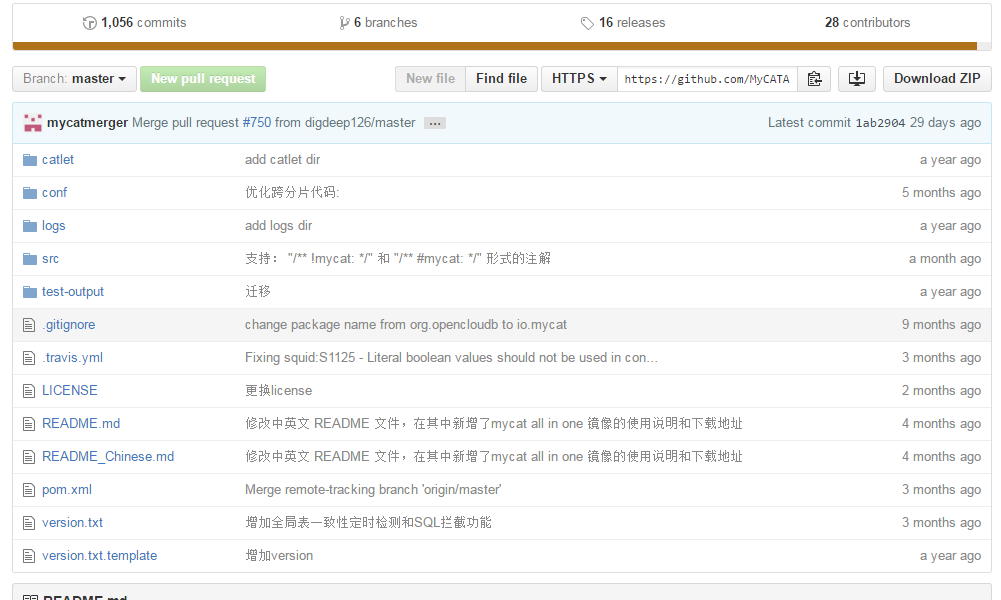
**Mycat路由解析开发指南**

**Mycat 新一代Mysql分布式集群，大数据处理中间件，中国第一开源软件，欢迎志愿者参与：**

**项目源码路径：**<https://github.com/MyCATApache/Mycat-Server>



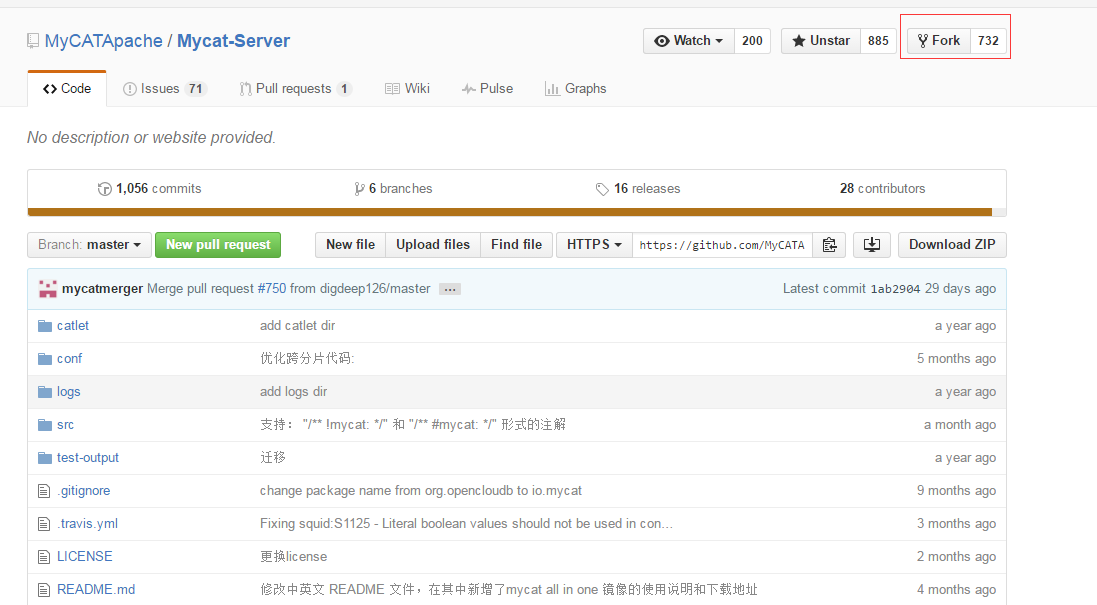
本文档由Mycat志愿者团队提供，转载请注明，谢谢。

# Github使用方法

1. 注册账户以及创建仓库

要想使用github第一步当然是注册github账号了。之后就可以创建仓库了（免费用户只能建公共仓库），Create a New Repository，填好名称后Create，之后会出现一些仓库的配置信息，这也是一个git的简单教程。

