MySQL分库分表篇

一、为什么要分库分表

关系型数据库以MySQL为例，单机的存储能力、连接数是有限的，它自身就很容易会成为系统的瓶颈。当单表数据量在百万以里时，我们还可以通过添加从库、优化索引提升性能。一旦数据量朝着千万以上趋势增长，再怎么优化数据库，很多操作性能仍下降严重。为了减少数据库的负担，提升数据库响应速度，缩短查询时间，这时候就需要进行`分库分表`。

二、如何分库分表

分库分表就是要将大量数据分散到多个数据库中，使每个数据库中数据量小响应速度快，以此来提升数据库整体性能。核心理念就是对数据进行切分（`Sharding`），以及切分后如何对数据的快速定位与整合。

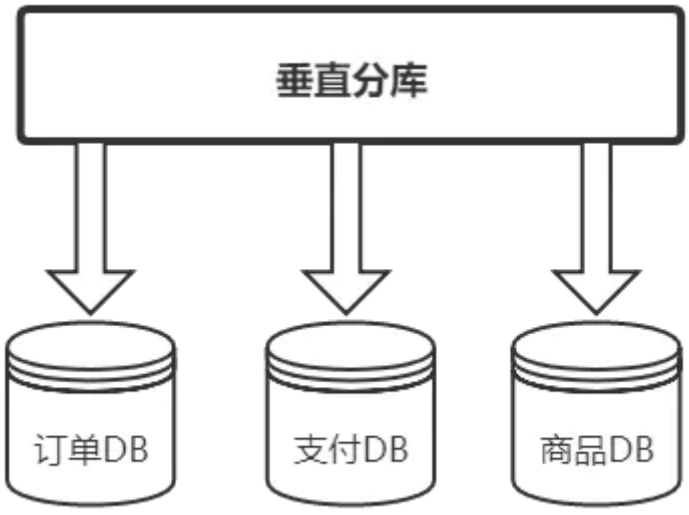
针对数据切分类型，大致可以分为：垂直（纵向）切分和水平（横向）切分两种。

2.1 垂直切分

垂直切分又细分为`垂直分库`和`垂直分表`

垂直分库

垂直分库是基于业务分类的，和我们常听到的微服务治理观念很相似，每一个独立的服务都拥有自己的数据库，需要不同业务的数据需接口调用。而垂直分库也是按照业务分类进行划分，每个业务有独立数据库，这个比较好理解。



垂直分表

`垂直分表`是基于数据表的列为依据切分的，是一种大表拆小表的模式。

例如：一个`order`表有很多字段，把长度较大且访问不频繁的字段，拆分出来创建一个单独的扩展表`work_extend`进行存储。

`order`表：

id	workNo	price	describe	.....
int（12)	int（2)	int（15)	varchar（2000)	

拆分后

`order`核心表：

id	workNo	price	.....
int（12)	int（2)	int（15)	

`work_extend`表：

id	workNo	describe	.....
int（12)	int（2)	varchar（2000)	

数据库是以行为单位将数据加载到内存中，这样拆分以后核心表大多是访问频率较高的字段，而且字段

长度也都较短，可以加载更多数据到内存中，增加查询的命中率，减少磁盘IO，以此来提升数据库性能。

优点：

- 业务间解耦，不同业务的数据进行独立的维护、监控、扩展
- 在高并发场景下，一定程度上缓解了数据库的压力

缺点：

- 提升了开发的复杂度，由于业务的隔离性，很多表无法直接访问，必须通过接口方式聚合数据，
- 分布式事务管理难度增加
- 数据库还是存在单表数据量过大的问题，并未根本上解决，需要配合水平切分

