一、ShardingProxy快速使用

ShardingProxy的功能同样是分库分表，但是他是一个独立部署的服务端，提供统一的数据库代理服务。注意，ShardingProxy目前只支持MySQL和PostgreSQL。并且，客户端连接ShardingProxy时，最好使用MySQL的JDBC客户端。下面我们来部署一个ShardingProxy服务。

1、ShardingProxy部署

ShardingProxy在windows和Linux上提供了一套统一的部署发布包。我们可以从ShardingSphere官网下载4.1.1版本的ShardingProxy发布包apache-shardingsphere-4.1.1-sharding-proxy-bin.tar.gz，解压到本地目录。配套资料中已经提供

> 注意不要有中文路径

首先，我们需要把MySQL的JDBC驱动包mysql-connector-java-8.0.20.jar手动复制到ShardingProxy的lib目录下。ShardingProxy默认只附带了PostgreSQL的JDBC驱动包，而不包含MySQL的JDBC驱动包。

然后，我们需要到conf目录下，修改server.yaml，将配置文件中的authentication和props两段配置的注释打开。

```
authentication:
  users:
    root:
      password: root
    sharding:
```

```
      password: sharding
      authorizedSchemas: sharding_db

props:
  max.connections.size.per.query: 1
  acceptor.size: 16  # The default value is available processors count * 2.
  executor.size: 16  # Infinite by default.
  proxy.frontend.flush.threshold: 128  # The default value is 128.
    # LOCAL: Proxy will run with LOCAL transaction.
    # XA: Proxy will run with XA transaction.
    # BASE: Proxy will run with B.A.S.E transaction.
  proxy.transaction.type: LOCAL
  proxy.opentracing.enabled: false
  proxy.hint.enabled: false
  query.with.cipher.column: true
  sql.show: false
  allow.range.query.with.inline.sharding: false
```

然后，我们修改conf目录下的config-sharding.yaml，这个配置文件就是shardingProxy关于分库分表部分的配置。整个配置和之前我们使用ShardingJDBC时的配置大致相同，我们在最下面按照自己的数据库环境增加以下配置：

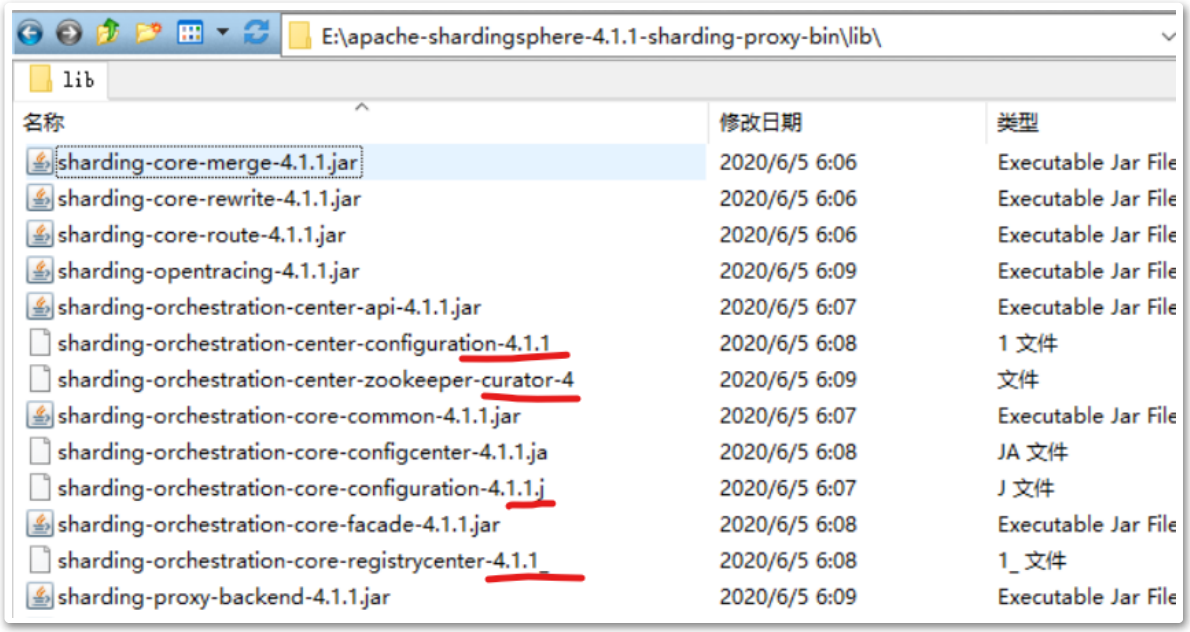
```
schemaName: sharding_db

dataSources:
  m1:
    url: jdbc:mysql://localhost:3306/userdb?serverTimezone=GMT%2B8&useSSL=false
    username: root
    password: root
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50

shardingRule:
  tables:
    course:
      actualDataNodes: m1.course_$->{1..2}
      tableStrategy:
        inline:
          shardingColumn: cid
          algorithmExpression: course_$->{cid%2+1}
      keyGenerator:
        type: SNOWFLAKE
        column: cid
```

