HSQLDB的数据存储机制分析

文庆福
2011013239 thssvince@163.com
清华大学软件学院11班

2014年 3月 16日

目录

1 实验简介

1.1 实验背景

HSQLDB 提供了三种类型的持久化表， 分别是 MEMORY TABLE （内存表），CACHED TABLE （缓存表）和 TEXT TABLE（文本符表）。

MEMORY TABLE 是使用 CREATE TABLE 命令的默认表类型。 MEMORY TABLE 数据全部驻留 在内存中，但是对于表结构或内容的任何修改都被写入到¡dbname¿.script 文件中。 script 文 件在下次数据库打开的时候被 MEMORY 读取，里边的所有内容在 MEMORY TABLE 中重新创 建。所以跟 TEMP TABLE 不同， MEMORY TABLE 被默认为是持久的。

CACHED TABLE 是在使用 CREATE CACHED TABLE 命令的时候生成的。它只有索引或部分 数据是驻留在内存中的，所以可以允许生成大容量表而不用占用几百兆的内存。CACHED TABLE 的另外一个优点，即使它存储了大量的数据，数据库引擎只需花费很短的时间就可以 启动。它的不足是在速度上有所降低。如果你的数据集相对小的时候，尽量不要使用 CACHED TABLE。在小容量和大容量表共存的实际应用中，最好对小容量的表使用默认的 MEMORY TABLE。

TEXT TABLE 是在 1.7.0 版本中开始支持的，它使用 CSV（逗号分割数值）或其他分隔符 的文本文件作为数据源。你可以指定一个已有的 CSV 文件（比如其它数据库或程序导出的 数据）作为 TEXT TABLE 的数据源，你也可以指定一个空文件用数据库引擎来填充数据。 TEXT TABLE 的内存利用效率比较高，因为它只缓存部分文本数据和所有的索引。TEXT TABLE 的数 据源如果需要的话，可以重新分配到不同的文件。

本次实验就是要结合 HSQLDB 的数据文件结构，分析 HSQLDB 的数据存储机制，并了解 HSQLDB 运行时的缓存管理机制。

1.2 实验环境

- 操作系统：Windows 8 企业版
- JDK: OpenJDK 7（64-Bit）
- HSQLDB: 2.3.1
- IDE: Eclipse Standard(Kepler Service Release 1)

2 运行与分析

2.1 创建表格

```
CREATE TABLE ACCOUNT1(
    ACCOUNT_NUMBER CHARACTER(10),
    BRANCH_NAME CHARACTER(15),
    BALANCE NUMERIC(12,2) ,
    primary key(account_number));


CREATE CACHED TABLE ACCOUNT2(
    ACCOUNT_NUMBER CHARACTER(10),
    BRANCH_NAME CHARACTER(15),
    BALANCE NUMERIC(12,2) ,
    primary key(account_number));


CREATE TEXT TABLE ACCOUNT3(
    ACCOUNT_NUMBER CHARACTER(10),
    BRANCH_NAME CHARACTER(15),
    BALANCE NUMERIC(12,2) ,
    primary key(account_number));


SET TABLE PUBLIC.ACCOUNT3 SOURCE "account;fs=—";
```

2.2 调试跟踪打开数据库的过程，对比三种表打开过程的区别

2.2.1 载入 script 文件前的准备工作

1. 调试启动 server 进程，在 server 的 run 方法中，首先打开 server 的 socket，新建线程组，随后进入 OpenDatabases 过程。

2. 进入 openDatabases 后，首先调用 setDBInfoArrays 设置每个数据库的基本描述信息。其次，对 server 上的每一个 database，调用 DatabaseManager.getDatabase 方法，得到一个数据库实例。

3. DatabaseManager.getDatabase 中 调用了一个同名的 getDatabase 方法。在这个同名

getDatabase 方法中调用了getDatabaseObject， 通过 databaseMap 索引到数据库的路径 path 找到待打开的数据库，若要查找的数据库不在表中，则新建一个 Database 对象，并加入表中。返回得到的数据库对象。

