RT设备

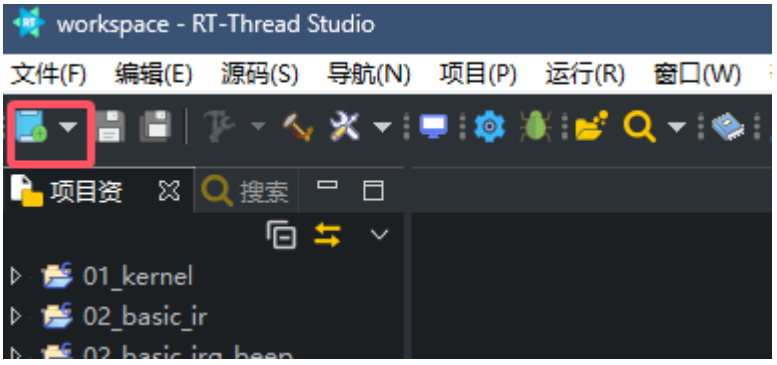
简介:

1. RTC （Real-Time Clock）实时时钟可以提供精确的实时时间，它可以用于产生年、月、时、分、秒等信息。目前实时时钟芯片大多采用精度较高的晶体振荡器作为时钟源。有些时钟芯片为了在主电源掉电时还可以工作，会外加电池供电，使时间信息一直保持有效。
2. 在RT-Thread的编辑器中，可以通过RT-Thread_Setting来设置RTC的使用，下面主要介绍RT_ide内RTC驱动的使用方法。

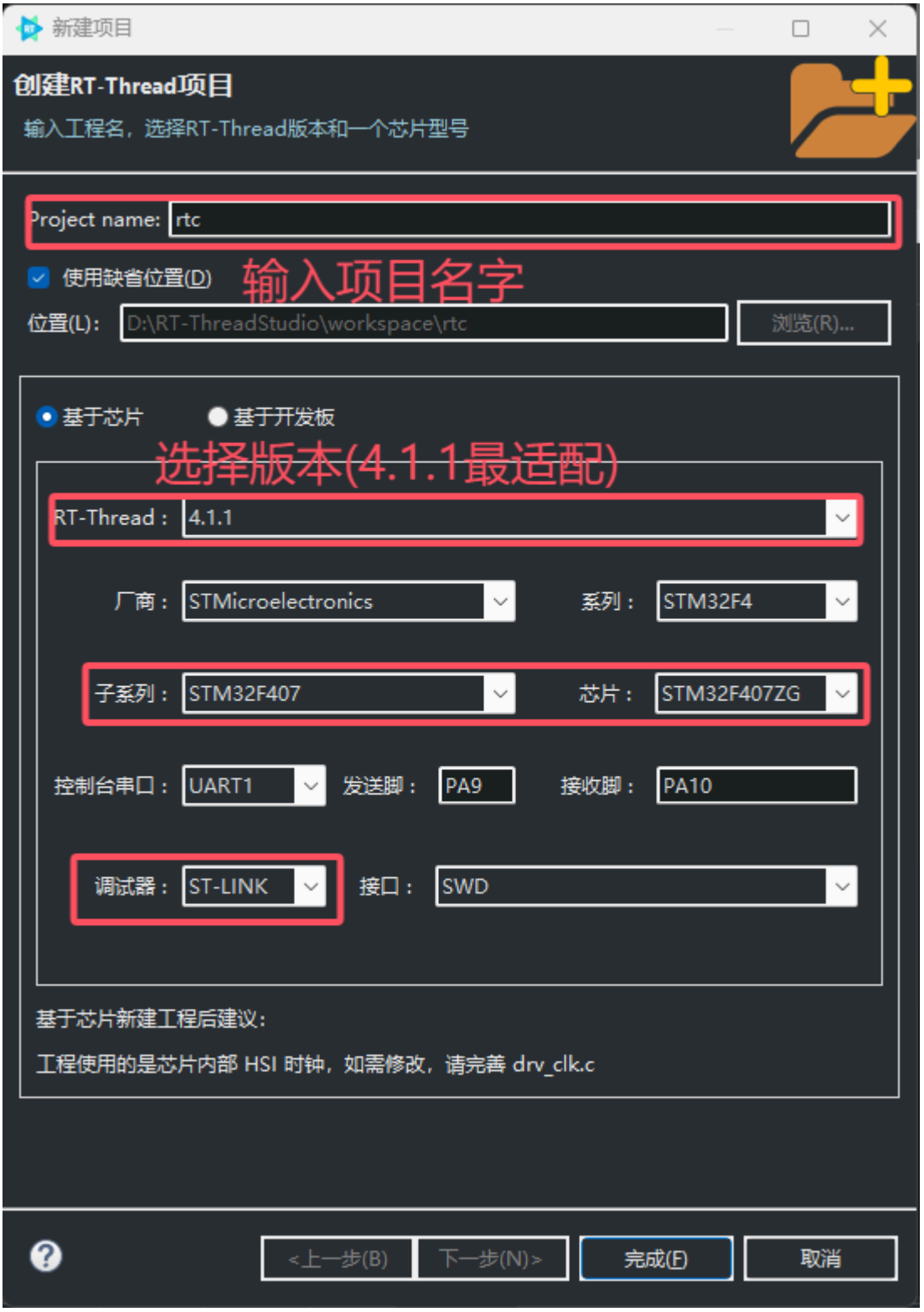
详细步骤如下:

