HSQLDB 的恢复机制分析

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

2014年 4月 20日

目录

1 实验简介

1.1 实验背景

恢复机制是数据库系统必不可少的组成部分, 它负责将数据库恢复到故障发生前的某个一致的状态。我们曾分析过 HSQLDB 中的文件存储结构, 这些文件不仅提供了数据的磁盘存储, 还提供了数据库恢复的支持。

本次实验，我们将分析 HSQLDB 的恢复机制，研究其日志记录系统及基于日志恢复的过程。

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 HSQLDB 的数据恢复流程

2.1 记录日志

HSQLDB 会将数据的插入、删除、提交、回滚等操作都记录在 log 文件中。对于 autocommit 的，每条插入、删除语句之后，都会额外记录一个 COMMIT 操作。而对于非 autocommit 的，只有在执行 commit 之后，才会将这些操作记录在 log 文件中。

2.2 恢复流程

当打开数据库时，如果数据库是已修改状态，HSQLDB 就会处理 log 文件，按照 log 文件中记录到操作顺序执行。

当执行完成后，会调用 checkpoint 方法将数据写回，同时整理 script 文件。

3 HSQLDB 的日志与恢复

HSQLDB 中读写数据库日志的类主要包括org.hsqldb.scriptio.ScriptWriterText、org.hsqldb.scriptio.ScriptReaderText、org.hsqldb.persist.Log 和 org.hsqldb.persist.Logger 。

其中，ScriptWriterText 类用于处理所有的 script 和 log，提供基本的写入操作，可以以脚本命令的形式保存当前数据库的快照。 而ScriptReaderText 类是用于读取所有 script 和 log，提供基本的读入操作。

Log 类主要就是用来管理数据库文件的，包括.properties、.script、.data、 .backup、.log file 和 .lobs 文件。它使用 ScriptWriterText 类完成日志和脚本的写入，使用 ScriptReaderText 类完成日志和脚本的读取。除此之外，还提供了恢复等方法。

Logger 类则是高层次封装的接口，HSQLDB 中的各种日志的行为都是通过Logger 类操作的。

3.1 ScriptWriterText 类

ScriptWriterText 类继承自 ScriptWriterTextBase 类，ScriptWriterTextBase 是用于数据写入操作的，就不做过多分析了。下面我们主要分析下 ScriptWriterText 类中的两个方法 writeRow 和 writeRowOutToFile。

- writeRow

```
1  public void writeRow(Session session, Row row,
2                       Table table) throws IOException {
3      schemaToLog = table.getName().schema;
4      writeSessionIdAndSchema(session);
5      rowOut.reset();
6      ((RowOutputTextLog) rowOut).setMode(RowOutputTextLog.MODE_INSERT);
7      rowOut.write(BYTES_INSERT_INTO);
8      rowOut.writeString(table.getName().statementName);
9      rowOut.write(BYTES_VALUES);
10     rowOut.writeData(row, table.getColumnTypes());
11     rowOut.write(BYTES_TERM);
12     rowOut.write(BYTES_LINE_SEP);
13     writeRowOutToFile();
14 }
```

在上面函数中首先调用 writeSessionIdAndSchema 将 session 和 schema 写入，格式化是 /*C session_id*/SET SCHEMA schema_name，接下来依次写入 INSERT INTO、表名、VALUES，最终会调用 writeRowOutToFile 方法写入 log 文件。

- writeRowOutToFile

```
1  void writeRowOutToFile() throws IOException {
2      synchronized (fileStreamOut) {
3          fileStreamOut.write(rowOut.getBuffer(), 0, rowOut.size());
4          byteCount += rowOut.size();
5          lineCount++;
6      }
7  }
```

首先，对流对象做互斥操作，显然文件 IO 是不能异步进行的。然后调用流写入方法 write 将数据写入，并更新写入的字节数和行数。

3.2 ScriptReaderText 类

ScriptReaderText 类主要用来读取 script 文件和 log 文件。下面也分析下它的主要方法：

- readLoggedStatement

```
1  public boolean readLoggedStatement (Session session) {
2      if (! sessionChanged ) {
3          String s;
4          try {
5              s = dataStreamIn .readLine ();
6          } catch ( EOFException e) {
7              return false ;
8          } catch ( IOException e) {
9              throw Error.error(e, ErrorCode .FILE_IO_ERROR , null );
10         }
11         lineCount ++;
12         statement = StringConverter . unicodeStringToString (s);
13         if ( statement == null ) {
14             return false ;
15         }
16     }
17     processStatement (session );
18     return true ;
19 }
```

