Shell

HRTOS Shell 模块使用手册

1.1 模块概述

HRTOS Shell 是运行在 HRTOS 上的轻量级命令行交互模块。用户通过 UART 串口输入命令，Shell 完成输入缓冲、命令解析、命令匹配和命令执行，并将执行结果输出到串口终端。

Shell 当前提供五个内置主命令：

命令	功能
`help`	显示命令列表
`version`	显示系统版本与配置信息
`task`	查看任务状态、挂起和恢复任务、修改任务优先级
`wait`	查看处于等待状态的任务
`msg`	查看消息队列状态

Shell 将交互功能与内核管理接口结合，可用于开发调试、运行状态检查、现场故障诊断及功能验证。

需要注意：Shell 是系统调试与管理接口，不等同于完整的命令解释器。当前实现主要支持以空格分隔参数的简单命令，不提供引号、转义、管道或命令历史等功能。

1.2 模块架构

Shell 的主要处理流程如下：



```
shell_process()
        |
        ▼
shell_parse_line()
        |
        ▼
shell_cmd_execute()
        |
        ▼
对应的命令处理函数
        |
        ▼
  UART 输出结果
```

1.2.1 文件职责

文件或模块	主要职责
`shell.h`	Shell 配置宏、全局变量和公共接口声明
`shell_cmd.h`	命令处理函数及命令表结构定义
Shell 初始化模块	清理输入状态并显示提示符
UART 接收模块	处理接收到的字符并提交输入事件
Shell 核心模块	周期处理、命令行分割和参数生成
`shell_port.c`	提供串口字符串输出适配
Shell 格式化输出模块	提供轻量级 `shell_printf()`
`shell_cmd.c`	定义命令表并分派命令
`help` 命令模块	显示支持的主命令
`version` 命令模块	输出系统版本信息
`task` 命令模块	任务查询与管理
`wait` 命令模块	查询等待状态
`msg` 命令模块	查询消息队列信息
BSP UART 模块	配置 UART、Timer1 和串口发送

具体目录名称以实际工程组织为准。

1.3 配置参数与全局变量

1.3.1 配置宏

相关定义位于 `shell.h`：

```
#define SHELL_MAX_LINE_LEN    32
#define SHELL_MAX_ARGC        8
#define SHELL_EVENT_RX        1

#define SHELL_CMD_COUNT       5
```

宏	默认值	说明
`SHELL_MAX_LINE_LEN`	32	输入缓冲区长度，包含字符串结束符所需空间
`SHELL_MAX_ARGC`	8	最多保存的命令参数数量，包含主命令
`SHELL_EVENT_RX`	1	Shell 输入完成后使用的事件标识
`SHELL_CMD_COUNT`	5	命令表中的主命令数量

在当前配置下，输入缓冲区最多保存 31 个字符，最后一个字节用于字符串结束符 `'\0'`。参数数组最多保存 8 个参数。

如果需要增加输入长度或参数数量，应同步评估 RAM 占用和解析函数的适配情况。

1.3.2 全局变量

```
extern char shell_line[SHELL_MAX_LINE_LEN];
extern unsigned char shell_pos;
```

- `shell_line`：保存当前接收到的命令行。
- `shell_pos`：指示下一次写入字符的位置。

串口接收处理与 Shell 任务都会访问这些变量，因此它们属于中断和任务之间共享的状态。修改输入逻辑时，应特别注意缓冲区一致性以及中断与任务之间的同步。

1.4 初始化与启动

1.4.1 当前初始化流程

当前应用入口执行以下操作：

```
void hrtos_main(void)
{
    shell_init();
    bsp_uart_init();
    os_event_init(SHELL_EVENT_RX);

    os_task_create(shell_task, 1, 6, 6);
    os_task_create(uart_isr, 0, 9, 0);

    os_task_create(demo_task1, 2, 1, 2);
    os_task_create(demo_task2, 3, 2, 2);
}
```

以上参数应结合 HRTOS 当前版本的 `os_task_create()` 接口解释。这里展示的是工程中的实际调用，不代表所有参数均适用于其他版本。

任务入口	ID	优先级	堆栈参数
`uart_isr`	0	9	0

任务入口	ID	优先级	堆栈参数
`shell_task`	1	6	6
`demo_task1`	2	1	2
`demo_task2`	3	2	2

表中堆栈参数的实际含义及单位应以 HRTOS 内核接口定义为准。`uart_isr` 的注册方式属于当前工程的设计，不能仅凭函数名称推断它就是传统意义上由硬件直接调用的 ISR；实际中断入口和分派机制应以 HRTOS 中断模块为准。

