SphereEx | 让数据和应用的连接如此简单

Apache ShardingSphere 可插拔内核与动态组合配置

端正强
duanzhengqiang@apache.org

端正强

SphereEx 高级中间件开发工程师
Apache ShardingSphere Committer

- 2018 年开始接触 Apache ShardingSphere 中间件，曾主导公司内部海量数据的分库分表，有着丰富的实践经验；
- 热爱开源，乐于分享，目前专注于 Apache ShardingSphere 内核模块开发；

目录

01 简介 Introduction

02 可插拔内核 Extensible Kernel

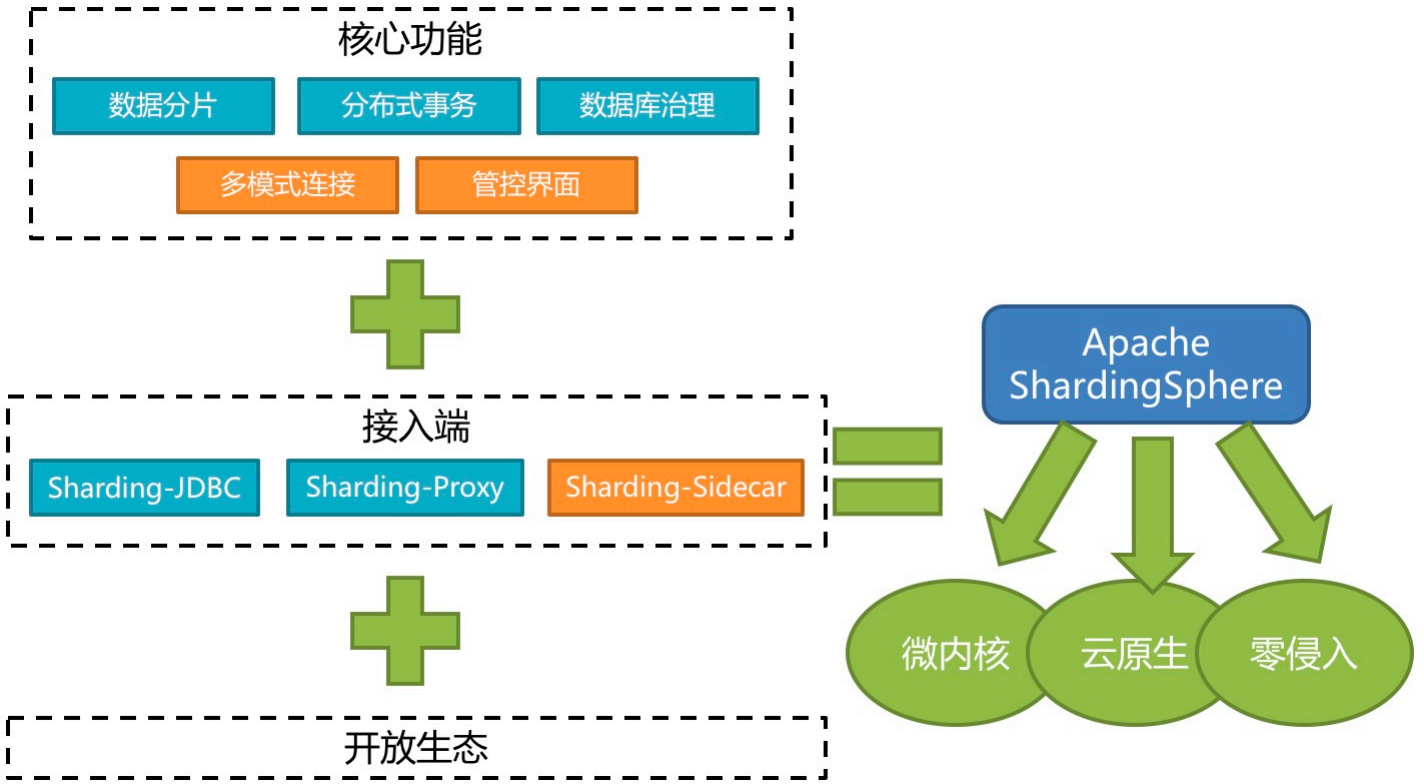
03 动态组合配置 Mixed Configuration

04 社区 Community

目录

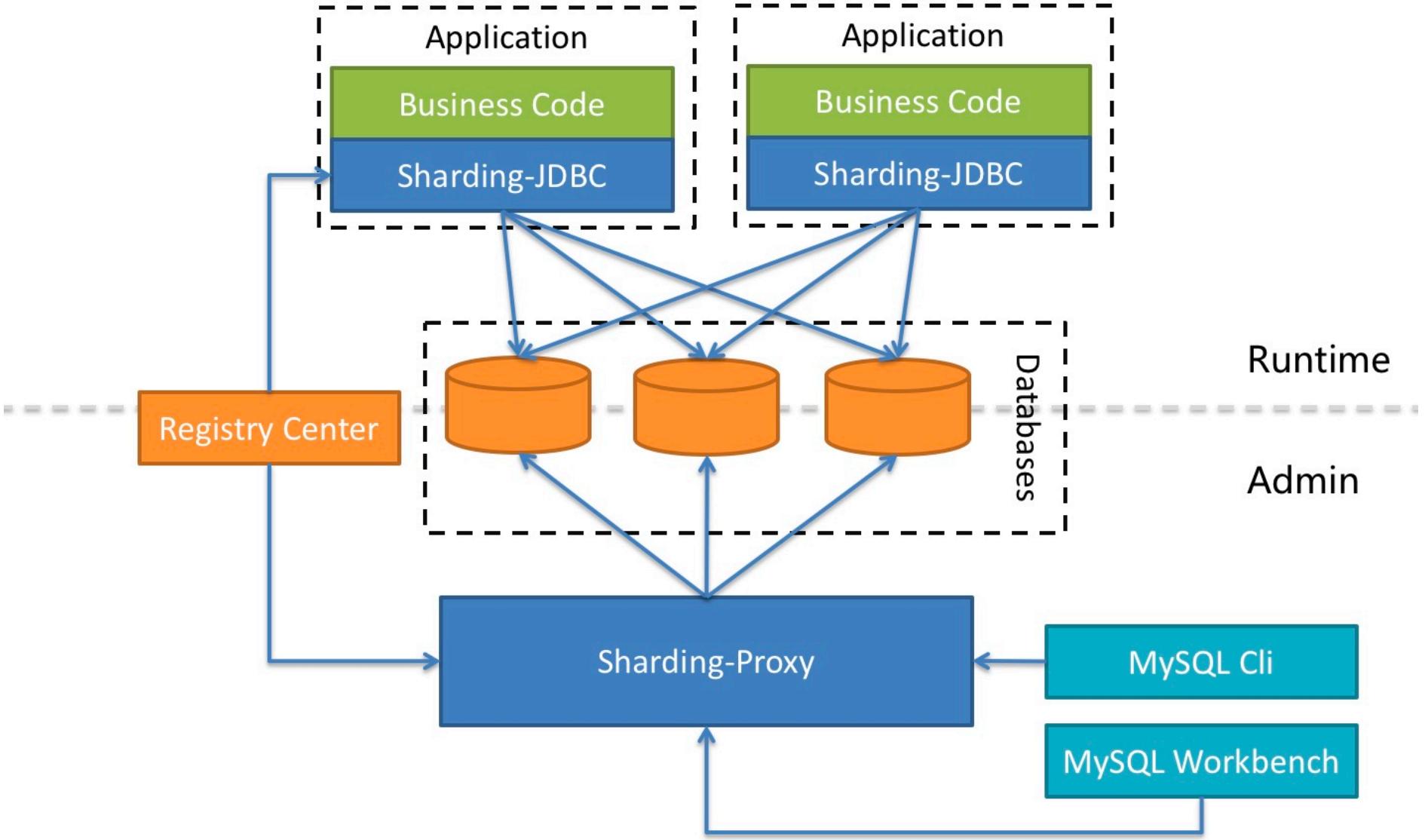
01 简介 Introduction

简介-生态圈



- 开源分布式数据库生态圈，提供了标准化的数据分片、分布式事务及数据库治理功能；
- 由 JDBC、Proxy 和 Sidecar 三款产品组成，支持独立部署及混合部署；
- 开放的生态体系，通过可插拔架构，方便用户根据业务场景进行灵活扩展；

简介-架构


Application
Business Code
Sharding-JDBC
Application
Business Code
Sharding-JDBC
Registry Center
Databases
Runtime
Admin
Sharding-Proxy
MySQL Cli
MySQL Workbench

