3 Mycat分库分表

3.0 准备

1. 准备两个数据库服务

   dhost1： 192.168.100.218

   dhost2： localhost

2. 在dhost2服务创建两个数据库

```
CREATE DATABASE db1 CHARACTER SET 'utf8';

CREATE DATABASE db2 CHARACTER SET 'utf8';
```

3. 配置好dataHost dataNode

```
<mycat:schema xmlns:mycat="http://io.mycat/">
    <!-- 注意：里面的元素一定要按 schema 、dataNode 、 dataHost的顺序配置 -->
    <schema name="mydb1" checkSQLschema="true" sqlMaxLimit="100"
dataNode="dn2">

    </schema>

    <dataNode name="dn1" dataHost="dhost1" database="orders" />
    <dataNode name="dn2" dataHost="dhost2" database="db1" />
    <dataNode name="dn3" dataHost="dhost2" database="db2" />

    <dataHost name="dhost1" maxCon="1000" minCon="10" balance="1"
 writeType="0" dbType="mysql" dbDriver="native">
        <heartbeat>select user()</heartbeat>
        <writeHost host="myhostM1" url="192.168.100.218:3306"
user="mike" password="Mike666!">
        </writeHost>
    </dataHost>
    <dataHost name="dhost2" maxCon="1000" minCon="10" balance="1"
        writeType="0" dbType="mysql" dbDriver="native">
        <heartbeat>select user()</heartbeat>
```

```
        <writeHost host="myhostM2" url="localhost:3306" user="mike"
password="Mike666!"></writeHost>
    </dataHost>
</mycat:schema>
```

3.1 表分类

分片表

分片表，是指那些有很大数据，需要切分到多个数据库的表，这样每个分片都有一部分数据，所有分片构成了完整的数据。

```
<table name="t_goods" primaryKey="vid" autoIncrement="true"
dataNode="dn1,dn2" rule="rule1" />
```

非分片表

一个数据库中并不是所有的表都很大，某些表是可以不用进行切分的，非分片是相对分片表来说的，就是那些不需要进行数据切分的表。

```
<table name="t_node" primaryKey="vid" autoIncrement="true" dataNode="dn1"
/>
```

示例

商家表，数据量500万内。

```
CREATE TABLE t_shops(
    id bigint PRIMARY KEY AUTO_INCREMENT,
    name varchar(100) not null
);
```

```
<table name="t_shops" primaryKey="id" dataNode="dn1" />
```

```
INSERT INTO t_shops(name) values('mike001');
```

ER表

Mycat 中的ER 表是基于E-R 关系的数据分片策略，子表的记录与所关联的父表记录存放在同一个数据分片上，保证数据join 不会跨库操作。

ER分片是解决跨分片数据join 的一种很好的思路，也是数据切分规划的一条重要规则。

```
 <table name="customer" primaryKey="ID" dataNode="dn1,dn2" rule="sharding-by-intfile">
        <childTable name="orders" primaryKey="ID" joinKey="customer_id" parentKey="id">
            <childTable name="order_items" joinKey="order_id" parentKey="id" />
        </childTable>
</table>
```

全局表

一个真实的业务系统中，往往存在大量的类似字典表的表，这些表基本上很少变动。

问题：业务表往往需要和字典表join查询，当业务表因为规模而进行分片以后，业务表与字典表之间的关联跨库了。

解决：Mycat中通过表冗余来解决这类表的join，即它的定义中指定的dataNode上都有一份该表的拷贝。（将字典表或者符合字典表特性的表定义为全局表。 ）

```
<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />
```

示例：

省份表 t_province，在各数据节点所在库上分别创建全局表：

```
CREATE TABLE t_province(
        id INT PRIMARY KEY,
  name varchar(100) not null
);
```

```
<table name="t_province" primaryKey="ID" type="global" dataNode="dn1,dn2" />
```

重启mycat服务

插入数据看看

```
INSERT INTO t_province(id,name) values(1001,'浙江');
INSERT INTO t_province(id,name) values(1002,'江苏');
INSERT INTO t_province(id,name) values(1003,'上海');
INSERT INTO t_province(id,name) values(1004,'广东');
```

