# 存储管理

2016300030061伍业军

## 1.文件结构

### 1.1磁盘数据存放

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 块0 | 块1 | 块2 | 块3 | …… |

8KB

|  |
| --- |
| Freespacestart |
| Freespaceend |
| Link1 |
| Link2 |
| …… |
| Freespace |
| …… |
| Tuple2 |
| Tuple1 |

4B

4B

4B

4B

4B+AttrNum\*8B

4B+AttrNum\*8B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AttrNum | Attr 1 | Attr 2 | …… | Attr n |

### 1.2缓冲区

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 缓冲区0 | 缓冲区1 | 缓冲区2 | …… | 缓冲区999 |

8KB

总大小8KB\*1000=8000KB

## 2.定义

//缓冲区指针

class buffPointer {

int blockNum; //块号

Boolean flag; //标记改块是否为脏（true为脏）

int buf\_id; //缓冲区索引号

}

final private int attrstringlen=8; //属性最大字符串长度为8Byte

final private int bufflength=1000;//缓冲区大小为1000个块

final private int blocklength=8\*1024;//块大小为8KB

private List<buffPointer> BuffPointerList = new ArrayList<>(); //构建缓冲区指针表

private ByteBuffer MemBuff=ByteBuffer.allocateDirect(blocklength\*bufflength);//分配blocklength\*bufflength大小的缓冲区

private boolean[] buffuse=new boolean[bufflength];//缓冲区可用状态表，true为可用

private int blockmaxnum=-1;//最大的块号

private int[] blockspace=new int[10];//块空闲空间信息

## 3.模块

### 3.1数据编码模块

//编码字符串为byte

private byte[] str2Bytes(String s)

//解码byte为字符串

private String byte2str(byte[] b,int off,int len)

//编码int为byte

private byte[] int2Bytes(int value, int len)

//解码byte为int

private int bytes2Int(byte[] b, int start, int len)

### 3.2磁盘文件操作模块

//将缓冲区指针所指的块存入磁盘

private boolean save(buffPointer blockpointer)

//根据块号加载块到缓冲区

private buffPointer load(int block)

//将每一个块的空闲空间大小存入磁盘

private boolean saveBlockSpace()

//从磁盘将没一个块的空闲空间大小加载到内存

private void loadBlockSpace()

### 3.3缓冲区管理

//新建块

private buffPointer creatBlock()

//从缓冲区中根据块号寻找块，找不到返回null，找到返回该缓冲区块的指针

private buffPointer findBlock(int x)

//初始化缓冲区使用位图，程序运行时初始化全为true表示可用

private void initbuffues()

//更新缓冲区指针序列：将p置为缓冲区列表首位（为实现缓冲区块LRU置换算法）

private void updateBufferPointerSequence(buffPointer p)

//将执行层传入的元组写入缓冲区，并返回块号和偏移

public int[] writeTuple(Tuple t)

//根据执行层传入的块号和偏移，读取相应元组并返回

public Tuple readTuple(int blocknum,int offset)

//根据执行层传入的旧元组块号、偏移以及新的元组，对元组进行更新修改

public void UpateTuple(Tuple tuple,int blockid,int offset)

//将缓冲区刷新到磁盘

public boolean flush()

## 4.其他数据存储接口

### 4.1执行层系统表存储接口

#### 4.1.1 ObjectTable

public boolean saveObjectTable(ObjectTable tab) //将执行层传入的ObjectTable存入磁盘

public ObjectTable loadObjectTable() //从磁盘加载ObjectTable传给执行层

#### 4.1.2 ClassTable

public boolean saveClassTable(ClassTable tab) //将执行层传入的ClassTable存入磁盘

public ClassTable loadClassTable() //从磁盘加载ClassTable传给执行层

#### 4.1.3 DeputyTable

public boolean saveDeputyTable(DeputyTable tab) //将执行层存入的DeputyTable存入磁盘

public DeputyTable loadDeputyTable() //从磁盘加载DeputyTable传给执行层

#### 4.1.4 BiPointerTable

public boolean saveBiPointerTable(BiPointerTable tab) //将执行层传入的BiPointerTable存入磁盘

public BiPointerTable loadBiPointerTable() //从磁盘加载BiPointerTable传给执行层

#### 4.1.5 SwitchingTable

public boolean saveSwitchingTable(SwitchingTable tab) //将执行层传入的SwitchingTable存入磁盘

public SwitchingTable loadSwitchingTable() //从磁盘加载SwitchingTable传给执行层

### 4.2日志层相关文件存储接口

//将日志层传入的日志块存入磁盘

public boolean saveLog(LogTable log)

//从磁盘加载日志块传给日志层

public LogTable loadLog(int logid)

//设置日志块检查点为1

public boolean setLogCheck(int logid)

//设置检查点号

public boolean setCheckPoint(int logid)

//加载日志检查检查点号给日志层

public int loadCheck()