2.2 水平切分

前边说了垂直切分还是会存在单表数据量过大的问题，当我们的应用已经无法在细粒度的垂直切分时，依旧存在单库读写、存储性能瓶颈，这时就要配合水平切分一起了。

水平切分将一张大数据量的表，切分成多个表结构相同，而每个表只占原表一部分数据，然后按不同的条件分散到多个数据库中。

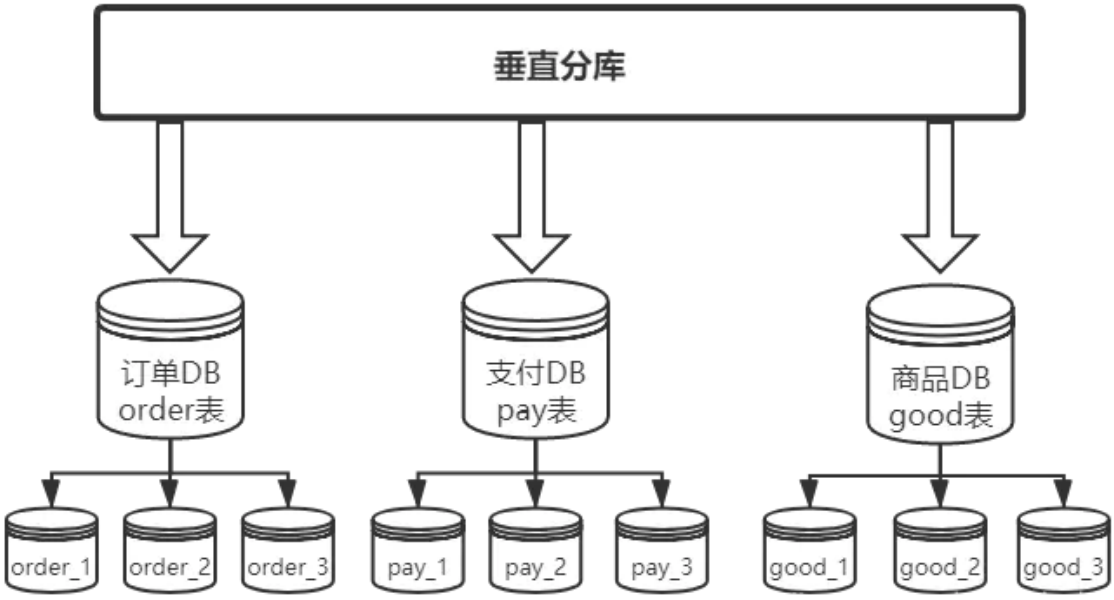
假如一张`order`表有2000万数据，水平切分后出来四个表，`order_1`、`order_2`、`order_3`、`order_4`，每张表数据500万，以此类推。

`order_1`表：

水平切分又分有`库内分表`和`分库分表`

库内分表

库内分表虽然将表拆分，但子表都还是在同一个数据库实例中，只是解决了单一表数据量过大的问题，并没有将拆分后的表分布到不同机器的库上，还在竞争同一个物理机的CPU、内存、网络IO。



分库分表

分库分表则是将切分出来的子表，分散到不同的数据库中，从而使得单个表的数据量变小，达到分布式的效果。

优点：

- 解决高并发时单库数据量过大的问题，提升系统稳定性和负载能力
- 业务系统改造的工作量不是很大

缺点：

- 跨分片的事务一致性难以保证
- 跨库的join关联查询性能较差
- 扩容的难度和维护量较大，（拆分成几千张子表想想都恐怖）

三、数据该往哪个库的表存?

分库分表以后会出现一个问题，一张表会出现在多个数据库里，到底该往哪个库的表里存呢?

3.1 根据取值范围

按照`时间区间`或`ID区间`来切分，举个栗子：假如我们切分的是用户表，可以定义每个库的`User表`里只存100000条数据，第一个库`userId`从1 ~ 9999，第二个库10000 ~ 19999，第三个库20000~29999.....以此类推。

优点：

- 单表数据量是可控的
- 水平扩展简单只需增加节点即可，无需对其他分片的数据进行迁移
- 能快速定位要查询的数据在哪个库

缺点：

- 由于连续分片可能存在数据热点，如果按时间字段分片，有些分片存储最近时间段内的数据，可能会被频繁的读写，而有些分片存储的历史数据，则很少被查询

3.2 hash取模

hash取模mod（**对hash结果取余数 (hash() mod N)**）的切分方式比较常见，还拿`User表`举例，对数据库从0到N-1进行编号，对`User表`中`userId`字段进行取模，得到余数`i`，`i=0`存第一个库，`i=1`存第二个库，`i=2`存第三个库....以此类推。

这样同一个用户的数据都会存在同一个库里，用`userId`作为条件查询就很好定位了。

优点：

- 数据分片相对比较均匀，不易出现某个库并发访问的问题

缺点：

- 但这种算法存在一些问题，当某一台机器宕机，本应该落在该数据库的请求就无法得到正确的处理，这时宕掉的实例会被踢出集群，此时算法变成hash(userId) mod N-1，用户信息可能就不再在同一个库中。

四、有哪些分库分表的工具?

