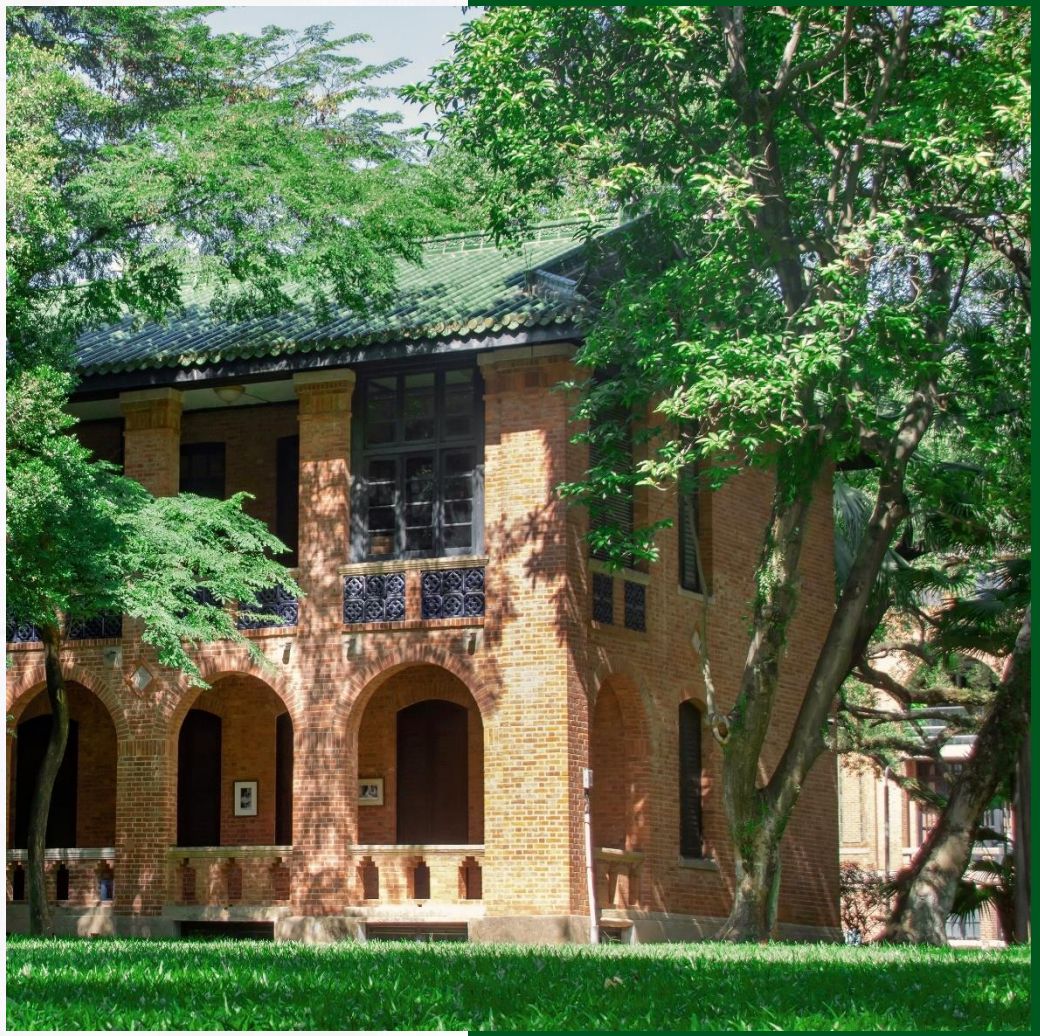
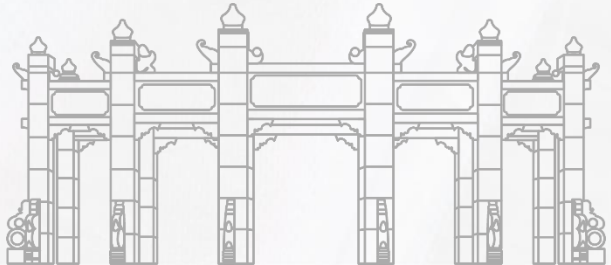
中山大學
SUN YAT-SEN UNIVERSITY

鸿蒙操作系统
文件系统





HarmonyOS的文件管理系统

文件管理子系统为OpenHarmony提供一套完整的文件数据管理解决方案，提供安全、易用的文件访问能力和完善的文件存储管理能力。

- 为应用提供安全的沙箱隔离技术，保证应用数据安全基础上权限最小化；
- 统一的公共文件管理能力，统一公共数据访问入库，保证用户数据安全、纯净；
- 分布式文件系统和云接入文件系统访问框架，应用可以像使用本地文件一样使用分布式和云端文件；
- 支持公共数据、跨应用、跨设备的系统级文件分享能力；
- 提供系统的存储管理能力和基础文件系统能力。

图8-1 HarmonyOS的文件管理子系统架构图

HarmonyOS的文件管理系统

1. **文件访问接口**：提供完整文件JS接口，支持基础文件访问能力；提供本地文件、分布式文件扩展接口。
2. **存储管理**：提供数据备份恢复框架能力，支持系统和应用数据备份克隆等场景；提供应用空间清理和统计、配额管控等空间管理能力；提供挂载管理、外卡管理、设备管理及多用户管理等存储存储管理能力。
3. **公共文件**：公共数据沙箱隔离，保证用户数据安全、纯净；统一公共数据访问入口，仅medialibrary；提供统一的FMF的文件管理框架。

