TinyDB设计文档

一、通信模块

实现目标：用户登录系统，对系统默认数据库进行一系列操作，退出系统。

1.1server

1. 使用TThreadPoolServer实现对多个client的响应:
2. 向IServiceHandle构造函数传入Manager对象,控制client对应的数据库系统.

```
server = new
TThreadPoolServer(newTThreadPoolServer.Args(transport).processor(processor));
```

1.2client

1. 用string.split(" "); 解析输入的命令, 不同指令调用不同实现函数.
2. 实现函数:
    1. getTime 获取当前时间

```
void getTime()
```

2. connect, disconnect, executeStatement 调用Iservicehandle的connect接口, 获取server的响应

```
Long connect(String username, String password)
void disconnect(Long sessionId)
void executeStatement(String msg, Long sessionId)
```

1.3IServiceHandle

添加userInfo和sessionId到manager, 管理用户信息和当前的连接情况.

1. connect：

输入username和password，若正确则返回success和sessionId，manager的sessionId更新。错误则返回fail。

2. disconnect:

输入sessionId，若不是当前的sessionId则返回fail，若是当前sessionId则更新manager的sessionId，返回success。

3. executeStatement:

利用antlr4解析输入的statement，并将已经封装好的string类型结果返回给client。

二、异常处理模块

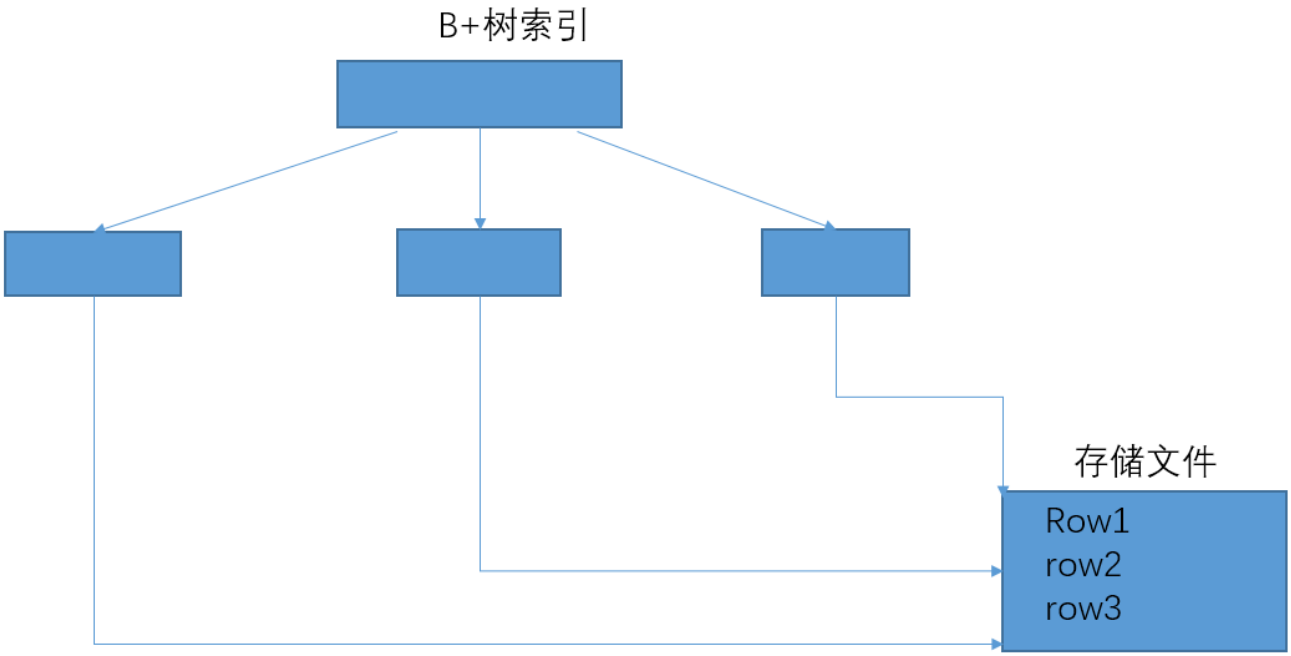
1. DuplicateKeyException：多用于storage和schema模块

