# Totem移动端Android开发

## 开发目标

* 实现以下几类sql语句（语法详见编译章）

创建类（例）：CREATE CLASS person ( id int，age int, name text);

创建代理类（例）：CREATE SELECTDEPUTYCLASS adult   
ATTRIBUTE（place text）

SELECT id as aId，age as aAge FROM person WHERE age>=18;

向源类插入记录（例）：INSERT INTO person VALUES（0,19，张三）；

基础查询（例）：SELECT \* FROM person WHERE id=1；

跨类查询（例）：SELECT adult->person.age FROM adult WHERE aId=1；

* 基本查询执行逻辑无误
* 实现对外存与内存的管理（包括页面设计，缓冲区替换）
* 实现对单一属性建立B-Tree索引
* 实现对数据修改的wal日志记录，能够进行故障恢复

## 开发环境

Totem移动端基于Android 7.0操作系统进行开发，选用当前最流行的Android开发工具Android Studio。在编译层，采用javaCC工具进行语法分析并生成编译层代码。

### Android Studio

安装使用Android Studio首先需要配置好JDK与Android SDK，谷歌为了简化搭建环境步骤，将以上所需的工具都集成好了，可在Android Studio官网进行下载安装即可。如果已有SDK，配置环境时不需再次安装。

### JavaCC(Java Compiler Compiler)

JavaCC(Java Compiler Compiler)是一个用JAVA开发的受欢迎的语法分析生成器。这个分析生成器工具可以读取上下文无关且有着特殊意义的语法并把它转换成可以识别且匹配该语法的JAVA程序。

值得注意的是，JavaCC采用的是自顶向下的分析方法，而且没有回溯功能，因此在设计文法时需要回避一下问题：

1. 存在形如A→αβ1|αβ2的产生式，即有多个候选式的前缀相同（称为公共左因子，或左因子），则可能造成虚假匹配，使得在分析过程中可能需要进行大量回溯。
2. 存在左递归（文法中有形如A→Aα的产生式），分析过程又使用最左推导，就会使分析过程陷入死循环 。

JavaCC的输入文档是一个词法和语法的规范文件，其中也包括一些动作的描述，它的后缀是jj或者jjt。通过JavaCC处理生成7个java类来完成语法分析的工作。

一个jj文档包括以下部分：

1. Options{}部分：这个部分对产生的语法分析器的特性进行说明，例如向前看的token的个数（用来解除冲突）。这部分可以省略的，因为每一个选项都有默认值，当我们没有对某个选项进行说明时，它就采用默认值。示例如下：

*options {*

*STATIC = false ; //生成非静态类*

*LOOKAHEAD = 2;//向前看2个字母,消除歧义用的*

*DEBUG\_PARSER = true;//以debug形式生成,便于调试*

*}*

1. 分析器类的声明：这个部分指定了分析器类的名字，以及其他类中成员的声明，这个部分是必需有的，调用分析器的接口定义由该部分实现。示例如下：

*PARSER\_BEGIN(Sqlparser)*

*public class Sqlparser{*

*public static void sqlParser(String sqlInput){*

*InputStream s=new ByteArrayInputStream(sqlInput.getBytes());*

*System.out.println("reading from standard input");*

*Sqlparser p= new Sqlparser(s);*

*try*

*{p.start();*

*}*

*catch(Exception e) {*

*System.out.println("oops");*

*System.out.println(e.getMessage());*

*e.printStackTrace();*

*}*

*}*

*}*

*PARSER\_END(Sqlparser)*

1. 词法分析器：主要定义了SKIP和TOKEN，SKIP指在进行语法分析时可略过的输入，TOKEN指由用户定义的终结符。示例如下：

*SKIP : { " " }*

*SKIP : { "\n" | "\r" | "\r\n" }*

*TOKEN:/\*定义整数\*/*

*{*

*<INTEGER:["1"-"9"](<DIGIT>)\*>*

*}*

1. 语法分析器：语法中的每一个非终结符都对应一个函数，利用函数调用来表示非终结符之间的结构关系。在进行语法规约时，可在语法中任意插入“{}”，括号体中写入当语法规约到该处时应调用的函数或方法。示例如下：