1. GitHub协作  
   先从主库fork出自己的分支， 然后进行代码的修改等操作，操作完之后从本地库上推到自己的服务器分支，然后 服务器分支Pull Request到主干。



1. 本地仓库由git维护的三棵“树"组成：第1个是工作目录，它持有实际文件（即可以直接打开修改的文件）；第2个是缓存区（index），它像个缓存区域，临时保存你的改动；最后一个是HEAD，指向你最近一次提交后的结果（这个结果还是在本地电脑里面）。

1）将改动添加到缓存区：

git add <filename>或者 git add \* ;

2)上面是git基本工作流程的第一步，使用如下命令以实际提交改动：

git commit -m "提交的备注信息"

3）现在改动已经提交到了HEAD，但是还没有到你的远程仓库，执行如下将改动提交到远程仓库（可以理解为github网站自己的主页）：其中master可以换成其他想要的分支

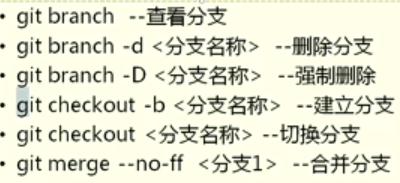
git push origin master

1. 参与社区协作项目的常用命令：



客户端的 git push: 将本地库更新到自己的服务器分支。

1. 管理分支的常用命令：



1. 参与到社区化写作：

  1）进入github的项目的主页面，点击右上角的Fork按钮即可建立自己的分支。（这里fork是指复制到自己的github网站账号上面，要想复制到本地，还需要一些add指令）

    //这里的分支是指上一步fork到自己github网站账号上面的分支

  2）从自己的分支复制到本地。git clone git@github.com:thatcat/manual56.git

   //将主库的URL添加到本地的版本库，然后可以进行下一步的操作

  3）添加主库URL：git remote add mysql2cn git@github.com:mysql2cn/mamual56.git

  4)获取主库的修改：git fetch mysql2cn //将**主库**的修改同步到**本地**

5)合并主库代码到自己的本地分支上：git merge mysql2cn/master master

    //也可以用git pull origin master 同步自己github网站的分支同步到本地

  6)上推到自己的github网站账号：git push origin master

  7)将自己的github网站分支将更新同步到主库：在自己的github网站分支按“pull request”按钮

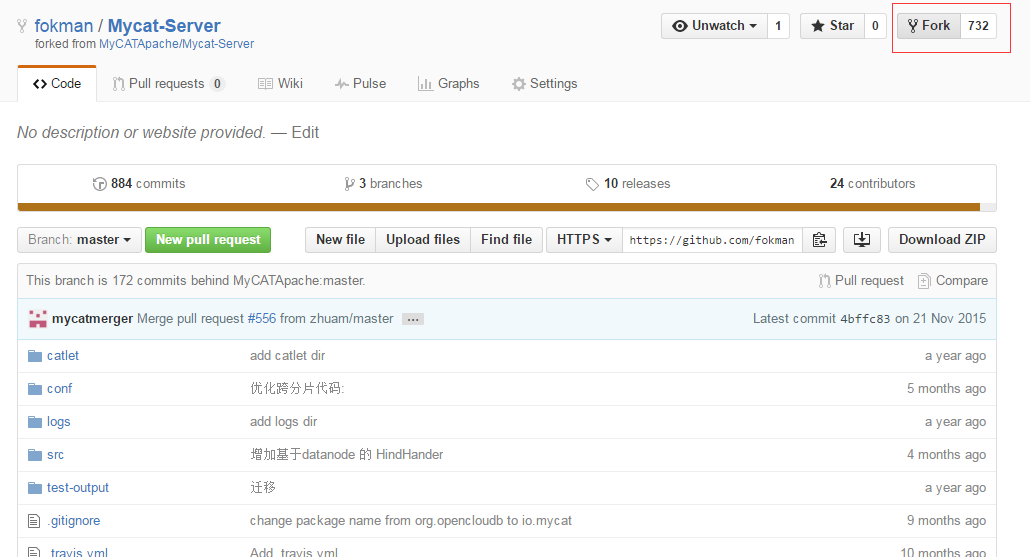
  总结： （1）完整流程：github的项目—>fork到自己的github网站分支—>clone到自己的本地版本—>修改后push到自己的github网站分支—>将自己gitub网站分支pull request到github的原项目

