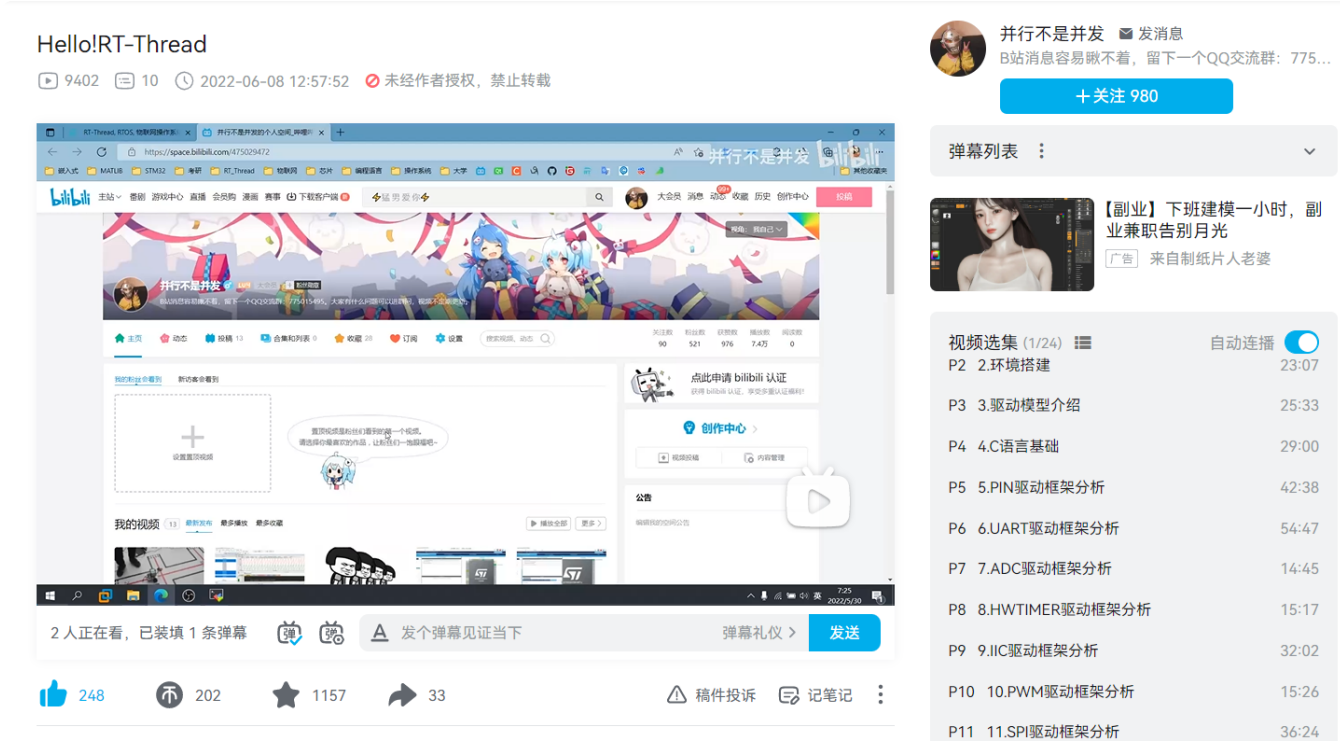