1. 新建文件:

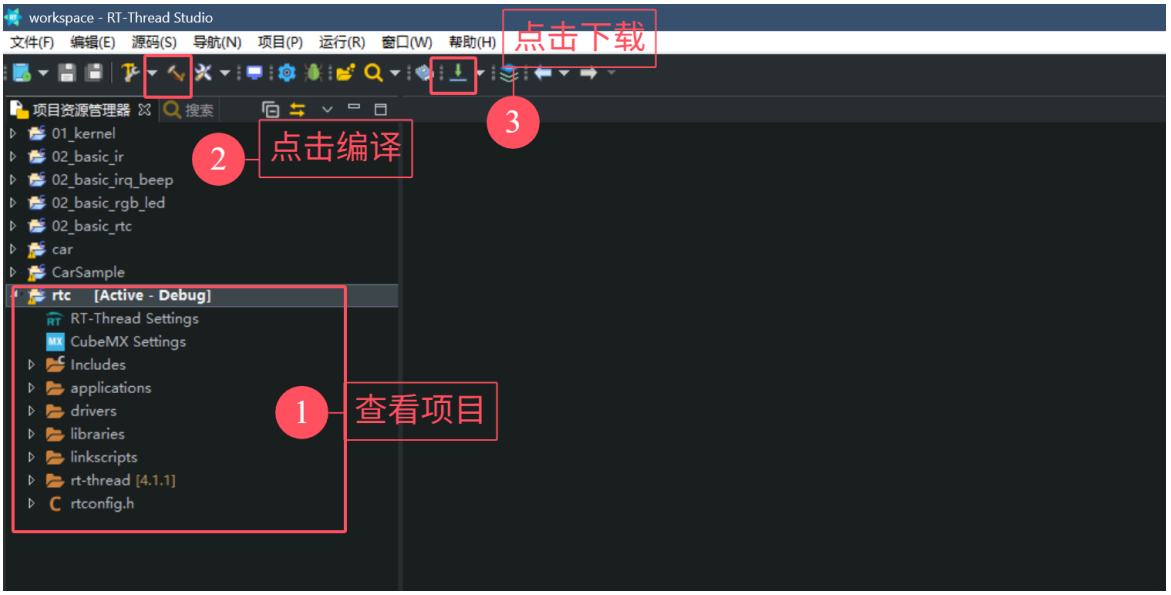
- 点击左上角，新建rt_thread项目



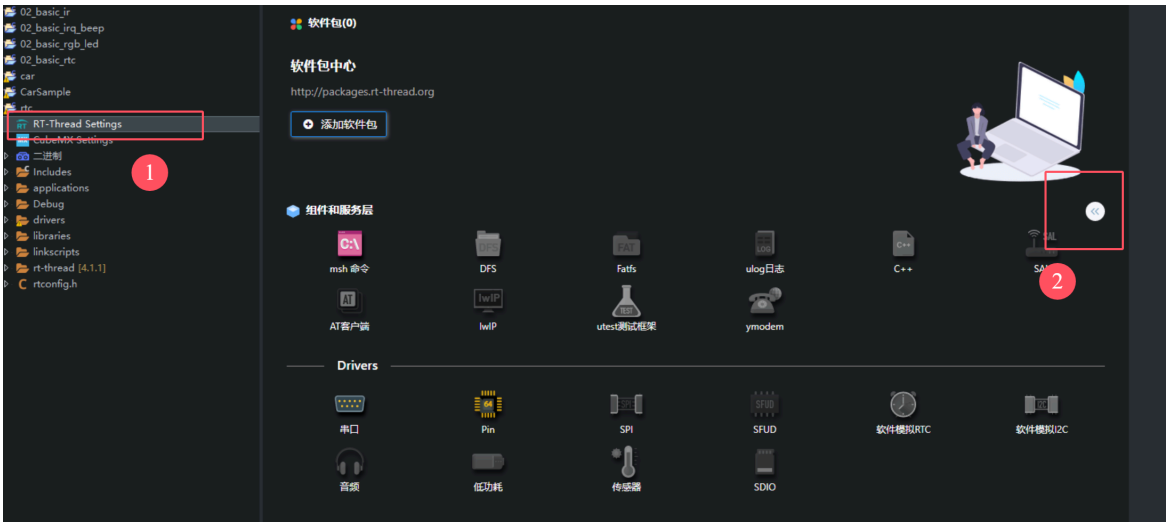
- 配置项目，点击右下角完成



