芋艿v的博客

愿编码半生，如老友相伴！

微信公众号福利：芋艿的后端小屋

0. 阅读源码葵花宝典

1. RocketMQ / MyCAT / Sharding-JDBC 详细中文注释源码

2. 您对于源码的疑问每条留言都将得到认真回复

3. 新的源码解析文章实时收到通知，每周六十点更新

4. 认真的源码交流微信群

分类

Docker [2]

MyCAT [9]

Nginx [1]

RocketMQ [14]

Sharding-JDBC [17]

分享

技术杂文 [2]

Sharding-JDBC 源码分析 —— SQL 解析（二）之SQL解析

2017-07-26 更新日期:2017-07-31 总阅读量:45次

文章目录

1. 1. 概述
2. 2. SQLParsingEngine
3. 3. SQLParser SQL解析器
    3.1. 3.1 AbstractParser
    3.2. 3.2 SQLParser
        3.2.1. 3.2.1 #parseExpression() 和 SQLExpression
        3.2.2. 3.2.2 #parseAlias()
        3.2.3. 3.2.3 #parseSingleTable()
        3.2.4. 3.2.4 #skipJoin()
        3.2.5. 3.2.5 #parseWhere()
4. 4. StatementParser SQL语句解析器
    4.1. 4.1 StatementParser
    4.2. 4.2 Statement
    4.3. 4.3 预告
5. 5. 彩蛋

分享

> □□□关注**微信公众号：【芋艿的后端小屋】**有福利：
>
> 1. RocketMQ / MyCAT / Sharding-JDBC **所有**源码分析文章列表
> 2. RocketMQ / MyCAT / Sharding-JDBC **中文注释源码 GitHub 地址**
> 3. 您对于源码的疑问每条留言**都**将得到**认真**回复。**甚至不知道如何读源码也可以请教噢**。
> 4. **新的**源码解析文章**实时**收到通知。**每周更新一篇左右**。
> 5. **认真的**源码交流微信群。

- 1. 概述
- 2. SQLParsingEngine
- 3. SQLParser SQL解析器

- 3.1 AbstractParser
- 3.2 SQLParser
  - 3.2.1 #parseExpression() 和 SQLExpression
  - 3.2.2 #parseAlias()
  - 3.2.3 #parseSingleTable()
  - 3.2.4 #skipJoin()
  - 3.2.5 #parseWhere()
- 4. StatementParser SQL语句解析器
  - 4.1 StatementParser
  - 4.2 Statement
- 5. 彩蛋

---

1. 概述

上篇文章《词法解析》分享了**词法解析器Lexer**是如何解析 SQL 里的词法。本文分享**SQL解析引擎**是如何解析与理解 SQL 的。因为本文建立在《词法解析》之上，你需要阅读它后在开始这段旅程。□如果对词法解析不完全理解，请给我的公众号**（芋艿的后端小屋）**留言，我会**逐条认真耐心**回复。

