

RT-Thread操作系统的μC/OS-II兼容层

让基于μC/OS-II开发的应用层无感地迁移到RT-Thread操作系统

中文 | English

RT-Thread操作系统的μC/OS-II兼容层

让基于μC/OS-II开发的应用层无感地迁移到RT-Thread操作系统

0 前排提示

1 概述

- 1.1 RT-Thread的其他RTOS兼容层
- 1.2 本兼容层适合于
- 1.3 版本详细信息
- 1.4 官网

2 使用

- 2.1 Keil-MDK仿真工程
- 2.2 迁移步骤
- 2.3 os_cfg.h配置文件
- 2.4 运行
 - 2.4.1 手动初始化流程
 - 2.4.2 自动初始化流程
- 2.5 注意

3 接口

- 3.1 新增的API
- 3.2 没有实现兼容的API (仅2个)
- 3.3 钩子函数
- 3.4 统计任务 (OS_TaskStat())
- 3.5 任务控制块、内核对象控制块 (结构体)
- 3.6 全局变量

4 Env工具自动化配置到工程中

- 4.1 配置方法
- 4.2 可选功能说明
 - 4.2.1 Enable uCOS-II wrapper automatically init
 - 4.2.2 Enable uCOS-II wrapper tiny mode

5 友情链接

- 5.1 RT-Thread Nano移植教程
- 5.2 RT-Thread FinSH控制台教程

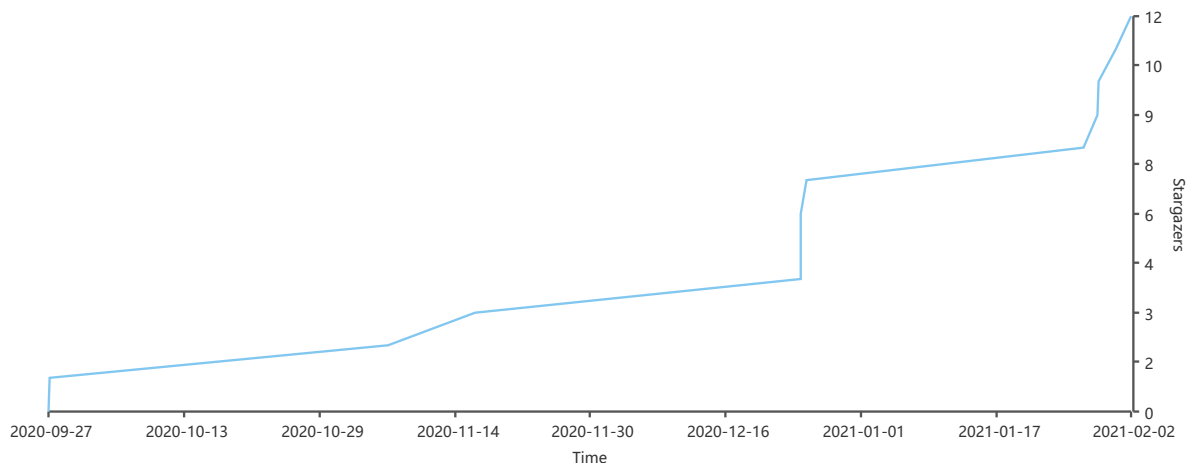
6 其他

- 6.1 联系方式
- 6.2 主页
- 6.3 开源协议
- 6.4 支持

0 前排提示

本文含有图片，受限于中国大陆互联网环境，访问github时，**readme.md(本文件)**的图片一般加载不出来，因此我导出了.pdf文件。如果您需要详细阅读，可以将项目下载或clone下来，阅读[docs/中文说明文档.pdf](#)文件。

如果你喜欢本项目，请点击右上角的Star予以支持，开源项目的成就感就靠star了，谢谢！



1 概述

这是一个针对RT-Thread国产操作系统的 μ COS-II操作系统兼容层，可以让基于美国Micrium公司的 μ COS-II操作系统的项目快速、无感地迁移到RT-Thread操作系统上。在兼容层的设计、编写上尊重原版 μ C/OS-II，保证原版 μ C/OS-II的原汁原味。本项目100%由C语言完成。

