**实施命令**

FreeRTOS + CLI是一个可扩展的框架，允许应用程序编写者定义和注册他们自己的命令行输入命令。本页描述如何编写实现命令行为的函数。

**功能输入和输出**

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）;**

以下是对调用该函数时将传递给该函数的参数以及必须返回的值的说明。

**参数：**

|  |  |
| --- | --- |
| *pcWriteBuffer* | 这是应将任何生成的输出写入其中的缓冲区。例如，如果函数只是要返回固定的字符串“ Hello World”，则将该字符串写入pcWriteBuffer。输出**必须始终**为null终止。 |
| *xWriteBufferLen* | 这是pcWriteBuffer参数指向的缓冲区的大小。向pcWriteBuffer中写入多个xWriteBufferLen字符将导致缓冲区溢出。 |
| *pcCommandString* | 指向整个命令字符串的指针。可以访问整个命令字符串，从而使函数实现可以提取命令参数（如果有）。FreeRTOS + CLI提供的[辅助函数](https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_CLI/FreeRTOS_Plus_CLI_Accessing_Command_Line_Parameters.html)可以接受命令字符串并返回命令参数-因此不需要显式的字符串解析。本页提供了示例。 |

**返回值：**

执行某些命令将导致产生多行输出。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，则输出可能如下所示：

**file1.txt**

**file2.txt**

**file3.txt**

为了最大程度地减少RAM使用率并确保确定性地使用RAM，FreeRTOS + CLI允许实现命令行为的函数一次输出一行。函数返回值用于指示输出行是输出的结尾还是是否有更多行要生成。

如果生成的输出在输出的末尾，则返回**pdFALSE**，这意味着不再有任何行可以生成，并且命令执行已完成。

如果返回的输出不是输出的结尾，则返回**pdTRUE**，并且在命令执行完成之前仍然需要生成一行或多行。

继续示例“ dir”命令，该命令输出三个文件名：

1. 第一次调用实现dir命令的函数时，可能仅输出第一行（file1.txt）。如果这样做，函数必须返回pdTRUE来指示还有更多行。

1. 第二次调用执行dir命令的函数时，可能仅输出第二行（file2.txt）。如果这样做，函数必须再次返回pdTRUE，以指示后面还有更多行。

1. 第三次调用实现dir命令的函数时，仅输出第三行（file3.txt）。这次，没有更多的行要输出，因此该函数必须返回pdFALSE。

或者，如果有足够的RAM，并且在xWriteBufferLen中传递的值足够大，则可能同时返回了所有三行-在这种情况下，函数必须在首次执行时返回pdFALSE。

每次执行命令时，FreeRTOS + CLI都会重复调用实现命令行为的函数，直到该函数返回pdFALSE。

**例子**

下面提供了以下示例：

1. 一个不带参数并返回单个字符串的命令。
2. 该命令不带参数，一次返回一行，返回多个字符串。
3. 需要固定数量参数的命令。
4. 该命令接受可变数量的参数，并且一次返回一行可变数量的字符串。

**示例1：不带参数的命令**

FreeRTOS vTaskList（）API函数生成一个表，其中包含有关每个任务状态的信息。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演示了一次简单的情况，即一次输出整个表。代码中的注释提供了更多说明。

**/ \*此函数实现命令的行为，因此必须具有正确的**

**原型。\* /**

**静态BaseType\_t prvTaskStatsCommand（int8\_t \* pcWriteBuffer，**

**size\_t xWriteBufferLen，**

**const int8\_t \* pcCommandString）**

**{**

**/ \*为简单起见，此函数假定输出缓冲区足够大，**

**可以容纳通过执行vTaskList（）API函数生成的所有文本，**

**因此不使用xWriteBufferLen参数。\* /**

**（void）xWriteBufferLen;**

**/ \* pcWriteBuffer直接用作vTaskList（）参数，因此**

**通过执行vTaskList（）生成的表将直接写入输出中**

**缓冲。\* /**

**vTaskList（pcWriteBuffer + strlen（pcHeader））;**

**/ \*整个表被直接写入输出缓冲区。**

**该命令的执行已完成，因此返回pdFALSE。\* /**

**返回pdFALSE;**

**}**

**示例1：一次输出多行**

**示例2：一次返回多行**

在FreeRTOS + CLI中注册的每个命令都有其自己的帮助字符串。帮助字符串是一行文本，演示了如何使用命令。FreeRTOS + CLI包含一个“帮助”命令，该命令返回所有帮助字符串，为用户提供可用命令的列表以及有关如何使用每个命令的说明。示例2显示了help命令的实现。与示例1一次性生成所有输出的示例1不同，示例2一次仅生成一行。请注意，此函数**不可**重入。