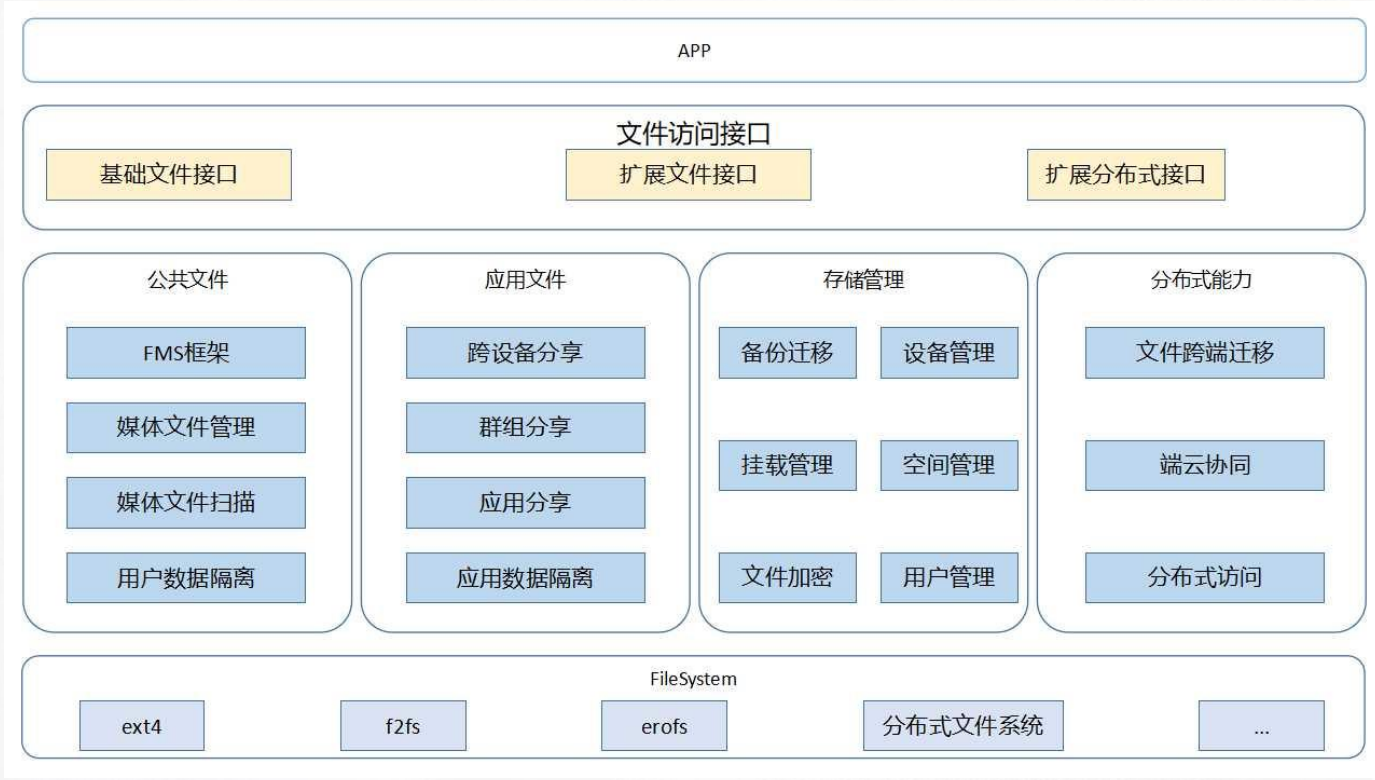


图8-1 HarmonyOS的文件管理子系统架构图

- **HarmonyOS的文件管理系统**

4. **应用文件**：为应用提供安全的沙箱隔离技术，保证应用数据安全基础上权限最小化；支持应用间文件分享和文件跨设备分享，支持群组分享。
5. **分布式能力**：提供基础分布式跨端访问能力，支持同账号分布式访问和异账号临时访问；支持文件跨端迁移能力，支撑应用迁移、分布式剪切板等分布式场景。
6. **基础文件系统**：支持ext4、f2fs、exfat、ntfs等本地文件系统；支持分布式文件系统、nfs等网络文件系统；文件系统相关工具。

