课程内容

1、sharding-proxy的安装与启动

2、sharding-proxy的基本特性

3、ShardingSphere&Atlas&Mycat对比

4、分库分表影响

shardingsphere-proxy安装与启动

下载shardingsphere-4.0.0-RC1版本

http://mirrors.tuna.tsinghua.edu.cn/apache/incubator/shardingsphere/4.0.0-RC2/apache-shardingsphere-incubating-4.0.0-RC2-sharding-proxy-bin.tar.gz

```
解压
tar -zxvf apache-shardingsphere-incubating-4.0.0-RC2-sharding-proxy-
  bin.tar.gz
```

解压后的目录：

```
[root@192 apache-shardingsphere-incubating-4.0.0-RC2-sharding-proxy-bin]# ls
bin  conf  DISCLAIMER  lib  LICENSE  licenses  NOTICE  README.txt
```

引入依赖

如果后端连接MySQL数据库，需要手动下载mysql驱动jar包，并将将mysql-connector-java-${version}.jar拷贝到${sharding-proxy}\lib目录

启动配置

```
规则配置-可根据需要选择
编辑%SHARDING_PROXY_HOME%\conf\config-xxx.yaml
(config-sharding.yaml-数据分片设置,参考sharding-jdbc)
(config-master_slave.yaml-主从读写分离设置,参考sharding-jdbc)
(config-encrypt.yaml-数据脱敏设置,参考sharding-jdbc)
全局配置：注册中心、认证信息以及公用属性
编辑%SHARDING_PROXY_HOME%\conf\server.yaml
```

数据分片 config-sharding.yaml

```
schemaName: sharding_db #逻辑数据源名称


dataSources:
ds0:
url: jdbc:mysql://127.0.0.1:3306/ds0?serverTimezone=UTC&useSSL=false
```

```
username: root
password:
connectionTimeoutMilliseconds: 30000 //连接超时毫秒数
idleTimeoutMilliseconds: 60000 //空闲连接回收超时毫秒数
maxLifetimeMilliseconds: 1800000 //连接最大存活时间毫秒数
maxPoolSize: 65 //最大连接数
ds1:
url: jdbc:mysql://127.0.0.1:3306/ds1?serverTimezone=UTC&useSSL=false
username: root
password:
connectionTimeoutMilliseconds: 30000
idleTimeoutMilliseconds: 60000
maxLifetimeMilliseconds: 1800000
maxPoolSize: 65
#同sharding-jdbc设置
shardingRule:
tables:
t_order:
actualDataNodes: ds${0..1}.t_order${0..1}
databaseStrategy:
inline:
shardingColumn: user_id
algorithmExpression: ds${user_id % 2}
tableStrategy:
inline:
shardingColumn: order_id
algorithmExpression: t_order${order_id % 2}
keyGenerator:
type: SNOWFLAKE
column: order_id
t_order_item:
actualDataNodes: ds${0..1}.t_order_item${0..1}
databaseStrategy:
inline:
shardingColumn: user_id
algorithmExpression: ds${user_id % 2}
tableStrategy:
inline:
shardingColumn: order_id
algorithmExpression: t_order_item${order_id % 2}
keyGenerator:
type: SNOWFLAKE
column: order_item_id
bindingTables:
```

```
- t_order,t_order_item
defaultTableStrategy:
none:
```

读写分离 config-master_slave.yaml

```
schemaName: master_slave_db

dataSources:
ds_master:
url: jdbc:mysql://127.0.0.1:3306/ds_master?serverTimezone=UTC&useSSL=false
username: root
password:
connectionTimeoutMilliseconds: 30000
idleTimeoutMilliseconds: 60000
maxLifetimeMilliseconds: 1800000
maxPoolSize: 65
ds_slave0:
url: jdbc:mysql://127.0.0.1:3306/ds_slave0?serverTimezone=UTC&useSSL=false
username: root
password:
connectionTimeoutMilliseconds: 30000
idleTimeoutMilliseconds: 60000
maxLifetimeMilliseconds: 1800000
maxPoolSize: 65
ds_slave1:
url: jdbc:mysql://127.0.0.1:3306/ds_slave1?serverTimezone=UTC&useSSL=false
username: root
password:
connectionTimeoutMilliseconds: 30000
idleTimeoutMilliseconds: 60000
maxLifetimeMilliseconds: 1800000
maxPoolSize: 65
#主从路由规则
masterSlaveRule:
name: ds_ms
masterDataSourceName: ds_master
slaveDataSourceNames:
- ds_slave0
- ds_slave1
loadBalanceAlgorithmType: ROUND_ROBIN
```

