Android 数据库 SQLite 性能优化①

林培杰, 朱安南, 程树英

(福州大学 物理与信息工程学院, 福州 350108)

摘 要: SQLite 是 Android 平台的重要数据库引擎, 具有零配置、支持事务、移植性好等特点, 负责多种格式数据的存储. 随着用户数据规模的扩大与复杂度的增加, 对数据库性能提出更高的要求. 数据库性能直接影响应用程序的性能和用户的体验. 本文分析了 SQLite 事务机制和索引结构, 并通过手动添加事务和合理创建 B-tree 索引, 大大缩短了 Android SQLite 数据库的插入与查询时间, 提高了 SQLite 性能.

关键词: Android SQLite; 性能优化; 事务; 索引

Performance Optimization of Android Database SQLite

LIN Pei-Jie, ZHU An-Nan, CHENG Shu-Ying

(College of Physics and Information Engineering, Fuzhou University , Fuzhou 350108,China)

Abstract: SQLite is an important database engine on Android platform. It has features of zero-configuration, supporting transactions and excellent portability. And it is responsible for multiple formats of data storage. With the expansion and complexity of the user data, there is higher demand upon the SQLite performance which may affect the performance of the application program and the user experience. This paper analyzes the transaction mechanism and the index structure of SQLite. By means of turning on transaction manually and creating B-tree index properly, it greatly reduces the time needed to insert and query data from Android SQLite, which improves SQLite performance ultimately.

Key words: Android SQlite; performance optimization; transaction; index

随着嵌入式移动设备的普及和用户数据的高频存取, 数据库性能得到日益重视. SQLite 是由 D.RichardHipp 开发并于 2000 年发布的一款开源、轻量级的关系型数据库, 占用极少的内存资源, 并具有良好的可移植性[1]. 目前, SQlite 数据库广泛应用于主流的移动设备[2], 如 iPhone 和 Android 平台利用 SQLite 数据库引擎实现结构化数据的存储, 并适用于多种数据类型的存储, 如联系人、Notes 和用户配置等等. 数据库操作的执行时间, 关系到整个应用程序的性能. 特别是在 Android 应用程序中, 如果插入或更新数据库操作时间过长, 则会出现应用程序无响应(ANR, Application Not Responding), 势必降低用户体验. 因此, 除了将数据库的操作放在新的线程, 避开 UI 主线程, 对数据库的插入、查询等重要操作进行时间优化, 显得尤为重要.

1 Android SQLite简介

Android 在运行环境中集成了 SQLite 数据库, 可供每个应用程序使用且无服务进程[3]. 由于支持 SQLite 的 Dalvki 虚拟机是基于 Java 的, 因此需对 SQLite API (Application Programming Interface, 应用程序编程接口) 重新封装为可供 Android 层使用的 API 接口. SQLiteOpenHelper 类是 Android 为 SQLite 数据库提供的一个重要抽象类, 用于管理数据库的创建、版本更新、打开和关闭等操作. SQLiteOpenHelper 工作流程如图 1 所示, 对数据库进行操作时, 首先检查目标数据库是否存在, 若不存在, 调用 onCreate()方法创建数据库; 其次检测版本号, 若与构造函数中传入的

① 收稿时间:2013-08-19;收到修改稿时间:2013-09-10

版本不一致, 则执行 onUpgrade()方法更新数据库版本. 通过调用 SQLiteOpenHelper 的 getReadableDatabase()或 getWriteableDatabase()方法, 打开并返回一个可读/可写的数据库实例 SQLiteDatabase db. 用户通过 db 对象完成数据库的添加(insert)、删除(delete)、修改(update)、查询(select)等操作.

