HSQL 数据存储机制分析

曹智林 2014013462
zhilin.cao@foxmail.com

一、数据库准备

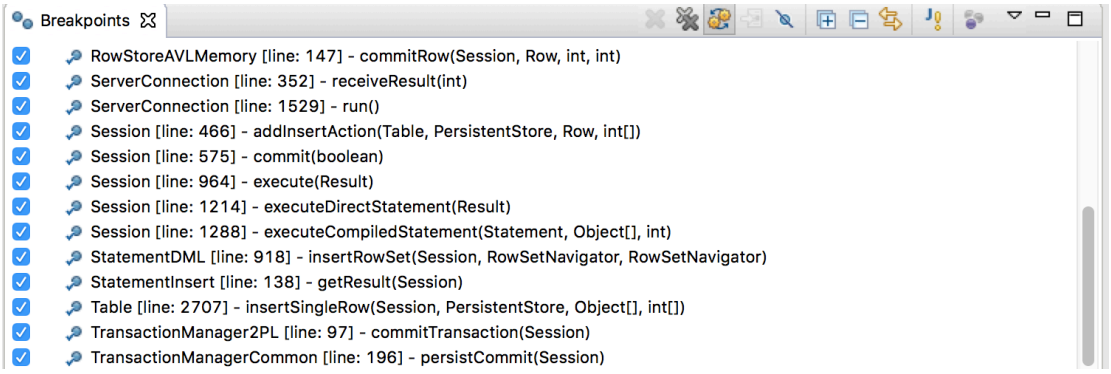
清空数据库，分别创建 Cached Table，Memory Table 和 Text Table，执行语句如下：

```
CREATE TABLE BRANCH(
    BRANCH_NAME CHARACTER(15),
    BRANCH_CITY CHARACTER(30),
    ASSETS NUMERIC(16,2),
    primary key(branch_name));
CREATE CACHED TABLE CUSTOMER(
    CUSTOMER_NAME CHARACTER(20),
    CUSTOMER_STREET CHARACTER(30),
    CUSTOMER_CITY CHARACTER(30),
    primary key(customer_name));
CREATE TEXT TABLE ACCOUNT(
    ACCOUNT_NUMBER CHARACTER(10),
    BRANCH_NAME CHARACTER(15),
    BALANCE NUMERIC(12,2),
    primary key(account_number));
SET TABLE PUBLIC.ACCOUNT SOURCE “account;fs=|”;
```

二、Memory Table 调试

（1）Insert

添加如下所示的各断点，并执行命令：insert into public.branch values ('bankname', 'city', 10);



在Debug过程中，可以看到Memory Table在添加记录时，调用StatementDMQL.execute 和 StatementInsert.getResult 来处理insert语句，通过insertSingleRow 的方法插入单个记录，先创建一个RowAVL对象，再存储它。

```
149         if (isSimpleInsert) {
150             Type[] colTypes = baseTable.getColumnTypes();
151             Object[] data = getInsertData(session, colTypes,
152                                           insertExpression.nodes[0].nodes);
153 
154             return insertSingleRow(session, store, data);
155         }
```

```
959    Result insertSingleRow(Session session, PersistentStore store,
960                           Object[] data) {
961
962        session.sessionData.startRowProcessing();
963        baseTable.setIdentityColumn(session, data);
964
965        if (baseTable.triggerLists[Trigger.INSERT_BEFORE_ROW].length > 0) {
966            baseTable.fireTriggers(session, Trigger.INSERT_BEFORE_ROW, null,
967                                   data, null);
968        }
969
970        baseTable.insertSingleRow(session, store, data, null);
971        performIntegrityChecks(session, baseTable, null, data, null);
972
```

最后有一个事务的提交操作，persistCommit操作需要将RowAVL对象提交到.log文件。所有insert语句写入log后，会增加一个commit记录写入文件。

```
118            adjustLobUsage(session);
119            persistCommit(session);
120
121            session.isTransaction = false;
122
123            endTransactionTPL(session);
124        } finally {
125            writeLock.unlock();
126        }
```