支持版本： μ C/OS-II 2.00-2.93全部版本

1.1 RT-Thread的其他RTOS兼容层

RT-Thread操作系统的 μ COS-III兼容层：<https://github.com/mysterywolf/RT-Thread-wrapper-of-uCOS-III>

1.2 本兼容层适合于

- 之前学习过 μ COS-II操作系统，意图转向学习RT-Thread国产操作系统。本兼容层可以帮您用已有的 μ COS-II编程经验和习惯快速将项目跑起来，日后在应用过程中深入熟悉RT-Thread的API函数，逐步向RT-Thread过度，降低您的学习门槛和时间成本。**有了本兼容层，对RT-Thread API以及编程风格的不熟悉再也不是您学习RT-Thread的阻力！**
- 现有任务（线程）模块采用 μ COS-II编写，想要用在基于RT-Thread的工程上
- 老项目需要从 μ COS-II操作系统向RT-Thread操作系统迁移
- 当需要快速基于RT-Thread开发产品，但是工程师之前均采用 μ C/OS开发，从未用过RT-Thread的开发经验。本兼容层可以帮助让工程师快速基于 μ C/OS-II开发经验开发产品，简化软件的重用、缩短微控制器新开发人员的学习过程，并缩短新设备的上市时间。
- 避免在从 μ COS-II迁移到RT-Thread时，由于 μ COS-II的编程经验导致的思维定式引发的错误，这种错误一般很难被发现

例如：

1. 软件定时器参数的不同
2. 任务堆栈的数据类型不同

1.3 版本详细信息

组件名称	版本号	配置文件	说明
RT-Thread nano	3.1.3	rtconfig.h	
μC/OS-II	2.93.00	os_cfg.h app_hooks.c	兼容层兼容2.00-2.93全部μCOS-II版本

1.4 官网

RT-Thread: <https://www.rt-thread.org/>

文档中心: <https://www.rt-thread.org/document/site/tutorial/nano/an0038-nano-introduction/>