简介-对比

	ShardingSphere-JDBC	ShardingSphere-Proxy
数据库	任意	MySQL/PostgreSQL
连接消耗数	高	低
异构语言	仅 Java	任意
性能	损耗低	损耗略高
无中心化	是	否
静态入口	无	有

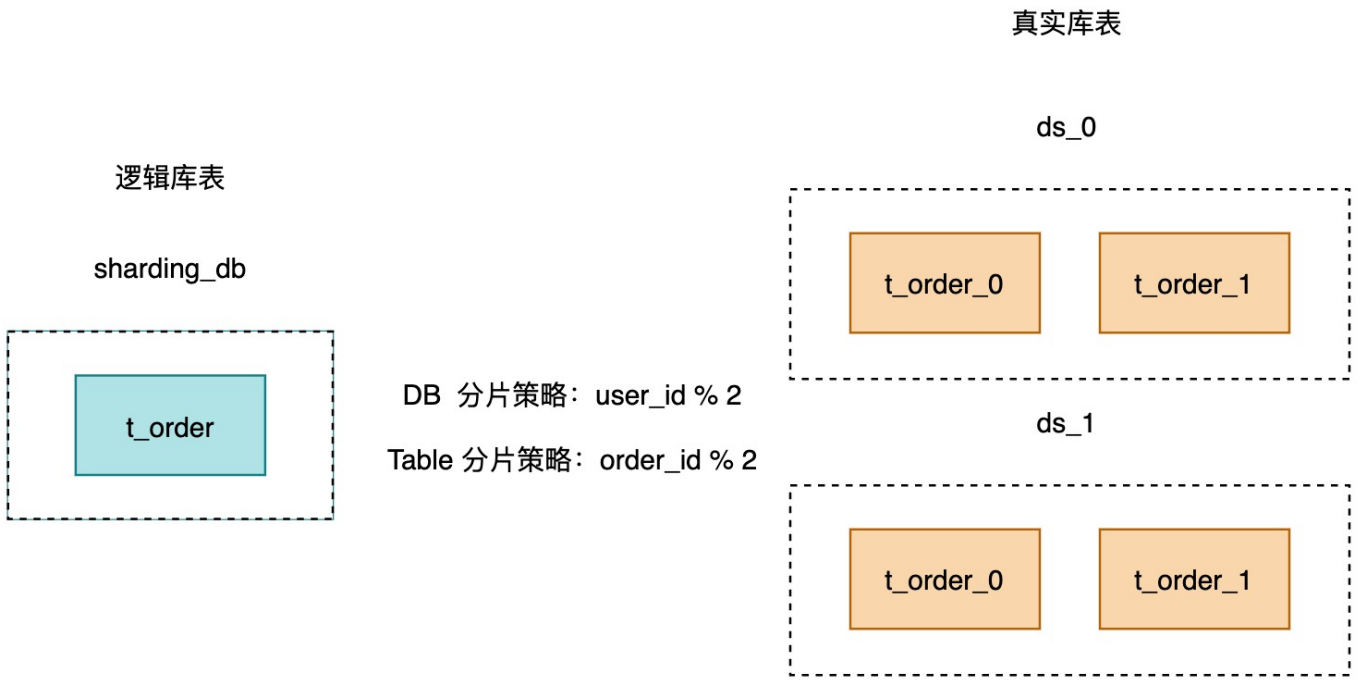
目录

02 可插拔内核 Extensible Kernel

可插拔内核-业务场景

为了应对千万级的订单存量数据，以及快速增长的增量数据，某用户引入了 Apache ShardingSphere 进行分片，并配置了如下规则。按照 user_id 和 order_id 取模分片，共分为 4 片。

```yaml
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order_inline
      databaseStrategy:
        standard:
          shardingColumn: user_id
          shardingAlgorithmName: database_inline

  shardingAlgorithms:
    database_inline:
      type: INLINE
      props:
        algorithm-expression: ds_${user_id % 2}
    t_order_inline:
      type: INLINE
      props:
        algorithm-expression: t_order_${order_id % 2}
```



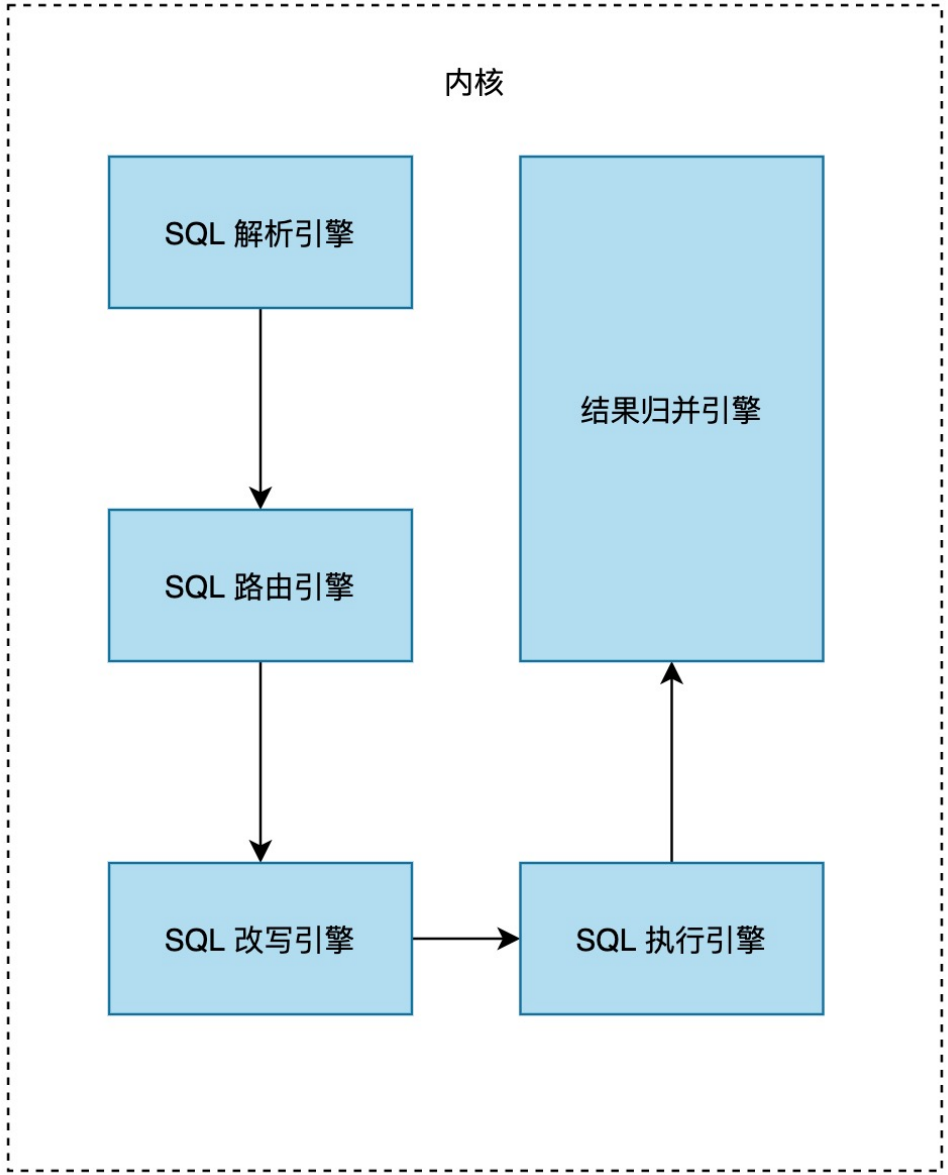
可插拔内核-业务场景

使用 Apache ShardingSphere 进行分片后，用户只需要像使用原始数据库那样，基于逻辑表 t_order 进行增删改查即可，背后的分片逻辑处理都是由 Apache ShardingSphere 内核模块帮助用户完成，整个过程对用户透明。

那么，内核是如何完成整个分片逻辑的处理呢？

```
mysql> CREATE TABLE t_order(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100));
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO t_order(order_id, user_id, content) VALUES(20, 1, 'TEST1'), (21, 1, 'TEST2');
Query OK, 2 rows affected (0.02 sec)

mysql> SELECT * FROM t_order WHERE user_id = 1 AND order_id IN (20, 21);
+----------+---------+---------+
| order_id | user_id | content |
+----------+---------+---------+
|       20 |       1 | TEST1   |
|       21 |       1 | TEST2   |
+----------+---------+---------+
2 rows in set (0.04 sec)
```