> 这一段就是按照我们之前的application01.properties文件中的规则配置的。可以看到，整个配置其实是大同小异的。

然后，还一个小问题要注意，我们进入ShardingProxy的Lib目录，里面会有些jar包因为名字太长了，导致有些文件的后缀被截断了，我们要手动把他们的文件后缀给修改过来。



然后，我们就可以启动ShardingProxy的服务了。启动脚本在bin目录下。其中，windows平台对应的脚本是start.bat，Linux平台对应的脚本是start.sh和stop.sh

启动时，我们可以直接运行start.bat脚本，这时候，ShardingProxy默认占用的是3307端口。为了不跟我们之前搭建的多个MySQL服务端口冲突，我们定制下启动端口，改为3316端口。

```
start.bat 3316
```

> 为什么windows平台上没有stop.bat呢？因为start.bat会独占一个命令行窗口，把命令行窗口关闭，就停止了ShardingProxy的服务。

启动完成后，可以看到几行关键的日志标识服务启动成功了。

```
[INFO ] 10:46:53.930 [main] c.a.d.xa.XATransactionalResource - resource-1-m1: refreshed XAResource
[INFO ] 10:46:54.580 [main] ShardingSphere-metadata - Loading 1 logic tables' meta data.
[INFO ] 10:46:54.717 [main] ShardingSphere-metadata - Loading 8 tables' meta data.
[INFO ] 10:46:56.953 [nioEventLoopGroup-2-1] i.n.handler.logging.LoggingHandler - [id: 0xc90e0eef] REGISTERED
[INFO ] 10:46:56.958 [nioEventLoopGroup-2-1] i.n.handler.logging.LoggingHandler - [id: 0xc90e0eef] BIND: 0.0.0.0/0.0.0.0:3316
[INFO ] 10:46:56.960 [nioEventLoopGroup-2-1] i.n.handler.logging.LoggingHandler - [id: 0xc90e0eef, L:/0:0:0:0:0:0:0:0:3316] ACTIVE
```

2、ShardingProxy使用

这样，我们就可以像连接一个标准MySQL服务一样连接ShardingProxy了。

```
D:\dev-hook\mysql-8.0.20-winx64\bin>mysql.exe -P3316 -uroot -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 8.0.20-Sharding-Proxy 4.1.0

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-------------+
| Database    |
+-------------+
| sharding_db |
+-------------+
1 row in set (0.03 sec)

mysql> use sharding_db
Database changed
mysql> show tables;
+--------------------+
```

```
| Tables_in_coursedb |
+--------------------+
| course             |
| t_dict             |
+--------------------+
2 rows in set (0.16 sec)

mysql> select * from course;
+--------------------+-------+---------+---------+
| cid                | cname | user_id | cstatus |
+--------------------+-------+---------+---------+
| 545703330389118976 | java  |    1001 | 1       |
| 545703330804355072 | java  |    1001 | 1       |
| 545703330842103808 | java  |    1001 | 1       |
| 545703330879852544 | java  |    1001 | 1       |
| 545703330917601280 | java  |    1001 | 1       |
+--------------------+-------+---------+---------+
5 rows in set (0.08 sec)
```

> 之前在ShardingJDBC部分完成了的其他几种分库分表策略以及读写分离策略，就请大家自行验证了。

3、ShardingProxy的服务治理

从ShardingProxy的server.yaml中看到，ShardingProxy还支持非常多的服务治理功能。在server.yaml配置文件中的orchestration部分属性就演示了如何将ShardingProxy注册到Zookeeper当中。

```yaml
orchestration:
  orchestration_ds:
    orchestrationType:
registry_center,config_center,distributed_lock_manager
    instanceType: zookeeper
    serverLists: localhost:2181
    namespace: orchestration
    props:
      overwrite: false
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
      maxRetries: 3
      operationTimeoutMilliseconds: 500
```

ShardingSphere在服务治理这一块主要有两个部分：

一是数据接入以及弹性伸缩。简单理解就是把MySQL或者其他数据源的数据快速迁移进ShardingSphere的分片库中。并且能够快速的对已有的ShardingShere分片库进行扩容以及减配。这一块由ShardingSphere-scaling产品来提供支持。只是这个功能在目前的4.1.1版本中，还处于Alpha测试阶段。