            （2）将原项目的修改同步到自己的分支：github的项目—>fetch、merge到本地版本库—>push到github网站分支—>将自己gitub网站分支pull request到github的原项目

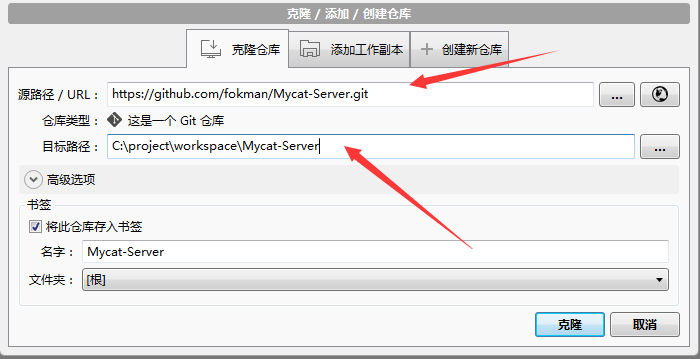
1. 创建新仓库：1)在github网站new一个仓库，复制该仓库URL；2）在电脑本地的git bash克隆刚刚new出来的URL:git clone URL；

## Checkout项目

采用SourceTree或者Github client来checkout项目

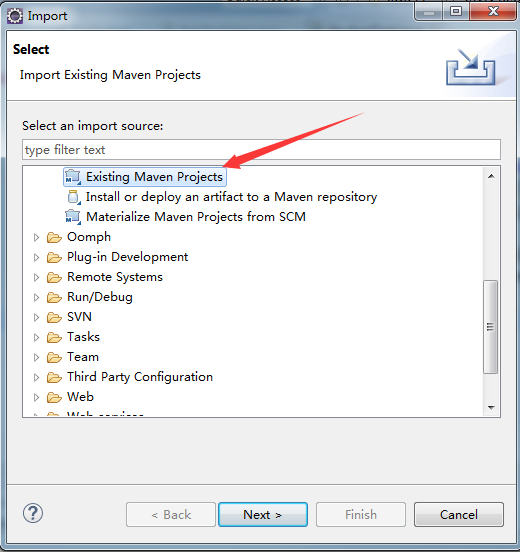


使用Source Tree导入git repository

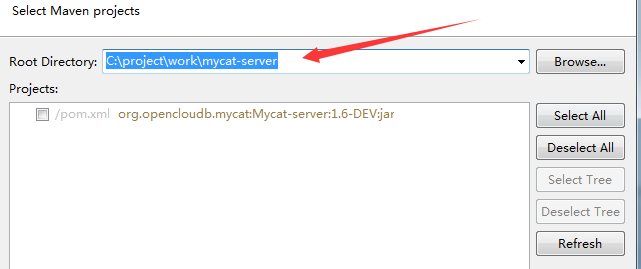


**Eclispe中启动**

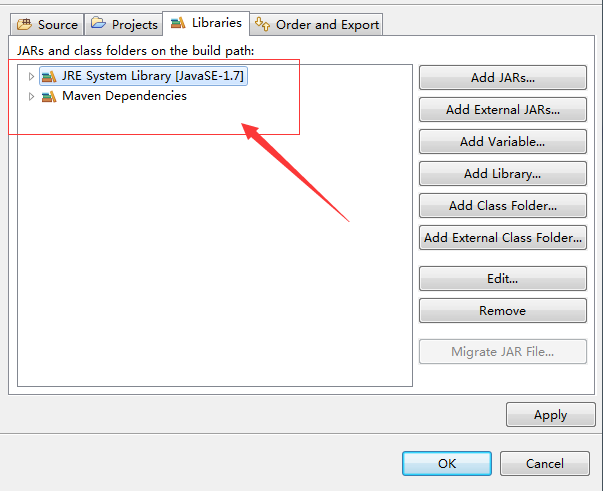
从eclipse Import Mycat-server project ，选择Existing Maven Projects