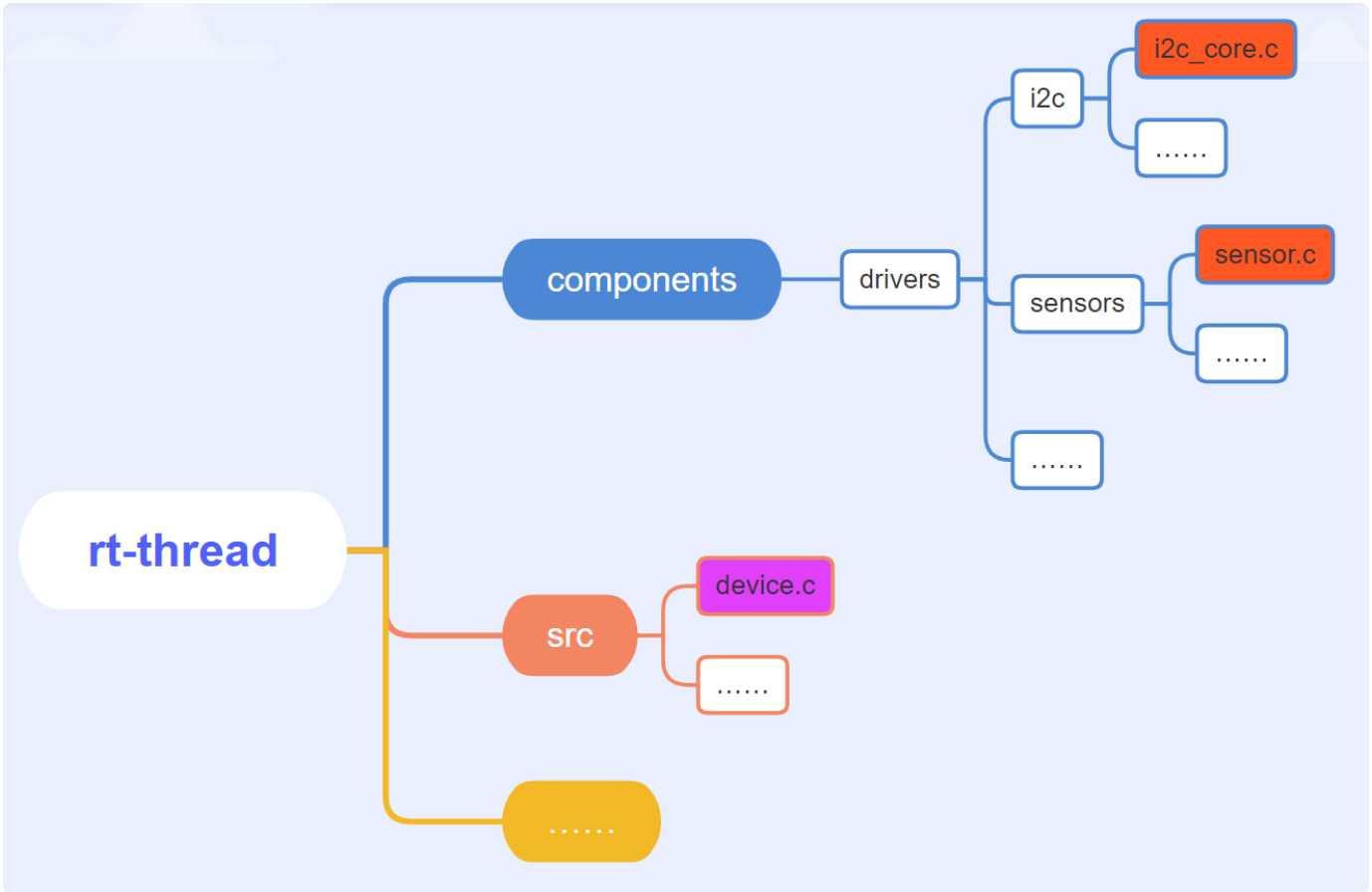
I/O设备模型

