Lecture14 File System Structure

1. 虚拟文件系统(VFS)

如何在一个系统中同时支持多个文件系统?

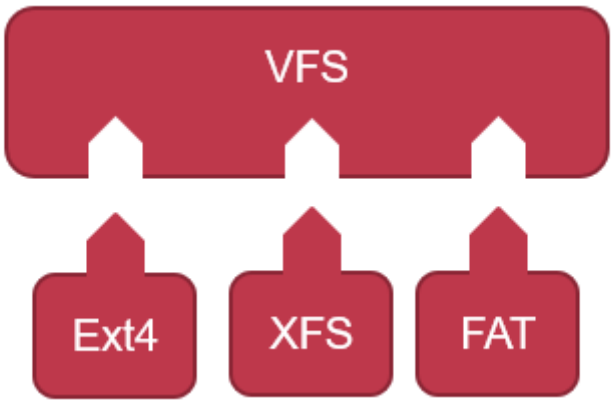
- 计算机中的异构文件系统
    1. Linux和Windows双启动，两个分区有各自的文件系统
    2. Mac用APFS，U盘一般用FAT/exFAT，移动硬盘用NTFS
- 如何对用户屏蔽文件系统的异构型?

> 使用VFS(Virtual File System)
>
> 中间层，对上提供POSIX API，对下对接不同文件系统的驱动

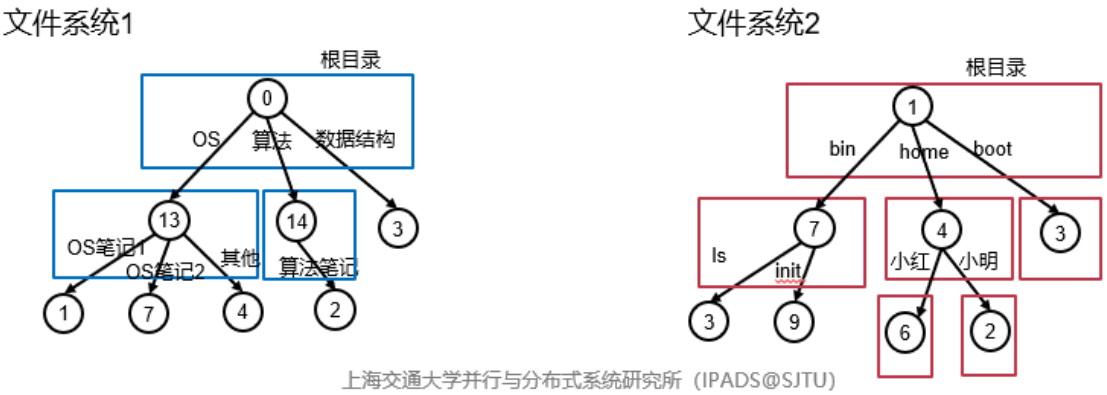
Linux中的虚拟文件系统VFS

> VFS是in-memory-only的，而inode是in disk的

- Linux的VFS定义了一系列接口，在读取一个inode文件时:
    1. VFS先找到该inode所属文件系统
    2. 再调用该文件系统的读取接口