这个方法在恢复时会用到，顾名思义，用于读取日志中的语句。第 2-16 行将读取日志中的一行，并在第 17 行交给 processStatement 方法处理。

- processStatement

```
1  void processStatement (Session session) {
```

```
        ......
        sessionChanged = false ;
        rowIn. setSource ( statement );
        statementType = rowIn. getStatementType ();
        if ( statementType == ANY_STATEMENT ) {
            ......
        } else if ( statementType == COMMIT_STATEMENT ) {
            ......
        } else if ( statementType == SET_SCHEMA_STATEMENT ) {
            ......
        }
        String name = rowIn. getTableName ();
        ......
        Type [] colTypes;
        if ( statementType == INSERT_STATEMENT ) {
            ......
        }
        try {
            rowData = rowIn.readData(colTypes );
        } catch ( IOException e) {
            throw Error.error(e, ErrorCode .FILE_IO_ERROR , null );
        }
    }
```

这与 SQL 语句的解析是相似的，首先判断语句类型，对于未知类型、COMMIT 及 SET SCHEMA 语句，直接返回。对于其他语句，填充相应的数据域，包括表名，列类型，数据等。

- readDLL

```
protected void readDDL(Session session) {
    for (; readLoggedStatement (session ); ) {
        Statement cs = null ;
        Result result = null ;
        if (rowIn. getStatementType () == INSERT_STATEMENT ) {
            isInsert = true ;
            break ;
        }
        try {
            cs = session. compileStatement ( statement );
            result = session. executeCompiledStatement (cs ,
            ValuePool . emptyObjectArray );
        } catch ( HsqlException e) {
            result = Result. newErrorResult (e);
        }
```

```
16          if (result.isError ()) {
17              ......
18          }
19      }
20  }
```

readDLL 方法用于读取并执行 script 文件中的 DDL 语句。

第 2-19 行将调用 readLoggedStatement 方法不断读取语句。

第 5 行判断是否是插入语句，如果是，则退出。（script 文件由 DDL 语和 INSERT 语句与 SET SCHEMA语句两部分构成。）

第 9-15 行将编译、执行读到的一条语句，这一过程已在之前的实验中进行过详细的分析。

- readExistingData

```
1   protected void readExistingData (Session session) {
2       try {
3           String tablename = null ;
4           ......
5           for (; isInsert || readLoggedStatement (session );
6                   isInsert = false ) {
7               if ( statementType == SET_SCHEMA_STATEMENT ) {
8                   session. setSchema ( currentSchema );
9                   continue ;
10              } else if ( statementType == INSERT_STATEMENT ) {
11                  if (! rowIn. getTableName (). equals( tablename )) {
12                  tablename = rowIn. getTableName ();
13                  String schema = session. getSchemaName ( currentSchema );
14                  currentTable =
15                      database. schemaManager . getUserTable (session ,
16                          tablename , schema );
17                  currentStore =
18                      database. persistentStoreCollection .getStore(
19                          currentTable );
20                  }
21                  currentTable . insertFromScript (session , currentStore ,
22                      rowData );
23              } else {
24                  throw Error.error( ErrorCode . ERROR_IN_SCRIPT_FILE ,
25                      statement );
26              }
27          }
28          ......
29      } catch ( Throwable t) {
```

```
30              ......
31          }
32      }
```

readExistingData 方法用于从 script 文件中读取并向数据库中插入数据。

这里的处理方法与 readDLL 方法类似：不断调用 readLoggedStatement 方法读取语句，如果是 SET SCHEMA 语句，则调用 session 对象的 setSchema方法设置上下文的 Schema；如果是插入语句，则在设置表名、Schema 后，获取表对象，然后调用表对象的 insertFromScript 方法插入数据。

