课堂主题

Sharding JDBC架构和核心概念、Sharding JDBC安装和核心组件、Sharding JDBC分片策略和读写分离

课堂目标

理解Sharding JDBC架构和核心概念（数据分片、SQL、分片策略、分片算法、配置）

能够在项目中引入Sharding JDBC开源组件

理解Sharding JDBC核心组件（解析引擎、路由引擎、改写引擎、执行引擎、归并引擎）

掌握Sharding JDBC分片策略

掌握Sharding JDBC读写分离

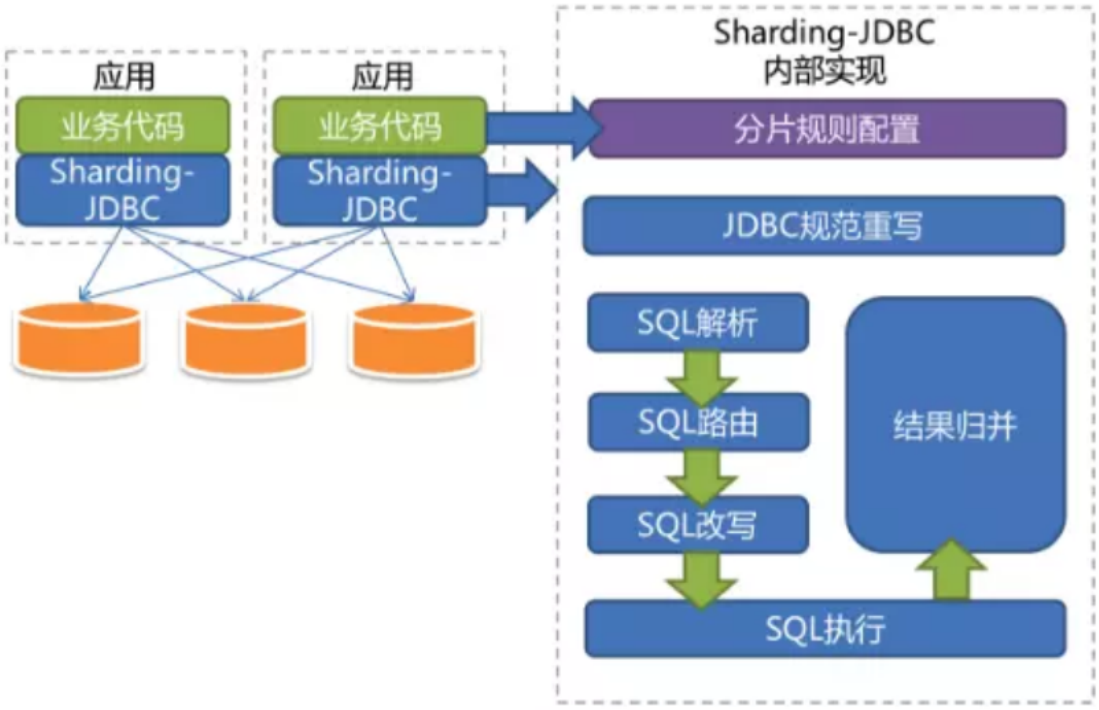
能够在项目中配置数据分片、读写分离、广播表、绑定表

什么是Sharding JDBC

官方网站：http://shardingsphere.apache.org/index_zh.html

Apache ShardingSphere(Incubator) 是一套开源的分布式数据库中间件解决方案组成的生态圈，它由Sharding-JDBC、Sharding-Proxy和Sharding-Sidecar（规dd划中）这3款相互独立，却又能够混合部署配合使用的产品组成。

