## 开发背景

## 基本定义

## WondFS概述

## Disk Layout

## 架构设计

## Disk Layer

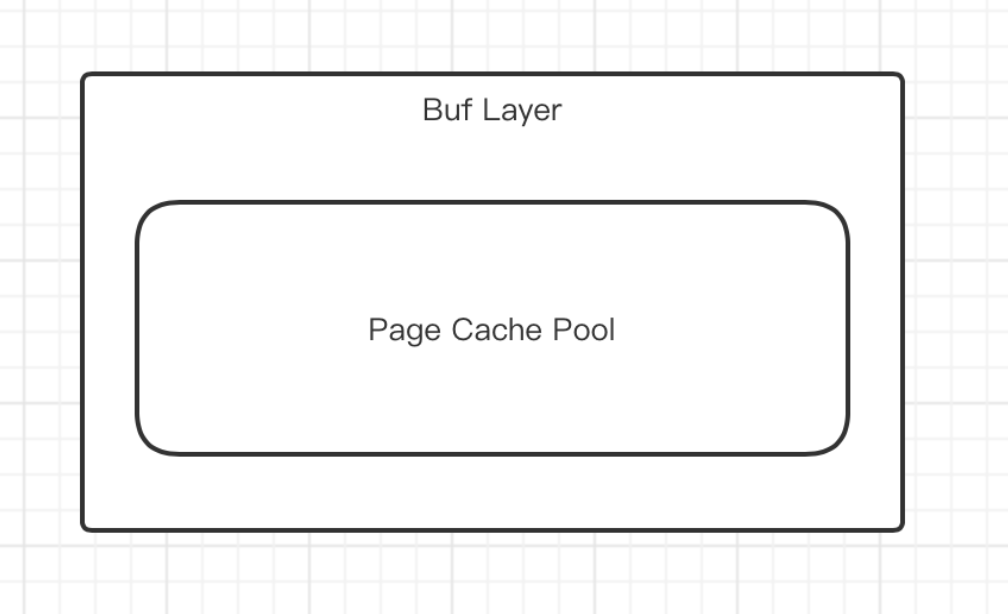
## Translation Layer

## Buf Layer

**8.1 基本设计**

Buf Layer以Page为单位缓存数据，对上提供数据的访问接口。Buf Layer中包含一个LRU结构的缓存池，当缓存池满后，选择最近最久未使用的数据予以淘汰。缓存池中以一个page中的数据为单位缓存。当读取某一page的数据时，根据局部性原理，会读取相应block中的所有page并一起放入缓存池中。当写入page或擦除一系列page时，会更新缓存中的数据，并将对物理存储的一系列更新操作同步到文件系统架构的下层。

**8.2 层次结构**



**8.3 功能实现**

（一）相关数据结构

1. Buf

* address
* data

缓存池中缓存的数据项，与物理存储中具体page的数据保持一致。

1. BufCache

* capacity
* cache
* translation\_layer

缓存池的容量默认为1024，即最多可以缓存1024个page，cache是一个lru数据结构的泛型实现，translation\_layer持有文件系统下层的数据结构。

（二）关键函数

1. read

读取一个page的数据，当缓存命中时，直接从缓存池中读取。当缓存未命中时，调用Translation Layer中的函数读取数据，并一次读取一个block的数据内容，将新读取的数据插入缓存池。

1. write

调用Translation Layer中的函数写入一个page的数据，当缓存池中包含该page时，同步更新缓存。

1. **write\_block\_direct**

调用Translation Layer中的函数写入一个block的数据，当缓存池中包含该block中的某个page时，同步更新缓存。这个方法可以绕过Translation Layer中的写缓存，保证数据可以直接同步到物理存储中。

**erase**

调用Translation Layer中的函数擦除某个block，当缓存池中包含该block中的某个page时，同步更新缓存。

## Core Layer

## Inode Layer

**10.1 基本设计**

Inode Layer是文件系统的重要一个层次，在Inode Layer数据被组织成文件管理，对上提供inode的视图。Inode的术语可能有两个含义，一个可能指的是包含文件元信息和数据块位置的磁盘数据结构，在WondFS，我们将Inode的磁盘结构称为Raw Inode，并放在KV Region中通过LSM-Tree进行管理。我们在这里提到的Inode指的是内存中的Inode，不仅包含了磁盘上Raw Inode包含的所有信息，还有额外的在内存管理需要的额外信息。

值得一提的是，Raw Inode中数据块的地址存储的是LBA，也就是逻辑地址，根据逻辑地址通过Translation Layer可以读取。但是Inode中数据块的地址存储的并不是LBA，而是通过Core Layer中的VAM组件抽象出来的虚拟地址。VAM组件存储了逻辑地址和虚拟地址之间的映射关系。之所以需要抽象出虚拟地址的概念，这是因为在WondFS的GC中，GC位于Core Layer层次，这意味着，当一个inode处于打开的情况下，如果GC改变了这个Inode中某些数据块的逻辑地址，只需要在内存中改变VAM的映射关系。这对于架构设计来说是非常合理的，因为VAM和GC处于同一架构层次，他们之间的互相调用都是允许的，但是Inode Layer位于Core Layer的上层，GC对于逻辑地址的改变不应该通过调用上层函数的方式将影响传导到Inode中。上层可以调用下层，下层不允许调用上层，这对于架构设计的稳定性和合理性有极大的约束性帮助。

**10.2 层次结构**

**10.3 功能实现**

(一) InodeManager

（1）相关数据结构

* size
* capacity
* inode\_buffer
* lock
* core\_manager

size是打开的inode数量，capacity是Inode Manager支持的最多打开文件数，默认情况下最多在内存中打开30个文件。lock是锁，控制InodeManager在并发情况下的数据安全。core\_manager持有下层的控制管理类。inode\_buffer存储打开的文件。

（2）相关函数

1. i\_alloc

调用core\_manager的allocate\_inode方法创建一个新的inode，存储在inode\_buffer中，并返回打开后的文件。

1. i\_get

根据ino获取inode。如果inode已经被打开，则将其ref\_cnt加1后返回inode。如果inode还没被打开，调用core\_manager的get\_inode方法获取inode，将其存储在inode\_buffer中，并返回打开后的文件。

1. i\_dup

将inode的ref\_cnt加1。

1. i\_put

将inode的ref\_cnt减1，删除对内存中inode的一个引用，如果这是最后一个引用，那么inode在inode\_buffer中的空间就可以被回收，关闭文件。

1. Inode

（1）相关数据结构

* file\_type
* ino
* size
* ref\_cnt
* n\_link
* mode
* uid
* gid
* last\_accessed
* last\_modified
* last\_metadata\_changed

（2）相关函数

1. read\_all

在读出inode文件中的所有数据。

1. read

## Common Layer

## System Layer