Sharding分库分表实战 上

图灵：楼兰

一、分库分表基础

1、什么是分库分表？什么时候要分库分表？

先将tulingmall-order的jdbc数据源调整到本地，然后在本地数据库中，已经插入了20+W的订单数据，全都是Monkey用户插入的。

> 调整方法： 1、在启动类上排除掉SpringBootConfiguration.class，这个是ShardingSphere进行分库分表的配置类。
> @SpringBootApplication(exclude = {SpringBootConfiguration.class})
>
> 2、调整配置文件，打开application.yml中的datasource配置。

然后在前端tuling-front用Monkey/123用户登录，进入"我的订单"页面，这个页面可以查询到所有的订单。--页面没有分页，但是能够加载更多的订单。简单分析下这个查询订单的SQL：

```
select * from (
    SELECT
        o.id,
        o.status,
        o.total_amount,
        o.pay_amount,
        o.order_sn,
        o.member_id,
        ot.id ot_id,
        ot.product_id ot_product_id,
        ot.product_name ot_product_name,
        ot.product_pic ot_product_pic,
        ot.product_price ot_product_price,
        ot.product_sku_id ot_product_sku_id,
        ot.product_sku_code ot_product_sku_code,
        ot.product_quantity ot_product_quantity,
        ot.product_attr ot_product_attr
        FROM
          oms_order o
        LEFT JOIN
```

```
            oms_order_item ot ON o.id = ot.order_id
        WHERE
            o.delete_status = 0 and o.member_id=8
            ORDER BY o.create_time desc
)t  limit 0,10
```

> 查第一页会非常快，只要0.002秒。但是往后翻页时，效率会越来越低。当翻页翻到 10000,10时，执行时间需要3秒多。100000,10时执行时间需要4秒多。

查询会耗时2秒多。而简单执行下面这个SQL语句，会要消耗7秒多。

```
select * from (
select 1 from oms_order o LEFT JOIN oms_order_item ot on o.id = ot.order_id
) t limit 200000,10
```

> 这个时长如果需要响应到页面上，那就已经是无法忍受了。这还只是一个简单的查询，并且返回的数据其实还不多。如果数据更多(比如导出)，查询更复杂，并发量更大，那消耗的时间会更长。
>
> 单表数据量太大带来的问题还不止是查询变慢，如果数据更多，还很容易造成系统假死。如果我们的订单页面不断的加载订单，那浏览器也会崩溃。

分库分表的作用：1、加大存储，2、提升查询效率，3、提升数据库的并发能力。

阿里的开发规范中建议预估三年内单表数据量上500W，或者大小上2G，就要考虑分库分表。

二、定制分库分表策略

这一部分，我们来进行实战，对最为重要的订单数据进行分库分表配置。

> 实战代码演示，参见tulingmall-order-sharding模块。主要是通过在application.properties文件中定制数据分片策略

定制分库分表策略的注意点：

- 分库分表一般要在开发之前设计，尽量避免临时设计。只针对最核心的表。
- 分库分表后，对表的查询必须足够简单，不要有跨表，跨库的复杂查询。
- 分库和分表尽量同时进行，分库可以分担网络压力。

关于数据分片策略。

- 分库策略：将核心的订单表和订单物品表从业务库中移出来，单独放到一个库。我们分成两个库。分散网络压力。
- 分表策略：在每个库分两个表，尽量将数据平均的分配到这四个表中。分散单表查询压力。

> 考虑的问题：
>
> 1、回顾下ShardingSphere的几种分片策略实现方式：inline, standard, complex, hint。
>
> 2、两个库，四个表。要如何将数据分配均匀？
>
> 3、如何定制适合业务场景的数据分片策略？
>
> 有哪些常用的数据分片策略？取模分片、按时间范围分片、按业务要素(如地区、前缀等)分片。。。。
>
> 取模分片能够将数据分配得尽量的平均，但是不利于扩展。
>
> 按范围分片便于扩展但是他的数据分布又不够均匀。
>
> 是不是可以定制一种分片策略，将这两种分片策略结合起来？比如大尺度上按范围分片，但是在每个数据范围内，使用取模分片。这种分片策略要如何在ShardingSphere中实现？提供思路，留给大家自己实现。