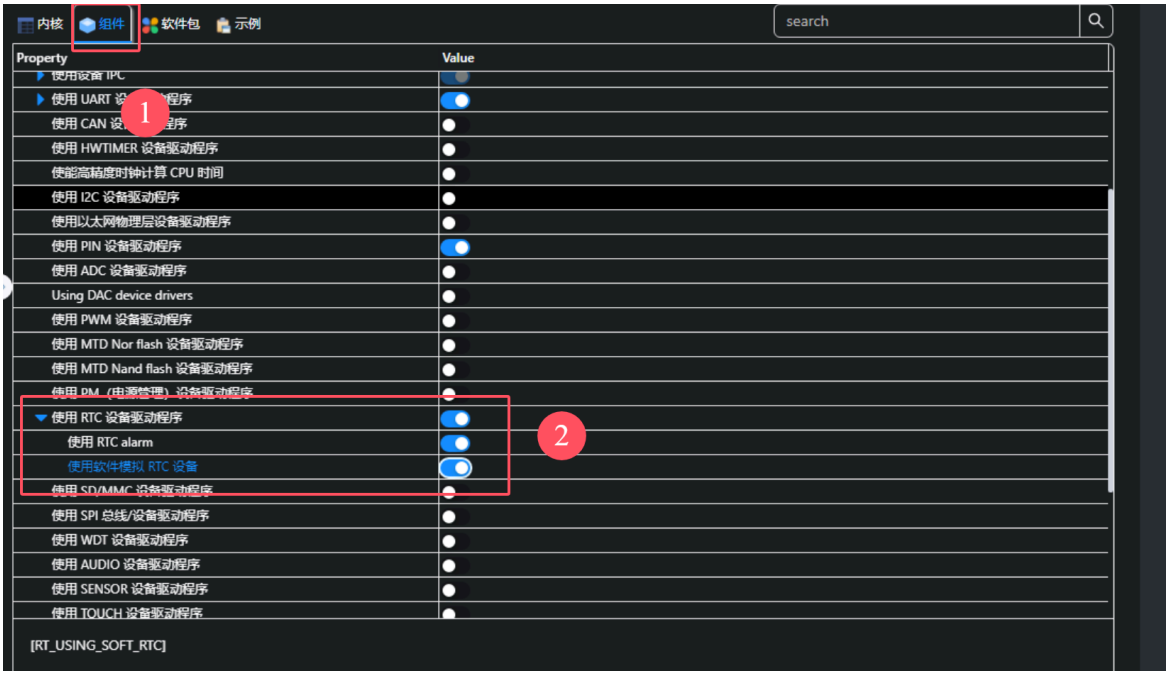
- 生成项目如图所示，左上角编译，若无报错（有警告是正常的），则生成完毕



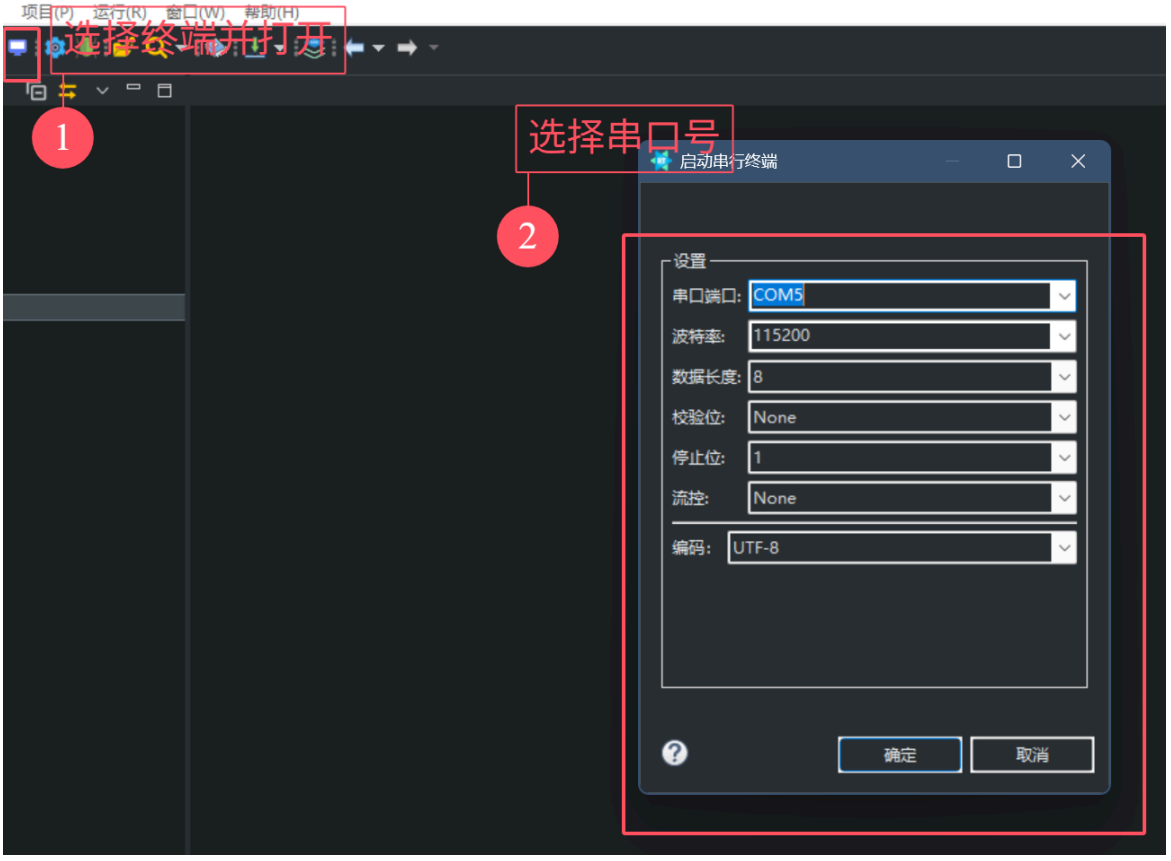
- 在左上角RT-Thread Settings处使能rtc驱动