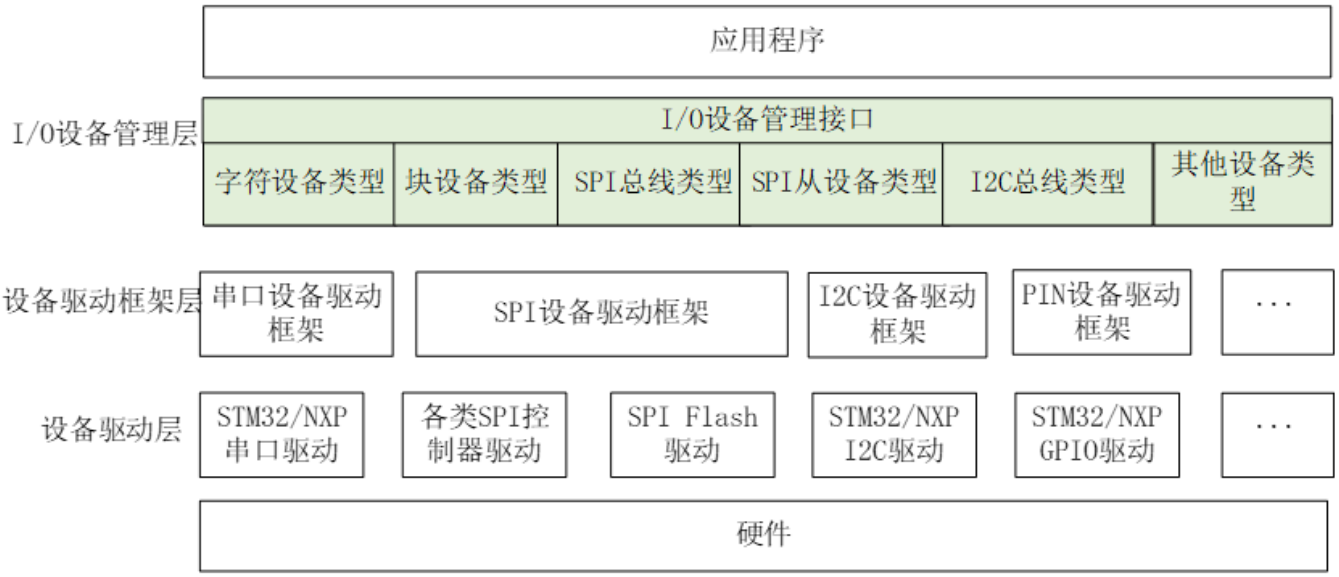
框架视频： 《hello!RT-Thread》


Hello!RT-Thread
9402
10
2022-06-08 12:57:52
未经作者授权，禁止转载
并行不是并发
发消息
B站消息容易瞅不着，留下一个QQ交流群：775...
+关注 980
弹幕列表
【副业】下班建模一小时，副业兼职告别月光
广告
来自制纸片人老婆
视频选集 (1/24)
自动连播
P2 2.环境搭建 23:07
P3 3.驱动模型介绍 25:33
P4 4.C语言基础 29:00
P5 5.PIN驱动框架分析 42:38
P6 6.UART驱动框架分析 54:47
P7 7.ADC驱动框架分析 14:45
P8 8.HWTIMER驱动框架分析 15:17
P9 9.IIC驱动框架分析 32:02
P10 10.PWM驱动框架分析 15:26
P11 11.SPI驱动框架分析 36:24
2 人正在看，已装填 1 条弹幕
发个弹幕见证当下
弹幕礼仪
发送
248
202
1157
33
稿件投诉
记笔记

I/O设备源码结构


rt-thread
components
drivers
i2c
i2c_core.c
.....
sensors
sensor.c
.....
.....
src
device.c
.....
.....

IO设备框架



- I/O 设备管理层实现了对 **设备驱动程序的封装** 。应用程序通过 I/O 设备层提供的标准接口访问底层设备， **设备驱动程序的升级、更替不会对上层应用产生影响** 。这种方式使得设备的硬件操作相关的代码能够独立于应用程序而存在，双方只需关注各自的功能实现，从而降低了代码的 **耦合性、复杂性** ，提高了系统的 **可靠性** 。
- 设备驱动框架层是对 **同类硬件设备驱动** 的抽象，将不同厂家的同类硬件设备驱动中 **相同的部分抽取出来** ，将 **不同部分留出接口** ，由 **驱动程序** 实现。
- 设备驱动层是一组 **驱使硬件设备工作的程序** ，实现 **访问硬件设备** 的功能。它负责 **创建和注册 I/O 设备** 。

> 1、应用程序可以不经过设备框架直接去调用硬件层嘛？？？优缺点？？
>
> 2、如果我们更换一款硬件，哪些部分需要做出相应调整？？
>
> 3、设备框架层的相同部分？？不同部分又是什么意思？？