- 主键生成策略：ShardingSphere内置UUID和SNOWFLAKE两种。扩展自定义主键生成策略

application.properties分库分表配置如下：

```
# 配置ds0 和ds1两个数据源
spring.shardingsphere.datasource.names=ds,ds0,ds1
#ds1 配置
spring.shardingsphere.datasource.ds.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.ds.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.ds.url=jdbc:mysql://localhost:3306/micromall?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
#初始连接数
spring.shardingsphere.datasource.ds.initialSize=5
```

```
#最小空闲连接数
spring.shardingsphere.datasource.ds.minIdle=10
#最大连接数
spring.shardingsphere.datasource.ds.maxActive=30
spring.shardingsphere.datasource.ds.validationQuery=SELECT 1 FROM DUAL
spring.shardingsphere.datasource.ds.username=root
spring.shardingsphere.datasource.ds.password=root
#ds0 配置
spring.shardingsphere.datasource.ds0.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.ds0.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.ds0.url=jdbc:mysql://localhost:3306/micromall_ds_0?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds0.initialSize=5
spring.shardingsphere.datasource.ds0.minIdle=10
spring.shardingsphere.datasource.ds0.maxActive=30
spring.shardingsphere.datasource.ds0.validationQuery=SELECT 1 FROM DUAL
spring.shardingsphere.datasource.ds0.username=root
spring.shardingsphere.datasource.ds0.password=root
#ds1 配置
spring.shardingsphere.datasource.ds1.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.ds1.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.ds1.url=jdbc:mysql://localhost:3306/micromall_ds_1?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds1.initialSize=5
spring.shardingsphere.datasource.ds1.minIdle=10
spring.shardingsphere.datasource.ds1.maxActive=30
spring.shardingsphere.datasource.ds1.validationQuery=SELECT 1 FROM DUAL
spring.shardingsphere.datasource.ds1.username=root
spring.shardingsphere.datasource.ds1.password=root

# 对于没有做任何业务拆分的表，直接走本默认数据源即可
spring.shardingsphere.sharding.default-data-source-name=ds

spring.shardingsphere.sharding.default-database-strategy.hint.algorithm-class-name=com.tuling.tulingmall.sharding.OrderAllRangeHintAlgorithm
spring.shardingsphere.sharding.default-table-strategy.hint.algorithm-class-name=com.tuling.tulingmall.sharding.OrderAllRangeHintAlgorithm
# oms_order分片策略
# 节点 ds0.oms_order_0,ds0.oms_order_1,ds1.oms_order_0,ds1.oms_order_1
spring.shardingsphere.sharding.tables.oms_order.actual-data-nodes=ds$->{0..1}.oms_order_$->{0..1}
#分库策略
```

```
spring.shardingsphere.sharding.tables.oms_order.database-strategy.inline.sharding-column=id
spring.shardingsphere.sharding.tables.oms_order.database-strategy.inline.algorithm-expression=ds$->{id % 2}
# 分表策略
spring.shardingsphere.sharding.tables.oms_order.table-strategy.inline.sharding-column=id
# 注意下，对于除法，groovy会计算出浮点数，而不是整数。即 3/2=1.5，如果需要计算出整数 3.intdiv(2)=1
spring.shardingsphere.sharding.tables.oms_order.table-strategy.inline.algorithm-expression = oms_order_$->{((id+1) % 4).intdiv(2)}
# 复合分片算法
#spring.shardingsphere.sharding.tables.oms_order.table-strategy.complex.sharding-columns=id
#spring.shardingsphere.sharding.tables.oms_order.table-strategy.complex.algorithm-class-name = com.tuling.tulingmall.sharding.OrderComplexShardingAlgorithm
# 使用SNOWFLAKE算法生成主键
spring.shardingsphere.sharding.tables.oms_order.key-generator.column=id
spring.shardingsphere.sharding.tables.oms_order.key-generator.type=SNOWFLAKE
spring.shardingsphere.sharding.tables.oms_order.key-generator.props.worker.id=123
# 使用自定义主键生成策略
#spring.shardingsphere.sharding.tables.oms_order.key-generator.type=CUSTOM
#spring.shardingsphere.sharding.tables.oms_order.key-generator.props.redis.prefix=order:id:prefix:

# 节点 ds0.oms_order_item_0,ds0.oms_order_item_1,ds1.oms_order_item_0,ds1.oms_order_item_1
spring.shardingsphere.sharding.tables.oms_order_item.actual-data-nodes=ds$->{0..1}.oms_order_item_$->{0..1}
# 分库策略 按order_id分片
spring.shardingsphere.sharding.tables.oms_order_item.database-strategy.inline.sharding-column=order_id
spring.shardingsphere.sharding.tables.oms_order_item.database-strategy.inline.algorithm-expression=ds$->{order_id % 2}
# 分表策略
spring.shardingsphere.sharding.tables.oms_order_item.table-strategy.inline.sharding-column=order_id
spring.shardingsphere.sharding.tables.oms_order_item.table-strategy.inline.algorithm-expression=oms_order_item_$->{((order_id+1) % 4).intdiv(2)}
# 使用SNOWFLAKE算法生成主键
spring.shardingsphere.sharding.tables.oms_order_item.key-generator.column=id
```

```
spring.shardingsphere.sharding.tables.oms_order_item.key-generator.type=SNOWFLAKE
spring.shardingsphere.sharding.tables.oms_order_item.key-generator.props.worker.id=123

# 配置绑定表，防止笛卡尔乘积
spring.shardingsphere.sharding.binding-tables[0]=oms_order,oms_order_item
# 打印SQL语句
spring.shardingsphere.props.sql.show=true
```