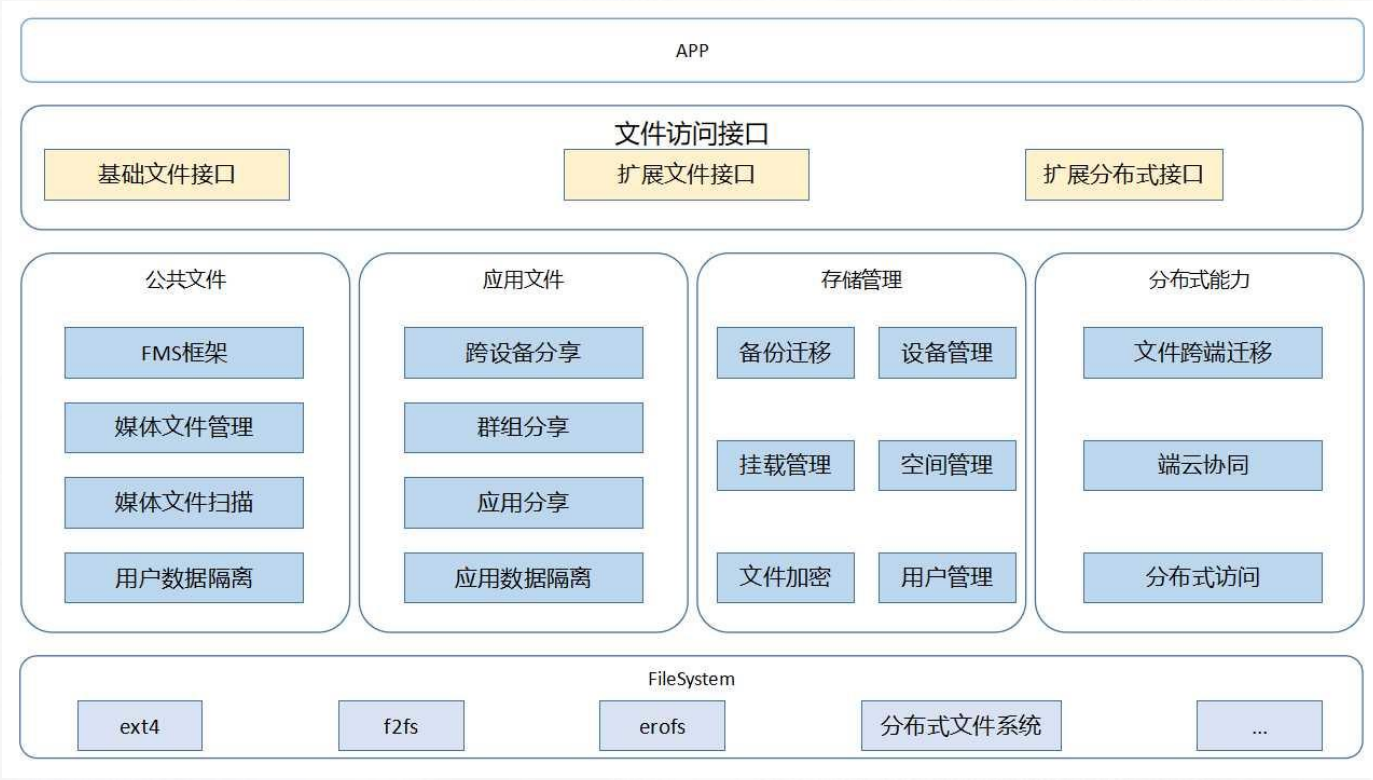


图8-1 HarmonyOS的文件管理子系统架构图

文件分类

在文件管理系统中，按文件所有者不同，可以将文件类别分为应用文件、用户文件和系统文件。

- **应用文件**：文件所有者为应用，包括应用安装文件、应用资源文件、应用缓存文件等。
- **用户文件**：文件所有者为登录到该终端设备上的用户，包括用户私有的图片、视频、音频和文档等。
- **系统文件**：与应用和用户无关的其他文件，包括公共库、设备文件和系统资源文件等。

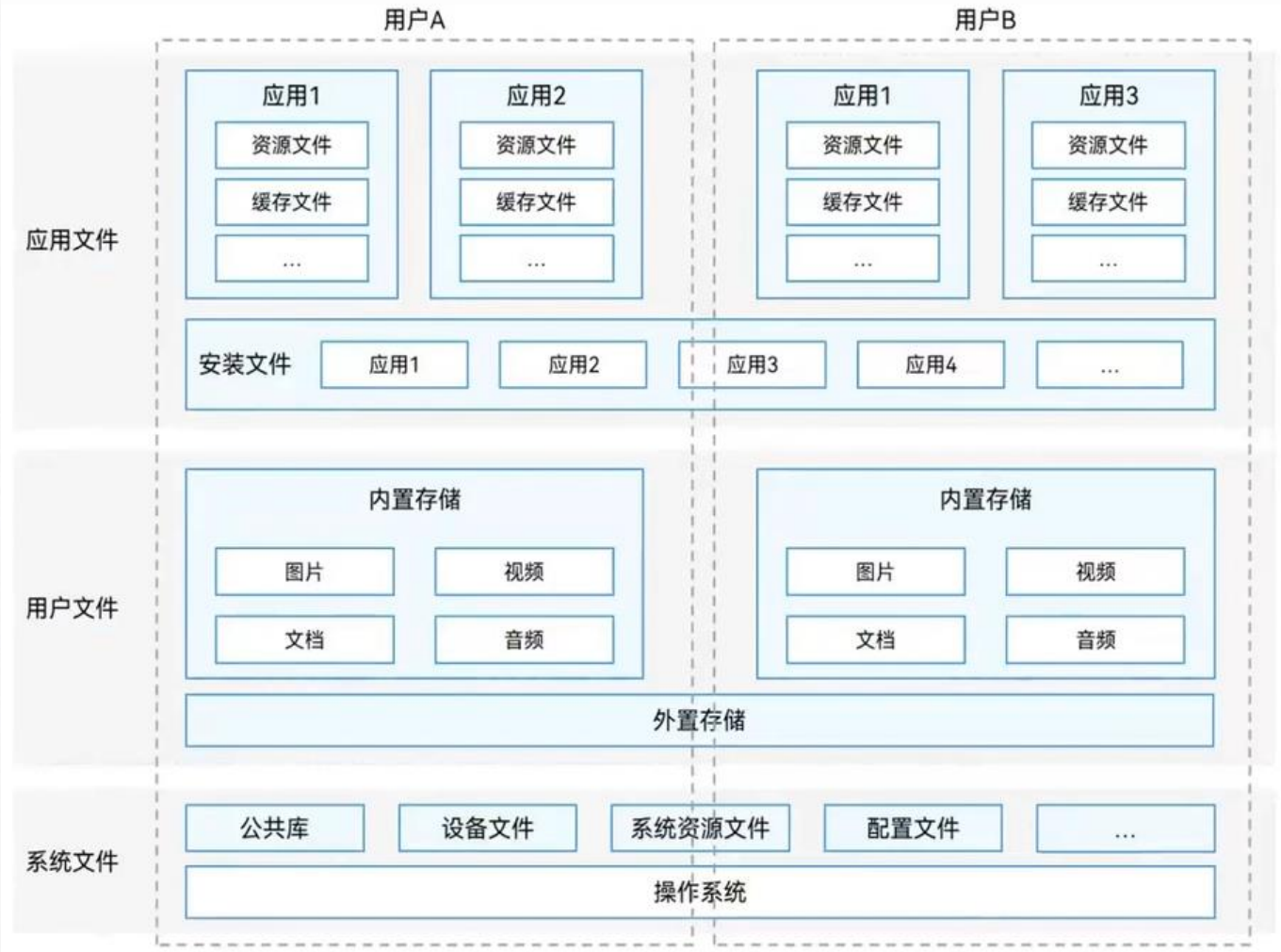


图8-2 HarmonyOS的文件分类模型

应用沙箱隔离机制

应用沙箱是一种以安全防护为目的的隔离机制，避免数据受到恶意路径穿越访问。在这种沙箱的保护机制下，应用可见的目录范围即为“应用沙箱目录”。

- 对于每个应用，系统会在内部存储空间映射出一个专属的“应用沙箱目录”，它是“应用文件目录”与一部分系统文件(应用运行必需的少量系统文件)所在的目录组成的集合。
- 应用沙箱限制了应用可见的数据的最小范围。在“应用沙箱目录”中，应用仅能看到自己的应用文件以及少量的系统文件(应用运行必需的少量系统文件)。因此，本应用的文件也不为其他应用可见，从而保护了应用文件的安全。
- 应用可以在“应用文件目录”下保存和处理自己的应用文件；系统文件及其目录对于应用是只读的；而应用若需访问用户文件，则需要通过特定API同时经过用户的相应授权才能进行。