可插拔内核-内核流程

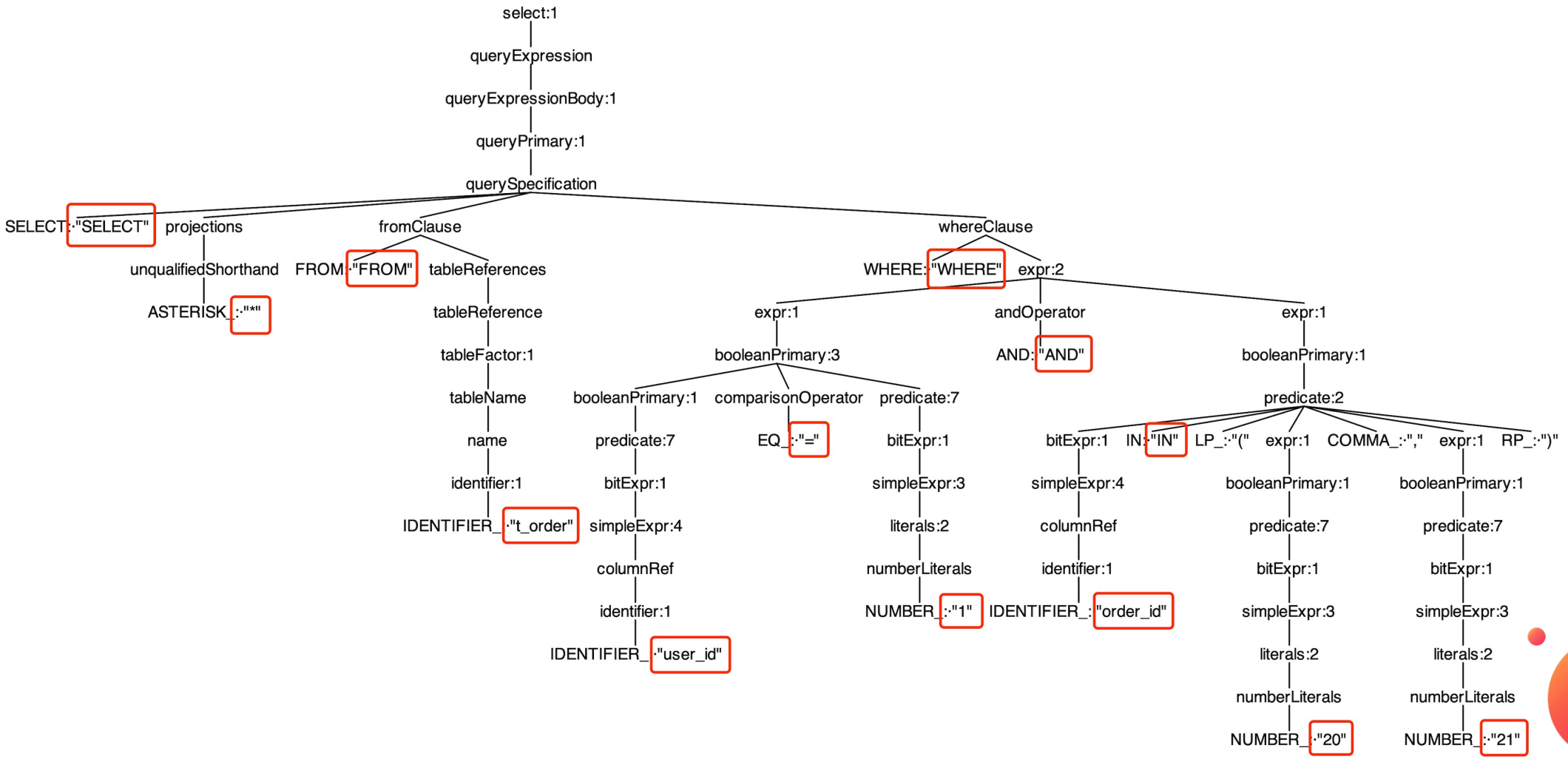


- SQL 解析引擎负责对用户输入的 SQL 进行解析，将 SQL 语法树处理为 SQLStatement 供内核使用；
- SQL 路由引擎根据解析上下文信息，以及用户配置的分片策略，生成路由结果，目前支持分片路由和广播路由；
- SQL 改写引擎负责将原始 SQL 改写为在真实数据库中可以正确执行的真实 SQL，可以分为正确性改写和优化改写；
- SQL 执行引擎负责将路由和改写完成之后的真实 SQL，安全高效地发送到底层数据源进行执行；
- 结果归并引擎负责将各个数据节点返回的数据结果集，组合成为一个结果集并正确的返回给用户；

可插拔内核-内核流程-SQL 解析引擎

解析流程：SQL -> AST -> SQLStatement

SQL示例：SELECT * FROM t_order WHERE user_id = 1 AND order_id IN (20, 21);