4. 获得 Database 对象之后，根据当前数据库的不同状态而采取不同的处理方式，当前数据库处于 SHUTDOWN 状态，应执行 open 操作。open 中主要就是 reopen，在该方法中，设置数据库为 OPENING 状态，初始化一系列的 manager 实例，之后进入 logger.open 方法中。

5. logger.open 主要打开一些数据库对象独有的数据文件，调用 log.open 开启日志。log.open 之后，首先初始化文件参数，然后系统获取数据库是否被修改过的状态值，分为三种：MODIFIED，NEW，和 NOT_MODIFIED。这里进入了 NOT_MODIFIED，即数据库未被修改过。之后进入 processScript 处理 .script 文件。

2.2.2 载入 script，建立表格

在 processScript 中首先新建 ScriptReaderBase 对象，然后新建 Session 对象，调用 scr.readAll 读取 script 文件。readAll 中调用两个函数，readDDL 与 readExistingData。

readDDL 读取 .scipt 文件中的数据库命令，对于从 session 中读取到的 statement，如果不是 INSERT 语句， 则对其进行编译和运行。这里我们只关注之前执行的三条 CREATE TABLE 语句就好了。

readDDL 首先调用 readLoggedStatement 方法从源文件中读取一行，并得到该命令的一些类型信息，生成 statement 对象。读取之后执行 compileStatement 方法对其进行编译。

```
public Statement compileStatement(String sql) {
    parser.reset(sql);
    Statement cs =
        parser.compileStatement(ResultProperties.defaultPropsValue);
    cs.setCompileTimestamp(Long.MAX_VALUE);
    return cs;
}
```

其中 compileStatement 方法主要调用 compilePart 方法，得到 statement。

```
Statement compileStatement(int props) {
        Statement cs = compilePart(props);
        if (token.tokenType == Tokens.X_ENDPARSE) {
            if (cs.getSchemaName() == null) {
                cs.setSchemaHsqlName(session.getCurrentSchemaHsqlName());
```

```
            }
            return cs;
        }
        throw unexpectedToken();
    }
```

在 compilePart 方法判断 token 的类型，并对不同类型的 token 进行不同操作。我们关注的是 CREATE 语句，都会执行 Tokens.CREATE 下的操作，即执行 compileCreate 返回 statement 对象

在 compileCreate 中，首先设置 TableType 的默认值为 MEMORY_TABLE，然后调用 read 函数读取 Token, 并判断 Token 的类型。这里的 Token 类型总共有 Tokens.GLOBAL, Tokens.TEMP, Tokens.TEMPORARY, Tokens.MEMORY, Tokens.CACHED, Tokens.TEXT, Tokens.TABLE, Tokens.OR 这些类型。

这里我们考虑 Token.MEMORY，进入该分支后，首先读取一个 token，按照语法规则，该 token必须为 table，然后将 isTable 设置为 true，退出分支。Tokens.CACHED、Tokens.TEXT 的处理流程与之相似。随后，因为 isTable 为 True, 则执行compileCreateTable 函数。

```
if (isTable) {
            return compileCreateTable(tableType);
        }
```

在 compileCreateTable 中，对于 TEXT 类型的 Table 区别处理，新建 TextTable 对象，其余则新建 Table 对象。 在 Table 的构造后数中，根据不同 Table 类型，进行不同处理。若 CACHED_TABLE 不是基于文件的，则这里对它的操作与 MEMORY_TABLE 是一样的。对于 TEXT_TABLE，如果不是基于文件的，则会报错。跳出分支后，装载 triggers，对应设置 readOnly。

回退到 compileCreateTable，执行 compileCreateTableBody 方法，完成一些属性的定义工作。

然后回退到 readDDL 中， compileStatement 执行完之后，将会executeCompiledStatament。运行到 cs.execute，进入后发现，该 execute 中调用 getResult 函数，根据 type 进行分支选择，这里会进入 CREATE_TABLE 分支。

在三条 CREATE 语句执行完之后，还会执行 SET TABLE PUBLIC.ACCOUNT3 SOURCE ”account;fs=—” 语句，同样的，也是进入 compilePart，并选择 set 分支。进入 compileSet 再选择 Token.TABLE 分支继而选择 Token.SOURCE 分支，进入compileTextTableSource 。 在这里解析出 source 的相关参数 。 然后进入 executeCompiledStatement 运行 statement。 这里的 statement 实例为 StatementCommand。