区别于 Lexer，Parser **理解SQL**：

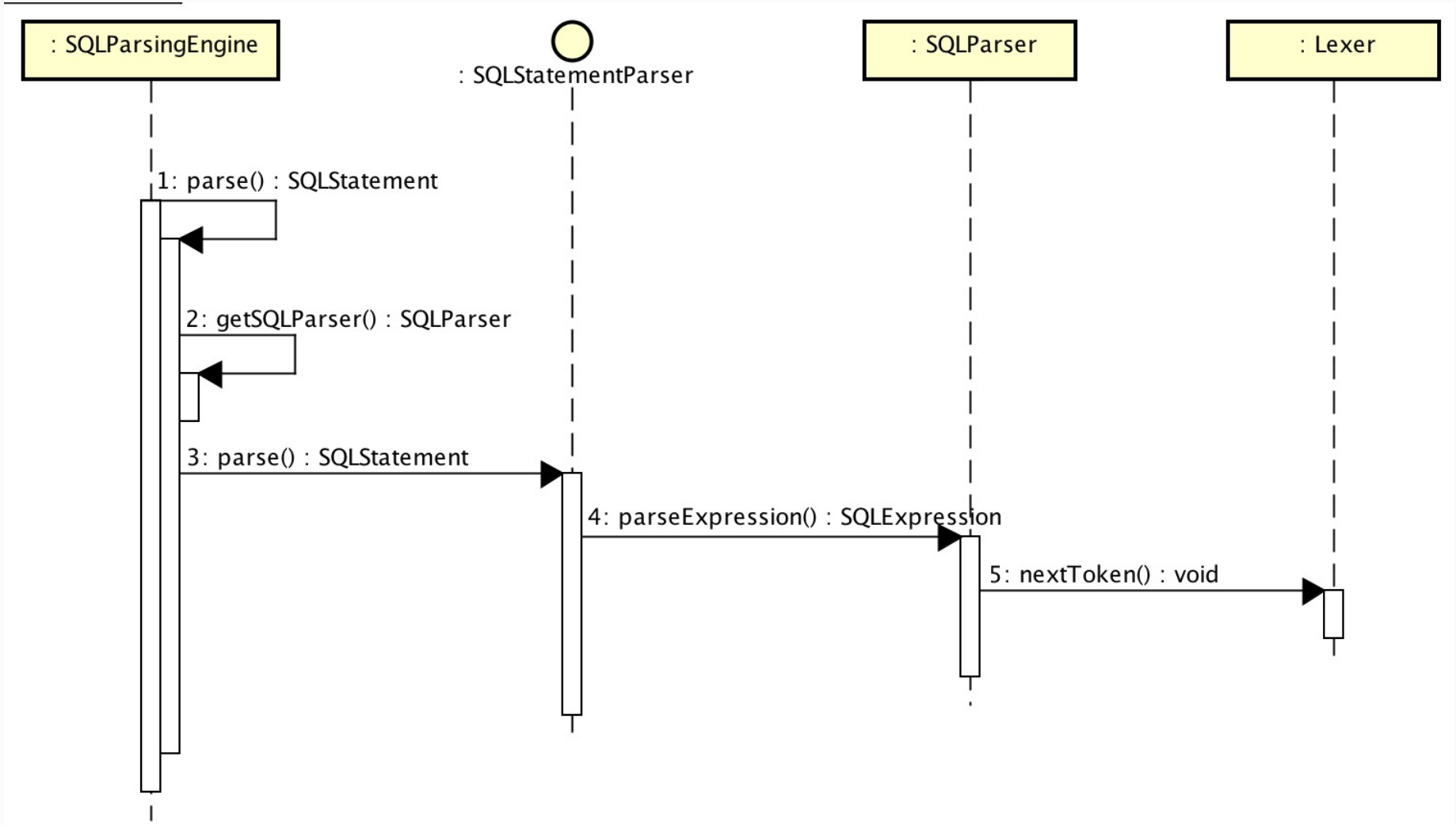
- **提炼分片上下文**

- **标记需要SQL改写的部分**

Parser 有三个组件：

- SQLParsingEngine ：SQL 解析引擎
- SQLParser ：SQL 解析器
- StatementParser ：SQL语句解析器

SQLParsingEngine 调用 StatementParser 解析 SQL。
StatementParser 调用 SQLParser 解析 SQL 表达式。
SQLParser 调用 Lexer 解析 SQL 词法。



😁 是不是觉得 SQLParser 和 StatementParser 看起来很接近？下文为你揭开这个答案。

> **Sharding-JDBC 正在收集使用公司名单：传送门。**
>
> **□ 你的登记，会让更多人参与和使用 Sharding-JDBC。传送门**
>
> **Sharding-JDBC 也会因此，能够覆盖更多的业务场景。传送门**
>
> **登记吧，骚年！传送门**

2. SQLParsingEngine

SQLParsingEngine，SQL 解析引擎。其 `#parse()` 方法作为 SQL 解析入口，本身不带复杂逻辑，通过调用 SQL 对应的 StatementParser 进行 SQL 解析。

