面向嵌入式内存数据库SQLite的非易失性内存存储技术

**技术领域**

本发明属于嵌入式内存数据库的技术领域

**背景技术**

1.内存数据库存储简介

磁盘数据库是把磁盘作为数据库的存储介质，通过在磁盘上对数据库进行管理。磁盘数据库需要频繁引入 I/O 操作。同时磁盘设备本身就是要依靠机械运动访问数据。近年来，内存容量不断提高，以及计算机进入 64 位时代，支持更大的存储空间，那么对数据库系统实时响应能力要求日益提高，充分利用内存技术提升数据库性能成为一个研究热点。 内存数据库可以利用内存的高速读写特性提高访问效率，同时，根据内存数据库的新特性，根据内存的特性重新设计了数据库引擎。在对数据的缓存、快速索引算法、提高数据库并行能力方面改进。 内存数据库通常在系统运行时将整个数据库加载到内存中，从而提高数据库运行时的性能。典型的内存数据库有 Mcobject 公司的 e XtremeDB，Oracle 公司的 TimesTen和 Redis等。 e XtremeDB 内存实时数据库主要应用于实时系统以及嵌入式系统。e XtremeDB与大部分实时内存数据库不同，此内存数据库是为嵌入式系统而设计，不是通过内存数据库修改而来。 Oracle 内存数据库 TimesTen 是一个关系数据库，主要是针对于内存进行了优化，目前在企业中大部分应用。其特性是具有很高的实时性和非常高的吞吐能力。其利用标准的 SQL 语句进行操作，具有很高的通用型。 Redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sortedset --有序集合)和 hash（哈希类型）。这些数据类型支持的操作较多，例如 push/pop、add/remove及取交集并集和差集及等，同时为了保证一致性，这些操作都具有原子性的。在此基础上，redis 支持各种不同方式的排序。Redis 根据配置文件有不同的持久化机制，例如定期的把数据库的镜像文件写入磁盘或者把更新数据库的命令追加到日志文件末尾，或者主从机备份方式。 这些内存数据库的设计实现都是面向 DRAM。由于 DRAM 具有掉电丢失数据的性质，目前的内存数据库只能将数据库暂存在内存中，而真实数据只能持久化地保存在磁盘或者 SSD 等外存中。因此，数据库在打开和关闭时，都要在内存和外存之间传输大量数据，其慢速 I/O 操作会极大地降低性能。 此外，为了保证数据的持久性和一致性，现有的内存数据库在运行过程中，需要把修改的内存数据备份到磁盘中。以 redis 为列，其提供两种持久化数据库的机制。一是快照处理（snapshotting），在 Redis 的实现中成为 RDB 模式，此时Redis 在指定时间间隔内生成数据库在内存中的数据集的快照，并写回外存中。然而，RDB 模式不能保证最近一个时间间隔内更新操作的持久化和一致性。如果在时间间隔内系统掉电，则 Redis 就会丢失最近一个时间间隔内的全部数据库更新操作。为了提供更好的数据持久化和一致性保障，Redis 提供另一种称为追加文件（Append-Only File, AOF）的机制。AOF 机制可以把每一次数据库的更新操作都以日志的形式持久化地记录在外存中。然而，AOF 会发起大量 I/O 操作，极大地降低系统性能。