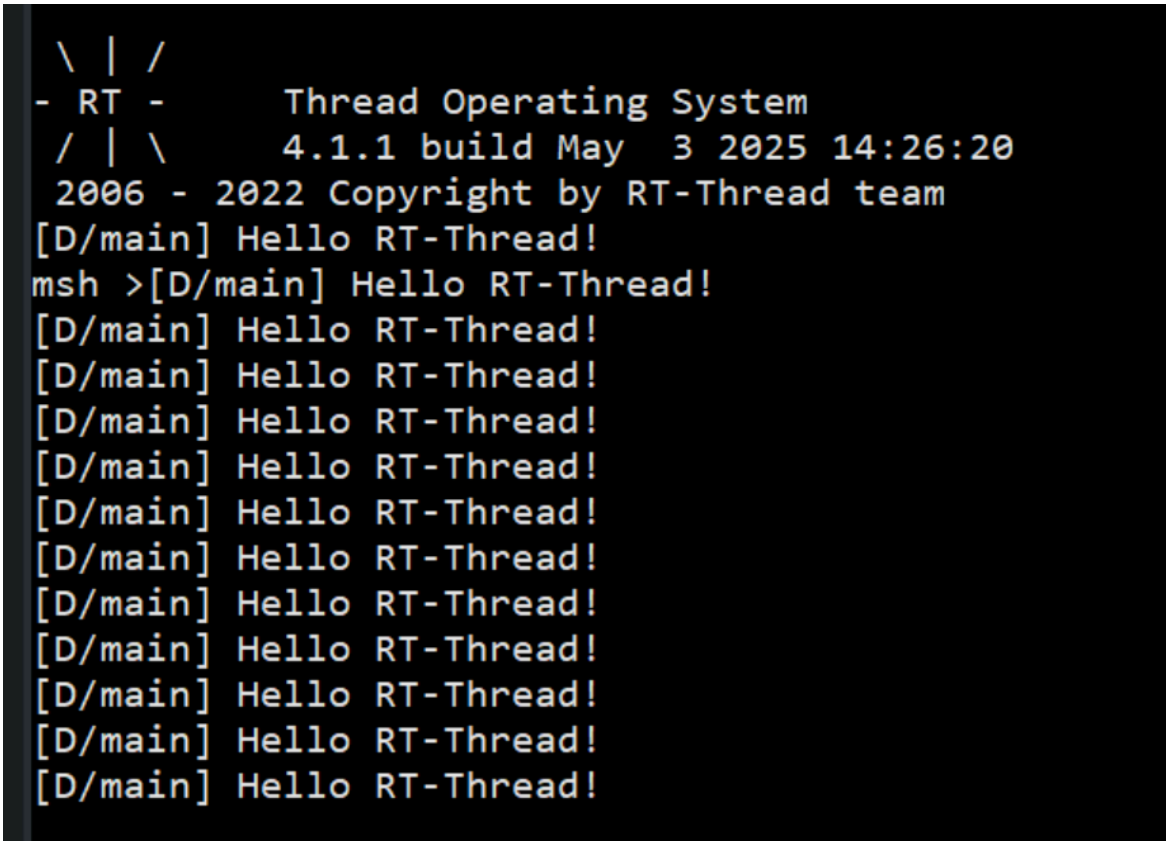
- 选择组件，使能RTC驱动程序，Ctrl+S保存



- 查看终端能否正常打印（要用USB转TTL链接电脑查看现象）



- 打印现象



2. 书写代码

###

1. `time_t`

`time_t` 是 C 和 C++ 中定义的一种数据类型，通常用于表示时间点。它是由标准库 `<time.h>`（或在 C++ 中 `<ctime>`）定义的，通常以秒为单位存储自 Unix 纪元（1970 年 1 月 1 日 00:00:00 UTC）以来的时间。