> 注意： 1、我们配置的分库分表策略是将oms_order和oms_order_item两个表配置分库分表策略，其他的没有配置的表还是会走ds数据库。
>
> 2、注意理解下分库分表的策略。如何将记录平均分配到两个库的四个表中？如果分表策略只是oms_order_${order_id % 2}，能平均分配吗？
>
> 3、为什么要配置绑定表？

三、实现订单分库分表

这一部分主要是将数据分片策略整合到tulingmall-order-sharding模块当中。

基础环境配置：

1、将项目中的mysql数据库配置迁移到本地数据库，后续将在本地数据库完成分库分表开发。

> 所有服务都安装在192.168.65.232服务器上。包括RocketMQ,RabbitMQ,Zookeeper,Nacos,Seata,MySQL,Redis...
>
> 可以在本地hosts文件中增加一个配置 192.168.65.232 tlshop.com。这是因为在nacos的配置中心，大都是把服务按照tlshop.com的域名配置的。

2、将tulingmall-product模块的bootstrap.xml中去掉tulingmall-db-common.yml的配置依赖，再在application.yml中，将mysql数据库改为本地数据库配置。

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/micromall?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
```

这一步是为了让product模块不去依赖nacos上的配置，而采用本地数据库mysql。因为系统中订单流程是跟产品功能绑定的。必须要先将产品加入到购物车，然后在购物车中确认商品后才能生成订单。所以在调整订单模块之前需要将产品模块的数据库一起调整过来。

3、将192.168.65.232服务器上mysql里micromall库的数据全都移植到本地mysql的micromall中。

其他模块比如用户模块，可以不用调整数据源，因为我们不会修改这部分数据。

实现分库分表的核心步骤：

1、定义分库分表数据源：使用ShardingSphere的DataSource替换原始的DataSource

2、定制分库分表规则：将oms_order和oms_order_item两张表映射成为ShardingSphere中的逻辑表，并配置分库分表规则，将这两张表移动到单独的数据库中。其他表不要动。

3、分库分表规则优化：分库分表规则、绑定表、读写分离、自定义路由策略、自定义主键生成策略等。

开发过程

1、加入分库分表的数据源配置

在tulingmall-order模块的application.yml中，去掉spring.datasource部分的配置。然后引入application.properties，并在其中配置分库分表。

2、将TulingmallOrderApplication 启动类上排除掉的SpringBootConfiguration.class给加回来。

这样就完成了简单的分库分表。

四、补充内容 配置ShardingProxy

> 这一部分是要配置shardingproxy支持同样的分片策略。

1、理想与现实的差距

分库分表后对整个业务功能的影响是很大的，所以通常建议是要在系统设计之初就想明白要不要进行分库分表以及如何分库分表。但是很多公司在实践过程中，对于分库分表都是后知后觉的，很多业务都是等数量累积到了一定程度，数据量的问题爆发出来了，才想起要分库分表。那这时要怎么处理？

2、旧数据处理方式

如果要在中途进行分库分表，要分为两个步骤：

首先：要评估数据分片方案，对关键的SQL进行整理并分析。分库分表后，有很多SQL是无法支持的，这些SQL一定要优先从业务中去掉。这个步骤很容易被忽略，但是一定是必不可少的。**有哪些SQL是ShardingSphere不支持的？** 参见官网。

然后：当你制定好了分库分表方案后，不要急于迁移旧数据。最好是在业务中对SQL进行数据双写。即老数据库写一份，新的分片后的数据库也写一份。观察一段时间，等业务稳定了之后，再考虑全部转移到分片后的新数据库中。**这同样需要多方定制ShardingSphere的分片策略，简单的inline是很难达到这个目的的。**

接下来：进行旧数据迁移时，**可以采用ShardingProxy来协助进行数据转移。**部署同样分片策略的ShardingProxy，一方面可以在MySQL的客户端工具中快速验证分片策略，另外可以使用sqoop、keetle等工具来协助进行数据转移。

下面来实战在ShardingProxy中复制

config-sharding.yaml配置：