进入后选择 StatementTypes.SET_TABLE_SOURCE 运行。然后到达 connect 方法，将 source 和 table 连接起来。在 connect 方法中，建立了 cache。

```
cache = (TextCache) database.logger.openTextFilePersistence(
            this, dataSource, withReadOnlyData, isReversed);
store.setCache(cache);
```

2.3 分析数据存储的细节，包括存储格式

2.3.1 compileStatement

在客户端输入 SHUTDOWN 并执行，服务器接收到客户端的请求，调用 receiveResult 方法，然后根据 resultMode 分支选择处理，从resultIn 进入，执行 execute 函数，一直到 executeDirectStatement。F5 进入后可以看到，首先 compileStatement，得到一个指令的列表 list。这里compilePart 得到词法分析器提供的 token，选择 Tokens.SHUTDOWN 分支运行。进入到 compileShutdown 后，首先检查当前 session 的用户是否具有管理员的权限，若权限不够则无法完成 shutdown。后面完成对 shutdown 语句的解析，并创建相应 的 StatementCommand 对象。之后就是执行 executeCompiledStatement。

2.3.2 executeCompiledStatement

在 executeCompiledStatement 中，进入 cs.execute 执行命令。在 execute 中，主要调用 getResult 方法得到结果，该方法中根据 type 类型来进行对应操作，对于 shutdown 命令，选择 DATABASE_SHUTDOWN 分支执行。

```
case StatementTypes.DATABASE_SHUTDOWN : {
    try {
        int mode = ((Integer) parameters[0]).intValue();
        session.database.close(mode);
        return Result.updateZeroResult;
    } catch (HsqlException e) {
        return Result.newErrorResult(e, sql);
    }
}
```

在这里，进行 database.close(mode) 操作。进入该函数会发现，服务器会首先关闭所有 session, 并清理所有 session。session 的 close 方法中，会调用 clearAllTables 清除所有 tables。

2.3.3 保存到 script 文件

在接下来的 logger.closePersistence(closemode) 会关闭 log 进程。由于是用 shutdown 方式结束关闭数据库，这会将关掉数据库文件，并将所有 cached tables 和 memory tables

中的数据写入 script 文件中。 writeAll 方法将所有操作都写入该文件，writeDDL 将非缓存的操作写入 script 中，然后执行 writeExistingData，将操作中插入的数据写入 script 中。

```
public void writeAll() {
    try {
        writeDDL();
        writeExistingData();
        finishStream();
    } catch (IOException e) {
        throw Error.error(ErrorCode.FILE_IO_ERROR);
    }
}
```

2.4 分析数据访问的细节，如何存取数据

2.4.1 MEMORY TABLE

MEMORY TABLE 是以 SQL 语句的形式存在于 script 文件中，在 script 文件中有这样一行：

INSERT INTO ACCOUNT1 VALUES('1234 ','GongShang ',12.00)

在服务器开启时，会载入 script 文件，执行这些语句，将数据插入到数据库中。所以在数据库运行过程中，MEMORY TABLE 的所有数据都在内存当中。而当服务器时，又会将 log 文件写入到 script 文件当中以保存 MEMORY TABLE 的数据。

2.4.2 CACHED TABLE

CACHED TABLE 则是存数在外存中的 .data 文件中。

0000 0000 0000 0000 0000 0000 0000 0000

0000 0020 0000 0000 0000 0000 6800 0000

0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 ...

