FreeRTOS 生态系统之 CLI

中文使用手册

作者：阿普当

目录

1 简介.....3
2 配置和使用.....3
2.1 实现一个命令.....3
2.2 命令参数.....10
2.3 注册一条命令.....11
2.4 命令解释器.....12
2.5 创建一个 FreeRTOS+CLI 任务.....13
2.6 缓冲区管理.....16

1 简介

FreeRTOS+CLI(命令行接口)提供了一种简单、小、可扩展和 RAM 高效的方法，使您的 FreeRTOS 应用程序能够处理输入的命令行。添加命令所需的步骤如下图所示。

2 配置和使用

2.1 实现一个命令

FreeRTOS+CLI 是一个可扩展的框架，它允许应用程序编写者定义和注册自己的命令行输入命令。

本章节描述如何编写实现命令行为的函数。

函数的输入和输出

实现用户定义命令行为的函数必须具有以下接口(原型)：

```
BaseType_t xFunctionName( int8_t *pcWriteBuffer,
                          size_t xWriteBufferLen,
                          const int8_t *pcCommandString );
```

下面是对调用函数时传递给函数的参数的描述，以及必须返回的值。

参数：

pcWriteBuffer	这是一个缓冲区，任何生成的输出都应该写入这个缓冲区。例如，如果函数只是返回固定字符串“HelloWorld”，则字符串将写入 pcWriteBuffer。输出必须始终为空终止。
xWriteBufferLen	这是 pcWriteBuffer 参数指向的缓冲区的大小。将超过 xWriteBufferLen 的字符写入 pcWriteBuffer 将导致缓冲区溢出。

pcCommandString	指向整个命令字符串的指针。访问整个命令字符串允许函数实现提取命令参数(如果存在的话)。FreeRTOS+CLI 提供了接受命令字符串并返回命令参数的辅助函数，因此不需要显式字符串解析。本章提供了一些例子。

返回：

执行某些命令将产生多于一行的输出。例如，文件系统“dir”(或“ls”)命令将为目录中的每个文件生成一行输出。如果目录中有三个文件，输出可能如下所示：

```
file1.txt
file2.txt
file3.txt
```

为了尽量减少 RAM 的使用，并确保 RAM 的使用是确定性的，FreeRTOS+CLI 允许实现命令行为的函数一次只输出一行（注：这个其实是针对有多行输出的命令而说的，不用一次都输出，一行一行的输出就可以）。函数返回值用于指示输出行是否是输出的末尾，或者是否生成更多行。

如果生成的输出是输出的结尾，则返回 pdFALSE，这意味着没有更多的行要生成，并且命令执行已经完成。

如果返回的输出不是输出的结束，并且在命令执行完成之前仍然要生成一个或多个行，则返回 pdTRUE。

我们继续看输出三个文件名的“dir”命令的示例：

1. 第一次调用实现 dir 命令的函数时，可以只输出第一行(file1.txt)。如果这样做了，则函数必须返回 pdTRUE 以指示后面还有更多行。
2. 第二次调用实现 dir 命令的函数时，可以只输出第二行(file2.txt)。如果这样做了，则函数必须再次返回 pdTRUE，以指示后面还有更多的行。
3. 第三次调用实现 dir 命令的函数时，只输出第三行(file3.txt)。这一次，没有更多的行要输出，所以函数必须返回 pdFALSE。

或者，如果有足够的 RAM，并且 xWriteBufferLen 中传递的值足够大，则可以同时返回所有三行——在这种情况下，函数必须在第一次执行时返回 pdFALSE。

每次执行命令时，FreeRTOS+CLI 都会反复调用实现命令行为的函数，直到函数返回 pdFALSE 为止。

示例：

下面提供了以下示例：

1. 不接受参数并返回单个字符串的命令。
2. 不接受参数并返回多个字符串(一次一行)的命令。
3. 要求一定数量的参数的命令。
4. 一种命令，它接受可变数量的参数，并每次返回一个可变数目的字符串。