- 操作系统同时使用多个文件系统，虚拟文件系统提供统一的管理，对应用程序提供统一的视图和抽象（**通过挂载等操作配合在一起**）



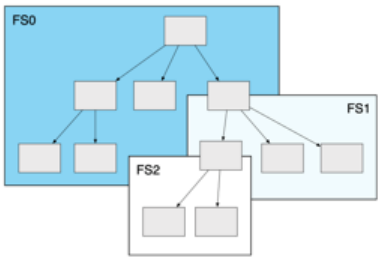
上海交通大学并行与分布式系统研究所（IPADS@SJTU）

- VFS维护一个统一的文件系统树

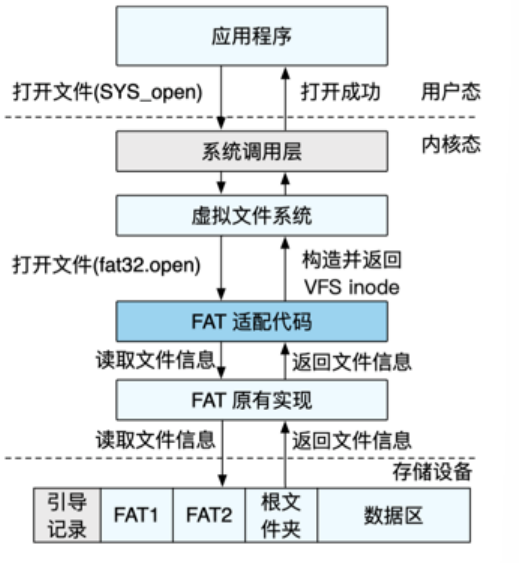
1. 操作系统内核启动时会挂在一个**根文件目录**
2. 其它文件系统可以**挂载**在文件系统树的目录上

- VFS维护所有的挂载信息
  1. 查找文件时的每一步，检查当前目录是否为挂载点
  2. 若是，则使用被挂载的文件系统继续进行访问

实例：VFS对接FAT32



	常用挂载点	描述
procfs	/proc	查看和操作进程相关的信息和配置
sysfs	/sys	查看和操作与进程无关的系统配置
debugfs		用于内核状态的调试
cgroupfs		用于管理系统中的 cgroups
configfs	/sys/kernel/config	创建、管理和删除内核对象
hugetlbfs		查看和管理系统中的大页信息

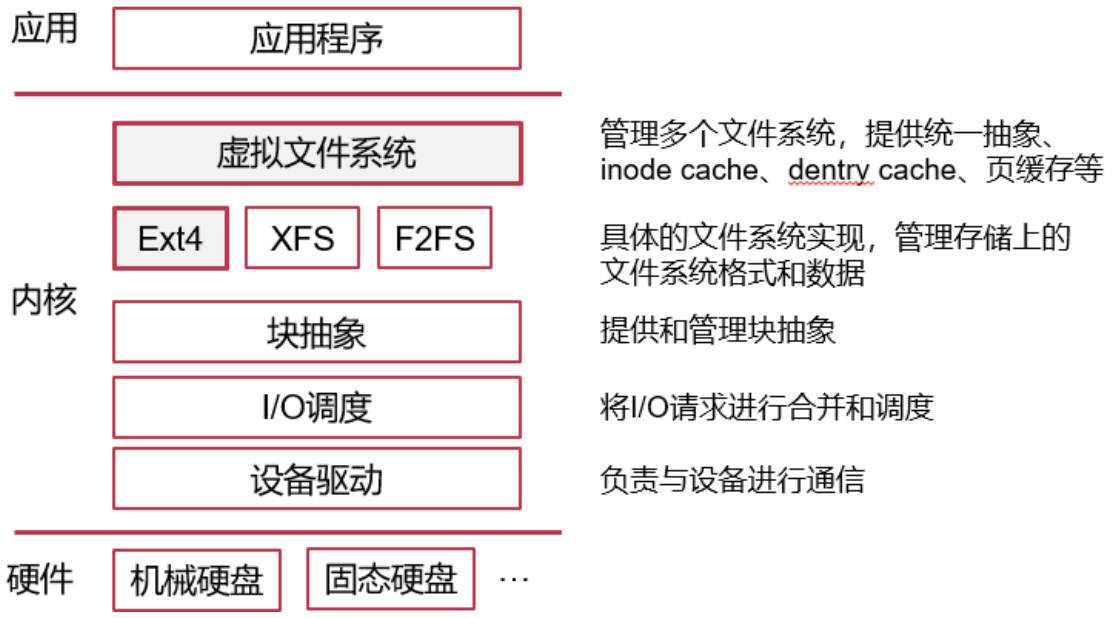


> VFS层对上提供接口，每一个文件都有一个inode
>
> 磁盘上的FAT并没有inode：硬盘上的数据结构
>
> 内存中的VFS需要inode：只在内存中的数据结构（停电之后就没了）

