Rockchip RT-Thread USB 开发指南

文件标识：RK-KF-YF-106

发布版本：V1.3.0

日期：2024-09-23

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文介绍了基于 RT-Thread 的 USB 框架，以及常用的 USB Device/Host 使用方法。

产品版本

芯片名称	**RT-Thread** 版本
RK2108、RK625、RK1808、RK3308、RK3358、RK3568	RT-Thread 3.1.x 及以上版本
RK2118、RK3506	RT-Thread 4.1.x

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师
软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	吴良峰	2019-07-23	初始版本
V1.0.1	陈谋春	2020-03-17	修正链接
V1.0.2	吴良峰	2020-05-25	修正格式
V1.1.0	王明成	2021-03-12	新增 UVC 使用示例
V1.2.0	吴良峰、郑见炜	2024-06-11	新增 RK2118 RT-Thread 4.1.x 说明 新增 Device WINUSB/CDC_VCOM 使用示例 完善 Device HID/MSC 使用示例 完善 Device ADB 使用示例，支持 ADB push/pull/shell 功能 新增 Device 控制器的端点硬件 FIFO 配置说明 新增 Device VBUS Detect 软硬件使用说明 新增 Device speed 配置说明 新增 Host MSC 使用示例 完善 Host 配置说明
V1.3.0	吴良峰	2024-09-23	新增 RT-Thread 4.1.x Device UAC 复合设备的使用方法 新增 RT-Thread 4.1.x Multi Host 的配置方法 新增 RT-Thread 4.1.x RK3506 支持说明

目录

Rockchip RT-Thread USB 开发指南

1. Rockchip MCU USB 简介

- 支持 High-Speed，并向下兼容 Full-Speed 和 Low-Speed
- 支持 USB 控制器 DMA 传输模式
- 支持 Device mode 或者 OTG mode，具体请参考芯片数据手册

表 1 Rockchip MCU USB

芯片	**USB** 控制器型号	支持模式	支持速率	传输方式
RK2108	DWC2	Device only	支持 HS/FS，不支持 LS	DMA
RK2118	DWC2	OTG（Device/Host 动态切换）	Device 支持 HS/FS，Host 支持 HS/FS/LS	DMA
RK625	DWC2	OTG（Device/Host 动态切换）	Device 支持 HS/FS，Host 支持 HS/FS/LS	DMA

Note.

1. Device/Host 模式切换，目前仅支持通过 menuconfig 进行静态配置；
2. Device 默认支持 High-Speed，如果要限制最高速率为 Full-Speed，请参考章节 USB Device speed 配置说明；
3. 其它 AP 芯片，如：RK3308、RK3568 的 USB 特性，请参考文档《Rockchip_Developer_Guide_USB_CN》。

2. 软件开发

2.1 USB 代码路径

RT-Thread USB 组件位于 `rt-thread/components/drivers/usb`，是 RT-Thread 依据 USB2.0 协议规范将 USB 协议栈逻辑高度抽象，支持 Host（主机）模式和 Device（从机）模式。 该组件在驱动移植方面提供了友好的移植接口，开发者可将厂商 BSP 中的 HCD（Host Controller Driver）和 PCD（Peripheral Controller Driver）驱动代码直接接入到 RT-Thread。

USB 组件的框架代码路径如下：

```
components/drivers/usb/usbdevice/core  /* USB Device 核心代码，实现 Device 协议栈 */
components/drivers/usb/usbdevice/class /* USB Device 设备类驱动 */
components/drivers/usb/usbhost/core    /* USB Host 核心代码，实现 Host 协议栈 */
components/drivers/usb/usbhost/class   /* USB Host 设备类驱动 */
```

USB 驱动适配层的代码路径如下：

RT-Thread v3.1.x

```
bsp/rockchip/common/drivers/drv_usbd.c    /* RT-Thread 与 HAL USB Device DWC2 控制器驱动之间的适配层 */
bsp/rockchip/common/drivers/drv_usbh.c    /* RT-Thread 与 HAL USB Host DWC2 控制器驱动之间的适配层 */
bsp/rockchip/common/drivers/drv_usbhost.c /* RT-Thread 与 HAL USB Host EHCI&OHCI 控制器驱动之间的适配层 */
```

RT-Thread v4.1.x

```
bsp/rockchip/common/drivers/drv_usbd.c    /* RT-Thread 与 HAL USB Device DWC2 控制器驱动之间的适配层 */
bsp/rockchip/common/drivers/drv_usbotgh.c /* RT-Thread 与 HAL USB Host DWC2 控制器驱动之间的适配层 */
bsp/rockchip/common/drivers/drv_usbhost.c /* RT-Thread 与 HAL USB Host EHCI&OHCI 控制器驱动之间的适配层 */
```

Note.

- USB Device Class 支持

1. CDC-ACM（Communication Device Class - Abstract Control Model，USB 虚拟串口）
2. CDC-ECM（Communication Device Class - Ethernet Networking Control Model，USB 以太网控制模型）
3. HID（Human Interface Device, 人体学输入设备，支持 Keyboard/Mouse/General HID/Media keyboard）
4. MStorage（Mass Storage，大容量存储设备）
5. RNDIS（Remote Network Driver Interface Specification，远程网络驱动接口）
6. WINUSB（Windows USB, 微软通用 USB）

- USB Host Class 支持

1. MStorage（Mass Storage，大容量存储设备）
2. HUB（基础驱动已支持，功能在调试中）
3. HID（基础驱动已支持，功能在调试中）

2.2 USB 框架介绍

USB 框架是基于 RT-Thread 提供的 I/O 设备模型框架 实现的，如下图 1 所示，从上到下分别是 I/O 设备管理层、RTT USB 组件、RTT 与 HAL 的适配层、HAL 层 USB 控制器驱动。

图 1 USB 框架图

- 应用程序通过 I/O 设备管理接口来访问 USB 设备类驱动，然后通过 USB 设备类驱动与底层的USB 控制器进行数据（或控制）交互
- I/O 设备管理层实现了对 USB 设备类驱动程序的封装，提供标准接口给应用程序
- RTT USB 组件实现了 USB Device/Host 协议栈以及各种 USB 设备类驱动。USB 设备类驱动通过 USB 协议栈（USB Core）提供的标准接口与底层的 USB 控制器进行数据交互。USB 协议栈通过适配层提供的标准接口与 USB 控制器进行数据（或控制）交互
- RTT 与 HAL 的适配层，作用是配置 Rockchip USB 控制器的硬件信息，并通过 HAL 层 USB 控制器驱动提供的接口初始化 USB 控制器。同时，根据 RTT USB 组件定义的标准接口，将 HAL 层 USB 控制器驱动抽象为方便 RTT USB 组件直接访问的接口函数
- HAL 层 USB 控制器驱动实现了对 USB 控制器进行读写操作的底层代码，可以适配不同的 RTOS

Note.

RT-Thread 官方论坛上有开发者总结了 USB 组件之 USB device 框架的简单概览，有兴趣请参考：

USB组件之USB device框架简单概览

2.3 USB 配置

2.3.1 USB Device 配置

USB Device 可以支持一个 interface 或者多个 interface 组合的复合设备（Composite Device），USB 组件允许开发者通过宏 `RT_USB_DEVICE_COMPOSITE` 开启复合功能。

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [ ] Using USB host
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0007) USB Product ID
            [ ]   Enable composite device
                    Device type (Using custom class by register interface)  --->
```

Note.

1. `USB Vendor ID` 为 0x2207，是 USB-IF 官方授权给 Rockchip 的 VID，具有唯一性。USB-IF 规定 VID 为每个 Vendor 独有，同一个 VID 不能授权给不同的 Vendor 使用。因此 Rockchip USB VID 仅供产品开发和调试使用，如果产品需要过 USB-IF 认证，需要开发商自己申请 USB-IF 会员，以获取官方授权的唯一 Vendor ID；
2. `USB Product ID` 为 0x0007，可以根据实际产品来定义，规则是 0x00XX，保持高8位为0，避免与 Maskrom/Loader USB PID 冲突；
3. `Device type` 可以根据产品实际需求，进一步选择 CDC/MSC/HID/WINUSB/UAC1 等；
4. `Enable composite device` 用于支持 USB 复合设备，如：MSC + CDC-ACM 的组合。如果是单一 USB 功能，可以不用使能该选项；
5. USB Device 功能主要适用于 MCU 芯片，如：RK2108/RK2118/RK625。

2.3.2 USB Host 配置

USB Host 目前仅支持 Mass Storage 功能，HUB 和 HID 功能正在开发中。

Mass Storage 功能的配置方法

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [*] Using USB host
            [*]   Enable Udisk Drivers
            (/)     Udisk mount dir
            [ ]   Enable HID Drivers
            [ ] Using USB device
            (4096) usb thread stack size
```

USB Host 驱动适配层配置方法

```
RT-Thread rockchip common drivers  --->
    RT-Thread rockchip USB Host driver  --->
        [ ] DWC2 HCD Support
        [ ] EHCI HCD Support
        [ ] OHCI HCD Support
```

Note.

