PCC POSTGRESCONF CN 2021 | PGConf.Asia 12.14-17

PostgreSQL 增量服务生态实践

端正强
SphereEx 高级中间件工程师

https://2021.postgresconf.cn

个人简介

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 ，曾主导公司内部海量数据的分库分表，在数据库、中间件的开发等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；

CONTENT

PART 01

ShardingSphere 简介

ShardingSphere 简介

ShardingSphere 是一款开源的数据服务能力增强引擎，提供数据分片、分布式事务、数据安全等能力。秉承 Database Plus 理念，旨在构建异构数据库上层的服务标准和生态。

ShardingSphere Apache 2.0 License

Apache 基金会顶级开源项目

- 15000+ Stars
- 300+ Contributors
- 180+ Modules
- 5000+ Forks

ShardingSphere 简介 - Database Plus

产品定位：

- 构建异构数据库的上层标准和生态；
- 提供精准化和差异化的能力；

核心能力：连接、增量、可插拔

ShardingSphere 简介 - Database Plus

连接

通过对数据库协议、SQL 方言以及数据库存储的灵活适配，快速的连接应用与多模式的异构数据库。

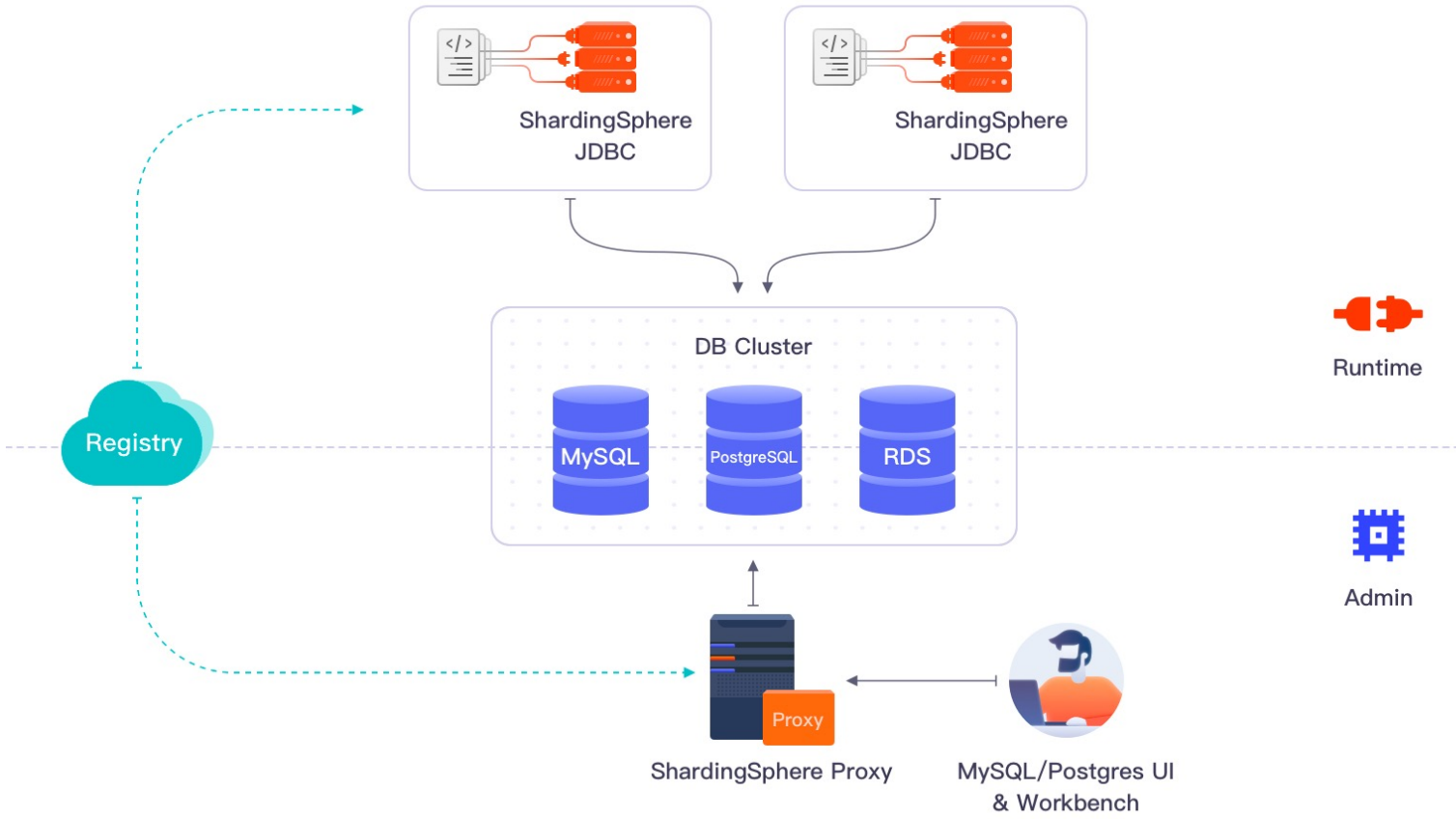
增量

获取数据库的访问流量，并提供流量重定向（数据分片、读写分离、影子库）、流量变形（数据加密、数据安全）、流量鉴权（安全、审计、权限）、流量治理（熔断、限流）以及流量分析（服务质量分析、可观察性）等透明化增量功能。

可插拔

项目采用微内核 + 三层可插拔模型，使内核、功能组件以及生态对接完全能够灵活的方式进行插拔式扩展，开发者能够像使用积木一样定制属于自己的独特系统。

ShardingSphere 简介 - 部署架构



- ShardingSphere 由 JDBC、Proxy 和 Sidecar（规划中）三个接入端组成，既支持独立部署，又可以混合部署；
- 三个接入端均提供标准化的基于数据库作为存储节点的增量功能，可适用于如 Java 同构、异构语言、云原生等各种多样化的应用场景；