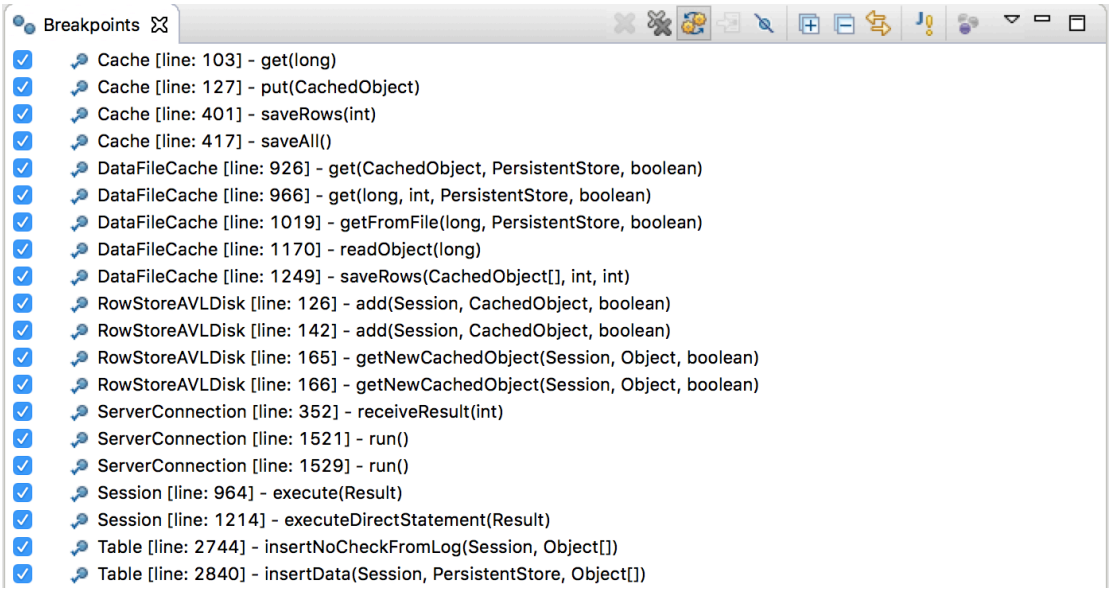
（2）Delete

删除语句的调用函数与 insert 语句类似，只是具体的 getResult 方法不同。

三、Cached Table 调试

（1）Insert

添加如下所示的各断点，并执行命令：insert into public.customer values ('Taylor Swift', 'Lincon Road', 'New York');



Cashed Table 的insert语句的处理在开始的receiveResult，以及最后的persistCommit都没有区别。但是Cashed Table 创建的是RowAVLDisk对象，然后再调用add.

```
160    public CachedObject getNewCachedObject(Session session, Object object,
161                                           boolean tx) {
162
163        Row row;
164
165        if (largeData) {
166            row = new RowAVLDiskLarge(table, (Object[]) object, this);
167        } else {
168            row = new RowAVLDisk(table, (Object[]) object, this);
169        }
170
171        add(session, row, tx);
172
173        return row;
174    }
```

```
124    public void add(Session session, CachedObject object, boolean tx) {
125
126        int size = object.getRealSize(rowOut);
127
128        size += indexList.length * NodeAVLDisk.SIZE_IN_BYTE;
129        size = rowOut.getStorageSize(size);
130
131        object.setStorageSize(size);
132
133        long pos = tableSpace.getFilePosition(size, false);
134
135        object.setPos(pos);
136
137        if (tx) {
138            RowAction.addInsertAction(session, table, (Row) object);
139            database.txManager.addTransactionInfo(object);
140        }
141
142        cache.add(object, false);
143
144        storageSize += size;
145    }
```