> Q：FAT没有inode，如何挂载到VFS?
>
> A：FAT的驱动提供inode（途中的FAT适配代码，也就是驱动）

2. 存储结构与缓存

宏内核(Linux)中的存储栈

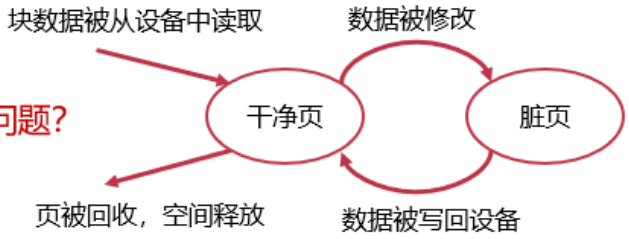


页缓存(Page Cache)

> - 存储访问非常耗时
> - 文件访问具有时间局部性
>   - 一些目录/文件的数据块会被频繁的读取或写入

- 通过**缓存**提升文件系统性能
  - 在一个块被读入内存并被访问完成后，并不立即回收内存
    - 将块数据暂时缓存在内存中，下一次被访问时可以避免磁盘读取
  - 在一个块被修改后，并不立即将其写回设备
    - 将块数据暂时留在内存中，此后对于该数据块的写可直接修改在此内存中
  - 定期或在用户要求时才将数据写回设备