RT-Thread 如何管理设备

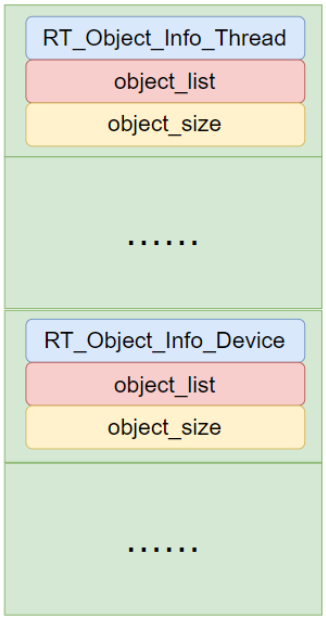
rtdef.h #449 C 复制代码

```c
/**
 * The information of the kernel object
 */
struct rt_object_information
{
    enum rt_object_class_type type;                     /**< object class type */
    rt_list_t                 object_list;              /**< object list */
    rt_size_t                 object_size;              /**< object size */
};
```

内核会创建一大堆对象

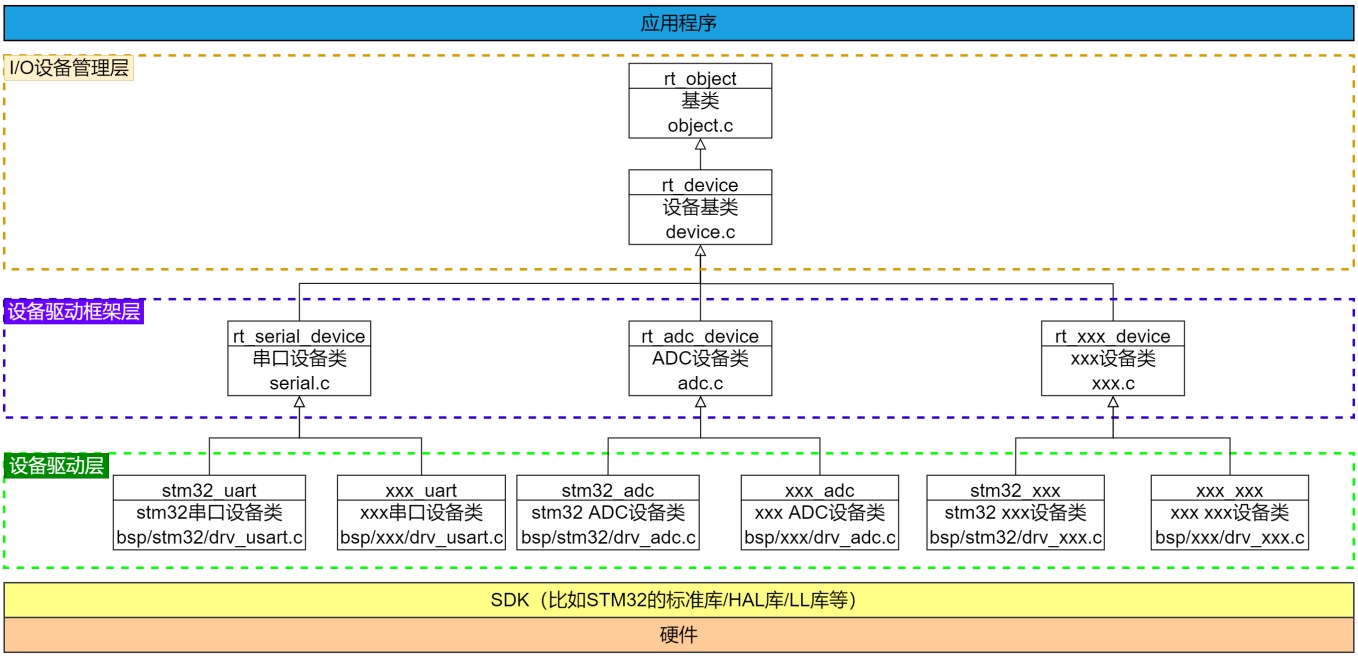
object.c #69 C 复制代码

```c
static struct rt_object_information _object_container[RT_Object_Info_Unknown]
```