核心代码如下：

```
// SQLParsingEngine.java
public SQLStatement parse() {
    // 获取 SQL解析器
    SQLParser sqlParser = getSQLParser();
    //
    sqlParser.skipIfEqual(Symbol.SEMI); // 跳过 ";"
    if (sqlParser.equalAny(DefaultKeyword.WITH)) { // WITH Syntax
        skipWith(sqlParser);
    }
    // 获取对应 SQL语句解析器 解析SQL
    if (sqlParser.equalAny(DefaultKeyword.SELECT)) {
        return SelectParserFactory.newInstance(sqlParser).parse();
    }
    if (sqlParser.equalAny(DefaultKeyword.INSERT)) {
        return InsertParserFactory.newInstance(shardingRule, sqlParser).parse();
    }
    if (sqlParser.equalAny(DefaultKeyword.UPDATE)) {
        return UpdateParserFactory.newInstance(sqlParser).parse();
    }
    if (sqlParser.equalAny(DefaultKeyword.DELETE)) {
        return DeleteParserFactory.newInstance(sqlParser).parse();
```

```
    }
    throw new SQLParsingUnsupportedException(sqlParser.getLexer().getCurrentToken().getType());
}
```

3. SQLParser SQL解析器

SQLParser，SQL 解析器。和词法解析器 Lexer 一样，不同数据库有不同的实现。

类图如下（**包含所有属性和方法**）（**放大图片**）：

3.1 AbstractParser

AbstractParser，SQLParser 的抽象父类，对 Lexer 简单封装。例如：

- `#skipIfEqual()`：判断当前词法标记类型是否与其中一个传入值相等
- `#equalAny()`：判断当前词法标记类型是否与其中一个传入值相等

这里有一点我们需要注意，SQLParser 并不是等 Lexer 解析完词法(Token)，再根据词法去理解 SQL。而是，在理解 SQL 的过程中，调用 Lexer 进行分词。

```
// SQLParsingEngine.java#parse()片段
if (sqlParser.equalAny(DefaultKeyword.SELECT)) {
    return SelectParserFactory.newInstance(sqlParser).parse();
}
// AbstractParser.java
public final boolean equalAny(final TokenType... tokenTypes) {
```

```
        for (TokenType each : tokenTypes) {
            if (each == lexer.getCurrentToken().getType()) {
                return true;
            }
        }
        return false;
    }
```

- ↑↑↑ 判断当前**词法**是否为 SELECT。实际 AbstractParser 只知道当前词法，并**不知道**后面还有哪些词法，也**不知道**之前有哪些词法。

我们来看 AbstractParser 里比较复杂的方法 `#skipParentheses()` 帮助大家再理解下。请认真看代码注释噢。

```
// AbstractParser.java
/**
* 跳过小括号内所有的词法标记.
*
* @return 小括号内所有的词法标记
*/
public final String skipParentheses() {
    StringBuilder result = new StringBuilder("");
    int count = 0;
    if (Symbol.LEFT_PAREN == getLexer().getCurrentToken().getType()) {
        final int beginPosition = getLexer().getCurrentToken().getEndPosition();
        result.append(Symbol.LEFT_PAREN.getLiterals());
        getLexer().nextToken();
        while (true) {
            if (equalAny(Symbol.QUESTION)) {
                increaseParametersIndex();
```

```
            }
            // 到达结尾 或者 匹配合适数的)右括号
            if (Assist.END == getLexer().getCurrentToken().getType() || (Symbol.RIGHT_PAREN == getLexer
                break;
            }
            // 处理里面有多个括号的情况，例如：SELECT COUNT(DISTINCT(order_id) FROM t_order
            if (Symbol.LEFT_PAREN == getLexer().getCurrentToken().getType()) {
                count++;
            } else if (Symbol.RIGHT_PAREN == getLexer().getCurrentToken().getType()) {
                count--;
            }
            // 下一个词法
            getLexer().nextToken();
        }
        // 获得括号内的内容
        result.append(getLexer().getInput().substring(beginPosition, getLexer().getCurrentToken().getEn
        // 下一个词法
        getLexer().nextToken();
    }
    return result.toString();
}
```

这个类其它方法很重要，逻辑相对简单，我们就不占用篇幅了。大家一定要看哟，后面调用非常非常多。

AbstractParser.java 传送门。**也可以关注我的公众号**（芋艿的后端小屋）**发送关键字【sjdbc】获取**增加方法内注释的项目地址**。

3.2 SQLParser

SQLParser，SQL 解析器，**主要提供只考虑 SQL 块的解析方法，*不考虑 SQL 上下文***。下文即将提到的 StatementParser 将 SQL 拆成对应的**块**，调用 SQLParser 进行解析。 这么说，可能会有些抽象，我们下面来一起看。