数据脱敏

```
dataSource: #省略数据源配置

```

```
encryptRule:
encryptors:
<encryptor-name>:
type: #加解密器类型，可自定义或选择内置类型：MD5/AES
props: #属性配置，注意：使用AES加密器，需要配置AES加密器的KEY属性：aes.key.value
aes.key.value:
tables:
<table-name>:
columns:
<logic-column-name>:
plainColumn: #存储明文的字段
cipherColumn: #存储密文的字段
assistedQueryColumn: #辅助查询字段，针对ShardingQueryAssistedEncryptor类型的加
  解密器进行辅助查询
encryptor: #加密器名字
props:
query.with.cipher.column: true #是否使用密文列查询
```

全局配置 server.yaml

Sharding-Proxy使用conf/server.yaml配置注册中心、认证信息以及公用属性。

```
数据治理

orchestration:
name: #治理实例名称
overwrite: #本地配置是否覆盖注册中心配置。如果可覆盖，每次启动都以本地配置为准
registry: #注册中心配置
type: #配置中心类型。如：zookeeper
serverLists: #连接注册中心服务器的列表。包括IP地址和端口号。多个地址用逗号分隔。
  如: host1:2181,host2:2181
namespace: #注册中心的命名空间
digest: #连接注册中心的权限令牌。缺省为不需要权限验证
operationTimeoutMilliseconds: #操作超时的毫秒数，默认500毫秒
maxRetries: #连接失败后的最大重试次数，默认3次
retryIntervalMilliseconds: #重试间隔毫秒数，默认500毫秒
timeToLiveSeconds: #临时节点存活秒数，默认60秒
例:
orchestration:
name: orchestration_ds
overwrite: true
registry:
type: zookeeper
namespace: orchestration
serverLists: localhost:2181
```

```
认证信息(连接到sharding-proxy需要进行身份认证)
authentication:
users:
root: # 自定义用户名
password: root # 自定义用户名
sharding: # 自定义用户名
password: sharding # 自定义用户名
authorizedSchemas: sharding_db, masterslave_db(数据分片中配置的逻辑数据源)
# 该用户授权可访问的数据库，多个用逗号分隔。缺省将拥有root权限，可访问全部数据库。
#省略与Sharding-JDBC一致的配置属性
props:
acceptor.size: #用于设置接收客户端请求的工作线程个数，默认为CPU核数*2
proxy.transaction.type: #默认为LOCAL事务，允许LOCAL，XA，BASE三个值，
XA采用Atomikos作为事务管理器，BASE类型需要拷贝实现ShardingTransactionManager的
  接口的jar包至lib目录中
proxy.opentracing.enabled: #是否开启链路追踪功能，默认为不开启。
check.table.metadata.enabled: #是否在启动时检查分表元数据一致性，默认值: false
proxy.frontend.flush.threshold: # 对于单个大查询,每多少个网络包返回一次
executor.size: 16
sql.show: false (or true)是否打印SQL语句
```