3.3 Log 类

Log 类位于 org.hsqldb.persist 包中，因此该类不仅处理记录 log，也负责进行一些和数据存储有关的操作。

对于记录 log，比较简单，例如对于插入语句：

```
1  void writeInsertStatement (Session session , Row row , Table t) {
2      try {
3          dbLogWriter . writeInsertStatement (session , row , t);
4      } catch ( IOException e) {
5          throw Error.error( ErrorCode .FILE_IO_ERROR , logFileName );
6      }
7      if ( maxLogSize > 0 && dbLogWriter .size () > maxLogSize ) {
8          database.logger. setCheckpointRequired ();
9      }
10 }
```

这里会调用 dbLogWriter 对象（一个 ScriptWriterText 类实例）的 writeInsertStatement 方法。下面我们着重来分析和恢复有关的方法：

```
1. void open() {
2       initParams();
3       int state = properties.getDBModified();
4       switch (state) {
5           case HsqlDatabaseProperties.FILES_NEW :
6               break;
7           case HsqlDatabaseProperties.FILES_MODIFIED :
8               database.logger.logInfoEvent("open start - state modified");
9               deleteNewAndOldFiles();
10              deleteOldTempFiles();
11              if (properties.isVersion18()) {
12                  if (fa.isStreamElement(scriptFileName)) {
```

```
13                      processScript();
14                  } else {
15                      database.schemaManager.createPublicSchema();
16                  }
17                  HsqlName name = database.schemaManager.findSchemaHsqlName(
18                      SqlInvariants.PUBLIC_SCHEMA);
19                  if (name != null) {
20                      database.schemaManager.setDefaultSchemaHsqlName(name);
21                  }
22              } else {
23                  processScript();
24              }
25              processLog();
26              checkpoint();
27              break;
28          case HsqlDatabaseProperties.FILES_MODIFIED_NEW :
29              database.logger.logInfoEvent("open start - state new files");
30              renameNewDataFile();
31              renameNewScript();
32              deleteLog();
33              backupData();
34              properties.setDBModified(
35                  HsqlDatabaseProperties.FILES_NOT_MODIFIED);
36          // continue as non-modified files
37          // fall through
38          case HsqlDatabaseProperties.FILES_NOT_MODIFIED :
39              database.logger.logInfoEvent(
40                  "open start - state not modified");
41              /**
42               * if startup is after a SHUTDOWN SCRIPT and there are CACHED
43               * or TEXT tables, perform a checkpoint so that the .script
44               * file no longer contains CACHED or TEXT table rows.
45               */
46              processScript();
47              if (!filesReadOnly && isAnyCacheModified()) {
48                  properties.setDBModified(
49                      HsqlDatabaseProperties.FILES_MODIFIED);
50                  checkpoint();
51              }
52              break;
53      }
54      if (!filesReadOnly) {
55          openLog();
56      }
```

```
57  }
```

这个方法用于打开日志。第 2 行初始化日志的基本参数。第 3 行获取数据库的修改状态，并分别处理：

- FILES_NEW（新文件），直接跳转到第 54 行。
- FILES_MODIFIED（修改过），这表明数据库没有正常关闭，需要进行恢复。首先删除扩展名为.new 和.old 的遗留文件。然后删除临时文件。接下来判断数据库版本，对于当前版本，将执行 processScript 方法处理.script 文件。然后调用 processLog 方法处理 log 文件。最后执行 checkpoint 操作。
- FILES_MODIFIED_NEW（修改过但是找不到 script 文件）对于这种情况，只能删除 log 文件，备份已有数据，接下来按未修改的情形处理。
- FILES_NOT_MODIFIED（未修改）直接处理 script 文件，根据注释可知，还需要再额外执行 checkpoint 操作，以保证 script 文件中不包含 cached 或 text 表中的行。 调用 openLog 方法加载 log 文件，同时修改数据库状态为已修改。

2. checkpoint

```
1  void checkpoint() {
2      if (filesReadOnly) {
3          return;
4      }
5      boolean result = checkpointClose();
6      checkpointReopen();
7      if (result) {
8          database.lobManager.deleteUnusedLobs();
9      } else {
10         database.logger.logSevereEvent(
11             "checkpoint failed - see previous error", null);
12     }
13 }
```

checkpoint 方法用于将所有脏数据写回。可以看到，这里的执行分两个阶段，分别是 checkpointClose 和 checkpointReopen。下面将分别分析这两个方法。