在cache. add中会调用put方法，对缓存进行调整。

```
902    public void add(CachedObject object, boolean keep) {
903
904        writeLock.lock();
905
906        try {
907            cacheModified = true;
908
909            cache.put(object);
910
911            if (keep) {
912                object.keepInMemory(true);
913            }
914
915            if (object.getStorageSize() > initIOBufferSize) {
916                rowOut.reset(object.getStorageSize());
917            }
918        } finally {
919            writeLock.unlock();
920        }
921    }
```

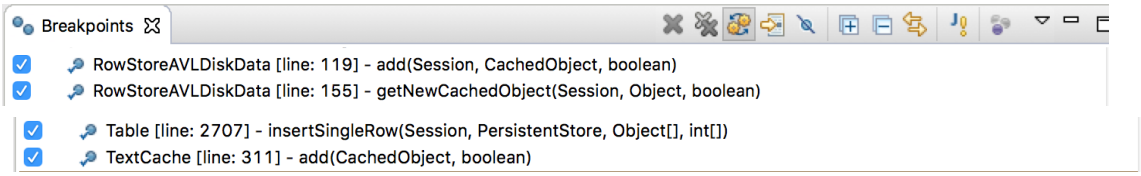
（2）Delete

与 Memory Table 相似，也涉及缓存调整。

四、Text Table 调试

（1）Insert

添加如下所示的各断点，并执行命令：insert into public.account values ('001', 'bank', 100);



Text Table 创建的是RowStoreAVLDisk对象，然后再调用add，其中最重要的是又调用了TextCache.add方法，再调用uncommittedCache.put。

```
150    public CachedObject getNewCachedObject(Session session, Object object,
151                                           boolean tx) {
152
153        Row row = new RowAVLDiskData(this, table, (Object[]) object);
154
155        add(session, row, tx);
156
157        return row;
158    }
```

```
119            cache.add(object, false);
120        } finally {
121            cache.writeLock.unlock();
122        }
123    }
```

```
309    public void add(CachedObject object, boolean keep) {
310
311        writeLock.lock();
312
313        try {
314            uncommittedCache.put(object.getPos(), object);
315        } finally {
316            writeLock.unlock();
317        }
318    }
319
```

最后也有 pesistCommit 的过程。

（2）Delete

与 Memory Table 相似，但是会改变相应文件存储内容。

五、问题回答

(1) memory table, text table和cached table在外存是如何存储的？

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。但是.script文件并不是实时更新的，每次数据库发生变化时，会将变化写入.log文件。在服务器正常关闭或启动时，会读取.log文件相应写入.script文件。再将.log文件清空。

Cached Table的外存存储与.log文件、.script文件和.data文件相关。.script文件存储标的结构信息。进行缓存交换时会删除.log中相应语句。如果有语句执行后，没有写入.data文件，在服务器正常关闭或瑕疵启动时，会读取.log文件对.data文件进行更新。

Text Table的外存存储中具体数据只与创建时指定的文件有关，表的一些结构信息则存储在.script文件中。

(2) 各数据文件分别用于存储什么信息？

.log文件用于记录数据库运行中的日志信息。数据库正常关闭时，此文件会被删除。否则在下次启动时用于恢复操作。.data文件用于存储Cached Table中的数据信息。.script文件以可执行语句的形式记录数据库的各种信息。记录Memory table中的数据。在数据库打开的过程中，.script文件中的语句会被读取并执行。.lck记录数据库的打开状态。.properties记录数据库的属性。.backup文件用来备份数据库。

(3) 三种表分别是怎样打开的？如何读取数据？

```
65    public void readAll(Session session) {
66        readDDL(session);
67        readExistingData(session);
68    }
```

首先是创建表结构，ScriptReaderBase.readDDL方法读入.script文件中的DDL的部分并执行，第二个ScriptReaderBase.readExistingData负责读入其他和数据有关的指令并执行。

```
2792    public void insertFromScript(Session session, PersistentStore store,
2793                                 Object[] data) {
2794
2795        systemUpdateIdentityValue(data);
2796
2797        if (session.database.getProperties().isVersion18()) {
2798            for (int i = 0; i < columnCount; i++) {
2799                if (data[i] != null) {
2800                    int length;
2801
```