两个数据节点库上都同时写入了该数据。

3.2 分片规则配置

分表规则定义

在conf/rule.xml中定义分片规则：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mycat:rule SYSTEM "rule.dtd">
<mycat:rule xmlns:mycat="http://io.mycat/">
    <tableRule name="rule1">
        <rule>
            <columns>id</columns>
            <algorithm>func1</algorithm>
        </rule>
    </tableRule>
    <function name="func1"
class="io.mycat.route.function.PartitionByLong">
        <property name="partitionCount">8</property>
        <property name="partitionLength">128</property>
    </function>
</mycat:rule>
```

tableRule标签说明：

- name 属性指定唯一的名字，用于标识不同的表规则。
- 内嵌的rule 标签则指定对物理表中的哪一列进行拆分和使用什么路由算法。
    - columns 内指定要拆分的列名字。
    - algorithm 使用function 标签中的name 属性。连接表规则和具体路由算法。当然，多个表规则可以连接到同一个路由算法上。table 标签内使用。让逻辑表使用这个规则进行分片。

function标签说明：

- name 指定算法的名字。
- class 制定路由算法具体的类名字。
- property 为具体算法需要用到的一些属性。

3.3 分表分库原则

分表分库原则

分表分库虽然能解决大表对数据库系统的压力，但它并不是万能的，也有一些不利之处，因此**首要问题是分不分库，分哪些库，什么规则分，分多少分片。**

- 原则一：能不分就不分，1000 万以内的表，不建议分片，通过合适的索引，读写分离等方式，可以很好的解决性能问题。
- 原则二：分片数量尽量少，分片尽量均匀分布在多个DataHost 上，因为一个查询SQL 跨分片越多，则总体性能越差，虽然要好于所有数据在一个分片的结果，只在必要的时候进行扩容，增加分片数量。
- 原则三：分片规则需要慎重选择，分片规则的选择，需要考虑数据的增长模式，数据的访问模式，分片关联性问题，以及分片扩容问题，最常用的分片策略为范围分片，枚举分片，一致性Hash 分片，这几种分片都有利于扩容。
- 原则四：尽量不要在一个事务中的SQL 跨越多个分片，分布式事务一直是个不好处理的问题。
- 原则五：查询条件尽量优化，尽量避免Select * 的方式，大量数据结果集下，会消耗大量带宽和CPU 资源，查询尽量避免返回大量结果集，并且尽量为频繁使用的查询语句建立索引。

这里特别强调一下分片规则的选择问题，如果某个表的数据有明显的时间特征，比如订单、交易记录等，则他们通常比较合适用时间范围分片，因为具有时效性的数据，我们往往关注其近期的数据，查询条件中往往带有时间字段进行过滤，比较好的方案是，当前活跃的数据，采用跨度比较短的时间段进行分片，而历史性的数据，则采用比较长的跨度存储。

总体上来说，分片的选择是取决于最频繁的查询SQL 的条件，因为不带任何Where 语句的查询SQL，会便利所有的分片，性能相对最差，因此这种SQL 越多，对系统的影响越大，所以我们要尽量避免这种SQL 的产生。

SQL统计分析

如何准确统计和分析当前系统中最频繁的SQL 呢？有几个简单做法：

- 采用特殊的JDBC 驱动程序，拦截所有业务SQL，并写程序进行分析
- 采用Mycat 的SQL 拦截器机制，写一个插件，拦截所欲SQL，并进行统计分析

- 打开MySQL 日志，分析统计所有SQL。

找出每个表最频繁的SQL，分析其查询条件，以及相互的关系，并结合ER 图，就能比较准确的选择每个表的分片策略。

库内分表说明

对于大家经常提起的同库内分表的问题，这里做一些分析和说明，同库内分表，仅仅是单纯的解决了单一表数据过大的问题，由于没有把表的数据分布到不同的机器上，因此对于减轻MySQL 服务器的压力来说，并没有太大的作用，大家还是竞争同一个物理机上的IO、CPU、网络。

此外，库内分表的时候，要修改用户程序发出的SQL，可以想象一下A、B 两个表各自分片5 个分表情况下的Join SQL 会有多么的反人类。这种复杂的SQL 对于DBA 调优来说，也是个很大的问题。因此，Mycat 和一些主流的数据库中间件，都不支持库内分表，但由于MySQL 本身对此有解决方案，所以可以与Mycat 的分库结合，做到最佳效果，下面是MySQL 的分表方案：

- MySQL 分区；
- MERGE 表（MERGE 存储引擎）。

通俗地讲MySQL 分区是将一大表，根据条件分割成若干个小表。mysql5.1 开始支持数据表分区了。如：某用户表的记录超过了600 万条，那么就可以根据入库日期将表分区，也可以根据所在地将表分区。当然也可根据其他的条件分区。

MySQL 分区支持的分区规则有以下几种：

- RANGE 分区：基于属于一个给定连续区间的列值，把多行分配给分区。
- LIST 分区：类似于按RANGE 分区，区别在于LIST 分区是基于列值匹配一个离散值集合中的某个值来进行选择。
- HASH 分区：基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含MySQL 中有效的、产生非负整数值的任何表达式。
- KEY 分区：类似于按HASH 分区，区别在于KEY 分区只支持计算一列或多列，且MySQL 服务器提供其自身的哈希函数。必须有一列或多列包含整数值。

在Mysql 数据库中，Merge 表有点类似于视图，mysql 的merge 引擎类型允许你把许多结构相同的表合并为一个表。之后，你可以执行查询，从多个表返回的结果就像从一个表返回的结果一样。每一个合并的表必须有完全相同表的定义和结构，但是只支持MyISAM 引擎。

Mysql Merge 表的优点：

- 分离静态的和动态的数据；
- 利用结构接近的的数据来优化查询；

- 查询时可以访问更少的数据；
- 更容易维护大数据集。

在数据量、查询量较大的情况下，不要试图使用Merge 表来达到类似于Oracle 的表分区的功能，会很影响性能。我的感觉是和union 几乎等价。

Mycat 建议的方案是Mycat 分库+MySQL 分区，此方案具有以下优势：

- 充分结合分布式的并行能力和MySQL 分区表的优化；
- 可以灵活的控制表的数据规模；
- 可以两个维度对表进行分片，MyCAT 一个维度分库，MySQL 一个维度分区。

3.4 数据拆分原则

1. 达到一定数量级才拆分（800 万）
2. 不到800 万但跟大表（超800 万的表）有关联查询的表也要拆分，在此称为大表关联表
3. 大表关联表如何拆：小于100 万的使用全局表；大于100 万小于800 万跟大表使用同样的拆分策略；无法跟大表使用相同规则的，可以考虑从java 代码上分步骤查询，不用关联查询，或者破例使用全局表。
4. 破例的全局表：如item_sku 表250 万，跟大表关联了，又无法跟大表使用相同拆分策略，也做成了全局表。破例的全局表必须满足的条件：没有太激烈的并发update，如多线程同时update 同一条id=1 的记录。虽有多线程update，但不是操作同一行记录的不在此列。多线程update 全局表的同一行记录会死锁。批量insert没问题。
5. 拆分字段是不可修改的
6. 拆分字段只能是一个字段，如果想按照两个字段拆分，必须新建一个冗余字段，冗余字段的值使用两个字段的值拼接而成（如大区+年月拼成zone_yyyymm 字段）。
7. 拆分算法的选择和合理性评判：按照选定的算法拆分后每个库中单表不得超过800 万
8. 能不拆的就尽量不拆。如果某个表不跟其他表关联查询，数据量又少，直接不拆分，使用单库即可。

3.5 DataNode 的分布问题

DataNode 代表MySQL 数据库上的一个Database，因此一个分片表的DataNode 的分布可能有以下几种：

- 都在一个DataHost 上
- 在几个DataHost 上，但有连续性，比如dn1 到dn5 在Server1 上，dn6 到dn10 在Server2 上，依次类推
- 在几个DataHost 上，但均匀分布，比如dn1,dn2,d3 分别在Server1,Server2,Server3 上，dn4 到dn5 又重复如此

一般情况下，不建议第一种，二对于范围分片来说，在大多数情况下，最后一种情况最理想，因为当一个表的数据均匀分布在几个物理机上的时候，跨分片查询或者随机查询，都是到不同的机器上去执行，并行度最高，IO 竞争也最小，因此性能最好。

当我们有几十个表都分片的情况下，怎样设计DataNode 的分布问题，就成了一个难题，解决此难题的最好方式是试运行一段时间，统计观察每个DataNode 上的SQL 执行情况，看是否有严重不均匀的现象产生，然后根据统计结果，重新映射DataNode 到DataHost 的关系。

Mycat 1.4 增加了distribute 函数，可以用于Table 的dataNode 属性上，表示将这些dataNode 在该Table 的分片规则里的引用顺序重新安排，使得他们能均匀分布到几个DataHost 上：