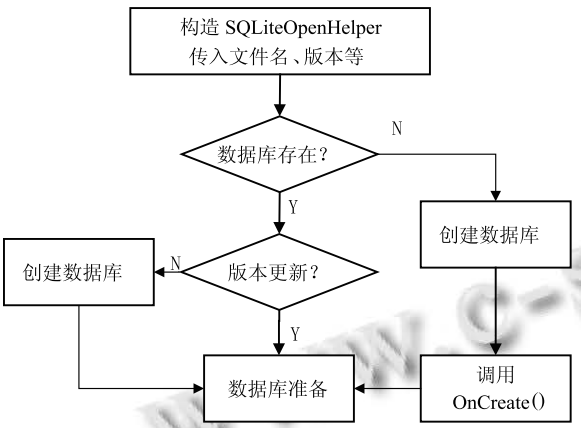

图 1　SQLiteOpenHelper 工作流程

Android 应用程序所创建的 sqlite 数据库是私有的, 并保存在/data/data/packageName/database 目录下. 数据库查看方法有: ①利用 adb 工具访问 SQLite 数据库, 运行 cmd 后执行“adb shell”进入 shell 命令模式并找到数据库目录, 再根据 SQLite 命令对数据库进行查询、添加等操作; ②使用 DDMS 导出 SQLite 数据库, 再通过图形界面管理工具(sqlite administrator 等)查看具体数据. 特别需要注意的是, 在真机调试时, 需要获得 root 权限方可获取数据库文件及数据.

2　SQlite事务处理

2.1 事务机制与锁

Android 平台特别注重 SQLite 事务管理, 而良好的事务机制是保证数据库性能的关键[3]. SQLite 事务的操作, 利用了“原子提交”的重要特性, 意味着在同一个事务中的所有操作可能被完整地完成或不完成, 不存在仅有部分完成的情况. 用户可以根据需要指定事务的持续时间, 即执行相当任务后, 再提交事务. 而事务与锁在查询处理数据库时密切相关, 通过锁来确保数据库原子提交特性.

SQLite 事务与锁的关系如图 2 所示[4]. SQLite 有五种锁状态: 未加锁(UNLOCKED)、共享(SHARED)、保留 (RESERVED) 、 未 决 (PENDING) 和 独 享 (EXCLUSIVE). 所有事务初始为未加锁状态, 在数据的插入前需要读取数据并检查数据格式, 此时需获得该数据库文件的共享锁, 避免其他的写操作. 在数据修改前需要获取保留锁, 并生成回滚日记, 确保在一个时间点上, 仅有一个进程去写入数据, 以及在异常情况下恢复原始数据. 接着将保留锁提升为未决锁, 获取独享锁, 避免其他的读写操作, 再将更改过后的缓存数据更新到存储器中. 最后释放独享锁, 完成事务的提交. 每次事务需要经过“事务打开”、 “执行事务”、 “关闭事务”, 对于自动提交事务, 所有插入操作均各自经历完整的状态转换为:

UNLOCKED-->PENDING-->SHARED-->UNLOCKED-->PENDING-->SHARED-->UNLOC-KED…

如果采用手动提交, 执行提交命令后会让 SHARED 状态直接转为 UNLOCKED, 所有操作在同一个共享状态执行, 其状态转换路径为:

UNLOCKED-->PENDING-->SHARED-->UNLOCKED

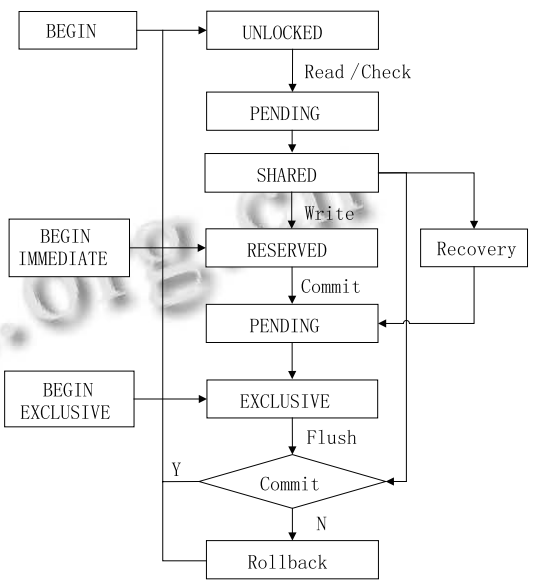

图 2　SQLite 事务与锁

2.2 事务优化

Android SQLite 是一种储存在单一磁盘文件中的数据库. 因此, 对它的访问、读写等操作, 实质是对文件的不断访问与读写, 频繁的操作将耗用大量进程时间, 极大影响数据库的存取性能[5]. 每次操作默认启动一个事务, 而通过手动添加事务, 能够有效避免频繁的打开和关闭事务. 首先调用 SQLiteDatabase 的

