**Mycat路由新解析器选型分析与结果**

# 前言

Mycat之前只支持一种解析器fdb parser(FoundationDB SQL Parser)，该解析器来源于Apache Derby parser，fdb parser为FoundationDB的一个子项目，FoundationDB是支持ACID事务处理的NoSQL数据库，提供非常好的性能、数据一致性和操作弹性，基于表格存储。FoundationDB项目地址：<https://foundationdb.com/>

fdb-sql-parser 项目地址：<https://github.com/FoundationDB/sql-parser>

fdbparser解析器存在的问题：

1. **修改解析器源码的门槛太高。**使用了javacc解析器，如果要修改解析器的源码必须搞清楚javacc的原理（修改解析器源码是有时碰到不支持的语法，要修改解析器来支持）
2. **没有好的api接口**获取ast语法树中的表名、拆分字段条件等，所以路由解析时的代码很难有好的结构，就是写的很让人看不懂。
3. **支持的语句太少。**如insert into …. On duplicate key update….，带注释的create table语句不支持，还有很多就不列举了
4. **解析性能很差。**我们公司的sql一般都很长（select语句），一个长点的sql解析花了3、4秒解析出ast语法树。这个在业务上无法让人忍受（当然，这么慢与我的开发机器有关，2核4G的破机器，如果用好的服务器可能也用不了这么久）。

# 解析器选择

从开源项目中找java语言开发的sql解析器，找到了两种：

jsqlparser

项目地址：<https://github.com/JSQLParser/JSqlParser>

Druid SQL Parser

<https://github.com/alibaba/druid/wiki/SQL-Parser>

该解析器为阿里的温少开发，他在金蝶开发过ksql解析器（ksql屏蔽了oracle、sqlserver、db2这3种数据库的差异，使业务开发人员只需要写ksql就行，不需要管后面连接的到底是那3种数据库中的哪一种，实际就是将ksql转成3种数据库的方言）。后来温少去了阿里，开发了druid和fastjson，fastjson号称最快的json解析器，比jackson都快，这个应该是有测试数据为证的。温少在解析器领域的比较有研究。

Druid主要特点是连接池的监控功能，sql解析器只是它的副业，但其解析性能却非常出色。

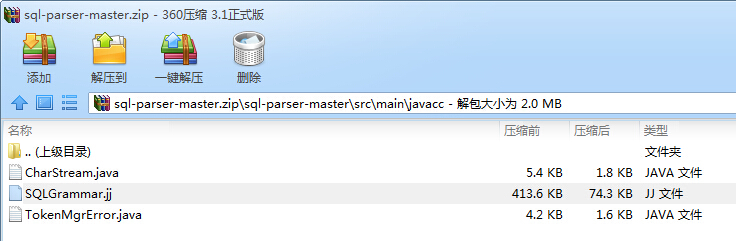
## 解析器实现方式

Fdbparser和jsqlparser都是基于javacc实现的，jsqlparser的官网明确说明是基于javacc

The parser is built using **JavaCC**

Fdbparser打开源码包能看到sql-parser-master.zip\sql-parser-master\src\main\javacc\

SQLGrammar.jj文件，这是使用javacc的典型标识。



## 几种解析器的性能对比

**对比方式：**

对同一个sql语句，使用3种解析器解析出ast语法树（这是编译原理上的说法，在sql解析式可能就是解析器自定义的statement类型），执行10万次、100万次的时间对比。

**import** java.sql.SQLSyntaxErrorException;

**import** net.sf.jsqlparser.JSQLParserException;

**import** net.sf.jsqlparser.parser.CCJSqlParserUtil;

**import** net.sf.jsqlparser.statement.Statements;

**import** org.opencloudb.parser.SQLParserDelegate;

**import** com.alibaba.druid.sql.ast.SQLStatement;

**import** com.alibaba.druid.sql.dialect.mysql.parser.MySqlStatementParser;

**import** com.foundationdb.sql.parser.QueryTreeNode;

**public** **class** TestParser {

**public** **static** **void** main(String[] args) {

String sql = "insert into employee(id,name,sharding\_id) values(5, 'wdw',10010)";

**int** count = 1000000;

**long** start = System.*currentTimeMillis*();

System.***out***.println(start);

**try** {

**for**(**int** i = 0; i < count; i++) {

QueryTreeNode ast = SQLParserDelegate.*parse*(sql,"utf-8" );

}

} **catch** (SQLSyntaxErrorException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**long** end = System.*currentTimeMillis*();

System.***out***.println(count + "times parse,fdb cost:" + (end - start) + "ms");

start = end;

**try** {

**for**(**int** i = 0; i < count; i++) {

Statements stmt = CCJSqlParserUtil.*parseStatements*(sql);

}

} **catch** (JSQLParserException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

end = System.*currentTimeMillis*();

System.***out***.println(count + "times parse,JSQLParser cost:" + (end - start) + "ms");

start = end;