```
<table name="oc_call" primaryKey="ID" dataNode="distribute(dn1$0-372,dn2$0-
372)"
       rule="latest-monthcalldate"/>
```

其中dn1xxx 与dn2xxxx 是分别定义在DataHost1 上与DataHost2 上的377 个分片

3.6 Mycat内置的常用分片规则

1 分片枚举（列表分片）

通过在配置文件中配置可能的枚举id，自己配置分片，本规则适用于特定的场景，比如有些业务需要按照省份或区县来做保存，而全国省份区县固定的，这类业务使用本条规则，配置如下：

```
<tableRule name="sharding-by-intfile">
    <rule>
        <columns>user_id</columns>
        <algorithm>hash-int</algorithm>
    </rule>
</tableRule>
<function name="hash-int"
class="io.mycat.route.function.PartitionByFileMap">
    <property name="mapFile">sharding-by-enum.txt</property>
    <property name="type">0</property>
    <property name="defaultNode">0</property>
</function>
```

function分片函数中配置说明：

- 算法实现类为：io.mycat.route.function.PartitionByFileMap
- mapFile 标识配置文件名称；
- type 默认值为0，0 表示Integer，非零表示String；
- defaultNode defaultNode 默认节点:小于0 表示不设置默认节点，大于等于0 表示设置默认节点为第几个数据节点。

  默认节点的作用：枚举分片时，如果碰到不识别的枚举值，就让它路由到默认节点 如果不配置默认节点（defaultNode 值小于0 表示不配置默认节点），碰到不识别的枚举值就会报错。

  ```
  like this：can't find datanode for sharding column:column_name
  val:ffffffff
  ```

- sharding-by-enum.txt 放置在conf/下，配置内容示例：

```
10000=0      #字段值为10000的放到0号数据节点
10010=1
```

示例

客户表t_customer

```
CREATE TABLE t_customer(
        id BIGINT PRIMARY KEY,
        name varchar(100) not null,
        province int not null
);
```

按省份进行数据分片，表配置：

```
<table name="t_customer" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
       rule="sharding-by-province" />
```

分片规则配置 rule.xml：

```
<mycat:rule xmlns:mycat="http://io.mycat/">
        <tableRule name="sharding-by-province">
                <rule>
                        <columns>province</columns>
                        <algorithm>sharding-by-province-func</algorithm>
```

```
                </rule>
        </tableRule>

        <function name="sharding-by-province-func"
                class="io.mycat.route.function.PartitionByFileMap">
                <property name="mapFile">sharding-by-
province.txt</property>
                <property name="type">0</property>
                <property name="defaultNode">0</property>
        </function>