μCOS-II: <https://www.micrium.com/>

文档中心: <https://doc.micrium.com/pages/viewpage.action?pageId=10753158>

2 使用

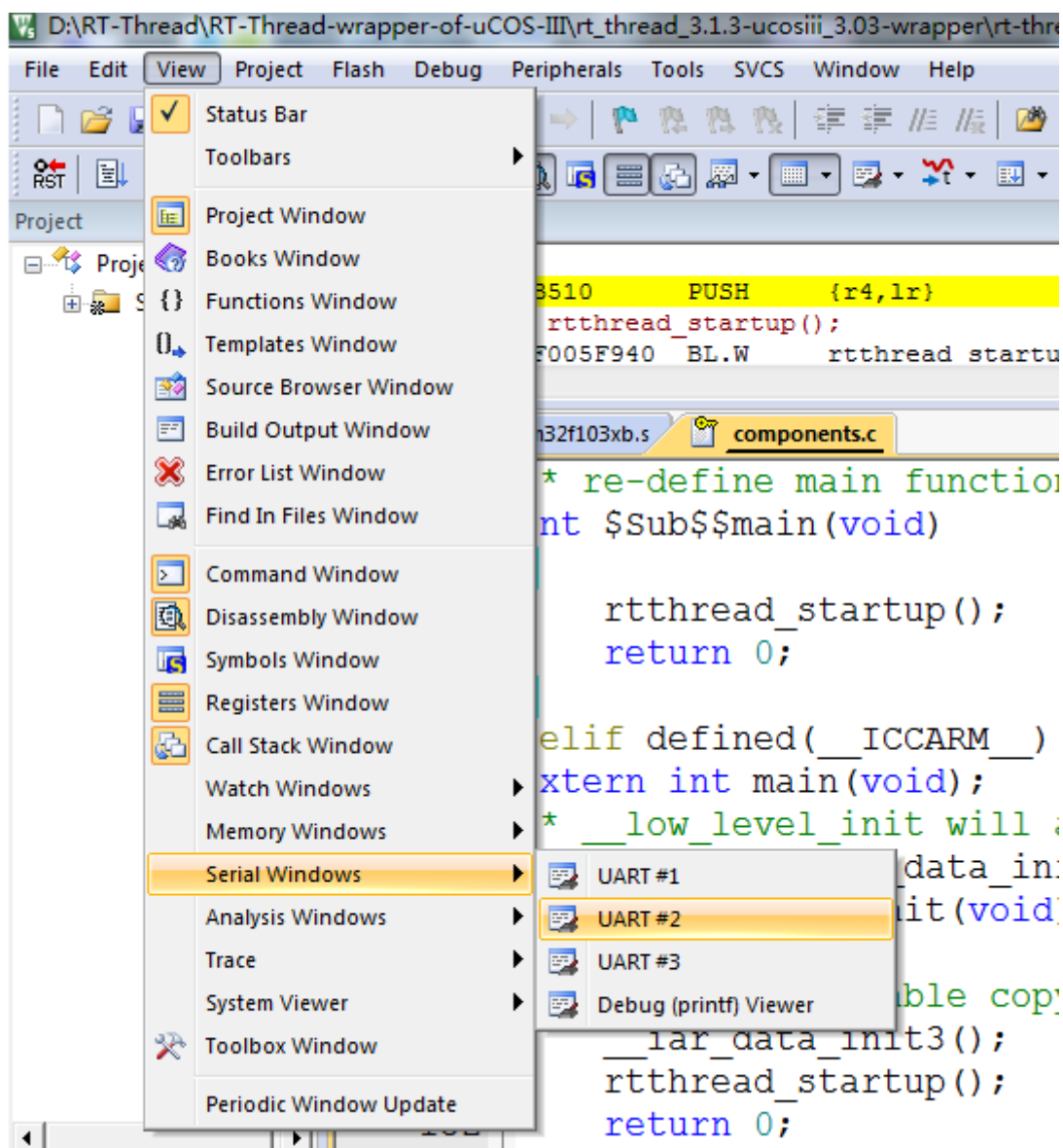
2.1 Keil-MDK仿真工程

本仿真工程是基于STM32F103平台。

Keil工程路径: [rt-thread-3.1.3/bsp/stm32f103/Project.uvprojx](#)

需要提前安装好RT-Thread Nano-3.1.3 [Keil支持包](#)。

注意：调试串口使用的是USART2，不是USART1



2.2 迁移步骤

(如果使用的是RT-Thread Nano版请参见以下步骤；若使用RT-Thread完整版可以直接跳转至[Env工具自动化配置到工程中](#)章节)

1. 将uCOS-II文件夹内的所有文件都加入到你的工程中，最好保持原有文件夹的结构。相较于原版μCOS-II增加了 `os_rtwrap.c` 文件，负责对RT-Thread和μCOS-II的转换提供支持。
2. 配置 `os_cfg.h`
每个选项的配置说明和原版μCOS-II一致，若有不同，我已经在注释中有所解释。
原版μCOS-II配置说明可参见：
a) 《嵌入式实时操作系统μC/OS-II（第二版）》北京航空航天大学出版社 邵贝贝等译
b) Micrium公司μCOS-II[在线文档](#)
3. μCOS-II原版定时器回调函数是在定时器线程中调用的，而非在中断中调用，因此要使用μCOS-II兼容层的软件定时器，需要将`rtconfig.h`中的宏定义 `RT_USING_TIMER_SOFT` 置1。
4. 由于兼容层采用rt-thread内核自带的堆内存分配方式，因此免去了原版uCOS-II中配置任务以及各内核对象内存池大小的步骤，遂需要在`rtconfig.h`中定义 `RT_USING_MEMHEAP`

2.3 os_cfg.h配置文件

```
#define OS_TMR_CFG_TICKS_PER_SEC 10u /* Rate at which timer management task
runs (Hz) */
```

在原版μCOS-II中，该宏定义定义了软件定时器的时基信号，这与RT-Thread的软件定时器有本质的不同，在RT-Thread中，软件定时器的时基信号就等于OS Ticks。因此为了能够将μCOS-II软件定时器时间参数转为RT-Thread软件定时器的时间参数，需要用到该宏定义。请使该宏定义与原工程使用μCOS-II时的该宏定义参数一致。需要注意的是，虽然在兼容层中定义了软件定时器的时基频率，但是在兼容层内部使用的RT-Thread软件定时器的时基频率等同于OS Ticks，因此OS_TMR结构体的OSTmrMatch成员变量其保存的数值是以OS Ticks频率来计算的。