数据治理服务启动特别注意：

```
以下示例：
authentication:
users:
root:
password: root
tuling:
password: tuling
authorizedSchemas: shop_ds_proxy
#
orchestration:
name: tuling_orchestration_proxy
overwrite: false
registry:
type: zookeeper
serverLists: 192.168.241.198:2181
namespace: sharding_zookeeper_proxy
启动sharing-proxy时需要先在zookeeper上创建对应的Node节点如下：
${namespace}/${orchestration.name}/config/authentication
对应上述配置节点Node应为：/sharding_zookeeper_proxy/tuling_orchestration_prox
  y/config/authentication
然后设置该数据节点数据为：
users:
```

```
root:
password: root
tuling:
password: tuling
authorizedSchemas: shop_ds_proxy
如果不设置，则抛节点值null空指针异常，proxy服务无法正常启动
```

Yaml语法说明

!! 表示实例化该类

- 表示可以包含一个或多个

[] 表示数组，可以与减号相互替换使用

启动proxy服务

使用默认配置项启动

```
${sharding-proxy}\bin\start.sh
```

配置端口启动（默认端口3307）

```
${sharding-proxy}\bin\start.sh ${port}
```

停止服务

```
${sharding-proxy}\bin\stop.sh
```

sharding-proxy的基本特性

自定义分片算法:

当用户需要使用自定义的分片算法类时，无法再通过简单的inline表达式在yaml文件进行配置。可通过以下方式配置使用自定义分片算法。

1. 实现ShardingAlgorithm接口定义的算法实现类。
2. 将上述java文件打包成jar包。
3. 将上述jar包拷贝至ShardingProxy解压后的conf/lib目录下。
4. 将上述自定义算法实现类的java文件引用配置在yaml文件里tableRule的algorithmClassName属性上

分布式事务

Sharding-Proxy接入的分布式事务API同Sharding-JDBC保持一致，支持LOCAL，XA，BASE类型的事务。

XA事务

Sharding-Proxy原生支持XA事务，默认的事务管理器为Atomikos。 可以通过在Sharding-Proxy的conf目录中添加jta.properties来定制化Atomikos配置项。

BASE事务

BASE目前没有打包到Sharding-Proxy中，使用时需要将实现了ShardingTransactionManagerSPI的jar拷贝至conf/lib目录，然后切换事务类型为BASE。

SCTL(Sharding-Proxy control language)

SCTL为Sharding-Proxy特有的控制语句，可以在运行时修改和查询Sharding-Proxy的状态，目前支持的语法为：

语句	说明
sctl:set transaction_type=XX	修改当前TCP连接的事务类型, 支持LOCAL，XA，BASE。例：sctl:set transaction_type=XA
sctl:show transaction_type	查询当前TCP连接的事务类型
sctl:show cached_connections	查询当前TCP连接中缓存的物理数据库连接个数
sctl:explain SQL语句	查看逻辑SQL的执行计划，例：sctl:explain select * from t_order;
sctl:hint set MASTER_ONLY=true	针对当前TCP连接，是否将数据库操作强制路由到主库
sctl:hint set DatabaseShardingValue=yy	针对当前TCP连接，设置hint仅对数据库分片有效，并添加分片值，yy：数据库分片值
sctl:hint addDatabaseShardingValue xx=yy	针对当前TCP连接，为表xx添加分片值yy，xx：逻辑表名称，yy：数据库分片值
sctl:hint addTableShardingValue xx=yy	针对当前TCP连接，为表xx添加分片值yy，xx：逻辑表名称，yy：表分片值
sctl:hint clear	针对当前TCP连接，清除hint所有设置
sctl:hint show status	针对当前TCP连接，查询hint状态，master_only:true/false，sharding_type:databases_only/databases_tables
sctl:hint show table status	针对当前TCP连接，查询逻辑表的hint分片值

Sharding-Proxy 默认不支持hint，如需支持，请在conf/server.yaml中，将props的属性proxy.hint.enabled设置为true。在Sharding-Proxy中，HintShardingAlgorithm的泛型只能是String类型。