问题：数据不及时写回，会造成什么问题？



> A：若不及时写回，此时若crash掉了，则数据会发生丢失

页缓存之外

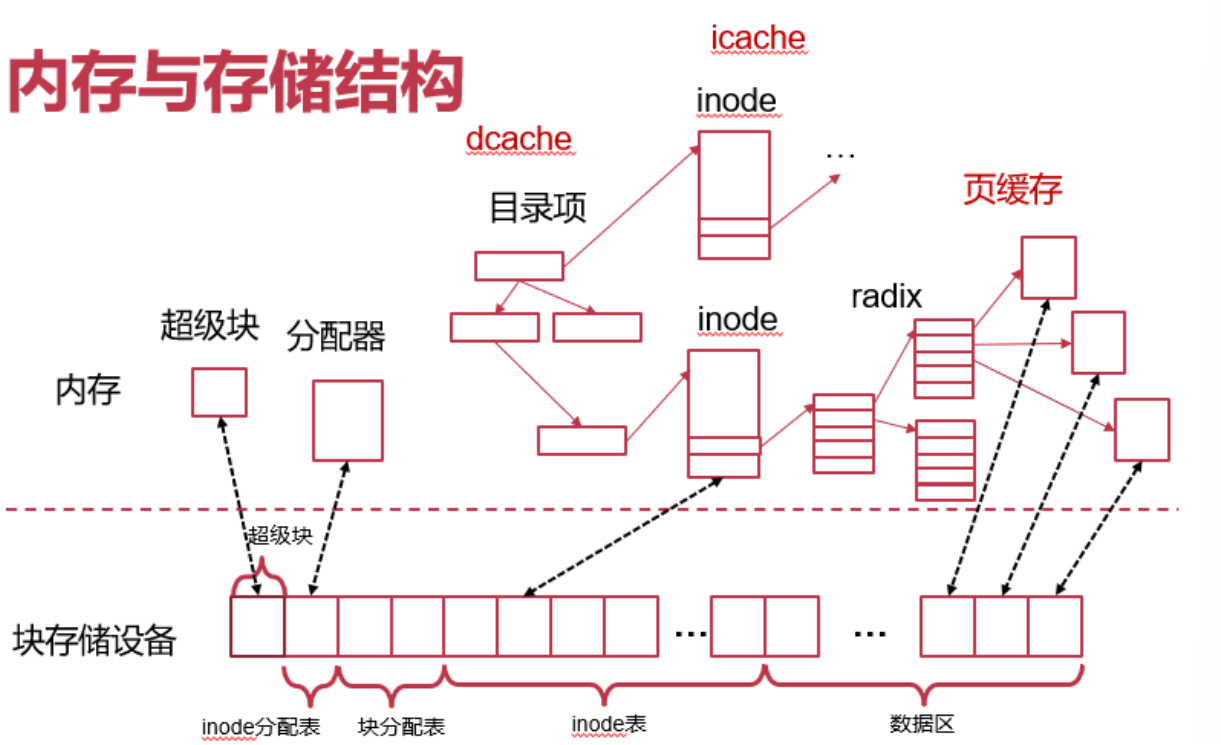
存储中的每个数据结构，在内存中均有对应的结构

- 存储的数据页：页缓存中的内存页
- 存储中的inode：icache中的inode
- 存储中的目录项：dcache中的目录项
- 存储中的超级块：内存中的超级块结构

- 存储中的分配表：内存中的分配器

> Q：为什么要为每个结构设计单独的缓存，能否只使用页缓存？
>
> A：页缓存可以是可以，但是粒度太大，太浪费内存了。inode的cache可能只有几百个bytes，dentry的cache可能只有十几二十个bytes，如果单独为每一个inode或者是dentry分配一个物理页，则存储空间浪费巨大