自己开发分库分表工具的工作量是巨大的，好在业界已经有了很多比较成熟的分库分表中间件，我们可

以将更多的时间放在业务实现上

- sharding-jdbc（当当）
- TSharding（蘑菇街）
- Atlas（奇虎360）
- Cobar（阿里巴巴）
- MyCAT（基于Cobar）
- Oceanus（58同城）
- Vitess（谷歌）

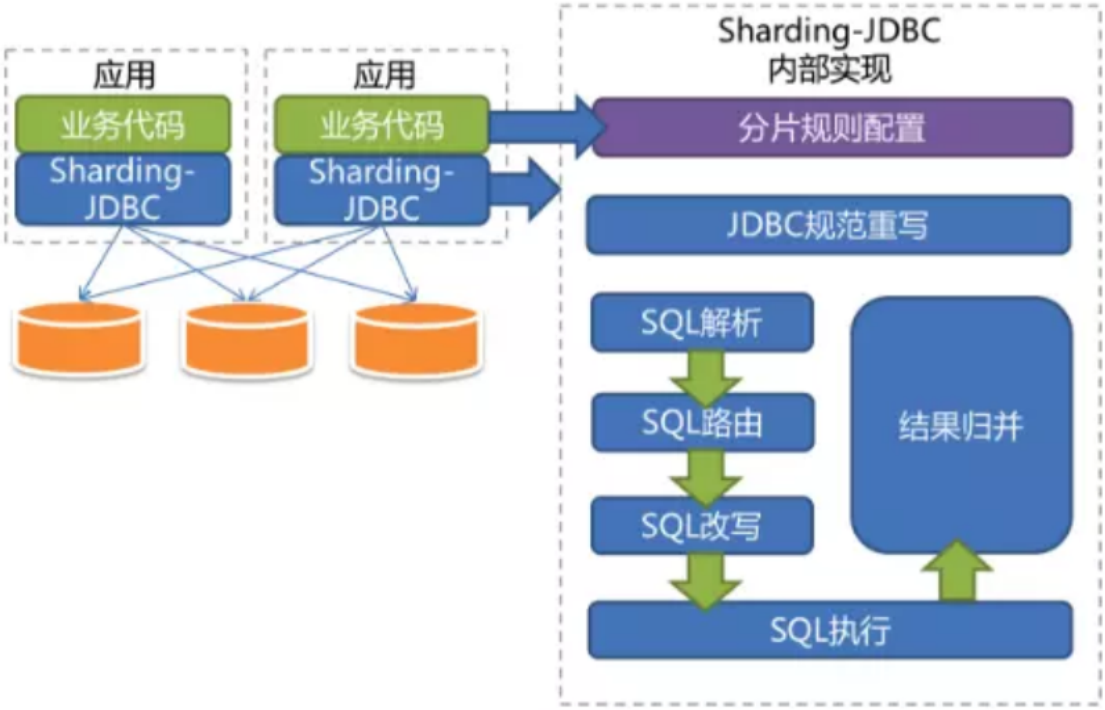
五、Sharding JDBC介绍

5.1 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。

