Hash存储引擎

设计目标

- 数据持久化到磁盘中
- 支持快速精确查找
  - 内存效率 O(1)
  - 访问磁盘1次
- 支持内存缓存
- 插入效率要高
  - 异步插入
  - 重做日志
- 支持断电恢复

概要设计

参照Bitcask设计，并进行简化

内存中使用HashMap存储索引

磁盘中存在一个顺序的数据文件。

当有更新或者删除时，不删除原来数据。只需标记旧数据为删除，可选的创建一个新的数据即可。

为了解决数据文件垃圾数据过多，需要在某刻进行文件整理。为了方便整理，将数据文件按照固定大小分块。

内存管理

采用固定式内存管理，主要内存由如下几个部分构成

- HashMap大小
- 缓存内存大小

缓存方案

参见《2-LRU缓存》，重用其实现。需要用户指定占用内存大小

HashMap

简单起见，使用《2-LRU缓存》最大容量设计为整数最大值来充当HashMap。需要用户指定占用内存大小，这将直接影响存储引擎能存储多少条数据

结构为 `<key, value>`

- key 为 主键
- value 为 在磁盘中的位置等信息
  - 版本号
  - 在磁盘中的位置

文件结构

用户根据机器情况指定文件的最大容量。

存储结构是

- 第0块为元数据
- 第1块为位图， `0` 表示未被使用， `1` 表示已使用
- 第2块到第n块，用于存放数据，每一项为一个元组： `<版本号，主键长度，值长度，键，值>`
  - 版本号以1开始依次递增，当该项数据被删除，将版本号置0

为了简单起见，使用固定的块大小 `4M` 。这样数据文件理论最大容量为： `4M*4M+1M` 约 `16T`

基本操作

提供增删改查操作

插入

- 记录到重做日志
- 将数据写入 `写缓存`
- 返回

删除

- 记录到重做日志
- 查找 `读缓存`
  - 若找到，从读缓存中删除，移到到 `写缓存` ，并标记为 `已删除`
- 查找 `写缓存`
  - 若找到，标记为已删除
- 查找HashMap
  - 找到其位置，并写入 `写缓存` ，标记为 `已删除`
- 删除HashMap中这一项
- 返回成功

更新

- 先执行删除
- 在执行插入

查找

- 从读缓存中找
  - 若找到，返回
- 从写缓存中找
  - 若找到，则返回（状态不为删除）
- 从HashMap中确定其在内存中的位置，从磁盘中读
  - 放入 `读缓存`
  - 返回

辅助操作

持久化操作

- 原子的从 `写缓存` 中拿数据，写入磁盘，确定写入完成后，原子的从 `写缓存中删除` 并创建一个索引插入HashMap

正常停止

- 等待所有持久化线程结束
- 将HashMap写入磁盘，创建索引文件
- 程序退出

正常启动

- 读取索引文件，创建HashMap
- 完成

断电重启

- 通过位图遍历整个数据文件，创建HashMap，针对每一个项，若未被标记位删除则插入HashMap
- 执行重做日志
- 完成

垃圾回收

当 `无效项数/总项数` 达到一定阈值时执行垃圾回收

- 按照位图遍历，将其搬移到最后