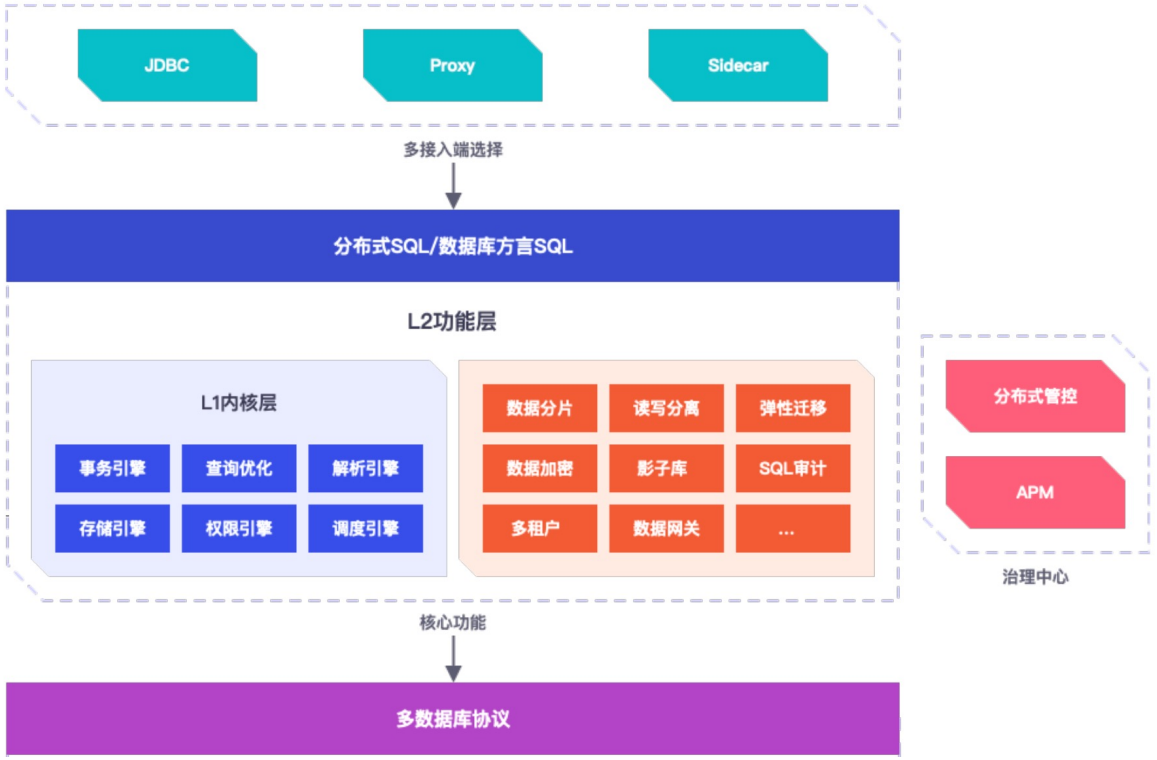
ShardingSphere 简介 - 接入端对比

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

PART 02

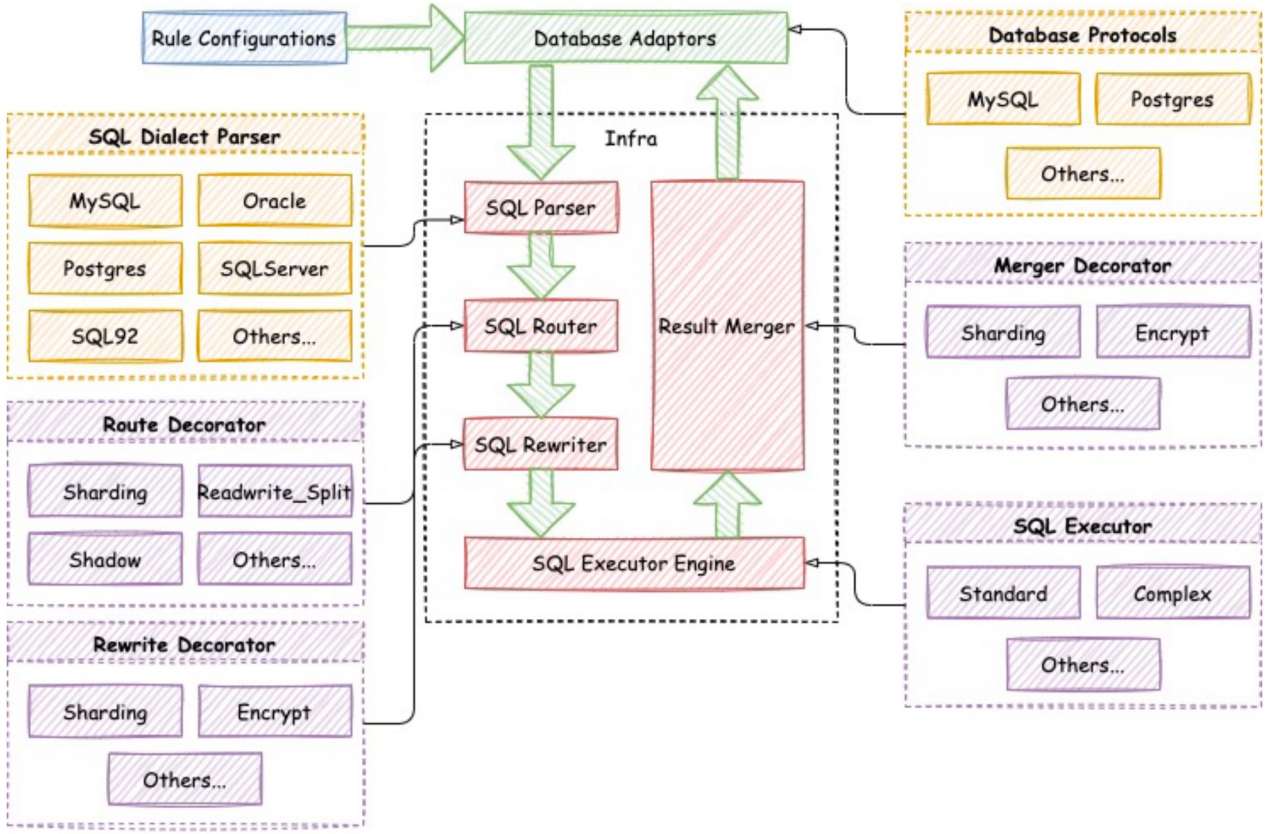
ShardingSphere 可插拔架构内核

ShardingSphere 可插拔架构内核 - 三层架构



- L1 内核层：面向数据库内核，包括数据库事务引擎，查询优化器等；
- L2 功能层：ShardingSphere 最核心所在，可定制化开发平台。具有高定制化、高度内聚、灵活扩展等特点；
- L3 生态层：通过三个接口分别实现数据库协议、 SQL 方言和数据库存储对接，用于打造异构数据网关；