</mycat:rule>
```

sharding-by-province.txt文件中枚举分片

```
1001=0
1002=1
1003=2
1004=0
```

测试：插入数据

```
insert into t_customer(name,province) values('mike01',1001);
insert into t_customer(name,province) values('mike02',1002);
insert into t_customer(name,province) values('mike03',1003);
insert into t_customer(name,province) values('mike04',1004);
insert into t_customer(name,province) values('mike05',1005);
```

2 范围分片

此分片适用于，提前规划好分片字段某个范围属于哪个分片

```
<tableRule name="range-sharding">
    <rule>
        <columns>user_id</columns>
        <algorithm>rang-long</algorithm>
    </rule>
</tableRule>
<function name="rang-long"
class="io.mycat.route.function.AutoPartitionByLong">
    <property name="mapFile">range-partition.txt</property>
    <property name="defaultNode">0</property>
</function>
```

配置说明：

- mapFile 代表配置文件路径
- defaultNode 超过范围后的默认节点。

所有的节点配置都是从0 开始，及0 代表节点1。

mapFile中的定义规则：

```
        start <= range <= end.
        range start-end=data node index
        K=1000,M=10000.
```

配置示例：

```
0-500M=0
500M-1000M=1
1000M-1500M=2
```

或

```
0-10000000=0
10000001-20000000=1
```

示例

在mycat中定义分片表：

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
       rule="range-sharding-by-members-count" />
```

```
<tableRule name="range-sharding-by-members-count">
    <rule>
        <columns>members</columns>
        <algorithm>range-members-count</algorithm>
    </rule>
</tableRule>
<function name="range-members-count"
class="io.mycat.route.function.AutoPartitionByLong">
    <property name="mapFile">company-range-partition.txt</property>
    <property name="defaultNode">0</property>
</function>
```

company-range-partition.txt中分片定义：

```
0-10=0
11-50=1
51-100=2
101-1000=0
1001-9999=1
10000-9999999=2
```

创建表

```
CREATE TABLE t_company(
        id BIGINT PRIMARY KEY,
    name varchar(100) not null,
        members int not null
);
```

测试：

```
INSERT INTO t_company(name,members) VALUES('company01',10);
INSERT INTO t_company(name,members) VALUES('company01',20);
INSERT INTO t_company(name,members) VALUES('company01',200);
```

3 按日期范围分片

此规则为按日期段进行分片。

```
<tableRule name="sharding-by-date">
    <rule>
        <columns>create_time</columns>
        <algorithm>sharding-by-date</algorithm>
    </rule>
</tableRule>
<function name="sharding-by-date"
class="io.mycat.route.function.PartitionByDate">
    <property name="dateFormat">yyyy-MM-dd</property>
    <property name="sBeginDate">2018-01-01</property>
    <property name="sEndDate">2019-01-02</property>
    <property name="sPartionDay">10</property>
</function>
```

配置说明：

- columns ：标识将要分片的表字段
- algorithm ：分片函数
- dateFormat ：日期格式
- sBeginDate ：开始日期
- sEndDate：结束日期
- sPartionDay ：分区天数，即默认从开始日期算起，分隔10 天一个分区

sBeginDate,sEndDate配置情况说明：

- **sBeginDate,sEndDate 都有指定**

此时表的dataNode 数量的>=这个时间段算出的分片数，否则启动时会异常：

```
Exception in thread "main" java.lang.ExceptionInInitializerError
        at io.mycat.MycatStartup.main(MycatStartup.java:53)
Caused by: io.mycat.config.util.ConfigException: Illegal table conf : table
[ T_ORDER ] rule function [ shardi
 partition size : 4 > table datanode size : 3, please make sure table
datanode size = function partition size
```

如果配置了sEndDate 则代表数据达到了这个日期的分片后循环从开始分片插入。

- **没有指定 sEndDate 的情况**

  数据分片将依次存储到dataNode上，数据分片随时间增长，所需的dataNode数也随之增长，当超出了为该表配置的dataNode数时，将得到如下异常信息：

  ```
  [SQL]
  INSERT INTO t_order(order_time,customer_id,order_amount) VALUES ('2019-02-05',1001,203);
  [Err] 1064 - Can't find a valid data node for specified node index :T_ORDER -> ORDER_TIME -> 2019-02-05 -> Index : 3
  ```

示例

```
<table name="t_order" primaryKey="order_id" autoIncrement="true"
       dataNode="dn1,dn2,dn3" rule="order-sharding-by-date" />
```

```
<tableRule name="order-sharding-by-date">
    <rule>
        <columns>order_time</columns>
        <algorithm>sharding-by-date</algorithm>
    </rule>
</tableRule>
<function name="sharding-by-date"
class="io.mycat.route.function.PartitionByDate">
    <property name="dateFormat">yyyy-MM-dd</property>
    <property name="sBeginDate">2019-01-01</property>
    <property name="sEndDate">2019-02-02</property>
    <property name="sPartionDay">20</property>
</function>
```

```
CREATE TABLE t_order (
        order_id        BIGINT PRIMARY KEY,
        order_time      DATETIME,
        customer_id     BIGINT,
        order_amount    DECIMAL(8,2)
);
```