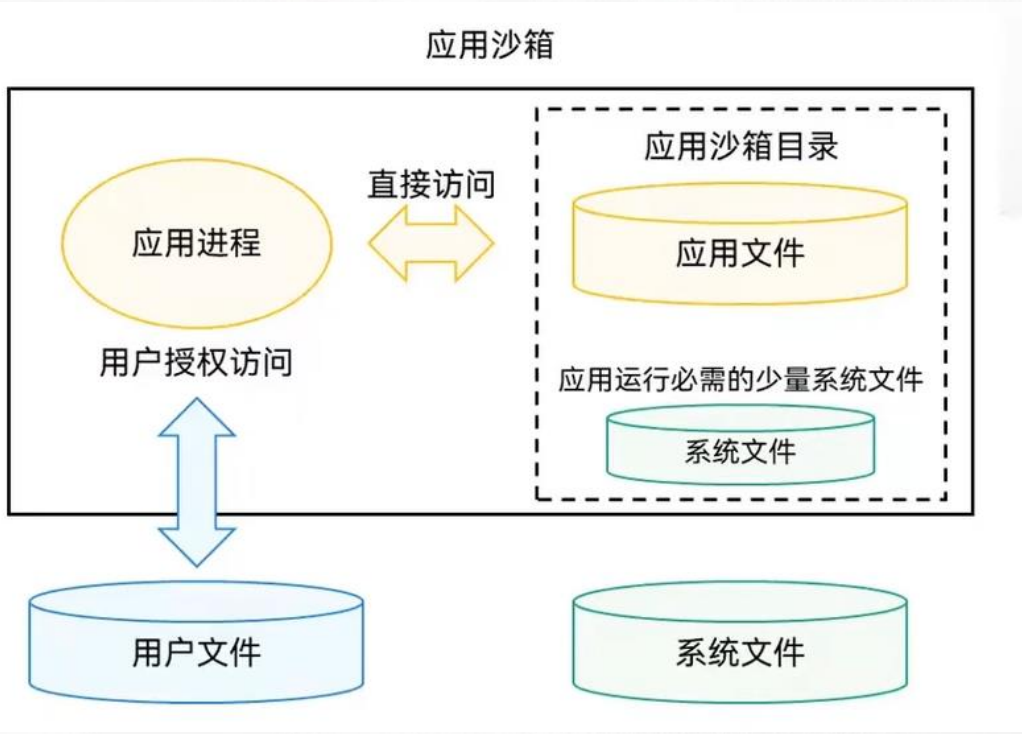


图8-3 应用沙箱模型

应用沙箱隔离机制

在应用沙箱保护机制下，应用无法获知除自身应用文件目录之外的其他应用或用户的数据目录位置及存在。同时，所有应用的目录可见范围均经过权限隔离与文件路径挂载隔离，形成了独立的路径视图，屏蔽了实际物理路径。

- 如下图所示，在普通应用（也称三方应用）视角下，不仅可见的目录与文件数量限制到了最小范围，并且可见的目录与文件路径也与系统进程等其他进程看到的不同。
- 实际物理路径与沙箱路径并非1:1的映射关系，沙箱路径总是少于系统进程视角可见的物理路径。

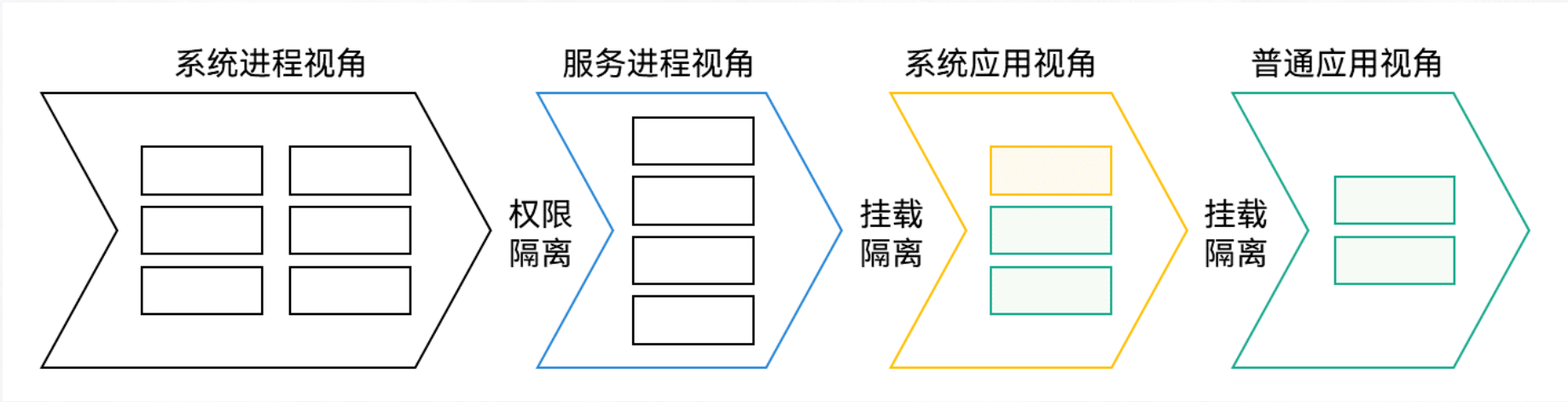


图8-4 应用沙箱路径（不同权限与角色的进程下可见的文件路径不同）

HarmonyOS的虚拟文件系统

1. HarmonyOS的文件系统与一般的Linux和UNIX文件系统非常接近。
2. 由于不同类型的文件系统接口不统一，若系统中有多个文件系统类型，访问不同的文件系统就需要使用不同的非标准接口。
3. 通过在系统中添加VFS层，提供统一的抽象接口，屏蔽了底层文件系统的差异，使得访问文件系统的系统调用不用关心底层的存储介质和文件系统类型，提高开发效率。