2.非易失性内存技术

DRAM 是具有高效的读写的特性，同时又字节可寻址。这些特性使得其存取效率和 CPU 相匹配。但是 DRAM 也有一个弊端就是其易失性，掉电后存储的数据都会丢失，这也是为什么每次重启需要很长时间重新加载数据的原因。在互联网数据量迅速发展的促使下，数据的实时处理能力的需求不断提高。同时为了克服 DRAM 的易失性，非易失性内存技术也在迅速发展。所以，随着对存储速度的需求不断提高，类似于内存的非易失性存储技术也迅速发展。例如 3D Xpoint和STT-RAM等。 有新的突破性技术以挑战者姿态进入市场，特别是诸如电阻式RAM(RRAM)和相变 RAM(PCRAM)等非易失性内存(NVM)，它们承诺提供高性能、低功耗、以及无限的使用寿命。磁 RAM(MRAM)也是这些新兴技术之一。 MRAM 有了很长的发展历史，早在 25 年前就已经有人在研究。这种存储设备和基于电荷来存储信息的存储介质有很多不同，其通过物理变化等变化来存储信息。目前 MRAM 被应用于不同领域，包括场开关和热辅助等。 自旋转移力矩（STT）MRAM 在存储技术中应用非常广泛。它具有低功耗和低成本特点，同时有具有高密度的制造工艺。这种存储技术可以在掉电后依然保存数据，具有持久性。 另外一种 3D XPoint 技术发展更加迅速。3D XPoint 抛弃了在 NAND 芯片的核心-----晶体管。NAND 技术存在一个很大的问题，对于单个 bit 来讲，它不能做刷新操作。因此，如果要对每个 bit 数据进行更改就需要对整块数据做刷新操作。 3D XPoint 的工作原理与 NAND 存在着根本性的不同。NAND 是通过电子来定义 bit 数据的，而 3D XPoint 则不同，其通过存储单元的电阻大小来定义 0 和 1，其通过改变电阻的大小来改变存储的数据。截止到目前为止，非易失性内存技术日渐成熟，为该发明使用奠定了基础。

3. SQLite整体描述

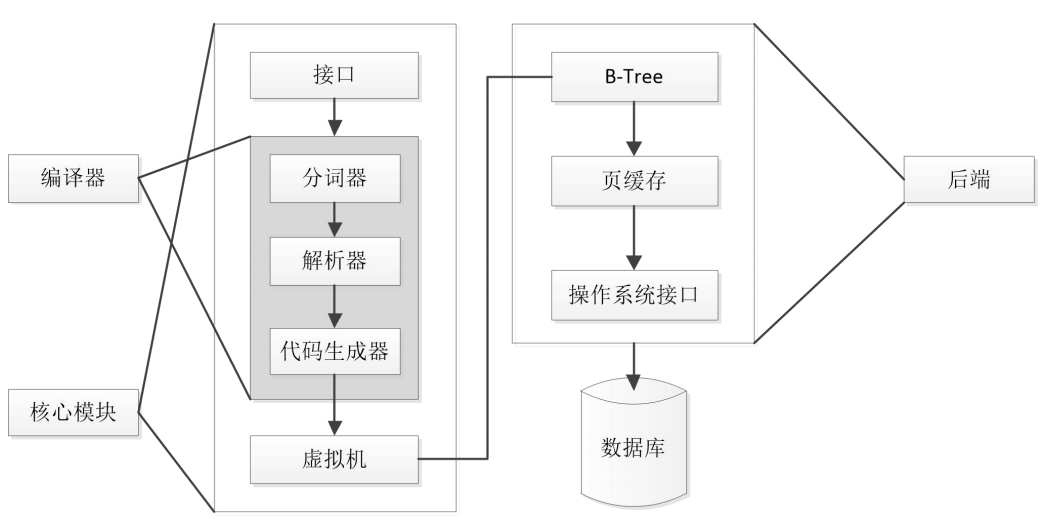


图1. NVMSQLite嵌入式内存系统架构示意图

SQLite 的体系结构具有模块化特点，它的整个体系结构可分为前端解析系统和后端引擎两个部分，也可细分为 3 个子系统和8个相互独立的模块，具体结构如图1.所示。前端解析系统包含三个模块，分别为分词器、解析器、代码生成器，主要负责的工作是 对应用程序传递过来的字符串进行处理，对其中包含的 SQL 语句和命令进行分析、优化和转 换，最终将其变为虚拟机能够执行的内部编码。而后引擎由虚拟机、B-Tree、页缓存和操 作系统接口构成，其中虚拟机位于核心位置，为 NVMSQLite的引擎，用来解释执行前端解析系统生成的内部编码。NVMSQLite的整个体系结构将一次数据库数据查询过程分割为几个不连续的阶段，与工厂流水线上流程类似，在体系的顶端对查询语句进行编译，中间阶段去解释执行， 最终的底部则是处理数据在操作系统中的存储。