由于兼容层采用rt-thread内核自带的堆内存分配方式，因此免去了原版uCOS-II中配置任务以及各内核对象内存池大小的步骤，遂相关宏定义在兼容层中**均被删除**。

2.4 运行

2.4.1 手动初始化流程

本兼容层完全兼容官方给出的标准初始化流程，如果您兼容老项目，μCOS-II初始化部分无需做任何修改。

2.4.2 自动初始化流程

如果您在应用层中不想手动初始化本兼容层，可以在rtconfig.h文件中定义

PKG_USING_UCOSII_WRAPPER_AUTOINIT宏定义。请参见[4.2.1章节](#)（如无特殊要求，建议采用该种方式）。

2.5 注意

1. μCOS-II的任务堆栈大小单位是sizeof(CPU_STK)，而RT-Thread的线程堆栈大小单位是sizeof(rt_uint8_t)，虽然在兼容层已经做了转换，但是在填写时一定要注意，所有涉及到μCOS-II的API、宏定义全部是按照μCOS-II的标准，即堆栈大小为sizeof(CPU_STK)，**切勿混搭**！这种错误极其隐晦，一定要注意！**下面是混搭的错误示例：**

```
ALIGN(RT_ALIGN_SIZE)
static rt_uint8_t thread2_stack[1024]; // 错误：混搭RT-Thread的数据类型定义线程堆栈

OSTaskCreateExt(task,
    0,
    &task_stack[TASK_SIZE-1],
    TASK_PRIO,
    0,
    &task_stack[0],
    sizeof(thread2_stack), // 任务堆栈大小(错误：这个参数的单位是
    sizeof(CPU_STK))
    0,
    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
```

下面是正确写法：

```

#define THREAD_STACK_SIZE      256 //正确，要通过宏定义单独定义堆栈大小，单位为
sizeof(CPU_STK)
ALIGN(RT_ALIGN_SIZE)
    static CPU_STK thread2_stack[THREAD_STACK_SIZE]; //正确，使用uCOS-II自己的数
据类型定义任务堆栈

OSTaskCreateExt(task,
    0,
    &task_stack[TASK_SIZE-1],
    TASK_PRIO,
    0,
    &task_stack[0],
    THREAD_STACK_SIZE, //任务堆栈大小(正确)
    0,
    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

```

3 接口

3.1 新增的API

额外实现 `OSMutexCreateEx()` 函数，该函数并不在uCOS-II原版的函数中，`OSMutexCreate()` 函数中第一个参数 `prio` 在兼容层中没有任何意义，因此该函数将 `OSMutexCreate()` 函数中的第一个参数略去，以方便用户使用。原因是由于uCOS-II的实现方式过于落后，不支持相同任务在同一优先级。推荐用户使用这个API：

```
OS_EVENT *OSMutexCreateEx (INT8U *perr);
```

额外实现 `OSQCreateEx()` 函数，该函数并不在uCOS-II原版的函数中，`OSQCreateEx()` 函数中第一个参数 `size` 在本兼容层中没有意义，因此该函数将 `OSQCreateEx()` 函数中的第一个参数略去，以方便用户使用。推荐用户使用这个API：

```
OS_EVENT *OSQCreateEx (INT16U size);
```

3.2 没有实现兼容的API (仅2个)

INT8U	<code>OSTaskCreate</code>	(void void OS_STK INT8U	(*task)(void *p_arg), *p_arg, *ptos, prio);
INT16U	<code>OSEventPendMulti</code>	(OS_EVENT OS_EVENT void INT32U INT8U	**pevents_pend, **pevents_rdy, **pmsgs_rdy, timeout, *perr);

3.3 钩子函数

μCOS-II的钩子函数仅对μCOS-II兼容层负责。 即如果你注册了 `OSTaskDelHook` 函数，他仅会在调用 `OSTaskDel`函数时被调用，不会在调用 `rt_thread_detach` 函数时被调用(这个由RTT的钩子函数负责)。这样做是为了层次分明，防止μCOS-II兼容层插手RT-Thread内部事务。