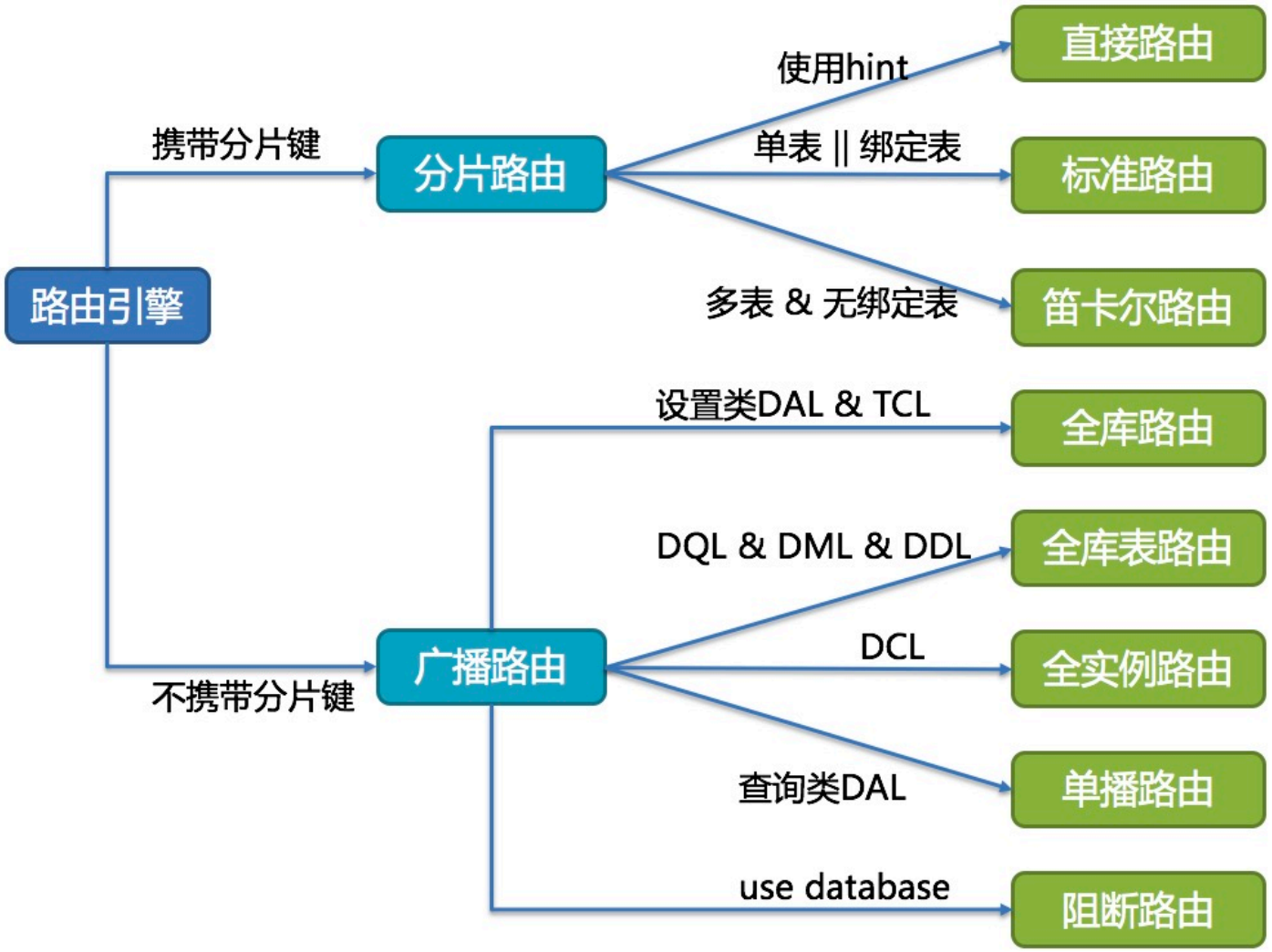
可插拔内核-内核流程-SQL 解析引擎

目前，SQL 解析引擎支持 MySQL、PostgreSQL、SQLServer 和 Oracle 等主流数据库，以及其他遵循 SQL92 标准的数据库。

SQL 解析引擎也提供了独立的 SQL 解析功能，可以方便地集成到其他项目中，用来实现 SQL 解析、SQL 格式化相关的功能。

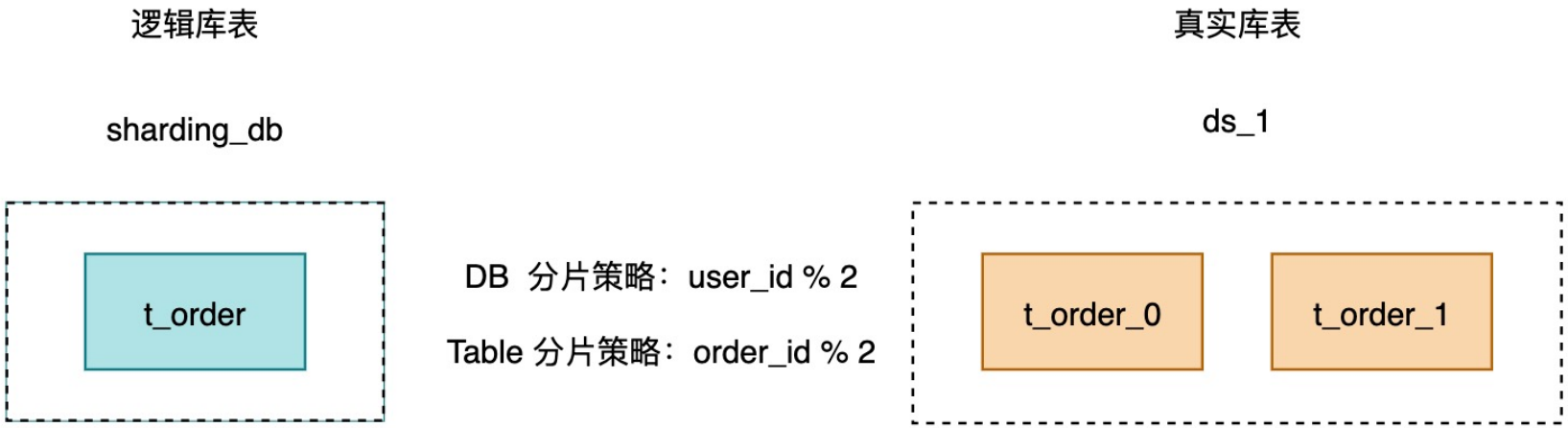
数据库	支持状态
MySQL	支持，完善
PostgreSQL	支持，完善
SQLServer	支持
Oracle	支持
SQL92	支持

可插拔内核-内核流程-SQL 路由引擎

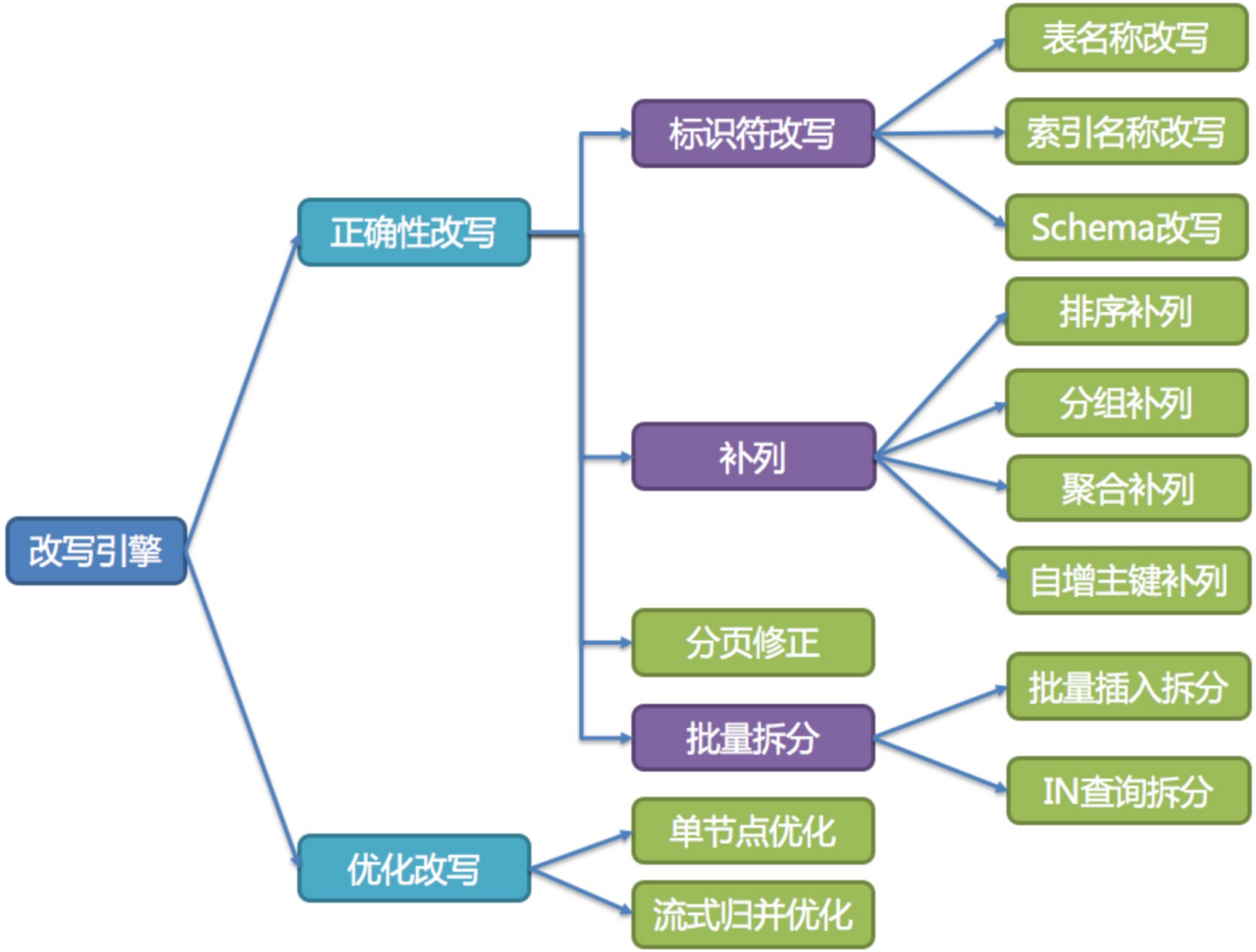


可插拔内核-内核流程-SQL 路由引擎

SQL示例：SELECT * FROM t_order WHERE user_id = 1 AND order_id IN (20, 21);

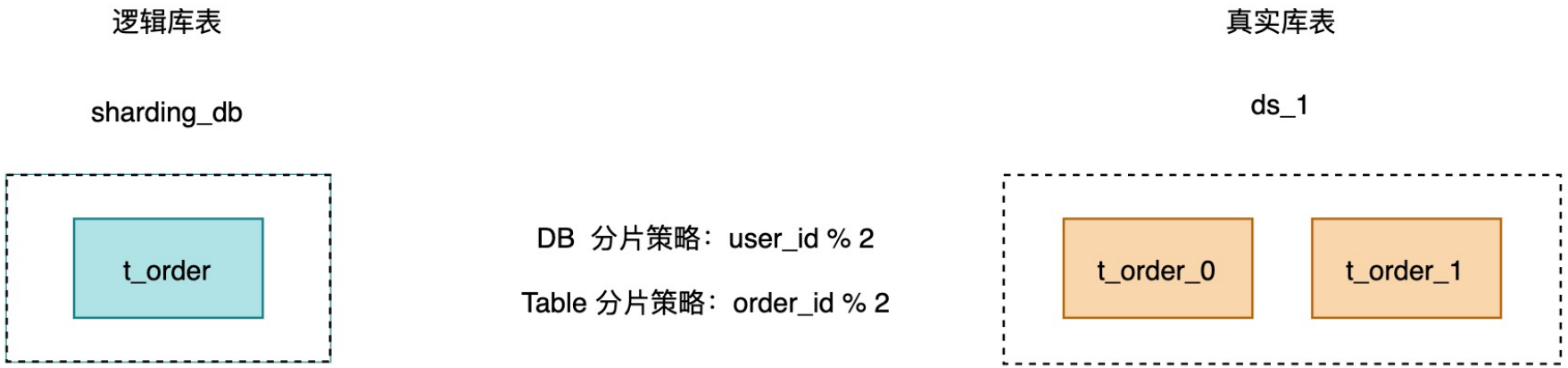


可插拔内核-内核流程-SQL 改写引擎



可插拔内核-内核流程-SQL 改写引擎

SQL示例：SELECT * FROM t_order WHERE user_id = 1 AND order_id IN (20, 21);