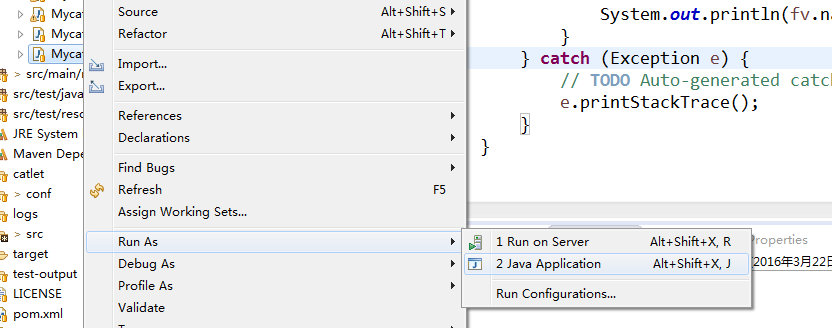
**指定下载完成后的源代码工程路径**



指定JDK版本为1.7



在eclipse中运行MycatStartup 类



# Mycat分片函数的扩展

下面说明分片函数的定义和配置，以Mycat中的例子为例，长整形范围自动分片规则**auto-sharding-long**的定义如下：

<tableRule name="**auto-sharding-long**">

<rule>

<columns>**id**</columns>

<algorithm>**rang-long**</algorithm>

</rule>

</tableRule>

<function name="rang-long" class="org.opencloudb.route.function.AutoPartitionByLong">

<property name="mapFile">autopartition-long.txt</property>

</function>

上述表明：用字段id作为分片字段，并采用命名为**rang-long**的分片函数**： org.opencloudb.route.function.AutoPartitionByLong** 来完成分片映射，此分片函数有一个属性**mapFile**，配置为autopartition-long.txt，意思是从这个文件中加载分片定义：

# range start-end ,data node index

0-2000000=0

2000001-4000000=1

4000001-6000000=2

6000001-8000000=3

8000001-10000000=4

0到2000000的id分片到节点0，2000001到4000000的id分片到节点1，依次类推。

Mycat的分片函数的的目标只有一个：根据某个字段的值，返回其所对应的分片节点DataNode的index，index从零开始；若字段的值为非法，则返回NULL。

用户自定义扩展的分片函数要实现org.opencloudb.config.model.rule.RuleAlgorithm接口：

public interface RuleAlgorithm {

void init();

Integer calculate(String columnValue);

}

分片函数可以定义类的属性，并生成对应的set方法，当此函数被加载的时候，会从 rule.xml文件中找相关的属性，并注入（调用set方法），最后调用init()接口，完成初始化过程，这个过程可以实现资源文件的加载和数据解析。

Calculate方法接受字段的值，并计算返回一个分片节点的index。

附：AutoPartitionByLong的例子

public class AutoPartitionByLong implements RuleAlgorithm {

private String mapFile;

private LongRange[] longRongs;

@Override

public void init() {

initialize();

}

public void setMapFile(String mapFile) {

this.mapFile = mapFile;

}

@Override

public Integer calculate(String columnValue) {

long value = Long.valueOf(columnValue);

Integer rst = null;

for (LongRange longRang : this.longRongs) {

if (value <= longRang.valueEnd && value >= longRang.valueStart) {

return longRang.nodeIndx;

}

}

return rst;

}

private void initialize() {

BufferedReader in = null;

try {

// FileInputStream fin = new FileInputStream(new File(fileMapPath));

InputStream fin = this.getClass().getClassLoader()

.getResourceAsStream(mapFile);

if (fin == null) {

throw new RuntimeException("can't find class resource file "

+ mapFile);

}

in = new BufferedReader(new InputStreamReader(fin));