示例 1：无参数的命令

FreeRTOS 的 vTaskList()API 函数生成一个包含每个任务状态信息的表。该表包含每个任务的一行文本。示例 1 中实现的命令输出此表。示例 1 演示了整个表同时输出的简单情况。代码中的注释提供了更多解释。

```
/* This function implements the behaviour of a command, so must have the correct
prototype. */
static BaseType_t prvTaskStatsCommand( int8_t *pcWriteBuffer,
                                       size_t xWriteBufferLen,
                                       const int8_t *pcCommandString )
{
    /* For simplicity, this function assumes the output buffer is large enough
    to hold all the text generated by executing the vTaskList() API function,
    so the xWriteBufferLen parameter is not used. */
    ( void ) xWriteBufferLen;

    /* pcWriteBuffer is used directly as the vTaskList() parameter, so the table
    generated by executing vTaskList() is written directly into the output
    buffer. */
    vTaskList( pcWriteBuffer + strlen( pcHeader ) );

    /* The entire table was written directly to the output buffer.  Execution
    of this command is complete, so return pdFALSE. */
    return pdFALSE;
}
```

示例 **1:** 一次输出多行

示例 2：返回多行，一次一行

在 FreeRTOS+CLI 中注册的每个命令都有自己的帮助字符串。帮助字符串是演示如何使用命令的一行文本。FreeRTOS+CLI 包含一个“help”命令，该命令返回所有帮助字符串，向用户提供可用的命令列表以及如何使用每个命令的说明。示例 2 展示了帮助命令推荐的实现。与示例 1 所有输出都是一次生成不同的是，示例 2 一次只生成一行。注意，此函数是不可重入的。

```
/* This function implements the behaviour of a command, so must have the correct
prototype. */
static BaseType_t prvHelpCommand( int8_t *pcWriteBuffer,
                                  size_t xWriteBufferLen,
                                  const int8_t *pcCommandString )
{/* Executing the "help" command will generate multiple lines of text, but this
function will only output a single line at a time.  Therefore, this function is
called multiple times to complete the processing of a single "help" command.  That
means it has to remember which help strings it has already output, and which
still remain to be output.  The static pxCommand variable is used to point to the
next help string that needs outputting. */
static const xCommandLineInputListItem *pxCommand = NULL;
signed BaseType_t xReturn;
```

```
        if( pxCommand == NULL )
        {
            /* pxCommand is NULL in between executions of the "help" command, so
if
            it is NULL on entry to this function it is the start of a new "help" command
            and the first help string is returned.    The following line points
pxCommand
            to the first command registered with FreeRTOS+CLI. */
            pxCommand = &xRegisteredCommands;
        }

        /* Output the help string for the command pointed to by pxCommand, taking
        care not to overflow the output buffer. */
        strncpy( pcWriteBuffer,
                 pxCommand->pxCommandLineDefinition->pcHelpString,
                 xWriteBufferLen );

        /* Move onto the next command in the list, ready to output the help string
        for that command the next time this function is called. */
        pxCommand = pxCommand->pxNext;

        if( pxCommand == NULL )
        {
            /* If the next command in the list is NULL, then there are no more
            commands to process, and pdFALSE can be returned. */
            xReturn = pdFALSE;
        }
        else
        {
            /* If the next command in the list is not NULL, then there are more
            commands to process and therefore more lines of output to be generated.
            In this case pdTRUE is returned. */
            xReturn = pdTRUE;
        }

        return xReturn;
    }
```