- **用法**: `time_t` 通常用来存储时间戳，例如当前时间或某个特定时间。
- **与时间相关的函数配合使用**: 它经常与标准库中的时间函数一起使用，例如 `time()`、`localtime()` 和 `gmtime()`。

---

2. `rt_device_t`

函数原型

```c
struct rt_device
{
    struct rt_object          parent;         /* 内核对象基类 */
    enum rt_device_class_type type;           /* 设备类型 */
    rt_uint16_t               flag;           /* 设备参数 */
    rt_uint16_t               open_flag;      /* 设备打开标志 */
    rt_uint8_t                ref_count;      /* 设备被引用次数 */
    rt_uint8_t                device_id;      /* 设备 ID, 0 - 255 */

    /* 数据收发回调函数 */
    rt_err_t (*rx_indicate)(rt_device_t dev, rt_size_t size);
    rt_err_t (*tx_complete)(rt_device_t dev, void *buffer);

    const struct rt_device_ops *ops;          /* 设备操作方法 */

    /* 设备的私有数据 */
    void *user_data;
};
typedef struct rt_device *rt_device_t;

struct rt_device_ops
{
    /* 通用设备接口 */
    rt_err_t  (*init)   (rt_device_t dev);                  /* 初始化设备 */
    rt_err_t  (*open)   (rt_device_t dev, rt_uint16_t oflag); /* 打开设备 */
    rt_err_t  (*close)  (rt_device_t dev);                /* 关闭设备 */
    rt_size_t (*read)   (rt_device_t dev, rt_off_t pos, void *buffer,
rt_size_t size); /* 从设备读取数据 */
    rt_size_t (*write)  (rt_device_t dev, rt_off_t pos, const void *buffer,
rt_size_t size); /* 向设备写入数据 */
    rt_err_t  (*control)(rt_device_t dev, int cmd, void *args); /* 控制设备 */
};
```

- **作用**: 通过 `rt_device_t` 定义设备的句柄。

---

3. `rt_device_find()`

```
rt_device_t rt_device_find(const char *name)
{
    return (rt_device_t)rt_object_find(name, RT_Object_Class_Device);
}
```

- **作用**: 用于查找设备，获得当前设备的句柄。

---

4. `rt_device_open()`

```
rt_err_t rt_device_open(rt_device_t dev, rt_uint16_t oflag);
```

参数	描述
`dev`	设备句柄
`oflag`	设备打开模式标志

返回值:

返回值	意义
`RT_EOK`	设备打开成功
`-RT_EBUSY`	如果设备注册时指定的参数中包括 `RT_DEVICE_FLAG_STANDALONE` 参数，此设备将不允许重复打开
其他错误码	设备打开失败

---

5. `set_date()` 和 `set_time()`

- **函数原型**:

  ```
  rt_err_t set_date(rt_uint32_t year, rt_uint32_t month, rt_uint32_t day);
  rt_err_t set_time(rt_uint32_t hour, rt_uint32_t minute, rt_uint32_t
  second);
  ```

- **来源**: 这两个函数来自于在 `RT-Thread Settings` 中使能 RTC 驱动后生成的 `rtc.c` 文件。
- **作用**: 用于设置日期和时间。
-

6. 获取时间

```
time_t now;
now = time(RT_NULL);
```

`time()` 函数

`time()` 是一个标准 C 库函数，定义在 `<time.h>` 中，其作用如下：

1. **返回当前时间戳**:
   - 以秒为单位返回自 Unix 纪元（1970 年 1 月 1 日 00:00:00 UTC）以来的时间。
2. **参数说明**:
   - 如果传入参数为 `NULL`（或这里的 `RT_NULL`），函数直接返回当前时间戳。
   - 如果传入一个指针（如 `time_t *`），函数会将时间戳存储到该指针指向的变量中，同时返回时间戳。

