HSQLDB的恢复机制分析

软件62 周展平 2016013253

问题思考

1. 各文件在恢复机制中的作用

- .properties文件：存储tx_timestamp属性、modified属性以及version属性。在恢复机制中用到modified属性。
- .script文件：存储所有操作，当数据库重新打开时重新执行其中的操作来恢复数据库
- .data文件：存储持久化的数据
- .log文件：存储checkpoint之后的所有操作，执行checkpoint时会合并到到script中
- .script.new文件：用于暂存.script文件
- .backup.new文件：用于暂存.data文件

2. 数据库的更改状态

共有4种状态，在HsqlDatabaseProperties类中有8个静态常量来表示状态，分别对应状态机标识以及.properties文件中的值：

```
// db files modified
public static final int     FILES_NOT_MODIFIED = 0;
public static final int     FILES_MODIFIED     = 1;
public static final int     FILES_MODIFIED_NEW = 2;
public static final int     FILES_NEW          = 3;
private static final String MODIFIED_NO        = "no";
private static final String MODIFIED_YES       = "yes";
private static final String MODIFIED_YES_NEW   = "yes-new-files";
private static final String MODIFIED_NO_NEW    = "no-new-files";
```

3. 插入或删除数据后各文件的变化

假设操作执行成功，则

- .script文件：无变化
- .log文件：增加了对应的SQL语句；如果处于autocommit状态，则还会有一行commit语句
- .data文件：无变化
- .properties文件：如果处于autocommit状态，则会将modified属性改为FILES_MODIFIED

4. 执行checkpoint后各文件的变化

假设checkpoint的操作全部成功执行，则

- .script文件：内容包括执行前的所有内容以及.log文件中的内容
- .log文件：被清空
- .properties文件：modified属性改为FILES_MODIFIED
- .data文件：无变化

5. cached table和memory table数据的恢复

- memory table：恢复过程与实验指导中的步骤基本相同，在checkpoint函数中调用checkpointClose函数和checkpointReopen函数，分别负责恢复数据和启动新的checkpoint。在checkpoint中，恢复数据过程实质是将.log文件中的语句合并到.script文件中，当然中间会使用.new文件作为副本。下一次打开数据库时，执行.script所有的语句就可以在内存中恢复所有数据。
- cached table：与memory table不同，cached table的数据存储在.data文件中。因此，在重新打开数据库的时候，会执行.log文件中的语句得到恢复后的数据，然后将数据更新到.data文件中。在恢复过程中，不会将对数据的操作语句合并到.script文件中，因为cached table的数据不是通过执行.script文件得到的，而是从.data文件中读入的。

6. 并发事务与未提交事务的恢复

- 对于并发事务，在Logger类的checkpoint函数中通过synchronized关键字控制最多一个线程进行具体操作。具体的实现如下：

```
public void checkpoint(Session session, boolean defrag, boolean lobs) {

        if (!backupState.compareAndSet(stateNormal, stateCheckpoint)) {
            throw Error.error(ErrorCode.ACCESS_IS_DENIED);
        }

        database.lobManager.lock(); //加锁

        try {
            synchronized (this) {
                checkpointInternal(session, defrag); //实际执行checkpoint操作

                if (lobs) {
                    database.lobManager.deleteUnusedLobs();
                }
            }
        } finally {
            backupState.set(stateNormal);
            checkpointState.set(stateCheckpointNormal);
            database.lobManager.unlock(); //释放锁
        }
    }
```

  对于处理较大文件的database.lobManager，HSQLDB采用锁的机制控制并发。checkpoint的具体操作则被封装到了另外一个函数checkpointInternal中，然后调用Log类的checkpoint函数完成checkpoint操作。

- 通过实验发现，对于3种表，未提交事务都不会被恢复。

7. 思考题：将write、checkpoint在各个阶段可能出现的问题与恢复机制对应起来

对于checkpoint函数，我们主要分析在checkpointClose函数中可能出现的问题，源码如下：

```
/**
     * Performs checkpoint including pre and post operations. Returns to the
     * same state as before the checkpoint.
     */
boolean checkpointClose() {

    if (filesReadOnly) {
        return true;
    }
```

```
    database.logger.logInfoEvent("checkpointClose start");
    synchLog();
    database.lobManager.synch();
    database.logger.logInfoEvent("checkpointClose synched");
    deleteOldDataFiles();

    try {
        writeScript(false);
        database.logger.logInfoEvent("checkpointClose script done");

        if (cache != null) {
            cache.reset();
            cache.backupDataFile(true);
        }

        properties.setProperty(HsqlDatabaseProperties.hsqldb_script_format,
                               database.logger.propScriptFormat);
        properties.setDBModified(
            HsqlDatabaseProperties.FILES_MODIFIED_NEW);
    } catch (Throwable t) {

        // backup failed perhaps due to lack of disk space
        deleteNewScript();
        deleteNewBackup();
        database.logger.logSevereEvent("checkpoint failed - recovered", t);

        return false;
    }

    closeLog();
    deleteLog();
    renameNewScript();
    renameNewBackup();

    try {
        properties.setDBModified(
            HsqlDatabaseProperties.FILES_NOT_MODIFIED);
    } catch (Throwable e) {
        database.logger.logSevereEvent(
            "logger.checkpointClose properties file save failed", e);
    }

    database.logger.logInfoEvent("checkpointClose end");

    return true;
}
```