**/ \*此函数实现命令的行为，因此必须具有正确的**

**原型。\* /**

**static BaseType\_t prvHelpCommand（int8\_t \* pcWriteBuffer，**

**size\_t xWriteBufferLen，**

**const int8\_t \* pcCommandString）**

**{**

**/ \*执行“ help”命令将生成多行文本，但是此**

**函数一次仅输出一行。因此，该函数被**

**多次调用以完成单个“帮助”命令的处理。这**

**意味着它必须记住它已经输出了哪些帮助字符串，还有哪些**

**仍需要输出。静态pxCommand变量用于指向**

**下一个需要输出的帮助字符串。\* /**

**静态const xCommandLineInputListItem \* pxCommand = NULL;**

**带签名的BaseType\_t xReturn;**

**if（pxCommand == NULL）**

**{**

**/ \* pxCommand在执行“ help”命令之间为NULL，因此，如果**

**此函数输入时为NULL ，则它是新的“ help”命令的开始，**

**并且第一个帮助字符串为回到。下一行将pxCommand**

**指向在FreeRTOS + CLI中注册的第一个命令。\* /**

**pxCommand =＆xRegisteredCommands;**

**}**

**/ \*输出pxCommand指向的命令的帮助字符串，**

**注意不要溢出输出缓冲区。\* /**

**strncpy（pcWriteBuffer，**

**pxCommand-> pxCommandLineDefinition-> pcHelpString，**

**xWriteBufferLen）;**

**/ \*移至列表中的下一个命令，准备**

**在下次调用此函数时输出该命令的帮助字符串。\* /**

**pxCommand = pxCommand-> pxNext;**

**if（pxCommand == NULL）**

**{**

**/ \*如果列表中的下一个命令为NULL，则不再有**

**要处理的命令，并且可以返回pdFALSE。\* /**

**xReturn = pdFALSE;**

**}**

**else**

**{**

**/ \*如果列表中的下一个命令不为NULL，则将**

**处理更多命令，因此将生成更多行输出。**

**在这种情况下，将返回pdTRUE。\* /**

**xReturn = pdTRUE;**

**}**

**return xReturn;**

**}**

**示例2：生成多行输出，一次生成一行**

**示例3：带有固定数量参数的命令**

一些命令带有参数。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是复制命令的框架，并提供了用于演示如何访问和使用命令参数的框架。

注意，如果该命令在注册时声明为带有两个参数，除非提供了恰好两个参数，否则FreeRTOS + CLI甚至不会调用该命令。

**/ \*此函数实现命令的行为，因此必须具有正确的**

**原型。\* /**

**静态BaseType\_t prvCopyCommand（int8\_t \* pcWriteBuffer，**

**size\_t xWriteBufferLen，**

**const int8\_t \* pcCommandString）**

**{**

**int8\_t \* pcParameter1，\* pcParameter2;**

**BaseType\_t xParameter1StringLength，xParameter2StringLength，xResult;**

**/ \*从**

**命令字符串中获取源文件的名称及其名称的长度。源文件的名称是第一个参数。\* /**

**pcParameter1 =** [**FreeRTOS\_CLIGetParameter**](https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_CLI/FreeRTOS_Plus_CLI_Accessing_Command_Line_Parameters.html)