1. USB Host 功能适用于 MCU 芯片和 AP 芯片。Host 驱动适配层的配置，需要根据实际芯片 USB 控制器的型号选择 DWC2 或者 EHCI + OHCI；

2. 由于 USB 驱动适配层的限制，不支持 `Using USB host` 和 `Using USB device` 同时使能，否则，会导致 USB 功能异常。

2.4 USB Device 创建和注册

USB Device 通过 `rt_device_register()` 接口注册到 RT-Thread I/O 设备管理器中。

这里的 USB Device 是指 USB 控制器在 RT-Thread I/O 设备管理器中注册的设备，该设备只提供给 USB 设备类驱动（如HID/MStorage等）访问，应用程序无法直接访问 USB 控制器设备。此外，不同的 USB 设备类驱动，可能还会在 RT-Thread I/O 设备管理器创建和注册对应的设备，以方便应用程序访问 USB 设备，比如 `components/drivers/usb/usbdevice/class/hid.c` 通过 `rt_device_register` 接口创建 `hidd` 设备节点。

USB 控制器 Device 的创建和注册是在 RTT 与 HAL 的适配层 `bsp/rockchip/common/drivers/drv_usbd.c` 实现，流程如下图 2 所示。

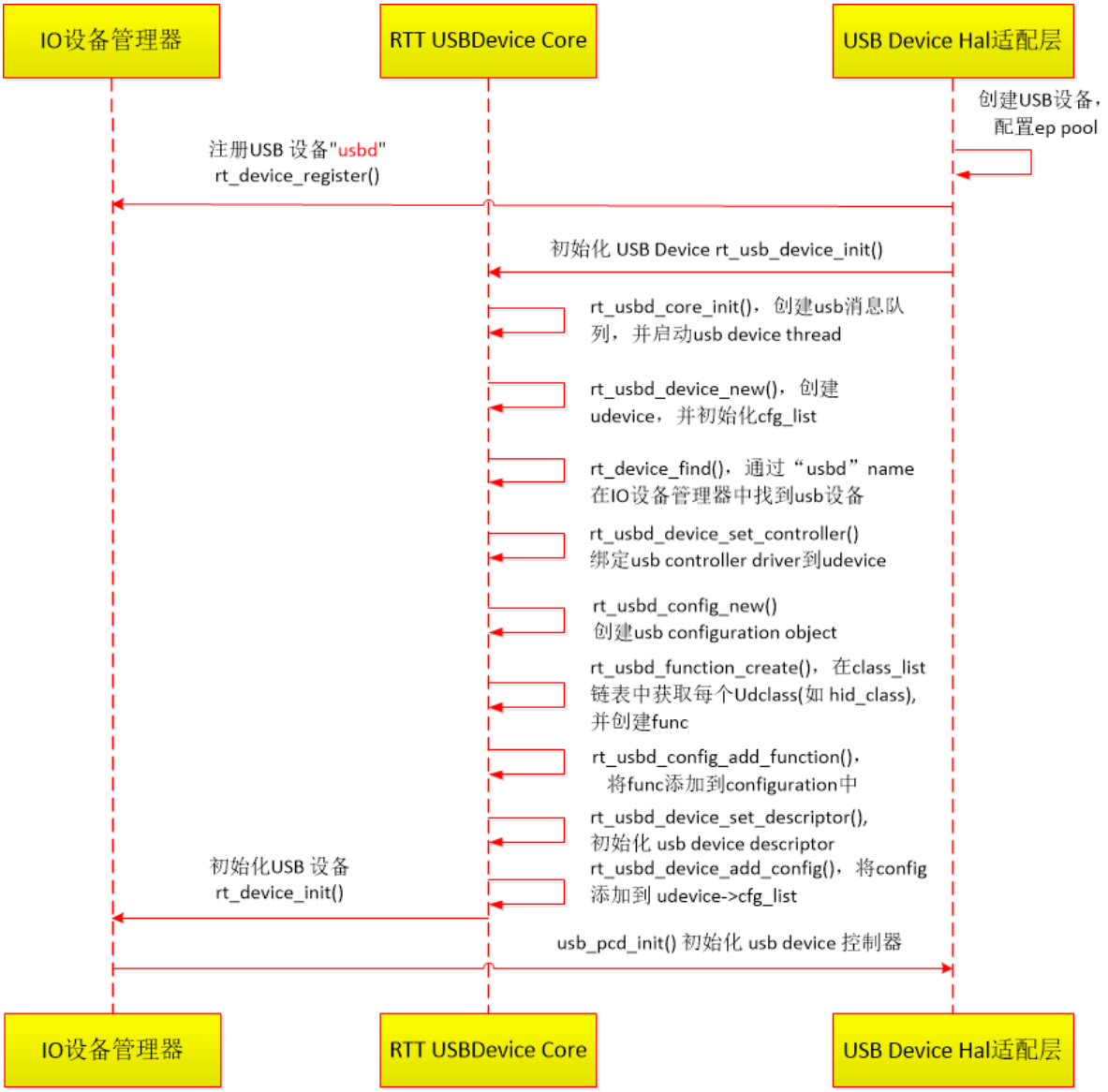

图 2 USB Device 的创建和注册流程

执行命令 `list_device` 可以看到已经生成的 USB Device 设备：

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
usbd     USB Slave Device     0
```

2.5 USB Host 创建和注册

USB Host 通过 `rt_device_register()` 接口注册到 RT-Thread I/O 设备管理器中。通过创建一个 USB Hub 线程实现对端口的实时监听，当端口状态出现变化时枚举设备。同时，通过初始化一个设备类驱动链表，将不同的驱动添加到链表中实现对各种设备类驱动的管理。

RT-Thread 支持三种不同的 USB HOST 控制器类型的驱动适配层，分别是：DWC2/EHCI/OHCI，对应的代码路径请参考章节 USB 代码路径

以控制器 DWC2 的创建和注册为例，流程如下图 3 所示。

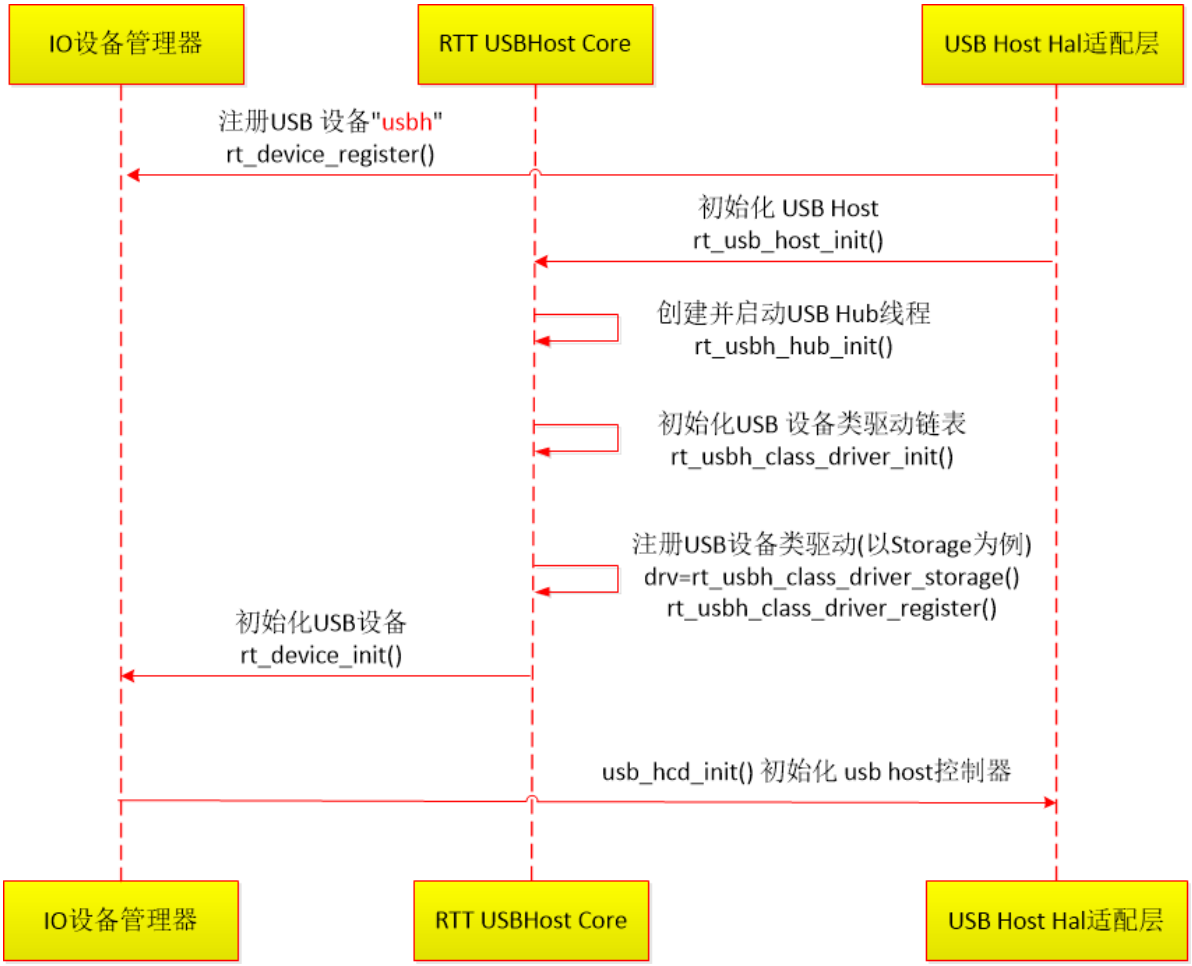

图 3 USB Host 的创建和注册流程

Note.

如果控制器类型为 EHCI/OHCI，对应的驱动适配层文件是 `bsp/rockchip/common/drivers/drv_usbhost.c`，创建和注册流程与 DWC2 控制器基本一致。

执行命令 `list_device` 可以看到已经生成的 USB Host 设备：

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
usbh     USB Host Bus         0
```