接下来以PIN设备举例

PIN设备



面向对象的思维很重要

```c
struct rt_device_pin
{
    struct rt_device parent;
    const struct rt_pin_ops *ops;
};
```

```c
struct rt_device
{
    struct rt_object          parent;

    enum rt_device_class_type type;
    rt_uint16_t               flag;
    rt_uint16_t               open_flag;

    rt_uint8_t                ref_count;
    rt_uint8_t                device_id;

    /* device call back */
    rt_err_t (*rx_indicate)(rt_device_t dev, rt
    rt_err_t (*tx_complete)(rt_device_t dev, vo
```

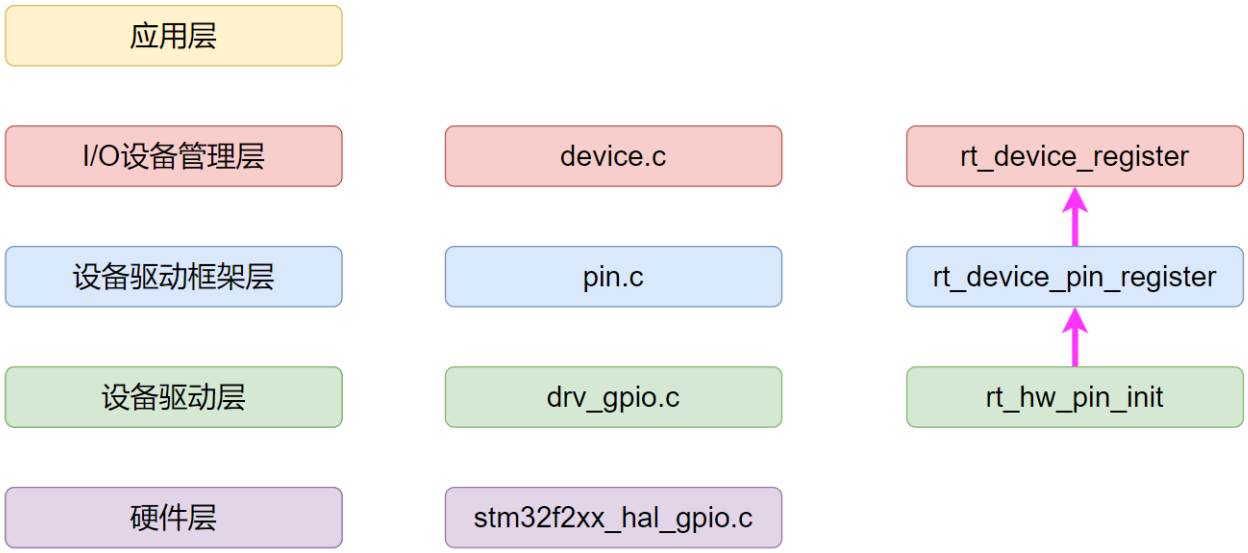
```c
struct rt_object
{
    char       name[RT_NAME_MAX];
    rt_uint8_t type;
    rt_uint8_t flag;

#ifdef RT_USING_MODULE
    void      *module_id;
#endif /* RT_USING_MODULE */
    rt_list_t  list;
};
typedef struct rt_object *rt_object_t;
```