Sharding JDBC架构



Sharding JDBC核心概念

数据分片

数据分片分为垂直分片和水平分片。

SQL

逻辑表

真实表

数据节点

绑定表

广播表

分片策略

包含分片键和分片算法。分片键是用于分片的数据库字段，是将数据库(表)水平拆分的关键字段。

分片算法

精确分片算法、范围分片算法、复合分片算法、Hint分片算法

分片策略

标准分片策略、复合分片策略、行表达式分片策略、Hint分片策略

配置

分片规则：分片规则配置的总入口。包含数据源配置、表配置、绑定表配置以及读写分离配置等。

Sharding JDBC对多数据库的支持

Sharding JDBC安装

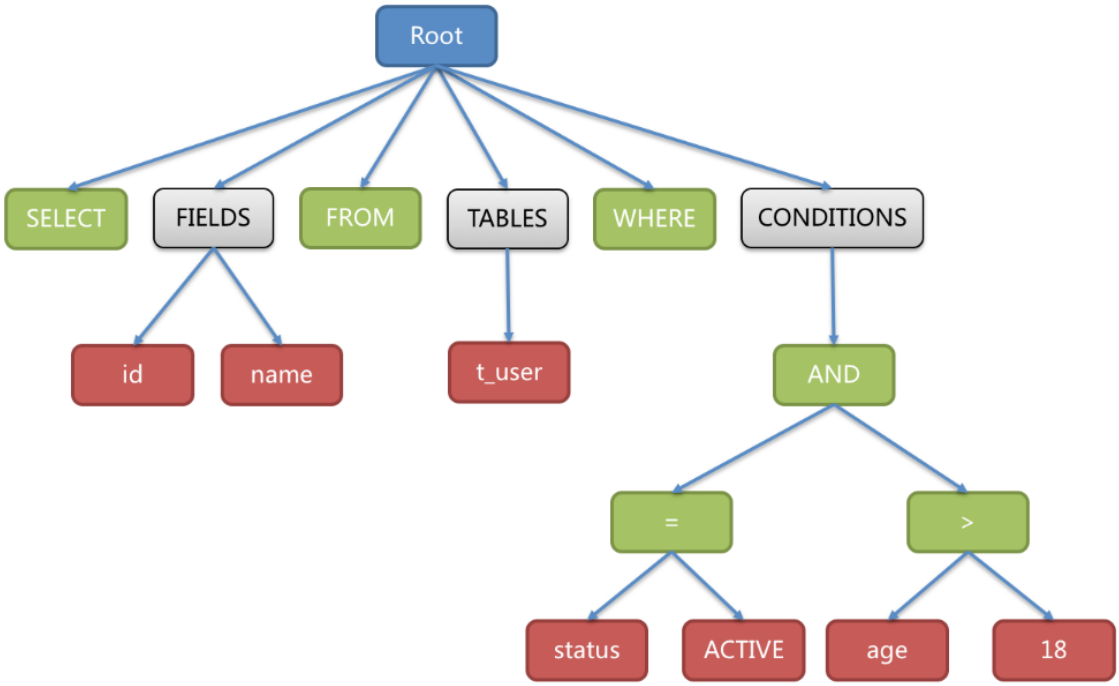
