XRADIO FatFs Developer Guide

Revision 1.1

Oct 29, 2019

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-10-12	Initial Version
1.1	2019-10-29	Optimize Format

Table 1- 1 Revision History

Contents

Tables

Figures

1 概述

1.1 属性介绍

FatFs 是用于小型嵌入式系统的通用 FAT / exFAT 文件系统模块。FatFs 模块是按照 ANSI C（C89）编写的，并且与磁盘 I / O 层完全分开。因此，它独立于平台。它可以并入资源有限的小型微控制器中，例如 8051，PIC，AVR，ARM，Z80，RX 等。

特征：

- DOS / Windows 兼容的 FAT / exFAT 文件系统。
- 平台无关。易于移植。
- 程序代码和工作区的占用空间非常小。
- 支持以下各种配置选项：
  - ANSI / OEM 或 Unicode 中的长文件名。
  - exFAT 文件系统。
  - RTOS 的线程安全。
  - 多个卷（物理驱动器和分区）。
  - 可变扇区大小。
  - 多个代码页，包括 DBCS。
  - 只读，可选 API，I / O 缓冲区等。

1.2 系统组织结构

图 1-1 给出了嵌入式系统 FatFs 模块典型的关系依赖图。图中假设使用 SPI 接口访问 SD 卡，蓝色区域表示 FatFs 模块，注意 FatFs 模块并不包含也从不关心绿色区域的底层磁盘 IO 层。对于使用 SPI 访问 SD 卡的应用，FatFs 官网提供的例程中，有使用硬件 SPI 和模拟 SPI 的例程。

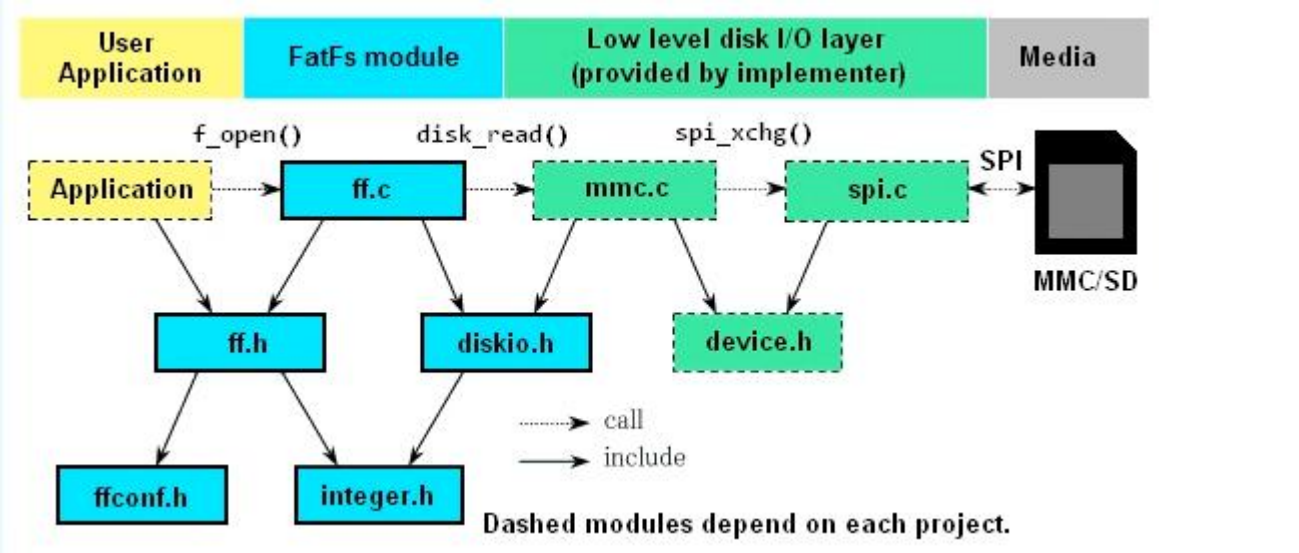

图 1-1 FatFs 关系依赖图

如果只有一个磁盘设备，并且使用 FatFs 提供的磁盘接口，则不需要其它模块，只管编写应用程序即可，如图 1-2(a)所示。如果使用多种不同接口，则需要编写另外一层软件，用来在驱动器和 FatFs 之间翻译命令和数据流，如图 1-2(b)所示。

