService;
```

```
    return HDF_SUCCESS;
}

int SampleDriverInit(struct HdfDeviceObject *deviceObject)
{
    HDF_LOGE("SampleDriverInit enter");
     return HDF_SUCCESS;
}

 void SampleDriverRelease(struct HdfDeviceObject *deviceObject)
{
    HDF_LOGE("SampleDriverRelease enter");
    return;
}

 struct HdfDriverEntry g_sampleDriverEntry = {
    .moduleVersion = 1,
    .moduleName = "sample_driver",
    .Bind = SampleDriverBind,
    .Init = SampleDriverInit,
    .Release = SampleDriverRelease,
};
```

HDF_INIT(g_sampleDriverEntry);

3.2 导出驱动程序入口符号

实现驱动程序初始化后，需要将驱动程序入口通过驱动声明宏导出，这样驱动框架才能在启动时识别到驱动程序的存在，驱动才能被加载：

#define HDF_INIT(module)　HDF_DRIVER_INIT(module)

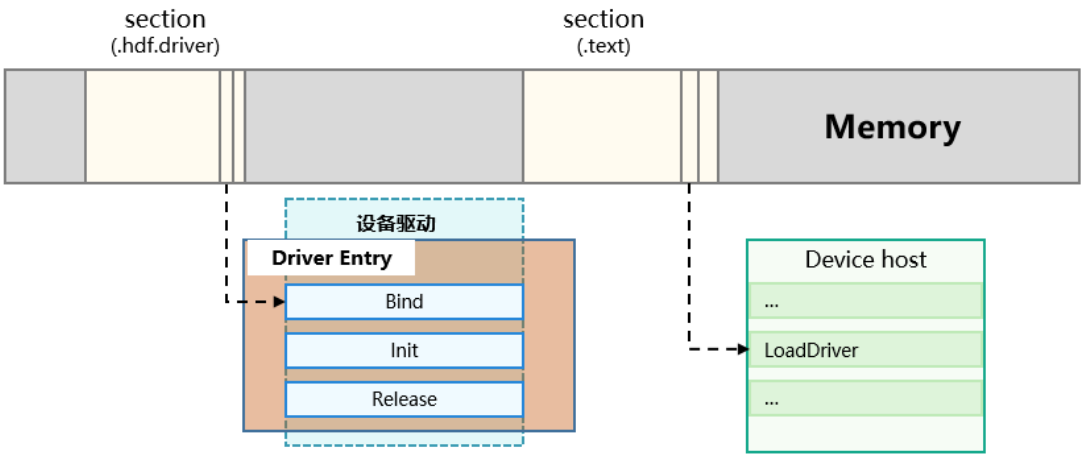
这里将 HDF_INIT 宏展开：

#define HDF_SECTION __attribute__((section(".hdf.driver")))

#define HDF_DRIVER_INIT(module) \

const size_t USED_ATTR module##HdfEntry HDF_SECTION = (size_t)(&(module))

下面是实现原理：



来源：HarmonyOS 开发者微信号 https://mp.weixin.qq.com/s/crH9YJvG1WwTWrQ-jn__nA

图 4 Driver Entry 内存分布

可以看到 HDF_INIT 宏是定义了一个“驱动模块名+HdfEntry”的符号放到".hdf.driver" 所在 section，该符号指向的内存地址即为驱动程序入口结构体的地址。这个特殊的 section 将用于开机启动时查找设备驱动。

3.3 添加设备配置

在设备对应的 device_info.hcs 添加 sample 驱动的配置：