LinkedList<LongRange> longRangeList = new LinkedList<LongRange>();

for (String line = null; (line = in.readLine()) != null;) {

line = line.trim();

if (line.startsWith("#") || line.startsWith("//"))

continue;

int ind = line.indexOf('=');

if (ind < 0) {

System.out.println(" warn: bad line int "+mapFile+ " :"+line);

continue;

}

try {

String pairs[] = line.substring(0, ind).trim().split("-");

long longStart = Long.parseLong(pairs[0].trim());

long longEnd = Long.parseLong(pairs[1].trim());

int nodeId = Integer.parseInt(line.substring(ind + 1)

.trim());

longRangeList

.add(new LongRange(nodeId, longStart, longEnd));

} catch (Exception e) {

}

}

longRongs = longRangeList.toArray(new LongRange[longRangeList

.size()]);

} catch (Exception e) {

if (e instanceof RuntimeException) {

throw (RuntimeException) e;

} else {

throw new RuntimeException(e);

}

} finally {

try {

in.close();

} catch (Exception e2) {

}

}

}

static class LongRange {

public final int nodeIndx;

public final long valueStart;

public final long valueEnd;

public LongRange(int nodeIndx, long valueStart, long valueEnd) {

super();

this.nodeIndx = nodeIndx;

this.valueStart = valueStart;

this.valueEnd = valueEnd;

}

}

}

单元测试代码如下：

public class AutoPartitionByLongTest {

@Test

public void test()

{

AutoPartitionByLong autoPartition=new AutoPartitionByLong();

autoPartition.setMapFile("autopartition-long.txt");

autoPartition.init();

String idVal="0";

Assert.assertEquals(true, 0==autoPartition.calculate(idVal));

idVal="2000000";

Assert.assertEquals(true, 0==autoPartition.calculate(idVal));

idVal="2000001";

Assert.assertEquals(true, 1==autoPartition.calculate(idVal));

idVal="4000000";

Assert.assertEquals(true, 1==autoPartition.calculate(idVal));

idVal="4000001";

Assert.assertEquals(true, 2==autoPartition.calculate(idVal));

}

}

# 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为新版解析器，fdbparser为老板的解析器。

# 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);

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。Druid解析出路由信息可以有两种方式：visitor方式、statement方式，其api比较方便的提取表名、条件表达式、字段列表、值列表等信息。而且可以很容易的通过ast语法树改写sql，这对人工智能路由比较有帮助。

# 路由解析流程

本文采用自顶向下方式，从粗到细逐步展开路由解析流程。

## 路由解析总体流程



图3-1路由解析总体流程

其中RouteStrategy路由为子流程，见[3.2RouteStrategy路由子流程](#_RouteStrategy路由流程)对其展开讲解。HintHandler路由也是子流程，但非主流程故本文不做重点讲解。



图3-2路由解析序列图

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。

## RouteStrategy路由子流程



图3-3 RouteStrategy路由流程

该流程是fdbparser和druidparser两种解析策略的公共流程。该流程封装在AbstractRouteStrategy类的route方法中，相当于两种策略的模板方法。子流程“Ast语法树解析”对应routeNormalSqlWithAST方法，下一节将对ast语法树解析流程再展开讲解（以DruidMysqlRouteStrategy策略类为例）。

## DruidMysqlRouteStrategy的AST语法树解析流程



图3-4 DruidMysqlRouteStrategy之AST语法树解析流程

## DruidParser解析子流程

### 此处DruidParser解析的含义

**说明：**DruidParser解析指的是利用ast语法树（SQLStatement，这是druid解析器已经解析出来的）解析出表名、条件表达式、字段列表、值列表等信息，用于我们计算路由的过程。



图3-5 DruidParser解析子流程

该流程封装在DefaultDruidParser类的parser方法中。

### 两种解析方式

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;

## 路由计算

### 路由计算时机

路由计算有两个时机：DruidParser解析的同时计算路由、DruidParser解析完之后再计算路由。

一般情况下都是解析完之后再计算，有些特殊场景需要边解析边计算路由（复杂的改写sql，见[6.2节改写insert into …values (),()….语句](#_支持insert_into_…)）。

两种时机的控制由RouteResultset类中setFinishedRoute(true)来控制，如果DruidParser边解析边计算，计算完之后需要调用setFinishedRoute(true)，如果要等到解析完之后再计算，则不要设置该变量。