2.6 USB Device 使用示例

2.6.1 USB Device 设备管理接口

USB 应用程序通过 RT-Thread 提供的 I/O 设备管理接口来访问 USB 硬件，相关接口如下所示：

表 2 USB Device 设备管理接口

函数	描述
rt_device_find()	查找 USB 设备
rt_device_open()	打开 USB 设备
rt_device_read()	读取 USB 数据
rt_device_write()	写入 USB 数据
rt_device_control()	控制 USB 设备

2.6.2 USB Device HID 使用示例

RT-Thread USB 组件支持常见的 USB Device HID 功能，包括：Keyboard, Mouse, General HID, media keyboard。用户可以根据实际的需求配置对应的 HID 功能。下面以 RK2108 USB HID Keyboard 为例，说明 HID 的配置和使用方法。

USB Device HID 配置

执行命令 `scons --menuconfig`，将 USB 配置为 HID。

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [ ] Using USB host
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0007) USB Product ID
            [ ]   Enable composite device
                Device type (Enable to use device as HID device)  --->
            [*]   Use to HID device as Keyboard
            (1)     Number of Keyboard(max 3)
            [ ]   Use to HID device as Mouse
            [ ]   Use to HID device as General HID device
            [ ]   Use to HID device as media keyboard
```

执行命令 `list_device` 可以看到已经生成的 USB HID 设备，属于字符设备类型：

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
hidd     Character Device     0
```

USB Device HID 的创建和注册流程

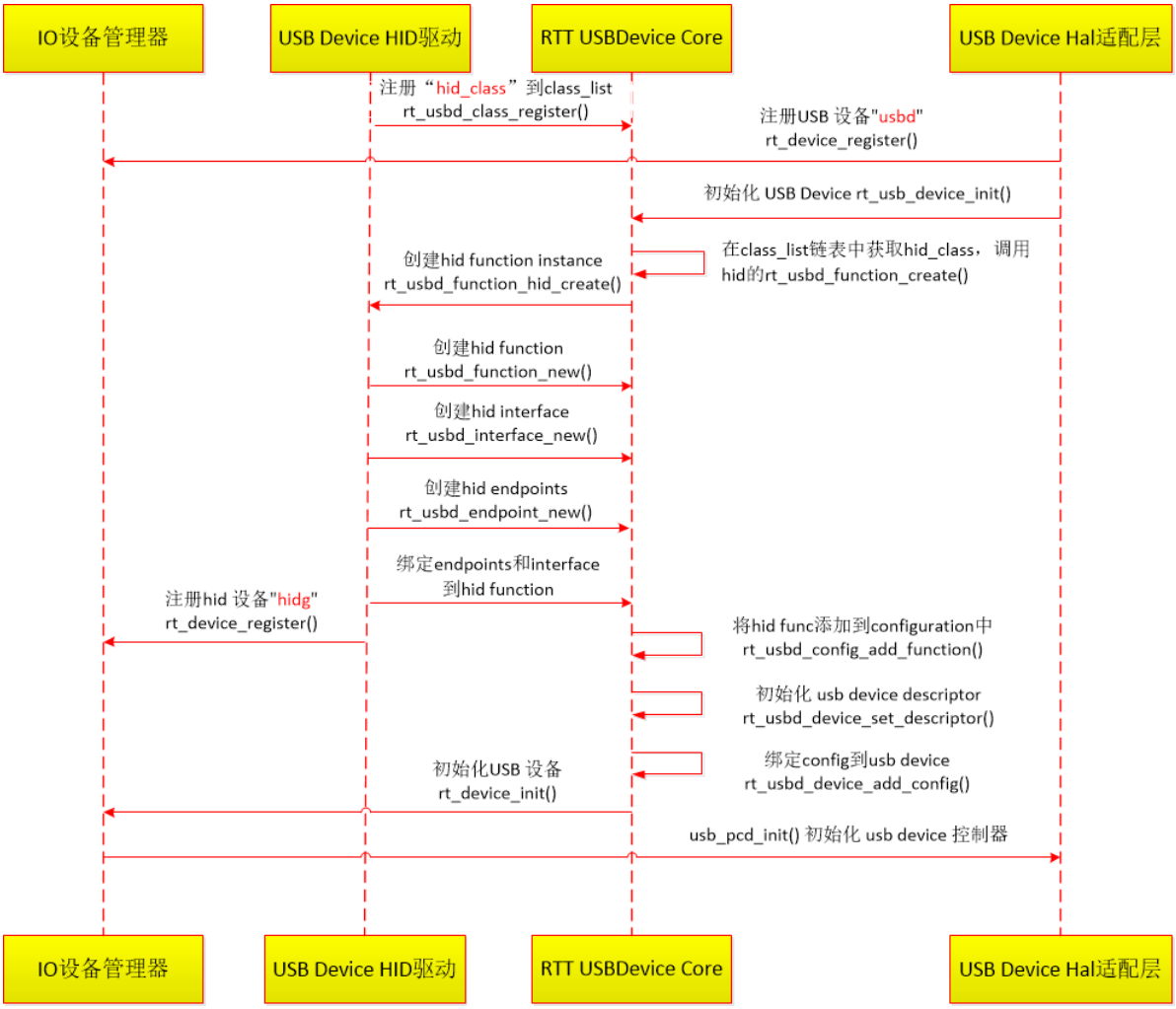

图 4 USB Device HID 的创建和注册流程

应用程序访问 **HID** 设备的流程

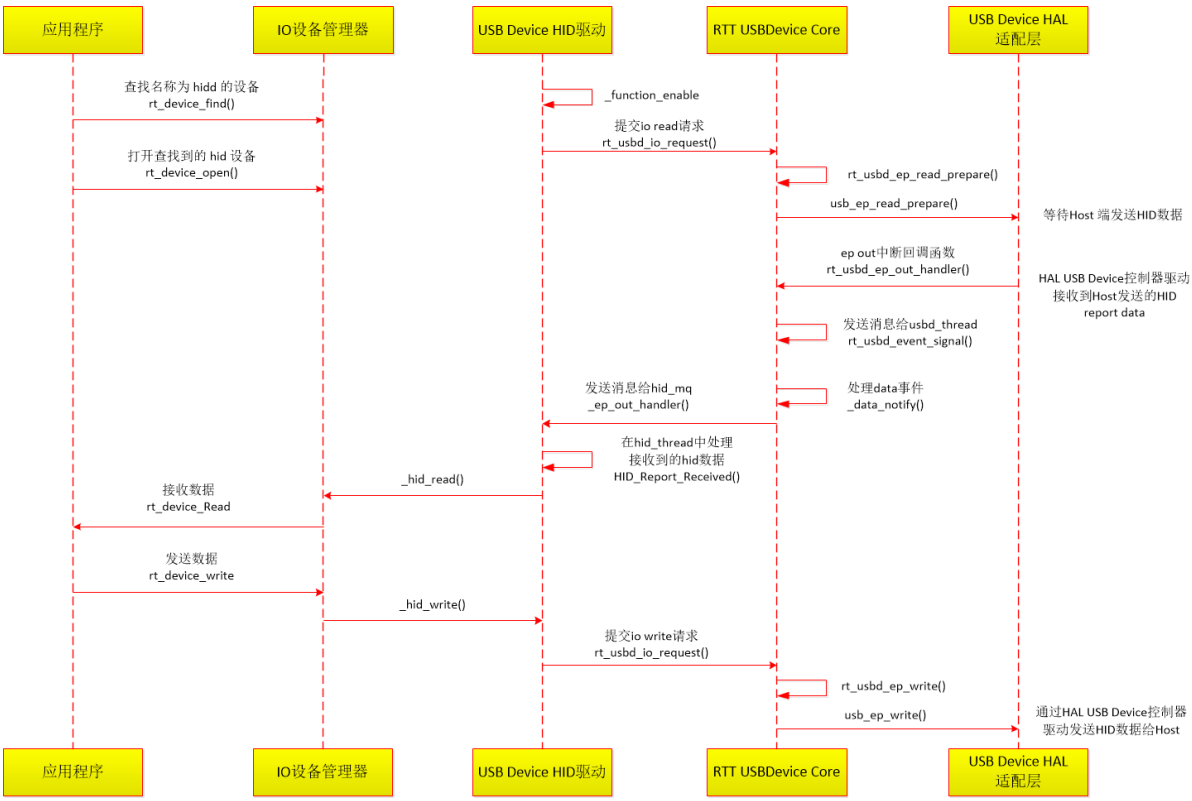

图 5 HID 应用程序访问设备的流程

USB Device HID 应用开发参考用例

RT-Thread/IoT_Board 的 USB 鼠标例程

RT-Thread RK2108 HID 参考用例

USB Device HID 传输性能优化

RT-Thread USB Device HID 受限于 HID 驱动的报告描述符以及 USB 控制器 INT 端点的 FIFO 配置，一次最大只能传输 64 bytes 的数据，传输性能比较差。可以参考章节 USB Device 端点 FIFO 配置说明 优化 HID 的传输性能，支持一次传输 1024 bytes 的数据。