**发明内容**

针对嵌入式内存数据库SQLite现有技术不足，解决现有技术在断电后数据丢失问题，本发明提供了一种基于开源SQLite内存数据的非易失内存存储技术。以下称添加非易失内存存储技术的嵌入式内存数据库为NVMSQLite。

接下来将展示非易失内存存储技术的实现细节：

1. NVMSQLite整体架构

NVMSQLite和SQLite整体架构基本一样，不同之处在于，NVMSQLite的数据库是运行在非易失性内存上，而且操作系统接口是面向非易失性内存的。

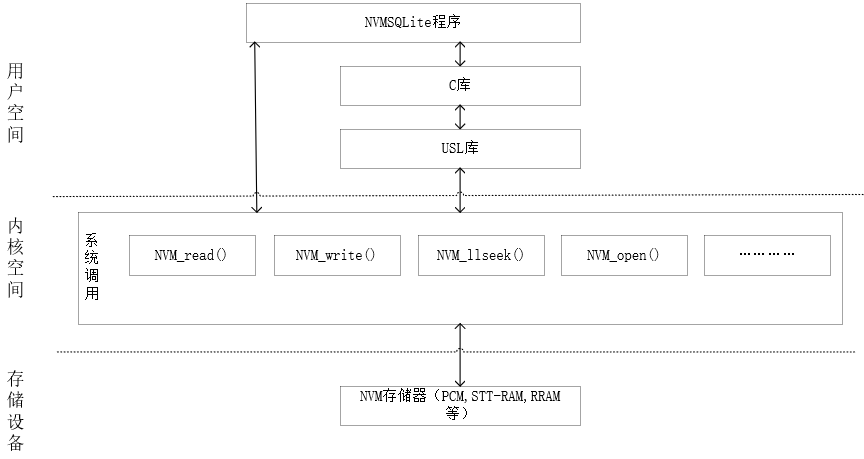


图2. NVMSQLite存储访问路径示意图

NVMSQLite存储访问路径如图2所示。NVMSQLite程序包括运行必要资源，不包括数据库数据， 作为一个进程运行在用户空间中，NVMSQLite数据库的元数据和数据存储在NVM存储器当中。和原来SQLite相比，NVMSQLite精简了传统磁盘文件系统I/O栈，重新设计了访问NVM存储器空间的I/O流程，因而需要在操作系统中添加访问NVM存储器空间的系统调用，并设计了新的库文件对NVM系统进行封装。

2.NVMSQLite在非易失性内存中空间分配



图3**.** 非易失性内存空间分配示意图

NVMSQLite使用基于页的方式管理非易失性内存，即将非易失性内存按照每页4KB的大小进行划分，并用空闲链表将空闲页进行管理。非易失性内存空间分配如图1所示，NVMSQLite将非易失性内存空间分为三个部分，非易失性内存元数据，NVMSQLite元数据，NVMSQLite数据库存储区。

非易失性内存元数据包括非易失性内存大小，非易失性内存划分页的大小，非易失性内存划分页的数量，非易失性内存中页的分配状况。其中非易失性内中页的分配状况使用位图表示，页位图区使用比特的方式记录了页表中页空间使用情况，比特“0”表示页为空闲状态可以用来分配，比特”1”则表示页已被使用。页表存储了文件系统中所有的页信息。

NVMSQLite元数据包括：持久化数据库的映射表，数据库的大小，整个数据库数据结构的索引和数据库的日志。映射表用来组织持久化内存数据库中使用的物理内存空间。其结构与系统页表的形式相同。当对NVMSQLite对应进程的持久化内存数据库初始化时，非易失性内存上的持久化数据库通过将映射表插入对应进程的页表中而暴露给进程的虚拟地址空间。然后，用户可以通过其进程虚拟地址空间访问非易失性内存上的持久化数据库，获取进程的虚拟地址空间的大小等于非易失性内存分配给持久化内存数据库的大小。