### DruidParser解析的同时计算路由

该场景一般是要改写复杂sql，将一个sql拆成多个sql发往不同的dataNode时才使用。

选择该时机时，一般都是继承DefaultDruidParser，然后在重写statementParse方法，在该方法完成路由计算与设置，

rrs.setNodes(nodes);//设置路由节点

rrs.setFinishedRoute(true);//设置为完成了路由

参考DruidInsertParser类对insert into… values (),()…的改写。代码目前注释掉了，因为druid项目有些方法还没有提供，已经对druid项目pull request，待merge。

### DruidParser解析完之后再计算路由

除特殊需求外，大多数情况下都是采用该时机计算路由。具体的计算在DruidMysqlRouteStrategy类的tryRouteForTables方法中。

调用示例：tryRouteForTables(schema, druidParser.getCtx(), druidParser.getCtx().getTables(), rrs, isSelect(statement));

### 路由计算算法

**无表**：一般是查询系统参数，如select @@xxx;这种直接返回schema的中的任意一个dataNode。

**单表**：全局表，select语句返回该表的任意一个dataNode，insert 、delete、truncate等其他返回所有dataNode；拆分表，insert必须带拆分字段，根据拆分字段和拆分算法找dataNode，select语句，如果条件中有拆分字段，根据拆分字段找dataNode，没有拆分字段，返回所有dataNode；非拆分的单表，直接返回对应的dataNode。

**多表**：对每个表分别得到dataNode集合，然后对集合求交集。没有交集的抛异常invalid route in sql, multi tables found but datanode has no intersection，有交集的返回交集中的所有dataNode。

本文只简要介绍了一些路由算法的原则和思路，具体的可以看DruidMysqlRouteStrategy类中的tryRouteForTables方法。

# 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），用于后面计算路由。

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。用于后续计算路由 |

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Statement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。

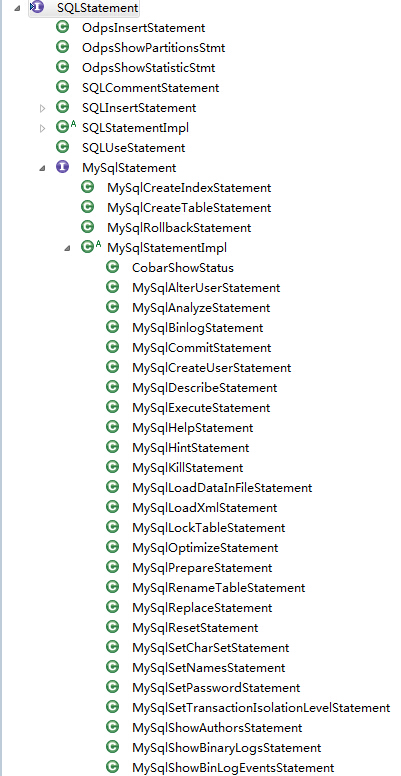


图4-2 SQLStatement的子类

### 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。

# 开发中会遇到的问题及解决方案

## 碰到不支持的sql语法怎么办？

如果是

String sql = “xxxxxxxxxxxxxxxxxx”;

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

SQLStatement statement = parser.parseStatement();

这一步都解析不出SQLStatement的，要么是自己语句真的写错了，可以到mysql上执行看看是否mysql都不识别该语句。如果在mysql中支持，但是这一步解析报错的，就去给druid项目提issue。

## 路由解析错误怎么办？

路由解析完成了，但是解析的不对，比如某个语句本来应该到dataNode1执行的，结果被解析到dataNode2或者dataNode1和dataNode2都执行了。

出现该问题一般是DruidMysqlRouteStrategy类中以下代码中工厂方法获取到的是DefaultDruidParser类型的实例，而DefaultDruidParser中的解析不足以满足该类型得到正确的路由。

DruidParser druidParser = DruidParserFactory.*create*(statement);

druidParser.parser(schema, rrs, statement);

**解决方案：**

需要在DruidParserFactory工厂方法中增加新的SQLStatement类型的DruidParser实例。

**public** **static** DruidParser create(SQLStatement statement) {

DruidParser parser = **null**;

**if**(statement **instanceof** SQLSelectStatement) {

parser = **new** DruidSelectParser();