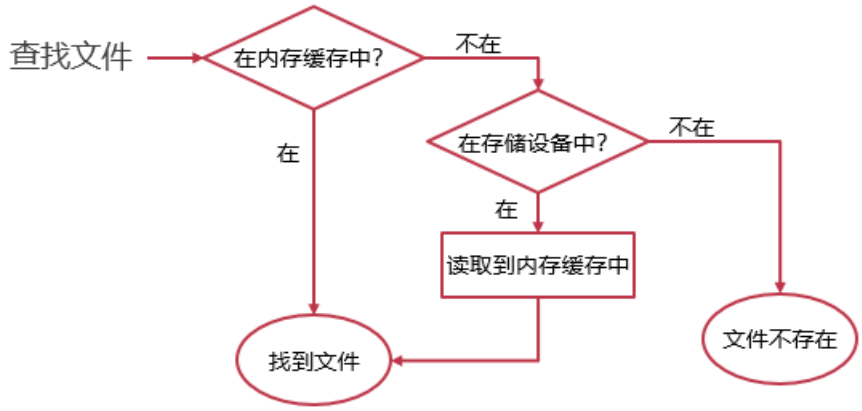
内存与存储结构



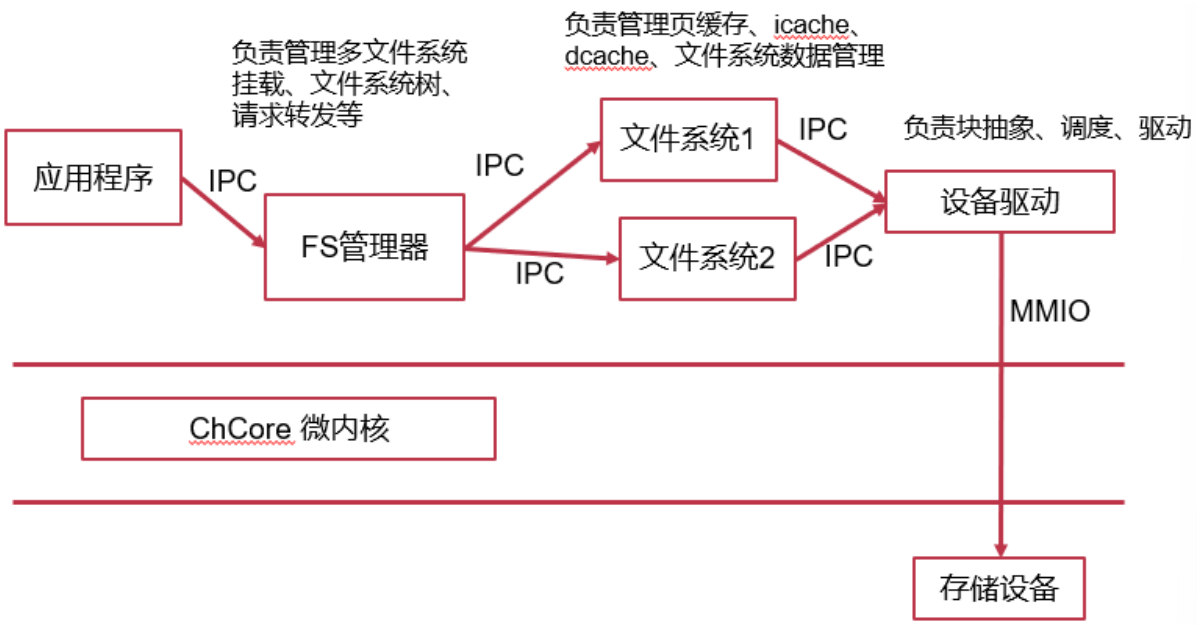
有缓存情况下的文件查找

由于内存大小限制，内存中缓存的数据是存储中数据的子集

当要访问的数据不在内存中时，会从存储中读取并构造内存中相应的对象

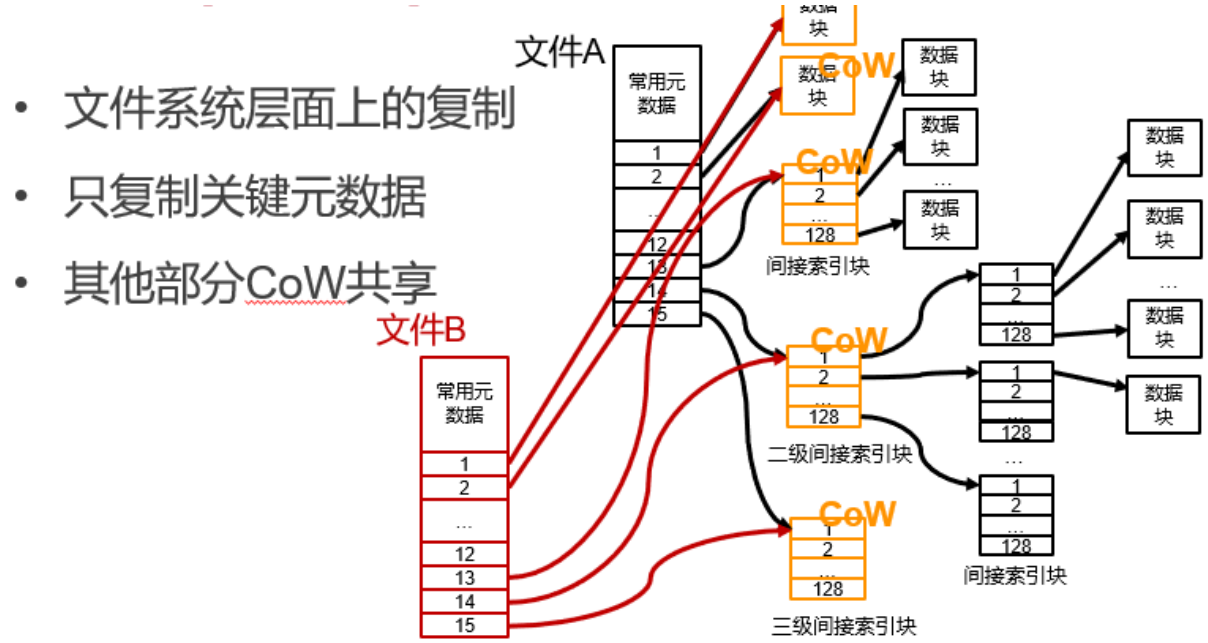


ChCore中的文件与存储结构



3.文件系统高级功能

克隆(Clone)



