

# 芋艿v的博客

愿编码半生，如老友相伴！



分享

# 扫一扫二维码关注公众号

关注后，可以看到

「RocketMQ」

「MyCAT」

所有源码解析文章

—— 近期更新「Sharding-JDBC」中 ——

你有233个小伙伴已经关注

分享

## 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<sup>2</sup>
- MyCAT<sup>9</sup>
- Nginx<sup>1</sup>
- RocketMQ<sup>14</sup>
- Sharding-JDBC<sup>17</sup>
- 技术杂文<sup>2</sup>

分享

# Sharding-JDBC 源码分析 —— SQL 解析（五）之更新SQL

🕒2017-07-31 更新日期:2017-07-31 总阅读量:7次

## 文章目录

1. 1. 概述
2. 2. UpdateStatement
3. 3. #parse()
  - 3.1. 3.1 #skipBetweenUpdateAndTable()
  - 3.2. 3.2 #parseSingleTable()
  - 3.3. 3.3 #parseSetItems()
  - 3.4. 3.4 #parseWhere()
4. 666. 彩蛋



扫一扫二维码关注公众号

关注后，可以看到

「RocketMQ」

「MyCAT」

所有源码解析文章

— 近期更新「Sharding-JDBC」中 —

你有233个小伙伴已经关注

□□□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

分享

- 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. UpdateStatement](#)
- [3. #parse\(\)](#)
  - [3.1 #skipBetweenUpdateAndTable\(\)](#)
  - [3.2 #parseSingleTable\(\)](#)
  - [3.3 #parseSetItems\(\)](#)
  - [3.4 #parseWhere\(\)](#)
- [666. 彩蛋](#)

## 1. 概述

本文前置阅读：

- [《SQL 解析（一）之词法解析》](#)
- [《SQL 解析（二）之SQL解析》](#)

本文分享**更新SQL解析**的源码实现。

更新SQL解析比查询SQL解析复杂度低的多的多。不同数据库在插入SQL语法上也统一的多。本文分享 MySQL 更新SQL解析器 **MySQLUpdateParser**。

MySQL UPDATE 语法一共有 2 种：

- 第一种：**Single-table syntax**

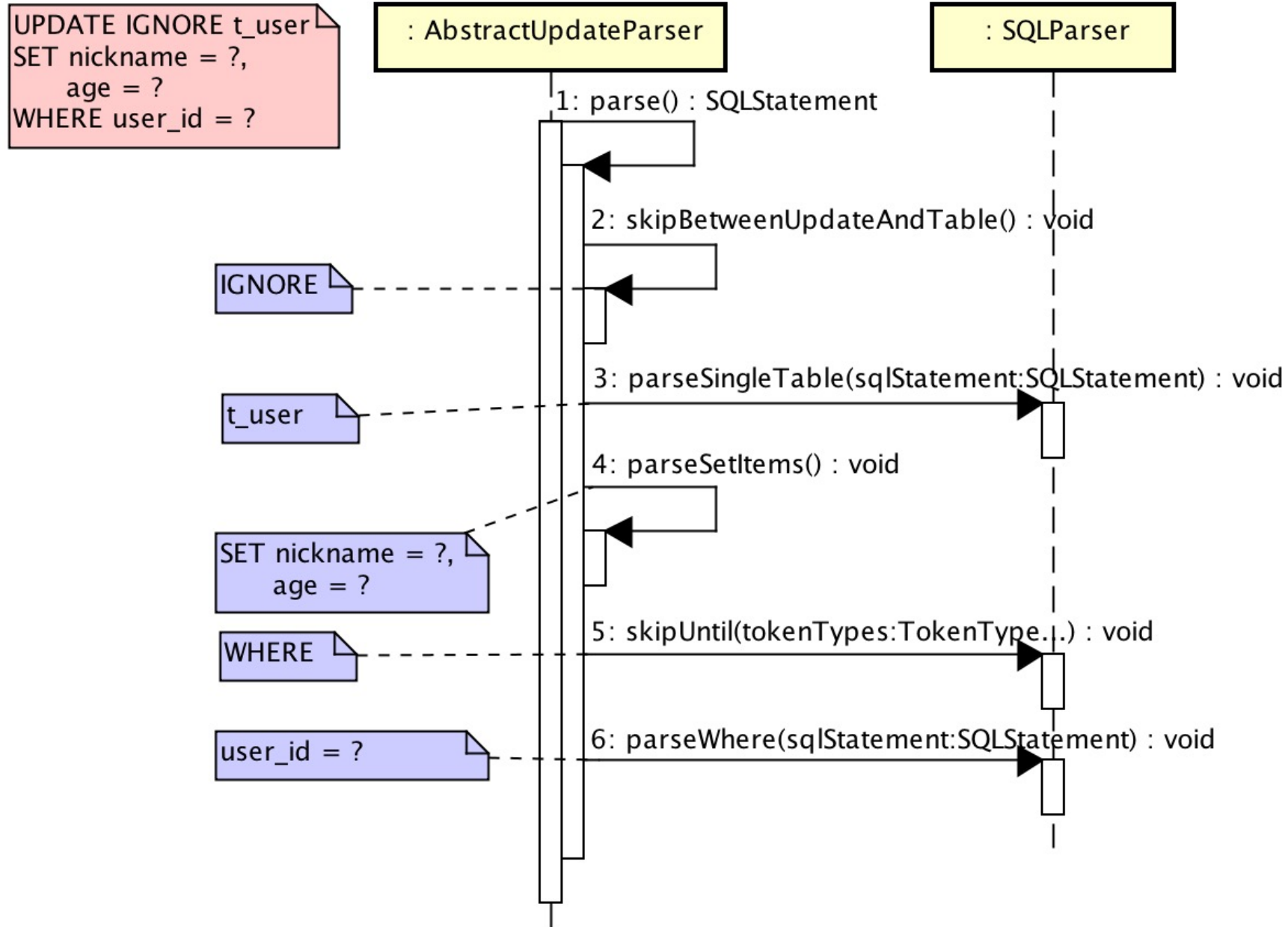
```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
  SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
```