2. KeyNoteExistException：多用于storage和schema模块
3. SyntaxErrorException：用于query语句parse时报错处理

三、存储模块

3.1基本架构：

用B+树做记录的索引，key为entry，叶子节点的value记录对应的row在数据文件中的位置指针。



3.2序列化与反序列化

1. 由于Row与Entry的结构默认已经继承Serializable，因此B+树索引利用JAVA提供的序列化存储到“dbname_tablename_attributename.idx"文件中。
2. 数据文件利用**带空闲列表的定长记录的存储形式**存储到“dbname_tablename.data"文件。文件头会维护空闲列表的最后一个被删除的记录的位置pointer与自增主键值。

3.3对记录的操作

1. 插入
    1. NOT_NULL约束的检测
    2. 对主键的检测
    3. 查看有无空闲列表区域
        1. 无：寻址到文件末尾
        2. 有：寻址到空闲区域，并更新空闲列表指针
    4. 将row转换为Bytes
    5. 在当前文件位置写入Bytes
    6. 更新索引树
    7. 更新文件头
2. 删除

    1. 从索引树获取到待删除记录在文件中的指针
    2. 删除索引树中的索引
    3. 定位到文件中的位置，并覆盖为空闲列表指针
    4. 更新文件头
3. 修改
    1. 利用上述method，先删除，后插入
4. 查询
    1. 读取索引树，定位到文件指定位置
    2. 在当前文件位置读取Bytes
    3. 将Bytes软换为row

3.4二进制操作

1. 支持基本数据类型与Bytes的相互转换
    1. INT, LONG, FLOAT, DOUBLE, STRING
2. 为定长记录存储计算row的单位Bytes

3.5测试文件

1. 测试索引与记录的序列化
2. 测试删除记录
3. 测试修改记录
4. 测试查询记录
5. 测试反序列化

- 测试结果 - schema : (id: int, name: string, primarykey(id))

```
测试初始化:
0, XiaoLi
1, XiaoLi
2, XiaoLi
3, XiaoLi
4, XiaoLi
测试删除id=2:
0, XiaoLi
1, XiaoLi
3, XiaoLi
4, XiaoLi
测试修改数据，更新id=3的名字为XiaoMing:
0, XiaoLi
1, XiaoLi
3, XiaoMing
4, XiaoLi
读取id=3的数据:
3, XiaoMing
测试从文件重新载入table，结果需要与之前一致:
0, XiaoLi
1, XiaoLi
3, XiaoMing
4, XiaoLi
```

四、元数据管理模块

4.1基本思路

在database类中维护一个.meta文件来记录当前该数据库所包含的表的数量、各表的名称，包含的列的名称、类型、是否为主键等信息。

在Manager类中用schema文件记录当前保存的database信息，包括database的数量，各个database的名称。

4.2所需文件

文件名：schema

文件内容：数据库总数（int）

```
数据库名称长度（int）

数据库名称（bytes）
```

文件名：databaseName.meta

文件内容：

表数量（int）

表元数据长度（int）

表元数据（bytes）(结构如下)

```
  表名称长度（int）

  表名称（bytes）

  列数（int）

类型（byte）

列名称长度（int）

列名称（bytes）

是否主键（int）

是否非空（boolean）

最大长度（int）
```