Logic SQL: SELECT * FROM t_order WHERE user_id = 1 AND order_id IN (20, 21)

Actual SQL: ds_1 ::: SELECT * FROM t_order_0 WHERE user_id = 1 AND order_id IN (20, 21)
Actual SQL: ds_1 ::: SELECT * FROM t_order_1 WHERE user_id = 1 AND order_id IN (20, 21)

可插拔内核-内核流程-SQL 执行/归并引擎

SQL 执行引擎内部根据路由结果以及用户配置的 maxConnectionSizePerQuery 参数，决定使用内存限制模式还是连接限制模式。最终通过归并引擎，对各个数据节点返回的结果集进行处理，统一返回给用户。

真实库： ds_1

真实表： t_order_0

```
mysql> SELECT * FROM t_order_0 WHERE user_id = 1 AND order_id IN (20, 21);
+----------+---------+---------+
| order_id | content | user_id |
+----------+---------+---------+
|       20 | 222     |       1 |
+----------+---------+---------+
1 row in set (0.00 sec)
```

真实库： ds_1

真实表： t_order_1

```
mysql> SELECT * FROM t_order_1 WHERE user_id = 1 AND order_id IN (20, 21);
+----------+---------+---------+
| order_id | content | user_id |
+----------+---------+---------+
|       21 | 222     |       1 |
+----------+---------+---------+
1 row in set (0.00 sec)
```

逻辑库： sharding_db

逻辑表： t_order

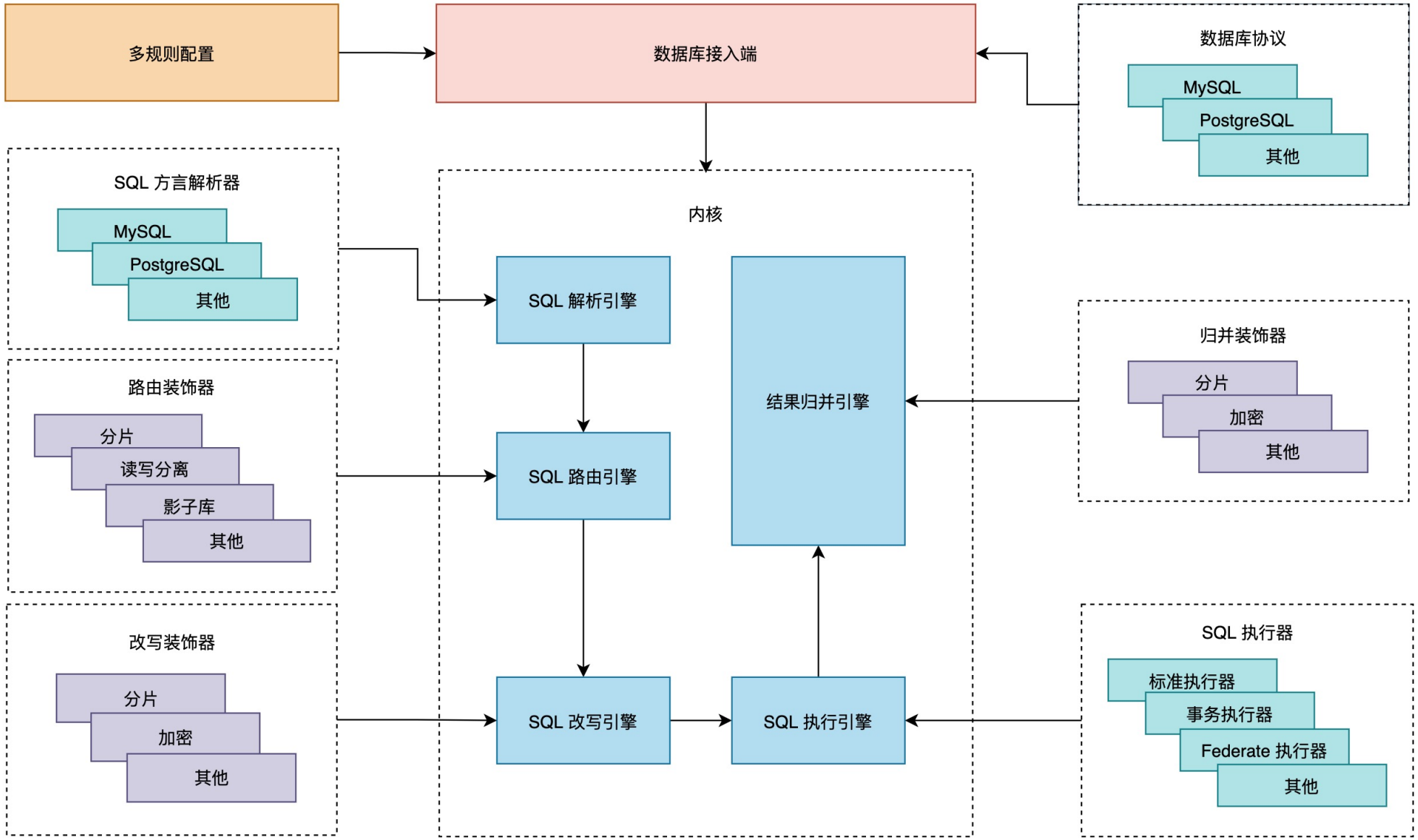
```
mysql> SELECT * FROM t_order WHERE user_id = 1 AND order_id IN (20, 21);
+----------+---------+---------+
| order_id | content | user_id |
+----------+---------+---------+
|       20 | 222     |       1 |
|       21 | 222     |       1 |
+----------+---------+---------+
2 rows in set (0.01 sec)
```

可插拔内核-挑战

用户真实的业务场景复杂多样，现有内核的功能，无法满足用户的全部需求。

那么 Apache ShardingSphere 如何应对业务场景的复杂多样性呢？

可插拔内核-可插拔架构


多规则配置
数据库接入端
数据库协议
MySQL
PostgreSQL
其他
SQL 方言解析器
MySQL
PostgreSQL
其他
内核
SQL 解析引擎
结果归并引擎
归并装饰器
分片
加密
其他
路由装饰器
分片
读写分离
影子库
其他
SQL 路由引擎
改写装饰器
分片
加密
其他
SQL 改写引擎
SQL 执行引擎
SQL 执行器
标准执行器
事务执行器
Federate 执行器
其他

