LSM 树（log structured merge tree）将对数据的修改增量保持在内存中，达到指定的大小限制后将这些修改操作批量写入磁盘，读取时需要合并磁盘中的历史数据和内存中最近的修改操作。LSM 树有效地规避了磁盘随机写入问题，但读取时可能需要访问较多的磁盘文件。LevelDB 采用了 LSM 树存储引擎（图 1），主要包括：内存中的 MemTable 和不可变 MemTable（immutable MemTable，也称为 Frozen MemTable）以及磁盘上的几种主要文件：当前（current）文件、清单（manifest）文件、操作日志（commit log，也称为提交日志）文件以及 SSTable 文件。当应用写入一条记录时，LevelDB 会首先将修改操作写入到操作日志文件，成功后再将修改操作应用到 MemTable，这样就完成了写入操作。

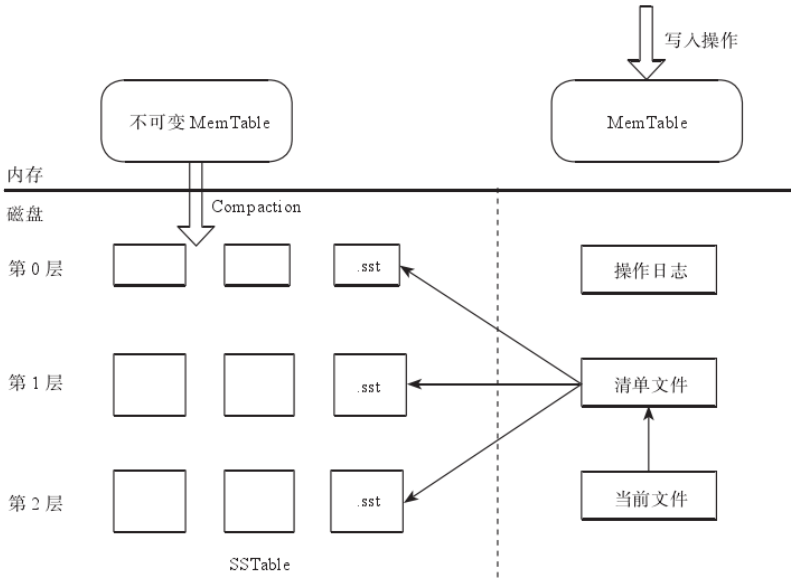

图 1 LevelDB 的 LSM 树存储引擎

当 MemTable 占用的内存达到一个上限值后，需要将内存的数据转储到外存文件中。LevelDB 会将原先的 MemTable 冻结成为不可变 MemTable，并生成一个新的 MemTable。新到来的数据被记入新的操作日志文件和新生成的 MemTable 中。顾名思义，不可变 MemTable 的内容是不可更改的，只能读取不能写入或者删除。LevelDB 后台线程会将不可变 MemTable 的数据排序后转储到磁盘，形成一个新的 SSTable 文件，这个操作称为 Compaction。SSTable 文件是内存中的数据不断进行 Compaction 操作后形成的，且 SSTable 的所有文件是一种层级结构，第 0 层为 Level 0，第 1 层为 Level 1，以此类推。

直观上，LevelDB 每次查询都需要从老到新读取每个层级的 SSTable 文件以及内存中的 MemTable。LevelDB 做了一个优化，由于 LevelDB 对外只支持随机读取单条记录，查询时 LevelDB 首先会去查看内存中的 MemTable，如果 MemTable 包含记录的主键及其对应的值，则返回记录即可；如果 MemTable 没有读到该主键，则接下来到同样处于内存中的不可变 Memtable 中去读取；类似地，如果还是没有读到，只能依次从新到老读取磁

盘中的 SSTable 文件。LevelDB 写入操作很简单，但是读取操作比较复杂，需要在内存以及各个层级文件中按照从新到老依次查找，代价很高。为了加快读取速度，LevelDB 内部会执行 Compaction 操作来对已有的记录进行整理压缩，从而删除一些不再有效的记录，减少数据规模和文件数量。

LevelDB 的 Compaction 操作分为两种：minor compaction 和 major compaction。Minor compaction 是指当内存中的 MemTable 大小到了一定值时，将内存数据转储到 SSTable 文件中。每个层级下有多个 SSTable，当某个层级下的 SSTable 文件数目超过一定设置值后，levelDB 会从这个层级中选择 SSTable 文件，将其和高一层级的 SSTable 文件合并，这就是 major compaction。major compaction 相当于执行一次多路归并：按照主键顺序依次迭代出所有 SSTable 文件中的记录，如果没有保存价值，则直接抛弃；否则，将其写入到新生成的 SSTable 文件中。