μCOS-II的钩子函数在两个文件中实现：`os_cpu.c` 和 `app_hooks.c`。按照μCOS-II的思想，`os_cpu.c` 提供原始的钩子函数（即这些钩子函数被相应的函数直接调用），该文件以及其内部的钩子函数是移植工程师编写的内容，应用工程师不应该操作这个文件的内容，`os_cpu.c` 文件的钩子函数提供相应的函数指针供 `app_hooks.c` 文件内的钩子函数注册和使用，这个文件内的钩子函数应用工程师是可以操作的。换句话说，我们有什么需要在钩子函数中调用的函数，应该放在 `app_hooks.c` 文件中。

以下原版μCOS-II钩子函数将予以取消，由RT-Thread接管相关钩子函数接管：

```
void OSTaskReturnHook (OS_TCB *p_tcb);
void OSTaskSwHook (void);
void OSTimeTickHook (void);
```

同时，上述钩子函数对应的应用级钩子函数也被取消：

```
void App_TaskReturnHook (OS_TCB *p_tcb);
void App_TaskSwHook (void);
void App_TimeTickHook (void);
```

3.4 统计任务 (OS_TaskStat())

在μCOS-II中，统计任务是一个系统任务，通过 `OS_TASK_STAT_EN` 宏决定是否开启，可以在系统运行时做一些统计工作。CPU的利用率用一个0-100之间的整数表示（对应0% - 100%）。

但是RT-Thread并没有统计任务，因此需要创建一个任务来兼容原版μCOS-II的统计任务，完成上述功能。该统计任务会在兼容层初始化时自动创建，用户无需干预。**用户仅需调用 `OSCPUUsage` 全局变量即可获取当前的CPU使用率，CPU使用率的计算策略和原版μCOS-II完全一致。**

3.5 任务控制块、内核对象控制块（结构体）

本兼容层尽可能的兼容任务、内核对象控制块（结构体）的每个成员变量，确保迁移过来的老程序如果直接访问这些结构体的成员变量也是可以直接运行，无需做修改的（尽管直接访问结构体的成员变量μCOS-II官方并不建议甚至十分反对）。

例如，`OS_TCB` 结构体的各个成员变量如下，可以看到，其包含了原版绝大多数成员变量。如果不用兼容原版成员变量，可以定义宏 `PKG_USING_UCOSII_WRAPPER_TINY`，可以看到 `OS_TCB` 结构体大幅度缩减。也可以将 `OS_TASK_PROFILE_EN`、`OS_TASK_NAME_EN`、`OS_CFG_TASK_REG_TBL_SIZE` 关闭以进一步裁剪。

```
typedef struct os_tcb {
    struct rt_thread OSTask;
    OS_STK             *OSTCBStkPtr;           /* Pointer to current top of stack
                                                */

    #if OS_TASK_CREATE_EXT_EN > 0u
```

```

    void                *OSTCBExtPtr;                /* Pointer to user definable data
for TCB extension      */
    OS_STK              *OSTCBStkBottom;             /* Pointer to bottom of stack
                      */
    INT32U              OSTCBStkSize;                /* Size of task stack (in number of
stack elements)        */
    INT16U              OSTCBOpt;                    /* Task options as passed by
OSTaskCreateExt()      */
    INT16U              OSTCBId;                     /* Task ID (0..65535)
                      */
#endif

    struct os_tcb       *OSTCBNext;                  /* Pointer to next      TCB in the
TCB list                */
    struct os_tcb       *OSTCBPrev;                  /* Pointer to previous TCB in the
TCB list                */

#if OS_TASK_CREATE_EXT_EN > 0u
#if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
    OS_TLS              OSTCBTLStbl[OS_TLS_TBL_SIZE];
#endif
#endif

#ifndef PKG_USING_UCOSII_WRAPPER_TINY
#if (OS_EVENT_EN)
    OS_EVENT            *OSTCBEvtPtr;                /* Pointer to          event
control block          */
#endif
#if (OS_FLAG_EN > 0u)
    OS_FLAGS            OSTCBFlagsRdy;                /* Event flags that made task ready
to run                  */
#endif
    INT32U              OSTCBDly;                    /* Nbr ticks to delay task or,
timeout waiting for event */
#endif
    INT8U              OSTCBStat;                    /* Task      status
                      */
    INT8U              OSTCBStatPend;                /* Task PEND status
                      */
    INT8U              OSTCBPrio;                    /* Task priority (0 == highest)
                      */

#if OS_TASK_DEL_EN > 0u
    INT8U              OSTCBDeReq;                    /* Indicates whether a task needs to
delete itself          */
#endif

#if OS_TASK_PROFILE_EN > 0u
    OS_STK              *OSTCBStkBase;                /* Pointer to the beginning of the
task stack              */
    INT32U              OSTCBStkUsed;                /* Number of bytes used from the
stack                    */
#endif

#if OS_TASK_NAME_EN > 0u
    INT8U              *OSTCBTaskName;
#endif