示例 **2**：产生多行输出，一次输出一行

示例 3：具有固定数量参数的命令

有些命令接受参数。例如，文件系统“copy”命令需要源文件的名称和目标文件的名称。示例 3 是一个复制命令的框架，它被用来演示如何访问和使用命令参数。

注意，如果此命令在注册时声明为接受两个参数，则 FreeRTOS+CLI 在无法确定是否提供了两个参数之前是不会调用该命令的。

```
/* This function implements the behaviour of a command, so must have the correct
prototype. */
static BaseType_t prvCopyCommand( int8_t *pcWriteBuffer,
                                  size_t xWriteBufferLen,
                                  const int8_t *pcCommandString )
{
int8_t *pcParameter1, *pcParameter2;
BaseType_t xParameter1StringLength, xParameter2StringLength, xResult;

    /* Obtain the name of the source file, and the length of its name, from
    the command string. The name of the source file is the first parameter. */
    pcParameter1 = FreeRTOS_CLIGetParameter
                   (
                     /* The command string itself. */
                     pcCommandString,
                     /* Return the first parameter. */
                     1,
                     /* Store the parameter string length. */
                     &xParameter1StringLength
                   );

    /* Obtain the name of the destination file, and the length of its name. */
    pcParameter2 = FreeRTOS_CLIGetParameter( pcCommandString,
                                             2,
                                             &xParameter2StringLength );

    /* Terminate both file names. */
    pcParameter1[ xParameter1StringLength ] = 0x00;
    pcParameter2[ xParameter2StringLength ] = 0x00;

    /* Perform the copy operation itself. */
    xResult = prvCopyFile( pcParameter1, pcParameter2 );

    if( xResult == pdPASS )
    {
        /* The copy was successful.    There is nothing to output. */
        *pcWriteBuffer = NULL;
    }
    else
    {
        /* The copy was not successful.    Inform the users. */
        snprintf( pcWriteBuffer, xWriteBufferLen, "Error during copy\r\n\r\n" );
```

```
    }

    /* There is only a single line of output produced in all cases.  pdFALSE is
    returned because there is no more output to be generated. */
    return pdFALSE;
}
```

示例 3：访问和使用命令参数

示例 4：可变参数数量的命令

示例 4 演示如何创建和实现接受可变参数的命令。FreeRTOS+CLI 不会检查所提供的参数的数量，而命令的实现只是每次回显一个参数。例如，如果已分配的命令字符串为“echo_properties”，则如果用户输入：

“echo_properties one two three four”

则生成的输出将是：

参数为：

1：one

2：two

3：three

4：four

```
static BaseType_t prvParameterEchoCommand(      int8_t *pcWriteBuffer,
                                                size_t xWriteBufferLen, c
                                                onst int8_t *pcCommandString )

{
int8_t *pcParameter;
BaseType_t lParameterStringLength, xReturn;
/* Note that the use of the static parameter means this function is not reentrant. */
static BaseType_t lParameterNumber = 0;

    if( lParameterNumber == 0 )
    {
        /* lParameterNumber is 0, so this is the first time the function has been
        called since the command was entered.  Return the string "The parameters
        were:" before returning any parameter strings. */
        sprintf( pcWriteBuffer, "The parameters were:\r\n" );

        /* Next time the function is called the first parameter will be echoed
        back. */
        lParameterNumber = 1L;

        /* There is more data to be returned as no parameters have been echoed
        back yet, so set xReturn to pdPASS so the function will be called again. */
        xReturn = pdPASS;
```

```
    }
    else
    {
        /* lParameter is not 0, so holds the number of the parameter that should
        be returned.  Obtain the complete parameter string. */
        pcParameter = ( int8_t * ) FreeRTOS_CLIGetParameter
                                    (
                                        /* The command string itself. */
                                        pcCommandString,
                                        /* Return the next parameter. */
                                        lParameterNumber,
                                        /* Store the parameter string length. */
                                        &lParameterStringLength
                                    );

        if( pcParameter != NULL )
        {
            /* There was another parameter to return.  Copy it into pcWriteBuffer.
            in the format "[number]: [Parameter String". */
            memset( pcWriteBuffer, 0x00, xWriteBufferLen );
            sprintf( pcWriteBuffer, "%d: ", lParameterNumber );
            strncat( pcWriteBuffer, pcParameter, lParameterStringLength );
            strncat( pcWriteBuffer, "\r\n", strlen( "\r\n" ) );

            /* There might be more parameters to return after this one, so again
            set xReturn to pdTRUE. */
            xReturn = pdTRUE;
            lParameterNumber++;
        }
        else
        {
            /* No more parameters were found.  Make sure the write buffer does
            not contain a valid string to prevent junk being printed out. */
            pcWriteBuffer[ 0 ] = 0x00;

            /* There is no more data to return, so this time set xReturn to
            pdFALSE. */
            xReturn = pdFALSE;

            /* Start over the next time this command is executed. */
            lParameterNumber = 0;
        }
    }
```

```
    return xReturn;
}
```

示例 4：访问可变数量的参数

2.2 命令参数

FreeRTOS_CLIGetParameter()

FreeRTOS_CLI.h

```
const uint8_t *FreeRTOS_CLIGetParameter( const int8_t *pcCommandString,
                                         uint8_t ucWantedParameter,
                                         uint8_t *pucParameterStringLength )
```

有些命令使用参数。例如，文件系统“copy”命令需要源文件的名称和目标文件的名称。本章节描述了一个名为 FreeRTOS_CLIGetParameters()的辅助函数，它由 FreeRTOS+CLI 提供，以简化输入参数解析。