SQLParser 看起来方法特别多，合并下一共 5 种：

方法	说明
#parseExpression()	解析表达式
#parseAlias()	解析别名
#parseSingleTable()	解析单表
#skipJoin()	跳过表关联词法
#parseWhere()	解析查询条件

看了这 5 个方法是否有点理解了？SQLParser 不考虑 SQL 是 SELECT / INSERT / UPDATE / DELETE ，它考虑的是，**给我的是 WHERE 处解析查询条件，或是 INSERT INTO 解析单表 等**，提供 SELECT / INSERT / UPDATE / DELETE 需要的 SQL 块公用解析。

3.2.1 #parseExpression() 和 SQLExpression

SQLExpression，SQL表达式接口。目前 6 种实现：

类	说明	对应Token

SQLIdentifierExpression	标识表达式	Literals.IDENTIFIER
SQLPropertyExpression	属性表达式	无
SQLNumberExpression	数字表达式	Literals.INT, Literals.HEX
SQLPlaceholderExpression	占位符表达式	Symbol.QUESTION
SQLTextExpression	字符表达式	Literals.CHARS
SQLIgnoreExpression	分片中无需关注的SQL表达式	无

- SQLPropertyExpression 例如：`SELECT * FROM t_order o ORDER BY o.order_id` 中的 `o.order_id`。**SQLPropertyExpression 从 SQLIdentifierExpression 进一步判断解析而来。**

- SQLIgnoreExpression 例如：`SELECT * FROM t_order o ORDER BY o.order_id % 2` 中的`o.order_id % 2`。**复合表达式都会解析成 SQLIgnoreExpression。**

解析 SQLExpression 核心代码如下：

```
// SQLParser.java
/**
* 解析表达式.
*
* @return 表达式
```

```
     */
    // TODO 完善Expression解析的各种场景
    public final SQLExpression parseExpression() {
        // 解析表达式
        String literals = getLexer().getCurrentToken().getLiterals();
        final SQLExpression expression = getExpression(literals);
        // SQLIdentifierExpression 需要特殊处理。考虑自定义函数，表名.属性情况。
        if (skipIfEqual(Literals.IDENTIFIER)) {
            if (skipIfEqual(Symbol.DOT)) { // 例如，ORDER BY o.uid 中的 "o.uid"
                String property = getLexer().getCurrentToken().getLiterals();
                getLexer().nextToken();
                return skipIfCompositeExpression() ? new SQLIgnoreExpression() : new SQLPropertyExpression(
            }
            if (equalAny(Symbol.LEFT_PAREN)) { // 例如，GROUP BY DATE(create_time) 中的 "DATE(create_time)"
                skipParentheses();
                skipRestCompositeExpression();
                return new SQLIgnoreExpression();
            }
            return skipIfCompositeExpression() ? new SQLIgnoreExpression() : expression;
        }
        getLexer().nextToken();
        return skipIfCompositeExpression() ? new SQLIgnoreExpression() : expression;
    }
    /**
     * 获得 词法Token 对应的 SQLExpression
     *
     * @param literals 词法字面量标记
     * @return SQLExpression
     */
```

```
private SQLExpression getExpression(final String literals) {
    if (equalAny(Symbol.QUESTION)) {
        increaseParametersIndex();
        return new SQLPlaceholderExpression(getParametersIndex() - 1);
    }
    if (equalAny(Literals.CHARS)) {
        return new SQLTextExpression(literals);
    }
    // TODO 考虑long的情况
    if (equalAny(Literals.INT)) {
        return new SQLNumberExpression(Integer.parseInt(literals));
    }
    if (equalAny(Literals.FLOAT)) {
        return new SQLNumberExpression(Double.parseDouble(literals));
    }
    // TODO 考虑long的情况
    if (equalAny(Literals.HEX)) {
        return new SQLNumberExpression(Integer.parseInt(literals, 16));
    }
    if (equalAny(Literals.IDENTIFIER)) {
        return new SQLIdentifierExpression(SQLUtil.getExactlyValue(literals));
    }
    return new SQLIgnoreExpression();
}
/**
* 如果是 复合表达式，跳过。
*
* @return 是否跳过
*/
```

```
private boolean skipIfCompositeExpression() {
    if (equalAny(Symbol.PLUS, Symbol.SUB, Symbol.STAR, Symbol.SLASH, Symbol.PERCENT, Symbol.AMP, Symbol
        skipParentheses();
        skipRestCompositeExpression();
        return true;
    }
    return false;
}
/**
* 跳过剩余复合表达式
*/
private void skipRestCompositeExpression() {
    while (skipIfEqual(Symbol.PLUS, Symbol.SUB, Symbol.STAR, Symbol.SLASH, Symbol.PERCENT, Symbol.AMP,
        if (equalAny(Symbol.QUESTION)) {
            increaseParametersIndex();
        }
        getLexer().nextToken();
        skipParentheses();
    }
}
```