以 RK2108 平台为例：

```
diff --git a/bsp/rockchip/rk2108/board/common/board_base.c
b/bsp/rockchip/rk2108/board/common/board_base.c
index 79cc20854..a494d7528 100644
--- a/bsp/rockchip/rk2108/board/common/board_base.c
+++ b/bsp/rockchip/rk2108/board/common/board_base.c
@@ -484,8 +484,8 @@ RT_WEAK struct ep_id g_usb_ep_pool[] =
     { 0x2,  USB_EP_ATTR_BULK,       USB_DIR_OUT,    512,  ID_UNASSIGNED },
     { 0x3,  USB_EP_ATTR_ISOC,       USB_DIR_IN,     1024, ID_UNASSIGNED },
     { 0x4,  USB_EP_ATTR_ISOC,       USB_DIR_OUT,    512,  ID_UNASSIGNED },
-    { 0x5,  USB_EP_ATTR_INT,        USB_DIR_IN,     64,   ID_UNASSIGNED },
-    { 0x6,  USB_EP_ATTR_INT,        USB_DIR_OUT,    64,   ID_UNASSIGNED },
+    { 0x5,  USB_EP_ATTR_INT,        USB_DIR_IN,     1024,   ID_UNASSIGNED },
+    { 0x6,  USB_EP_ATTR_INT,        USB_DIR_OUT,    1024,   ID_UNASSIGNED },
     { 0xFF, USB_EP_ATTR_TYPE_MASK,  USB_DIR_MASK,   0,    ID_ASSIGNED   },
 };
 #endif
diff --git a/components/drivers/include/drivers/usb_common.h
b/components/drivers/include/drivers/usb_common.h
index d16b42ea7..2370cdc88 100644
```

```
--- a/components/drivers/include/drivers/usb_common.h
+++ b/components/drivers/include/drivers/usb_common.h
@@ -508,7 +508,7 @@ typedef struct uhid_descriptor* uhid_desc_t;
 struct hid_report
 {
     rt_uint8_t report_id;
-    rt_uint8_t report[63];
+    ALIGN(32) rt_uint8_t report[1024];
     rt_uint8_t size;
 };
 typedef struct hid_report* hid_report_t;
diff --git a/components/drivers/usb/usbdevice/class/hid.c
b/components/drivers/usb/usbdevice/class/hid.c
index f0e75178b..fdb12001e 100644
--- a/components/drivers/usb/usbdevice/class/hid.c
+++ b/components/drivers/usb/usbdevice/class/hid.c
@@ -194,14 +194,14 @@ const rt_uint8_t _report_desc[]=
     COLLECTION(1),      0x01,
     REPORT_ID(1),       HID_REPORT_ID_GENERAL,

-    REPORT_COUNT(1),    RT_USB_DEVICE_HID_GENERAL_IN_REPORT_LENGTH,
+    REPORT_COUNT(2),    0xff, 0x03,
     USAGE(1),           0x03,
     REPORT_SIZE(1),     0x08,
     LOGICAL_MINIMUM(1), 0x00,
     LOGICAL_MAXIMUM(1), 0xFF,
     INPUT(1),           0x02,

-    REPORT_COUNT(1),    RT_USB_DEVICE_HID_GENERAL_OUT_REPORT_LENGTH,
+    REPORT_COUNT(2),    0xff, 0x03,
     USAGE(1),           0x04,
     REPORT_SIZE(1),     0x08,
     LOGICAL_MINIMUM(1), 0x00,
@@ -347,7 +347,7 @@ const static struct uhid_comm_descriptor _hid_comm_desc =
         USB_DESC_TYPE_ENDPOINT,
         USB_DYNAMIC | USB_DIR_IN,
         USB_EP_ATTR_INT,
-        0x40,
+        1024,
         0x01,
     },

@@ -357,7 +357,7 @@ const static struct uhid_comm_descriptor _hid_comm_desc =
         USB_DESC_TYPE_ENDPOINT,
         USB_DYNAMIC | USB_DIR_OUT,
         USB_EP_ATTR_INT,
-        0x40,
+        1024,
         0x01,
     },
 };
@@ -607,7 +607,7 @@ static rt_size_t _hid_write(rt_device_t dev, rt_off_t pos,
const void *buffer, r
         rt_memcpy((void *)report.report,(void *)buffer,size);
         report.size = size;
```

```
          hiddev->ep_in->request.buffer = (void *)&report;
-         hiddev->ep_in->request.size = (size+1) > 64 ? 64 : size+1;
+         hiddev->ep_in->request.size = (size+1) > MAX_REPORT_SIZE ?
MAX_REPORT_SIZE : size+1;
          hiddev->ep_in->request.req_type = UIO_REQUEST_WRITE;
          rt_usbd_io_request(hiddev->func->device, hiddev->ep_in, &hiddev->ep_in-
>request);
          return size;
diff --git a/components/drivers/usb/usbdevice/class/hid.h
b/components/drivers/usb/usbdevice/class/hid.h
index 4d0f314a1..ea9938220 100644
--- a/components/drivers/usb/usbdevice/class/hid.h
+++ b/components/drivers/usb/usbdevice/class/hid.h
@@ -34,8 +34,8 @@ extern "C" {
 #define USB_HID_REQ_SET_IDLE         0x0a
 #define USB_HID_REQ_SET_PROTOCOL     0x0b

-#define MAX_REPORT_SIZE              64
-#define HID_RX_BUFSIZE               64
+#define MAX_REPORT_SIZE              1024
+#define HID_RX_BUFSIZE               1024

 /* HID Report Types */
 #define HID_REPORT_INPUT             0x01
```

2.6.3 USB Device ADB 使用示例

RT-Thread 官方提供了 ADBD（Android Debug Bridge daemon）软件包，支持通过 USB 或者 TCPIP 与 PC 之间进行文件传输或者执行 shell 的通道。

USB ADB 功能基于 WINUSB 驱动实现数据传输，ADBD 应用程序通过操作设备节点 `/dev/winusb`，可以与 USB 控制器进行数据交互。

ADBD 软件包的说明，请参考 RT-Thread 官网文档中心：https://packages.rt-thread.org/detail.html?package=adbd

ADBD 软件包的工程下载地址：https://github.com/heyuanjie87/adbd

USB Device ADB 配置

配置 **USB** 组件

ADB 的 WINUSB 设备类驱动在 ADBD 软件包 `packages/adbd-latest/core/adbusb.c` 中实现，因此，在 USB 组件中选择使用 custom class。

`USB Product ID` 建议配置为 0x0006，保持与 Rockchip AP 芯片平台 ADB 功能的 PID 一致，避免手动安装 PC ADB 驱动。

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            -*- Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0006) USB Product ID
            [ ]   Enable composite device
                  Device type (Using custom class by register interface)  --->
                      (X) Using custom class by register interface
```

配置 **ADBD online packages**

```
[*] Enable RT-Thread online packages  --->
    RT-Thread online packages  --->
        tools packages  --->
            [*] ADBD: Android Debug Bridge daemon implementation in RT-Thread  --
->
                [ ]   Using TCPIP transfer
                [*]   Using USB transfer
                (1280) Set transfer thread stack size
                [*]   Enable Shell service
                [*]   Enable File service
                (2304)  Set file service thread stack size
                (2000)  Set file service receive timeout(ms)
                [ ]   Enable external MOD
                [ ]   Enable ADB service discover
                      Version (latest)  --->
```

Note.

`Using TCPIP transfer` 用于 TCPIP 传输方式，需要 disable，否则 ADBD 默认使用 TCPIP 传输方式，会影响 USB ADB 功能。

配置 **ADBD** 依赖的 **POSIX**

```
RT_USING_LEGACY [=y]
RT_USING_DFS [=y]
DFS_USING_POSIX [=y]
RT_USING_POSIX_FS [=y]
RT_USING_POSIX_DEVIO [=y]
RT_USING_POSIX_STDIO [=y]
RT_USING_POSIX_POLL [=y]
RT_USING_POSIX_SELECT [=y]
RT_USING_POSIX_SOCKET [=y]
```

配置 **ADB push/pull** 文件依赖的文件系统

启用文件系统的方法，请参考章节 USB Device MSC 使用示例 中关于 “USB Device MSC 文件系统自动挂载表” 的说明。

完成上述配置后，保存并退出配置界面。

下载 **ADBD online packages**

以 RK2118 SDK 为例：

```
~/rt-thread/bsp/rockchip/rk2118$ source ~/.env/env.sh
~/rt-thread/bsp/rockchip/rk2118$ pkgs --update
[Use Gitee server - auto decision based on IP location]
Cloning into '/home/wlf/rt-thread/bsp/rockchip/rk2118/packages/adbd-latest'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (39/39), done.
remote: Total 42 (delta 2), reused 37 (delta 2), pack-reused 0
Receiving objects: 100% (42/42), 10.13 MiB | 1.74 MiB/s, done.
Resolving deltas: 100% (2/2), done.
==============================>  ADBD latest is downloaded successfully.

