SQLite 的 WAL 模式与并发控制

叶家炜

Jul 17, 2026

SQLite 仍然是当今最流行的嵌入式数据库，它以零配置、单文件部署以及极低的资源占用赢得了从移动端到边缘设备的广泛认可。然而长期以来，其默认的回滚日志机制在并发读写场景下表现乏力，写操作独占数据库文件导致读请求被阻塞，读者也可能因等待写锁而产生「写者饿死」的现象。WAL（Write-Ahead Logging）模式的引入彻底改变了这一局面，它允许写操作在不阻塞读操作的前提下完成，同时通过快照隔离保证每个读事务都能看到一致的数据版本。本文将系统梳理 WAL 的工作原理、并发模型、关键配置以及实际应用中的避坑指南，帮助读者在生产环境中安全、高效地使用 SQLite 的 WAL 模式。

1　传统 Rollback Journal 模式回顾

在默认的回滚日志模式下，SQLite 每当需要修改数据库页时，会先把原始页面的完整副本写入一个独立的 journal 文件。只有当 journal 文件成功落盘后，实际的数据页修改才会被写入主数据库文件；提交时，SQLite 直接删除 journal 文件以表示事务已完成；回滚时则根据 journal 内容恢复原始页面。这种「写前备份」的策略虽然保证了原子性，却引入了严格的锁协议：写事务在 PENDING 阶段即阻止新的读事务进入，进入 EXCLUSIVE 阶段后更会独占整个数据库文件，导致读写完全串行。典型的高并发读写工作负载因此会遭遇严重的性能瓶颈，写者可能长时间等待已有读事务结束，而新的读事务又在等待写锁释放，形成恶性循环。

2　WAL 模式核心原理

WAL 模式的核心思想是将修改操作先写入独立的日志文件，而不是直接修改主数据库文件。日志文件以「帧」为单位记录每一次页面变更，每一帧包含帧头、页面编号、页面内容以及校验和。主数据库文件、WAL 文件以及共享内存文件三者共同构成 WAL 模式的三文件模型，其中共享内存文件（.db-shm）用于在多进程间共享 WAL 索引，避免重复扫描 WAL 文件。写操作首先在 WAL 文件末尾追加新帧，提交时仅需将 WAL 文件的元数据更新为最新状态，而不必等待数据页落盘到主文件，从而大幅降低写延迟。检查点（Checkpoint）机制负责将 WAL 中已提交的帧批量写回主数据库文件，常见的检查点类型包括 passive、full、restart 和 truncate，不同类型在性能与文件大小控制上各有侧重。

3　WAL 模式下的并发模型

WAL 模式将锁分为读锁、写锁和检查点锁三种状态，读锁允许多个读事务同时持有，写锁则由单个写事务独占，检查点锁在执行检查点时使用。得益于这种锁设计，多个读事务可以与一个写事务同时进行，写事务在追加 WAL 帧时仅需短暂持有写锁，之后即可释放，极大提升了并发度。每个读事务通过 WAL 索引定位到其启动时刻

的最新帧集合，从而实现快照隔离：读事务看不到后续写事务的修改，写事务也无法看到其他未提交的写事务。需要注意的是，写事务在启动时仍需等待所有更早的读事务结束，以避免破坏已有快照；若写事务长时间被阻塞，可能引发写者饥饿，SQLite 通过 WAL 文件大小阈值触发检查点来缓解这一问题。

4 WAL 模式配置与调优

开启 WAL 模式只需执行一行 PRAGMA 语句：

```
PRAGMA journal_mode = WAL;
```

该语句会立即创建 WAL 文件和共享内存文件，并将连接的日志模式持久化到数据库头页，后续连接打开同一数据库时自动继承 WAL 模式。synchronous 参数控制 WAL 帧落盘的严格程度，OFF 提供最高性能但可能丢失最近提交，NORMAL 在多数操作系统崩溃场景下仍能保证一致性，FULL 和 EXTRA 则进一步增加 fsync 调用以换取更强的耐久性。wal_autocheckpoint 参数指定 WAL 文件累计多少页后自动触发被动检查点，默认值为 1000；若工作负载写密集，可适当调大该值以减少检查点开销，但需权衡 WAL 文件持续增长带来的磁盘空间压力。mmap_size 参数允许将数据库文件映射到进程地址空间，减少系统调用次数，对读多写少的场景尤为有效。

5 实际场景与最佳实践

在移动端本地缓存、桌面单机应用以及边缘设备数据采集等典型场景中，WAL 模式均表现出色。读多写少的负载可适当增大 wal_autocheckpoint，并保持 synchronous = NORMAL 以兼顾性能与安全性；写多读少的负载则建议定期手动执行检查点：

```
PRAGMA wal_checkpoint(TRUNCATE);
```

该语句会将 WAL 中所有已提交帧写回主文件并截断 WAL 文件，避免文件无限增长。跨进程访问时需确保所有连接使用相同的文件路径与锁协议，否则可能出现文件锁冲突；备份 WAL 模式数据库时，应先执行「备份 API」或手动检查点后再拷贝主文件，以免备份出不一致的 WAL 状态。

6 WAL 的局限与替代方案

WAL 模式依然存在单写者限制，即同一时刻只能有一个写事务在追加 WAL 帧；跨网络文件系统或分布式存储时，文件锁的可靠性下降，可能导致数据损坏；移动设备闪存的磨损问题也因频繁追加 WAL 帧而加剧。SQLite 3.35 引入的 BEGIN CONCURRENT 实验特性尝试在单文件内支持有限的多写者并发，但仍处于开发阶段。对于需要多主复制或水平扩展的场景，Litestream、rqlite 或 Dqlite 等外部方案提供了更成熟的分布式能力；当数据规模或并发度超出 SQLite 设计边界时，迁移到 PostgreSQL 或 MySQL 仍是更稳妥的选择。

WAL 模式通过将修改先写入日志文件，实现了读写并发与快照隔离，显著提升了 SQLite 在高并发场景下的表现。实际部署时，建议先使用 PRAGMA journal_mode = WAL 开启模式，再根据读写比例调整 synchronous 与 wal_autocheckpoint 参数，并定期监控 WAL 文件大小与检查点执行情况。遇到性能瓶颈时，可尝试增大 mmap_size 或改用 BEGIN IMMEDIATE 显式加锁以减少锁竞争。掌握这些配置与最佳实践后，开发者即可在嵌入式与边缘计算领域充分发挥 SQLite 的轻量与高效优势。