- 第二种：**Multiple-table syntax**

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
  SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
  [WHERE where_condition]
```

Sharding-JDBC 目前仅支持第一种。业务场景上使用第二种的很少很少。

Sharding-JDBC 更新SQL解析主流程如下：



```
// AbstractUpdateParser.java
```

```
@Override
public UpdateStatement parse() {
    sqlParser.getLexer().nextToken(); // 跳过 UPDATE
    skipBetweenUpdateAndTable(); // 跳过关字，例如：MYSQL 里的 LOW_PRIORITY、IGNORE
    sqlParser.parseSingleTable(updateStatement); // 解析表
    parseSetItems(); // 解析 SET
    sqlParser.skipUntil(DefaultKeyword.WHERE);
    sqlParser.setParametersIndex(parametersIndex);
    sqlParser.parseWhere(updateStatement);
    return updateStatement; // 解析 WHERE
}
```

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. UpdateStatement

更新SQL 解析结果。

```
public final class UpdateStatement extends AbstractSQLStatement {
}
```

🐱 对，没有其他属性。

我们来看下 `UPDATE t_user SET nickname = ?, age = ? WHERE user_id = ?` 的解析结果：



```

updateStatement = {com.dangdang.ddframe.rdb.sharding.parsing.parser.statement.update.UpdateStatement@15
  ▶ f type = {com.dangdang.ddframe.rdb.sharding.constant.SQLType@1527} "UPDATE"
  ▼ f tables = {com.dangdang.ddframe.rdb.sharding.parsing.parser.context.table.Tables@1515} "Tables(tables=[Tab
    ▼ f tables = {java.util.ArrayList@1533} size = 1
      ▼ 0 = {com.dangdang.ddframe.rdb.sharding.parsing.parser.context.table.Table@1535} "Table(name=t_user
        ▶ f name = "t_user"
        ▶ f alias = {com.google.common.base.Absent@1538} "Optional.absent()"
      ▼ f conditions = {com.dangdang.ddframe.rdb.sharding.parsing.parser.context.condition.Conditions@1528} "Conc
        ▼ f conditions = {java.util.LinkedHashMap@1540} size = 1
          ▼ 0 = {java.util.LinkedHashMap$Entry@1543} "Column(name=user_id, tableName=t_user)" -> "Condition(c
            ▼ 0 = {com.dangdang.ddframe.rdb.sharding.parsing.parser.context.condition.Column@1544} "Colum
              ▶ f name = "user_id"
              ▶ f tableName = "t_user"
            ▼ value = {com.dangdang.ddframe.rdb.sharding.parsing.parser.context.condition.Condition@1545} "C
              ▼ f column = {com.dangdang.ddframe.rdb.sharding.parsing.parser.context.condition.Column@1544}
                ▶ f name = "user_id"
                ▶ f tableName = "t_user"
              ▶ f operator = {com.dangdang.ddframe.rdb.sharding.constant.ShardingOperator@1551} "EQUAL"
              ▶ f positionValueMap = {java.util.LinkedHashMap@1552} size = 0
              ▶ f positionIndexMap = {java.util.LinkedHashMap@1553} size = 1
          ▼ f sqlTokens = {java.util.LinkedList@1529} size = 1
            ▼ 0 = {com.dangdang.ddframe.rdb.sharding.parsing.parser.token.TableToken@1557} "TableToken(beginPosi
              ▶ f beginPosition = 7
              ▶ f originalLiterals = "t_user"

```

### 3. #parse()

#### 3.1 #skipBetweenUpdateAndTable()

在 `UPDATE` 和 表名 之间有些词法，对 SQL 路由和改写无影响，进行跳过。