beginTransaction()方法开启一个事务, 接着执行批量的操作, 并调用 setTransactionSuccessful() 方法设置事务的标志为成功, 最后调用 endTransaction() 方法提交本次事务范围内的所有操作. 当然如果执行出现异常情况或者没有设置事务成功标志, 则会回滚数据, 撤销了当前的所有操作. 手动事务部分相关代码如下:

```
db.beginTransaction();
//启动一个事务
try{
    ……
 //执行批量处理操作
    db.setTransactionSuccessful();
//设置事务处理成功标志
  }catch(Exception e){
        e.printStackTrace();
        }finally{
        db.endTransaction();
//提交本次事务
  }
```

在联想 A656 手机上测试, 通过多次插入相同数据, 比较了手动提交事务与自动提交事务的时间性能. 测试结果如表 1 所示, 测试表明: 在插入条数较少情况下, 手动提交事务并没有多大改善; 而当插入条目很大情况下, 时间的节省就显得很可观. 由此可见, 使用了手动事务, 将当前事务的所有插入数据添加到缓存, 在事务提交时再一次性写入数据库文件, 大大减少了 I/O 操作, 极大地提高了数据库的效率.

表 1 事务时间测试

插入数目(条)	自动提交	手动提交
1	53.4ms	38.29s
5	222.4ms	49.5s
10	379.5ms	40.1s
100	3.8s	0.076s
1000	38.2s	0.25s
5000	190.3s	0.8s
10000	384.2s	1.6s

3 查询优化

查询是数据库最频繁的操作, 关系到数据库性能的重要指标. 而影响数据库查询的因素一般有: 检索数目、排序、索引和查询语句. 例如, 在 Android 中提供多种查询方式, 而通过 rawQuery()方法直接执行原始 SQL 语句, 比封装过后的方法 query()速度更快. 对于任何 DBMS(Database Management System, 数据库管理系统), 索引是进行优化的最主要因素. 合理的添加索引, 可以更有效地提高复杂数据的查询速度. 本节重点介绍通过创建索引提高查询效率.

3.1 SQLite 索引

SQLite 是一款轻型、快速的数据库, 然而随着查询数据变得更大、更复杂时, 使其查询时间相形见绌. 数据库的索引好比书籍中的目录, 是一种为用户快速找出满足指定条件的数据, 旨在提高数据库的查询效率. 通过创建索引将数据的一列或多列有序排列, 只需扫描少量的索引页和数据页, 从而节省了大量时间. 甚至可以利用索引实现多表的高速连接和减少排序时间.

SQLite 分别使用 B-tree 和 B+tree 处理索引和数据表. B-tree 索引具有较高的存储效率和优越的索引结构, 只存储关键字段的值和对应记录的 rowid 值, 加快了存取速度. 在 B-Tree 中包含一个根节点、若干个分支节点和叶子节点. B-tree 查找是一个沿着指针交叉查找结点和关键码的过程. 查找关键字 k 的算法为: 将 k 与根节点中的 key[i](1≤i≤n)进行比较: ①如果 k=key[i], 则查找成功; ②如果 k<key[1], 则沿着指针 p[0]所指的子节点继续查找; ③如果 key[i]<k<key[i+1], 则沿着指针 p[i]所指的子节点继续查找; ④如果 k>key[n], 则沿着指针 p [n]所指的子节点继续查找, 并以此类推. 以图 3 为例具体介绍 B-tree 索引结构及查询流程. B-tree 每个结点是多关键码的有序表, 如查找关键词 96, 首先 p 指向仅含有一个关键字的根节点(a), 关键字小于 96, 则指针指向分支节点(c), (c)节点的关键字 58 和 77 均小于 96, 因此 p 指针指向叶子节点(i), 并按顺序查找到关键字 96.

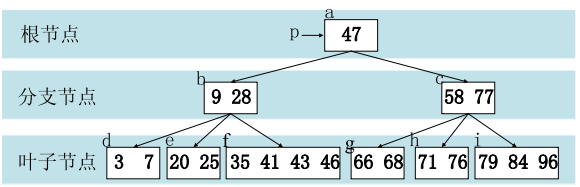

图 3 B-tree 索引结构示意图

3.2 索引测试

对存在磁盘上的索引查找, 产生I/O消耗, 而最大限度的减少磁盘读取操作是衡量索引优劣的重要指标[6].