测试

```
INSERT INTO t_order(order_time,customer_id,order_amount) VALUES ('2019-01-05',1001,201);
INSERT INTO t_order(order_time,customer_id,order_amount) VALUES ('2019-01-25',1001,202);
INSERT INTO t_order(order_time,customer_id,order_amount) VALUES ('2019-02-15',1001,203);
INSERT INTO t_order(order_time,customer_id,order_amount) VALUES ('2019-03-15',1001,203);
```

请去看数据的分布！

4 自然月分片

按月份列分区，每个自然月一个分片。

```
<tableRule name="sharding-by-month">
    <rule>
        <columns>create_time</columns>
        <algorithm>sharding-by-month</algorithm>
    </rule>
</tableRule>
<function name="sharding-by-month"
class="io.mycat.route.function.PartitionByMonth">
    <property name="dateFormat">yyyy-MM-dd</property>
    <property name="sBeginDate">2014-01-01</property>
</function>
```

配置说明：

- columns： 分片字段，字符串类型
- dateFormat ： 日期字符串格式,默认为yyyy-MM-dd
- sBeginDate ： 开始日期，无默认值
- sEndDate：结束日期，无默认值
- 节点从0 开始分片

使用场景：

场景1：默认设置（不指定sBeginDate、sEndDate）

节点数量必须是12 个，对应1 月~12 月

- "2017-01-01" = 节点0
- "2018-01-01" = 节点0
- "2018-05-01" = 节点4
- "2019-12-01" = 节点11

场景2 ：仅指定sBeginDate

sBeginDate = "2017-01-01" 该配置表示"2017-01 月"是第0 个节点，从该时间按月递增，无最大节点

- "2014-01-01" = 未找到节点
- "2017-01-01" = 节点0
- "2017-12-01" = 节点11
- "2018-01-01" = 节点12
- "2018-12-01" = 节点23

场景3： 指定sBeginDate=1月、sEndDate=12月

sBeginDate = "2015-01-01" sEndDate = "2015-12-01" 该配置可看成与场景1 一致。

- "2014-01-01" = 节点0
- "2014-02-01" = 节点1
- "2015-02-01" = 节点1
- "2017-01-01" = 节点0
- "2017-12-01" = 节点11
- "2018-12-01" = 节点11

场景4：

sBeginDate = "2015-01-01"sEndDate = "2015-03-01" 该配置表示只有3 个节点；很难与月份对应上；平均分散到3 个节点上

5 取模

此规则为对分片字段进行十进制运算，来分片数据。

```
<tableRule name="mod-sharding">
    <rule>
        <columns>user_id</columns>
        <algorithm>mod-fun</algorithm>
    </rule>
</tableRule>
<function name="mod-fun" class="io.mycat.route.function.PartitionByMod">
        <!-- how many data nodes -->
        <property name="count">3</property>
</function>
```

配置说明：

- count 指明dataNode 的数量，是求模的基数

此种在批量插入时可能存在批量插入单事务插入多数据分片，增大事务一致性难度。

6 取模范围分片

此种规则是取模运算与范围约束的结合，主要为了后续数据迁移做准备，即可以自主决定取模后数据的节点 分布。

```
<tableRule name="sharding-by-pattern">
    <rule>
        <columns>user_id</columns>
        <algorithm>sharding-by-pattern</algorithm>
    </rule>
</tableRule>
<function name="sharding-by-pattern"
class="io.mycat.route.function.PartitionByPattern">
    <property name="patternValue">256</property>
    <property name="defaultNode">2</property>
    <property name="mapFile">partition-pattern.txt</property>
</function>
```

partition-pattern.txt

```
1-32=0          #余数为1-32的放到数据节点0上
33-64=1
65-96=2
97-128=3
129-160=4
161-192=5
193-224=6
225-256=7
0-0=7
```

配置说明：

- patternValue 即求模基数
- defaoultNode 默认节点，如果配置了默认节点，如果id 非数据，则会分配在 defaoultNode 默认节点
- mapFile 指定余数范围分片配置文件

7 二进制取模范围分片

本条规则类似于十进制的求模范围分片，区别在于是二进制的操作,是分片列值的二进制低10位&1111111111。 此算法的优点在于如果按照10 进制取模运算，在连续插入1-10 时候1-10 会被分到1-10 个分片，增大了插入的事务控制难度，而此算法根据二进制则可能会分到连续的分片，减少插入事务控制难度。

二进制低10&1111111111 的结果是 0-1023 一共是1024个值，按范围分成多个连续的片（最大1024个片）

```
<tableRule name="rule1">
    <rule>
        <columns>user_id</columns>
        <algorithm>func1</algorithm>
    </rule>
</tableRule>
<function name="func1" class="io.mycat.route.function.PartitionByLong">
    <property name="partitionCount">2,1</property>
    <property name="partitionLength">256,512</property>
</function>
```

配置说明：

- partitionCount 分片个数列表。
- partitionLength 分片范围列表

分区长度:默认为最大2^n=1024 ,即最大支持1024 分区

约束:

- count,length 两个数组的长度必须是一致的。
- 1024 = sum((count[i] * length[i]))，count 和length 两个向量的点积恒等于1024

用法例子:

本例的分区策略：希望将数据水平分成3 份，前两份各占25%，第三份占50%。（本例非均匀分区） // |<———————1024———————————>|

// |<—-256—>|<—-256—>|<————-512————->| // | partition0 | partition1 | partition2 | // | 共2 份,故count[0]=2 | 共1 份，故count[1]=1 |