```
sample_host :: host {
    hostName = "sample_host";
    sample_device :: device {
        device0 :: deviceNode {
            policy = 2;
            priority = 100;
            preload = 1;
            permission = 0664;
            moduleName = "sample_driver";
            serviceName = "sample_service";
        }
    }
}
```

在配置中定义的 device 将在加载过程中产生一个设备实例，通过 moduleName 字段指定设备对应的驱动名称，从而将设备与驱动关联起来。其中，设备与驱动可以是一对多的关系，即可以实现一个驱动支持多个同类型设备。

3.4 驱动启动过程

我们添加的驱动是如何被执行的呢？简单来说，在系统启动时，驱动框架先启动，通过解析配置文件获取到设备列表，通过读取".hdf.driver"段读取到驱动程序（Driver Entry）列表，然后遍历设备列表与驱动程序列表进行匹配，并加载匹配成功的驱动。

驱动框架有两大核心管理者：

•**DeviceManager**：负责设备的管理，包括设备加载、卸载和查询等设备相关功能。

•**DeviceServiceManager**：负责管理设备发布的接口服务，提供接口服务的发布和查询等功能。

驱动加载主要由 DeviceManager 主导，首先 DeviceManager 要解析配置文件中的 Host 列表，根据 Host 列表中的信息来实例化对应的 Host 对象。Host 解析配置文件获取到关联的设备列表，遍历设备列表去获取与之匹配的驱动程序名称，然后基于驱动程序名称遍历前面提到的".hdf.driver" section 获得驱动程序地址。

下面介绍具体过程。