注意图 1-2 中灰色部分属于底层磁盘 I/O 层，FatFs 从来不关心这一层是如何实现的！

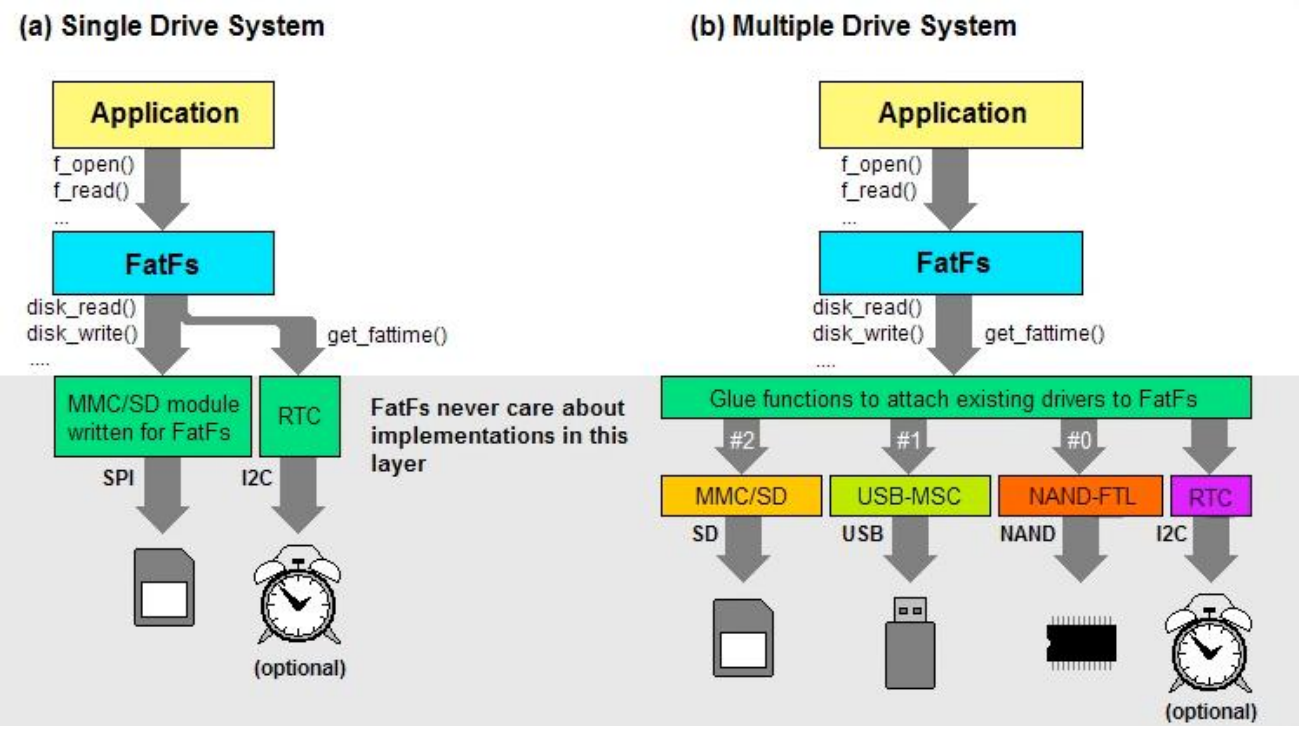

图 1-2 单个设备和多个设备系统

1.3 代码位置

1.3.1 模块源代码位置

表 1-1 模块源代码位置

文件名	位置
diskio.c、ff.c、fferrno.c、fferrno.h	sdk/src/fs/fatfs/
diskio.h、ff.h、ffconf.h、integer.h	sdk/include/fs/fatfs/

1.3.2 模块示例代码位置

表 1-2 模块示例代码位置

文件名	位置
main.c	sdk/project/example/fs/main.c
board_config.c	sdk/project/common/board/xradio_evb/board_config.c