```
// MySQLUpdateParser.java
@Override
protected void skipBetweenUpdateAndTable() {
    getSqlParser().skipAll(MySQLKeyword.LOW_PRIORITY, MySQLKeyword.IGNORE);
}

// OracleUpdateParser.java
@Override
protected void skipBetweenUpdateAndTable() {
    getSqlParser().skipIfEqual(OracleKeyword.ONLY);
}
```

## 3.2 #parseSingleTable()

解析表，请看《SQL 解析（二）之SQL解析》的 `#parseSingleTable()` 小节。

## 3.3 #parseSetItems()

解析 `SET` 后语句。

```
// AbstractUpdateParser.java
/**
 * 解析多个 SET 项
 */
private void parseSetItems() {
    sqlParser.accept(DefaultKeyword.SET);
}
```

```
do {
    parseSetItem();
} while (sqlParser.skipIfEqual(Symbol.COMMA)); // 以 "," 分隔
}

/**
 * 解析单个 SET 项
 */
private void parseSetItem() {
    parseSetColumn();
    sqlParser.skipIfEqual(Symbol.EQ, Symbol.COLON_EQ);
    parseSetValue();
}

/**
 * 解析单个 SET 项
 */
private void parseSetColumn() {
    if (sqlParser.equalAny(Symbol.LEFT_PAREN)) {
        sqlParser.skipParentheses();
        return;
    }
    int beginPosition = sqlParser.getLexer().getCurrentToken().getEndPosition();
    String literals = sqlParser.getLexer().getCurrentToken().getLiterals();
    sqlParser.getLexer().nextToken();
    if (sqlParser.skipIfEqual(Symbol.DOT)) { // 字段有别名
        // TableToken
        if (updateStatement.getTables().getSingleTableName().equalsIgnoreCase(SQLUtil.getExactlyValue(literals))) {
            updateStatement.getSqlTokens().add(new TableToken(beginPosition - literals.length(), literals));
        }
        sqlParser.getLexer().nextToken();
    }
}
```

分享

```
    }  
}  
/**  
 * 解析单个 SET 值  
 */  
private void parseSetValue() {  
    sqlParser.parseExpression(updateStatement);  
    parametersIndex = sqlParser.getParametersIndex();  
}
```

## 3.4 #parseWhere()

解析 WHERE 条件。解析代码：[《SQL 解析（二）之SQL解析》的#parseWhere\(\)小节](#)。

## 666. 彩蛋

😁比更新SQL解析是不是简单，更不用对比查询SQL解析。😁有一种在水更的感觉。嘿嘿，下一篇（[《删除SQL解析》](#)）会更加容易。

道友，帮我分享一波怎么样？

📁 [Sharding-JDBC](#)



分享

PREVIOUS:

« [Sharding-JDBC 源码分析 —— SQL 解析（六）之删除SQL](#)

NEXT:

» [Sharding-JDBC 源码分析 —— SQL 解析（四）之插入SQL](#)

© 2017 [王文斌](#) && 总访客数 769 次 && 总访问量 2220 次 && Hosted by [Coding Pages](#) && Powered by [hexo](#) && Theme by [coney](#)

分享