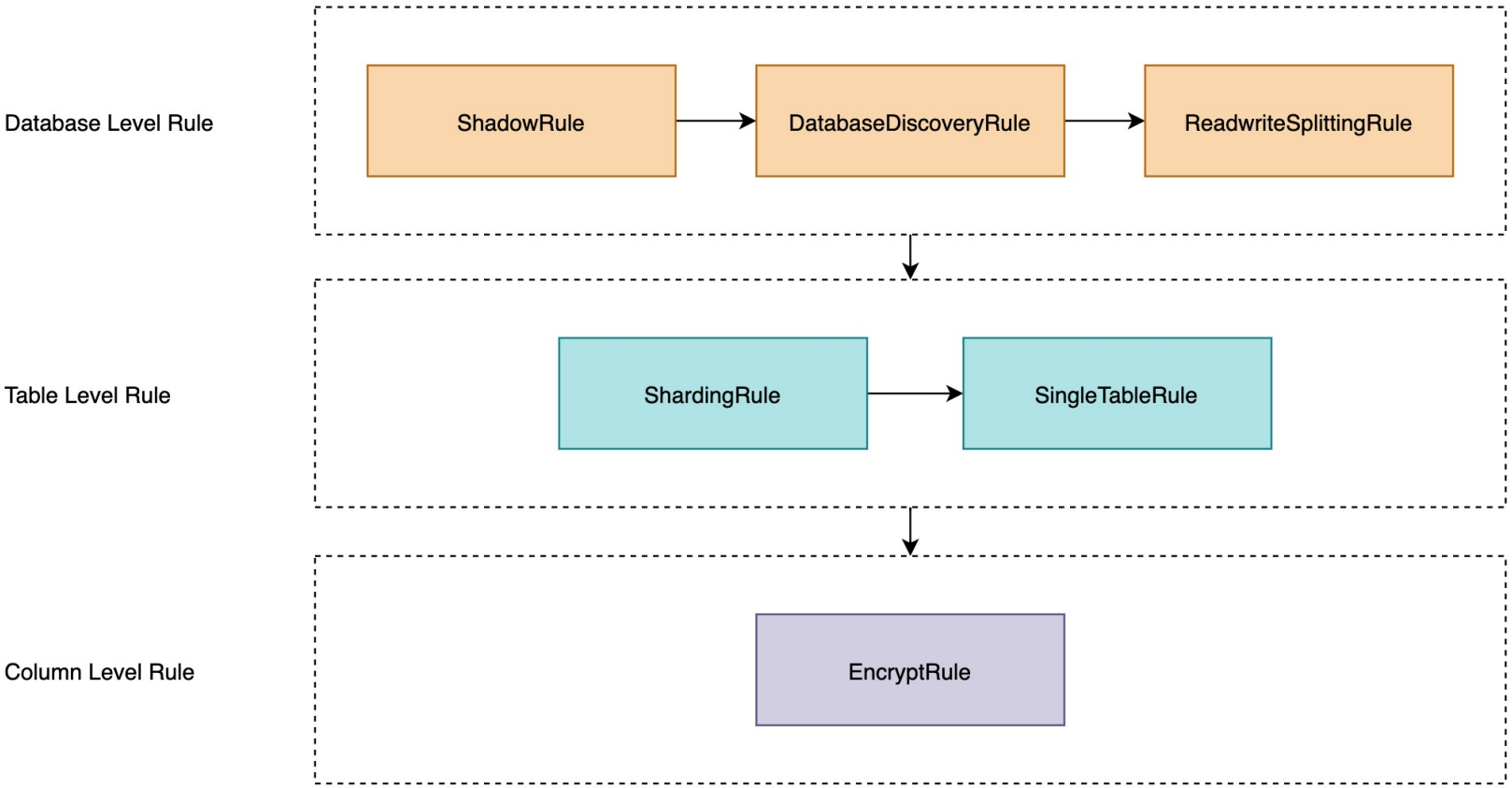
可插拔内核-优势

	传统内核	可插拔内核
定位	分库分表中间件	分布式数据库生态圈
功能	提供功能	提供基础设施和最佳实践
耦合	耦合较大，存在功能依赖	相互隔离，互无感知
组合方式	固定的组合方式： 以数据分片为基础，叠加读写分离和数据加密等功能	自由的组合方式： 数据分片、读写分离和数据加密等功能自由组合使用

目录

03 动态组合配置 Mixed Configuration

动态组合配置-功能划分

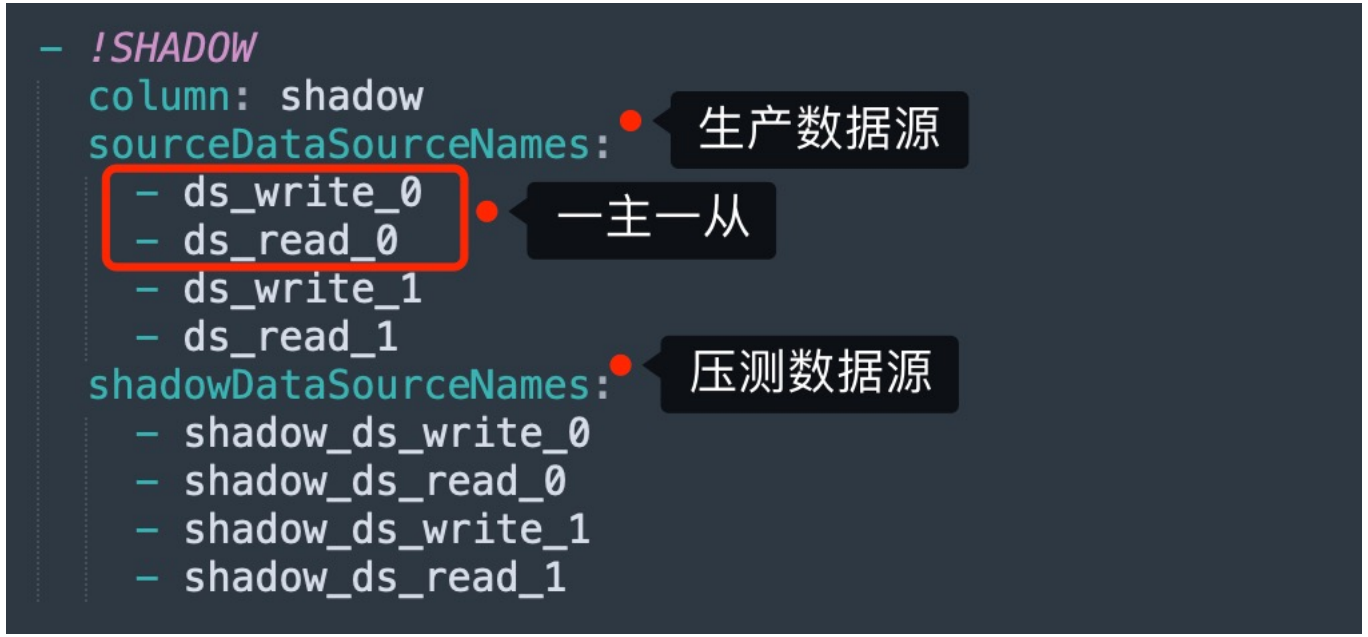


- ShardingSphere 内核功能以 Rule 为基础，根据用户配置的规则，加载对应的功能；
- 可以将 Rule 划分为三层，分别对应数据源、表和列的处理；
- SingleTableRule 为内置 Rule，为内核功能提供单表路由能力，无需用户配置；
- 内核功能按照 Rule 的层级加载和使用，多个功能以装饰的方式叠加使用；

动态组合配置-典型配置

以影子库压测（Shadow）、读写分离（ReadWriteSplitting ）、分片（Sharding）和加解密（Encrypt ）组合使用的业务场景为例，来看下典型的组合配置。

根据前文对功能的划分，内核功能是按照 Rule 的层级和顺序进行加载和使用的。因此，需要配置影子库压测的规则。影子库规则中，考虑了业务场景中读写分离和分片多数据节点的需求，配置了 4 组数据源映射关系。sourceDataSourceNames 为生产数据源，shadowDataSourceNames 为压测数据源，生产数据源和压测数据源中都包含了两组主从数据源。



动态组合配置-典型配置

影子库压测规则配置完成后，内核会根据映射关系，聚合出 4 个逻辑数据源——ds_write_0、ds_read_0、ds_write_1 和 ds_read_1。

按照加载顺序，其次需要配置读写分离规则，此时只需要面向逻辑数据源配置，即可完成功能的组合。读写分离规则同样聚合出新的逻辑数据源——ds_0 和 ds_0，提供给分片规则中的 actualDataNodes 使用。

```
- !READWRITE_SPLITTING
  dataSources:
    ds_0:
      writeDataSourceName: ds_write_0
      readDataSourceNames:
        - ds_read_0
      loadBalancerName: roundRobin
    ds_1:
      writeDataSourceName: ds_write_1
      readDataSourceNames:
        - ds_read_1
      loadBalancerName: roundRobin
```

读写分离聚合数据源

动态组合配置-典型配置

分片功能在整体层级上属于表级别，依赖数据库层级功能聚合出的逻辑数据源。

通过在 actualDataNodes 中配置表达式，基于聚合数据源 ds_0、ds_1 和真实表 t_order_0、t_order_1，分片功能聚合出逻辑表 t_order，用户只要面向逻辑表 t_order 编写 SQL，无需关心真实数据源和真实表。

```yaml
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order_inline
      databaseStrategy:
        standard:
          shardingColumn: user_id
          shardingAlgorithmName: database_inline
```

面向逻辑数据源，聚合出逻辑表

动态组合配置-典型配置

加解密功能属于列层级，基于聚合数据源和聚合表，配置逻辑列和真实列的映射关系，以及加解密算法。加解密功能不关心数据分布情况，只负责在数据库基础之上，提供增量的功能。