解析了 SQLExpression 有什么用呢？我们会在《查询SQL解析》、《插入SQL解析》、《更新SQL解析》、《删除SQL解析》。留个悬念😺，关注我的公众号**（芋艿的后端小屋）**，**实时收到新文更新通知**。

3.2.2 #parseAlias()

```
/**
```

```
     * 解析别名.不仅仅是字段的别名，也可以是表的别名。
     *
     * @return 别名
     */
    public Optional<String> parseAlias() {
        // 解析带 AS 情况
        if (skipIfEqual(DefaultKeyword.AS)) {
            if (equalAny(Symbol.values())) {
                return Optional.absent();
            }
            String result = SQLUtil.getExactlyValue(getLexer().getCurrentToken().getLiterals());
            getLexer().nextToken();
            return Optional.of(result);
        }
        // 解析别名
        // TODO 增加哪些数据库识别哪些关键字作为别名的配置
        if (equalAny(Literals.IDENTIFIER, Literals.CHARS, DefaultKeyword.USER, DefaultKeyword.END, DefaultK
            String result = SQLUtil.getExactlyValue(getLexer().getCurrentToken().getLiterals());
            getLexer().nextToken();
            return Optional.of(result);
        }
        return Optional.absent();
    }
```

3.2.3 #parseSingleTable()

```
    /**
```

```
     * 解析单表.
     *
     * @param sqlStatement SQL语句对象
     */
    public final void parseSingleTable(final SQLStatement sqlStatement) {
        boolean hasParentheses = false;
        if (skipIfEqual(Symbol.LEFT_PAREN)) {
            if (equalAny(DefaultKeyword.SELECT)) { // multiple-update 或者 multiple-delete
                throw new UnsupportedOperationException("Cannot support subquery");
            }
            hasParentheses = true;
        }
        Table table;
        final int beginPosition = getLexer().getCurrentToken().getEndPosition() - getLexer().getCurrentToke
        String literals = getLexer().getCurrentToken().getLiterals();
        getLexer().nextToken();
        if (skipIfEqual(Symbol.DOT)) {
            getLexer().nextToken();
            if (hasParentheses) {
                accept(Symbol.RIGHT_PAREN);
            }
            table = new Table(SQLUtil.getExactlyValue(literals), parseAlias());
        } else {
            if (hasParentheses) {
                accept(Symbol.RIGHT_PAREN);
            }
            table = new Table(SQLUtil.getExactlyValue(literals), parseAlias());
        }
        if (skipJoin()) { // multiple-update 或者 multiple-delete
```

分享

```
            throw new UnsupportedOperationException("Cannot support Multiple-Table.");
        }
        sqlStatement.getSqlTokens().add(new TableToken(beginPosition, literals));
        sqlStatement.getTables().add(table);
    }
```

3.2.4 #skipJoin()