FreeRTOS_CLIGetParameters()以完整的命令字符串和被请求参数的位置作为输入，并生成指向请求参数的开始和以字节为单位的参数字符串长度的指针作为输出。

参数：

pcCommandString	指向整个命令字符串的指针，由用户输入。
ucWantedParameter	被请求的参数在命令字符串中的位置。例如，如果输入命令是“Copy[源文件][目的文件]”，则将 ucwanted 参数设置为 1，以请求源文件参数的名称和长度。将 ucwanted 参量设置为 2，以请求目的文件参数的名称和长度。
pucParameterStringLength	被请求的参数的字符串长度在 *pucParameterStringLength 中返回。例如，如果参数文本是“filename.txt”，那么 *pucParameterStringLength 将被设置为 12，因为字符串中有 12 个字符。

返回：

返回指向被请求参数的开始的指针。例如，如果完整的命令字符串是“copy file1.txt file2.txt”，而 ucwantedParameters 是 2，则 FreeRTOS_CLIGetParameters()将返回一个指向“file2.txt”的‘f’的指针。

2.3 注册一条命令

FreeRTOS_CLIRegisterCommand()

FreeRTOS_CLI.h

```
BaseType_t FreeRTOS_CLIRegisterCommand( CLI_Command_Definition_t
*pxCommandToRegister )
```

本章节描述 FreeRTOS_CLIRegisterCommand()，它是用于向 FreeRTOS+CLI 注册命令的 API 函数。通过将实现命令行为的函数与文本字符串相关联，并将关联告知 FreeRTOSCLI，可以注册该命令。然后，每次输入命令文本字符串时，FreeRTOS+CLI 将自动运行该函数。

注意：出现在代码中的 FreeRTOS_CLIRegisterCommand()原型接受一个指向 CLI_Command_Definition_t 类型的 Const 结构的 Const 指针。Const 限定符在这里被移除，以使原型更易于阅读。

参数：

pxCommandToRegister	正在注册的命令，该命令由 CLI_Command_Definition_t 类型的结构定义。结构如下表所述。

返回：

如果命令已成功注册，则返回 pdPASS。

如果由于创建新列表项可用的 FreeRTOS 堆不足，无法注册命令，则返回 pdFAIL。

CLI_Command_Definition_t

命令由 CLI_Command_Definition_t 类型的结构定义。结构如下所示。代码中的注释描述了结构成员。

```
typedef struct xCLI_COMMAND_DEFINITION
{
    /* 命令行输入字符串。这是用户为运行命令而输入的字符串。
       例如，FreeRTOS+CLI help 函数使用字符串“help”。如果
       用户键入“help”，则执行“help”命令。 */
    const int8_t * const pcCommand;

    /* 描述命令及其预期参数的字符串。这是执行 Help 命令时输出的字符串。字符
串必须以命令本身开始，并以“\r\n”结尾。例如，Help 命令本身的帮助字符串是：“help:
Returns a list of all the commands\r\n” */
    const int8_t * const pcHelpString;

    /* 指向实现命令行为的函数的指针(实际上是函数名)*/
    const pdCOMMAND_LINE_CALLBACK pxCommandInterpreter;

    /* 命令所需的参数。只有在命令行中输入的参数数量与此编号匹配时，
  FreeRTOS+CLI 才会执行该命令。*/
```

```
        int8_t cExpectedNumberOfParameters;
    } CLI_Command_Definition_t;
```

CLI_Command_Definition_t 结构体

示例

一个 FreeRTOS+CLI 演示实现了一个文件系统“del”命令。命令定义如下。

```
static const CLI_Command_Definition_t xDelCommand =
{
    "del",
    "del <filename>: Deletes <filename> from the disk\r\n",
    prvDelCommand,
    1
};
```

文件系统“**del**”命令结构

注册此命令后：

- 每次用户键入“del”时都执行 prvDelCommand()。
- 用户键入“help”时,“**del <filename>: Deletes <filename> from the disk\r\n**”会被输出以描述 del 命令。
- del 命令需要一个参数(被删除文件的名称)。如果输入参数的数量不是 1，则 FreeRTOS+CLI 将输出一个错误字符串，而不执行 prvDelCommand()。