另一方面，ShardingSphere支持将复杂的分库分表配置上传到统一的注册中心中集中管理。目前支持的注册中心有Zookeeper和Etcd。而ShardingSphere也提供了SPI扩展接口，可以快速接入Nacos、Apollo等注册中心。在ShardingProxy的server.yaml中我们已经看到了这一部分的配置示例。

另外，ShardingSphere针对他的这些生态功能，提供了一个ShardingSphere-UI产品来提供页面支持。ShardingSphere-UI是针对整个ShardingSphere的一个简单有用的Web管理控制台。它用于帮助用户更简单的使用ShardingSphere的相关功能。目前提供注册中心管理、动态配置管理、数据库编排管理等功能。

> 配套资料中也收集了ShardingSphere-UI的最新版本5.0.0-alpha版的运行包。解压后执行其中的start.bat就可以直接运行。

4、Shardingproxy的其他功能

影子库

这部分功能主要是用于进行压测的。通过给生产环境上的关键数据库表配置一个影子库，就可以将写往生产环境的数据全部转为写入影子库中，而影子库通常会配置成跟生产环境在同一个库，这样就可以在生产环境上直接进行压力测试，而不会影响生产环境的数据。

在conf/config-shadow.yaml中有配置影子库的示例。其中最核心的就是下面的shadowRule这一部分。

```
#shadowRule:
#  column: shadow
#  shadowMappings:
# 绑定shadow_ds为ds的影子库
#    ds: shadow_ds
```

数据加密

在conf/config-encrypt.yaml中还演示了ShardingProxy的另一个功能，数据加密。默认集成了AES对称加密和MD5加密。还可以通过SPI机制自行扩展更多的加密算法。

5、ShardingProxy的SPI扩展

上一部分提到了ShardingSphere保留了大量的SPI扩展接口，对主流程封闭、对SPI开放。这在ShardingJDBC中还体现不出太大的作用，但是在ShardingProxy中就能极大程度提高服务的灵活性了。

在ShardingProxy中，只需要将自定义的扩展功能按照SPI机制的要求打成jar包，就可以直接把jar包放入lib目录，然后就配置使用了。

例如如果想要扩展一个新的主键生成策略，只需要自己开发一个主键生成类

```
package com.roy.shardingDemo.spiextention;

import org.apache.shardingsphere.spi.keygen.ShardingKeyGenerator;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicLong;

/**
 * @author : 楼兰
 * @date : Created in 2020/12/17
 * @description:
 **/

public final class MykeyGenerator implements ShardingKeyGenerator {

    private AtomicLong atom = new AtomicLong(0);

    private Properties properties = new Properties();

    public synchronized Comparable<?> generateKey() {
        //读取了一个自定义属性
        String prefix = properties.getProperty("mykey-offset", "100");
        LocalDateTime ldt = LocalDateTime.now();
        String timestampS =
DateTimeFormatter.ofPattern("HHmmssSSS").format(ldt);
        return Long.parseLong(""+prefix+timestampS+atom.incrementAndGet());
    }
    //扩展算法的类型
```

```
30      public String getType() {
31          return "MYKEY";
32      }
33
34      public Properties getProperties() {
35          return this.properties;
36      }
37
38      public void setProperties(Properties properties) {
39          this.properties = properties;
40      }
41  }
```

然后增加一个META-INF\services\org.apache.shardingsphere.spi.keygen.ShardingKeyGenerator文件，并在文件中写明自己的实现类。

`com.roy.shardingDemo.spiextention.MykeyGenerator` 将扩展类和这个SPI服务文件一起打成jar包，就可以直接放到ShardingProxy的lib目录下。

接下来就可以在config-sharding.yaml中以类似下面这种配置方式引入了。

```
shardingRule:
  tables:
    course:
      actualDataNodes: m1.course_$->{1..2}
      tableStrategy:
        inline:
          shardingColumn: cid
          algorithmExpression: course_$->{cid%2+1}
      keyGenerator:
#         type: SNOWFLAKE
        type: MYKEY # 自定义的主键生成器
        column: cid
```