*//insert into source class only*

*void insert():{*

*Token className;*

*String valueList;*

*InsertClass insert=new InsertClass();*

*}*

*{*

*<INSERT><INTO>className=classname()<VALUES><LEFT>valueList=valuelist()<RIGHT>*

*{insert.clsname=className.image;*

*insert.insertValues=valueList;*

*insert.insertValues();//insert into class file*

*insert.updatePro();//update propagation*

*}*

*}*

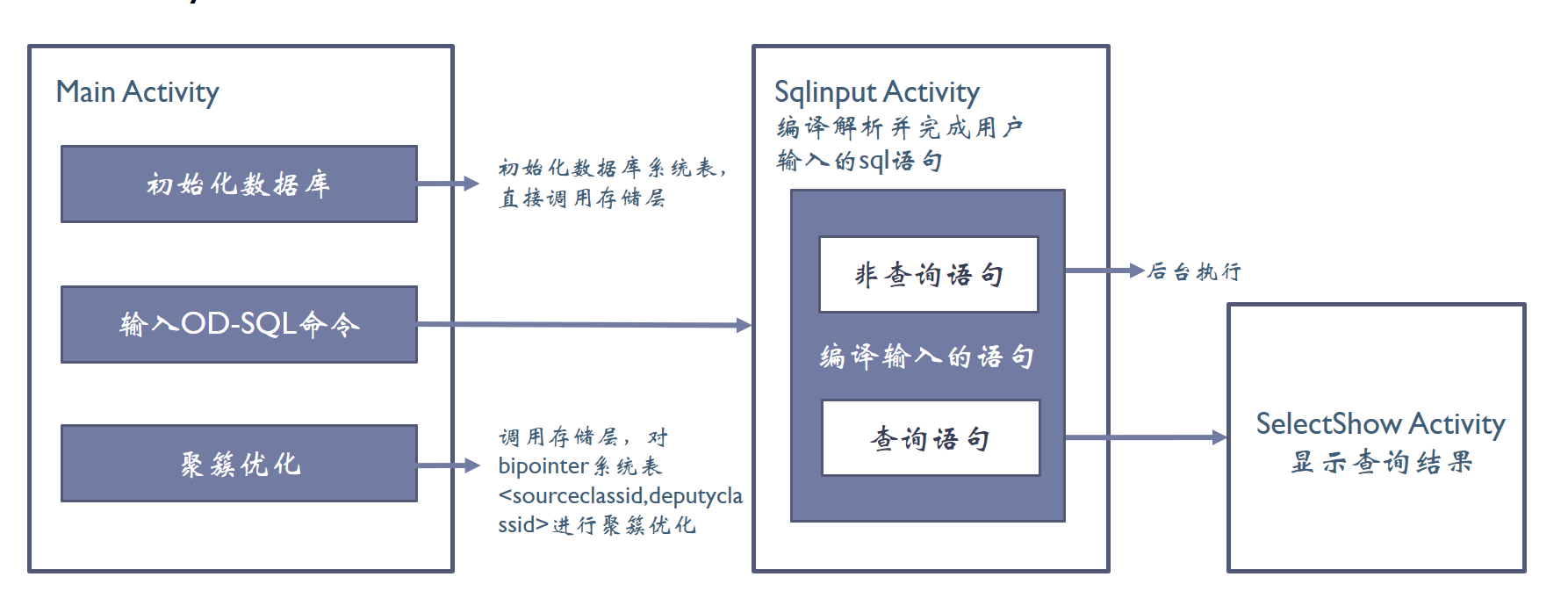
## 代码结构

**编译层**

**存储层（执行）**

**界面**

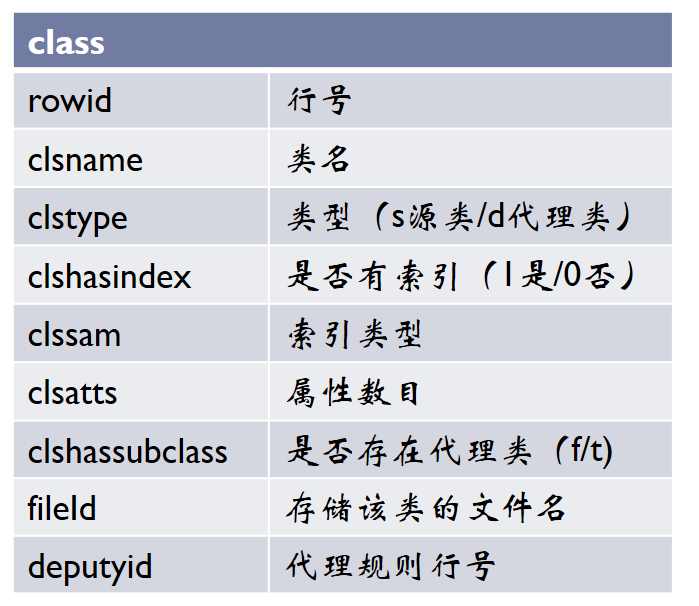
### 界面设计



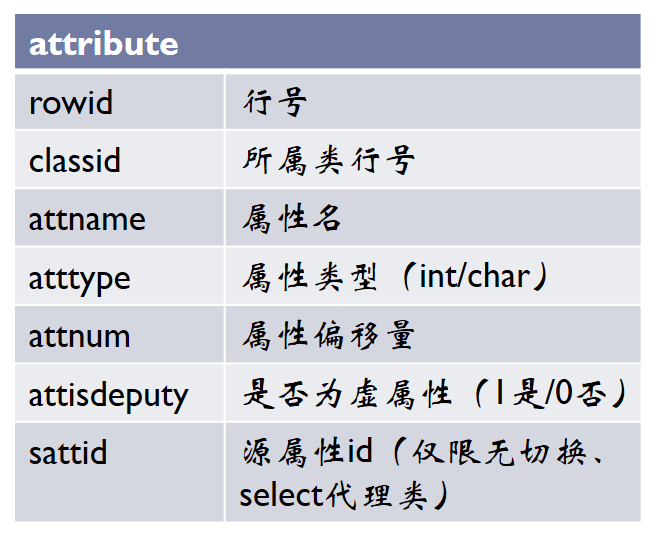
### 编译层

#### 系统表设计

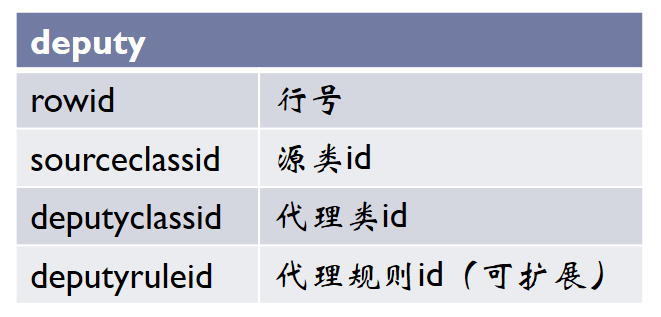
* class系统表：用于记录类于代理类的信息，以该表rowid为元组唯一标识。如下表所示：



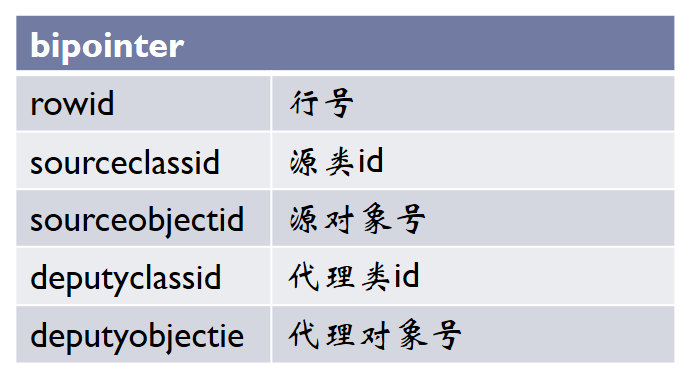
* attribute系统表，主要记录各类属性相关信息，以该表行号rowid为元组唯一标识。如下所示：