ShardingSphere 可插拔架构内核 - 内核流程



- 内核流程中的元数据加载、SQL 解析、SQL 路由 、SQL 改写、SQL 执行和结果归并，都提供了丰富的扩展点；
- 默认实现了数据分片、读写分离、数据加密、影子库压测、及高可用等功能；
- 用户可以基于扩展点进行定制，打造属于自己的分布式数据库系统；

ShardingSphere 可插拔架构内核 - 解决方案

分布式数据库	数据安全	数据库网关	全链路压测
数据分片	数据加密	异构数据库支持	影子库
读写分离	行级权限（TODO）	SQL 方言转换（TODO）	可观测性
分布式事务	SQL 审计（TODO）		
弹性伸缩	SQL 防火墙（TODO）		
高可用			

PART 03

PostgreSQL 数据分片 & 加密实战

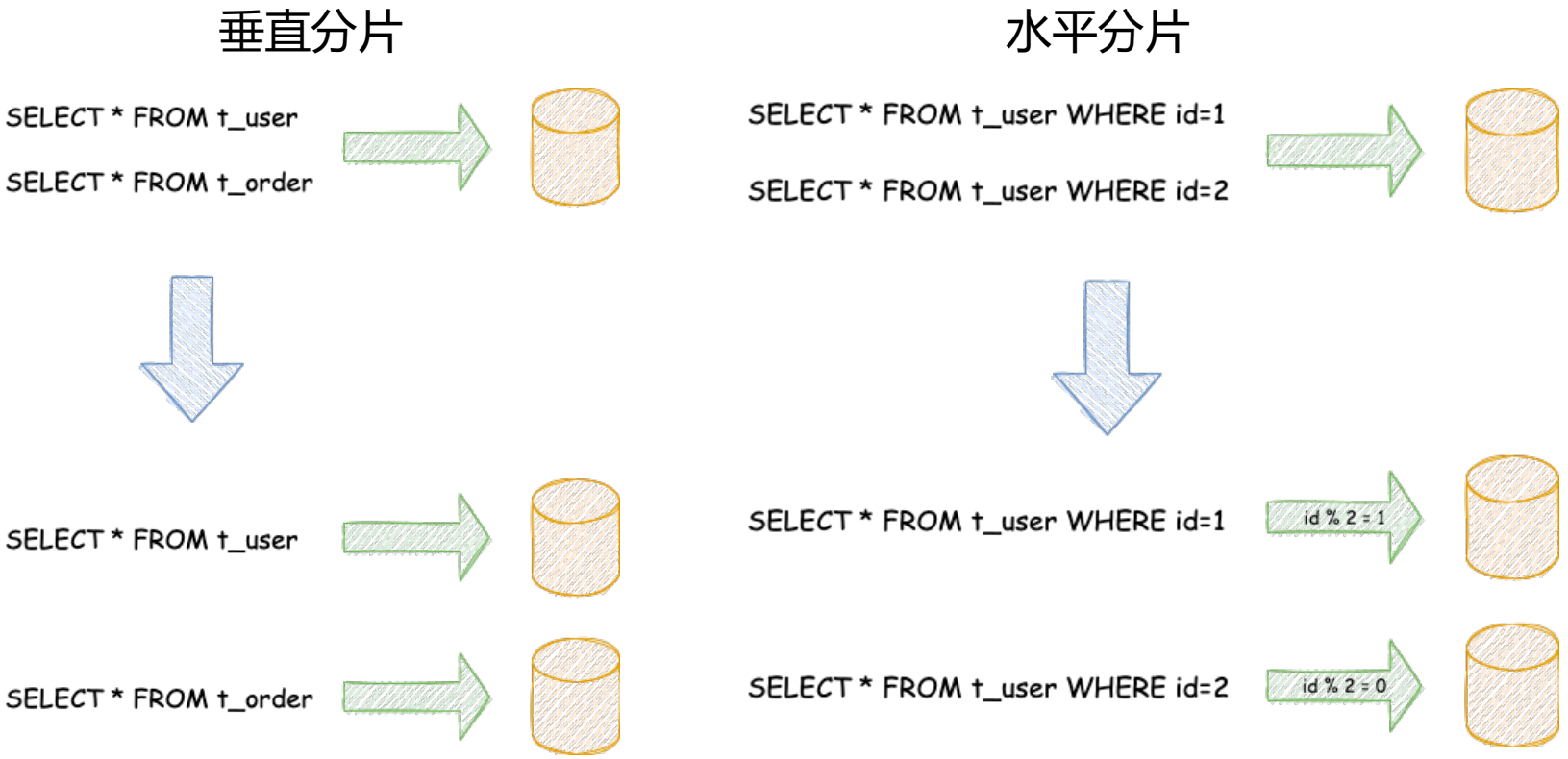
PostgreSQL 数据分片 & 加密实战 - 业务挑战

- **挑战一：**信息化时代背景下，数据呈现爆炸式增长的趋势。将数据集中存储至单一 PostgreSQL 节点的解决方案，在性能、可用性和运维成本这三方面已经难于满足海量数据的场景。PostgreSQL 如何应对海量数据的存储和扩容？

- **挑战二：**数据安全一直是企业极为重视和敏感的话题，当前环境下，数据安全问题日趋严重，PostgreSQL 如何灵活透明地为用户提供加密服务？

解决方案：基于 Apache ShardingSphere DatabasePlus 提供的连接、增量、可插拔的核心能力，整合 Apache ShardingSphere 数据分片&加密功能，实现数据安全的 PostgreSQL 分布式数据库。

PostgreSQL 数据分片 & 加密实战 - 数据分片



- 数据分片：指将存储在单个数据库中的数据按照一定的规则拆分成多个数据库或表，从而提高性能和可用性，通常可以分为垂直分片和水平分片；
- Apache ShardingSphere 数据分片模块为用户提供了透明化分片功能，允许用户像使用原生数据库一样使用分片数据库集群；

PostgreSQL 数据分片 & 加密实战 - 数据分片