跳过表关联词法，支持 `SELECT * FROM t_user, t_order WHERE ...`，`SELECT * FROM t_user JOIN t_order ON ...`。下篇《查询SQL解析》**解析表**会用到这个方法。

```
// SQLParser.java
/**
* 跳过表关联词法.
*
* @return 是否表关联.
*/
public final boolean skipJoin() {
    if (skipIfEqual(DefaultKeyword.LEFT, DefaultKeyword.RIGHT, DefaultKeyword.FULL)) {
        skipIfEqual(DefaultKeyword.OUTER);
        accept(DefaultKeyword.JOIN);
        return true;
    } else if (skipIfEqual(DefaultKeyword.INNER)) {
        accept(DefaultKeyword.JOIN);
        return true;
    } else if (skipIfEqual(DefaultKeyword.JOIN, Symbol.COMMA, DefaultKeyword.STRAIGHT_JOIN)) {
        return true;
```

```
    } else if (skipIfEqual(DefaultKeyword.CROSS)) {
        if (skipIfEqual(DefaultKeyword.JOIN, DefaultKeyword.APPLY)) {
            return true;
        }
    } else if (skipIfEqual(DefaultKeyword.OUTER)) {
        if (skipIfEqual(DefaultKeyword.APPLY)) {
            return true;
        }
    }
    return false;
}
```

3.2.5 #parseWhere()

解析 WHERE 查询条件。目前支持 AND 条件，不支持 OR 条件。近期 OR 条件支持的可能性比较低。另外条件这块对括号解析需要继续优化，实际使用请勿写冗余的括号。例
如：`SELECT * FROM tbl_name1 WHERE ((val1=?) AND (val2=?)) AND val3 =?`。

根据不同的运算操作符，分成如下情况：

运算符	附加条件	方法
=		#parseEqualCondition()
IN		#parseInCondition()
BETWEEN		#parseBetweenCondition()

运算符	附加条件	方法
<, <=, >, >=	Oracle 或 SQLServer 分页	#parseRowNumberCondition()
<, <=, >, >=		#parseOtherCondition()
LIKE		parseOtherCondition

代码如下：

```
// SQLParser.java
/**
* 解析所有查询条件。
* 目前不支持 OR 条件。
*
* @param sqlStatement SQL
*/
private void parseConditions(final SQLStatement sqlStatement) {
    // AND 查询
    do {
        parseComparisonCondition(sqlStatement);
    } while (skipIfEqual(DefaultKeyword.AND));
    // 目前不支持 OR 条件
    if (equalAny(DefaultKeyword.OR)) {
        throw new SQLParsingUnsupportedException(getLexer().getCurrentToken().getType());
    }
}
// TODO 解析组合expr
/**
```

```
     * 解析单个查询条件
     *
     * @param sqlStatement SQL
     */
    public final void parseComparisonCondition(final SQLStatement sqlStatement) {
        skipIfEqual(Symbol.LEFT_PAREN);
        SQLExpression left = parseExpression(sqlStatement);
        if (equalAny(Symbol.EQ)) {
            parseEqualCondition(sqlStatement, left);
            skipIfEqual(Symbol.RIGHT_PAREN);
            return;
        }
        if (equalAny(DefaultKeyword.IN)) {
            parseInCondition(sqlStatement, left);
            skipIfEqual(Symbol.RIGHT_PAREN);
            return;
        }
        if (equalAny(DefaultKeyword.BETWEEN)) {
            parseBetweenCondition(sqlStatement, left);
            skipIfEqual(Symbol.RIGHT_PAREN);
            return;
        }
        if (equalAny(Symbol.LT, Symbol.GT, Symbol.LT_EQ, Symbol.GT_EQ)) {
            if (left instanceof SQLIdentifierExpression && sqlStatement instanceof SelectStatement
                    && isRowNumberCondition((SelectStatement) sqlStatement, ((SQLIdentifierExpression) left
                parseRowNumberCondition((SelectStatement) sqlStatement);
            } else if (left instanceof SQLPropertyExpression && sqlStatement instanceof SelectStatement
                    && isRowNumberCondition((SelectStatement) sqlStatement, ((SQLPropertyExpression) left).
                parseRowNumberCondition((SelectStatement) sqlStatement);
```

```
        } else {
            parseOtherCondition(sqlStatement);
        }
    } else if (equalAny(DefaultKeyword.LIKE)) {
        parseOtherCondition(sqlStatement);
    }
    skipIfEqual(Symbol.RIGHT_PAREN);
}
```