/home/wlf/rt-thread/bsp/rockchip/rk2118/packages/adbd-latest
==============================>  adbd update done

Operation completed successfully.
```

修改 **ADBD package** 源码，以适配 **Rockchip** 平台

```
diff --git a/core/adbusb.c b/core/adbusb.c
index 29455d0..034e9bb 100644
--- a/core/adbusb.c
+++ b/core/adbusb.c
@@ -232,7 +232,7 @@ static rt_err_t _ep_out_handler(ufunction_t func, rt_size_t
size)
 static rt_err_t _ep_in_handler(ufunction_t func, rt_size_t size)
 {
     winusb_device_t wd = (winusb_device_t)func->user_data;
-    wd->wrcnt -= size;
+    wd->wrcnt -= wd->ep_in->request.size;
     rt_wqueue_wakeup(&(wd->wq), (void *)POLLOUT);

     return RT_EOK;
@@ -284,13 +284,12 @@ static rt_err_t _interface_handler(ufunction_t func, ureq_t
setup)
     return RT_EOK;
 }

-#ifdef RT_USING_POSIX
 static int _ep_alloc_request(uep_t ep)
 {
     int size;

     size = EP_MAXPACKET(ep);
-    ep->buffer = rt_malloc(size);
+    ep->buffer = rt_malloc_align(RT_ALIGN(size, USB_DMA_ALIGN_SIZE),
USB_DMA_ALIGN_SIZE);
     if (!ep->buffer)
         return -1;

@@ -300,7 +299,6 @@ static int _ep_alloc_request(uep_t ep)

     return 0;
```

```
 }
-#endif

 static rt_err_t _function_enable(ufunction_t func)
 {
@@ -316,7 +314,7 @@ static rt_err_t _function_enable(ufunction_t func)
         return -1;
     if (_ep_alloc_request(wd->ep_in) != 0)
     {
-        rt_free(wd->ep_out->buffer);
+        rt_free_align(wd->ep_out->buffer);
         return -1;
     }
     _readreq(func, wd);
@@ -331,8 +329,8 @@ static rt_err_t _function_disable(ufunction_t func)
     RT_ASSERT(func != RT_NULL);
     wd = func->user_data;

-    rt_free(wd->ep_out->buffer);
-    rt_free(wd->ep_in->buffer);
+    rt_free_align(wd->ep_out->buffer);
+    rt_free_align(wd->ep_in->buffer);
     wd->ep_out->buffer = 0;
     wd->ep_in->buffer = 0;

diff --git a/services/shell_service.c b/services/shell_service.c
index 6252942..80e6cb0 100644
--- a/services/shell_service.c
+++ b/services/shell_service.c
@@ -344,6 +344,7 @@ static bool send_ready(struct shell_ext *ext, char *args)

 extern int libc_stdio_set_console(const char* device_name, int mode);
 extern int libc_stdio_get_console(void);
+extern int ioctl(int fildes, int cmd, ...);

 static int _shell_open(struct adb_service *ser, char *args)
 {
```

执行 `list_device` 和 `mount` 命令，查看 `winusb` 节点是否正常创建以及文件系统是否正常挂载。

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
winusb   Miscellaneous Device 0
usbd     USB Slave Device     0
root     Block Device         1
snor     MTD Device           0
fspi0    Character Device     0
i2c0     I2C Bus              0
pin      Pin Device           0
uart0    Character Device     2

msh />mount
filesystem  device  mountpoint
----------  ------  ----------
```

```
devfs        (NULL)   /dev
elm          root     /
```

ADB 功能

上位机的 ADB 工具和驱动在 ADBD packages 的 `adbd-latest/tools` 目录下。

支持 adb push/pull 文件

支持 adb shell（进入 adb shell 命令行后，Console 将从物理串口切换到 USB）

2.6.4 USB Device MSC 使用示例

USB MSC（Mass Storage Class）的功能，可以将设备作为 U 盘使用。当设备通过 USB 连接 PC 后，PC 可以识别到一个可移动的存储设备，方便设备和 PC 进行文件的拷贝。

以 RT-Thread RK2108 MSC 的使用为例

USB Device MSC 配置

执行命令 `scons --menuconfig`，将USB 配置为 Mass Storage device。

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0007) USB Product ID
            [ ]   Enable composite device
                    Device type (Enable to use device as Mass Storage device)  --
->
            (root) msc class disk name
            (4096) msc buffer length
```

Note.

1. 需要手动修改 `msc class disk name` 为 `root`
2. 需要使能 `CONFIG_RT_USING_DFS_MNTTABLE`
3. `msc buffer length` 用于优化 USB 控制器的传输性能，值越大，通常传输性能越好。该值不能低于 512，且不能大于 65536。

USB Device MSC 文件系统自动挂载表

USB Device MSC 功能依赖于文件系统挂载表。如果没有配置文件系统自动挂载表，可能会出现能枚举到设备但是无法读写的情况。

目前 SDK 支持 Littlefs 和 FAT/EXFAT 文件系统，如果产品需要文件系统，可以在 `menuconfig` 里把相应的文件系统打开，并在板级目录下新建一个文件 `mnt.c`，用来配置文件系统自动挂载表，下面是一个挂载表的配置示例：

配置文件系统

```
RT_USING_DFS [=y] #文件系统总开关
DFS_FILESYSTEMS_MAX [=2] #最大允许的挂载节点数目（即分区数目），因为devfs要占用一个，所以这个值要>=（需要挂载的物理分区+1）
DFS_FILESYSTEM_TYPES_MAX [=2] #最大允许的文件系统数目，支持elmfat和devfs，所以是2
DFS_FD_MAX [=16] #最大可同时操作的文件句柄
RT_USING_DFS_MNTTABLE [=y] #启用自动挂载表
RT_USING_DFS_ELMFAT [=y] #启用elmfat文件系统，兼容fat文件系统
RT_USING_DFS_DEVFS [=y] #启用设备文件系统，可以通过文件接口访问设备驱动
RT_DFS_ELM_MAX_SECTOR_SIZE [=4096] # 设置扇区大小为4096，以兼容底层spi flash的擦除块大小
```

参考 RK2108 EVB：`bsp/rockchip/rk2108/board/rk2108d_evb/mnt.c`

```
#ifdef RT_USING_DFS_MNTTABLE
#include <dfs_fs.h>

#define PARTITION_ROOT          "root"

/******************** Private Variable Definition ***************************/
/** @defgroup MNT_Private_Variable Private Variable
 *  @{
 */

/**
 * @brief  Config mount table of filesystem
 * @attention The mount_table must be terminated with NULL, and the partition's name
 * must be the same as above.
 */
const struct dfs_mount_tbl mount_table[] =
{
    {PARTITION_ROOT, "/", "elm", 0, 0},
#if defined(RT_USING_LITTLEFS) && defined(RT_USING_SPINAND)
    {"spinand0", "/nand", "lfs", 0, 0},
#elif defined(RT_USING_SDIO0) && defined(RT_SDCARD_MOUNT_POINT)
    {"sd0", RT_SDCARD_MOUNT_POINT, "elm", 0, 0},
#endif
    {0}
};
#endif
```

`dfs_mount_tbl` 结构体说明

```
/* mount table */
struct dfs_mount_tbl
{
    const char   *device_name;    /* 设备节点名 */
    const char   *path;           /* 挂载点目录 */
    const char   *filesystemtype; /* 文件系统名字(elm对应fat) */
    unsigned long rwflag;         /* rw(1为读写，0为只读，elm不支持这个标识，填0) */
    const void   *data;           /* 文件系统私有数据(elm不支持，填0) */
};
```

USB Device MSC 的使用

开机后，如果设备没有通过 USB 连接到 PC，`root` 分区默认挂载到设备端的文件系统上。此时，设备端可以对 `root` 分区进行读写。

当设备通过 USB 连接到 PC 后，USB 驱动会自动将 `root` 分区从设备端的文件系统上卸载掉，此时，设备端无法对 `root` 分区进行读写，而 PC 端可以自动识别到U盘，并可以对U盘进行读写。

当设备断开 USB 连接后，USB 驱动会自动卸载 `root` 分区，并重新挂载到设备端的文件系统上。

2.6.5 USB Device UAC1 使用示例

UAC（USB Audio Class）功能，可以将设备做为 PC 或其他上位机的扩展声卡使用，RT-Thread SDK 仅支持 UAC V1.0 功能，不支持 UAC V2.0 功能。

USB Device UAC1 配置（RT-Thread v3.1.x）

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [ ] Using USB host
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0019) USB Product ID
            [ ]   Enable composite device
                    Device type (Enable to use device as UAC1 device)  --->
            (es8388p) uac1 playback card name
            (pdmc) uac1 capture card name
            [ ]   Enable to use pll compensation for clock synchronization
```

USB Device UAC1 配置（RT-Thread v4.1.x）

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [ ] Using USB host
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0019) USB Product ID
            [ ]   Enable composite device
                    Device type (Enable to use device as UAC1 device)  --->
            (es8388p) uac1 playback card name
            (pdmc) uac1 capture card name
            [ ]   Enable to poll ep first transfer status
```

USB Device UAC1 + HID 复合设备的配置（RT-Thread v4.1.x）

Note.

1. UAC1 功能开启时，必须关闭 VAD 功能，即设置 `# CONFIG_RT_USING_VAD is not set`；
2. `RT_USB_AUDIO_P_NAME` 对应实际的播放设备节点，需要根据实际的音频硬件设计进行修改。RK2108 SDK 参考设计为 `es8388p`；

3. `RT_USB_AUDIO_C_NAME` 对应实际的录音设备节点，需要根据实际的音频硬件设计进行修改。RK2108 SDK 参考设计为 `pdmc`；
4. `RT_USB_AUDIO_PLL_COMPENSATION` 用于使能 PLL 补偿，实现设备端的时钟动态调整。该功能用于设备放音场景，可以解决 USB 传输时钟（来自 USB HOST SOF）和 UAC1 设备端 I2S 时钟异源之间的时钟不同步，导致 UAC1 设备端出现 buffer overrun 或者 underrun 的问题。在使能该功能后，设备端能够主动调整自身数据处理的时钟，以适应 USB HOST 的数据发送率。这种方法的优点是，不依赖于 Windows/Linux/MAC OS 等USB 主机系统，具有更好的兼容性。缺点是：UAC1 设备端，需要根据不同 SoC 的 clock 设计方案，进行软件开发，实现时钟动态调整，有可能受限于芯片的 clock 设计。目前主要应用在 RK2108 芯片平台；
5. `RT_USB_AUDIO_USING_EP_POLL` 表示使用 HAL 层 USB 音频端点查询方式替代默认的中断方式，获取音频流启动传输的状态。如果产品上支持 UAC1 复合设备（如：UAC1 + HID），建议默认使能，可以避免端点中断风暴的潜在风险（与连接的 USB 主机行为有关），但同时，查询方式也会引入音频流延时的影响，SDK 默认设置查询间隔 `EP_STATUS_POLL_INTERVAL_MS` 为 5ms，如果产品对延时敏感，可以根据应用需求，修改查询间隔。
6. RT-Thread v3.1.x 的 UAC1 驱动默认不支持 USB 音频端点查询方式，可以通过更新驱动补丁的方式支持。

   补丁下载地址：https://redmine.rock-chips.com/attachments/1217535

系统启动后，如果找不到播放设备，PC 虽能能识别 UAC1 设备，但无法播放音频。而如果找不到录音设备，系统会出现 fault 出错。因此在打开 UAC1 功能之前，请使用 `list_device` 命令检查是否已经创建 Sound device。以 RK2108 SDK 为例，`es8311p` 对应音频播放设备，`pdmc` 对应音频录音设备。

如果未发现音频播放和录音设备，请在 menuconfig 中打开相应的 Audio 编译开关。