4.3基本功能

1. 实现表的创建、删除、修改；
    - 创建
        - 从.meta文件读取数据库所包含的表的信息，调用Table()创建表

        - 将表添加入所属的数据库
    - 删除
        - 删除此表的索引、.data文件
        - 从database移除此table
    - 修改
        - "增删改attribute，这样就要同时修改table中每个row的值".
        - 按照当前表中的数据和新加入或删除的列的信息重新写表的.data文件（包括更新其freelistPtr，每个row对应新添加的列的数值为null）
        - 删除原来的.data文件并更新dataFile
2. 实现数据库的创建、删除、切换；
    - 创建
        - 检验当前没有此database
        - 创建database
    - 删除
        - 清除数据库信息
        - 从Manager移除数据库
        - 若移除的数据库为当前数据库，将当前数据库默认切换成数据库列表中第一个
        - 若所有数据库都被删除，创建默认数据库TEST
    - 切换
        - 在manager中记录当前数据库名称（string）
        - 切换时先退出当前database，并保存其信息（.meta |.data| .idx）
        - 将currentDatabase设定为要切换的数据库名
3. 实现表和数据库的元数据（有哪些数据库，数据库里有哪些表，每个表的结构如何）的持久化。
    - 参见所需文件
4. 重启数据库时从持久化的元数据中恢复系统信息。
    - 若文件正常，按照schema读取数据库数量以及各数据库信息（名称、table数量、table信息等等）
    - 若文件不正常打开或不存在，创建默认数据库TEST

4.4测试文件

1. 创建数据库
2. 删除数据库
3. 切换数据库
4. 创建表
5. 删除表
6. 表的修改（添加/删除）
7. 从文件恢复数据库

五、查询模块

5.1辅助类

1. **最终结果存储类 Result:**

private List rows; // 按列存储结果

private List columns; // 存储对应的column

表单返回：通过setColumn设置columns，通过addRow添加一列结果。

字符串返回：通过setMessage静态方法设置。

通过toString方法将结果转为字符串输出。

2. **解析过程中语法错误处理 SQLErrorListener：**

继承BaseErrorListener，重载syntaxError函数，返回报错的具体位置和相关信息。

3. **条件判断**

MetaInfo 存储基本的table信息

ComparerData 用于比较的某一方信息 table_column和literal两类

MultipleCondition 存储比较双方的comparerData, 以及comparator

4. **返回类型存储**

ResultColumn 存储select statement要求返回的column信息

5.2语句实现类

1.schema层面操作: SchemaStatement

数据结构:

type: 标识以下7种类型

databaseName: 需要操作的数据库名称

tableName: 需要操作的表单名称

1. create database

调用方法: Manager.createDatabaseIfNotExists

2. drop database

调用方法:Manager.deleteDatabase

3. use database切换数据库

调用方法: Manager.switchDatabase

4. drop table

调用方法: manager.getCurrentDB().drop

5. show database 显示当前所有数据库 // 【数据库名称 | 包含的表单数】

调用方法: manager.getDatabases

6. show table 显示当前数据库的所有table // 【表单名称 | 包含属性数】

调用方法: manager.getCurrentDB().getTables

7. show meta 显示table的具体信息

//【属性名 | 类型 | 主键 | 是否为空 | 最大长度】

2.新建表单：CreateTableStatement

数据结构:

tableName 表单名称

columnDefs 属性list 【primary和not null约束已经放入】

调用方法: manager.getCurrentDB().create

3.插入row到表单: InsertTableStatement

数据结构：

tableName 表单名称

columnsName 指令中的属性名称集合

rowValue 对应属性集合的取值

调用方法：manager.getCurrentDB().select(tableName).insert(row)

4.删除表单中的row: DeleteTableStatement

数据结构:

tableName 表单名称

multipleCondition 条件约束

调用方法：manager.getCurrentDB().select(tableName).delete(row)

5.更新表单中的row: UpdateTableStatement

数据结构：

tableName 表单名称

columnName 被修改的column

value 所赋值

multipleCondition 条件约束

调用方法：manager.getCurrentDB().select(tableName).update(row)