`#parseComparisonCondition()` 解析到 `左SQL表达式(left)` 和 运算符，调用相应方法进一步处理。我们选择 `#parseEqualCondition()` 看下，其他方法有兴趣跳转 SQLParser 查看。

```
// SQLParser.java
/**
* 解析 = 条件
*
* @param sqlStatement SQL
* @param left 左SQLExpression
*/
private void parseEqualCondition(final SQLStatement sqlStatement, final SQLExpression left) {
    getLexer().nextToken();
    SQLExpression right = parseExpression(sqlStatement);
    // 添加列
    // TODO 如果有多表,且找不到column是哪个表的,则不加入condition,以后需要解析binding table
    if ((sqlStatement.getTables().isSingleTable() || left instanceof SQLPropertyExpression)
            // 只有对路由结果有影响的才会添加到 conditions。SQLPropertyExpression 和 SQLIdentifierExpressi
            && (right instanceof SQLNumberExpression || right instanceof SQLTextExpression || right ins
        Optional<Column> column = find(sqlStatement.getTables(), left);
```

```
        if (column.isPresent()) {
            sqlStatement.getConditions().add(new Condition(column.get(), right), shardingRule);
        }
    }
}
```

`#parseEqualCondition()` 解析到 `右SQL表达式(right)`，并判断 `左右SQL表达式` 与路由逻辑是否有影响，如果有，则加入到 Condition。**这个就是 `#parseWhere()` 的目的：解析 WHERE 查询条件对路由有影响的条件。**《路由》相关的逻辑，会单独开文章介绍。这里，我们先留有映像。

4. StatementParser SQL语句解析器

4.1 StatementParser

StatementParser，SQL语句解析器。每种 SQL，都有相应的 SQL语句解析器实现。不同数据库，继承这些 SQL语句解析器，实现各自 SQL 上的差异。大体结构如下：

SQLParsingEngine 根据不同 SQL 调用对应工厂创建 StatementParser。核心代码如下：

```
public final class SelectParserFactory {

    /**
     * 创建Select语句解析器.
     *
     * @param sqlParser SQL解析器
```

```
     * @return Select语句解析器
     */
    public static AbstractSelectParser newInstance(final SQLParser sqlParser) {
        if (sqlParser instanceof MySQLParser) {
            return new MySQLSelectParser(sqlParser);
        }
        if (sqlParser instanceof OracleParser) {
            return new OracleSelectParser(sqlParser);
        }
        if (sqlParser instanceof SQLServerParser) {
            return new SQLServerSelectParser(sqlParser);
        }
        if (sqlParser instanceof PostgreSQLParser) {
            return new PostgreSQLSelectParser(sqlParser);
        }
        throw new UnsupportedOperationException(String.format("Cannot support sqlParser class [%s].",
    }
}
```

调用 `StatementParser#parse()` 实现方法，对 SQL 进行解析。具体解析过程，另开文章分享。

4.2 Statement

不同 SQL 解析后，返回对应的 SQL 结果,即 Statement。大体结构如下：

Statement 包含两部分信息：

- 分片上下文：用于 SQL 路由。

- SQL 标记对象：用于 SQL 改写。

我们会在后文增删改查SQL解析的过程中分享到它们。

4.3 预告

Parser	Statement	分享文章
SelectStatementParser	SelectStatement + AbstractSQLStatement	《查询SQL解析》
InsertStatementParser	InsertStatement	《插入SQL解析》
UpdateStatementParser	UpdateStatement	《更新SQL解析》
DeleteStatementParser	DeleteStatement	《删除SQL解析》

5. 彩蛋

老铁，是不是有丢丢长？
如果有地方错误，烦请指出□。

如果有地方不是很理解，可以加我的公众号**（芋艿的后端小屋）**留言，我会**逐条认真耐心**回复。
如果觉得还凑合，劳驾分享朋友圈或者基佬。

《查询SQL解析》已经写了一半，预计很快...

分享

Sharding-JDBC

PREVIOUS:
« Sharding-JDBC 源码分析 —— SQL 解析（三）之查询SQL
NEXT:
» Sharding-JDBC 源码分析 —— SQL 解析（一）之词法解析