**Mycat路由解析开发指南**

# 切换解析器配置

解析器的切换修改server.xml文件中的以下内容：

在<system></system>内部加入

<!--默认的sql解析器,可选值fdbparser,druidparser -->

<property name=*"defaultSqlParser"*>druidparser</property>

druidparser为新解析器，fdbparser为原先的解析器。

# Druid解析器的优势

1. 性能更高。

druidparser为新解析器，该解析器单独从解析性能上比原解析器（fdbparser）快5倍以上，甚至10倍以上，sql越长，快的倍数越多。曾经对一个长sql解析测试，能达到40倍左右。

1. 支持的语法更多。

下面列举一些fdbparser不支持，但是druidparser支持的语法：

（1）Insert into ….on duplicate key update…..

（2）Insert into (),()…语句

（3）带注释（comment）的create table语句

（4）alter table … change….语句;

（5）alter table … modify….语句;

（6）添加索引时带索引名称的；如alter table coding\_rule add unique ux\_indexname (prefix);

（7）create table tablename select \* from othertable;

之前很多不支持的语法mycat需要使用hint来支持，现在只需要换成druidparser就可以了。

1. 编码更容易。

# 路由解析入口代码

路由解析入口都从RouteService类的route方法进入，然后根据是否有注解决定是走HintHandler还是RouteStrategy进行路由解析。



# 相关类图和序列图

## 路由策略相关类图



图4-1 整体路由策略相关类图

路由解析使用了策略模式，每种解析器实现一种路由策略。还可以继续扩展，如Druid解析再细分Mysql、postgresql、oracle等实现策略。本次只实现druid解析的mysql的策略，其他暂时忽略。

## Druid语法解析

### 解析方式

Druid解析有两种方式：vistor方式和statement方式。

#### Vistor方式的用法：

String sql = “select \* from tableName”;

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

SQLStatement statement = parser.parseStatement();

MycatSchemaStatVisitor visitor = **new** MycatSchemaStatVisitor();

stmt.accept(visitor);

经过上面的步骤后，你可以很方便的从visitor中获取表名、条件、表别名map、字段列表、值类表等信息。用这些信息就可以做路由计算了。

#### Statement方式的用法

String sql = “select \* from tableName”;

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

SQLStatement statement = parser.parseStatement();

SQLSelectStatement selectStmt = (SQLSelectStatement) statement;

然后就可以从selectStmt里面得到想要的信息去了。

如果sql = “delete from tableName”;

就要转型为MySqlDeleteStatement

MySqlDeleteStatement deleteStmt = (MySqlDeleteStatement) statement;

### Druid语法树解析相关类图



图4-2 Druid语法解析相关类图

类图说明：DruidMysqlRouteStrategy会根据解析出来的Statement（AST语法树）来调用相应的解析器进行解析，解析后的结果会存放到DruidShardingParseInfo 类中（解析结果信息：表名、条件等），用户后面计算路由。

DruidParser接口方法介绍：

|  |  |
| --- | --- |
| 方法名 | 用途 |
| parser | 解析的入口方法 |
| visitorParse | 通过visitor解析，可以很方便的获取到表名、条件、字段列表、值列表等  对各种语句的statement都适用 |
| statementParse | statement方式解析。子类覆盖该方法一般是将SQLStatement转型后再解析  （如转型为MySqlInsertStatement） |
| changeSql | 该方法用来改写sql。如select语句加limit，insert语句加自增长值等。  主要是为了代码结构化，  实际你完全可以把这里面的工作放到statementParse中来做 |
| getCtx | 获取解析结果。返回DruidShardingParseInfo对象。该对像包含解析到的表名列表 条件列表等信息。用户后续计算路由 |

DruidParser接口有一个默认实现DefaultDruidParser，该类相当于一个模板类，parser方法是其模板方法。模板方法规定了解析步骤：visitorParse、statementParse、changeSql、ctx.setSql(stmt.toString())4个步骤挨个执行。

所有的子类都继承自该模板类。

Druid对SQLStatement解析时，大多数类型的statement通过visitorParse这一个方法解析完就得到了我们计算分库路由的所有信息（表名、条件字段等），如果visitorParse后还有信息没解析出来，就通过statementParse，通过这两种方式的解析之后，所有的路由需要的信息都会得到。

### 每种Stament是否必须有一个DruidParser的实现类？

**Druid的SQLState**ment有很多子类，如下图，我们是否需要每种statement都实现一个子类呢？不需要都实现，一般的statement我们使用visitorParse方式解析就能得到我们进行路由的所有信息了，visitorParse在模板类DefaultDruidParser中已经有了统一的实现。如果没有特殊需求的，让他走默认的DefaultDruidParser解析足矣。

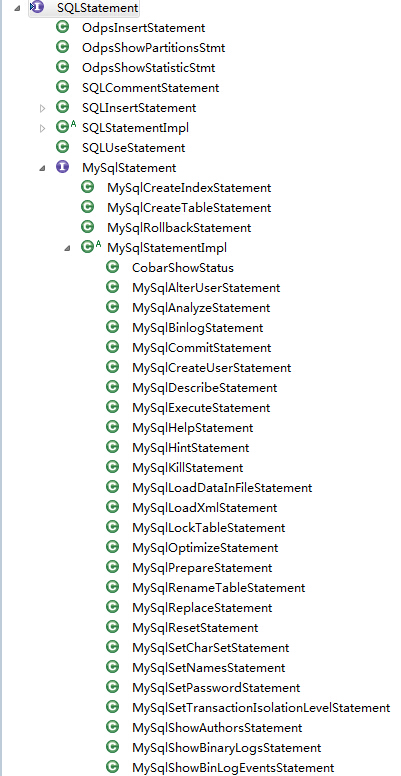


图4-2 SQLStatement的子类

### DefaultDruidParser类的子类应该重写哪些方法？

如果visitorParse解析得不到任何路由相关的信息，如表名、条件等，就直接重写为空方法（没用的防止它耗性能所以弄成空实现），同时重写statementParse方法，使用stament方式解析。

有些时候statementParse能得到一部分信息，visitorParse也能得到一部分信息，就让两种解析方式共存。

changeSql方法的重写：该方法完全是为了代码结构化，你完全可以直接把改写sql的逻辑写在statementParse方法中。

## Druid路由解析序列图



图4-2 druid路由解析序列图

由于里面还有比较多细节，画在一个图中看着比较乱，所以只画出主要的大步骤。