2 模块结构体

2.1 FATFS 结构体

FATFS 结构（文件系统对象）持有单个逻辑驱动器的动态工作区。它由应用程序给定，并通过 f_mount 函数注册/注销到 FatFs 模块。必要时通过卷安装过程完成结构的初始化。应用程序不得修改此结构中的任何成员，否则 FAT 卷将被折叠。

```
typedef struct {
    BYTE    fs_type;      /* File system type (0:N/A) */
    BYTE    drv;          /* Physical drive number */
    BYTE    n_fats;           /* Number of FATs (1 or 2) */
    BYTE    wflag;        /* win[] flag (b0:dirty) */
    BYTE    fsi_flag;     /* FSINFO flags (b7:disabled, b0:dirty) */
    WORD    id;           /* File system mount ID */
    WORD    n_rootdir;        /* Number of root directory entries (FAT12/16) */
    WORD    csize;        /* Cluster size [sectors] */
#if _MAX_SS != _MIN_SS
    WORD    ssize;        /* Sector size (512, 1024, 2048 or 4096) */
#endif
#if _USE_LFN != 0
    WCHAR* lfnbuf;            /* LFN working buffer */
#endif
#if _FS_EXFAT
    BYTE*  dirbuf;            /* Directory entry block scratchpad buffer */
#endif
#if _FS_REENTRANT
    _SYNC_t    sobj;          /* Identifier of sync object */
#endif
#if !_FS_READONLY
    DWORD  last_clst;         /* Last allocated cluster */
    DWORD  free_clst;         /* Number of free clusters */
#endif
#if _FS_RPATH != 0
    DWORD  cdir;          /* Current directory start cluster (0:root) */
#if _FS_EXFAT
    DWORD  cdc_scl;       /* Containing directory start cluster (invalid when cdir
is 0) */
    DWORD  cdc_size;      /* b31-b8:Size of containing directory, b7-b0: Chain
status */
    DWORD  cdc_ofs;       /* Offset in the containing directory (invalid when cdir
is 0) */
#endif
#endif
    DWORD  n_fatent;      /* Number of FAT entries (number of clusters + 2) */
    DWORD  fsize;         /* Size of an FAT [sectors] */
    DWORD  volbase;       /* Volume base sector */
    DWORD  fatbase;       /* FAT base sector */
    DWORD  dirbase;       /* Root directory base sector/cluster */
    DWORD  database;      /* Data base sector */
    DWORD  winsect;       /* Current sector appearing in the win[] */
    BYTE   win[_MAX_SS]; /* Disk access window for Directory, FAT (and file data
```

```
at tiny cfg) */
} FATFS;
```

2.2 FIL 结构体

FIL 结构（文件对象）持有一个打开的文件的状态。它是由 f_open 创建和 f_close 丢弃。应用程序不得修改此结构中除 cltbl 以外的任何成员，否则 FAT 卷将被折叠。请注意，在 F_FS_TINY == 0 的情况下，在此结构中定义了扇区缓冲区，因此不应将该配置下的 FIL 结构定义为自动变量。

```
typedef struct {
    _FDID  obj;          /* Object identifier (must be the 1st member to detect
invalid object pointer) */
    BYTE   flag;         /* File status flags */
    BYTE   err;          /* Abort flag (error code) */
    FSIZE_t   fptr;         /* File read/write pointer (Zeroed on file open) */
    DWORD  clust;        /* Current cluster of fpter (invalid when fptr is 0) */
    DWORD  sect;         /* Sector number appearing in buf[] (0:invalid) */
#if !_FS_READONLY
    DWORD  dir_sect;     /* Sector number containing the directory entry */
    BYTE*  dir_ptr;      /* Pointer to the directory entry in the win[] */
#endif
#if _USE_FASTSEEK
    DWORD* cltbl;        /* Pointer to the cluster link map table (nulled on open,
set by application) */
#endif
#if !_FS_TINY
    BYTE   buf[_MAX_SS]; /* File private data read/write window */
#endif
} FIL;
```

2.3 FILINFO 结构体

FILINFO 结构包含有关 f_readdir，f_findfirst，f_findnext 和 f_stat 检索到的对象的信息。

```
typedef struct {
    FSIZE_t   fsize;        /* File size */
    WORD   fdate;        /* Modified date */
    WORD   ftime;        /* Modified time */
    BYTE   fattrib;      /* File attribute */
#if _USE_LFN != 0
    TCHAR  altname[13];         /* Altenative file name */
    TCHAR  fname[_MAX_LFN + 1];    /* Primary file name */
#else
    TCHAR  fname[13];       /* File name */
#endif
} FILINFO;
```

2.4 FRESULT 返回值

大多数 FatFs API 函数都以枚举类型 FRESULT 返回常见的结果代码。

```
typedef enum {
    FR_OK = 0,              /* (0) Succeeded */
    FR_DISK_ERR,         /* (1) A hard error occurred in the low level disk I/O layer
*/
    FR_INT_ERR,             /* (2) Assertion failed */
    FR_NOT_READY,        /* (3) The physical drive cannot work */
    FR_NO_FILE,             /* (4) Could not find the file */
    FR_NO_PATH,             /* (5) Could not find the path */
    FR_INVALID_NAME,     /* (6) The path name format is invalid */
    FR_DENIED,              /* (7) Access denied due to prohibited access or
directory full */
    FR_EXIST,            /* (8) Access denied due to prohibited access */
    FR_INVALID_OBJECT,      /* (9) The file/directory object is invalid */
    FR_WRITE_PROTECTED,     /* (10) The physical drive is write protected */
    FR_INVALID_DRIVE,       /* (11) The logical drive number is invalid */
    FR_NOT_ENABLED,         /* (12) The volume has no work area */
    FR_NO_FILESYSTEM,       /* (13) There is no valid FAT volume */
    FR_MKFS_ABORTED,     /* (14) The f_mkfs() aborted due to any problem */
    FR_TIMEOUT,             /* (15) Could not get a grant to access the volume within
defined period */
    FR_LOCKED,              /* (16) The operation is rejected according to the file
sharing policy */
    FR_NOT_ENOUGH_CORE,     /* (17) LFN working buffer could not be allocated */
    FR_TOO_MANY_OPEN_FILES, /* (18) Number of open files > _FS_LOCK */
    FR_INVALID_PARAMETER    /* (19) Given parameter is invalid */
} FRESULT;
```