然后调用以下函数在 FreeRTOS+CLI 中注册 del 命令：

```
FreeRTOS_CLIRegisterCommand( &xDelCommand );
```

使用 **FreeRTOS+CLI** 注册 **xDelCommand** 结构体

2.4 命令解释器

FreeRTOS_CLI.h

```
BaseType_t FreeRTOS_CLIProcessCommand( int8_t *pcCommandInput,
                                       int8_t *pcWriteBuffer,
                                       size_t xWriteBufferLen    );
```

本章节描述 FreeRTOS_CLIProcessCommand()函数。FreeRTOS_CLIProcessCommand()是一个 API 函数，它接受用户在命令提示符下输入的字符串，如果字符串匹配已注册的命令，则执行实现命令行为的函数。

参数：

pcCommandInput	完整的输入字符串，与用户在命令提示符(可能是串口控制台、键盘、telnet 客户端或其他用户输入客户端)中输入的字符串完全相同。
pcWriteBuffer	如果 pcCommandInput 不包含正确格式的命令，则 FreeRTOS_CLIProcessCommand()将向 pcWriteBuffer 缓冲区输出一个空的终止错误消息。 如果 pcCommandInput 包含一个正确格式的命令，则 FreeRTOS_CLIProcessCommand()将执行实现命令行为的函数，该函数将其生成的输出放入 pcWriteBuffer 缓冲区。
xWriteBufferLen	它是 pcWriteBuffer 参数所指向的缓冲区的大小。将超过 xWriteBufferLen 字符写入 pcWriteBuffer 将导致缓冲区溢出。

返回：

FreeRTOS_CLIProcessCommand()执行实现命令行为的函数，并返回它执行的函数返回的值。这些值在章节 2.1“实现一个命令”中进行了描述。

2.5 创建一个 FreeRTOS+CLI 任务

本章节描述了如何在提供了输入输出(IO)服务和 FreeRTOS+CLI 任务的情况下将 FreeRTOS+CLI 移植到实际硬件上。

输入输出服务

命令行接口从输入接收字符，并将字符写入输出。如何实现这一点的低层次细节取决于所使用的微控制器和微控制器提供的接口。

这里有一个 FreeRTOS+CLI 特性演示程序，它使用 FreeRTOS+IO 的 FreeRTOS_read()和 FreeRTOS_write()API 函数向 UART 提供必要的输入和输出。它创建的命令行接口使用标准的哑终端程序(如超级终端)访问。还有另一个 FreeRTOSCLI 特性演示，它使用 TCP/IP 套接字接口提供必要的输入和输出。它创建的命令行接口使用 telnet 客户端访问。在这两种情况下，运行 FreeRTOS+CLI 代码的 FreeRTOS 任务的结构都是相似的，如下所示。

FreeRTOS+CLI 任务示例

下面的源代码实现了管理 FreeRTOS+CLI 命令解释器接口的任务。FreeRTOS+ IO 的 FreeRTOS_read()和 FreeRTOS_write()API 函数用于提供 IO 接口。假设 FreeRTOS+ IO 描述符已经打开并配置为使用中断驱动的字符队列传输模式。