引入Maven依赖

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>sharding-jdbc-core</artifactId>
    <version>3.0.0</version>
</dependency>
```

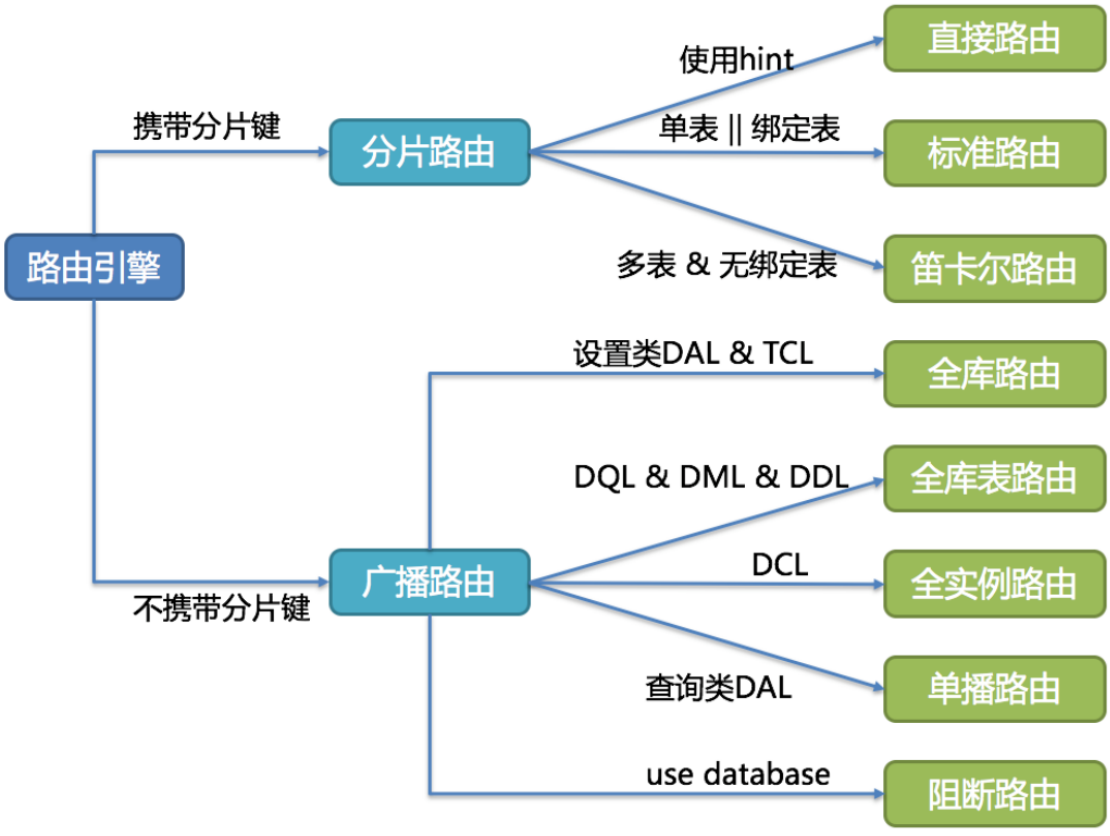
调用API编程实现，最新的是shardingJDBC4.0RC

Sharding JDBC核心组件

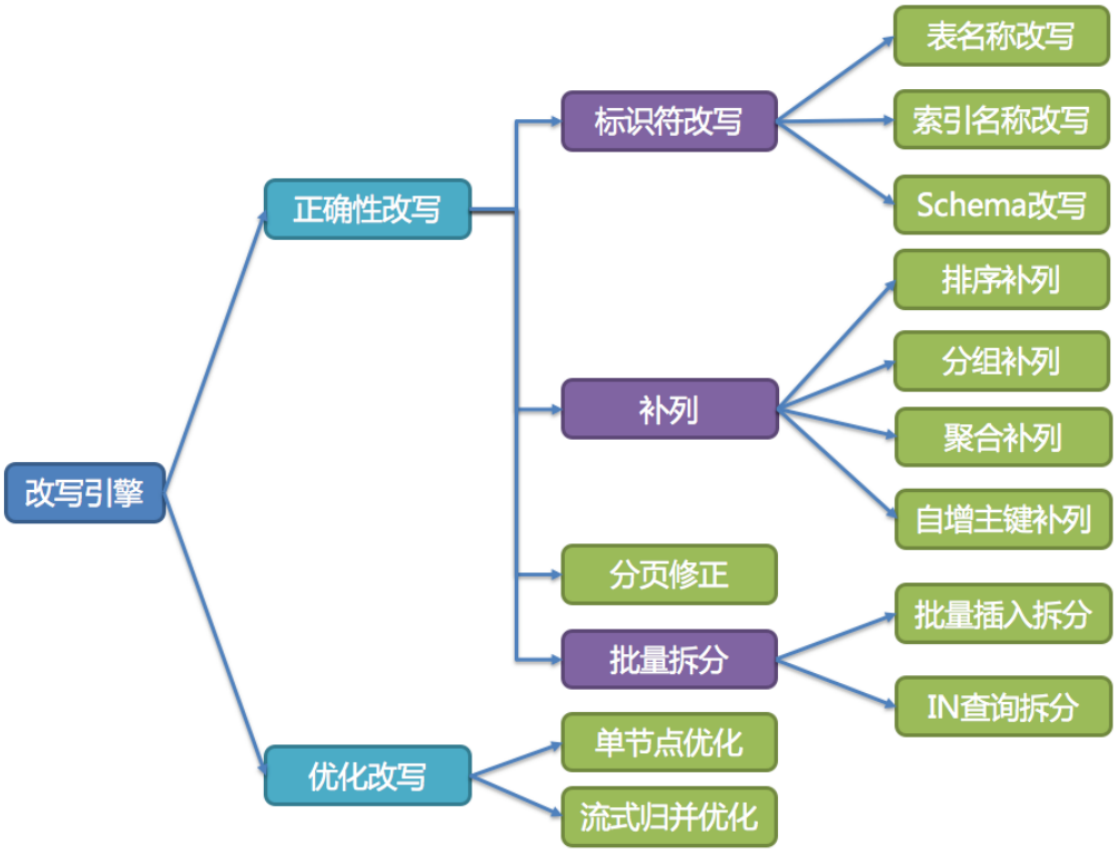
解析引擎



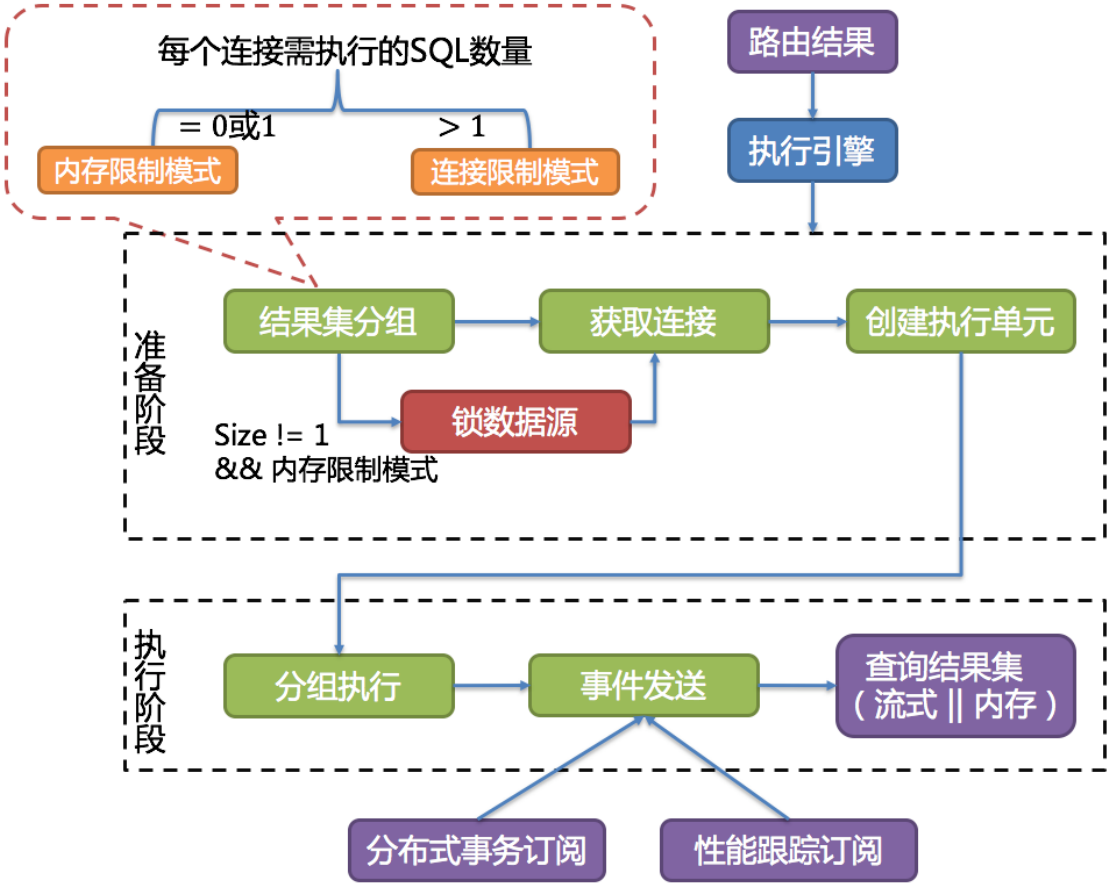
路由引擎