使用说明：

```
像使用SQL一样正常使用；
可以通过mysql-client工具连接使用
mysql -h 192.168.241.198 -P3308 -u tuling(authentication下配置的user) -ptuli
  ng((authentication下配置的pwd)
登录进来后执行sctl语句，如下图所示
```

```
[root@192 bin]# mysql -h localhost -P3308 -u tuling -ptuling
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'tuling'@'localhost' (using password: YES)
[root@192 bin]# mysql -h 192.168.241.198 -P3308 -u tuling -ptuling
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47
Server version: 5.6.0-Sharding-Proxy 4.0.0-RC2

Copyright (c) 2000, 2019, 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> sctl:show transaction_type;
+------------------+
| TRANSACTION_TYPE |
+------------------+
| LOCAL            |
+------------------+
1 row in set (0.00 sec)
```

注意事项

1. Sharding-Proxy默认使用3307端口，可以通过启动脚本追加参数作为启动端口号。如: bin/start.sh 3308
2. Sharding-Proxy使用conf/server.yaml配置注册中心、认证信息以及公用属性。
3. Sharding-Proxy支持多逻辑数据源，每个以config-前缀命名的yaml配置文件，即为一个逻辑数据源。

ShardingSphere&Atlas&Mycat对比

sharding + proxy

sharding + proxy 分库分表的路由通常在应用层(APP)与DB层之间实现，中间层根据偏向DB还是偏向应用层可分为：DB proxy 和 JDBC proxy。

无论是DB proxy还是Jdbc proxy都有很多开源实现，如下图简单做一个对比

实现方案	业界组件	原厂	功能特性	备注
DB proxy-based 多语言支持	Atlas	360	读写分离、静态分表	
	Meituan Atlas	美团	读写分离、单库分表	目前已经在原逐步下架。
	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	开源版本中数库只支持 MySQL，并且支持读写分离

	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 基于开源的 Coba品而研发
	Heisenberg	百度	热重启配置、可水平扩容、遵守 MySQL 原生协议、无语言限制。	
	Kingshard	Kingshard	由 Go 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。	
JDBC - based 支持多 ORM 框架，一般有语言限制。	TDDL	阿里淘宝	动态数据源、读写分离、分库分表	TDDL 分为两版本, 一个是中间件的版本, 一是直接 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	
DB - based 解决方案	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。	逐渐下沉为D务

ShardingSphere用户群

分库分表成本

Sharding + Proxy 本质上只解决了一个问题，那就是单机数据容量问题，但它有哪些成本呢？前面提了每种 proxy 都有比较大的硬伤。

1、应用限制

1. Sharding 后对应用和 SQL 的侵入都很大，需要 SQL 足够简单，这种简单的应用导致 DB 弱化为存储。
2. SQL 不能跨维度 join、聚合、子查询等。
3. 每个分片只能实现 Local index，不能实现诸如唯一键、外键等全局约束。

2、Sharding 业务维度选择

1. 有些业务没有天然的业务维度，本身选择一个维度就是个问题。
2. 大部分业务需要多维度的支持，多维度的情况下。
    - 哪个业务维度为主？
    - 其它业务维度产生了数据冗余，如果没有全局事务的话，很难保证一致性，全局事务本身实现很难，并且响应时间大幅度下降，业务相互依赖存在重大隐患，于是经常发生“风控把支付给阻塞了”的问题。
    - 多维度实现方式，数据库同步还是异步？同步依赖应用端实现双写，异步存在实效性问题，对业务有限制，会发生“先让订单飞一会的问题”。
    - 多维度数据关系表（mapping）维护。

3、Sharding key 选择（非业务维度选择）

1. 非业务维度选择，会存在“我要的数据到底在那个集群上”的问题。
2. 业务维度列如何选择 Sharding key ？
3. 热点如何均摊，数据分布可能有长尾效应。

4、Sharding 算法选择

1. Hash 算法可以比较好的分散热点数据，但对范围查询需要访问多个分片。反之 Range 算法又存在热点问题。所以要求在设计之初就要清楚自己的业务常用读写类型。
2. 转换算法成本很高。

5、高可用问题

1. 高可用的扩散问题（一个集群不可用，整个业务“不可用”）。
2. 如何应对脑裂的情况？
3. MGR 多主模式数据冲突解决方案不成熟，基本上还没公司接入生产系统。
4. PXC 未解决写入容量，存在木桶原则，降低了写入容量。