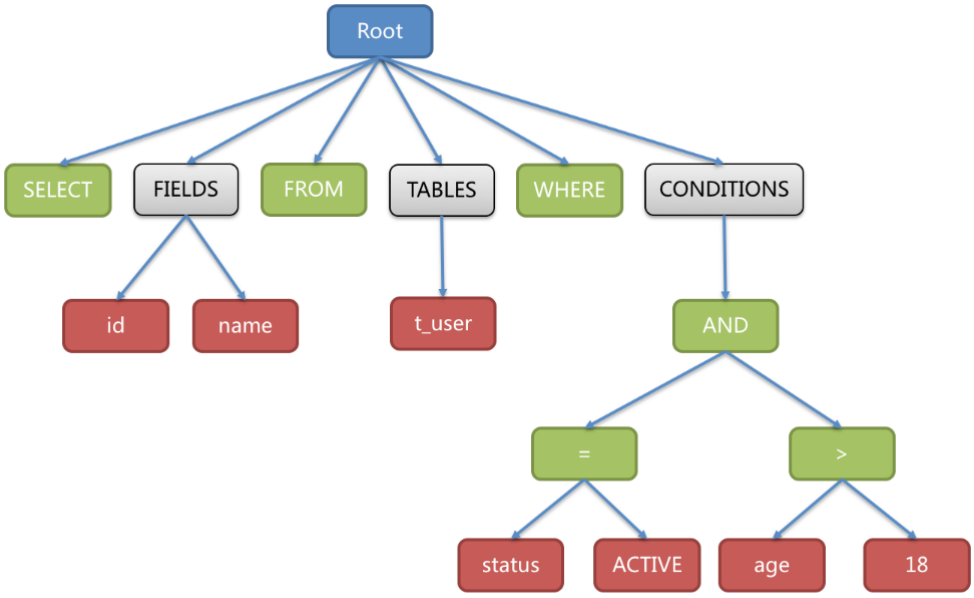
5.2 架构



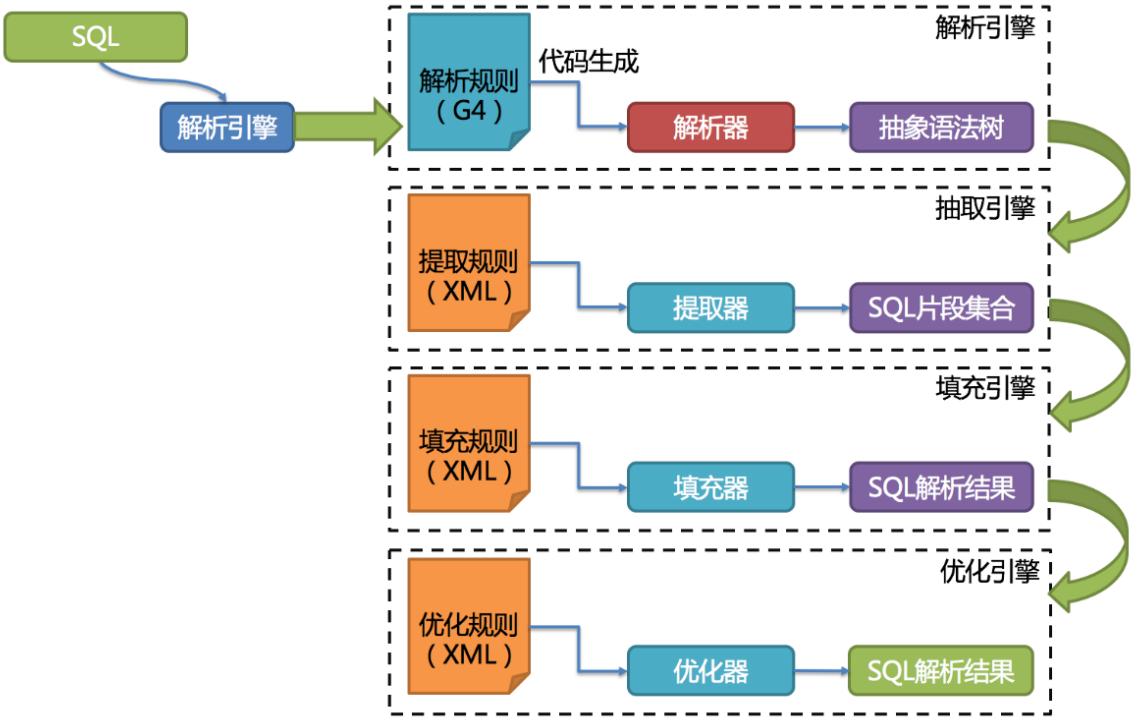
5.3 核心组件

解析引擎

解析过程分为词法解析和语法解析。 词法解析器用于将SQL拆解为不可再分的原子符号，称为Token。并根据不同数据库方言所提供的字典，将其归类为关键字，表达式，字面量和操作符。 再使用语法解析器将SQL转换为抽象语法树。



第三代SQL解析器则从3.0.x版本开始，ShardingSphere尝试使用ANTLR作为SQL解析的引擎。ANTLR是指可以根据输入自动生成语法树并可视化的显示出来的开源语法分析器