3 模块接口

这里只列出些常用的接口，更多的接口请往 fatfs 官网（http://elm-chan.org/fsw/ff/00index_e.html）了解。

3.1 文件系统加/卸载

FRESULT f_mount (FATFS* fs, /* [IN] Filesystem object */ const TCHAR* path, /* [IN] Logical drive number */ BYTE opt /* [IN] Initialization option */)	
功能	在 FatFs 模块上加载/卸载一个工作区(文件系统对象) 。
参数	fs：指向要加载文件系统对象的指针。空指针将卸载文件系统对象。 path：待要加载/卸载的物理逻辑驱动号。 opt：初始化选项，0：不要现在加载，1：立即加载。
返回值	成功：FR_OK 失败：FR_INVALID_DRIVE，FR_DISK_ERR，FR_NOT_READY，FR_NOT_ENABLED，FR_NO_FILESYSTEM

由于 SDK 支持了自动加卸载文件系统的功能，加卸载文件系统不是通过 f_mount 接口，而是通过 fs_ctrl_mount、fs_ctrl_unmount 接口

int fs_ctrl_mount(enum fs_mnt_dev_type dev_type, uint32_t dev_id)	
功能	在 FatFs 模块上加载一个工作区(文件系统对象) 。
参数	dev_type：待挂载的设备类型，目前可取值 FS_MNT_DEV_TYPE_SDCARD。 dev_id：待挂载的设备 ID。
返回值	成功：0 失败：-1

int fs_ctrl_unmount(enum fs_mnt_dev_type dev_type, uint32_t dev_id)	
功能	在 FatFs 模块上卸载一个工作区(文件系统对象) 。

参数	dev_type：待卸载的设备类型，目前可取值 FS_MNT_DEV_TYPE_SDCARD。 dev_id：待卸载的设备 ID。
返回值	成功：0 失败：-1

3.2 文件打开/创建

```
FRESULT f_open (
   FIL* fp,                /* [OUT] Pointer to the file object structure */
   const TCHAR* path, /* [IN] File name */
   BYTE mode               /* [IN] Mode flags */
)
```

功能	打开或创建一个文件。
参数	fs：将被打开或创建的文件对象结构的指针。 path：指向待打开或创建的文件。 mode：模式标志，用于指定文件的访问和打开方法的类型。由以下标志的组合指定。

FA_READ	指定对对象的读取访问权限。可以从文件中读取数据。
FA_WRITE	指定对对象的写访问。数据可以写入文件。与 FA_READ 结合使用以进行读写访问。
FA_OPEN_EXISTING	打开文件。如果文件不存在，该功能将失败。（默认）
FA_CREATE_NEW	创建一个新文件。如果该文件存在，则函数失败并显示 FR_EXIST。
FA_CREATE_ALWAYS	创建一个新文件。如果该文件存在，它将被截断并覆盖。
FA_OPEN_ALWAYS	打开文件（如果存在）。如果没有，将创建一个新文件。
FA_OPEN_APPEND	与 FA_OPEN_ALWAYS 相同，只是将读/写指针设置为文件的结尾。

返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_NOT_READY，FR_NO_FILE，FR_NO_PATH，FR_INVALID_NAME，FR_DENIED，FR_EXIST，FR_INVALID_OBJECT，FR_WRITE_PROTECTED，FR_INVALID_DRIVE，FR_NOT_ENABLED，FR_NO_FILESYSTEM...