**for**(**int** i = 0; i < count; i++) {

MySqlStatementParser parser = **new** MySqlStatementParser(sql);

SQLStatement statement = parser.parseStatement();

}

end = System.*currentTimeMillis*();

System.***out***.println(count + "times parse ,druid cost:" + (end - start) + "ms");

}

}

**输出结果：**

100000times parse,fdb cost:4549ms

100000times parse,JSQLParser cost:2892ms

100000times parse ,druid cost:456ms

**10万次：druid比fdbparser快10倍，比JSQLParser快6倍**

1000000times parse,fdb cost:30280ms

1000000times parse,JSQLParser cost:18983ms

1000000times parse ,druid cost:1912ms

**100万次：druid比fdbparser快15倍，比JSQLParser快近10倍**

**如果sql越长，其快的倍数会越多，之前在公司有一条长的select语句，druid解析能比fdbparser快40倍。**

## Druid速度快的原因分析

Druid的官网已经明确写出了druid的性能：

Druid的SQL Parser是手工编写，性能是antlr、javacc之类工具生成的数倍甚至10倍以上。

### 速度快原因分析

以下是我通过调试druid解析和fdbparser解析的执行过程得出的（JSQLParser没调试过）：

Druid的解析过程使用“统筹方法”，就是几件事情同时一起做。Druid解析包含词法解析和语法解析，这两件事情它只需要从字符的第一个到最后一个遍历一遍，就同时完成了词法解析和语法解析，语法树也已经构造完成。

词法解析得到一个个的Token，语法解析得到各种有含义的表达式。

如一个select \* from tableName语句，词法解析就是把这个字符串看成一个数组形式，一个索引pos不停的++，从这个字符串的第一个字符一直++达到最后字符的索引就结束了。

Fdbparser由于依赖于javacc解析，做了很多无用功，它是解析完所有的词法，中间的对象得到后最后再构造出ast语法树对象。

如com.foundationdb.sql.parser. NodeFactory类的方法：

**public** **final** QueryTreeNode getNode(**int** nodeType,

Object arg1,

Object arg2,

Object arg3,

Object arg4,

Object arg5,

Object arg6,

SQLParserContext pc)

**throws** StandardException {

QueryTreeNode retval = getNode(nodeType, pc);

retval.init(arg1, arg2, arg3, arg4, arg5, arg6);

**return** retval;

}

retval.init(arg1, arg2, arg3, arg4, arg5, arg6);就是在构造最终的对象。

这还不是最关键的慢的原因，还有个更重要的是，他借用javacc解析，解析过程中还要回调fdbparser自己的com.foundationdb.sql.parser. StringCharStream类中的词法遍历。

StringCharStream类中 中advance()是往后遍历，backup方法是往回退，想想，老被javacc回调走backup方法，能快的起来吗？这索引往后移动一位不容易，我这不停的F6想它索引赶紧遍历完，结果手都摁酸了，一个backup给我又退回去好远，简直快崩溃了。这sql语句insert into employee(id,name,sharding\_id) values(5, 'wdw',10010)，60多个字符，摁很久才能到头。

Druid的com.alibaba.druid.sql.parser. Lexer类是其词法解析，这个解析实际用到了双指针+buf来解决不回退的问题。一个指针pos不停的++，调试没见到 -- 过，另一个指针由mark和bufPos组成，这个是为了标记以读出字符串，pos指针是为了读出单个字符。

解析过程中就是不停的pos++，scanChar()、scanString()，读取到char后放入buf，碰到字符分割符号之类的就scanString（）读取成一个字符串。

相比fdbparser，druid减少了与javacc回调的过程，双指针不用像单指针一样需要回退，减少了很多无用功，所以能比较快。

## Druid支持的语法

**各种语法支持**

Druid的sql parser是目前支持各种数据语法最完备的SQL Parser。目前对各种数据库的支持如下：

|  |  |  |
| --- | --- | --- |
| 数据库 | DML | DDL |
| mysql | 完全支持 | 支持大部分 |
| oracle | 大部分 | 支持大部分 |
| postgresql | 完全支持 | 支持大部分 |
| sql server | 支持常用的 | 支持常用的ddl |
| db2 | 支持常用的 | 支持常用的ddl |

druid还缺省支持sql-92标准的语法，所以也部分支持其他数据库的sql语法。

Druid支持的语法比fdbparser多。

## 选择解析器的结论

经过上述分析和对比，最终选择新增druid解析器的支持，当然原解析器继续支持，两种解析器支持配置文件切换。

## Fdbparser与druidparser是并存还是直接舍弃一种？

来自群主的建议：Fdbparser虽然有很多弊端，但是也有它存在的理由，它更偏向于标准sql。

更能引导大家写标准sql，而不是mysql方言，这样应用程序才有可能支持多种数据库，领导说要换数据库时，不至于因为都是非标准sql要大改才能迁移到其他数据库。