moonlsm — MoonBit 生态里的 LevelDB

嵌入式 LSM 存储引擎 · `PingGuoMiaoMiao/moonlsm` · mooncakes 0.2.0 · MIT

> 写入先落 WAL 再返回，`kill -9` 之后已确认的数据一条不丢； 同一个包在 native / wasm-gc / js 三个后端上构建。

痛点

MoonBit 已有两万多个包，数据方向是官方点名的方向，但**没有一个真正的 LSM 引擎**： `moonkv` 是 Bitcask 式的追加写，`moonjournalkit` 只做 WAL 分帧， 两者都不做 SSTable 与 compaction。这意味着在 MoonBit 里做任何需要持久化的数据项目， 都得自己从 WAL 写起； `moonsql`、 `magpiedb` 这类上层项目因此没有可依赖的地基。

边界

做：WAL + SSTable + 三层 leveled compaction + bloom filter + MANIFEST 崩溃恢复 + 可插拔 `Storage` trait（native / wasm-gc / js）。 **不做**：SQL、事务与 MVCC、压缩、value log 分离、分布式。 这五项每一项都能单独吃掉一周，v1 明确排除，并写进 README 的能力矩阵。

API

```
let db = @moonlsm.Db::open(@moonlsm.NativeStorage::new("./data"), 8 * 1024 * 1024)
let seq  = db.put(b"user:1", b"alice")          // 返回即代表已 fsync，不是「已进内存」
let rows = db.scan(b"user:0", b"user:9", 100)   // [start, end)，跨层归并后有序
```

测试策略

把「不丢数据」拆成五层可验收的关卡，每层有独立判据，**上一层绿了才有资格跑下一层**： 211 条单元测试 → 协议连通探测 → 5000 步随机对拍 → 20 轮 `kill -9` → 100 万条性能基准。 崩溃契约 C1–C5（已确认必不丢 / 未确认可丢 / 绝不读半条 / 绝不静默丢 / 前缀一致） 每一条都对应到具体测试；写不进测试的（如「真断电」）明确标注为当前无法验证。

里程碑

版本	内容	状态
v0.1	主链路打通：WAL / SST / MANIFEST / 三层归并，20 轮 `kill -9` 零丢失	已完成
v0.2	可嵌入：API 冻结草案 + 可跑示例，发布到 mooncakes	已发布（0.2.0，2026-10-21）
v1.0	产品化：流式扫描游标、footer 自身 CRC、性能回归门禁	规划中

关键数字

项目	数值
代码规模	5,260 行库与 CLI + 4,112 行测试（测试占 44%）
单元测试	211 条（native）/ 186 条（wasm-gc、js 根包）
顺序写 / 随机写	55,658 / 12,699 ops/s（100 万条，每批 1000 条）
点查延迟 P50 / P99	0.053 / 0.111 ms（命中）；0.040 ms（未命中）
范围扫描 / 空间放大	117,483 rows/s；1.13×（100 万条）
崩溃一致性	20/20 轮，每轮 2.7–6.5 万条写入，丢失 0、半条 0
工程文档	8 篇 ADR、中英双语 README、磁盘格式契约、验收手册、基准报告

可复现

```
git clone https://github.com/PingGuoMiaoMiao/moonlsm && cd moonlsm
moon test                                              # 211 条单测
moon build --target native --release
cd benches/python && python crash_test.py --rounds 20   # 20 轮 kill -9
python demo.py                                          # 90 秒演示，四步带证据
```

CI：Ubuntu / macOS / Windows 三平台跑类型检查与单测，三个后端分别构建；崩溃测试是独立 workflow。

已知局限

不掩饰，都写在文档里：范围扫描比 SQLite 慢一个数量级， 根因是 `Db::scan` 仍是「整段读进数组再归并」而非流式游标；SST 的 data / index / filter block 已加 CRC32C，逐字节翻转 fuzz 的检出率从 **42% 提到 95.7%**， 剩下 8 个位置在 footer 的 `smallest_seq` 上——48 字节固定 footer 里没有位置再塞一个 CRC。

`kill -9` 只杀进程、不清操作系统页缓存，所以它证明不了 fsync 真的调了。 能暴露这件事的是 WAL 尾部截断注入——已实现，见 `corruption_wbtest.mbt`。