- 透明化地 SQL 执行，用户面向逻辑 SQL 进行交互；

- 内置丰富的分片算法：
  - 自动分片算法——取模分片、哈希取模取模分片、基于分片容量的范围取模分片、基于分片边界的范围分片、自动时间段分片；
  - 标准分片算法——行表达式分片、时间范围分片；
  - 复合分片算法——复合行表达式分片；
  - Hint 分片算法——Hint 行表达式分片；
  - 自定义类分片算法；

- 内置多种分布式 ID 生成算法：
  - UUID；
  - SNOWFLAKE；

```
logic_sql: SELECT * FROM t_order WHERE order_id = 1;
actual_sql: SELECT * FROM ds_1.t_order_1 WHERE order_id = 1;
```

PostgreSQL 数据分片 & 加密实战 - 数据分片

```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order_inline
      keyGenerateStrategy:
          column: order_id
          keyGeneratorName: snowflake
    t_order_item:
      actualDataNodes: ds_${0..1}.t_order_item_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order_item_inline
      keyGenerateStrategy:
        column: order_item_id
        keyGeneratorName: snowflake
  bindingTables:
    - t_order,t_order_item
  defaultDatabaseStrategy:
    standard:
      shardingColumn: user_id
      shardingAlgorithmName: database_inline
  defaultTableStrategy:
    none:
```