} **else** **if**(statement **instanceof** MySqlInsertStatement) {

parser = **new** DruidInsertParser();

} **else** **if**(statement **instanceof** MySqlDeleteStatement) {

parser = **new** DruidDeleteParser();

} **else** **if**(statement **instanceof** MySqlCreateTableStatement) {

parser = **new** DruidCreateTableParser();

} **else** **if**(statement **instanceof** MySqlUpdateStatement) {

parser = **new** DruidUpdateParser();

} **else** **if**(statement **instanceof** MySqlAlterTableStatement) {

parser = **new** DruidAlterTableParser();

} **else** {

parser = **new** DefaultDruidParser();

}

**return** parser;

}

同时仿照MySqlInsertStatement、MySqlDeleteStatement等实例，自己写一个特定的类来支持该类型的解析。

## 碰到过度支持的sql语法怎么办？

**过度支持的语法**指的是不该支持但是却支持的sql语法。

如replace into …语句，该语法存在矛盾性（该语句的含义是：数据库中不存在就insert，存在就replace，即更新。根据拆分表原则：insert 时必须带拆分字段，而更新时绝不能更新拆分字段，即绝不能带拆分字段，这两种逻辑是矛盾的），不能支持。

解决方案：

在DruidMysqlRouteStrategy类中有一个方法检验不不支持的SQLStatement类型。在里面增加一个分支对不该支持的语法抛出异常。

/\*\*

\* 检验不支持的SQLStatement类型 ：不支持的类型直接抛SQLSyntaxErrorException异常

\* **@param** statement

\* **@throws** SQLSyntaxErrorException

\*/

**private** **void** checkUnSupportedStatement(SQLStatement statement) **throws** SQLSyntaxErrorException {

**if**(statement **instanceof** MySqlReplaceStatement) {

**throw** **new** SQLSyntaxErrorException(" ReplaceStatement can't be supported,use insert into ...on duplicate key update... instead ");

}

}

# Druid改写sql

## 支持insert into … values (),()…语句

**声明：**该语法支持的代码已实现，但是还没启用（代码处于注释状态，需要等druid的MySqlInsertStatement类提供setValuesList接口）。

要支持该语句需要DruidParser在statementParse的过程中将sql做拆分，根据拆分字段的值，将一个insert语句拆分成多个insert语句，然后分片发到对应的分片执行。

做法：操作MySqlInsertStatement，获取里面的valuesList，根据拆分字段计算，把一个valuesList拆分成多个valuesList（每个dataNode对应一个valuesList）。

具体见DruidInsertParser类中的parserBatchInsert方法。

## Select语句添加limit

见DruidSelectParser类中的以下方法：

/\*\*

\* 改写sql：需要加limit的加上

\*/

@Override

**public** **void** changeSql(SchemaConfig schema, RouteResultset rrs, SQLStatement stmt) **throws** SQLNonTransientException {

SQLSelectStatement selectStmt = (SQLSelectStatement)stmt;

SQLSelectQuery sqlSelectQuery = selectStmt.getSelect().getQuery();

**if**(sqlSelectQuery **instanceof** MySqlSelectQueryBlock) {

MySqlSelectQueryBlock mysqlSelectQuery = (MySqlSelectQueryBlock)selectStmt.getSelect().getQuery();

**if**(isNeedAddLimit(schema, rrs, mysqlSelectQuery) ) {

Limit limit = **new** Limit();

limit.setRowCount(**new** SQLIntegerExpr(schema.getDefaultMaxLimit()));

mysqlSelectQuery.setLimit(limit);

}

}

}

## Select语句加减order by

跟加limit类似：

mysqlSelectQuery.setOrderBy(orderBy);

要去掉order by，mysqlSelectQuery.setOrderBy(null);

说明：增加方式如上，但目前没有使用到。

## Select语句加减group by

跟加limit类似：

mysqlSelectQuery.setGroupBy(groupBy);

去掉group by，mysqlSelectQuery.setGroupBy(null);

说明：增加方式如上，但目前没有使用到。

## Insert语句加自增长主键

操作MySqlInsertStatement

insertStmt.getColumns().add(column);

insertStmt.getValues().addValue(value);

说明：增加方式如上，但目前没有使用到。

## 其他改写

其他改写还有很多，可以通过druid的api自由发挥。