`RT_NULL`

`RT_NULL` 是一个宏，通常在嵌入式系统中用作空指针的定义，等价于标准 C 的 `NULL`。在此场景中，`RT_NULL` 表示不需要将时间戳存储到任何变量中，仅返回时间值。

7. `rt_kprintf("当前时间: %s\n", ctime(&now));`

- **作用**: 打印当前时间到终端。
- **函数解析**:
  - `rt_kprintf()` 是 RT-Thread 提供的打印函数，类似于标准 C 的 `printf()`，用于在终端输出调试信息。
  - `ctime()` 是标准 C 库函数，定义在 `<time.h>` 中，用于将时间戳（`time_t` 类型）转换为可读的字符串格式。
- **参数说明**:
  - `ctime(&now)` 将 `time_t` 类型的时间戳 `now` 转换为字符串格式。
  - `%s` 是格式化字符串，用于输出 `ctime()` 返回的字符串。
- **示例输出**:

```
当前时间: Mon Dec  3 11:15:50 2018
```

- **注意事项**:
  - `ctime()` 返回的字符串末尾包含换行符。
  - 使用 `ctime()` 时需确保传入的指针有效且非空。

代码书写思路:

1. **设备查找**:
   - 使用 `rt_device_find()` 函数查找 RTC 设备，确保设备存在并获取其句柄。
2. **设备初始化**:
   - 调用 `rt_device_open()` 打开 RTC 设备，确保设备可以正常使用。
3. **设置日期和时间**:
   - 使用 `set_date()` 和 `set_time()` 函数分别设置 RTC 的日期和时间。
4. **获取当前时间**:
   - 调用 `time()` 函数获取当前时间戳，并通过 `ctime()` 转换为可读的时间格式。
5. **延时操作**:

- 使用 `rt_thread_mdelay()` 延时 1 秒后再次获取时间，验证时间的递增。

6. **闹钟功能**:
   - 定义用户闹钟回调函数 `user_alarm_callback()`，用于处理闹钟触发事件。
   - 使用 `rt_alarm_create()` 创建闹钟，并设置触发时间为当前时间的 5 秒后。
   - 调用 `rt_alarm_start()` 启动闹钟。
7. **命令导出**:
   - 使用 `MSH_CMD_EXPORT()` 将闹钟示例代码导出为命令，便于在终端中测试。