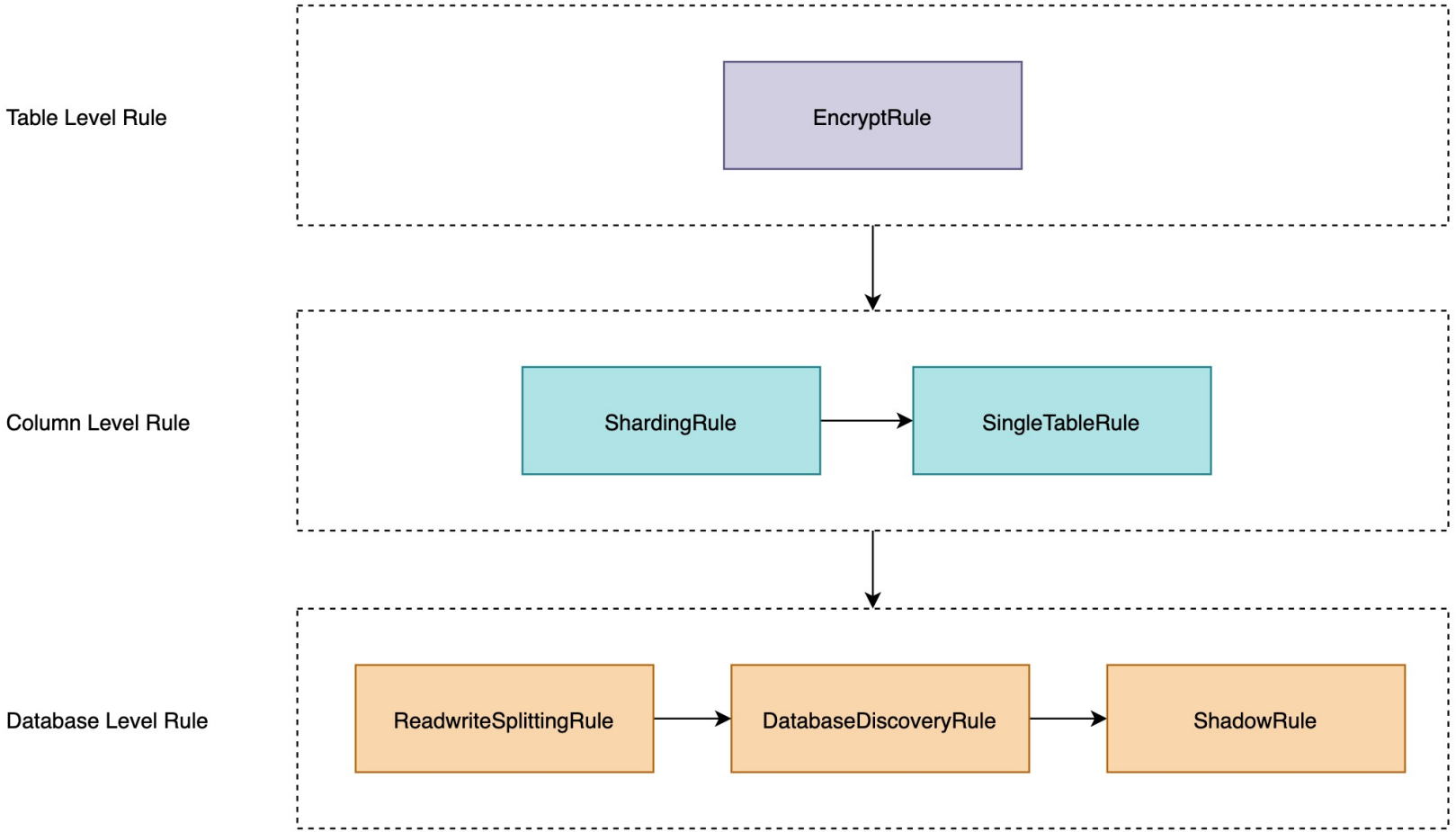
同样，用户在编写 SQL 时，只需要面向逻辑列 telephone，由内核负责加解密处理及 SQL 的改写。

```
- !ENCRYPT
  encryptors:
    aes_encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
  tables:
    t_order:
      columns:
        telephone:
          plainColumn: telephone_plain
          cipherColumn: telephone_cipher
          encryptorName: aes_encryptor
```

面向逻辑表配置加解密规则

逻辑表和真实表

动态组合配置-功能使用

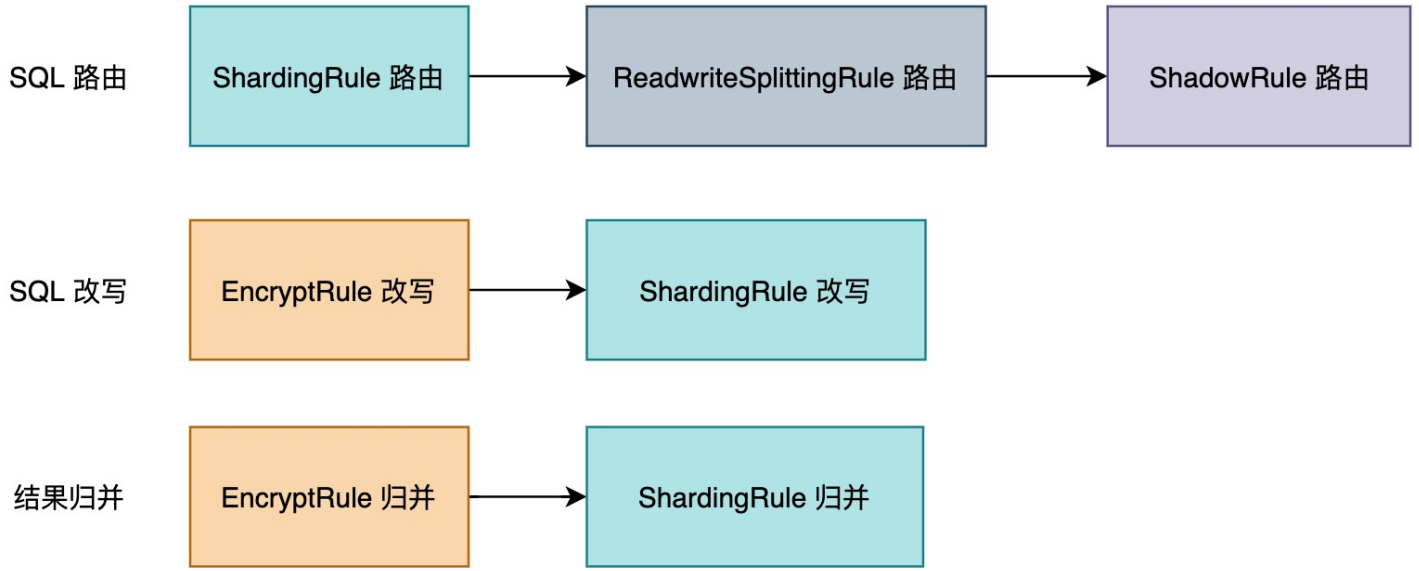


- Apache ShardingSphere 内核功能使用时，按照加载顺序逆序对数据源进行拆分，由聚合数据源路由到真实数据源；
- 逻辑表和逻辑列会根据用户配置的规则改写为真实表和真实列；
- 未来会考虑去除同一层级不同功能的依赖顺序，根据用户配置规则自动推断；

动态组合配置-功能使用

DDL 示例：CREATE TABLE t_order(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), telephone VARCHAR(100));

```
ds_write_1 ::: CREATE TABLE t_order_0(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
ds_write_1 ::: CREATE TABLE t_order_1(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
ds_write_0 ::: CREATE TABLE t_order_0(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
ds_write_0 ::: CREATE TABLE t_order_1(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
shadow_ds_write_1 ::: CREATE TABLE t_order_0(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
shadow_ds_write_1 ::: CREATE TABLE t_order_1(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
shadow_ds_write_0 ::: CREATE TABLE t_order_0(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
shadow_ds_write_0 ::: CREATE TABLE t_order_1(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100), user_cipher VARCHAR(100), user_plain VARCHAR(100))
```