3. checkpointClose

```
1  boolean checkpointClose () {
2      ......
3      deleteOldDataFiles ();
4      try {
5          writeScript ( false );
```

```
6       } catch ( HsqlException e) {
7           ......
8       }
9       try {
10          if (cache != null ) {
11              cache. commitChanges ();
12              cache. backupFile ();
13          }
14      } catch ( Exception ee) {
15          ......
16      }
17      ......
18      deleteLog ();
19      ......
20      try {
21          properties . setDBModified (
22          HsqlDatabaseProperties . FILES_NOT_MODIFIED );
23      } catch ( Exception e) {}
24          ......
25          return true ;
26  }
```

第 3 行，删除.old 文件。

第 5 行，调用 writeScript 方法，注意这里参数为 false，因此，对cached 和 text 表，只向 script 文件中写入 DDL 语句。

第 9-16 行调用 cache 的 commitChanges 方法，将数据写回磁盘，然后调用 backupFile 方法备份数据文件。

第 18 行删除日志文件。

第 21 行设置数据库属性为 FILES_NOT_MODIFIED。 经过这样的操作，脏数据被写入磁盘，同时 script 和 log 文件都成功更新。

4. checkpointReopen

```
1   boolean checkpointReopen () {
2       ......
3       try {
4           if (cache != null ) {
5               cache. openShadowFile ();
6           }
7           if ( dbLogWriter != null ) {
8               openLog ();
9           }
10          properties . setDBModified ( HsqlDatabaseProperties . FILES_MODIFIED );
11      } catch ( Exception e) {
```

```
12          return false ;
13      }
14      return true ;
15  }
```

这个方法很简单，第 5 行打开数据文件，第 8 行打开日志文件，第 10 行设置数据库状态为已修改。

5. processScript

```
1   private void processScript () {
2       ScriptReaderBase scr = null ;
3       try {
4           Crypto crypto = database.logger. getCrypto ();
5           if (crypto == null ) {
6               boolean compressed = database.logger. propScriptFormat == 3;
7               scr = new ScriptReaderText (database , scriptFileName ,
8                   compressed );
9           } else {
10              scr = new ScriptReaderDecode (database , scriptFileName , crypto ,
11                  false );
12          }
13          Session session =
14              database. sessionManager . getSysSessionForScript (database );
15          scr.readAll(session );
16          scr.close ();
17      } catch ( Throwable e) {
18          ......
19      }
20  }
```

第 4-12 行获取 ScriptReader，这中间要判断文件是否加密、是否压缩。

第 14 行获取系统 session。

第 15 行调用 readAll 方法读取 script 文件，并在系统 session 中执行。readAll 方法由两条语句组成，分别是 readDLL 和readExistingData。

6. processLog

```
1   private void processLog () {
2       if (fa. isStreamElement ( logFileName )) {
3           ScriptRunner . runScript (database , logFileName );
4       }
5   }
```