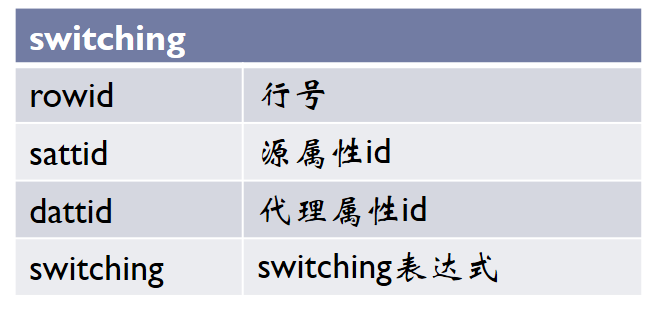
* deputy表，主要记录源类与其代理类之间的代理关系与规则，以该表rowid为元组唯一标识。如下所示：



* bipointer表，记录对象与对象之间的代理关系。如下所示：



* switching表，记录实属性与虚属性之间的切换操作。如下所示：



#### 语法设计

* 创建源类：

CREATE CLASS className

LEFT\_BRACKET attrName attrType [(COMMA attrName attrType)+] RIGHT\_BRACKET；

* 创建代理类：

CREATE SELECTDEPUTYCLASS className

[Attribute LEFT\_BRACKET attrName attrType [(COMMA attrName attrType)+] RIGHT\_BRACKET]

SELECT sAttr AS dAttr [(COMMA sAttr AS dAttr)+]

FROM sClassName

WHERE whereCluase；

* 向源类插入对象：

INSERT INTO className VALUES

LEFT\_BRACKET valueList RIGHT\_BRACKET；

* 普通查询语句：

SELECT attrList FROM className WHERE whereCluase；

* 跨类查询

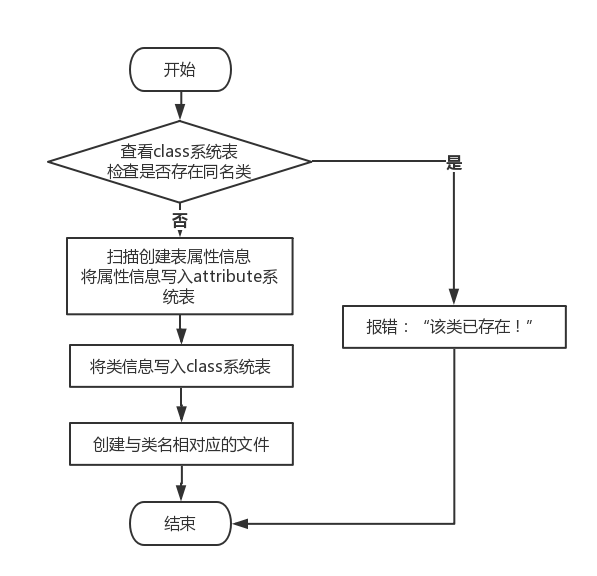
SELECT className [(CROSS className)+]DOTattrName