```
msh />list_device
device          type         ref count
------- -------------------- ----------
es8311c Sound Device         0
es8311p Sound Device         1
audpwmp Sound Device         0
pdmc    Sound Device         0
adcc    Sound Device         0
usbd    USB Slave Device     0
root    Block Device         1
snor    MTD Device           0
pwm0    Miscellaneous Device 0
dmac0   Miscellaneous Device 0
i2c0    I2C Bus              0
pin     Miscellaneous Device 0
uart2   Character Device     0
uart0   Character Device     2
```

2.6.6 USB Device UVC 使用示例

UVC（USB Video Class）功能，可以将设备做为 USB 视频采集设备使用，Rockchip RT-Thread UVC 基于 UVC 1.1 标准协议实现，支持 MJPEG 和 YUYV 两种格式视频数据，目前最多支持三路 UVC 复合设备。

UVC 使用限制：

1. 仅支持 BULK 传输方式，不支持 ISOC 传输方式；
2. 仅支持 RK625 芯片平台。

USB Device UVC 配置

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0007) USB Product ID
            [*]   Enable composite device
            [*]     Enable to use device as UVC device
            [*]   Use to UVC device as RGB
            [ ]   Use to UVC device as DEPTH
            [ ]   Use to UVC device as IR
            (10240) uvc device max buffer length
```

Note.

1. RGB，DEPTH，IR 需要根据具体 SoC 的硬件支持情况进行配置。如：RK625 SoC 可支持 RGB UVC，RK625 + RK1608 方案可支持三路 UVC；
2. `uvc device max buffer length` 可支持 4K-16K 范围配置。

UVC 设计简述

UVC 组件基于 RT-Thread USB Framework 实现，主要包括 UVC 线程、UVC 驱动、Buffer 管理和异常处理等子模块，其系统框图如下所示。

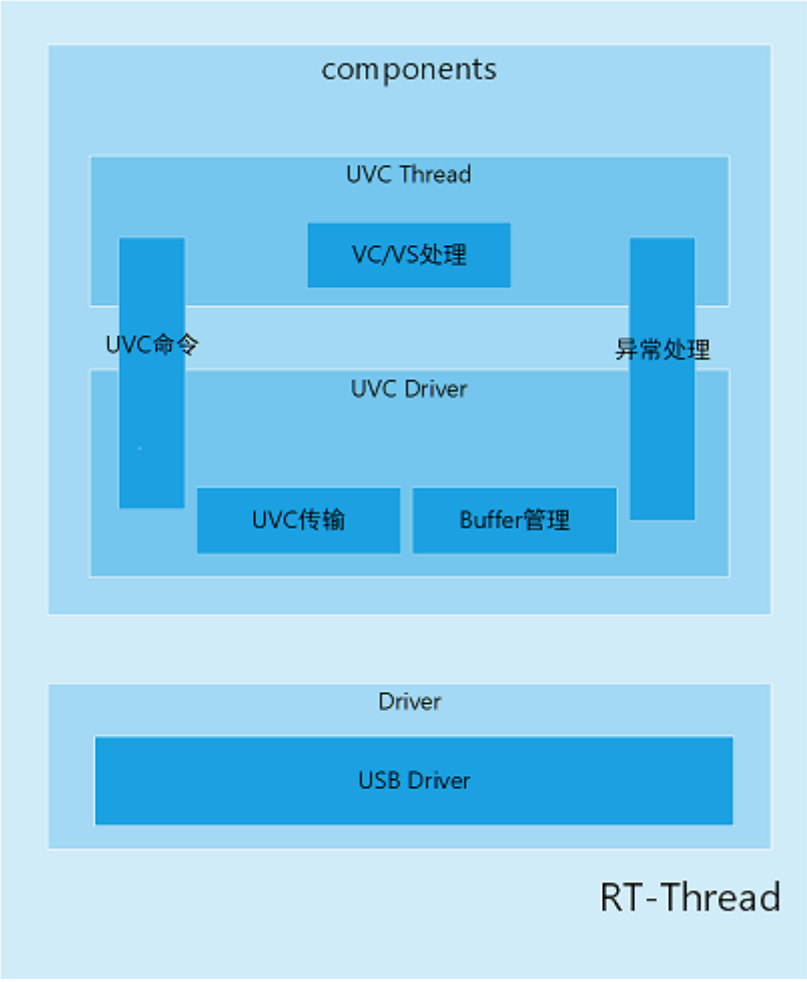

图 6 UVC 组件框图

- UVC Thread 主要实现 UVC VC/VS 请求响应，每路 UVC 有单独一个处理线程；
- UVC Driver 实现 UVC 设备描述符定义（包括图像分辨率定义）、UVC Buffer 状态管理、UVC 视频数据传输调度和异常处理等功能；
- UVC Thread 和 UVC Driver 通过 RT-Thread OS Event 通信；UVC Driver 和 USB Driver 通过 USB IO 接口通信。

UVC 日志解析

将 UVC 的设备接入上位机，并在上位机上使用视频预览软件打开 Camera，Rockchip SoC 串口在上电 Camera 初始化后，将有如下 UVC 相关 Log 输出：

```
[...1...]
INF: uvc(1) set format fcc 1196444237 width 640 height 480
VIDEO_OPEN(1)
INF: uvc(1) open camera device...
Open camera id width height 1 640 480
[...2...]
INF: uvc(1) buffer(0) addr 0x2006787c
INF: uvc(1) buffer(1) addr 0x2006987c
INF: uvc(1) buffer(2) addr 0x2006b87c
INF: uvc(1) buffer(3) addr 0x2006d87c
INF: uvc(1) mjpeg header addr 0x20087c00
INF: uvc(1) buffer size 8064
[...]
[...3...]
VIDEO_CLOSE(1)
INF: uvc(1) close camera device...
Close camera id 1
close rgb success
[...]
[...4...]
uvc(1) buffer: (0-1) (1-1) (2-1) (3-2)
ERR: uvc(1) buffer overflow irq 0x000100f0
```

- Log 1 指示被打开 Camera 的 FCC 及宽高等信息；
- Log 2 指示 UVC 环形 Buffer 的使用个数，地址和每个 Buffer 的大小；如果是 MJPEG 格式的数据，还会打印 MJPEG Header Buffer 信息；
- 在 Log 1 和 Log 2 成功输出后，如果 Sensor 有送数据出来，UVC 将会调度 Buffer 管理并传送数据给上位机；
- Log 3 指示上位机关闭 Camera 时的 Log；
- Log 4 指示诸如 AMCAP，GUVCView 等常用的视频预览软件在关闭预览或退出时不会发送 Close Camera 命令，只是向 USB 发送一条 EP Halt 命令。EP Halt 后，会触发 UVC Buffer Overflow，Buffer 管理器继而关闭 Buffer通道，停止 USB 数据传输。

2.6.7 USB Device WINUSB 使用示例

WINUSB 是 Windows 操作系统提供的一种通用 USB 驱动程序，无需安装驱动即可实现 Bulk 数据传输。

USB Device WINUSB 配置

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [ ] Using USB host
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0007) USB Product ID
            [ ]   Enable composite device
                    Device type (Enable to use device as winusb device)  --->
            ({6860DC3C-C05F-4807-8807-1CA861CC1D66}) Guid for winusb
            (16384) winusb transfer length
```

执行命令 `list_device` 可以看到已经生成的 USB WINUSB 设备：

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
winusb   Miscellaneous Device 0
usbd     USB Slave Device     0
```

针对 WINUSB 功能，Rockchip 提供了一个上位机的 Demo，可以用于 WINUSB 功能测试。

下载路径：https://redmine.rock-chips.com/attachments/1213728

2.6.8 USB Device CDC_VCOM 使用示例

CDC_VCOM 支持 USB 作为虚拟串口与上位机进行数据收发，基于 CDC-ACM 协议实现驱动。

USB Device CDC_VCOM 配置

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
            [ ] Using USB host
            [*] Using USB device
            (4096) usb thread stack size
            (0x2207) USB Vendor ID
            (0x0007) USB Product ID
            [ ]   Enable composite device
                    Device type (Enable to use device as CDC device)  --->
            (512) virtual com thread stack size
            (128) virtual com rx buffer size
            [ ]   Enable to use dma for vcom tx
            (32021919830108) serial number of virtual co
            (14)  serial number length of virtual com
            (1000) tx timeout(ticks) of virtual com
```

RT-Thread 3.1.x 实现 USB system console 的修改方法