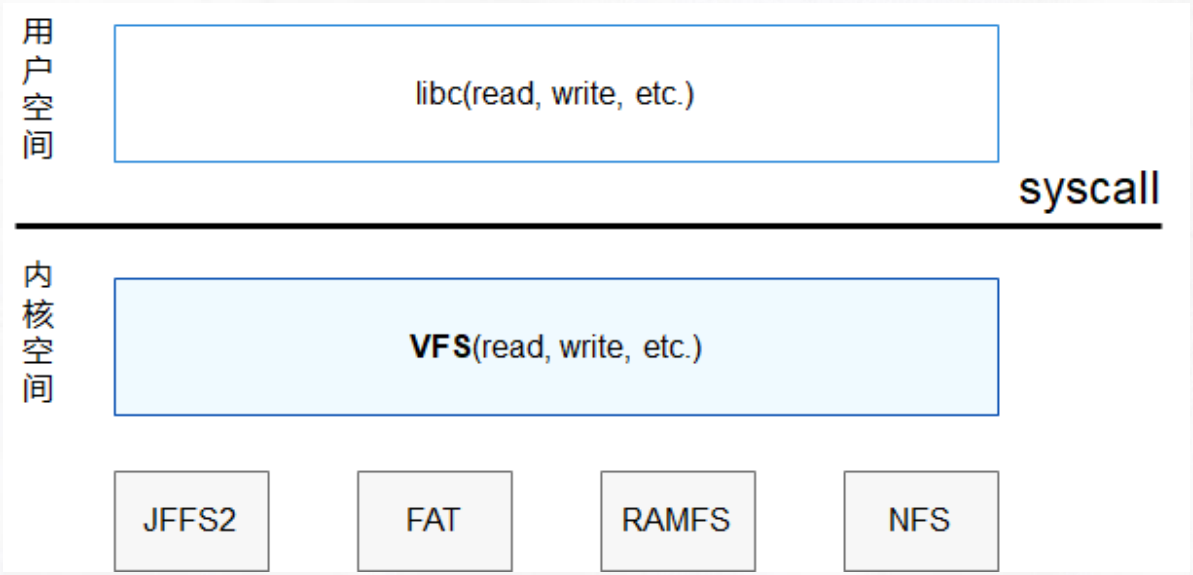


图8-5 HarmonyOS的虚拟文件系统

HarmonyOS的虚拟文件系统

1. OpenHarmony内核中，VFS框架是通过在内存中的树结构来实现的，树的每个结点都是一个inode结构体。设备注册和文件系统挂载后会根据路径在树中生成相应的结点。
2. VFS最主要是两个功能：
   - 查找节点。
   - 统一调用（标准）。

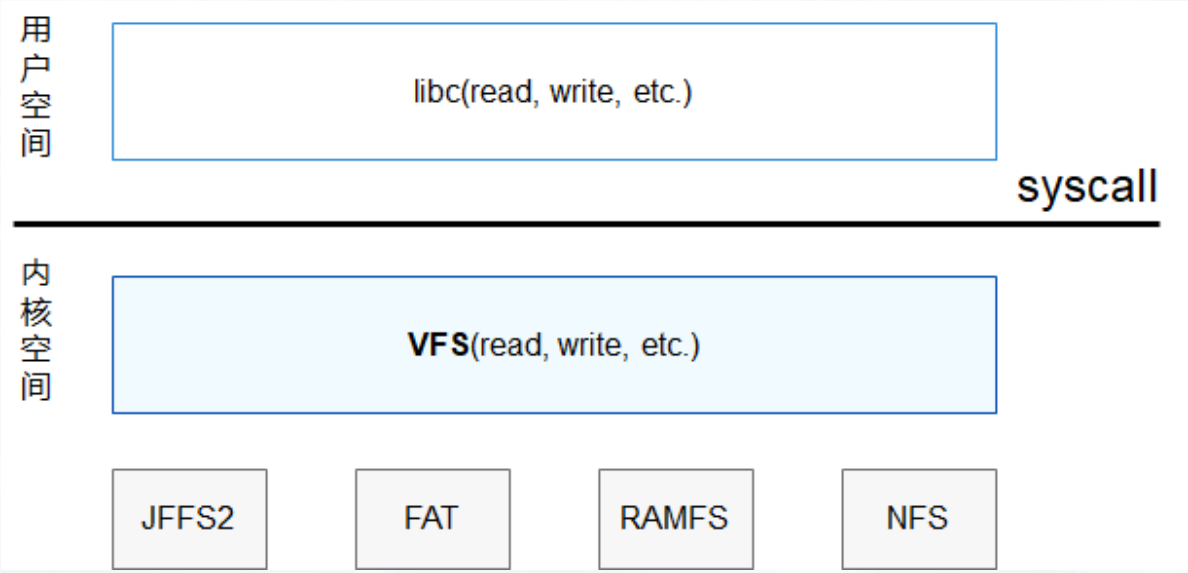


图8-5 HarmonyOS的虚拟文件系统

- **VFS的基础数据结构**

1. inode节点：LiteOS-a创建了两个文件系统的inode节点，分别是“/”和“/dev”。 “/”是文件系统的总根目录，“/dev”是所有设备的伪文件链接根。

```
struct inode
{
FAR struct inode *i_peer;     /*指向同级节点的指针*/
FAR struct inode *i_child;    /*指向下级节点的指针*/
int16_t i_crefs;              /*inode 节点引用数*/
uint16_t i_flags;             /*inode 标志位 */
unsigned long mountflags;     /*挂载标志*/
umion inode_ops_u u;          /*imnode 操作指令结构 */
#ifdefLOSCFG_FILE_MODE
unsigned int i_uid;
unsigned int i_gid;
mode_t i_mode;                /*liteOS 特有的数据，包括用户ID，组ID和存取模式标志位，如 0666 */
#endif
FAR void *i_private;          /*每个inode 特有的驱动私有数据*/
MOUNT_STATE e_status;         /*挂载状态*/
char i_name[1];               /*inode名字，变长*/
}
```