3.3 文件关闭

FRESULT f_close (FIL* fp /* [IN] Pointer to the file object */)	
功能	关闭一个打开的文件。
参数	fp：指向要关闭的文件对象结构的指针。
返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_INT_ERR，FR_INVALID_OBJECT，FR_TIMEOUT

3.4 文件读取数据

FRESULT f_read (FIL* fp, /* [IN] File object */ void* buff, /* [OUT] Buffer to store read data */ UINT btr, /* [IN] Number of bytes to read */ UINT* br /* [OUT] Number of bytes read */)	
功能	从文件中读取数据。
参数	fp：指向打开的文件对象结构的指针。 buff：指向缓冲区的指针，用于存储读取的数据。 btr：在 UINT 类型范围内要读取的字节数。 br：指向接收读取的字节数的 UINT 变量的指针。
返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_INT_ERR，FR_DENIED，FR_INVALID_OBJECT，FR_TIMEOUT
备注	该函数在读/写指针指向的位置开始从文件中读取数据。读/写指针随着读取的字节数而增加。函数成功后，应检查*br 以检测文件结尾。如果*br<btr，则表示读/写指针在读操作期间到达文件末尾。

3.5 文件写入数据

FRESULT f_write (FIL* fp, /* [IN] Pointer to the file object structure */ const void* buff, /* [IN] Pointer to the data to be written */ UINT btw, /* [IN] Number of bytes to write */ UINT* bw /* [OUT] Pointer to the variable to return number of bytes written */)	
功能	往文件写入数据。
参数	fp：指向打开的文件对象结构的指针。 buff：指向要写入的数据的指针。 btw：指定要在 UINT 类型范围内写入的字节数。 bw：指向 UINT 变量的指针，该变量接收写入的字节数。
返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_INT_ERR，FR_DENIED，FR_INVALID_OBJECT，FR_TIMEOUT
备注	该函数开始在读/写指针指向的位置将数据写入文件。读/写指针随着写入的字节数而增加。函数成功后，应检查*bw 以检测磁盘是否已满。如果*bw<btw，则意味着在写操作期间该卷已满。

3.6 文件/子目录删除

FRESULT f_unlink (const TCHAR* path /* [IN] Object name */)	
功能	从卷中删除文件或子目录。
参数	path：指向以空值结尾的字符串的指针，该字符串指定要删除的文件或子目录。
返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_INT_ERR，FR_NOT_READY，FR_NO_FILE，FR_NO_PATH，FR_INVALID_NAME，FR_DENIED，FR_WRITE_PROTECTED，FR_INVALID_DRIVE，FR_NOT_ENABLED，FR_NO_FILESYSTEM...

3.7 目录创建

FRESULT f_mkdir (const TCHAR* path /* [IN] Directory name */);	
功能	创建一个新目录。
参数	path：指向以空值结尾的字符串的指针，该字符串指定要创建的目录名称。
返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_INT_ERR，FR_NOT_READY，FR_NO_PATH，FR_INVALID_NAME， FR_DENIED，FR_EXIST，FR_WRITE_PROTECTED，FR_INVALID_DRIVE，FR_NOT_ENABLED， FR_NO_FILESYSTEM...

3.8 目录打开

FRESULT f_opendir (DIR* dp, /* [OUT] Pointer to the directory object structure */ const TCHAR* path /* [IN] Directory name */)	
功能	打开一个目录。
参数	dp：指向目录对象的指针以打开一个目录。 path：指向以空值结尾的字符串的指针，该字符串指定要打开的目录名称。
返回值	成功：FR_OK 失 败 ： FR_DISK_ERR ， FR_INT_ERR ， FR_NOT_READY ， FR_NO_PATH ， FR_INVALID_NAME ，FR_INVALID_DRIVE，FR_NOT_ENABLED，FR_NO_FILESYSTEM...
备注	f_opendir 功能打开一个 exsisting 目录，并为随后的目录对象使能 f_readdir 功能。

3.9 目录项读取

FRESULT f_readdir (DIR* dp, /* [IN] Directory object */

FILINFO* fno /* [OUT] File information structure */)	
功能	打开一个目录。
参数	dp：指向打开目录对象的指针。 fno：指向文件信息结构的指针，用于存储有关已读项目的信息。空指针会倒退目录的读取索引。
返回值	成功：FR_OK 失败：FR_DISK_ERR，FR_INT_ERR，FR_INVALID_OBJECT，FR_TIMEOUT，FR_NOT_ENOUGH_CORE
备注	f_readdir 函数读取目录项，信息有关的对象。可以通过 f_readdir 函数调用顺序读取目录中的项目。当所有目录项都已被读取并且没有要读取的项时，会将空字符串存储到 fno-> fname []中，而不会出现任何错误。将空指针赋予 fno 时，将回退目录对象的读取索引。

4 模块使用示例

4.1 模块使用流程

具体代码可参考 sdk/project/example/fs/main.c。