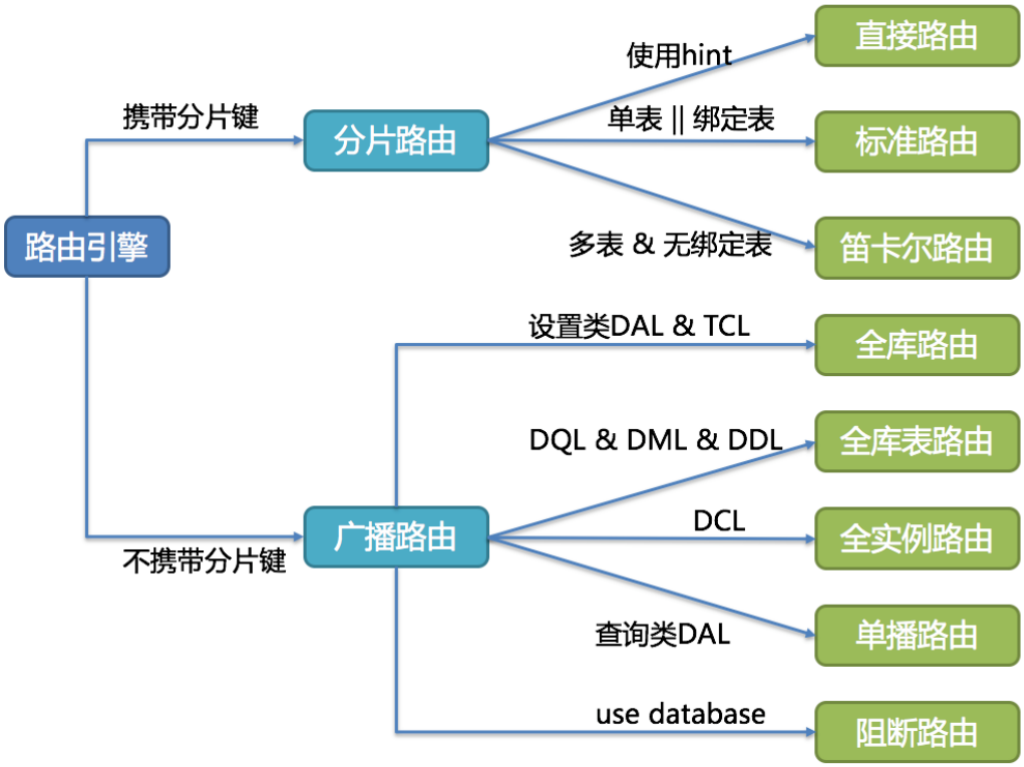


路由引擎

根据解析上下文匹配数据库和表的分片策略，并生成路由路径。 对于携带分片键的SQL，根据分片键的不同可以划分为单片路由(分片键的操作符是等号)、多片路由(分片键的操作符是IN)和范围路由(分片键的操作符是BETWEEN)。 不携带分片键的SQL则采用广播路由。