<初始化结构体 + 使用结构体>

结构体初始化： 设备的注册

使用结构体： 框架API调用

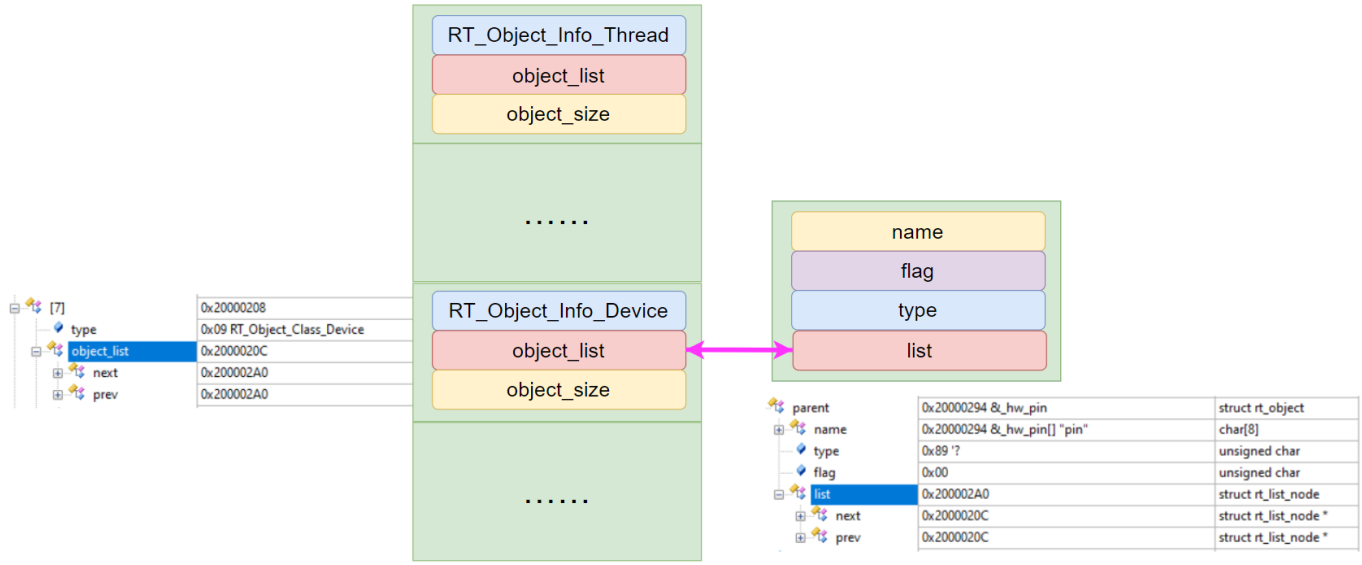
设备是如何注册到设备框架的



rt_hw_pin_init函数

```c
rt_hw_pin_init
    ->rt_device_pin_register("pin", &_stm32_pin_ops, RT_NULL);
        ->rt_device_register(&_hw_pin.parent, name, RT_DEVICE_FLAG_RDWR);
            ->rt_object_init(&(dev->parent), RT_Object_Class_Device, name);
                ->rt_list_insert_after(&(information->object_list), &(object->list));
```

_hw_pin	0x20000294 &_hw_pin	struct rt_device_pin
parent	0x20000294 &_hw_pin	struct rt_device
parent	0x20000294 &_hw_pin	struct rt_object
name	0x20000294 &_hw_pin[] "pin"	char[8]
type	0x89 '?'	unsigned char
flag	0x00	unsigned char
list	0x200002A0	struct rt_list_node
type	0x19 RT_Device_Class_Pin	enum (rt_device_class...
flag	0x0000	unsigned short
open_flag	0x0000	unsigned short
ref_count	0x00	unsigned char
device_id	0x00	unsigned char
rx_indicate	0x00000000	long f() *
tx_complete	0x00000000	long f() *
init	0x00000000	long f() *
open	0x00000000	long f() *
close	0x00000000	long f() *
read	0x08003F79 _pin_read	unsigned int f() *
write	0x08003FCD _pin_write	unsigned int f() *
control	0x08003F31 _pin_control	long f() *
user_data	0x00000000	void *
ops	0x0800C744 &_stm32_pin_ops	struct rt_pin_ops *
_stm32_pin_ops	0x0800C744 &_stm32_pin_ops	struct rt_pin_ops
_pin_read	0x08003F78	unsigned int f(struct rt...
_pin_write	0x08003FCC	unsigned int f(struct rt...
_pin_control	0x08003F30	long f(struct rt_device ...