6.select from where查询表单: SelectTableStatement

数据结构：

tableName 表单名称

whereCondition where的条件约束

resultColumns 期待的返回内容

isDistinct 是否去重

7.select from join where查询表单: SelectJoinTableStatement

数据结构：

tableName 表单名称

onCondition on条件约束

whereCondition where条件约束

resultColumns 期待的返回内容

isDistinct 是否去重

5.3SQL语法解析

提取和设置statement中需要的参数【语句实现类中数据结构对应内容】

1. errorListener: 对不正确command报错提醒， parser时使用自定义的error Listener
2. Listener: 解析command，获取各种statement需要的参数

5.4支持的语句

```
    create_db_stmt     创建数据库
    drop_db_stmt       删除数据库
    create_table_stmt  创建表单
    drop_table_stmt    删除表单
    insert_stmt        插入row
    delete_stmt        删除row
    select_stmt        选择row
    update_stmt        更新row
    show_db_stmt       展现当前所有数据库
    show_table_stmt    展现数据库db的所有表单
    show_meta_stmt     展现表单table的所有column
    use_db_stmt        切换数据库
    begin transaction  开始事务
    commit             提交事务
    checkpoint         持久化
```

在满足基本要求基础上加入distinct，补全join时未写出的table name功能，在数据库层面加上简单查询语句。

六、事务与恢复模块

6.1事务

1. read committed

> 在Read Committed隔离级别下，允许不可重复读（Non Repeatable Read）的问题。
> 不可重复读是指，在**同一个事务**内，多次读同一数据，在这个事务还没有结束时，如果另一个事务恰好修改了这个数据（已经提交），那么，在第一个事务中，两次读取的数据就可能不一致。
> 但不允许“脏读取”，即事务A中未提交的数据，不可被事务B读取。

- 因此在每个Table类的ReentrantReadWriteLock成员变量在自己的**查询方法**内进行对**共享锁**的上锁和释放锁。
- 而**排它锁**的管理由语句解析之后执行插入、删除、更新时进行上锁和释放锁。

2.运行步骤

1. 初始化LogManager的isTransaction静态变量为false
2. 当事务开始时，记录isTransaction为true指明事务已开始

3. 对有写操作的插入、删除、更新语句会对对应表加上排它锁，且把此排它锁加至LogManager的排它锁list静态变量
4. 当接收到commit操作时，释放所有排它锁list中的排它锁，并记录isTransaction为false指明事务已结束

3.测试结果

```
init table:
ID | INT | true | true | 0
NAME | STRING | false | true | 16
start transaction...
insert: require write lock
search: require read lock
commit: release write locks
search: release read lock
result table:
ID | INT | true | true | 0
NAME | STRING | false | true | 16
| 1 | XiaoLi |
```

如图所示，初始表为空，并开启两个线程

一个线程启动事务，并且在事务内插入一条id为1的语句后提交

另一个线程在上一个线程启动后，马上查询一条id为1的记录，只有在排它锁释放后才能获取到共享锁

6.2恢复

1.基本思路

除schema文件外，其余所有文件（.data | .meta）只在

- 切换数据库（use database)
- checkpoint
- 正常退出系统

时更新

2.具体流程

正常执行时：

单一指令：获取指令-->解析正确-->写入database.log文件-->内存中修改

事务：begin transaction-->在内存new一个String数组，记录事务中的指令-->commit-->将数组中log record写入.log文件

-->checkpoint或者切换数据库或正常退出系统时将更新持久化到文件并清除log

恢复时：

- 按照schema中的数据库名称恢复数据库（遍历文件夹，将不在schema中的文件夹删除）
- 在各个数据库中，先按照.meta文件和.data恢复表
- 读database.log文件，重新执行log中的指令，如果是事务只有 begin transaction没有commit 则不执行，删除相应的指令。

七、进阶功能

- select查询优化

使用助教提供的测试文件，以下为query的总时间：

用时（ms）	第一次	第二次	第三次	平均
优化前	223	136	271	210
优化后	109	62	100	90.33

- select时支持distinct语句
- checkpoint持久化

将.log中内容持久化到硬盘