该任务使用 FreeRTOS+CLI FreeRTOS_CLIProcessCommand()API 函数。
源代码中的注释提供了更多信息。注意，此函数是不可重入的。

```
#define MAX_INPUT_LENGTH        50
#define MAX_OUTPUT_LENGTH       100

static const int8_t * const pcWelcomeMessage =
   "FreeRTOS command server.\r\nType Help to view a list of registered commands.\r\n";

void vCommandConsoleTask( void *pvParameters )
{
Peripheral_Descriptor_t xConsole;
int8_t cRxedChar, cInputIndex = 0;
BaseType_t xMoreDataToFollow;
/* 输入和输出缓冲区被声明为静态的，以使它们远离堆栈*/
static int8_t pcOutputString[ MAX_OUTPUT_LENGTH ],
pcInputString[ MAX_INPUT_LENGTH ];

    /* 此代码假定已打开和配置了作为控制台使用的外围设备，并将其作为任务参数传递到任务中。将任务参数强制转换为正确的类型。 */
    xConsole = ( Peripheral_Descriptor_t ) pvParameters;

    /* 发送欢迎信息给用户，让用户知道已经连接上了控制台。 */
    FreeRTOS_write( xConsole, pcWelcomeMessage, strlen( pcWelcomeMessage ) );

    for( ;; )
    {
        /*此实现一次只读取一个字符。在阻塞状态下等待，直到接收到一个字符。*/
        FreeRTOS_read( xConsole, &cRxedChar, sizeof( cRxedChar ) );

        if( cRxedChar == '\n' )
        {
            /* 接收到一个换行符，因此输入命令字符串是完整的，可以处理。发送一个行分隔符，只是为了使输出更容易阅读。 */
            FreeRTOS_write( xConsole, "\r\n", strlen( "\r\n" );

            /*命令解释器会被反复调用，直到它返回 pdFALSE 为止。请参阅章节 2.1“实现一个命令”文档，以了解为什么会出现这种情况。*/
            do
            {
                /*将命令字符串发送到命令解释器。命令解释器生成的任何输出都将放在 pcOutputString 缓冲区中。*/
                xMoreDataToFollow = FreeRTOS_CLIProcessCommand
                                    (
```

```
                                        pcInputString,      /* The command string.*/
                                        pcOutputString,     /* The output buffer. */
                                        MAX_OUTPUT_LENGTH/* The size of the output
buffer. */
                                    );

                    /*将命令解释器生成的输出写入控制台。 */
                    FreeRTOS_write( xConsole, pcOutputString, strlen( pcOutputString ) );

                } while( xMoreDataToFollow != pdFALSE );

                /* 输入命令生成的所有字符串都已发送。命令的处理已经完成。清除输入
字符串，准备接收下一个命令。 */
                cInputIndex = 0;
                memset( pcInputString, 0x00, MAX_INPUT_LENGTH );
            }
            else
            {
                /* if()子句在收到换行符后执行处理。如果收到任何其他字符，则此 else
子句执行处理。*/

                if( cRxedChar == '\r' )
                {
                    /* Ignore carriage returns. */
                }
                else if( cRxedChar == '\b' )
                {
                    /* Backspace 被按下了。删除输入缓冲区中的最后一个字符——如果
有的话。*/
                    if( cInputIndex > 0 )
                    {
                        cInputIndex--;
                        pcInputString[ cInputIndex ] = '\0';
                    }
                }
                else
                {
                    /* 输入了一个字符。它不是一个新的行、Backspace 或回车，因此它
被接受为输入的一部分并放置到输入缓冲区中。当输入\n 时，完整的字符串将传递给命
令解释器。*/
                    if( cInputIndex < MAX_INPUT_LENGTH )
                    {
                        pcInputString[ cInputIndex ] = cRxedChar;
                        cInputIndex++;
```

```
                }
            }
        }
    }
}
```

实现 FreeRTOS+CLI 命令控制台的任务示例

2.6 缓冲区管理

FreeRTOS_CLI.h

```
int8_t *FreeRTOS_CLIGetOutputBuffer( void );
```

本章节描述了可选的 FreeRTOS_CLIGetOutputBuffer()函数。

命令解释器实现需要一个输出缓冲区，用于保存通过运行命令生成的任何输出。

如果 FreeRTOS+CLI 用于实现单个命令解释器接口，则可以在本地为执行 FreeRTOS_CLIProcessCommand()API 函数的任务或文件定义输出缓冲区。

如果使用 FreeRTOSCLI 在多个接口(例如 UART 和 TCP/IP 套接字)上实现命令解释器，那么两个接口可以以相同的方式提供自己的输出缓冲区。但是，如果一次只使用其中一个接口，则可以通过两个接口共享一个输出缓冲区来节约 RAM。FreeRTOS_CLIGetOutputBuffer()函数的提供使得这个实现变得更容易。

参数：

无

返回：

FreeRTOS_CLIGetOutputBuffer()只返回在 FreeRTOS+CLI 代码中声明的输出缓冲区的地址——无需命令接口实现来声明自己的地址。

缓冲区的大小由 configCOMMAND_INT_MAX_Output_Size 常数定义，每当使用 FreeRTOS+CLI 时，都应该在 FreeRTOSConfig.h 中定义该常数。

如果所有命令解释器接口都使用自己的本地定义缓冲区，并且不使用 FreeRTOS_CLIGetOutputBuffer()API 函数，则 configCOMMAND_INT_MAX_OUTPUT_SIZE 应设置为 1 以最小化 RAM 的使用。