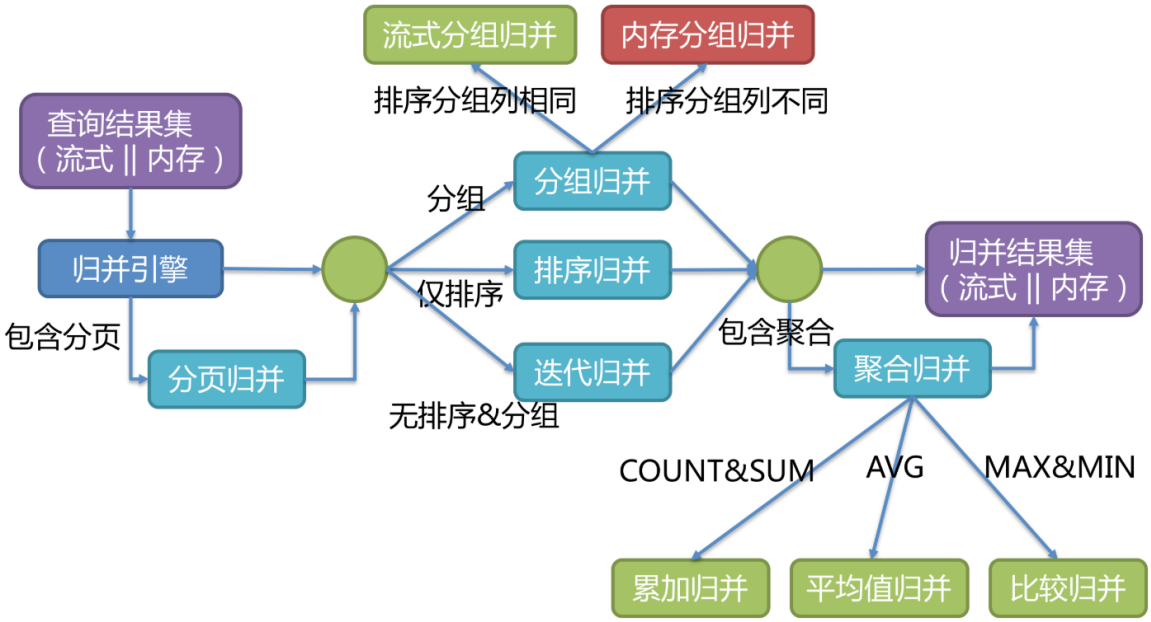
改写引擎


改写引擎
正确性改写
标识符改写
表名称改写
索引名称改写
Schema改写
补列
排序补列
分组补列
聚合补列
自增主键补列
分页修正
批量拆分
批量插入拆分
IN查询拆分
优化改写
单节点优化
流式归并优化

执行引擎


每个连接需执行的SQL数量
= 0或1
> 1
内存限制模式
连接限制模式
路由结果
执行引擎
准备阶段
结果集分组
获取连接
创建执行单元
锁数据源
Size != 1
&& 内存限制模式
执行阶段
分组执行
事件发送
查询结果集
(流式 || 内存)
分布式事务订阅
性能跟踪订阅

归并引擎



测试Demo

Java调用sharding JDBC

1、pom.xml