数据节点，是分片的最小存储单元，记录了逻辑表与真实表的映射关系。

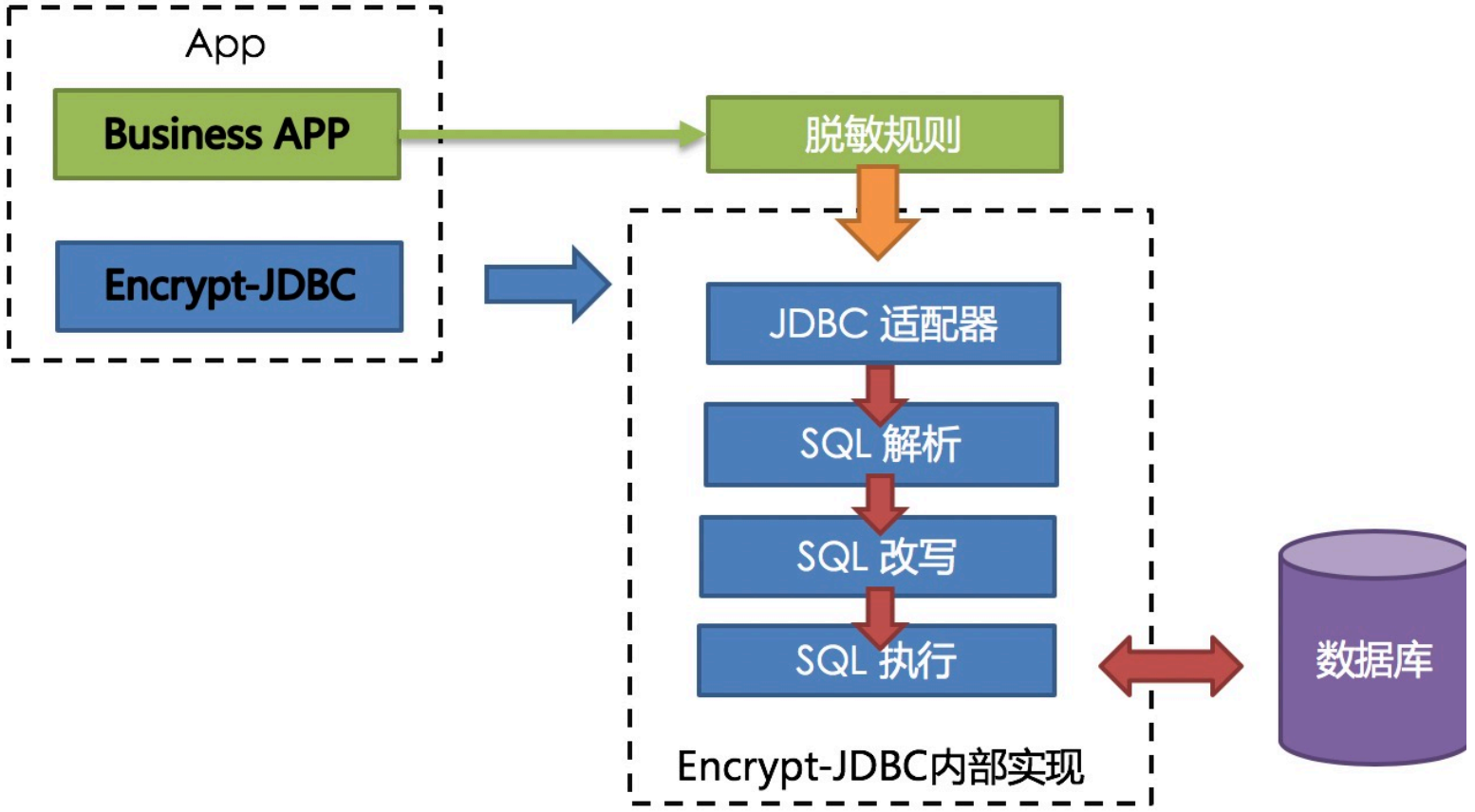
分布式 ID 生成

分片策略

分片算法

- t_order 对应数据分片的逻辑表，用户面向逻辑表编写 SQL，无需关心底层的真实表；
- actualDataNodes：数据节点，是数据分片的最小存储单元，记录了逻辑表和真实表的映射关系；
- tableStrategy 和 databaseStrategy 对应分片策略，包含了分片键和分片算法；
- keyGenerateStrategy 对应分布式 ID 生成策略，支持 SNOWFLAKE、UUID 及用户自定义生成算法；

PostgreSQL 数据分片 & 加密实战 - 数据加密



- 数据加密：通过加密规则对敏感信息进行转换，从而安全地保护隐私数据；
- 数据加密功能包含了 SQL 解析、SQL 改写、SQL 执行和结果归并等流程；
- Apache ShardingSphere 数据加密模块的为用户提供了安全透明的数据加密解决方案；

PostgreSQL 数据分片 & 加密实战 - 数据加密

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

加密器

明文列

密文列

- encryptors 属性用于配置加密算法，Apache ShardingSphere 内置了 AES、MD5 和 RC4 三种加密算法，也允许用户实现自定义加密算法；
- tables 属性用于配置加密表，plainColumn 对应明文列，用于切换过程中的明文查询，cipherColumn 对应密文列；
- 用户面向逻辑列 content 编写 SQL，Apache ShardingSphere 负责对逻辑列进行改写，并使用加密算法自动进行加解密处理；

