**RTT IO设备驱动简介**

要想将某个设备纳入到RTT的IO设备层中，需要为这个设备创建一个名为rt\_device的数据结构。该数据结构在rtdef.h中定义。

/\*\*

\* Device structure

\*/

struct rt\_device

{

struct rt\_object parent; /\*\*< inherit from rt\_object \*/

enum rt\_device\_class\_type type; /\*\*< device type \*/

rt\_uint16\_t flag; /\*\*< device flag \*/

rt\_uint16\_t open\_flag; /\*\*< device open flag \*/

rt\_uint8\_t ref\_count; /\*\*< reference count \*/

rt\_uint8\_t device\_id; /\*\*< 0 - 255 \*/

/\* 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);

/\* 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, rt\_uint8\_t cmd, void \*args);

void \*user\_data; /\*\*< device private data \*/

};

对这个数据结构做一些详细的说明。

* struct rt\_object parent;这个域是RTT的所谓的面向对象设计，跟我们关系不大。
* type域配合前面的parent域，来制定设备的类型，也与我们关系不大。
* flag和openflag用来存储设备的权限，比如是只读，还是读写等等。
* device\_id即设备号，每一个设备都拥有唯一的编号，内核可以根据这个编号查找到设备。

接下来就是定义了一组函数指针，用于操作这个设备的一些回调（callback）函数。他们分别是：

rx\_indicate

tx\_complete

init

open

close

read

write

control

以及一个指针变量，由用户根据实际需要填充

void \*user\_data;

如果在rtconfig.h中使能了RT\_USING\_DEVICE\_SUSPEND宏，还会增加两个函数

rt\_err\_t (\*suspend) (rt\_device\_t dev);

rt\_err\_t (\*resumed) (rt\_device\_t dev);

这些域并不一定全部填充，后面我们会看到对于有些函数，可以为其填充一个空函数。

RTT的设备管理，可以简单的概括为：每一个设备都会用于一个rt\_device数据结构，这些数据结构通过某种方式组织起来，每个数据结构都会有一个唯一的device\_id，以及一组硬件操作函数等等。这样硬件就被抽象成统一的逻辑设备了，即rt\_device。

还有一个小问题，device\_id是纯粹的数字，所以难以记忆，因此RTT中为其分配一个ascii码字符串来以方便是使用，比如将字符串”uart”和usart的rt\_device数据结构关联起来，这和网络里，ip地址不好记忆，因此使用域名系统是一个道理。

那么自然而然，我们需要一些函数来操作逻辑设备，这些函数在rt-thread/src/device.c文件中提供，它们是：

* rt\_err\_t rt\_device\_register(rt\_device\_t dev, const char \*name, rt\_uint16\_t flags)
* 将rt\_device数据结构加入到RTT的设备层中，这个过程称为“注册”。RTT的设备管理层会为这个数据结构创建唯一的device\_id。
* rt\_err\_t rt\_device\_unregister(rt\_device\_t dev)
  + 与注册相反，自然是注销了，将某个设备从RTT的设备驱动层中移除。
* rt\_device\_t rt\_device\_find(const char \*name)
  + 根据设备的字符串名查找某个设备。
* rt\_err\_t rt\_device\_init(rt\_device\_t dev)
  + 通过调用rt\_device数据结构中的init函数来初始设备。
* rt\_err\_t rt\_device\_init\_all(void)
  + 初始化RTT设备管理层中的所有已注册的设备
* rt\_err\_t rt\_device\_open(rt\_device\_t dev, rt\_uint16\_t oflag)
  + 通过调用rt\_device数据结构中的open函数来打开设备。
* rt\_err\_t rt\_device\_close(rt\_device\_t dev)
  + 通过调用rt\_device数据结构中的close函数来关闭设备。
* rt\_size\_t rt\_device\_read(rt\_device\_t dev, rt\_off\_t pos, void \*buffer, rt\_size\_t size)
  + 通过调用rt\_device数据结构中的read函数来从设备上读取数据。
* rt\_size\_t rt\_device\_write(rt\_device\_t dev, rt\_off\_t pos, const void \*buffer, rt\_size\_t size)
  + 通过调用rt\_device数据结构中的write函数来向设备写入数据（比如设备是flash，SD卡等，nand or nor flash等等）。

说明：关于这些函数各个参数的作用，建议参考官方提供的API文档。<http://www.rt-thread.org/rt-thread/rttdoc_1_0_0/group___device.html>