块初始化 block_init

1、创建主块操作对象

FileOperation
#fd_ #open_flags_ #file_name_
+open_file() +close_file() +flush_file() +flush_data() +unlink_file() +pread_file() +pwrite_file() +write_file() +get_file_size() +ftruncate_file() +seek_file() +get_fd() #check_file()

将主块文件路径传入作为
实参构建主块在操作对象

2.创建索引文件，

IndexHandle
-file_op_ -is_load_
+create() +load() +flush() +find_avail_key() +write_segment_meta() +read_segment_meta() +update_segment_meta() +delete_segment_meta() +update_block_info() +get_block_data_offset() -hash_compare() -hash_find() -hash_insert()

1、将mainblock id传进去作为实参

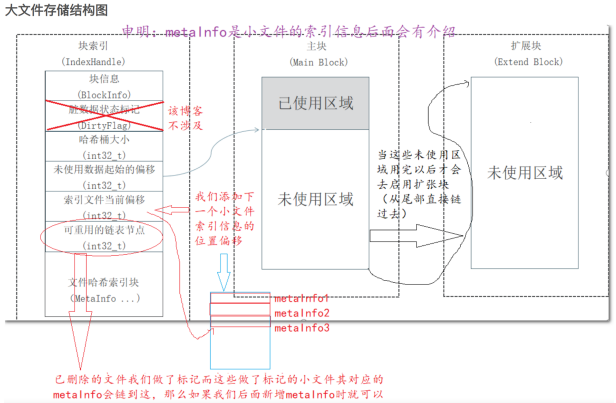
2.创建MMapFileOperation对象。
（浑索引文件路径作为实参，来创建和打开文件。）

当IndexHandle执行Create时
进行创建并映射

通过MMapFile对象进行

3.创建主块文件
通过FileOperation
f truncate方法来
创建大文件。

大文件存储结构图



写文件流程

1. 加载索引文件

```
ret = index_handle->load(block_id,bucket_size,mmap_option);
```

① 检查索引文件是否有效以及存在

② 将索引文件进行映射

③ 将相关结构进行检查

↓

文件大小 / 块大小 / "桶"大小

2. 将文件写入到主块文件

```
//创建主块操作对象
largefile::FileOperation* mainblock = new largefile::FileOperation(mainblock_path,O_RDWR|O_CREAT|O_LARGEFILE);
```

①通过索引文件找到目前块可用位置及文件编号.

② 通过main block对文件进行写入

```
mainblock->pwrite_file(buffer,sizeof(buffer),data_offset);
```

↓

(1) 如果位置 + 文件大小已经超过被映射的大小，那么映射大小就得改变.(主块的)

(2) 调整完映射大小，将内存重新进行映射

有映射的情况，如果没映射或映射空间不全就直接写磁盘

写入磁盘写出错，写while循环，结束条件是达到 Mask_Disk_Time 次或者数据读完.

通过偏移来判断.

```cpp
//写文件（带偏移）
int FileOperation::pwrite_file(const char* buf,const int32_t nbytes,const int64_t offset)
{
    //记录要写的剩余字节数
    int32_t left = nbytes;
    //读文件的偏移
    int64_t written_offset = offset;
    const char* p_tmp = buf;
    int64_t written_len = 0;

    //最多只能读MAX_DISK_TIMES次磁盘（如果读文件失败的话）
    int i = 0;
    while(left > 0)
    {
        i++;
        if(i >= MAX_DISK_TIMES)
        {
            break;
        }

        int fd = check_file();
        if(fd < 0)
        {
            return fd;
        }

        written_len = ::pwrite64(fd_,p_tmp,left,written_offset);
        if(written_len < 0)
        {
```

③更新索引文件.

(1). 创建 metaInfo 并初始化 file_no，主块上的偏移.

```cpp
//3.索引文件中写入metaInfo
largefile::MetaInfo meta;
meta.set_file_id(file_no);
meta.set_offset(data_offset);
meta.set_size(sizeof(buffer));
ret = index_handle->write_segment_meta(meta.get_key(),meta);
```

(2) 写入数据.

index_handle->write_segment_meta(meta.get_key(),meta);

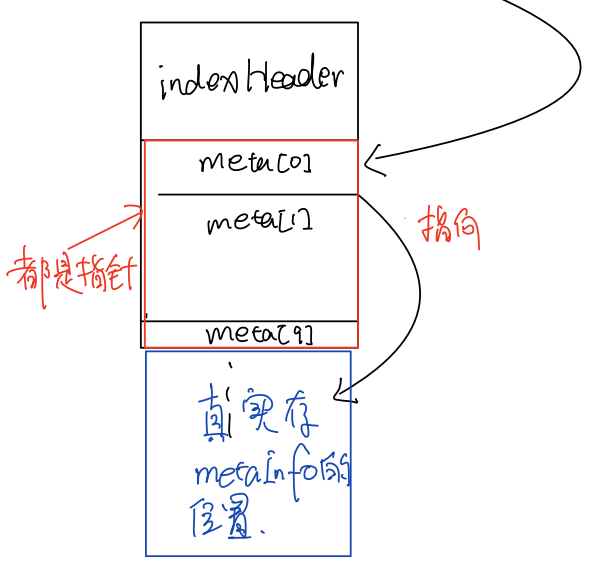
(1) 检查该 file_no 对应的文件是否已经存在了

↓

int IndexHandle::hash_find(const uint64_t key,int32_t& current_offset,int32_t& previous_offset)

↓

找到下标对应桶的开始位置.



(2) 插入数据

//如果不存在就将metaInfo写入到哈希中

ret = hash_insert(key,previous_offset,meta);

↓

☆ 通过 key 得到桶坐标、

☆ 查看可重用链表，看有没有可以重用的位置。

☆ 如果找到位置可以放meta，作为位置将Current记录下来

或者

☆ 通过 index_header()->index_file_size_; 找到第一个可存放meta的位置

☆ 如果该桶中之前有别的meta（桶非空）那么将更新 previous_offset，以及当前meta的next_pos

☆ 将 previous_offset 和当前 meta 写回。

3. 其它
   - ① 同步
   - ② 回收资源、

读文件流程

1. 加载索引文件

① 检查索引文件是否有效以及存在

② 将索引文件进行映射

③ 将相关结构进行检查

↓

文件大小 / 块大小 / "桶"大小

2. 读取文件.(知道 file_no，实际场景可能还存在 String->file_no)

① 使用索引文件将想要读取的 file_no 对应的 metaInfo 给读出来

```
//将对应key的所属meta进行取出
int32_t IndexHandle::read_segment_meta(const uint64_t key,MetaInfo& meta)
{
    int32_t current_offset = 0;
    int32_t previous_offset = 0;

    int32_t ret =hash_find(key,current_offset,previous_offset);
    //说明存在key
    if(ret == TFS_SUCCESS)
    {
        ret = file_op_->pread_file(reinterpret_cast<char*>(&meta),sizeof(MetaInfo),current_offset);
        return ret;
    }
    return ret;
}
```

↓

直接通过 Hash 值找到对应桶，利用拉链法找

② 利用 metaInfo 得到该文件的数据存在主块的位置

③ 通过 mainblock 的 pread_file 得到数据

3. 资源回收