如果需要平均分配设置：平均分为4 分片，partitionCount*partitionLength=1024

```
<function name="func1" class="io.mycat.route.function.PartitionByLong">
    <property name="partitionCount">4</property>
    <property name="partitionLength">256</property>
</function>
```

8 范围取模分片

先进行范围分片计算出分片组，组内再求模。

优点可以避免扩容时的数据迁移，又可以一定程度上避免范围分片的热点问题。综合了范围分片和求模分片的优点，分片组内使用求模可以保证组内数据比较均匀，分片组之间是范围分片可以兼顾范围查询。

最好事先规划好分片的数量，数据扩容时按分片组扩容，则原有分片组的数据不需要迁移。由于分片组内数据比 较均匀，所以分片组内可以避免热点数据问题。

```
<tableRule name="auto-sharding-rang-mod">
    <rule>
        <columns>id</columns>
        <algorithm>rang-mod</algorithm>
    </rule>
</tableRule>
<function name="rang-mod"
class="io.mycat.route.function.PartitionByRangeMod">
    <property name="mapFile">partition-range-mod.txt</property>
    <property name="defaultNode">21</property>
</function>
```

配置说明：

- mapFile 配置文件路径
- defaultNode 超过范围后的默认节点顺序号，节点从0 开始。

partition-range-mod.txt 以下配置一个范围代表一个分片组，=号后面的数字代表该分片组所拥有的分片的数量。

```
0-200M=5       //代表有5个分片节点
200M1-400M=1
400M1-600M=4
600M1-800M=4
800M1-1000M=6
```

9 一致性hash

一致性hash 算法有效解决了分布式数据的扩容问题。

```
<tableRule name="sharding-by-murmur">
    <rule>
        <columns>user_id</columns>
        <algorithm>murmur</algorithm>
    </rule>
</tableRule>
<function name="murmur"
class="io.mycat.route.function.PartitionByMurmurHash">
    <!-- 默认是0-->
    <property name="seed">0</property>
```

```
    <!-- 要分片的数据库节点数量，必须指定，否则没法分片-->
    <property name="count">2</property>
    <!-- 一个实际的数据库节点被映射为多少个虚拟节点，默认是160 -->
    <property name="virtualBucketTimes">160</property>
    <!--
    <property name="weightMapFile">weightMapFile</property>
    节点的权重，没有指定权重的节点默认是1。以properties 文件的格式填写，以从0 开始到
count-1 的整数值也就是节点索引为key，以节点权重值为值。所有权重值必须是正整数，否则以1
代替-->
    <!--
    <property name="bucketMapPath">/etc/mycat/bucketMapPath</property>
    用于测试时观察各物理节点与虚拟节点的分布情况，如果指定了这个属性，会把虚拟节点的
murmur hash 值与物理节点的映射按行输出到这个文件，没有默认值，如果不指定，就不会输出任何
东西-->
</function>
```

10 应用指定

此规则是在运行阶段有应用自主决定路由到那个分片。

```
<tableRule name="sharding-by-substring">
    <rule>
        <columns>user_id</columns>
        <algorithm>sharding-by-substring</algorithm>
    </rule>
</tableRule>
<function name="sharding-by-substring"
        class="io.mycat.route.function.PartitionDirectBySubString">
    <property name="startIndex">0</property><!-- zero-based -->
    <property name="size">2</property>
    <property name="partitionCount">8</property>
    <property name="defaultPartition">0</property>
</function>
```

配置说明：

此方法为直接根据字符子串（必须是数字）计算分区号（由应用传递参数，显式指定分区号）。 例如id=05-100000002 在此配置中代表根据id 中从startIndex=0，开始，截取siz=2 位数字即05，05 就是获取的分区，如果没传 默认分配到defaultPartition

11 截取字符ASCII求和求模范围分片

此种规则类似于取模范围约束，只是计算的数值是取前几个字符的ASCII值和，再取模，再对余数范围分片。

```
<tableRule name="sharding-by-prefixpattern">
    <rule>
        <columns>user_id</columns>
        <algorithm>sharding-by-prefixpattern</algorithm>
    </rule>
</tableRule>
<function name="sharding-by-pattern"
        class="io.mycat.route.function.PartitionByPrefixPattern">
    <property name="patternValue">256</property>
    <property name="prefixLength">5</property>
    <property name="mapFile">partition-pattern.txt</property>
</function>
```

partition-pattern.tx

range start-end =data node index

```
#ASCII
#8-57=0-9 阿拉伯数字
#64、65-90=@、A-Z
#97-122=a-z
1-4=0       # 余数1-4的放到0号数据节点
5-8=1
9-12=2
13-16=3
17-20=4
21-24=5
25-28=6
29-32=7
0-0=7
```

配置说明：

- patternValue 即求模基数，
- prefixLength ASCII 截取的位数，求这几位字符的ASCII码值的和，再求余patternValue

- mapFile 配置文件路径，配置文件中配置余数范围分片规则。

3.7 主键值生成

在实现分库分表的情况下，数据库自增主键已无法保证自增主键的全局唯一。

```
CREATE TABLE t_customer(
    id BIGINT PRIMARY KEY,
    name varchar(100) not null,
    province int not null
);
```

```
<table name="t_customer" primaryKey="id" autoIncrement="true"
       dataNode="dn1,dn2,dn3" rule="sharding-by-province" />
```