由 B-Tree 结构分析, 每次检索最多需要(h-1)次 I/O 操作. h 为树的深度, 而且一般很小. 而是否在数据库中创建索引, 决定于多方面因素. 特别是大规模的数据库中, 更需要建立索引来缩小查询时间. 本文在 Android 手机上对存放不同规模数据的数据库进行查询测试, 测试平均时间如表 2 所示. 测试结果表明: 随着数据库规模的扩大, 利用索引查询可以节省大量的时间, 加快数据库访问速度, 提高了整个应用程序的性能.

表 2 索引查询时间测试

规模(条)	自动提交	手动提交
10	1.1ms	0.91ms
100	1.4ms	0.93ms
1000	1.9ms	1.1ms
10000	11.2s	1.2s
100000	103.2s	1.4s

4 结语

SQLite 数据库是一款轻量级嵌入式数据库, 广泛应用于 Android 等移动设备. 数据库性能调优旨在适应数据库规模的扩大化和复杂化, 缩短查询、插入等数据库操作的时间, 提高应用程序性能. 而 SQLite 数据库实际是磁盘上的文件, 频繁或大量的对文件操作, 降低了程序运行速度. 本文充分分析了 SQLite 事务机制和索引结构, 通过手动添加事务减少了重复打开、提交事务, 减少耗时的 I/O 读写次数, 并通过创建 B-tree 结构的索引降低缩短查询时间. 在 Android 手机上测试, 测试表明二者分别在大量插入数据和从大数据库中查询上, 时间的缩短是非常可观的. Android SQLite 数据库性能的优化, 扩大了应用范围和建立更加良好的用户体验.

参考文献

1 Lv JY, Xu SG, Li YJ. Application research of embedded database SQLite. 2009 International Forum on Information Technology and Applications. IEEE, Computer Society. 2009. 539–543.

2 尧有平,薛小波.基于 ARM-Linux 的 SQLite 嵌入式数据库的研究.微计算机信息,2008,24(2):64–66.

3 Wang Y, Liu ZJ, Xiang W, Liu XH. Research on concurrrency and transaction optimization of relational database. 计算机技术与应用发展, 2008:321–325.

4 Allen G, Owens M. The definitive guide to SQlite. American, Apress, 2010:193–144.

5 Database Speed Comparison. http://www.sqlite.org/speed.html.

6 禹亮.基于内容的图像索引和浏览算法研究[学位论文].长沙:湖南大学,2007.

(上接第 200 页)

测试结果表明, 此应用满足即时通信应用的基本功能及需求.

5 结语

本文研究了 MQTT 协议的结构、消息格式, 并且使用此协议在 Android 平台上实现了一款基于 MQTT 协议的即时通信应用, 并验证了应用的使用效果. MQTT 作为一款轻量级的消息发布/订阅协议, 其在手机上的发展前景十分可观, 在 Android 推送方面的应用还有很大的发展空间, 尤其是此协议低带宽的特点, 使其在带宽受限的地方倍受青睐. 本文中的即时通信应用并不完善, 尤其是安全性方面未做考虑, 在未来的工作中将会侧重这方面的研究.

参考文献

1 MQ Telemetry Transport. http://mqtt.org.

2 姜梦兰.基于消息中间件服务可靠性保证方案的研究与实现[硕士学位论文].成都:电子科技大学,2010.

3 Hunkeler U, Truong HL, Stanford-Clark A. MQTT-S – A publish/subscribe protocol for wireless sensor networks. Communication Systems Software and Middleware and Workshops. Bangalore. 2008. 791–798.

4 MQTT V3.1 Protocol Specification.http://wenku.baidu.com. 2012

5 Lee S, Kim H, Hong D, Ju H. Correlation analysis of MQTT loss and delay according to QoS level. Information Networking (ICOIN). Bangkok. 2013. 714–717.

6 安卓推送技术探讨.http://wenku.baidu.com.2012.

7 袁远.基于 Android 平台端到端即时通信系统的分析与设计[硕士学位论文].北京:北京邮电大学,2012.

8 王楠,宋飞,周华春.一种基于 Android 平台的即时通信方案. 计算机应用与软件,2013,(4):107–109,148.