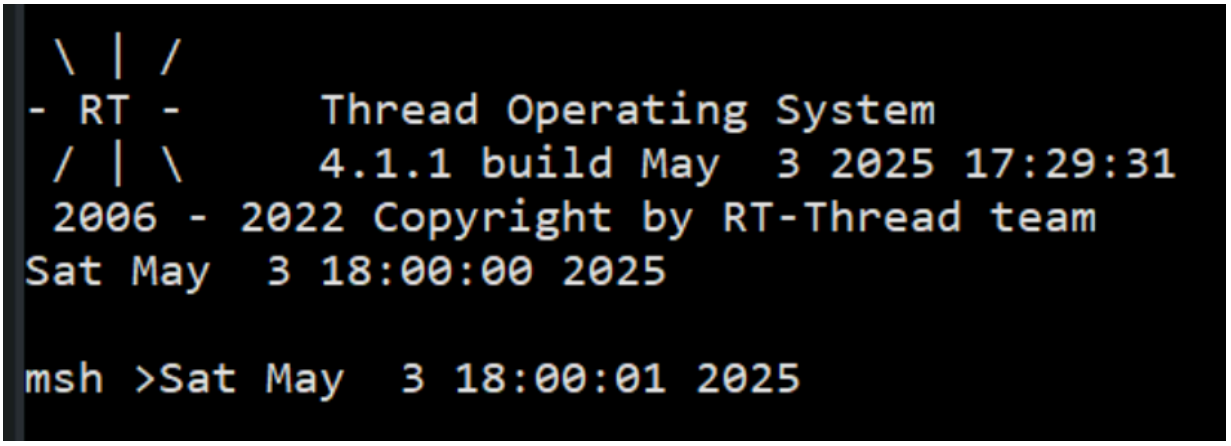
综合代码如下:

```c
#include <rtthread.h>
#include <rtdevice.h>

#define RTC_NAME       "rtc"

int main()
{
    rt_err_t ret = RT_EOK;
    time_t now;
    rt_device_t device = RT_NULL;

    /* 寻找设备 */
    device = rt_device_find(RTC_NAME);
    if (!device)
    {
        rt_kprintf("find %s failed!", RTC_NAME);
        return RT_ERROR;
    }

    /* 初始化RTC设备 */
    if(rt_device_open(device, 0) != RT_EOK)
    {
        rt_kprintf("open %s failed!", RTC_NAME);
        return RT_ERROR;
    }

    /* 设置日期 */
    ret = set_date(2025, 5, 3);
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC date failed\n");
        return ret;
    }

    /* 设置时间 */
    ret = set_time(18, 00, 00);
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC time failed\n");
        return ret;
    }

    /* 获取时间 */
```

```
    now = time(RT_NULL);
    rt_kprintf("%s\n", ctime(&now));

    /* 延时1秒 */
    rt_thread_mdelay(1000);

    /* 获取时间 */
    now = time(RT_NULL);
    rt_kprintf("%s\n", ctime(&now));

    return ret;
}

void user_alarm_callback(rt_alarm_t alarm, time_t timestamp)
{
    rt_kprintf("user alarm callback function.\n");
}

void alarm_sample(void)//设置闹钟中断函数
{
        struct rt_alarm_setup setup; // 定义闹钟设置结构体
        struct rt_alarm * alarm = RT_NULL; // 定义闹钟指针并初始化为 NULL
        static time_t now; // 定义静态变量存储当前时间戳
        struct tm p_tm; // 定义时间结构体

        if (alarm != RT_NULL) // 如果闹钟已存在，直接返回
            return;

        /* 获取当前时间戳，并把下5秒时间设置为闹钟时间 */
        now = time(NULL) + 5; // 获取当前时间戳并加上5秒
        gmtime_r(&now,&p_tm); // 将时间戳转换为 UTC 时间结构体

        setup.flag = RT_ALARM_ONESHOT; // 设置闹钟为单次触发模式
        setup.wktime.tm_year = p_tm.tm_year; // 设置闹钟触发的年份
        setup.wktime.tm_mon = p_tm.tm_mon; // 设置闹钟触发的月份
        setup.wktime.tm_mday = p_tm.tm_mday; // 设置闹钟触发的日期
        setup.wktime.tm_wday = p_tm.tm_wday; // 设置闹钟触发的星期
        setup.wktime.tm_hour = p_tm.tm_hour; // 设置闹钟触发的小时
        setup.wktime.tm_min = p_tm.tm_min; // 设置闹钟触发的分钟
        setup.wktime.tm_sec = p_tm.tm_sec; // 设置闹钟触发的秒钟

        alarm = rt_alarm_create(user_alarm_callback, &setup); // 创建闹钟并绑定回调函数
        if(RT_NULL != alarm) // 如果闹钟创建成功
        {
            rt_alarm_start(alarm); // 启动闹钟
        }
}
/* export msh cmd */
MSH_CMD_EXPORT(alarm_sample,alarm sample);//加入命令行
```

测试现象:

在初始化后,隔一秒会打印一次



此时,按下回车键并输入刚加入msh的命令:alarm_sample

则隔了5秒后会打印出如下情况

如此,则配置完成