```
diff --git a/components/libc/compilers/newlib/libc.c
b/components/libc/compilers/newlib/libc.c
```

```
index 2234c36f98..07ba2bafeb 100644
--- a/components/libc/compilers/newlib/libc.c
+++ b/components/libc/compilers/newlib/libc.c
@@ -23,6 +23,7 @@ int    _EXFUN(putenv,(char *__string));

 int libc_system_init(void)
 {
+    rt_console_set_device("vcom");
 #if defined(RT_USING_DFS) & defined(RT_USING_DFS_DEVFS) &
defined(RT_USING_CONSOLE)
     rt_device_t dev_console;

@@ -47,4 +48,4 @@ int libc_system_init(void)

     return 0;
 }
-INIT_COMPONENT_EXPORT(libc_system_init);
+INIT_APP_EXPORT(libc_system_init);
```

RT-Thread 4.1.x 实现 USB system console 的修改方法

依赖 POSIX 配置

```
RT_USING_POSIX_FS [=y]
RT_USING_POSIX_DEVIO [=y]
RT_USING_POSIX_STDIO [=y]
```

```
diff --git a/components/libc/posix/io/stdio/libc.c
b/components/libc/posix/io/stdio/libc.c
index cc891574f3..b12e24610b 100644
--- a/components/libc/posix/io/stdio/libc.c
+++ b/components/libc/posix/io/stdio/libc.c
@@ -20,6 +20,7 @@

 int libc_system_init(void)
 {
+    rt_console_set_device("vcom");
 #ifdef RT_USING_POSIX_STDIO
     rt_device_t dev_console;

@@ -31,7 +32,8 @@ int libc_system_init(void)
 #endif /* RT_USING_POSIX_STDIO */
     return 0;
 }
-INIT_COMPONENT_EXPORT(libc_system_init);
+INIT_APP_EXPORT(libc_system_init);
```

CDC_VCOM 兼容性问题处理

问题描述： 通过 USB 虚拟串口和上位机进行通信，Rockchip Device 端可以接收上位机发送的串口消息，但是 Device 端无法正常发送串口消息给上位机。

解决方法：

```
Subject: [PATCH] [COMPONENTS]: drivers: usb: cdc_vcom: Fix connected state
```

```
The current cdc vcom driver depends the setup wValue of
the request CDC_SET_CONTROL_LINE_STATE to set the connected
state. Refer to CDC Spec V1.2 Chapter 6.3.12 SetControlLineState,
the Bit0 of the wValue is used to indicates to DCE if DTE
(Data Terminal Equipment, for example, a Personal Computer)
is present or not. However, some DTE always set the Bit0 of
the wValue to 0, this cause cdc vcom failed to send message to
DTE. This patch ignores the Bit0 of wValue, and set the
connected state to true directly if it receives the request
CDC_SET_CONTROL_LINE_STATE.

diff --git a/components/drivers/usb/usbdevice/class/cdc_vcom.c
b/components/drivers/usb/usbdevice/class/cdc_vcom.c
index df5e03f54c..e201742747 100644
--- a/components/drivers/usb/usbdevice/class/cdc_vcom.c
+++ b/components/drivers/usb/usbdevice/class/cdc_vcom.c
@@ -463,7 +463,7 @@ static rt_err_t _interface_handler(ufunction_t func, ureq_t
setup)
         _cdc_get_line_coding(func->device, setup);
         break;
     case CDC_SET_CONTROL_LINE_STATE:
-        data->connected = (setup->wValue & 0x01) > 0?RT_TRUE:RT_FALSE;
+        data->connected = RT_TRUE;
         RT_DEBUG_LOG(RT_DEBUG_USB, ("vcom state:%d \n", data->connected));
         dcd_ep0_send_status(func->device->dcd);
         break;
```

2.6.9 USB Device speed 配置说明

USB Device 默认支持 High speed，如果产品上要求设备强制工作在 Full speed（如：UAC 产品形态），则可以配置

`#define RT_USING_USBD_SPEED_FULL`

2.6.10 USB Device 端点 FIFO 配置说明

USB Device 端点 FIFO 是指 USB 控制器内部分配给各个端点的硬件 FIFO，所有 Rx 端点复用一个 RxFIFO，而每个 Tx 端点独占一个 TxFIFO。具体代码实现在 USB Device DWC2 控制器驱动适配层驱动以及板级配置中。

以 RK2118 SDK 为例：

USB 端点的板级默认配置在 `rt-thread/bsp/rockchip/rk2118/board/common/board_base.c`，如下所示，g_usb_ep_pool 按照大部分产品形态的 USB 功能进行设置，可同时支持 2 个 BULK 端点，2 个 ISOC 端点，2 个 INT 端点。如果产品上要支持其它 USB 复合设备，则需要根据实际所需的 USB 端点数量和类型，进行配置。

```
#ifdef RT_USING_USB_DEVICE
RT_WEAK struct ep_id g_usb_ep_pool[] =
{
    { 0x0,  USB_EP_ATTR_CONTROL,    USB_DIR_INOUT,  64,   ID_ASSIGNED   },
    { 0x1,  USB_EP_ATTR_BULK,       USB_DIR_IN,     1024, ID_UNASSIGNED },
    { 0x2,  USB_EP_ATTR_BULK,       USB_DIR_OUT,    512,  ID_UNASSIGNED },
    { 0x3,  USB_EP_ATTR_ISOC,       USB_DIR_IN,     1024, ID_UNASSIGNED },
    { 0x4,  USB_EP_ATTR_ISOC,       USB_DIR_OUT,    512,  ID_UNASSIGNED },
    { 0x5,  USB_EP_ATTR_INT,        USB_DIR_IN,     64,   ID_UNASSIGNED },
    { 0x6,  USB_EP_ATTR_INT,        USB_DIR_OUT,    64,   ID_UNASSIGNED },
    { 0xFF, USB_EP_ATTR_TYPE_MASK,  USB_DIR_MASK,   0,    ID_ASSIGNED   },
};
#endif
```

USB HAL 的 FIFO 设置在`rt-thread/bsp/rockchip/common/drivers/drv_usbd.c`

调用 `HAL_PCDEx_SetRxFiFo` 设置 RxFIFO，单位 4 字节

调用 `HAL_PCDEx_SetTxFiFo` 设置 TxFIFO，单位 4 字节

```
static void usb_pcd_lowlevel_hw_init(struct PCD_HANDLE *pcd)
{
    uint8_t i;

    /* Initialize Low Level Driver */
    HAL_PCD_Init(pcd);

#ifndef RT_USB_DEVICE_UVC
    HAL_PCDEx_SetRxFiFo(pcd, 0x118);
#else
    /*
     * Set smaller RxFiFo for 3 * UVC interfaces, then
     * it can assign enough TxFiFo size (1024 Bytes)
     * for each UVC streaming IN endpoint to get higher
     * performance.
     */
    HAL_PCDEx_SetRxFiFo(pcd, 0x8C);
#endif

    /* Set TxFIFOs according to the g_usb_ep_pool array */
    for (i = 0; i < pcd->cfg.epNum + 1; i++)
    {
        if (g_usb_ep_pool[i].addr == 0xFF)
            break;

        if (g_usb_ep_pool[i].dir == USB_DIR_IN || g_usb_ep_pool[i].dir ==
USB_DIR_INOUT)
            HAL_PCDEx_SetTxFiFo(pcd, g_usb_ep_pool[i].addr,
                                (g_usb_ep_pool[i].maxpacket >> 2));
    }

    HAL_PCD_Start(pcd);
}
```

2.6.11 USB Device VBUS Detect 使用说明

支持三种不同的 VBUS Detect 方式

- GPIO VBUS Detect：通过 GPIO 检测 VBUS 电平

  对应宏定义 `USB_VBUS_PIN`

  参考 RK2118 evb 软硬件设计：`rt-thread/bsp/rockchip/rk2118/board/evb/board.h`
- PMIC VBUS Detect：通过 PMIC 检测 VBUS 电平

  对应宏定义 `RT_USING_USB_PMIC_VBUS`

  参考 RK2108 recording_pen 软硬件设计：`rt-thread/bsp/rockchip/rk2108/board/recording_pen_v10/board.h`
- USB PHY VBUS Detect：通过 USB PHY 检测 VBUS 电平

  对应宏定义 `RT_USING_BVALID_IRQ`

  参考 RK2118 iotest 软硬件设计：`rt-thread/bsp/rockchip/rk2118/board/iotest/board.h`

2.7 USB Host 使用示例

2.7.1 USB Host MSC 使用示例

USB Host MSC 组件配置

```
RT-Thread Components  --->
    Device Drivers  --->
        Using USB  --->
        [*] Using USB host
        [*]   Enable Udisk Drivers
        (/)     Udisk mount dir (NEW)
        [ ]   Enable HID Drivers (NEW)
        [ ] Using USB device
        (4096) usb thread stack size
```

USB Host 驱动适配层配置

需要根据芯片平台实际支持的 USB 控制器型号和 RT-Thread 软件版本进行配置。

RT-Thread v3.1.x