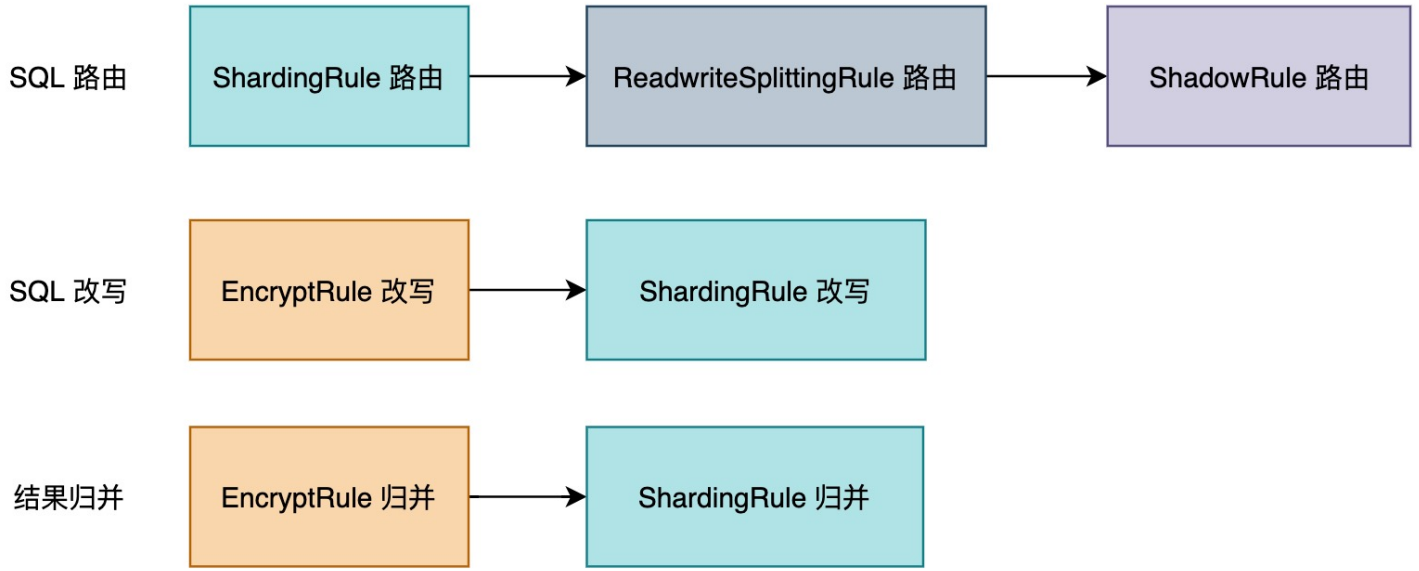


动态组合配置-功能使用

DML 示例：INSERT INTO t_order(order_id, user_id, content, telephone)
VALUES (1, 1, 'TEST1', '12345678900'), (2, 2, 'TEST2', '09876543211');

SELECT * FROM t_order WHERE order_id = 1 AND user_id = 1;

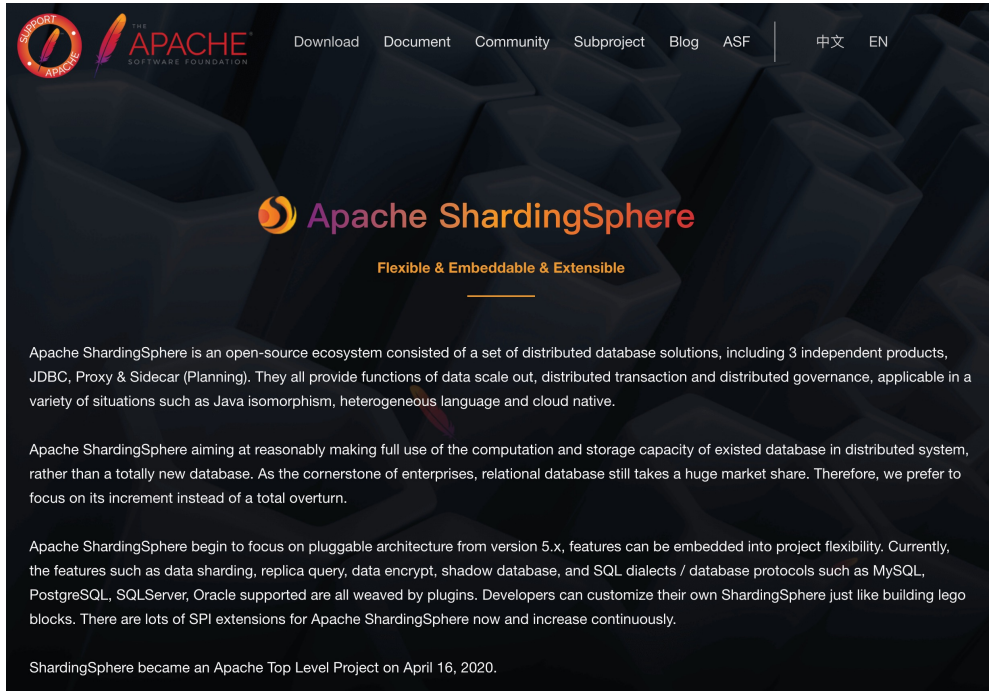
```
ds_write_1 ::: INSERT INTO t_order_1(order_id, user_id, content, user_cipher, user_plain) VALUES (1, 1, 'TEST1', 'kLjLJIMnfyHT2nA+viaoaQ==', '12345678900')
ds_write_0 ::: INSERT INTO t_order_0(order_id, user_id, content, user_cipher, user_plain) VALUES (2, 2, 'TEST2', 'OedX4HEwl3VOz1CMaCWz0g==', '09876543211')

ds_read_1 ::: SELECT `t_order_1`.`order_id`, `t_order_1`.`user_id`, `t_order_1`.`content`, `t_order_1`.`user_cipher` AS telephone FROM t_order_1 WHERE order_id = 1 AND
    user_id = 1
```

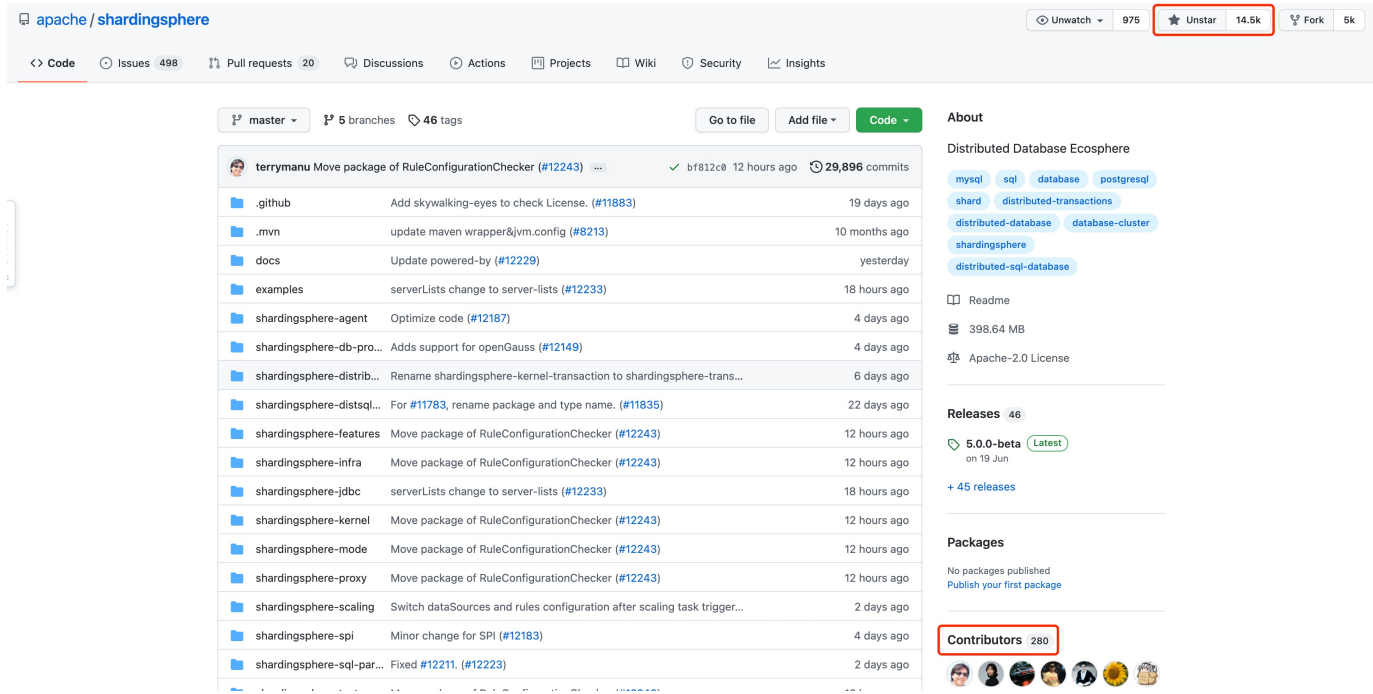


目录

04 社区 Community

社区-活跃度





官网: https://shardingsphere.apache.org
GitHub: https://github.com/apache/shardingsphere

社区-使用案例

https://shardingsphere.apache.org/community/en/powered-by/

社区-技术交流

https://shardingsphere.apache.org/

https://github.com/apache/shardingsphere

mailto:dev-subscribe@shardingsphere.apache.org

SphereEx | 让数据和应用的连接如此简单

谢谢观看

端正强
duanzhengqiang@apache.org

关注我们