用于根据分片键进行路由的场景，又细分为直接路由、标准路由和笛卡尔积路由这3种类型。

广播路由，对于不携带分片键的SQL，则采取广播路由的方式。根据SQL类型又可以划分为全库表路由、全库路由、全实例路由、单播路由和阻断路由这5种类型。

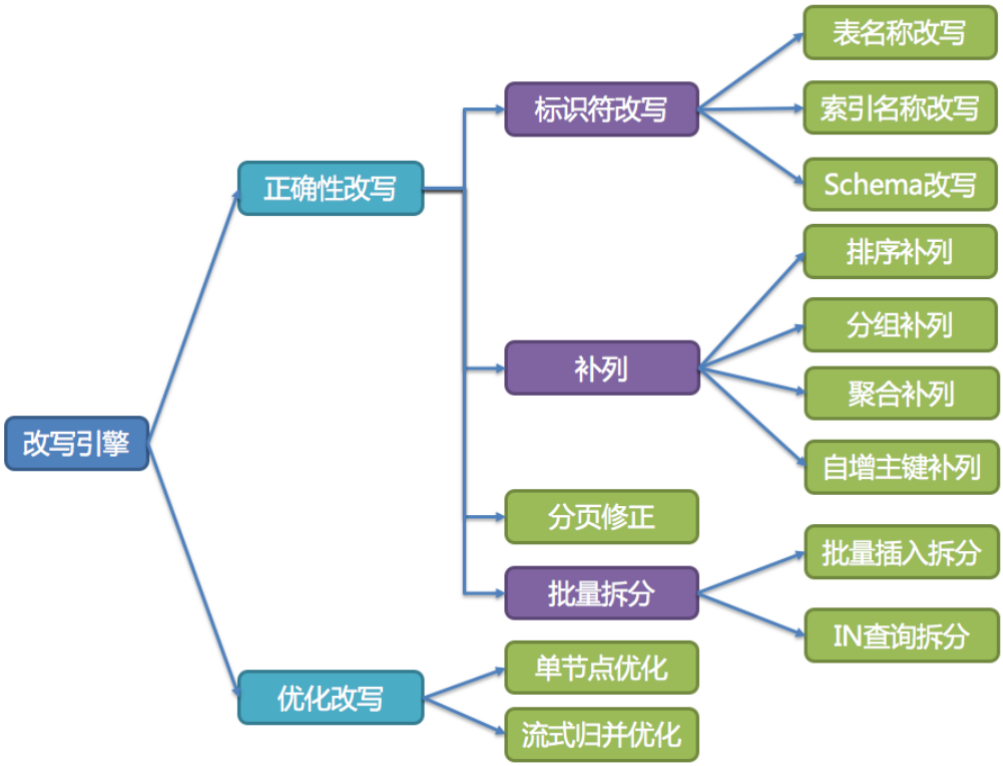


改写引擎

面向逻辑库与逻辑表书写的SQL，并不能够直接在真实的数据库中执行，SQL改写用于将逻辑SQL改写为在真实数据库中可以正确执行的SQL。 它包括正确性改写和优化改写两部分。

正确性改写:在包含分表的场景中，需要将分表配置中的逻辑表名称改写为路由之后所获取的真实表名称。仅分库则不需要表名称的改写。除此之外，还包括补列和分页信息修正等内容。