- **VFS的基础数据结构**

2. file：file结构是打开文件的底层表示形式。文件描述符实际上是指向这个结构体的索引。该结构体将文件描述符与文件状态和一组inode相关的驱动操作关联起来。

```
struct file
{
unsigned int f_magicnum;          /*文件魔法数*/
int f_oflags;                     /*文件打开模式标志位*/
FAR struct inode *f_inode;        /*inode 驱动接口 */
loff_t f_pos;                     /*文件偏移 */
unsigned_long f_refcount;         /*文件引用数*/
char *f_path;                     /*文件全路径名*/
void *f_priv;                     /*文件驱动的一些私有数据*/
const char *f_relpath;            /*真实路径*/
struct page_mapping *f_mapping;   /*文件所映射的内存区域*
void *f_dir;                      /*当打开目录时的目录结构*/
}
```

- **VFS的基础数据结构**

3. stat：stat结构体包含了跟一个文件的状态相关的几乎所有信息。应用层通过stat函数得到文件的stat结构体，并通过访问stat结构体的成员获取文件的各个属性和状态信息。

```
struct stat{
dev_t st_dev;                //包含文件的设备号
int __st_dev_padding;        //占位变量
long  __st_ino_truncated;    //已放弃不用
mode_t st_mode;              //文件对应的模式，文件，目录等
nlink_t st_nlink;            //文件的硬连接数
uid_t st_uid;                //所有者用户 1D
gid_t st_gid;                //所有者组 ID
dev_t st_rdev;               //如果是设备文件，则存放设备编号
int st_rdev_padding;         //占位变量
off_t st_size;               //以字节计算的文件大小
blksize_t st_blksize;        //IO 操作的块大小
blkcnt_t st_blocks;          //文件所占用的磁盘块数量
ino_t st_ino;                //inode 节点号
struct timespec st_atim;     //最后访问文件时间
struct timespec st_mtin;     //最后修改文件时间
struct timespec st_ctim;     //最后改变文件状态时间
}
```

HarmonyOS的inode树

1. 通过VFS层，可以使用标准的Unix文件操作函数（如open、read、write等）来实现对不同文件系统的访问。
2. VFS框架内存中的inode树结点有三种类型：
   - 虚拟结点：作为VFS框架的虚拟文件，保持树的连续性，如/usr、/usr/bin。
   - 设备结点：/dev目录下，对应一个设备，如/dev/mmc0blk0。
   - 挂载点：挂载具体文件系统，如/vs/sd、/mnt。

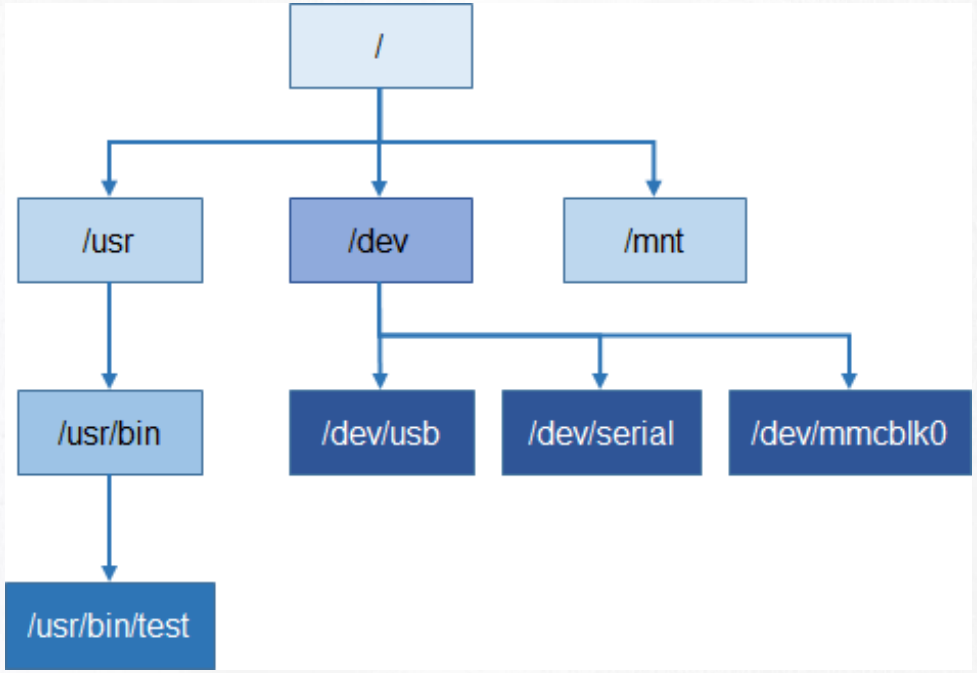


图8-6 HarmonyOS的inode树

HarmonyOS的inode树

3. VFS中的inode节点并不等同于具体的文件系统。其结构要比FAT等真实文件系统要简单，没有类似区块号、扇区等物理存储方面的信息。
4. 在挂载文件系统的时候，VFS的挂载点inode节点会对应到相应物理文件系统的根inode节点上。
   - 如挂载SDCARD到/mnt/sdcard后，SDCARD物理文件系统的根节点会对应到/mnt/sdcard关联的inode节点。