3.4.1　获取设备列表

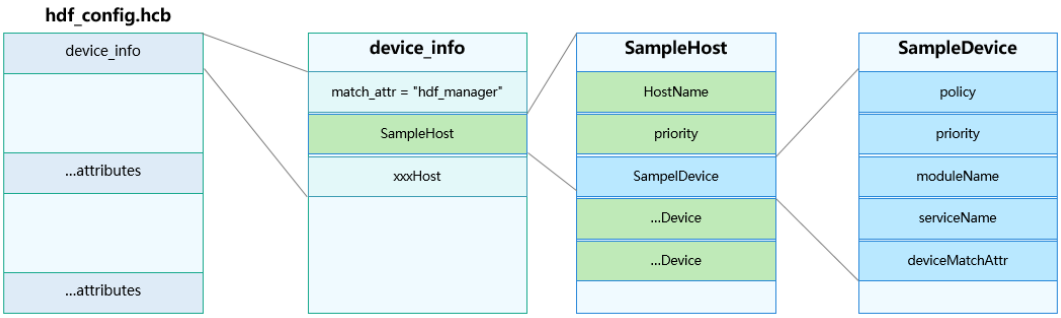


图 5 设备列表结构

配置文本编译后会变成二进制格式的配置文件，其中设备相关信息被存放在一个用“hdf_manager”标记的 device_info 配置块中，host 的内容以块的形式在 device_info 块中依次排列，host 块中记录了 host 名称、启动优先级和设备列表信息。设备信息中的 moduleName 字段将用于和驱动程序入口中的 moduleName 进行匹配，从而为设备匹配到正确的驱动程序。

3.4.2 获取驱动程序列表

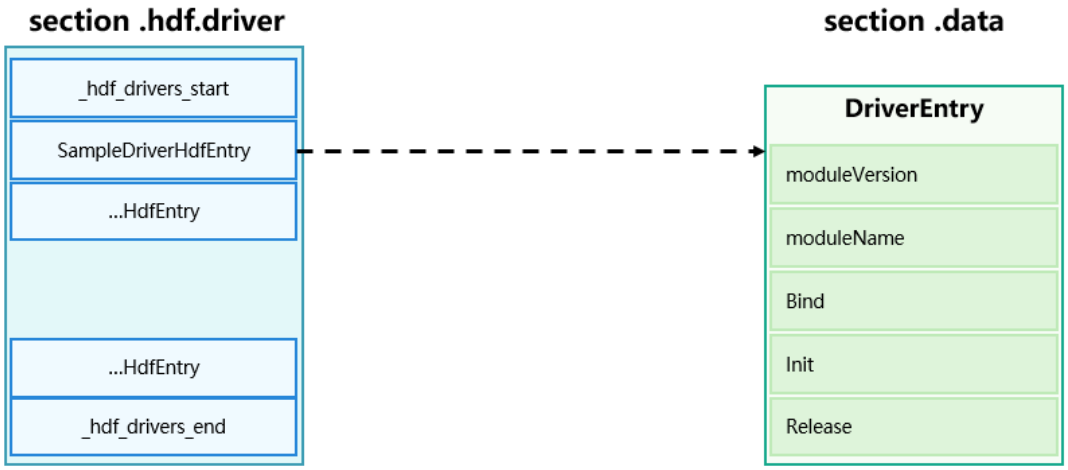


图 6 驱动程序（DriverEntry）内存布局

HDF 驱动框架通过驱动程序入口符号的地址集中存放到一个特殊的 section 来实现对驱动

的索引，这个 section 的开头和末尾插入了_hdf_drivers_start、_hdf_drivers_end 两个特殊符号，用于标记这个 section 的范围，两个特殊符号之间的数据即为驱动实现指针。

3.4.3 驱动程序加载流程

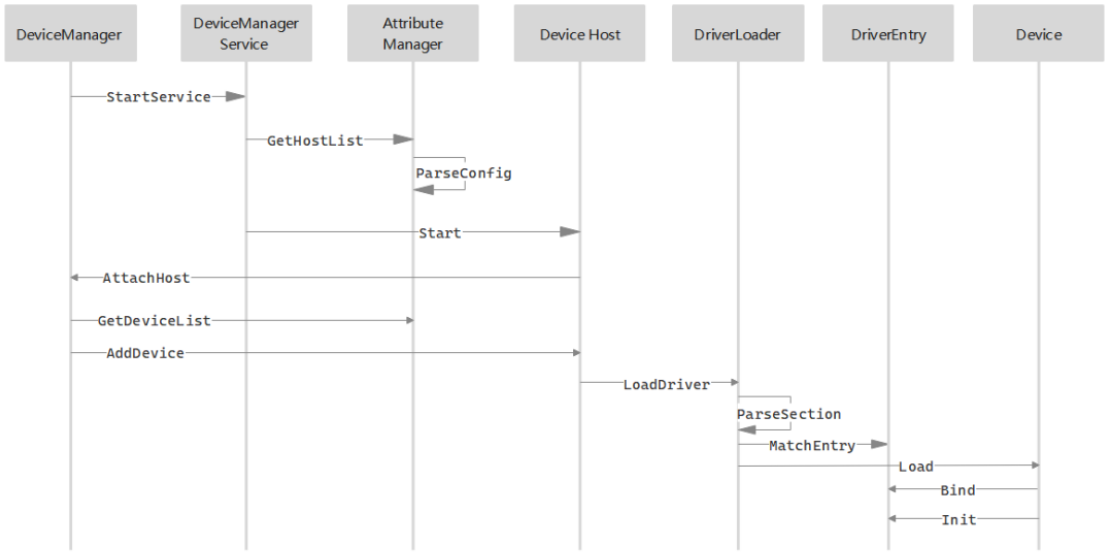


图 7 HDF 驱动加载流程

Device Manager 遍历设备列表，当查找到对应驱动实现时，为设备创建 Device 对象实例，如果设备配置中的 policy 字段为需要对外发布驱动接口(SERVICE_POLICY_CAPACITY)，那么驱动的 Bind 接口将首先被调用，用于关联设备和服务实例。然后驱动的 Init 接口将被调用，用于完成驱动的相关初始化工作。如果驱动被卸载或者因为硬件等原因 Init 接口返回失败，Release 将被调用，用于释放驱动申请的各类资源。

4、总结

本次和大家分享了 HarmonyOS 驱动的主要设计思想，重点分析了内核态驱动加载的过程，关于 HarmonyOS 驱动其他内容，后续会有更多技术文章向大家持续分享，敬请期待。