5. 第三方依赖，MHA（判断主库真死、新路由信息广播都需要一定的时间成本） 最快也需要 15s。

6. 虽然有 GTID，仍然需要手工恢复。

6、数据一致性（其实这个严格上不属于分库分表的问题，但这个太重要了，不得不说）

1. MySQL 双一方案(redo、binlog 提交持久化) 严重影响了写入性能。
2. 即使双一方案，主库硬盘挂了，由于异步复制，数据还是会丢。
3. 强一致场景需求，比如金融行业，MySQL 目前只能做到双一 + 半同步复制，既然是半同步，随时可能延迟为异步复制，还是会丢数据。
4. MGR ？ 上面说过，多写模式问题很多，距离接入生产系统还很远。
5. InnoDB Cluster ？ 先搞出来再说吧。

7、DB Proxy

1. 依赖网络层（LVS）实现负载均衡，跨 IDC 依赖 DNS，DNS + LVS + DBproxy + MySQL 网络链路过长，延迟增加、最重要的是存在全公司层面网络单点。
2. 部分产品对 Prepare 不友好，需要绑定 connection。

8、JDBC Proxy

1. 语言限制，需要单独对某语言写 Driver，应用不友好。
2. 并未实现 DB 层的透明使用。

9、全局 ID

1. 很简单的应用变成了很复杂的实现。
2. 采用 MySQL 自增 ID，写入扩大，单机容量有限。
3. 利用数据库集群并设置相应的步长，绝对埋坑的方案。
4. 依赖第三方组件，Redis Sequence、Twitter Snowflake ，复杂度增加，还引入了单点。
5. Guid、Random 算法，说好的连续性呢？还有一定比例冲突。
6. 业务属性字段 + 时间戳 + 随机数，冲突比例很高，依赖 NTP 等时间一致服务。

10、Double resource for AP

1. 同样的数据需要双倍的人力和产品。
2. 产品的重复，Hadoop、Hive、Hbase、Phoenix。
3. 人力的重复。
4. 数据迁移的复杂实现，Canal、databus、puma、dataX ？
5. 实时查询？普遍 T+1 查询。
6. TP 业务表变更导致 AP 业务统计失败， “老板问为啥报表显示昨天订单下降这么多，因为做个了 DDL。”

11、运维友好度 (DDL、扩容等)

1. 运维的复杂性是随着机器数量指数级增长的，Google 在 F1 之前维护了一个 100 多个节点的 MySQL sharding 就痛得不行了，不惜重新写了一个 Spanner 和 F1 搞定这个问题。

2. 传统 RDBMS 上 DDL 锁表的问题，对于数据量较大的业务来说，锁定的时间会很长，如果使用 pt-osc、gh-ost 这样第三方工具来实现非阻塞 DDL，额外的空间开销会比较大，另外仍然需要人工的介入确保数据的一致性，最后切换的过程系统可能会有抖动，pt-osc 还需要两次获取 metalock，虽然这个操作本事很轻量，可糟糕的是如果它被诸如 DDL的锁阻塞，它会阻塞所有的 DML，于是悲剧了。

12、与原有业务的兼容性

1. 时间成本，如果业务一开始设计时没有考虑分库分表或者中间件这类的方案，在应对数据量暴增的情况下匆忙重构是很麻烦的事情。

2. 技术成本，如果没有强有力和有经验的架构师，很难在业务早期做出良好的设计，另外对于大多数非互联网行业的开发者来说更是不熟悉。

13、Sharding 容量管理

1. 拆分不足，需要再次拆分的问题，工作量巨大。

2. 拆分充足，大部分业务增长往往比预期低很多，经常发生“又被 PM 妹纸骗了，说好的百万级流量呢”的问题，即时业务增长得比较好，往往需要一个很长的周期，机器资源浪费严重。

14、运维成本，人力成本

不解释，SRE、DBA 兄弟们懂的。