

final变量只能被初始化一次，它代表的是绑定关系的永久性，而不是被绑定的对象的值的永久性。声明时可以不赋值，但是必须在构造函数中赋值，static final要在static 方法中赋值

接口的方法默认为public abstract, final变量的值在编译期不一定能知道，比较好的习惯是将其名字大写。

== 测试是否引用同一内存对象

Equals则是根据实现，本类没有实现，就是距离最近的父类中的实现，Object类中是测试==。API要求如果equals返回true，那么hashCode()返回的值也必须相等。反过来不一定。

Synchronized保证对于方法的两次调用不会交叉，当其调用结束时，会自动建立和后续对其调用之间的happens-before关系，保证对于对象的修改对后续调用是可见的。

Server handleConnection 通过判断成员变量serverProtocol来确定使用哪种模式，如hsql或者http，并创建不同种类的connection，比如ServerConnection，创建时会保存一个递增且唯一的线程id，然后把当前连接加入到server中

Server.handleConnection->新建某种Connection对象->ServerConnection run-> init ->handshake 里面会给firstInt赋值，前8位是连接类型，整个firstInt表示版本号->使用hsql\_stream\_protocol时会进入receiveResult->newResult先根据不同的请求类型或者各种类型需要的一些字段信息，前4个字节是请求长度，把整个长度内容copy到in中，然后可以用封装好的RowInputBinary读取字符串，整数等信息 Result类的mode变量标识了结果的种类 如果是connect会在receiveResult->setDatabase->DatabaseManager.newSession->Database.connect->SessionManager.newSession 会创建新的session SessionManager会记录session的数目，保存当前的session 最后返回receiveResult并调用write 之后开始run里的事件处理循环，会反复调用receiveResult方法，除了CONNECT,SQLCANCEL,DISCONNECT,RESETSESSION,EXECUTE\_INVALID几种情况外，默认情况下是进入session对象的execute方法->可能会进入executeDirectStatement->compileStatements 解析sql语法->executeCompiledStatement 执行cs.execute(this)

server/serverconstants里定义了一系列常量，比如各个模式使用的默认端口等等。

# Row

所有元组基于CachedObject这一接口->Row->RowAVL->RowAVLDisk/RowAVLDiskData

RowAVLDisk->RowAVLDiskLarge

RowAVLDisk->RowAVLDiskDataChange

Row long position (file position???), Object[] rowData, RowAction rowAction, TableBase table

isInvariable -> false isBock -> false isMemory -> true

RowAVL NodeAVL nPrimaryNode 对应memory table的元组？

每一个RowAVL对应一系列NodeAVL 从nPrimaryNode开始，通过nNext连接 每一个对应一个Index???

RowAVLDisk static final int NO\_POS = -1, int storageSize, keepCount, accessCount, boolean isInMemory, isNew,hasDataChanges, hasNodesChanged表示cached, text table的元组？

# Node

CachedObject->NodeAVL->NodeAVLDisk

## Node的创建

executeCompiledStatement->StatementDMQL.execute->StatementInsert.getResult->StatementDML.insertSingleRow->Table.insertSingeRow->

memory table RowStoreAVLMemory->getNewCachedObject

cached table

RowStoreAVLDisk.getNewCachedObject->new RowAVLDisk->RowAVLDisk.setNewNodes-> new NodeAVLDisk

text table RowStoreAVLDiskData->getNewCachedObject

NodeAVL 对应index? NodeAVL nLeft nRight nParent nNext int isBalance int NO\_POS Row row

nNext被Row用来定位属于某一特定索引的node

NodeAVLDisk <-> RowAVLDisk ???

NodeAVLDisk int iLeft, iRight, iParent, iId, SIZE\_IN\_BYTE=4\*4 是cached table的节点？

PersistentStore是用于存储CachedOjbect的接口

NodeAVLDisk对NodeAVL的一系列方法进行了重写

# Index

SchemaObject->Index->IndexAVL->IndexAVLMemory

## Index的创建：

Session.exectueCompliedStatement->StatementSchema.execute->getResult->ParserTable.addTableConstriantDefinitions->Table.createPrimaryKey->TableBase.createPrimaryIndex->TableBase.getNewPrimaryIndex->Logger.newIndex->

Memory table->IndexAVLMemory

Cached table/Text table->IndexAVL

IndexAVL是AVL树，其中使用了int index的方式保存了表的列信息？？

# Table

Cloneable->TableBase->Table->TableDerived/ TextTable

SchemaObject->Table

cs变量是statement

Statementschema是对DDL的实现，比如create table在execute(Session)中

Mode 0 UPDATECOUNT, 34 EXECUTEDIRECT

Row里会有NodeAVL nPrimaryNode，而NodeAVL又有row

# 插入的过程

## Memory Table

Table.insertSingleRow->RowStoreAVLMemory.getNewCachedObject->new RowAVL->RowAVL.setNewNodes->new NodeAVL->Table.insertSingleRow-> Session.addInsertAction->TransactionManager2PL.addInsertAction->RowStoreAVL.insertRow->RowStoreAVL.indexRow->IndexAVLMemory.insert

## Cached Table

Table.insertSingleRow->RowStoreAVLDisk.getNewCachedObject->new RowAVLDisk->RowAVLDisk.setNewNodes-> new NodeAVLDisk-> RowStoreAVLDisk.getNewCachedObject->RowStoreAVLDisk.add->DataFileCache.add->Cache.put->cache.putNoCheck->BaseHashMap.addOrReomveOjbect

还要在

Session.addInsertAction->TransactionManager2PL.addInsertAction->RowStoreAVLDisk.insertRow->RowStoreAVL.indexRow->IndexAVL.insert

Session.executeCompiledStatement->StatementDMQL.execute->StatementInsert.getResult->Table.getRowStore->PersistenStoreCollectionDatabase.getStore 这里如果之前表没有store，那么会将返回的store和table关联->Logger.newStore这里会new RowStoreAVLDisk->Logger.getCache->Log.getCached->new DataFileCache

随后cache.open->DataFileCache.open->RAFile.newScaledRAFile->new RAFileHybrid->new java.io.File(name)返回到StatementInsert.getResult->StatementDML.insertSingleRow->接上面插入过程

DataFileCache.open还会写入一些文件基本的元信息

Database.close->Logger.close->Log.close->DataFileCache.close->DataFileCache.reset->Cache.saveAll 利用Cache.objectIterator来获得未保存的对象->Cache.saveRows->DataFileCache.saveRows->DataFileCache.saveRowNoLock->RowAVLDisk.write->RowAVLDisk.writeNodes 这些应该都是写到HsqlByteArrayOutputStream的buffer,最后用dataFile.write把buffer的数据写到文件

DataFileCache.reset还会在数据后面写入一些信息

## Text Table

Table(TextTable).insertSingleRow->RowStoreAVLMemory.getNewCachedObject->new RowAVLDiskData->RowAVLDiskData.setNewNodes->new NodeAVL-> RowStoreAVLMemory.getNewCachedObject->RowStoreAVLDiskData.add->DataFileCache(TextCache).add->LongKeyHashMap.put

还要在

Table.insertSingleRow-> Session.addInsertAction->TransactionManager2PL.addInsertAction->RowStoreAVLDiskData.indexRow->IndexAVL.insert

TOKENS中有一系列TOKEN的TYPE常量定义

DataFileCache里有文件名

create cached table c1(v1 char(10), v2 char(10));