Memory Table的数据存储在.script文件和.log文件中，初始化Table的各属性之后，调用Table.insertFromScript方法从.script文件中读取插入数据的命令并在内存中重建Memory Table。Memory Table 中的数据是存储在内存中的一颗平衡二叉树，所以就是在这颗树上访问数据。

Cached Table打开时从.调用DataFileCache.open方法打开.data文件建立缓存，.script文件中的所有语句用于初始化其索引。其数据也是存储在AVL树中的，先从缓存中读取数据，若未命中则进行缓存调整，从.data文件中取得所需数据。

Text Table在.script文件中指定其文件与分隔符。之后读取数据文件并建立缓存。数据存储在AVL数中，若缓存中查找不到，需到外存中查找。

(4) 数据的操作是怎样实现的？

Memory Table：先生成新的RowAVL对象，然后直接加入内存中的平衡二叉树，并建立索引，之后可再由索引找到具体对象进行删除等其他操作。

Cached Table：先根据对象的大小生成新的RowAVLDisk对象或者RowAVLDiskLarge对象，然后将它们加入到cache中，Cached Table维护了一张哈希表，命中时直接操作，不命中则需要访问外存，并进行内存调整。

Text Table：先生成新的RowAVLDiskData对象，然后加入缓存。

(5) 缓存的替换机制是怎样的？缓存的容量是如何维护的？

缓存中的各数据节点维护了accessCount变量，记录一段时间内该节点被访问的次数，当需要清除缓存时，访问次数最少的节点被移除。

```
158    boolean preparePut(int storageSize) {
159
160        boolean exceedsCount = size() + reserveCount >= capacity;
161        boolean exceedsSize = storageSize + cacheBytesLength > bytesCapacity;
162
163        if (exceedsCount || exceedsSize) {
164            cleanUp(false);
165
166            exceedsCount = size() + reserveCount >= capacity;
167            exceedsSize = storageSize + cacheBytesLength > bytesCapacity;
168
169            if (exceedsCount || exceedsSize) {
170                clearUnchanged();
171            } else {
172                return true;
173            }
174
175            exceedsCount = size() + reserveCount >= capacity;
176            exceedsSize = storageSize + cacheBytesLength > bytesCapacity;
177
```

在向缓存插入数据之前，会执行上面这个preparePut方法，检查缓存空间的剩余量。按情况进行clearUP(false),将最少访问的数据清除，若缓存仍然很满，就将缓存中没有被更改过的数据移出，若还是满，则执行cleanUp（true）,把所有能移出缓存的数据都移出，只留下建立索引所必须留在缓存的数据，如果还是满的话，就只能报错提示缓存已满了。

(6) 数据是怎样实现内外存交换的？在什么时候进行？

在数据库的初始化时，各文件会指导其初始化，此时会有内外存交换

在进行数据的增删查改时，如果能从缓存中直接找到，则直接访问。若没有就需从文件中读取所需数据。由 DataFileCache.get 方法可知，当所找数据不在缓存中时，就执行方法getFromFile,即会进行数据的内外存交换。与上面的问题相同,会涉及到拿出缓存中的部分数据，并写入文件，以及将读取的数据写入缓存。

```
1016    private CachedObject getFromFile(long pos, PersistentStore store,
1017                                     boolean keep) {
1018
1019        CachedObject object = null;
1020
1021        writeLock.lock();
1022
1023        try {
1024            object = cache.get(pos);
1025
1026            if (object != null) {
1027                if (keep) {
1028                    object.keepInMemory(true);
1029                }
1030
1031                return object;
1032            }
```

六、实验总结

在实验过程中手动寻找断点耗时很长，在参考往届学长的断点设置之后，思路更清晰。对于Text Table 具体的一些机制还有一些疑惑，比如索引与具体文件之间如何建立联系。如何做到删除记录等。