这里调用 ScriptRunner 类的静态方法 runScript 进行处理。

7. runScript

```
private static void runScript(Database database, ScriptReaderBase scr) {
        ......
        try {
            StopWatch sw = new StopWatch();
            while (scr.readLoggedStatement(current)) {
                ......
                Result result = null;
                statementType = scr.getStatementType();
                switch (statementType) {
                    case ScriptReaderBase.ANY_STATEMENT :
                        statement = scr.getLoggedStatement();
                        Statement cs;
                        try {
                            cs = current.compileStatement(statement);
                            if (database.getProperties().isVersion18()) {
                                // convert BIT columns in .log to BOOLEAN
                                if (cs.getType()
                                        == StatementTypes.CREATE_TABLE) {
                                    Table table =
                                        (Table) ((StatementSchema) cs)
                                            .getArguments()[0];
                                    for (int i = 0; i < table.getColumnCount();
                                            i++) {
                                        ColumnSchema column =
                                            table.getColumn(i);
                                        if (column.getDataType().isBitType()) {
                                            column.setType(Type.SQL_BOOLEAN);
                                        }
                                    }
                                }
                            }
                            result = current.executeCompiledStatement(cs,
                                    ValuePool.emptyObjectArray, 0);
                        } catch (Throwable e) {
                            result = Result.newErrorResult(e);
                        }
                        ......
                        break;
                    case ScriptReaderBase.COMMIT_STATEMENT :
                        current.commit(false);
                        break;
                    case ScriptReaderBase.INSERT_STATEMENT : {
                        current.sessionContext.currentStatement = dummy;
```

```
44                          current.beginAction(dummy);
45                          Object[] data = scr.getData();
46                          scr.getCurrentTable().insertNoCheckFromLog(current,
47                                  data);
48                          current.endAction(Result.updateOneResult);
49                          break;
50                      }
51                      case ScriptReaderBase.DELETE_STATEMENT : {
52                          current.sessionContext.currentStatement = dummy;
53                          current.beginAction(dummy);
54                          Table           table = scr.getCurrentTable();
55                          PersistentStore store = table.getRowStore(current);
56                          Object[]        data  = scr.getData();
57                          Row row = table.getDeleteRowFromLog(current, data);
58                          if (row != null) {
59                              current.addDeleteAction(table, store, row, null);
60                          }
61                          current.endAction(Result.updateOneResult);
62                          break;
63                      }
64                      case ScriptReaderBase.SET_SCHEMA_STATEMENT : {
65                          HsqlName name =
66                              database.schemaManager.findSchemaHsqlName(
67                                  scr.getCurrentSchema());
68                          current.setCurrentSchemaHsqlName(name);
69                          break;
70                      }
71                      case ScriptReaderBase.SESSION_ID : {
72                          break;
73                      }
74                  }
75                  if (current.isClosed()) {
76                      sessionMap.remove(currentId);
77                  }
78              }
79          } catch (HsqlException e) {
80              ......
81          } catch (OutOfMemoryError e) {
82              ......
83          } catch (Throwable e) {
84              ......
85          } finally {
86              ......
87          }
```

```
88          }
89      }
```

从上面代码可以看到，这里对每种语句进行分别处理：

- COMMIT：直接调用 session 的 commit 方法。
- INSERT：调用表对象的 insertNoCheckFromLog 方法，该方法与前文提到的 insertData 方法相比，多了一个操作：记录插入这个 Action
- DELETE：调用表对象的 deleteNoCheckFromLog 方法，该方法首先找到要删除的行，然后调用 addDeleteAction，这个方法在事务处理中已进行了分析。
- SET SCHEMA：调用 session 的 setCurrentSchemaHsqlName 方法直接设置 Schema。
- 其他语句：编译、运行即可。

4 问题总结

4.1 五个文件在恢复机制中分别有什么作用？

- .properties：记录的数据库的一些设置属性，在恢复中，最重要的是 modified 属性，如果该属性为 yes，则意味着需要恢复。
- .script：该文件记录了数据库表的定义，以及对于 memory table 的 insert 语句，在恢复前会和正常打开一样执行一次。
- .data：该文件保存了 cached table 的数据，在恢复起始，它的数据 是崩溃前的最后一次 checkpoint 操作后记录的数据。在恢复后，数 据会被更新到正确的状态。
- .backup：顾名思义，该文件是 .data 文件的备份。
- .log：该文件是恢复机制的核心，记录了数据库最近的改变，checkpoint 到崩溃之间的所有操作。

4.2 在插入或删除数据后，各数据文件有什么变化？

插入或删除数据后，只有.log 文件发生了变化，.log 文件会记录这些操作。

4.3 执行checkpoint后，各数据文件有什么变化？

执行 checkpoint 后，.log 文件会被删除；.script 文件会被更新，对 memory table，.script 文件会记录所有的数据；和.data 文件也会被更新，保存当前的数据；.properties 文件的 modified 属性会被置为 no。

4.4 cached table和memory table的数据分别是怎样恢复的？

对于恢复操作，这两个表并没有明显区别，唯一的区别是，checkpoint后，对于 cached table，数据会被写回.data 文件，而对于 memory table，数据会被写回.script 文件。

4.5 并发的事务如何恢复？

事务是否并发与恢复并无关系，因为隔离级别保证了记录到日志中的操作均是正确的。因此，只要按照日志文件中的顺序进行恢复，就不会有问 题。

4.6 未提交的事务是否恢复？

未提交的事务不会记录在日志文件中，因此不会恢复。