```



```

#if OS_TASK_REG_TBL_SIZE > 0u
    INT32U          OSTCBRegTbl[OS_TASK_REG_TBL_SIZE];
#endif
} OS_TCB;

```

3.6 全局变量

目前，本兼容层可以使用以下μCOS-II原版全局变量（位于 `ucos_ii.h`）。这些全局变量的具体含义请参见[2.2节](#)中所列举出的参考资料。

```

#if OS_TASK_STAT_EN > 0u
OS_EXT  INT8U          OSCPUUsage;          /* Percentage of CPU used
                                           */
OS_EXT  INT32U          OSIdleCtrMax;        /* Max. value that idle ctr
can take in 1 sec.  */
OS_EXT  INT32U          OSIdleCtrRun;        /* val. reached by idle ctr
at run time in 1 sec. */
OS_EXT  BOOLEAN        OSStatRdy;           /* Flag indicating that the
statistic task is rdy */
OS_EXT  OS_STK          OSTaskStatStk[OS_TASK_STAT_STK_SIZE]; /*
Statistics task stack */
#endif

#define OSIntNesting    rt_interrupt_get_nest() /* Interrupt nesting level
                                           */

#define OSLockNesting    rt_critical_level()    /* Multitasking lock nesting
level */

#define OSPrioCur rt_thread_self()->current_priority /* Priority of current
task */

OS_EXT  BOOLEAN        OSRunning;            /* Flag indicating
that kernel is running */

OS_EXT  INT8U          OSTaskCtr;            /* Number of tasks
created */

#if OS_TASK_STAT_EN > 0u
OS_EXT  volatile INT32U OSIdleCtr;           /* Idle counter
                                           */
#endif

#ifdef OS_SAFETY_CRITICAL_IEC61508
OS_EXT  BOOLEAN        OSSafetyCriticalStartFlag;
#endif

#define OSTCBCur ((OS_TCB*)rt_thread_self()) /* Pointer to
currently running TCB */
OS_EXT  OS_TCB          *OSTCBFreeList;      /* Pointer to list of
free TCBS */
OS_EXT  OS_TCB          *OSTCBLList;         /* Pointer to doubly
linked list of TCBS */
OS_EXT  OS_TCB          *OSTCBPrioTbl[OS_LOWEST_PRIO + 1u]; /* Table of
pointers to created TCBS */

```

```

OS_EXT OS_TCB          OSTCBTbl[OS_MAX_TASKS + OS_N_SYS_TASKS];    /* Table of
TCBs                      */

#if (OS_MEM_EN > 0u) && (OS_MAX_MEM_PART > 0u)
OS_EXT OS_MEM          *OSMemFreeList;                            /* Pointer to free list of
memory partitions        */
OS_EXT OS_MEM          OSMemTbl[OS_MAX_MEM_PART]; /* Storage for memory
partition manager        */
#endif

#if OS_TASK_REG_TBL_SIZE > 0u
OS_EXT INT8U           OSTaskRegNextAvailID;    /* Next available Task
register ID              */
#endif

#if OS_TIME_GET_SET_EN > 0u
#define OSTime          rt_tick_get()          /* Current value of system
time (in ticks)         */
#endif

#if OS_TMR_EN > 0u
#define OSTmrTime       rt_tick_get()          /* Current timer time
                        */
#endif

```

4 Env工具自动化配置到工程中

4.1 配置方法

uCOS-II兼容层在RT-Thread Nano版中需要手动添加到工程中，但如果使用RT-Thread完整版，则可以通过Env工具进行自动化添加到工程中。方法如下：

```

RT-Thread online packages
system packages --->
  [*] Micrium: Micrium software products porting for RT-Thread --->
    [*] uCOS-II Wrapper --->
      [*] Enable uCOS-II wrapper automatically init
      [*] Enable uCOS-II wrapper tiny mode
      version (latest) --->

