

MoonTxnKit 项目申报书

确定性状态事务与恢复内核 - 复审版

一、基本信息

项目名称	MoonTxnKit: 面向 MoonBit 的确定性状态事务与恢复内核
参赛者	马识途
联系方式	【提交前填写: 手机、邮箱】
GitHub 仓库	https://github.com/black-duck666/MoonTxnKit
GitLink 仓库	https://www.gitlink.org.cn/black-duck/MoonTxnKit
项目方向	MoonBit 基础生态库 / 原子状态迁移 / MVCC 与恢复
是否为移植项目	否
当前基础	21+ 次有效提交; 23 项测试; 四后端与 GitHub CI 通过

二、项目定位与真正价值

MoonTxnKit 不是 transactions 工作流程的教学玩具, 也不试图再造 SQL 数据库。项目定位在“直接修改 Map”与“引入完整数据库”之间: 为工作流调度器、规则引擎、模拟器、内存服务和测试替身提供可嵌入的确定性状态提交内核。

这类组件经常需要先检查多个业务事实, 再让一批状态全部生效或全部不生效, 同时能够解释并发冲突、记录提交结果并在重启后恢复。如果各项目自行实现, 会重复出现版本号、临时副本、补偿代码、锁和日志格式。MoonTxnKit 把这组共性收敛为稳定的 MoonBit API, 且不绑定磁盘、线程、网络 and 平台 IO。

三、目标用户与复用场景

1. 工作流与任务调度: 任务仍为 queued 且 owner 不存在时, 才能原子写入 running 与 owner, 多个 worker 竞争时最多一个成功。
2. 库存和资源预占: 库存、订单状态与幂等标记在同一快照检查, 扣减、状态迁移和预占记录使用同一提交版本。
3. 规则与策略引擎: 规则读取的一组事实在结论提交前不能被其他事务改变, Serializable 读集校验负责阻断过期结论。
4. 模拟器与测试替身: 相同命令序列在不同后端得到相同版本、冲突和恢复结果, 适合差异测试与回放。
5. 轻量持久化工具: 核心生成可校验逻辑 WAL, 上层可选择文件、浏览器存储、对象存储或数据库作为保存位置。

四、合理抽象: AtomicPlan

AtomicPlan 用 condition 描述提交前必须成立的业务事实, 用 write 描述必须一起生效的状态集合。Engine::execute 在同一 Serializable 快照读取全部条件; 条件不满足返回 ConditionFailed, 并发验证失败返回 CommitRejected, 两类失败具有不同语义和稳定 JSON 证据。

上层只需要表达“什么必须仍然成立”和“哪些状态必须一起变化”, 无需理解版本链、保存点或 WAL。任一条件失败时版本不会推进, 也不会留下部分写入。该抽象足以表达任务领取、幂等消费、订单状态机、资源租约和策略配置切换, 又没有把业务对象强制建模成数据库表。

五、项目边界与生态关系

项目不实现 SQL、索引、查询优化、文件刷盘、网络复制和分布式共识, 也不宣称替代成熟数据库。它服务于“不值得引入完整数据库, 但不能接受部分状态和不可恢复更新”的 MoonBit 组件。

2026 年 6 月 30 日以 MVCC、snapshot isolation、serializable transaction、write-ahead log、transactional key-value 等关键词检查 Mooncakes。moonbitlang/async 的 retry 处理任务失败重试, 不维护多版本状态; Lampese/lomo 的事务服务于 Loro CRDT 文档模型。未发现以通用原子状态计划、MVCC 隔离和逻辑恢复为组合定位的 MoonBit 包。

六、方案位置对比

1. 直接 Map

更新：依赖最少，但没有一致快照、整批原子性、并发冲突证据和恢复协议，复杂状态迁移最终会散落补偿与回滚代码。

2. 完整数据库：能力全面，但会引入

SQL、存储格式、运行时和部署约束；模拟器、规则引擎、浏览器工具和测试替身往往不需要这些重量。

3. MoonTxnKit：只提供状态事务、冲突和逻辑恢复，保持纯 MoonBit

与后端中立；上层保留领域模型、编码、执行器和持久化选择。

七、核心技术设计

引擎为每个键维护按提交版本递增的版本链。事务固定

snapshot_version，读取不超过快照版本的最新值；待提交写在事务内优先可见。

Snapshot Isolation 使用 first-committer-wins 检查写写冲突；Serializable

模式额外记录并验证读集，阻止检查依据在提交前被并发事务改变。整批写共享一个提交版本。

每个写事务生成带校验值的逻辑 WAL。恢复先验证校验值、提交版本连续性和重复记录一致性，再应用数据；相同记录可幂等跳过，损坏或分叉记录会被拒绝。

历史压缩以最老活跃快照为低水位，保留该快照可见锚点和其后版本，使长读事务在压缩前后仍看到同一状态。

八、创新点

1. 把业务前置条件与底层事务控制解耦，形成可被多个领域直接调用的 AtomicPlan，而不是只暴露 begin/put/commit 教学流程。

2. 把隔离、冲突、日志、恢复和低水位压缩组织成无平台依赖的纯状态内核，允许不同后端自由选择持久化与执行方式。

3. 业务条件失败、并发冲突、WAL 损坏和恢复结果均使用结构化类型与稳定 JSON，便于 CI、调试器和可视化工具消费。

4. 核心不读取系统时钟、不使用随机数，同一命令序列可以在 Native、JavaScript、Wasm 和 Wasm-GC 后端进行差异验证。

九、当前成果与可运行证据

仓库已形成 21+ 次有效提交，覆盖 MVCC 引擎、两种隔离级别、保存点、AtomicPlan、WAL、恢复、压缩、JSON 报告、CLI、文档和持续集成。

23 项确定性测试覆盖快照重复读、写写冲突、写偏斜阻断、保存点回滚、任务原子领取、库存预占、幂等消费、WAL 校验、幂等重放、损坏拒绝和低水位压缩；四个编译后端全部通过。

CLI 同时演示银行转账与工作任务领取：第一个 worker 原子写入 running 和 owner，第二个 worker 收到 condition_failed；随后五条 WAL 在新引擎中恢复，键值、历史版本和统计一致。

十、已知限制

当前键和值采用 String，尚未提供泛型编码适配；Serializable 只验证点读集合，尚无范围扫描和谓词冲突；逻辑 WAL 不直接承诺文件 fsync 或跨进程崩溃原子性。这些限制在 README 和设计文档中明确记录。

十一、后续计划与验收

1. 0.3：增加版本化 WAL 二进制编解码、Bytes/领域编码适配和排序模型随机对照测试。

2. 0.4：探索范围索引、谓词冲突跟踪和 Serializable Snapshot Isolation 依赖图。

3. 0.5：提供文件、浏览器存储和对象存储适配包，建立检查点与崩溃恢复演练。

4. 阶段验收继续以四后端测试、公开 CI、结构化差异输出和带日期的性能记录为准，不使用无法复现的性能宣传。