```
RT-Thread rockchip common drivers  --->
    RT-Thread rockchip USB Host driver  --->
        [ ] DWC2 HCD Support
        [ ] EHCI HCD Support
        [ ] OHCI HCD Support
```

Note.

1. `DWC2 HCD Support` 对应 DWC2 控制器驱动适配层；
2. `EHCI HCD Support` 和 `OHCI HCD Support` 对应 USB2 Host EHCI & OHCI 控制器驱动适配层；

RT-Thread v4.1.x

为了在不同芯片平台上支持多 USB Host 接口同时工作的功能，从 RT-Thread v4.1.x SDK 开始，USB Host 驱动适配层的配置从平台通用驱动移到芯片驱动中。以 RK3506 芯片平台同时支持 OTG0/1 两个 Host 接口为例，配置如下：

```
RT-Thread rockchip RK3506 drivers  --->
    Enable USB HOST  --->
        [*] Enable USB OTG HOST
        [*]   Enable USB OTG0 HOST
        [*]   Enable USB OTG1 HOST
```

MSC 功能依赖的文件系统相关配置

```
RT-Thread Components  --->
    [*] DFS: device virtual file system  ----
        [*]   Enable elm-chan fatfs
            elm-chan's FatFs, Generic FAT Filesystem Module  --->
```

Note.

目前 RT-Thread 只支持 FAT-32 格式的 U 盘。

MSC 功能依赖的 **cache** 配置

```
RT-Thread rockchip common drivers  --->
    [*] Enable cache
```

USB Host MSC 注册和枚举流程

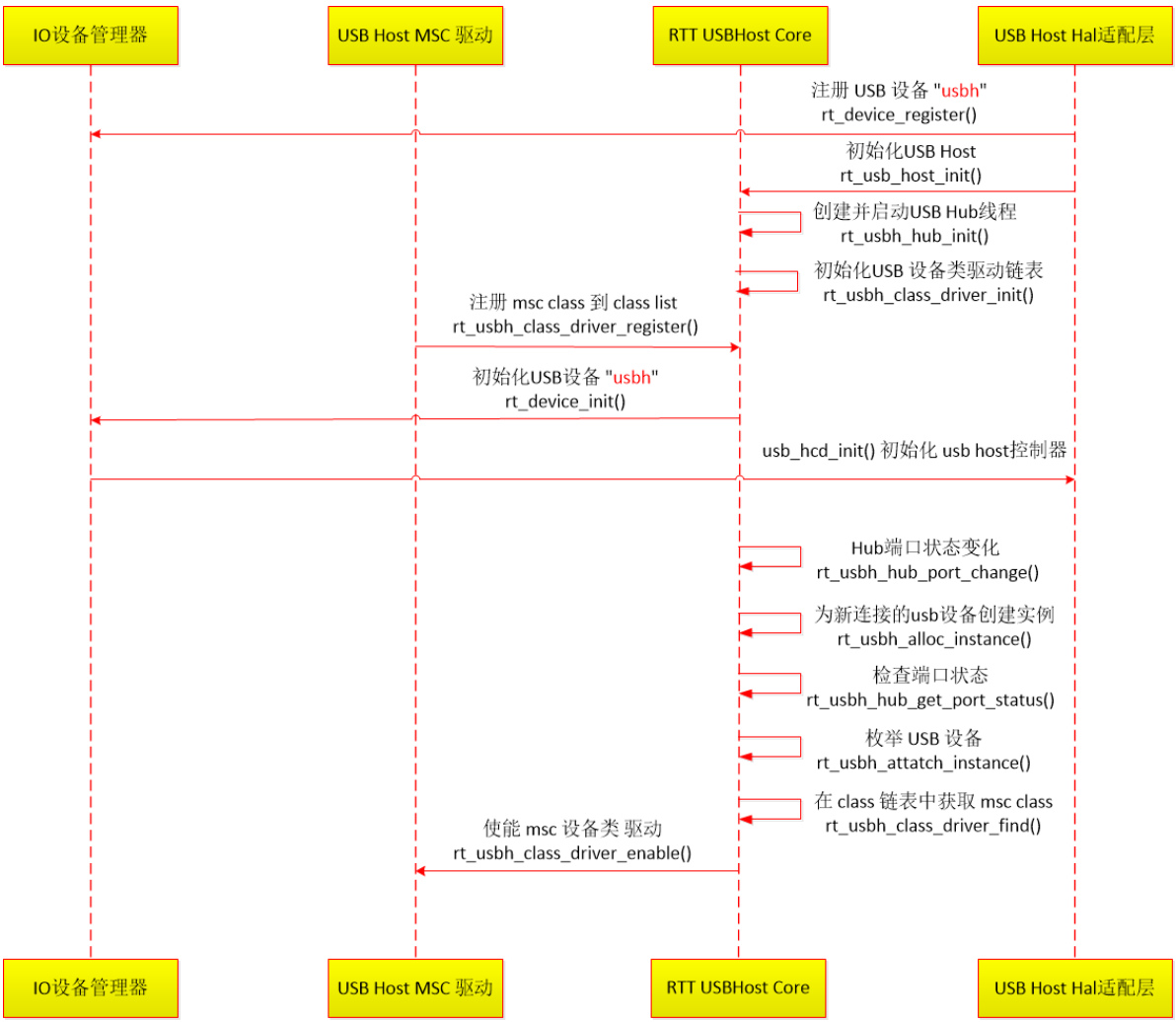

图 7 USB Host MSC 注册和枚举流程

应用程序访问**MSC**设备流程

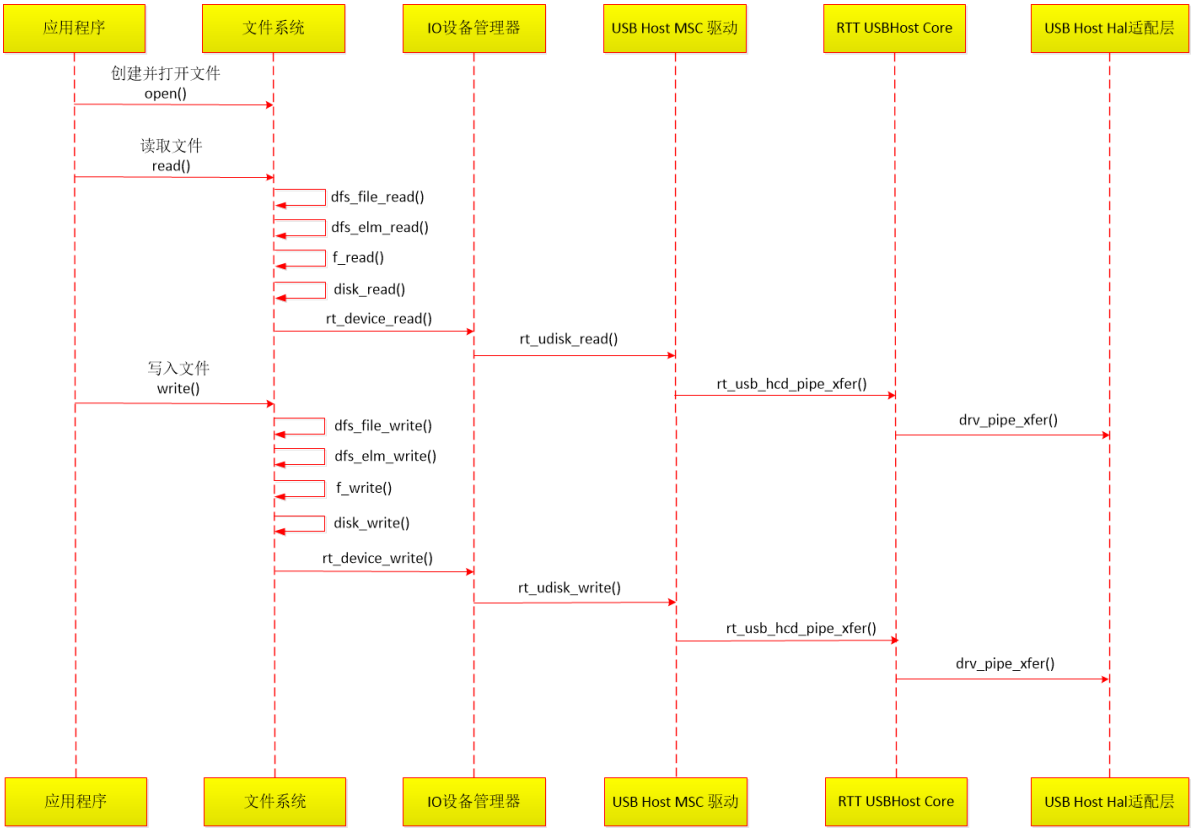

图 8 MSC 应用程序访问设备的流程

2.7.2 USB Host VBUS 控制说明

支持通过 GPIO 控制 VBUS 输出 5V 的方式，在对应的板级头文件中添加 VBUS 对应的 GPIO BANK_PIN 宏定义即可。

1. USB2.0 OTG Host VBUS GPIO

   以 RK3506 EVB1 为例，VBUS 宏定义在 `bsp/rockchip/rk3506-32/board/evb1/board.h`

   ```
   #define USB_OTG_HOST0_VBUS_PIN        BANK_PIN(GPIO_BANK1, 20)
   #define USB_OTG_HOST1_VBUS_PIN        BANK_PIN(GPIO_BANK1, 24)
   ```

2. USB2.0 Host VBUS GPIO

   以 RK3568 EVB1 为例，VBUS 宏定义在 `bsp/rockchip/rk3568-32/board/rk3568_evb1/board.h`

   ```
   #define USB_HOST_VBUS_PIN        BANK_PIN(GPIO_BANK0, 6)
   ```