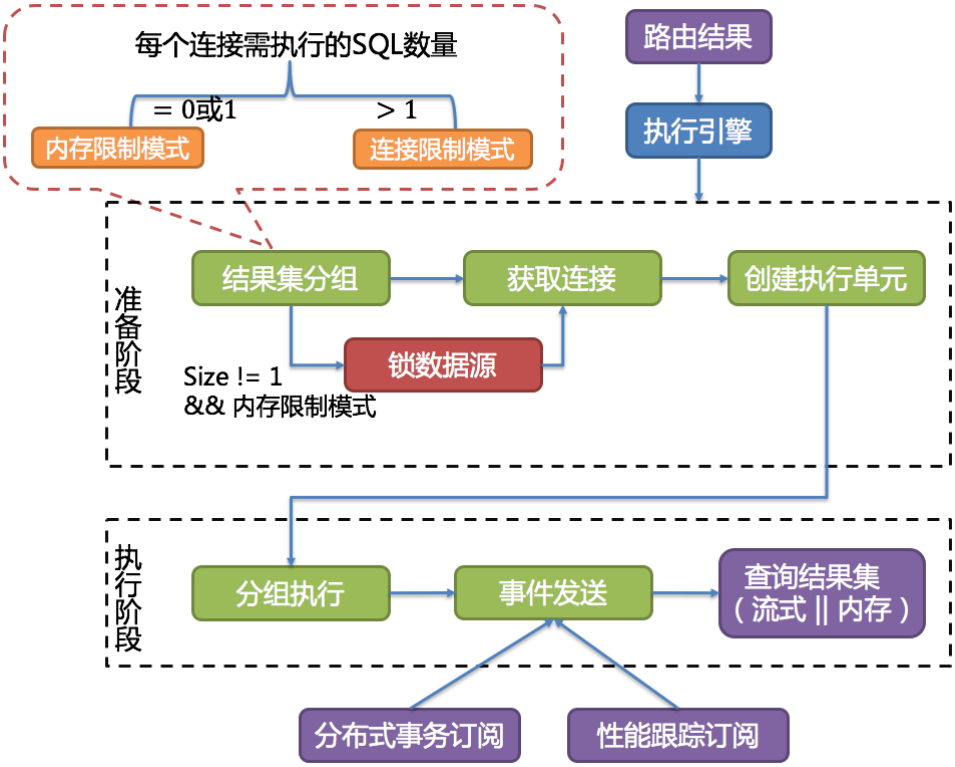
标识符改写:需要改写的标识符包括表名称、索引名称以及Schema名称。



执行引擎

ShardingSphere采用一套自动化的执行引擎，负责将路由和改写完成之后的真实SQL安全且高效发送到底层数据源执行。 它不是简单地将SQL通过JDBC直接发送至数据源执行；也并非直接将执行请求放入线程池去并发执行。它更关注平衡数据源连接创建以及内存占用所产生的消耗，以及最大限度地合理利用并发等问题。 执行引擎的目标是自动化的平衡资源控制与执行效率。

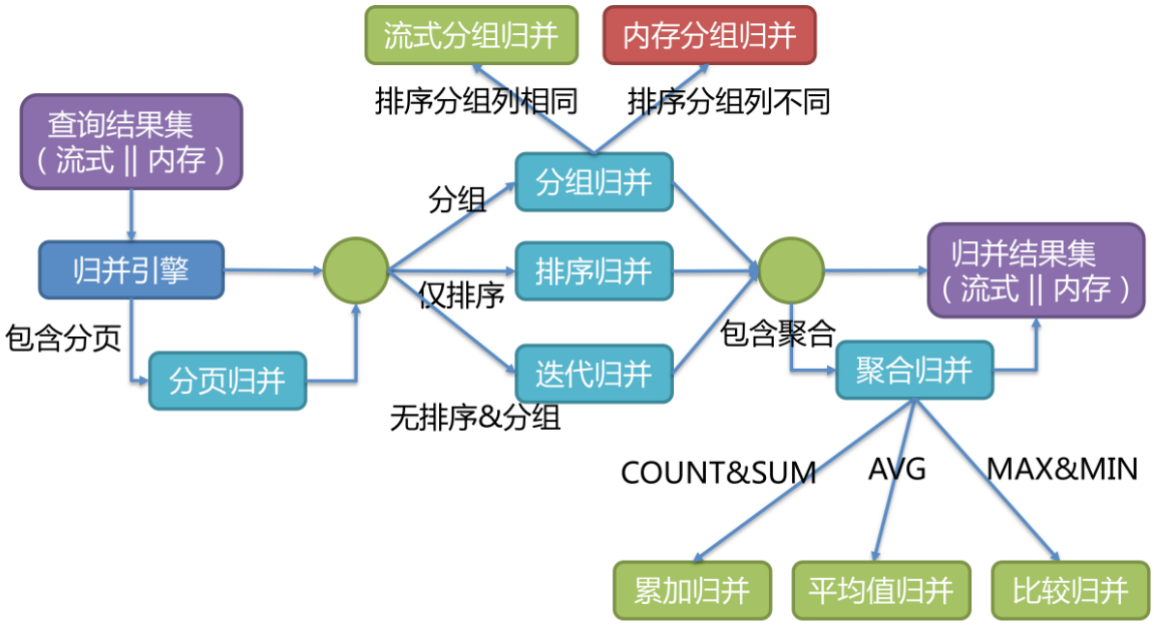
执行引擎分为准备和执行两个阶段。准备阶段用于准备执行的数据。它分为结果集分组和执行单元创建两个步骤。该阶段用于真正的执行SQL，它分为分组执行和归并结果集生成两个步骤。



归并引擎

将从各个数据节点获取的多数据结果集，组合成为一个结果集并正确的返回至请求客户端，称为结果归并。

ShardingSphere支持的结果归并从功能上分为遍历、排序、分组、分页和聚合5种类型，它们是组合而非互斥的关系。 从结构划分，可分为流式归并、内存归并和装饰者归并。流式归并和内存归并是互斥的，装饰者归并可以在流式归并和内存归并之上做进一步的处理。



5.4 案例：读写分离

1、创建表

```sql
CREATE TABLE `t_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `address` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
```

2、pom文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>sharding-jdbc-test</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.apache.shardingsphere</groupId>
            <artifactId>sharding-jdbc-core</artifactId>
            <version>4.1.1</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.20</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>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.23</version>
        </dependency>
```

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
            <scope>test</scope>
        </dependency>
    </dependencies>


</project>
```

