1.2 命令处理框架设计 (一)- 解析命令_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 1.2 命令处理框架设计 (一)– 解析命令涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

- 输入的命令是一个完整字符串，比如复制“cp test.txt test1.txt”, 在实际实现业务逻辑时需要进行拆分
- 具体在解析字符串的步骤如下:
    - step 1 : 设计自定义的数据结构存储拆分之后的命名信息
    - step 2 : 使用 strtok 函数对命令字符串进行拆分, 并存储到自定义数据结构中
    - step 3 : 按照命令名字分发到具体模块中执行
- 对于解析之后的字符串，需要保存到自定义的数据结构中
    - 命令名称
    - 参数个数
    - 参数列表
- 具体的数据结构设计如下:

```
#define SZ_NAME 8
#define SZ_ARG 32
#define SZ_COUNT 2

#include <stdio.h>
#include <string.h>

#define DEBUG

typedef struct command{
    char cmd_name[SZ_NAME];
    char cmd_arg_list[SZ_COUNT][SZ_ARG];
    int  cmd_arg_count;
}cmd_t;
```

- 数据结构初始化 调用 init_command_struct 函数, 具体实现如下:

```
void init_command_struct(cmd_t *pcmd)
{
    int i;

    memset(pcmd->cmd_name,0,SZ_NAME);

    for (i = 0;i < SZ_COUNT;i++){
        memset(pcmd->cmd_arg_list[i],0,SZ_ARG);
    }

    pcmd->cmd_arg_count = 0;
}
```

- 命令数据结构的调试打印 调用 print_command_info 函数，具体实现如下:

```
void print_command_info(cmd_t *pcmd)
{
    int i;

    printf("==================\n");
```

```
    printf("[DEBUG] cmd name : < %s >\n",pcmd->cmd_name);
    printf("[DEBUG] cmd arg count : < %d >\n",pcmd->cmd_arg_count);
    printf("[DEBUG] cmd arg list : ");

    for (i = 0;i < pcmd->cmd_arg_count;i++){
        printf(" %s ",pcmd->cmd_arg_list[i]);
    }

    printf("\n==================\n");
}
```

- 在 cmd_execute 函数中，定义命令数据结构，并进行初始化后，并进行调试

```
int cmd_execute( char *cmd_str)
{
    cmd_t command ;
    int ret;

    if (cmd_str == NULL)
        return -1;

    init_command_struct(&command);

#ifdef DEBUG
    print_command_info(&command);
#endif

    return 0;
}
```

- 命令的解析需要调用字符串处理函数 strtok 进行拆分
- strtok 函数具体信息如下:

函数头文件:

#include <string.h>

函数原型:

char *strtok(char *str, const char *delim);

函数功能:

根据指定的分割字符串进行分割

函数参数:

str：分割字符串的地址

delim：分割符

函数返回值:

成功：返回分割后字符串首地址

失败：返回 NULL

- 函数注意事项:
- - 第一次调用时，需要指定字符串的地址
  - 第二次调用时, 第一个参数可以填 NULL

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char str[] = "ABC 123 XYZ";
```

```
    char *first = NULL;
    char *other = NULL;

    first = strtok(str," ");

    printf(" first : %s\n",first);

    while((other = strtok(NULL," "))){
        printf(" other : %s\n",other);
    }
    return 0;
}

输出结果为 :
            ABC
            123
            XYZ
```

- 命令字符串通过 strtok 函数进行拆分后需要存储到自定义的数据结构

```
int cmd_parse(char *cmd_str,cmd_t *pcmd)
{
    char *p_cmd_name = NULL;
    char *p_cmd_arg = NULL;

    int index = 0;

    if (cmd_str == NULL || pcmd == NULL)
        return -1;

    p_cmd_name = strtok(cmd_str," ");

#ifdef DEBUG
    printf("[DEBUG]: cmd_name : %s\n",p_cmd_name);
#endif

    strcpy(pcmd->cmd_name,p_cmd_name);

    for(;;){
        p_cmd_arg = strtok(NULL," ");

        if (p_cmd_arg == NULL)
            break;

        strcpy(pcmd->cmd_arg_list[index++],p_cmd_arg);
    }

    pcmd->cmd_arg_count = index;

#ifdef DEBUG
    print_command_info(pcmd);
#endif

    return 0;
}
```

- 在实现了 cmd_parse 函数后，在 cmd_execute 函数中进行调用, 具体如下:

```
int cmd_execute( char *cmd_str)
{
    cmd_t command ;
    int ret;

    if (cmd_str == NULL)
        return -1;

    init_command_struct(&command);

    ret = cmd_parse(cmd_str,&command);
    if (ret == -1)
        return -1;

#ifdef DEBUG
    print_command_info(&command);
#endif


    return 0;
}
```

- 当命令行解析完成之后，则需要进行具体分发到各个模块具体执行, 这里调用 cmd_dispatch 函数, 具体实现如下:

```
int cmd_dispatch(cmd_t *pcmd)
{
    if (pcmd == NULL)
        return -1;

    if (strcmp(pcmd->cmd_name,"ls") == 0){

    }else if (strcmp(pcmd->cmd_name,"cp") == 0){

    }

    return 0;
}
```

- 在 cmd_execute 函数中调用 cmd_dispatch 函数

```
nt cmd_execute( char *cmd_str)
{
    cmd_t command ;
    int ret;

    if (cmd_str == NULL)
        return -1;

    init_command_struct(&command);

    ret = cmd_parse(cmd_str,&command);
    if (ret == -1)
        return -1;

#ifdef DEBUG
    print_command_info(&command);
#endif

    ret = cmd_dispatch(&command);
    if (ret == -1)
        return -1;

    return 0;
}
```