```

4.2 可选功能说明

4.2.1 Enable uCOS-II wrapper automatically init

uCOS-II兼容层支持按照uCOS-II原版的初始化步骤进行初始化，但是在有些情况，用户不想手动初始化uCOS-II兼容层，想要直接运行应用层任务或模块，则可以使用该宏定义。在 `rtconfig.h` 中定义本宏定义后，在RT-Thread初始化完成并进入到main线程之前会自动将uCOS-II兼容层初始化完毕，用户仅需要专注于uCOS-II的应用级任务即可。

若将该功能开启，则会在 `rtconfig.h` 文件中定义 `PKG_USING_UCOSII_WRAPPER_AUTOINIT` 宏。在 `os_rtwrap.c` 文件中的以下函数将被使能并在 **RT-Thread** 初始化时自动执行。

若没有使用完整版（即nano版）也想使用本功能，可以在 `rtconfig.h` 中手动添加定义宏定义 `PKG_USING_UCOSII_WRAPPER_AUTOINIT`。

```
/**
 * 自动初始化
 * uCOS-II兼容层支持按照uCOS-II原版的初始化步骤进行初始化，但是在有些情况，
 * 用户不想手动初始化uCOS-II兼容层，想要直接运行应用层任务或模块，则可以使用该
 * 宏定义。在rtconfig.h中定义本宏定义后，在RT-Thread初始化完成并进入到main线程之前
 * 会自动将uCOS-II兼容层初始化完毕，用户仅需要专注于uCOS-II的应用级任务即可。
 * The wrapper supports uCOS-II standard startup procedure. Alternatively,
 * if you want to run uCOS-II apps directly and ignore the startup procedure,
 * you can choose this option.
 */
#ifdef PKG_USING_UCOSII_WRAPPER_AUTOINIT
static int rt_ucosii_autoinit(void)
{
    OSInit();                /*uCOS-II操作系统初始化*/
    OSStart();               /*开始运行uCOS-II操作系统*/

    #if OS_TASK_STAT_EN > 0u
        OSStatInit();
    #endif

    return 0;
}
INIT_PREV_EXPORT(rt_ucosii_autoinit);
#endif
```

4.2.2 Enable uCOS-II wrapper tiny mode

如果你在使用过程中不需要兼容任务/内核对象结构体的成员变量，可使能该选项。ENV将自动在 `rtconfig.h` 文件中定义 `PKG_USING_UCOSII_WRAPPER_TINY` 宏定义。该模式可满足所有API的基本兼容需求，**建议勾选该选项**。

5 友情链接

5.1 RT-Thread Nano移植教程

官方文档：

<https://www.rt-thread.org/document/site/tutorial/nano/an0038-nano-introduction/>

视频教程：

基于 MDK 移植 RT-Thread Nano：<https://www.bilibili.com/video/BV1TJ411673o>

基于 IAR 移植 RT-Thread Nano：<https://www.bilibili.com/video/BV1BJ41177CW>

基于 CubeMX 移植 RT-Thread Nano：<https://www.bilibili.com/video/BV1KJ41167qg>

5.2 RT-Thread FinSH控制台教程

官方文档：

<https://www.rt-thread.org/document/site/programming-manual/finsh/finsh/>

视频教程：

<https://www.bilibili.com/video/BV1r741137sY?p=1>

6 其他

6.1 联系方式

维护：[Meco Man](#)

联系方式：jiantingman@foxmail.com

6.2 主页

<https://github.com/mysterywolf/RT-Thread-wrapper-of-uCOS-II>

<https://gitee.com/mysterywolf/RT-Thread-wrapper-of-uCOS-II>（国内镜像，定时同步）

6.3 开源协议

采用 Apache-2.0 开源协议，细节请阅读项目中的 LICENSE 文件内容。

6.4 支持

如果您喜欢本项目可以在本页右上角点一下Star，可以赏我五毛钱，用以满足我小小的虚荣心，并激励我继续维护好这个项目。