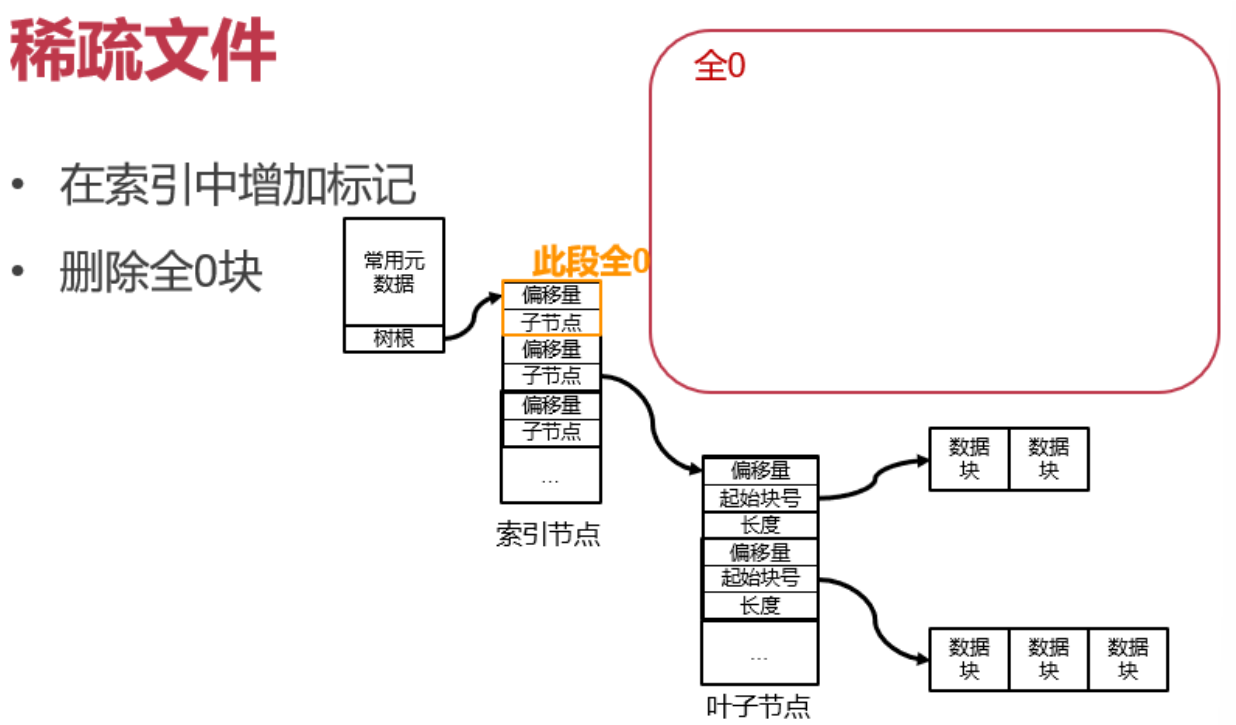
快照(Snapshot)