2.4.3 TEXT TABLE

对 TEXT TABLE，它存储于 account 文件下，使用的数据文件是纯文本文件，每一行文本对应一个数据行，不同类型的数据均以字符形式存储，字段之间以分隔符分隔。

2.5 分析 HSQLDB 的缓存管理机制

MEMORY TABLE 不涉及到缓存管理机制，所有操作都在内存中直接完成。 故我们只讨论 CACHED TABLE 和 TEXT TABLE 这两种表的缓存管理。

2.5.1 CACHED TABLE

CATHED TABLE 在获取数据行对象时，需要判断其是否在内存中，如果是，则直接返回；否则，尝试调用缓存管理器从缓存中获取，如果缓存中没有数据，则需要从文件读取并装入缓存。

```
public CachedObject get(CachedObject object, PersistentStore store,
                        boolean keep) {
    readLock.lock();
    long pos;
    try {
        if (object.isInMemory()) {
            if (keep) {
                object.keepInMemory(true);
            }
            return object;
        }
        pos = object.getPos();
        if (pos < 0) {
            return null;
        }
        object = cache.get(pos);
        if (object != null) {
            if (keep) {
                object.keepInMemory(true);
            }
            return object;
        }
    } finally {
        readLock.unlock();
    }
    return getFromFile(pos, store, keep);
```

```
    }
```

添加数据时，首先在缓存中为该 object 分配空间，会使用到 freeBlockManager，如果没有可用的空余 block，则扩大缓存空间。分配完成后，调用 Cache 类中的 put 方法将这个 object 加入索引表。

```
void put(CachedObject row) {
        int storageSize = row.getStorageSize();
        if (size() >= capacity
                || storageSize + cacheBytesLength > bytesCapacity) {
            cleanUp(false);

            if (size() >= capacity) {
                clearUnchanged();
            }
            if (size() >= capacity) {
                cleanUp(true);
            }
        }

        if (accessCount > ACCESS_MAX && updateAccess) {
            updateAccessCounts();
            resetAccessCount();
            updateObjectAccessCounts();
        }

        super.addOrRemoveObject(row, row.getPos(), false);
        row.setInMemory(true);

        cacheBytesLength += storageSize;
    }
```

删除数据时，release 方法将行对象从缓存中删除。

```
 CachedObject release(long pos) {
        CachedObject r = (CachedObject) super.addOrRemoveObject(null, pos,
            true);
        if (r == null) {
            return null;
        }
        cacheBytesLength -= r.getStorageSize();
        r.setInMemory(false);
        return r;
    }
```

2.5.2 TEXT TABLE

Text Table 的数据缓存与 Cache Table 相似。区别在于，获取行时会直接去查询缓存，若缓存中没有，则从文件读取。在添加数据时，首先加入缓存，然后写入数据文件中。

```
protected void saveRowNoLock(CachedObject row) {
        try {
            rowOut.reset();
            row.write(rowOut);
            dataFile.seek(row.getPos() * dataFileScale);
            dataFile.write(rowOut.getOutputStream().getBuffer(), 0,
                           rowOut.getOutputStream().size());
        } catch (Throwable t) {
            logSevereEvent("DataFileCache.saveRowNoLock", t, row.getPos());

            throw Error.error(ErrorCode.DATA_FILE_ERROR, t);
        }
    }
```

删除数据时，首先将文件中对应的数据行清空（以空格填充），然后从缓存中删除。缓存本身的维护还包括清理、除碎片、备份和恢复等操作。

DataFileCache的 defrag 方法负责整理数据文件，重新组织，消除 cache 碎片。实际是建立一个新的数据文件，将表中所有的行重新写入。

3 问题总结

3.1 memory table，text table 和 cached table 在外存是如何存储的？

见 2.4 节

3.2 各数据文件分别用于存储什么信息？

- .script 文件存储用于启动数据库的基本信息，数据库关闭时，log 文件写入 script 文件，在 script 文件中保存了对数据库所作的大部分操作。
- .log 文件是日志文件，记录了 server 的对数据库的所有操作。
- .data 文件是 CACHED TABLE 表用到的数据文件。
- account 文件是 TEXT TABLE 的数据文件。

- .properties 文件是数据库的属性配置文件。

3.3 三种表分别是怎样打开的？如何读取数据？

见第 2.2 节

3.4 数据的操作是怎样实现的？

见第 2.4 节

3.5 缓存的替换机制是怎样的？缓存的容量是如何维护的？

见第 2.5 节

3.6 数据是怎样实现内外存交换的？在什么时候进行？

见第 2.5 节