为此，MyCat 提供了全局sequence，并且提供了包含本地配置和数据库配置等多种实现方式。

1 本地文件方式

原理：此方式MyCAT 将sequence 配置到文件中，当使用到sequence 中的配置后，MyCAT 会更新 conf中的sequence_conf.properties 文件中sequence 当前的值。

配置方式：

1、在sequence_conf.properties 文件中做如下配置：

```
GLOBAL.HISIDS=
GLOBAL.MINID=1001
GLOBAL.MAXID=1000000000
GLOBAL.CURID=1000
```

其中HISIDS 表示使用过的历史分段(一般无特殊需要可不配置)，MINID 表示最小ID 值，MAXID 表示最大 ID 值，CURID 表示当前ID 值。

2、server.xml 中配置：

```
<system><property name="sequnceHandlerType">0</property></system>
```

注：sequnceHandlerType 需要配置为0，表示使用本地文件方式。

使用示例：

```
insert into table1(id,name) values(next value for MYCATSEQ_GLOBAL,‘test’);
```

缺点：当MyCAT 重新发布后，配置文件中的sequence 会恢复到初始值。 优点：本地加载，读取速度较快。

为表配置主键自增值的序列：

规则：在sequence_conf.properties 中配置以表名为名的序列

```
T_COMPANY.CURID=501
T_COMPANY.MINID=1
T_COMPANY.MAXID=1000000000
```

就可以使用了。

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
            rule="range-sharding-by-members-count" />
```

```
INSERT INTO t_company(name,members) VALUES('company06',200);

select * from t_company;
```

2 数据库方式

原理

在数据库中建立一张表，存放sequence 名称(name)，sequence 当前值(current_value)，步长(increment int类型，每次读取多少个sequence)等信息；

Sequence 获取步骤：

1. 当初次使用该sequence 时，根据传入的sequence 名称，从数据库这张表中读取current_value，和 increment 到MyCat 中，并将数据库中的current_value 设置为原current_value 值+increment 值。
2. MyCat 将读取到current_value+increment 作为本次要使用的sequence 值，下次使用时，自动加1，当 使用increment 次后，执行步骤1)相同的操作。

MyCat 负责维护这张表，用到哪些sequence，只需要在这张表中插入一条记录即可。若某次读取的 sequence 没有用完，系统就停掉了，则这次读取的sequence 剩余值不会再使用。

配置方式：

server.xml 配置：

```
<system><property name="sequnceHandlerType">1</property></system>
```

注：sequnceHandlerType 需要配置为1，表示使用数据库方式生成sequence。

数据库配置：

1) 创建MYCAT_SEQUENCE 表

```
-- 创建存放sequence 的表
DROP TABLE IF EXISTS MYCAT_SEQUENCE;

-- name sequence 名称
-- current_value 当前value
-- increment 增长步长! 可理解为mycat 在数据库中一次读取多少个sequence. 当这些用完后,
下次再从数
据库中读取。
CREATE TABLE MYCAT_SEQUENCE (
    name VARCHAR(50) NOT NULL,
    current_value INT NOT NULL,
    increment INT NOT NULL DEFAULT 100,
    PRIMARY KEY(name));

-- 插入一条sequence
INSERT INTO MYCAT_SEQUENCE(name,current_value,increment) VALUES ('GLOBAL',
100000,
100);
```

2) 创建相关function

```
-- 获取sequence当前值(返回当前值,增量)的函数
DROP FUNCTION IF EXISTS mycat_seq_currval;

CREATE FUNCTION mycat_seq_currval(seq_name VARCHAR(50))
RETURNS varchar(64)
```

```
BEGIN
        DECLARE retval VARCHAR(64);
        SET retval='-999999999,null';
        SELECT concat(CAST(current_value AS CHAR),',',CAST(increment AS
CHAR)) INTO retval
        FROM MYCAT_SEQUENCE
        WHERE name = seq_name;
        RETURN retval;
END;

-- 设置sequence 值的函数
DROP FUNCTION IF EXISTS mycat_seq_setval;

CREATE FUNCTION mycat_seq_setval(seq_name VARCHAR(50),value INTEGER)
RETURNS varchar(64)
BEGIN
        UPDATE MYCAT_SEQUENCE
        SET current_value = value
        WHERE name = seq_name;
        RETURN mycat_seq_currval(seq_name);
END;

-- 获取下一个sequence 值
DROP FUNCTION IF EXISTS mycat_seq_nextval;
CREATE FUNCTION mycat_seq_nextval(seq_name VARCHAR(50))
RETURNS varchar(64)
BEGIN
        UPDATE MYCAT_SEQUENCE
        SET current_value = current_value + increment
        WHERE name = seq_name;
        RETURN mycat_seq_currval(seq_name);
END;
```

注意：MYCAT_SEQUENCE 表和以上的3 个function，需要放在同一个节点上。function 请直接在具体节点的数据库上执行，如果执行的时候报： you might want to use the less safe log_bin_trust_function_creators variable

需要对数据库做如下设置： windows 下my.ini[mysqld]加上 log_bin_trust_function_creators=1 linux 下/etc/my.cnf 下my.ini[mysqld]加上 log_bin_trust_function_creators=1 修改完后，即可在mysql 数据库中执行上面的函数。

3) sequence_db_conf.properties 相关配置,指定sequence 相关配置在哪个节点上：

例如：