```
        <dependency>
            <groupId>io.shardingsphere</groupId>
            <artifactId>sharding-jdbc-core</artifactId>
            <version>3.0.0</version>
        </dependency>

        <!-- mysql 数据库驱动. -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!-- 数据源 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.26</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.6</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.6</version>
        </dependency>
```

2、sharding.java

```
Map<String, DataSource> map=new HashMap<>();
        map.put("kkb_ds_0",
createDataSource("root","root","jdbc:mysql://192.168.24.128:3306/kkb_ds_0"));
        map.put("kkb_ds_1",
createDataSource("root","root","jdbc:mysql://192.168.24.128:3306/kkb_ds_1"));

        ShardingRuleConfiguration config=new ShardingRuleConfiguration();

        // 配置Order表规则
        TableRuleConfiguration orderTableRuleConfig = new
TableRuleConfiguration();
        orderTableRuleConfig.setLogicTable("t_order");//设置逻辑表.

orderTableRuleConfig.setActualDataNodes("kkb_ds_${0..1}.t_order_${0..1}");//设置
实际数据节点.
        orderTableRuleConfig.setKeyGeneratorColumnName("oid");//设置主键列名称.

        // 配置Order表规则：配置分库 + 分表策略(这个也可以在ShardingRuleConfiguration进
行统一设置)
        orderTableRuleConfig.setDatabaseShardingStrategyConfig(new
InlineShardingStrategyConfiguration("uid", "kkb_ds_${uid % 2}"));
        orderTableRuleConfig.setTableShardingStrategyConfig(new
InlineShardingStrategyConfiguration("oid", "t_order_${oid % 2}"));
        config.getTableRuleConfigs().add(orderTableRuleConfig);

        try {
            DataSource ds=ShardingDataSourceFactory.createDataSource(map,
config, new HashMap(), new Properties());

            for(int i=1;i<=10;i++) {
                String sql="insert into t_order(uid,name) values(?,?)";
                execute(ds,sql,i,i+"aaa");
            }
            System.out.println("数据插入完成。。。");

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```