- 同样使用CoW
- 对基于inode表的文件系统
  - 将inode表拷贝一份作为快照保存
  - 标记已用数据区为CoW

- 对于树状结构的文件系统
  - 将树根拷贝一份作为快照保存
  - 树根以下的节点标记为CoW

稀疏文件

- 若一个文件大部分数据为0，则为稀疏文件
  - 比如说虚拟机镜像文件
- 稀疏文件中大量的0数据，浪费存储空间



文件系统的一些其他高级功能

- 加密
- 压缩
- 去重
- 数据和元数据校验
- 配额管理（QoS）
- 软件RAID
- 多设备管理
- 子卷
- 事务（Transaction）

4. 文件系统的多种形式

GIT：内容寻址文件系统

> *"文件系统之上的文件系统"*

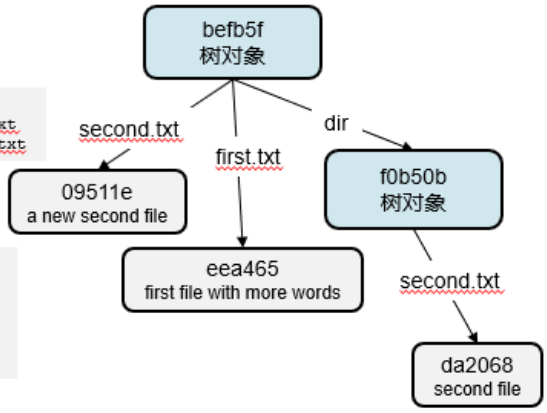
- Git表面上是一个版本控制软件，但实际上可以被看做是一个**内容寻址**的文件系统
- 其核心是一个键值存储(KV Store)
  - 值：加入Git的数据
  - 键：通过数据内容算出的40个字符SHA-1校验和
    - 前2个字符作为子目录名，后38个字符作为文件名
  - 所有对象均保存在`.git/objects`目录中（文件内容会被压缩）

- **BLOB对象**：对应文件系统中的文件

```
first file with more words
```

- **树对象**：对应文件系统中的目录

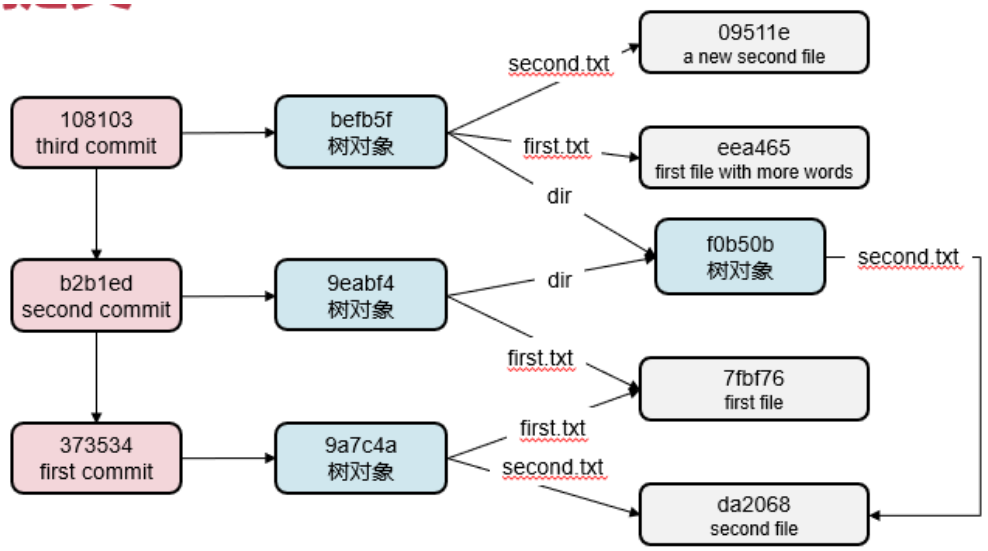
```
040000 tree f0b50bef52478d42d68ff21914c430d8148f23cd  dir
100644 blob eea465ab73dfb5cd24379efd61da949b3b5138ea  first.txt
100644 blob 09511ef0b04fb20dd32f3cb090f9c6de49cd2627  second.txt
```



- **提交对象**

```
tree befb5f375306393e542026391d95104579da48ce
parent b2b1eda3c44d0a2cb10ad7f522ae78f9ded7e95e
author IPADS <ipads@ipads.se.sjtu.edu.cn> 1587414317 +0800
committer IPADS <ipads@ipads.se.sjtu.edu.cn> 1587414317 +0800

third commit
```