1.4.2 Shell 初始化

```
void shell_init(void)
{
    shell_pos = 0;
    shell_line[0] = '\0';
    shell_prompt();
}
```

`shell_init()` 将输入位置复位，清空输入缓冲区的首字符，并显示提示符：

```
HRTOS>
```

当前 `hrtos_main()` 和 `shell_task()` 都调用了 `shell_init()`，因此启动过程中可能出现两次提示符输出。若希望启动界面只显示一次提示符，应统一初始化职责，避免重复初始化。

1.4.3 Shell 任务

当前 Shell 任务的主要逻辑如下：

```
static void shell_task(void)
{
    shell_init();

    while (1)
    {
        shell_process();
        os_delay(10);
    }
}
```

Shell 任务循环调用 `shell_process()`，并在每轮处理后调用 `os_delay(10)`。

延时参数的实际时间单位及其调度行为，应以当前 HRTOS 时间接口的定义为准，不应脱离内核配置直接将其视为固定毫秒数。

1.5 UART 配置

当前 BSP 使用 Timer1 模式 2 产生 UART 波特率时钟。

```
void bsp_uart_init(void)
{
    SCON = 0x50;

    TMOD &= 0x0F;
    TMOD |= 0x20;

    TL1 = 0xFD;
    TH1 = 0xFD;

    ET1 = 0;
    TR1 = 1;

    EA = 1;
    ES = 1;
}
```

主要配置如下：

配置项	当前设置	说明
UART 工作方式	`SCON = 0x50`	串口模式 1，允许接收
Timer1 模式	模式 2	8 位自动重装载定时器
`TH1`	`0xFD`	定时器重装载值
`TL1`	`0xFD`	定时器初始计数值
`ET1`	0	不启用 Timer1 中断
`TR1`	1	启动 Timer1
`ES`	1	使能串口中断
`EA`	1	使能总中断

实际波特率取决于目标 MCU 的串口时钟机制、晶振频率以及相关配置。传统 8051 的常见配置在特定条件下可使用 9600 baud，但更换晶振或芯片型号后，应重新计算并验证。

推荐使用串口终端进行连接测试。数据位、停止位、校验方式和波特率均应与实际硬件配置一致。

1.6 输入接收与命令处理

1.6.1 字符接收

UART 接收处理函数从 `SBUF` 读取字符，并清除接收标志 `RI`。

当前输入逻辑区分三种情况：

- **回车或换行：** 为输入行添加字符串结束符；若输入非空，则调用 `os_event_set_from_isr(SHELL_EVENT_RX)`。
- **退格字符：** 支持 `'\b'` 和 `0x7F`，在输入位置大于零时将位置减一。
- **普通字符：** 当缓冲区尚有空间时，保存字符并递增输入位置。

当前代码未启用逐字符回显和退格删除的终端显示逻辑，因此输入过程中不一定会在终端上看到已输入字符。相关 `shell_putc()` 调用在所提供的接收代码中处于注释状态。

1.6.2 命令行解析

解析函数为：

```
void shell_parse_line(char *line);
```

函数先输出收到的整行文本，然后调用内部的 `shell_strtok()` 按空格分割命令，生成 `argc` 和 `argv`，最后调用 `shell_cmd_execute()`。

例如输入：

```
task list
```

解析结果为：

```
argc = 2;
argv[0] = "task";
argv[1] = "list";
```

其中：

- `argc` 表示参数个数，包含主命令。
- `argv[0]` 是主命令名称。
- `argv[1]` 及后续元素是子命令和参数。

当前分割逻辑会将分隔空格替换为 `'\0'`，因此解析会直接修改原始输入缓冲区。连续空格会被跳过，但不支持带空格的引号参数。

1.6.3 命令匹配

命令分派函数为：

```
void shell_cmd_execute(int argc, char *argv[]);
```

函数遍历 `shell_cmd_table`，通过 `shell_strcmp()` 比较 `argv[0]` 与命令名称。匹配成功后，调用相应处理函数；如果没有找到匹配项，则输出：

```
Unknown command
```