PostgreSQL 数据分片 & 加密实战 - 功能使用

DDL 示例：

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

逻辑表　　逻辑列

```
Logic SQL: CREATE TABLE t_order(order_id INT PRIMARY KEY, user_id INT, content VARCHAR(100));
Actual SQL: ds_1 ::: CREATE TABLE t_order_0(order_id INT PRIMARY KEY, user_id INT, content_cipher VARCHAR(100), content_plain VARCHAR(100));
Actual SQL: ds_1 ::: CREATE TABLE t_order_1(order_id INT PRIMARY KEY, user_id INT, content_cipher VARCHAR(100), content_plain VARCHAR(100));
Actual SQL: ds_0 ::: CREATE TABLE t_order_0(order_id INT PRIMARY KEY, user_id INT, content_cipher VARCHAR(100), content_plain VARCHAR(100));
Actual SQL: ds_0 ::: CREATE TABLE t_order_1(order_id INT PRIMARY KEY, user_id INT, content_cipher VARCHAR(100), content_plain VARCHAR(100));

Logic SQL: CREATE TABLE t_user(user_id INT PRIMARY KEY, telephone VARCHAR(100));
Actual SQL: ds_1 ::: CREATE TABLE t_user(user_id INT PRIMARY KEY, telephone_cipher VARCHAR(100), telephone_plain VARCHAR(100));
```

PostgreSQL 数据分片 & 加密实战 - 功能使用

DML 示例：

INSERT INTO t_order(order_id, user_id, content) VALUES(1, 1, 'TEST11'), (2, 2, 'TEST22'), (3, 3, 'TEST33');
INSERT INTO t_user(user_id, telephone) VALUES(1, '11111111111'), (2, '22222222222'), (3, '33333333333');
SELECT * FROM t_order o INNER JOIN t_order_item i ON o.order_id = i.order_id WHERE o.user_id = 1 AND o.order_id = 1;

```
Logic SQL: INSERT INTO t_order(order_id, user_id, content) VALUES(1, 1, 'TEST11'), (2, 2, 'TEST22'), (3, 3, 'TEST33');
Actual SQL: ds_1 ::: INSERT INTO t_order_1(order_id, user_id, content_cipher, content_plain) VALUES(1, 1, '3qpLpG5z6AWjRX2sRKjW2g==', 'TEST11'
    ), (3, 3, 'oVkQieUbS3l/85axrf5img==', 'TEST33');
Actual SQL: ds_0 ::: INSERT INTO t_order_0(order_id, user_id, content_cipher, content_plain) VALUES(2, 2, 'mzIhTs2MD3dI4fqCc5nF/Q==', 'TEST22'
    );

Logic SQL: INSERT INTO t_user(user_id, telephone) VALUES(1, '11111111111'), (2, '22222222222'), (3, '33333333333');
Actual SQL: ds_1 ::: INSERT INTO t_user(user_id, telephone_cipher, telephone_plain) VALUES(1, 'jFZBCI7G9ggRktThmMlClQ==', '11111111111'), (2,
    'lWrg5gaes8eptaQkUM2wtA==', '22222222222'), (3, 'jeCwC7gXus4/1UflxeGW/w==', '33333333333');

Logic SQL: SELECT * FROM t_order o INNER JOIN t_order_item i ON o.order_id = i.order_id WHERE o.user_id = 1 AND o.order_id = 1;
Actual SQL: ds_1 ::: SELECT "o"."order_id", "o"."user_id", "o"."content_cipher" AS "content", "i"."item_id", "i"."order_id", "i"."user_id", "i
    "."content" FROM t_order_1 o INNER JOIN t_order_item_1 i ON o.order_id = i.order_id WHERE o.user_id = 1 AND o.order_id = 1;
```

PART 04

PostgreSQL 增量服务生态规划

PostgreSQL 增量服务生态规划-功能优化

- 提升 Apache ShardingSphere 对 PostgreSQL SQL 解析支持度；
- 提升 Apache ShardingSphere Proxy 接入端协议层面对 PostgreSQL 的支持度；
- 优化 Apache ShardingSphere Proxy 接入端的性能；
- 基于 Apache ShardingSphere 可插拔架构，赋予 PostgreSQL 更多能力，例如：SQL 审计、高可用等；

PostgreSQL 增量服务生态规划-社区建设

项目地址： https://shardingsphere.apache.org

GitHub 地址：https://github.com/apache/shardingsphere

开发者邮件列表：dev-subscribe@shardingsphere.apache.org

中文社区： https://community.sphere-ex.com

技术干货

技术交流

THANK YOU

CONTACT INFORMATION

CHINA POSTGRESQL ASSOCIATION | PCC POSTGRESCONF CN 2021 | PGConf.Asia 12.14-17