3 非易失性内存空间的分配回收策略

，采用空闲链表的方式对非易事行内存的空闲空间进行维护，每个页面的大小设为4KB。在NVMSQLite小于4KB的部分使用NVMslab分配器进行分配。NVMslab类似于linux中的slab分配器，最细粒度设为一次分配32B，配合着页面掩码。一个页面4KB，对应的页面掩码为128位。每当要分配一段小于4kB的空间时，首先判断分配器里面的空闲空间是否能满足要求，如果不能满足要求，就从空闲页面链表中获取一个4KB的物理页给NVMslab，然后再分配。

4 保证数据库一致性

当NVM持久化存放在NVM中后，对内存数据库的操作不再需要经过DRAM。对数据地任何修改操作都直接持久化地保存在数据库中，不需要再同步回磁盘中，省去全部地慢速I/O操作。同时也不存在与备份数据不一致的问题。

在NVMSQLite中提出一种基于日志方式，先来备份数据集更新操作本身，然后再做更新操作。在NVMSQLite元数据空间区中，划出一部分区域为日志区域。NVMSQLite嵌入式内存数据库每进行一步操作时，每一步操作作为一个条目记录在日志里面。存储一个日志指针，指针指向日志中当前操作的条目，指针所指之前地日志条目指代的操作全部失效，每完成一步操作，就往后挪一步指针。当日志区域全部用完时，指针从头开始。

数据库系统举例

1.NVMSQLite空间管理

为了便于管理，非易失性内存的空闲空间主要由空闲链表维护，每个页面的大小设为4KB。此外，对于嵌入式内存数据库NVMSQLite使用的大都是小于4KB的内存空间。为了提高非易失性内存的空间利用率，使用细粒度内存分配器和页面掩码辅助管理非易失性内存的物理空间。

NVMslab，用于分配小于4KB的非易失性内存空间，最细粒度设为一次分配32字节。每当要分配一段小于4KB的空间时，首先判断分配器里面的空闲空间是否满足要求。如果不能满足要求，就从空闲页面链表中获取一个4KB的物理页给NVMslab，然后在进行分配。

页面掩码的结构类似于数组，每个元素对应一个物理页，存放该页面的空间使用情况。每个元素是一个位图。由于默认要分配的空间以32字节为基本单位，所以一个4KB大小的空闲页对应在页码中的位图大小为128位。每从一个页面分配32字节，页码掩码中对应的位置为0；每从某个页面释放32字节，页面掩码中对应的位置为1.

每次请求分配非易失性内存空闲空间时，首先判断请求的大小。当请求的空间大于4KB时，以4KB单位向上取整，从空闲链表中直接分配连续的空闲块，并调整空闲链表中的基本信息。空闲链表的分配策略时首次适应的方式分配。如果要分配的空间小于4KB，就用细粒度内存分配器NVMslab。整体的逻辑如图四所示

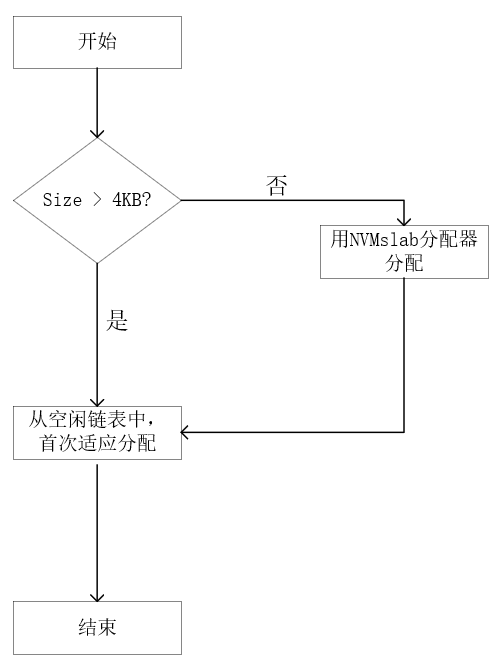


图4. 内存空间分配流程图

当释放一段存储空的时候，需要把对应的空间回收。首先对位图标志位调整，把位图中对应的标志位位1.。然后判断所在标志位对应页面的标志位是否全为1，如果全为1就说明整个页已经空了，就把这个页插入到空闲链表中，如果不是全为1，说明这个页面还有空间没有被回收，所以停止操作。其流程如图5所示