FROM className

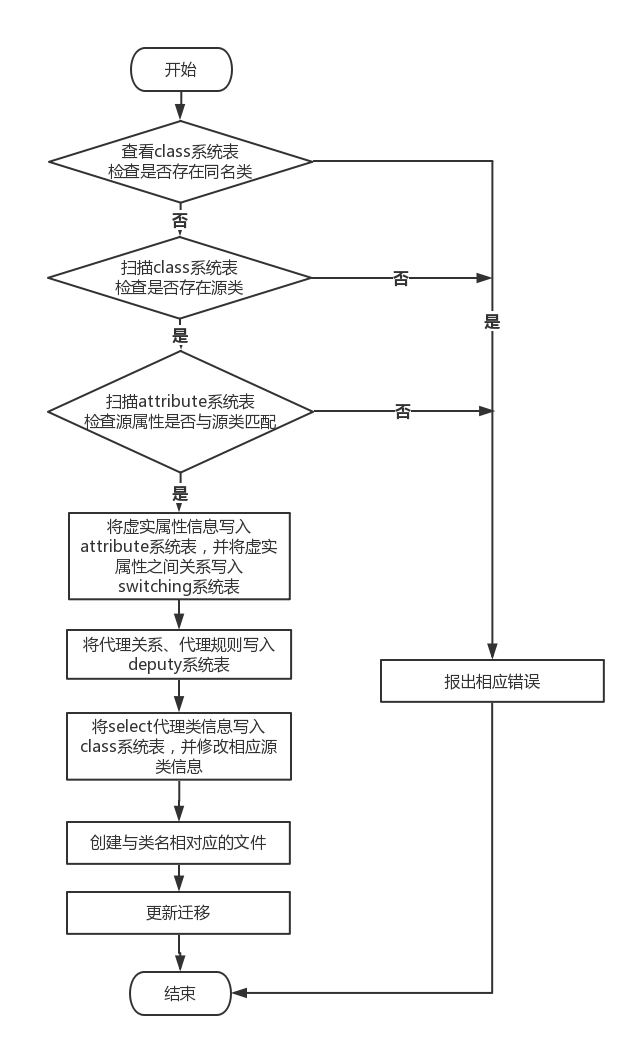
WHERE whereCluase；

#### 执行流程

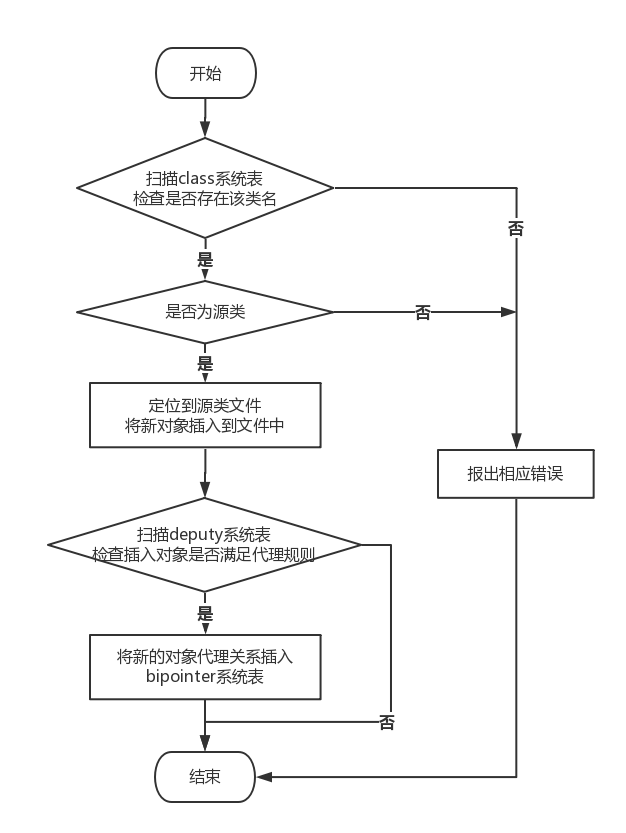
* 创建源类：



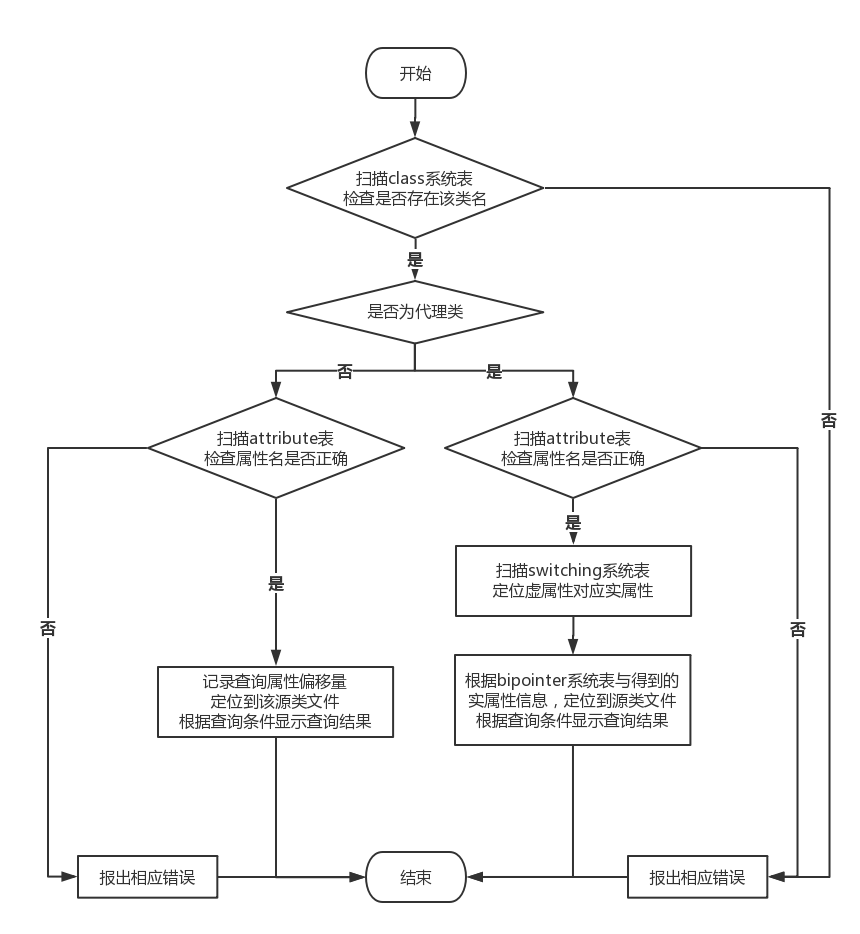
* 创建代理类：



* 向源类插入对象：



* 查询：



### 存储层

#### 页面设计

数据库中所有的表（包括系统表、索引）均以二进制文件的形式存储，每个文件以同等大小的连续页面组织起来。

由于手机采用闪存作为外存，因此设计数据库页面大小为4K，数据通过设定的缓冲区，以字节流的形式写入文件。其页面设置如下：

|  |
| --- |
| 头部信息（例如：指针项的起始位置、空闲空间的首尾偏移量等） |
| Pointer1（记录起始位置（定长记录）/首尾位置（变长记录）） |
| Pointer2 |
| Pointer3 |
| 空闲空间 |
| Record3 |
| Record2 |
| Record1 |

#### 缓冲区

在数据库初始化时，即在内存申请额定大小的缓冲区，将系统表读入缓冲区。缓冲区中页面格式与外存上页面格式相同，分别采用两个带头尾指针的单链表管理已使用的缓冲区与空闲缓冲区，需要在内存中记录两个链表的头尾指针。当需要读入新的数据，即查找空闲缓冲区是否还存在空页，若存在，从链头取出一页连接到已使用缓冲区链尾；若不存在，则需要采用缓冲区替换策略将某个缓冲区替换出去。其中，为了保证缓冲区替换的数据一致性，需要在内存中使用一张表记录内存地址与外存地址之间的映射等信息，如下表所示：

|  |  |
| --- | --- |
| 缓冲区信息表 | |