设备是如何调用的

```
    /* device call back */
    rt_err_t (*rx_indicate)(rt_device_t dev, rt_size_t size);
    rt_err_t (*tx_complete)(rt_device_t dev, void *buffer);

#ifdef RT_USING_DEVICE_OPS
    const struct rt_device_ops *ops;                    struct rt_device
#else
    /* common device interface */
    rt_err_t  (*init)   (rt_device_t dev);
    rt_err_t  (*open)   (rt_device_t dev, rt_uint16_t oflag);
    rt_err_t  (*close)  (rt_device_t dev);
    rt_size_t (*read)   (rt_device_t dev, rt_off_t pos, void *buffer, rt_size_t size);
    rt_size_t (*write)  (rt_device_t dev, rt_off_t pos, const void *buffer, rt_size_t size
    rt_err_t  (*control)(rt_device_t dev, int cmd, void *args);
#endif /* RT_USING_DEVICE_OPS */
```

rt_device_find函数

C | 复制代码

```
1   rt_device_find
2       ->rt_object_find(name, RT_Object_Class_Device);
```

怎么实现？？？

这里不是找的object？？？

rt_device_init函数

```c
rt_device_init(rt_device_t dev)
    ->(dev->init)
```

rt_device_open函数

```c
rt_device_open(rt_device_t dev, rt_uint16_t oflag)
    ->(dev->init)//如果没有初始化
    ->(dev->open)
```

rt_device_close函数

```c
rt_device_close(rt_device_t dev)
    ->(dev->close)
```

rt_device_read函数

```c
rt_device_read
    ->(dev->read)
```

……

```
#ifdef RT_USING_DEVICE_OPS
    _hw_pin.parent.ops          = &pin_ops;
#else
    _hw_pin.parent.init         = RT_NULL;
    _hw_pin.parent.open         = RT_NULL;
    _hw_pin.parent.close        = RT_NULL;
    _hw_pin.parent.read         = _pin_read;
    _hw_pin.parent.write        = _pin_write;
    _hw_pin.parent.control      = _pin_control;
#endif
```

rt_device_pin_register()

rt_pin_mode函数

C | 复制代码

```
1    rt_pin_mode
2        ->ops->pin_mode
```

```
struct rt_device_pin
{
    struct rt_device parent;
    const struct rt_pin_ops *ops;
};
```

```
struct rt_pin_ops
{
    void (*pin_mode)(struct rt_device *device, rt_base_t pin, rt_base_t mode);
    void (*pin_write)(struct rt_device *device, rt_base_t pin, rt_base_t value);
    int (*pin_read)(struct rt_device *device, rt_base_t pin);
    rt_err_t (*pin_attach_irq)(struct rt_device *device, rt_int32_t pin,
                               rt_uint32_t mode, void (*hdr)(void *args), void *args);
    rt_err_t (*pin_detach_irq)(struct rt_device *device, rt_int32_t pin);
    rt_err_t (*pin_irq_enable)(struct rt_device *device, rt_base_t pin, rt_uint32_t enabled
    rt_base_t (*pin_get)(const char *name);
};
```

```
const static struct rt_pin_ops _stm32_pin_ops =
{
    stm32_pin_mode,
    stm32_pin_write,
    stm32_pin_read,
    stm32_pin_attach_irq,
    stm32_pin_dettach_irq,
    stm32_pin_irq_enable,
    stm32_pin_get,
};

    return rt_device_pin_register("pin", &_stm32_pin_ops, RT_NULL);
}
```

rt_hw_pin_init()

如何去实现设备框架

为 **上层** 提供统一的 **操作函数** 。

为 **下层** 不同部分留出 **接口** 。

为什么要分析源码

一起为RT-Thread设计LCD驱动框架

书籍推荐


RT-Thread
中国自主物联网操作系统
RTOS
RT-Thread内核实现
与应用开发实战指南
基于STM32
刘火良 杨森 编著


公益
宝贝
RT-Thread
嵌入式实时操作系统
RT-Thread设计与实现
INTERNET
OF THINGS
嵌入式实时操作系统
RT-Thread设计与实现