```
USER_SEQ=test_dn1
```

使用示例：

```
insert into table1(id,name) values(next value for MYCATSEQ_GLOBAL,'test');
```

配置表的主键自增使用序列：

1 在序列定义表中增加名字为表名的序列：

```
INSERT INTO MYCAT_SEQUENCE(name,current_value,increment) VALUES
('T_COMPANY', 1,100);
```

2 在sequence_db_conf.properties中增加表的序列配置

```
T_COMPANY=dn1
```

3 主键自增就可以使用了

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
                        rule="range-sharding-by-members-count" />
```

```
INSERT INTO t_company(name,members) VALUES('company08',200);

select * from t_company;
```

3 本地时间戳方式

原理：

ID= 64 位二进制：42(毫秒)+5(机器ID)+5(业务编码)+12(重复累加)

换算成十进制为18 位数的long 类型，每毫秒可以并发12 位二进制的累加。

使用方式：

1 配置server.xml

```
<property name="sequnceHandlerType">2</property>
```

2 在mycat 下配置：sequence_time_conf.properties

```
WORKID=0-31     任意整数     表示机器id（或mycat实例id）
DATAACENTERID=0-31 任意整数     业务编码
```

多个mycat 节点下每个mycat 配置的WORKID，DATAACENTERID 不同，组成唯一标识，总共支持32*32=1024 种组合。

ID 示例：56763083475511 。

主键自增配置

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
                        rule="range-sharding-by-members-count" />
```

```
INSERT INTO t_company(name,members) VALUES('company09',200);

select * from t_company;
```

4 分布式ZK ID 生成器

```
<property name="sequnceHandlerType">3</property>
```

配置

1 Zk 的连接信息统一在myid.properties 的zkURL 属性中配置。**此只需关注zkURL。**

```
loadZk=false
zkURL=127.0.0.1:2181
clusterId=mycat-cluster-1
myid=mycat_fz_01
clusterSize=3
clusterNodes=mycat_fz_01,mycat_fz_02,mycat_fz_04
#server  booster  ;   booster install on db same server,will reset all
minCon to 2
type=server
boosterDataHosts=dataHost1
```

基于ZK 与本地配置的分布式ID 生成器，ID 结构：long 64 位，ID 最大可占63 位：

- |current time millis(微秒时间戳38 位,可以使用17 年)|clusterId（机房或者ZKid，通过配置文件配置5 位）|instanceId（实例ID，可以通过ZK 或者配置文件获取，5 位）|threadId（线程ID，9 位） |increment(自增,6 位)
- 一共63 位，可以承受单机房单机器单线程1000*(2^6)=640000 的并发。
- 无悲观锁，无强竞争，吞吐量更高

2 配置文件：sequence_distributed_conf.properties，只要配置里面：INSTANCEID=ZK 就是从ZK 上获取 InstanceID。(可以通过ZK 获取集群（机房）唯一InstanceID，也可以通过配置文件配置InstanceID)

测试：

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
                        rule="range-sharding-by-members-count" />
```

```
INSERT INTO t_company(name,members) VALUES('company10',200);

select * from t_company;
```

5 Zk 递增方式

```
<property name="sequnceHandlerType">4</property>
```

Zk 的连接信息统一在myid.properties 的zkURL 属性中配置。

配置：

配置文件：sequence_conf.properties 只要配置好ZK 地址和表名的如下属性

- TABLE.MINID 某线程当前区间内最小值
- TABLE.MAXID 某线程当前区间内最大值
- TABLE.CURID 某线程当前区间内当前值

文件配置的MAXID 以及MINID 决定每次取得区间，这个对于每个线程或者进程都有效。文件中的这三个属性配置只对第一个进程的第一个线程有效，其他线程和进程会动态读取ZK。

测试：

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
                        rule="range-sharding-by-members-count" />
```

```
INSERT INTO t_company(name,members) VALUES('company12',200);

select * from t_company;
```

6 last_insert_id() 问题

我们配置分片表主键自增。

```
<table name="t_company" primaryKey="id" autoIncrement="true"
dataNode="dn1,dn2,dn3"
                        rule="range-sharding-by-members-count" />
```

如需通过 select last_insert_id() 来获得自增主键值，则表定义中主键列需是自增的 AUTO_INCREMENT：

```
CREATE TABLE t_company(
        id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name varchar(100) not null,
        members int not null
);
```

如果没有指定 AUTO_INCREMENT，则select last_insert_id() 获取不到刚插入数据的主键值。

```
CREATE TABLE t_company(
        id BIGINT PRIMARY KEY,
    name varchar(100) not null,
        members int not null
);
```

Mybatis 中新增记录后获取last_insert_id 的示例：

```
<insert id="insert" parameterType="com.study.mike.user.model.User">
        insert into t_user (user_name,login_name,login_pwd,role_id)
    values(#{userName},#{loginName},#{loginPwd},#{roleId})
    <selectKey resultType="java.lang.Long" order="AFTER" keyProperty="id">
        select last_insert_id() as id
    </selectKey>
</insert>
```