```
schemaName: sharding_db

dataSources:
  ds:
    url: jdbc:mysql://localhost:3306/micromall?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
    password: root
  ds0:
    url: jdbc:mysql://localhost:3306/micromall_ds_0?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
    password: root
  ds1:
    url: jdbc:mysql://localhost:3306/micromall_ds_1?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
```

```
    password: root

shardingRule:
  tables:
    oms_order:
      actualDataNodes: ds$->{0..1}.oms_order_$->{0..1}
      databaseStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: ds$->{id % 2}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: oms_order_$->{(id+1) % 4 /2}
      keyGenerator:
        column: id
        type: SNOWFLAKE
        props:
          worker.id: 123
    oms_order_item:
      actualDataNodes: ds$->{0..1}.oms_order_item_$->{0..1}
      databaseStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: ds$->{id % 2}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: oms_order_item_$->{(order_id+1) % 4 / 2}
      keyGenerator:
        column: id
        type: SNOWFLAKE
        props:
          worker.id: 123
  bindingTables:
    - oms_order,oms_order_item
  defaultDataSourceName: ds
```

> 在定制分表策略时，这种配置方式是有问题的，因为groovy在进行除2的计算时，如果是奇数，会计算出浮点数0.5，所有会间断报错。这时需要将除法改成div方法。
>
> ```
> oms_order_$->{((id+1) % 4).intdiv(2)}
> oms_order_item_$->{((order_id+1) % 4).intdiv(2)}
> ```

开发完成后，将分库分表策略分配到ShardingProxy中，用于进行快速实验。

shardingProxy的config-sharding.yaml

```yaml
schemaName: sharding_db

dataSources:
  ds:
    url: jdbc:mysql://localhost:3306/micromall?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
    password: root
  ds0:
    url: jdbc:mysql://localhost:3306/micromall_ds_0?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
    password: root
  ds1:
    url: jdbc:mysql://localhost:3306/micromall_ds_1?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
    password: root

shardingRule:
  tables:
    oms_order:
      actualDataNodes: ds$->{0..1}.oms_order_$->{0..1}
      databaseStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: ds$->{id % 2}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: oms_order_$->{((id+1) % 4).intdiv(2)}
      keyGenerator:
        column: id
        type: SNOWFLAKE
        props:
          worker.id: 123
    oms_order_item:
      actualDataNodes: ds$->{0..1}.oms_order_item_$->{0..1}
      databaseStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: ds$->{id % 2}
      tableStrategy:
        inline:
          shardingColumn: order_id
```

```
          algorithmExpression: oms_order_item_$->{((order_id+1) %
4).intdiv(2)}
      keyGenerator:
        column: id
        type: SNOWFLAKE
        props:
          worker.id: 123
  bindingTables:
    - oms_order,oms_order_item
  defaultDataSourceName: ds
```

五、分库分表带来的问题

定制主键生成策略

主键是分库分表中非常重要的业务要素，通常分库分表都会采用主键来作为分片键，这个时候主键就不再只是用来提升查询效率了，还需要坚固数据分片的效率。要如何定制高效的主键生成策略?

很多SQL不支持

例如MySQL里会配for each标签来执行批量SQL，原始数据库是支持的，但是分库分表不支持。

查询的SQL比较多时，路由策略是否支持?

去官网上查一下分库分表的不支持项。

其他问题

数据迁移、扩缩容、公共表、读写分离、配置往注册中心集中配置

分布式事务处理

一旦涉及到分布式事务，就会带来非常多麻烦的事情。前面fox老师才讲了分布式事务要如何处理，分布式事务的各种处理机制你是否弄明白了?

在分库分表场景下的分布式事务是什么样的?要怎么处理?下一节课来讲。

> 文档：电商VIP-Sharding分库分表实战 上.md
> 链接：http://note.youdao.com/noteshare?id=c9222af5273865c1743f733f11a19a28&sub=D8FAD3179BBC46B392BE6D5E90745F98