当前命令匹配区分大小写，命令表中使用的是小写名称。

1.6.4 输入事件与处理流程

当前 `shell_process()` 的主要逻辑如下：

```
void shell_process(void)
{
    os_event_wait(SHELL_EVENT_RX, 0);

    shell_parse_line(shell_line);

    shell_pos = 0;
```

```
    shell_line[0] = '\0';

    os_event_delete(SHELL_EVENT_RX);

    shell_prompt();
}
```

其处理顺序为：

- 等待或检查 Shell 接收事件，具体语义由 HRTOS 事件 API 决定。
- 解析输入缓冲区。
- 重置输入位置和缓冲区首字符。
- 删除对应事件。
- 显示下一次输入提示符。

当前实现将输入缓冲区、输入位置与事件标识配合使用。若需要支持高频连续输入、多行缓存或更复杂的串口交互，应进一步检查事件清除时机及中断与任务之间的数据竞争。

1.7 内置命令

1.7.1 help：查看命令列表

用法：

```
help
```

当前输出：

```
Available commands:
  help
  version
  task
  wait
  msg
```

该命令只显示命令名称，不输出每个命令的详细参数帮助。

1.7.2 version：查看版本信息

用法：

```
version
```

当前源码输出：

```
HRTOS Version 4.0.0LTS
CPU: 8051
Task: 16
MsgQ: 3
Resource: 6
Website: https://hrtos.com
Email: admin@hrtos.com
```

其中任务数、消息队列数和资源数是当前版本输出的配置信息。它们是否与实际编译配置一致，应结合对应内核宏定义确认。

1.7.3 task：任务管理

当前支持以下子命令：

```
task list
task suspend <id>
task resume <id>
task priority <id> <pr>
```

1.7.3.1 查看任务列表

```
task list
```

当前输出列：

```
ID STATE PRIO
```

- `ID`：任务索引。
- `STATE`：由 `os_task_get_state()` 返回的状态值。
- `PRIO`：由 `os_task_get_priority()` 返回的优先级值。

该命令遍历任务索引 `0` 至 `15`，仅在 `os_task_is_valid(i) == 1` 时输出相应任务。

状态和优先级的数字含义应参照 HRTOS 内核的状态定义及优先级规则。

1.7.3.2 挂起任务

```
task suspend <id>
```

例如：

```
task suspend 2
```

命令先检查是否提供任务 ID，再将输入转换为数值，并检查 ID 是否小于 16。随后调用 `os_task_suspend(id)`。

成功时输出：

```
suspend ok
```

失败时输出：

```
suspend failed
```

需要注意：当前 `suspend` 和 `resume` 使用 `shell_atoi()` 转换参数。该转换函数不是严格的数字校验器；遇到非数字开头的参数时可能返回 0。因此，不应将当前实现描述为已经完整验证了所有任务 ID 输入格式。

1.7.3.3 恢复任务

```
task resume <id>
```

例如：

```
task resume 2
```

该命令调用 `os_task_resume(id)`。成功时输出 `resume ok`，失败时输出 `resume failed`。

1.7.3.4 修改优先级

```
task priority <id> <pr>
```

例如：

```
task priority 2 5
```

该子命令要求参数数量严格为 4，并使用严格数字解析函数检查任务 ID 和优先级：

- 任务 ID 必须为 `0-15`。
- 优先级必须为 `0-8`。

输入不合法时输出相应的 `Invalid task id` 或 `Invalid priority`；内核接口返回 `-1` 时输出 `priority failed`，其他返回值则输出 `priority ok`。