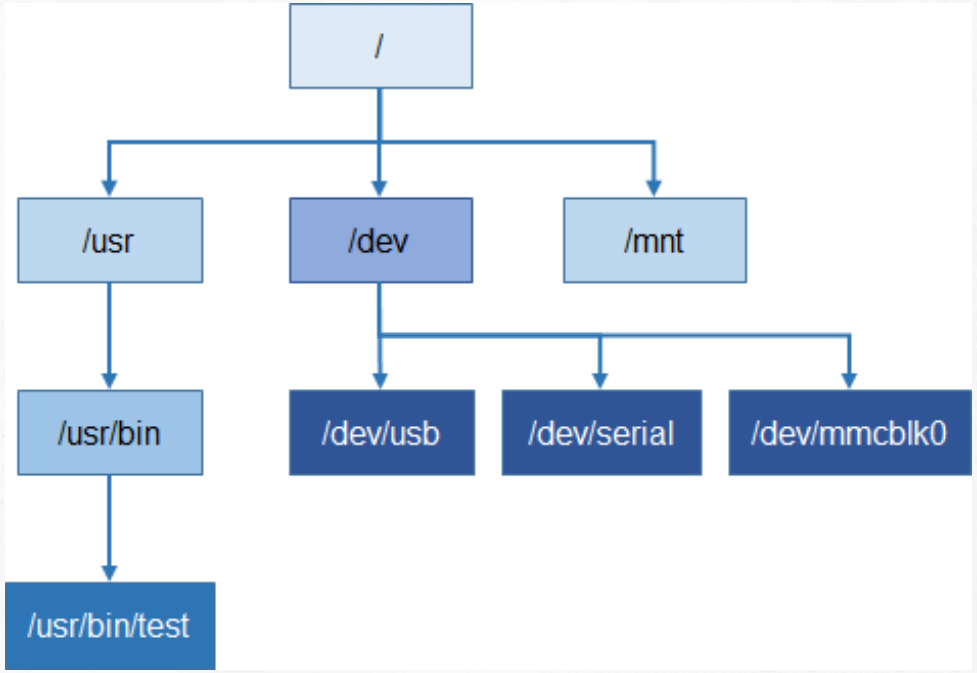


图8-6 HarmonyOS的inode树

HarmonyOS访问具体物理文件

找到inode节点以后，下一步就是要访问具体的物理文件。有三种不同的情况：

1. 直接访问文件。
   - 对于VFS内嵌支持的文件系统，可以通过注册的函数直接访问文件。
2. 访问块设备文件。
   - 对于块设备，因为块设备上也有inode信息，VFS通过inode节点对应的关系进行访问。
3. 访问挂载文件。
   - 对于挂载文件，VFS是通过调用挂载的时候注册的各个文件操作函数来访问具体的物理文件，这些具体的访问需要相应设备的驱动程序来提供。

- **HarmonyOS中的常用的文件操作API**

1. fs.stat：获取文件详细属性信息。
2. fs.access：检查文件是否存在。
3. fs.close：关闭文件。
4. fs.copyFile：复制文件。
5. fs.mkdir：创建目录。
6. fs.open：打开文件。
7. fs.read：从文件读取数据。
8. fs.rmdir：删除整个目录。
9. fs.unlink：删除文件。
10. fs.write：将数据写入文件。
11. fs.truncate：截断文件。
12. fs.rename：重命名文件或文件夹。
13. fs.fdatasync：实现文件内容数据同步。
14. fs.listFile：列出文件夹下所有文件名，支持递归列出所有文件名（包含子目录下），支持文件过滤。

- **FAT**

1. FAT文件系统是File Allocation Table（文件配置表）的简称， FAT文件系统有FAT12、FAT16、FAT32。
2. FAT 文件系统支持多种介质，特别在可移动存储介质（U盘、SD 卡、移动硬盘等）上广泛使用。可以使嵌入式设备和 Windows、 Linux 等桌面系统保持很好的兼容性，方便用户管理操作文件。
3. HarmonyOS 内核的 FAT 文件系统具有代码量和资源占用小、可裁切、支持多种物理介质等特性，并且与 Windows、 Linux 等系统保持兼容，支持多设备、多分区识别等功能。
4. HarmonyOS 内核支持硬盘多分区，可以在主分区以及逻辑分区上创建FAT文件系统。同时HarmonyOS 内核也可以识别出硬盘上其他类型的文件系统。
5. 最多支持同时打开的 fatfs 文件（文件夹）数为 512。
6. FAT 文件系统中，单个文件不能大于 4G。

- **FAT**

FAT文件系统的组成：

- 保留区：引导扇区和备份引导扇区，引导扇区中主要存放BIOS参数信息。如文件系统的文件布局，以及文件大小，块大小、簇大小等关键信息；
- FAT区：记录了文件所占用簇的情况。对于文件大小大于一个簇的文件，会在FAT区内形成簇链，记录文件由哪几个簇组成。
- 根目录区：只存在于FAT12/16系统，用于记录了根目录下的文件和目录信息。
- 数据区：由目录信息和文件数据组成。目录信息需要占据一个及以上的簇，用于描述对应文件夹目录下的文件情况；具体文件数据即为文件的内容。两部分数据通过短文件名SFN字段进行关联。

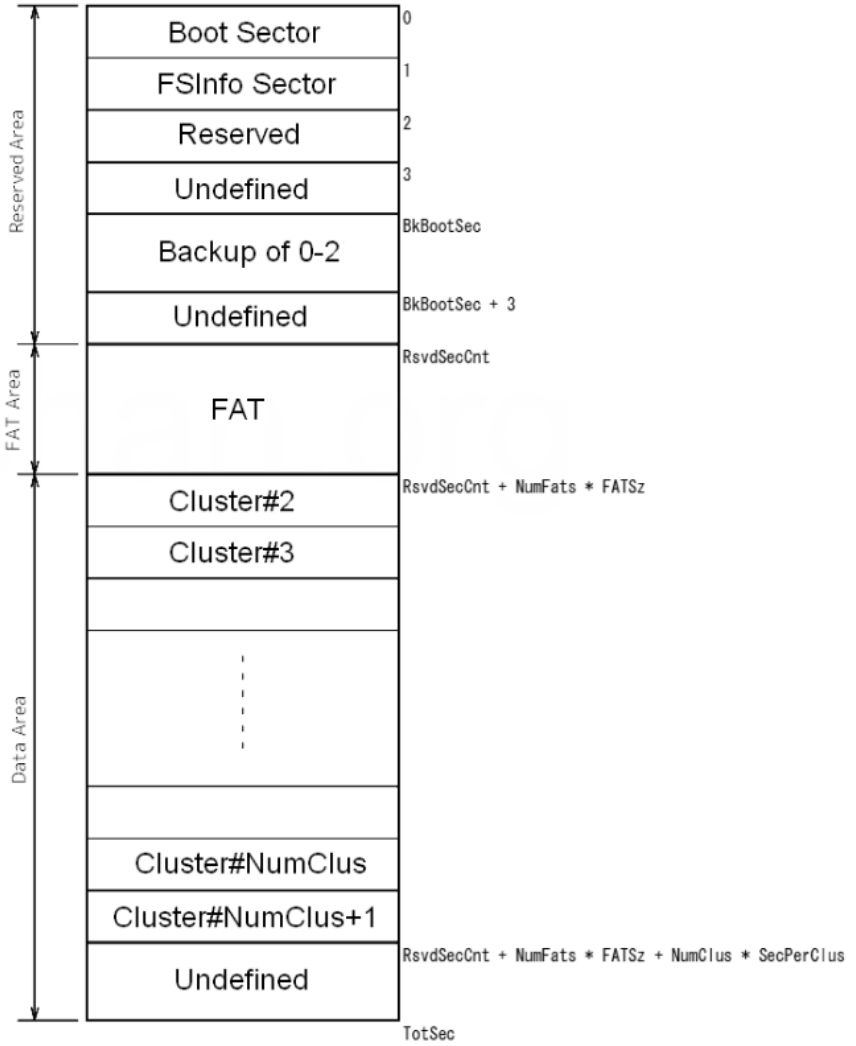


图8-7 FAT32文件系统布局[1]