3、实现读写分离

```
public class MasterSlaveDataSource {
    private static DataSource dataSource;

    public static DataSource getInstance() {
        if (dataSource != null) {
            return dataSource;
        }
        try {
            return create();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return null;
    }

    private static DataSource create() throws SQLException {
        // 配置真实数据源
        Map<String, DataSource> dataSourceMap = new HashMap<>();

        // 配置第 1 个数据源
        DruidDataSource masterDataSource = new DruidDataSource();
        masterDataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        masterDataSource.setUrl("jdbc:mysql://192.168.68.132:3306/hello?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8");
        masterDataSource.setUsername("root");
        masterDataSource.setPassword("root");
        dataSourceMap.put("master", masterDataSource);

        // 配置第 2 个数据源
        DruidDataSource slaveDataSource = new DruidDataSource();
        slaveDataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        slaveDataSource.setUrl("jdbc:mysql://192.168.68.133:3306/hello?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8");
        slaveDataSource.setUsername("root");
        slaveDataSource.setPassword("root");
        dataSourceMap.put("slave", slaveDataSource);

        //创建主从复制数据源
        MasterSlaveRuleConfiguration masterSlaveRuleConfig = new
MasterSlaveRuleConfiguration("masterSlaveDataSource", "master",
Arrays.asList("slave"));
        dataSource =
MasterSlaveDataSourceFactory.createDataSource(dataSourceMap,
masterSlaveRuleConfig , new Properties());
```

```
        //返回数据源
        return dataSource;
    }

}
```

5.5 案例：实现分库分表

1、创建表

```
CREATE TABLE `t_order0` (
  `order_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `info` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `t_order1` (
  `order_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `info` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2、实现分库分表

```
package com.kkb.common;

import com.alibaba.druid.pool.DruidDataSource;
import
org.apache.shardingsphere.api.config.sharding.ShardingRuleConfiguration;
import org.apache.shardingsphere.api.config.sharding.TableRuleConfiguration;
import
org.apache.shardingsphere.api.config.sharding.strategy.InlineShardingStrateg
yConfiguration;
import org.apache.shardingsphere.shardingjdbc.api.ShardingDataSourceFactory;

import javax.sql.DataSource;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

public class ShardingDataSource {
    private static DataSource dataSource;

    public static DataSource getInstance() {
        if (dataSource != null) {
            return dataSource;
        }
        try {
            return create();
        } catch (SQLException throwables) {
```

```
            throwables.printStackTrace();
        }
        return null;
    }

    private static DataSource create() throws SQLException {
        // 配置真实数据源
        Map<String, DataSource> dataSourceMap = new HashMap<>();

        // 配置第 1 个数据源
        DruidDataSource dataSource1 = new DruidDataSource();
        dataSource1.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource1.setUrl("jdbc:mysql://192.168.68.132:3306/hello?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8");
        dataSource1.setUsername("root");
        dataSource1.setPassword("root");
        dataSourceMap.put("ds0", dataSource1);

        // 配置第 2 个数据源
        DruidDataSource dataSource2 = new DruidDataSource();
        dataSource2.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource2.setUrl("jdbc:mysql://192.168.68.134:3306/hello?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8");
        dataSource2.setUsername("root");
        dataSource2.setPassword("root");
        dataSourceMap.put("ds1", dataSource2);

        // 配置 t_order 表规则
        TableRuleConfiguration orderTableRuleConfig = new
TableRuleConfiguration("t_order","ds${0..1}.t_order${0..1}");

        // 配置数据库表分片策略
        orderTableRuleConfig.setDatabaseShardingStrategyConfig(new
InlineShardingStrategyConfiguration("user_id", "ds${user_id % 2}"));
        orderTableRuleConfig.setTableShardingStrategyConfig(new
InlineShardingStrategyConfiguration("order_id", "t_order${order_id % 2}"));

        // 配置分片规则
        ShardingRuleConfiguration shardingRuleConfig = new
ShardingRuleConfiguration();
        shardingRuleConfig.getTableRuleConfigs().add(orderTableRuleConfig);

        // 创建数据源
        DataSource dataSource =
ShardingDataSourceFactory.createDataSource(dataSourceMap,
shardingRuleConfig, new Properties());
        return dataSource;
    }
}

```

3、代码

> 见资料