这里的范围是当前 Shell 命令实现中的参数检查范围，具体优先级的调度含义应以 HRTOS 内核定义为准。

1.7.4 wait：查看等待任务

用法：

```
wait list
```

输出表头：

```
ID  TYPE        OBJ  TICK  FLAG
```

该命令遍历 `OS_PROCESS_MAX` 范围内的任务，只显示 `OS_TASK[i].state == WAIT` 的条目。

字段	含义
`ID`	当前遍历的任务索引
`TYPE`	等待类型
`OBJ`	等待对象标识
`TICK`	等待计时值
`FLAG`	等待标志

当前实现识别以下等待类型：

- `DELAY`
- `SEM`
- `MUTEX`
- `MSG_SEND`
- `MSG_RECV`
- `EVENT`
- `MAILBOX`

未匹配到的类型输出 `UNKNOWN`。

该命令读取内核任务表中的字段，并不独立维护等待状态。显示结果的准确性取决于内核对任务状态、等待类型和相关字段的实际维护方式。

1.7.5 msg：查看消息队列

用法：

```
msg list
```

输出表头：

```
ID  SIZE  COUNT  HEAD  TAIL
```

当前实现遍历 `0` 至 `OS_MSGQ_MAX - 1`，读取消息队列的 `_size`、`count`、`head` 和 `tail` 字段。

字段	含义
`ID`	消息队列索引
`SIZE`	队列配置的大小字段
`COUNT`	当前队列计数字段
`HEAD`	队列头位置
`TAIL`	队列尾位置

这些字段的具体单位和边界含义，应以消息队列实现为准。当前命令是状态查询接口，不负责创建、删除或收发消息。

1.8 Shell 输出接口

1.8.1 shell_puts()

函数原型：

```
void shell_puts(const char *str);
```

功能：将字符串交给 BSP 串口发送函数输出。

当前实现：

```
void shell_puts(const char *str)
{
    bsp_uart_send_str_ram(str);
}
```

字符串必须以 `'\0'` 结束。该接口面向当前 RAM 字符串发送实现；如果需要输出程序存储区中的常量字符串，应确认 Keil C51 存储区模型与 BSP 接口是否兼容。

1.8.2 shell_putc()

函数原型：

```
void shell_putc(char ch);
```

该函数在 `shell.h` 中声明。所提供的 `shell_port.c` 代码只展示了 `shell_puts()` 的实现，未展示 `shell_putc()` 的函数体。

如果工程中没有其他文件提供该函数定义，则应补充实现或移除未使用的声明，以避免链接问题。

1.8.3 shell_printf()

函数原型：

```
void shell_printf(const char *fmt, ...);
```

当前轻量级格式化函数支持以下格式：

格式	含义
%d	有符号十进制整数
%u	无符号十进制整数
%x	十六进制整数，字母使用大写形式
%s	字符串
%c	字符
%%	输出百分号

示例：

```
shell_printf("Task ID: %d\r\n", task_id);
shell_printf("Count: %u\r\n", count);
shell_printf("Value: 0x%x\r\n", value);
shell_printf("Name: %s\r\n", name);
```

当前实现使用固定的 32 字节静态缓冲区，并通过 `tiny_vsprintf()` 格式化数据。它不支持完整标准库 `printf()` 的所有格式控制能力。

重要限制： 当前 `tiny_vsprintf()` 对 `%s` 的复制长度有限制，但其他格式分支没有统一的缓冲区边界检查。因此，不能保证任意格式串和任意参数组合都不会写越界。正式使用前，建议为所有输出路径增加统一的剩余空间检查，并验证格式化结果长度。

此外，静态输出缓冲区不适合未经同步的并发调用。若多个任务可能同时调用 `shell_printf()`，应结合 UART 发送机制评估输出串扰与重入问题。

1.9 自定义命令扩展

1.9.1 命令表

当前命令表结构定义如下：

```
typedef struct
{
    const char *name;
    void (*func)(int argc, char *argv[]);
    const char *help;
} shell_cmd_t;
```

当前命令表：

```
const shell_cmd_t shell_cmd_table[] =
{
    { "help",    shell_cmd_help,    "show command list" },
    { "version", shell_cmd_version, "show system version" },
    { "task",    shell_cmd_task,    "task management" },
    { "wait",    shell_cmd_wait,    "show waiting tasks" },
    { "msg",     shell_cmd_msg,     "message management" }
};
```

新增命令时，需要同时完成：

- 编写命令处理函数。
- 声明命令处理函数。
- 在命令表中添加对应表项。
- 更新 `SHELL_CMD_COUNT`，确保其与命令表实际条目数一致。
- 如果需要通过 `help` 查看新命令，更新帮助命令的显示逻辑。

1.9.2 命令函数

命令函数统一使用以下接口形式：

```
void shell_cmd_mycmd(int argc, char *argv[]);
```

示例：

```
void shell_cmd_mycmd(int argc, char *argv[])
{
    if (argc < 2)
    {
        shell_puts("Usage: mycmd <on|off>\r\n");
        return;
    }

    if (shell_strcmp(argv[1], "on") == 0)
    {
        shell_puts("Turn ON\r\n");
    }
    else if (shell_strcmp(argv[1], "off") == 0)
    {
        shell_puts("Turn OFF\r\n");
    }
    else
    {
        shell_puts("Unknown parameter\r\n");
    }
}
```