依次分析函数中各个主要操作失败是如何解决的：

（1）writeScript或backupDataFile失败：说明.script.new或.data的备份失败，之后会进入catch语句块，删除.new文件，直接退出。此时相当于没有进行数据恢复，但是.script、.log文件没有变化，下一次执行checkpoint时还可以进行数据恢复。

（2）closeLog或deleteLog失败：关闭、删除.log文件失败，不会对数据恢复造成影响。

（3）renameNewScript或renameNewBackup失败：.script.new文件和.backup.new文件没有被重命名回.script和.data文件。HSQLDB没有特别处理这个问题，可能会造成数据丢失。

（4）properties.setDBModified(HsqlDatabaseProperties.FILES_NOT_MODIFIED)失败：此时properties属性的值仍然为FILES_MODIFIED_NEW。下一次调用Log类的open方法时，会进入以下代码块：

```
case HsqlDatabaseProperties.FILES_MODIFIED_NEW :
    database.logger.logInfoEvent("open start - state new files");
    renameNewDataFile();
    renameNewScript();
    deleteLog();
    backupData();
    properties.setDBModified(HsqlDatabaseProperties.FILES_NOT_MODIFIED);
```

此代码块会将properties属性的值最终设置为FILES_NOT_MODIFIED。

8. 思考题：为什么在checkpoint的时候要新写一个script.new，而不是直接在script上添加?

原因是为了避免在将.log合并到.script的过程中出现异常，导致.script文件被破坏而无法恢复。采用HSQLDB的机制，如果cache.reset()和cache.backupDataFile(true)失败，则会删除.script.new文件，而.script文件的数据完好无损，下一次重新执行checkpoint的时候，还可以进行合并操作。

9. 思考题：单独写入一条log、更改数据库的MODIFIED状态这两个操作是否可以保证原子性?

- 写入Log时，Log类调用dbLogWriter属性(ScriptTextWriter类的实例)的writeInsertStatement、writeDeleteStatement、writeOtherStatement等方法，其中主要的功能是通过writeRow方法实现的。writeRow方法调用RowOutputTextLog类的各种write方法。在整个过程中，没有保证原子性的机制。
- 更改数据库的MODIFIED状态的操作是通过HSQLDatabaseProperties类的save方法完成的，其中包含了try和catch语句块。在try语句块中，首先调用HSQLProperties类的save方法将属性保存在.properties.new文件中，之后调用FileUtil类的renameElement方法将.properties.new文件重命名为.properties文件，如果原来存在同名文件则删除之。其中主要发挥作用的renameWithOverwrite方法如下：

```
/**
     * Rename the file with oldname to newname. If a file with newname already
     * exists, it is deleted before the renaming operation proceeds.
     *
     * If a file with oldname does not exist, no file will exist after the
     * operation.
     */
    private boolean renameWithOverwrite(String oldname, String newname) {

        File file = new File(oldname);

        delete(newname); //删除原来存在的.properties文件

        boolean renamed = file.renameTo(new File(newname)); //将.properties.new文
件重命名为.properties

        if (renamed) { //重命名成功
            return true;
        }

        System.gc(); //trick:java会出现没有IO占用但仍在内存中的bug，调用gc()方法可以解决
bug
        delete(newname); //重新删除原来存在的.properties文件

        if (exists(newname)) { //删除失败，将原来的文件.properties重命名
            new File(newname).renameTo(new File(newDiscardFileName(newname)));
        }

        return file.renameTo(new File(newname)); //现在可以进行重命名了
    }
```

注意到，在renameWithOverwrite方法中，首先删除了原来的文件，然后调用renameTo方法进行重命名。在Java中可能会出现一个文件没有被IO占用但仍在内存中的bug，此时delete操作会失败，解决方法是调用gc()方法、然后再一次调用renameTo方法。如果renameTo方法失败，则会判断是否是因为以上问题，如果是由于delete的bug导致的，并且bug仍然不能被解决，则将文件重命名为newDiscardFileName(newname)。否则再次调用renameTo方法。

事实上，如果两次renameTo方法都失败，则不能保证原子性。但是一般来说，可能失败的原因就是delete的bug导致的renameTo失败，因此一般情况下可以认为更改数据库的MODIFIED状态能够保证原子性。

10. 思考题：根据ScriptWriterBase.start判断writeDelay参数的作用。

writeDelay参数指定了执行任务的周期（单位ms）。start函数创建了一个HSQLDB实现的HSQLTimer，指定了timer开始的时间、周期、任务等属性。ScriptWriterBase的start函数被用来执行周期性的.log和.script文件的更新任务。