**（**

**/ \*命令字符串本身。\* /**

**pcCommandString，**

**/ \*返回第一个参数。\* /**

**1，**

**/ \*存储参数字符串的长度。\* /**

**＆xParameter1StringLength**

**）;**

**/ \*获取目标文件的名称及其名称的长度。\* /**

**pcParameter2 = FreeRTOS\_CLIGetParameter（pcCommandString，**

**2，**

**＆xParameter2StringLength）;**

**/ \*终止两个文件名。\* /**

**pcParameter1 [xParameter1StringLength] = 0x00;**

**pcParameter2 [xParameter2StringLength] = 0x00;**

**/ \*本身执行复制操作。\* /**

**xResult = prvCopyFile（pcParameter1，pcParameter2）;**

**if（xResult == pdPASS）**

**{**

**/ \*复制成功。没有任何输出。\* /**

**\* pcWriteBuffer = NULL;**

**}**

**else**

**{**

**/ \*复制失败。通知用户。\* /**

**snprintf（pcWriteBuffer，xWriteBufferLen，“在复制期间出错”）；**

**}**

**/ \*在所有情况下，只产生一行输出。**

**因为没有更多的输出要生成，所以返回pdFALSE 。\* /**

**返回pdFALSE;**

**}**

**示例3：访问和使用命令参数**

**示例4：带有可变数量参数的命令**

示例4演示了如何创建和实现一个接受可变数量参数的命令。FreeRTOS + CLI不会检查提供的参数数量，并且该命令的实现只是一次回显参数。例如，如果分配的命令字符串是“ echo\_parameters”，则用户输入：

“ echo\_parameters一二三四”

然后生成的将是：

参数为：  
1 ：1：  
2：2：  
3：3  
4：4

**静态BaseType\_t prvParameterEchoCommand（int8\_t \* pcWriteBuffer，**

**size\_t xWriteBufferLen，**

**int8\_t \* pcCommandString）**

**{**

**int8\_t \* pcParameter;**

**BaseType\_t lParameterStringLength，xReturn;**

**/ \*请注意，使用static参数意味着该函数不可重入。\* /**

**静态BaseType\_t lParameterNumber = 0;**

**if（lParameterNumber == 0）**

**{**

**/ \* lParameterNumber为0，因此这是**

**自输入命令以来第一次调用该函数。返回**

**任何参数字符串之前，返回字符串“ The parameters are：”。\* /**

**sprintf（pcWriteBuffer，“参数为：rn”）;**

**/ \*下次调用该函数时，将**

**回显第一个参数。\* /**

**lParameterNumber = 1L;**

**/ \*还有更多数据要返回，因为尚未回显任何参数**

**，因此将xReturn设置为pdPASS以便再次调用该函数。\* /**

**xReturn = pdPASS;**

**}**

**else**

**{**

**/ \* lParameter不为0，因此保存应**

**返回的参数编号。获取完整的参数字符串。\* /**

**pcParameter =（int8\_t \*）FreeRTOS\_CLIGetParameter**

**（**

**/ \*命令字符串本身\* /**

**pcCommandString，**

**/ \*返回下一个参数。\* /**

**lParameterNumber，**

**/ \*存储参数字符串的长度。\* /**

**＆lParameterStringLength**

**）;**

**if（pcParameter！= NULL）**

**{**

**/ \*还有另一个参数要返回。将其复制到pcWriteBuffer中。**

**格式为“ [数字]：[参数字符串”。\* /**

**memset（pcWriteBuffer，0x00，xWriteBufferLen）;**

**sprintf（pcWriteBuffer，“％d：”，lParameterNumber）;**

**strncat（pcWriteBuffer，pcParameter，lParameterStringLength）;**

**strncat（pcWriteBuffer，“ rn”，strlen（“ rn”））;**

**/ \*在此之后可能会有更多参数要返回，因此再次**

**将xReturn设置为pdTRUE。\* /**

**xReturn = pdTRUE;**

**lParameterNumber ++;**

**}**

**其他**

**{**

**/ \*找不到更多参数。确保写缓冲区中**

**没有有效的字符串，以防止垃圾内容被打印出来。\* /**

**pcWriteBuffer [0] = 0x00;**

**/ \*没有更多数据要返回，因此这次将xReturn设置为**

**pdFALSE。\* /**

**xReturn = pdFALSE;**

**/ \*从下次执行该命令开始。\* /**

**lParameterNumber = 0;**

**}**

**}**

**返回xReturn;**

**}**

**示例4：访问可变数量的参数**