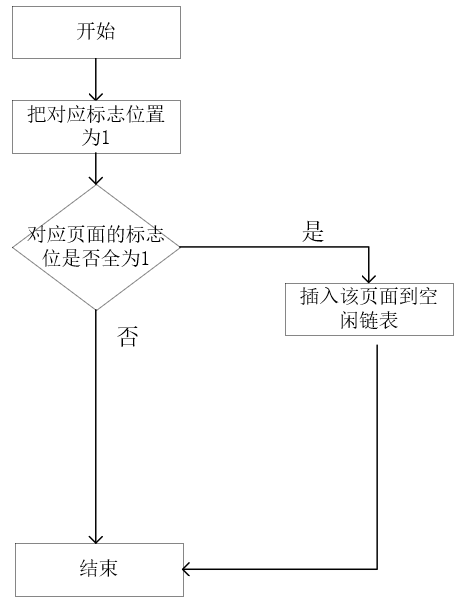


图5. 回收操作流程图

2.主要数据结构架构

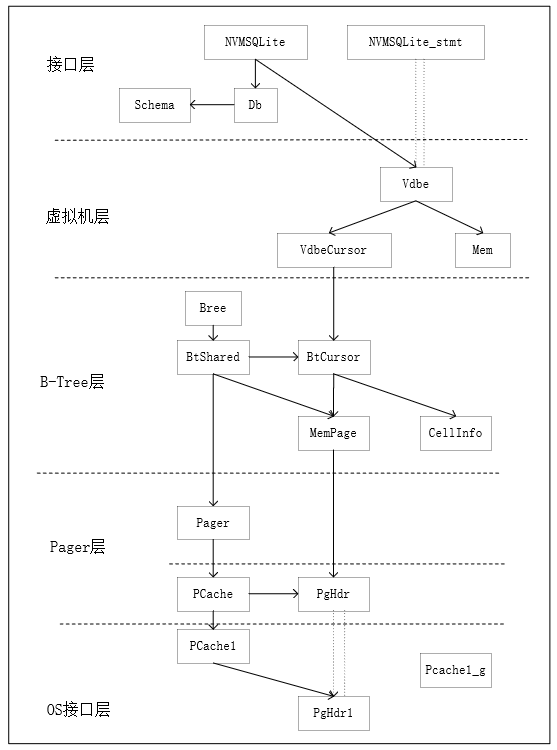


图6.NVMSQLite的主要数据结构

NVMSQLite结构代表的是“数据库连接”。每个数据库连接都是是NVMSQLite结构体的一个实例，在应用程序中可以连接多个数据库文件。结构成员aDb是Db结构对象，表示连接的数据库文件。在实际使用中，aDb表示为Db结构指针的数组，编号为0的是主数据库，为aDb[0];编号为1的是临时数据库，为aDB[1]；编号2以及之后的是附加数据库。

Db结构代表的是数据库连接中的每个数据库文件实例。结构中的Btree对象和Schema对象，表示数据库连接为每一个连接中的数据库文件建立一个Btree对象和一个Schema对象。

Vdbe结构是虚拟机的实例，包含了虚拟机的全部状态。应用程序中的“NVMSQLite\_stmt\*”指针实质上指向的就是Vdbe结构体的实例。每个数据库连接中可能同时存在多个虚拟机的实例，这些虚拟机的实例以双向链表的形式组织在一起。每个Vdbe结构中都有指向虚拟机前驱和后继的指针pPrev和pNext。

Mem结构中保存了同一个字段值的多种数据类型，如NULL，INTERGER，REAL,TEXT,BLOB等。在NVMSQLite内部，VDBE使用Mem结构处理几乎所有的SQL值。

VdbeCursor结构是Btree游标在虚拟机层的表现形式。虚拟机打开的每一个游标都是此结构的一个实例。

Btree结构是私有的，即连接中的每一个数据库文件都有自己所独有的Btee对象。数据库连接看不到Btree结构的内部，只能透过过指针来操作此结构。

BtShared结构实例表示的是一个独立的数据库文件。在实际情况中，一个数据库文件可能被两个或者更多的数据库连接所使用；此时，每个连接都有自己私有的Btree对象，但是所有的Btree对象指向的都是同一个BtShared对象。