| bufferId | 缓冲区号 |
| filePosition | 物理页号（文件名，页偏移量） |
| isFree | 是否空闲 |
| nextBuf | 下一个缓冲区 |
| pin | 申请使用的进程数 |
| usageCount | 最近使用次数 |
| isRevise | 是否为脏 |

替换算法可采用LRU或Clock算法。

#### 索引

为了使查询效率更高，通常需要在某些表属性上建立索引，在此仅考虑可导航连续值的b-tree索引，可按某表单属性的值建立一棵b-tree，以提高查询效率。

索引同样使用上述页面结构来存储，最终组合成一个索引文件。

需要完成编译语句：CREATE INDEX ON className.attrName；

在进行查询时，优先选择通过索引查询。

### 事务管理

采用WAL对崩溃的操作进行redo操作，当进行写或者删除操作时，在事务提交前保证日志已经记录到磁盘，数据文件可不做修改，读取时将读取数据文件与WAL文件即可获得最新的数据。

WAL机制的原理是：修改并不直接写入到数据库文件中，而是写入到另外一个称为WAL的文件中；如果事务失败，WAL中的记录会被忽略，撤销修改；如果事务成功，它将在随后的某个时间被写回到数据库文件中，提交修改。

同步WAL文件和数据库文件的行为被称为checkpoint（检查点），定期对WAL文件checkpoint，可减少崩溃后的redo操作。

WAL同样以表文件存储，其模式信息如下：

|  |  |
| --- | --- |
| WAL日志 | |
| trasId | 事务号 |
| filePositipon | 物理页面号 |
| itemId | 修改页面中记录的偏移量 |
| itemOff | 修改元组的属性偏移量 |
| newValue | 需要redo的新值 |