新增命令时，应检查参数数量、参数格式及取值范围。对会影响内核状态或硬件状态的操作，还应检查调用时机、权限和失败处理。

1.10 移植说明

移植到其他 8051 系列 MCU 时，主要检查以下部分。

1.10.1 UART 硬件适配

- UART 工作模式及寄存器定义。
- Timer1 或其他波特率时钟源的配置。
- 实际波特率与晶振频率。
- 接收中断入口和中断使能。
- 串口发送函数及字符串所在存储区。

1.10.2 HRTOS 接口适配

- 事件初始化与事件设置接口。
- 中断上下文中的事件操作是否合法。
- 任务创建接口及堆栈参数定义。
- 任务挂起、恢复和优先级设置接口。
- 任务状态、等待状态和消息队列数据结构。

1.10.3 内存与并发

- 输入缓冲区和参数数组的 RAM 占用。
- 格式化输出缓冲区的容量与越界保护。
- UART 发送是否会阻塞调用任务。
- 多任务输出时是否需要串口互斥。
- UART 中断与 Shell 任务共享数据时的同步策略。

1.11 常见问题

1.11.1 无法接收串口字符

检查：

- UART 是否正确初始化。
- `ES` 和 `EA` 是否按预期使能。
- 接收中断是否进入正确的 HRTOS 分派路径。
- 实际波特率是否与终端一致。
- `RI` 是否被正确处理。

1.11.2 输入字符没有回显

当前 UART 接收代码中的字符回显语句被注释，因此没有逐字符回显属于当前实现的行为。

如果需要输入时显示字符，应实现回显，并同时处理退格时的终端显示。应避免在中断上下文中执行过长的阻塞式串口发送。

1.11.3 输入命令后没有结果

检查：

- `SHELL_EVENT_RX` 是否正确初始化。
- 回车或换行是否触发事件设置。
- Shell 任务是否已成功创建并运行。
- `os_event_wait()` 的等待语义是否符合预期。
- 命令表中的数量与实际条目是否一致。

1.11.4 命令无法识别

当前命令匹配区分大小写。例如，命令表注册的是 `task`，应使用：

```
task list
```

同时检查命令是否已加入 `shell_cmd_table`，以及 `SHELL_CMD_COUNT` 是否同步更新。

1.11.5 任务 ID 输入异常

`task suspend` 和 `task resume` 当前使用宽松的 `shell_atoi()` 转换参数。非数字输入可能被转换为 0。

如果需要严格的输入验证，建议这两个子命令也使用严格的无符号整数解析函数，并在转换成功后再检查任务 ID 范围。

1.11.6 输出乱码或格式异常

检查：

- UART 波特率与晶振配置。
- 输出字符串的结束符。
- 格式说明符与实际参数类型是否匹配。
- `shell_printf()` 格式化缓冲区是否足够。
- 多任务并发输出是否造成数据覆盖。

1.12 当前实现边界

本节用于区分现有功能与尚需额外实现或验证的能力。

- **输入长度有限。** 默认缓冲区为 32 字节，最多保存 31 个命令字符。
- **参数数量有限。** 默认最多保存 8 个参数。
- **解析方式简单。** 当前按空格分割，不支持引号参数、转义或复杂命令语法。
- **没有逐字符回显。** 接收代码中的回显逻辑当前被注释。
- **输出格式有限。** `shell_printf()` 是轻量级实现，不是完整标准 `printf()`；当前实现还需要补足统一的边界检查。

- **任务命令依赖内核接口。** 实际状态、挂起、恢复和优先级修改结果由 HRTOS 内核接口决定。
- **共享输入状态需要关注同步。** 中断与任务共同访问输入缓冲区和输入位置，连续输入时应验证同步策略。
- **串口发送为轮询方式。** 当前 `bsp_uart_send_str_ram()` 逐字符等待 `TI`，可能阻塞调用者，不宜直接视为非阻塞输出接口。

1.13 命令速查表

命令	功能
`help`	显示主命令列表
`version`	显示系统版本与配置字段
`task list`	查看有效任务状态
`task suspend <id>`	挂起指定任务
`task resume <id>`	恢复指定任务
`task priority <id> <pr>`	修改任务优先级
`wait list`	查看等待状态的任务
`msg list`	查看消息队列信息

说明： 命令的实际执行结果取决于 HRTOS 内核接口、任务状态及当前编译配置。使用前请确认目标工程与本手册适用版本一致。