BtCursor结构是Btree游标的实例。每一个数据库文件都是通过自己私有的Btree游标来访问数据库内容的。一个数据库文件可以被多个数据库连接所共享，但是游标确实是私有的，不能共享。

CellInfo 结构的实例存储着与单元数据相关的信息，如单元指针、键值、单元数据的大小、记录头的大小、溢出页链表的首指针等。

MemPage 结构的实例存储着从原始文件页中解析得到的页的相关信息，如页的类型、页头的起始位置、页编号、本页的单元个数等。

Pager 结构的实例代表的是一个打开的页缓冲区，存储着页缓冲区以及相关的一些文件信息等，如页缓冲区指针、数据库文件名、日志文件名等。如果定义了数据库加密宏“SQLITE\_HAS\_CODEC”，则 Pager 结构体中将包含有与加解密功能相关的结构成员（加解密函数指针 xCodec、加解密密钥块 pCodec 等）；因此，NVMSQLite加解密功能的实现是通过 Pager 层来加解密数据库中的每一页的。

PCache 结构表示的是一个完整的页缓冲区，是用于内存分配的接口结构。用户可以根据实际需求选择使用默认的内存分配机制或替换内存分配系统。

PCache1 结构表示的是默认的内存分配系统，存储着默认内存分配的缓冲区的相关信息，如默认缓冲区大小、缓冲区是否可净化等。

PgHdr 结构的实例控制着缓冲区中的页，存储着页的相关信息，如页编号、页内容、页的用户数、拥有此页的缓冲区指针等。

PgHdr1 结构实例表示的缓冲区的入口，即缓冲区中页的存取地址。

3． 数据库元数据存储

如图6所示，一个NVMSQLite数据库的存储最高层结构为DB结构体，其中包含了每个数据库存储文件。如果每次系统启动时都能够找到DB结构体，就可以进一步检索到整个数据库。所以在数据库元数据区存储DB结构体。

在启动NVMSQLite时，有三个主要的元数据结构用于定位数据库。第一个元数据结构式映射表。映射表映射了NVMSQLite的数据集占用的物理空间以及NVMSQLite的数据集数据结构的物理空间。当NVMSQLite启动的时候，映射表嵌入到NVMSQLite相对应的进程页表中。这样NVMSQLite的进程能够访问用户空间中的数据库。

第二个元数据结构是NVMSQLite的虚拟空间管理。我们保留一个虚拟地址空间，其大小等于非易失性内存大小。虚拟空间管理的方式与linux中使用的伙伴系统相同。每次数据库提交更新请求时，NVMSQLite会为请求分配一段空闲虚拟空间。空闲虚拟空间大小等于或对其于非易失性内存物理存储器的物理空间。当数据库被删除时，想用的可用空间将被回收。

最后，需要的元数据结构用来指示整个数据库组织结构的地址，这样可以使得数据库的整个组织结构暴露于虚拟空间地址。

NVMSQLite启动后，首先就是要初始化全局服务器配置，然后加载配置文件来初始化服务器。当服务器初始化完成后加载数据库。经过上面的更改后，NVMSQLite启动与SQLite不同点在于，对服务器初始化时，直接通过数据库的元数据来初始化DB结构体。后面也不需要加载RDB以及AOF文件加载数据库。数据库的启动效率有大大的提升。

本发明公开了一种面向嵌入式内存数据库SQLite地非易失性内存存储技术。对开源数据库SQLite进行拓展，增加对非易失性内存存储技术。该技术中对非易失性内存空间区域进行划分，并利用空间链表对空闲页进行管理。当需要分配对于少于一页的空间时，用NVMslab分配器进行分配。在内核空间中对增加面向非易失性内存的系统调用，即cpu可以直接用指令访问非易失性内存，进行字节性管理。采用先记录操作，后更新数据的方式，保证数据的一致性。