LevelDB 源码解析-写入

本系列介绍LevelDB,对其源码进行解析。

WriteBatch类

在进行插入更新操作的时候，WriteBatch类负责进行批量的原子性操作。下面来看一看WriteBatch结构：

```
class WriteBatch {
 public:
  WriteBatch();
  ~WriteBatch();

  // Store the mapping "key->value" in the database.
  void Put(const Slice& key, const Slice& value);

  // If the database contains a mapping for "key", erase it.  Else do n
othing.
  void Delete(const Slice& key);

  // Clear all updates buffered in this batch.
  void Clear();

  // Support for iterating over the contents of a batch.
  class Handler {
   public:
    virtual ~Handler();
    virtual void Put(const Slice& key, const Slice& value) = 0;
    virtual void Delete(const Slice& key) = 0;
  };
  Status Iterate(Handler* handler) const;

 public:
  friend class WriteBatchInternal;

  std::string rep_;  // See comment in write_batch.cc for the format of
 rep_

  // Intentionally copyable
};
```

WriteBatch 提供 Put、Delete、Clear 操作。可以看到 rep_ 存储的方式利用的std::string。下面介绍一下存储的结构。

8byte	4byte	1byte	key's length (变长)	key's value	value's length(变长)	value
sequence	操作次数	操作类型	key 的长度	key 的内容	value 的长度	value 的内容

这里 key value 长度的写入是可变长度的，以便节省空间。

```
char* EncodeVarint32(char* dst, uint32_t v) {
  // Operate on characters as unsigneds
  unsigned char* ptr = reinterpret_cast<unsigned char*>(dst);
  static const int B = 128;
  if (v < (1<<7)) {
    *(ptr++) = v;
  } else if (v < (1<<14)) {
    *(ptr++) = v | B;
    *(ptr++) = v>>7;
  } else if (v < (1<<21)) {
    *(ptr++) = v | B;
    *(ptr++) = (v>>7) | B;
    *(ptr++) = v>>14;
  } else if (v < (1<<28)) {
    *(ptr++) = v | B;
    *(ptr++) = (v>>7) | B;
    *(ptr++) = (v>>14) | B;
    *(ptr++) = v>>21;
  } else {
    *(ptr++) = v | B;
    *(ptr++) = (v>>7) | B;
    *(ptr++) = (v>>14) | B;
    *(ptr++) = (v>>21) | B;
    *(ptr++) = v>>28;
  }
  return reinterpret_cast<char*>(ptr);
}
```

举例：

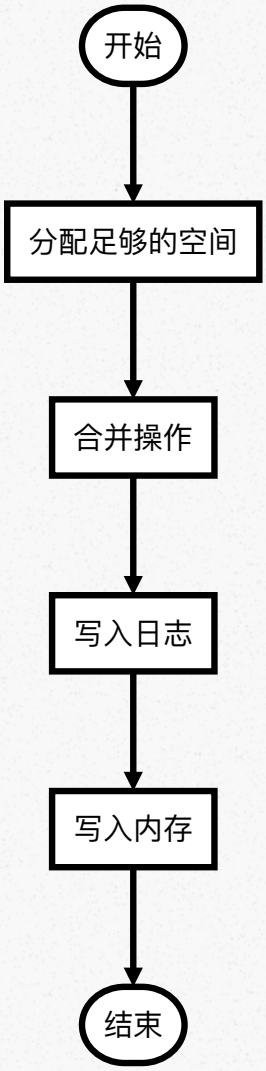
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	0x01 0x00 0x00 0x00	0x01	0x05	0x68 0x65 0x6C 0x6C 0x6F	0x05	0x77 0x6F 0x72 0x6C 0x64
			key		value	

sequence	操作次数	操作类型(put)	key 长度(5)	'hello'	value 长度	‘world’

LevelDB写入操作在函数:

`Status DBImpl::Write(const WriteOptions& options, WriteBatch* my_batch)`

主要流程:



下面逐个说明各个流程:

分配空间

主要在 `Status DBImpl::MakeRoomForWrite(bool force)` 里面。集中在各个状态的判断上。分别是

- level0文件是否超过kL0_SlowdownWritesTrigger(8)个数，如果超过则睡眠1000us, 等待背景线程合并level0文件
- 当前内存表满了，并且imm （不可变）内存仍然在compacted，则等待。
- 如果level0文件超过kL0_StopWritesTrigger(12)文件则等待

- 最后的情况就是当前内存表满了，imm(不可变内存表没有满）则进行切换，并触发compact后台线程

组提交

这部分主要是提交当前版本号以前的数据:

- 首先，将当前版本号以前的WriteBatch合并为一个WriteBatch 并且更新版本号+1
- 写入日志文件
- 写入内存表
- 更新当前版本号之前的Writers队列信息，done的动作由false变为true。
- 通知其它线程