> 如果两个文件没有改变的话，它们算出的SHA-1值就不会改变，在两次commit中就不会新建对象去存储

SQLite：文件系统的竞争者

- 表面上SQLite是一个数据库
- 但实际上SQLite也可以是一个文件系统!
- 其核心还是一个数据库...
  - 在关系型数据库的表中，记录文件名和BLOB类型文件数据
  - 通过查找文件名，获取对应文件数据
  - 存储大量小文件
- 文件系统里的文件里的文件系统里的文件

> Q：对于小文件，为何一般文件系统不如SQLite效率高?
>
> A：目录结构效率不高，目录太深，同时打开和查找的时间都很慢

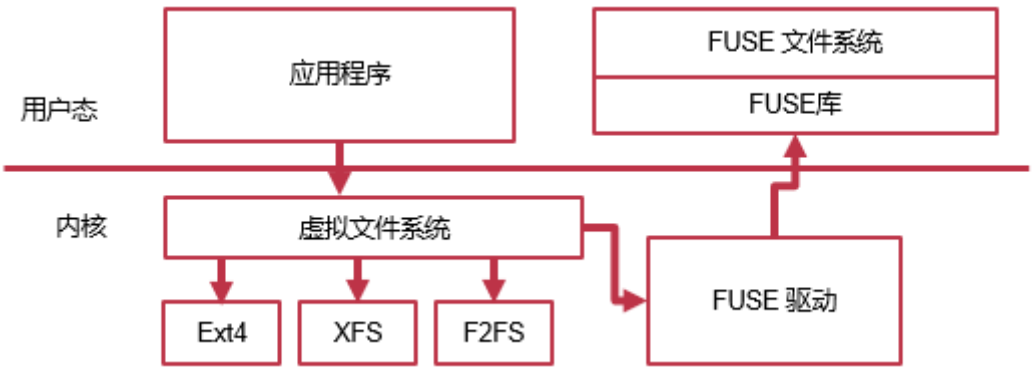
> Q：文件系统如何针对小文件进行改进?
>
> A：优化目录结构、用DB做FS的索引、或者再FS内置一个DB专门存储小文件

> Q：还有哪些针对小文件特殊处理的场景?
>
> A：HTML小图片拼在一起，用css切图（页面加载速度、网络传输）；传文件到远端或者U盘，先打包再传输；Git的push/pull是先打包再传输

5. FUSE：用户态文件系统框架

为什么要用户态文件系统?

1. 快速试验文件系统新设计（不用改kernel）
2. 大量第三方库可以使用
3. 方便调试
4. 无需担心把内核搞崩溃
5. 实现新功能



FUSE基本流程

1. FUSE文件系统向FUSE驱动注册（挂载）
2. 应用程序发起文件请求
3. 根据挂载点，VFS将请求转发给FUSE驱动
4. FUSE驱动通过中断、共享内存等方式将请求发给FUSE文件系统
5. FUSE文件系统处理请求
6. FUSE文件系统通知FUSE驱动请求结果
7. FUSE驱动通过VFS返回结果给应用程序

> Q：从这个流程中可以看出FUSE有什么问题?
>
> A：太慢了...

FUSE API

- 底层API
    1. 直接与内核交互
    2. 需要负责处理inode和查找等操作
    3. 需要处理内核版本等差异
- 高层API
    1. 构建于底层API之上
    2. 以路径名为参数
    3. 无需关注inode、路径和查找

现实中FUSE能用来做什么?

- 出Lab!
- SSHFS（用ssh挂载远端目录到本地）
- Android Sandbox
- GMailFs（以文件接口收发邮件）
- WikipediaFS（用文件查看和编辑Wikipedia）
- 网盘同步
- 分布式文件系统（Lustre、GlusterFS等）
- Since *everything is a file*, can everything be done with a filesystem?