[1] http://elm-chan.org/docs/fat_e.html#notes

- **JFFS2**

1. JFFS2 是 Journalling Flash File System Version 2（日志文件系统）的缩写，是MTD设备上实现的日志型文件系统。 JFFS2 主要应用于NOR FLASH，其特点是：可读写、支持数据压缩、提供了崩溃/掉电安全保护、提供“写平衡”支持等。
2. 闪存与磁盘介质有许多差异，因此直接将磁盘文件系统运行在闪存上存在性能和安全性上的不足。为解决这一问题，需要实现一个特别针对闪存的文件系统， JFFS2 就是这样一种文件系统。
3. HarmonyOS 内核的 JFFS2 主要应用于对 NOR Flash 闪存的文件管理，并且支持多分区。
4. 系统会自动对起始地址和分区大小根据 block 大小进行对齐操作。有效的分区号为 0~19。
5. 目前 JFFS2 文件系统用于 NOR Flash，最终调用 NOR Flash 驱动接口，因此使用 JFFS2 文件系统之前要确保硬件上有 NOR Flash，且驱动初始化成功。

[2] Woodhouse D. JFFS: The journalling flash file system[C]//Ottawa linux symposium. 2001, 2001.

NFS

1. NFS 是 Network File System（网络文件系统）的缩写。它最大的功能是可以通过网络，让不同的机器、不同的操作系统彼此分享其他用户的文件。因此，用户可以简单地将它看做是一个文件系统服务，在一定程度上相当于Windows 环境下的共享文件夹。
2. NFS 客户端用户，能够将网络远程的 NFS 服务端分享的目录挂载到本地端的机器中，运行程序和共享文件，但不占用当前的系统资源，所以，在本地端的机器看起来远程服务端的目录就好像是自己的一个磁盘一样。
3. 当前 NFS 支持 TCP 和 UDP 两种传输层协议，默认使用 TCP。

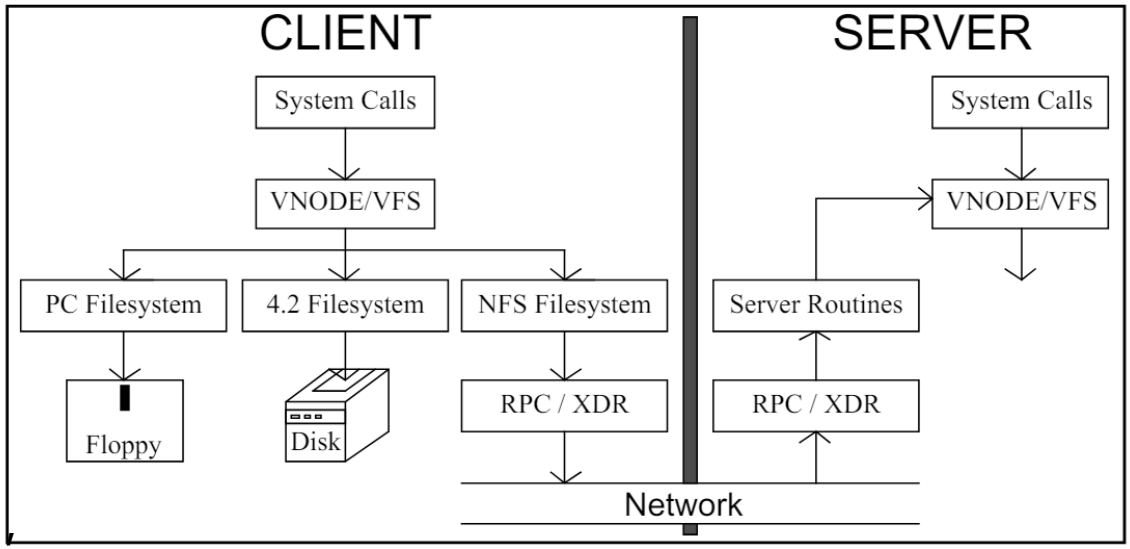


图8-8 NFS文件系统示意图[3]

[3] Sandberg R. The Sun network file system: Design, implementation and experience[C]//in Proceedings of the Summer 1986 USENIX Technical Conference and Exhibition. 1986.

RAMFS

1. RAMFS 是一个可动态调整大小的基于 RAM 的文件系统。RAMFS 没有后备存储源。向 RAMFS 中进行的文件写操作也会分配目录项和页缓存，但是数据并不写回到任何其他存储介质上，掉电后数据丢失。
2. RAMFS 文件系统把所有的文件都放在 RAM 中，所以读/写操作发生在 RAM 中，可以用 RAMFS 来存储一些临时性或经常要修改的数据，例如/tmp 和/var目录，这样既避免了对存储器的读写损耗，也提高了数据读写速度。
3. HarmonyOS 内核的 RAMFS 是一个简单的文件系统，它是基于 RAM 的动态文件系统的一种缓冲机制。
4. HarmonyOS 内核的 RAMFS 基于虚拟文件系统层（VFS），不能格式化。
5. HarmonyOS 内核的 RAMFS 只能挂载一次，一次挂载成功后，后面不能继续挂载到其他目录。

中山大學
SUN YAT-SEN UNIVERSITY

谢谢观看

SUN YAT-SEN UNIVERSITY