然后我们可以启动ShardingProxy，试试我们自定义的主键生成器。

```
mysql> select * from course;
+--------------------+-------+---------+---------+
| cid                | cname | user_id | cstatus |
+--------------------+-------+---------+---------+
|                222 | java2 |    1002 | 1       |
| 545730330389118976 | java  |    1001 | 1       |
| 545730330804355072 | java  |    1001 | 1       |
| 545730330842103808 | java  |    1001 | 1       |
| 545730330879852544 | java  |    1001 | 1       |
| 545730330917601280 | java  |    1001 | 1       |
+--------------------+-------+---------+---------+
6 rows in set (0.01 sec)

mysql> insert into course(cname,user_id,cstatus) values ('java2',1002,'1');
Query OK, 1 row affected (0.11 sec)

mysql> insert into course(cname,user_id,cstatus) values ('java2',1003,'1');
Query OK, 1 row affected (0.01 sec)

mysql> select * from course;
+--------------------+-------+---------+---------+
| cid                | cname | user_id | cstatus |
+--------------------+-------+---------+---------+
|                222 | java2 |    1002 | 1       |
|      1001509178012 | java2 |    1003 | 1       |
| 545730330389118976 | java  |    1001 | 1       |
| 545730330804355072 | java  |    1001 | 1       |
| 545730330842103808 | java  |    1001 | 1       |
| 545730330879852544 | java  |    1001 | 1       |
| 545730330917601280 | java  |    1001 | 1       |
|      1001509119631 | java2 |    1002 | 1       |
+--------------------+-------+---------+---------+
8 rows in set (0.01 sec)
```

从结果可以看到，插入的两条记录，自动生成的CID分别为1001509178012、1001509119631。这样我们就很快的完成了一个自定义的主键生成策略。

> 关于ShardingSphere的SPI扩展点，在配套资料《shardingsphere_docs_cn.pdf》的开发者手册部分有更全面详细的梳理。

二、ShardingSphere总结

我们现在已经学完了ShardingSphere除了Sharding-SideCar以外的所有产品了，整个sharding + proxy的所有这些功能，本质上其实都只解决了一个问题，就是单机数据库容量的问题。在软件层面对硬件资源进行管理，从而便于对数据库的横向扩展。

但是，我们也要意识到他带来的很多问题。

例如对业务的侵入大。业务系统写的SQL将不再是纯粹的能在服务器上运行的SQL了，对大量跨维度的JOIN、聚合、子查询、排序等功能在业务上很难进行验证。这必然会弱化数据库的功能。

并且，使用ShardingSphere管理后，数据库之间变成了结合非常紧密的依赖关系，对整个集群的扩容也会带来相当大的难度。

另外，ShardingSphere这种方式实际上将原本由业务管理SQL的工作方式，转化成了由业务管理逻辑SQL，而运维管理实际SQL的混合工作模式，再加上一大堆服务的引入，整个服务运维的维护工作量以及工作难度也上升了非常多。

当然，相信随着ShardingSphere后续版本的不断升级优化，这些问题都会得到不同程度的改善。

三、与其他相关产品的对比

业界组件	原厂	功能特性	备注
DBLE	爱可生开源社区	专注于mysql的高可扩展性的分布式中间件	基于MyCAT开发出来的增强版。
Meituan Atlas	美团	读写分离、单库分表	目前已经在原厂逐步下架。
Cobar	阿里（B2B）	Cobar 中间件以 Proxy 的形式位于前台应用和实际数据库之间，对前台的开放的接口是 MySQL 通信协议	开源版本中数据库只支持 MySQL，并且不支持读写分离。
MyCAT	阿里	是一个实现了 MySQL 协议的服务器，前端用户可以把它看作是一个数据库代理，用 MySQL 客户端工具和命令行访问，而其后端可以用MySQL 原生协议与多个 MySQL 服务器通信	MyCAT 基于阿里开源的 Cobar 产品而研发
Atlas	360	读写分离、静态分表	

业界组件	原厂	功能特性	备注
Kingshard		由 C 语言开发高性能 MySQL Proxy 项目，在满足基本的读写分离的功能上，Kingshard 的性能是直连 MySQL 性能的80%以上。	
TDDL	阿里淘宝	动态数据源、读写分离、分库分表	TDDL 分为两个版本, 一个是带中间件的版本, 一个是直接 JAVA library 的版本。
Zebra	美团点评	实现动态数据源、读写分离、分库分表、CAT监控	功能齐全且有监控，接入复杂、限制多。
MTDDL	美团点评	动态数据源、读写分离、分布式唯一主键生成器、分库分表、连接池及SQL监控	
Vitess	谷歌、Youtube	集群基于ZooKeeper管理，通过RPC方式进行数据处理，总体分为，server，command line，gui监控 3部分	Youtube 大量应用
DRDS	阿里	DRDS（Distributed Relational Database Service）专注于解决单机关系型数据库扩展性问题，具备轻量(无状态)、灵活、稳定、高效等特性，是阿里巴巴集团自主研发的中间件产品。	

> 有道云笔记分享链接：
>
> 文档：VIP04-ShardingProxy分库分表实战及同?..
> 链接：http://note.youdao.com/noteshare?id=5523f907a314692c87a24764e6f948d6&sub=F7EFFB349A6947B88C89FAC9CDC7D529