Added by 郭 晓文, last edited by 朱 海清 on 四月 25, 2013

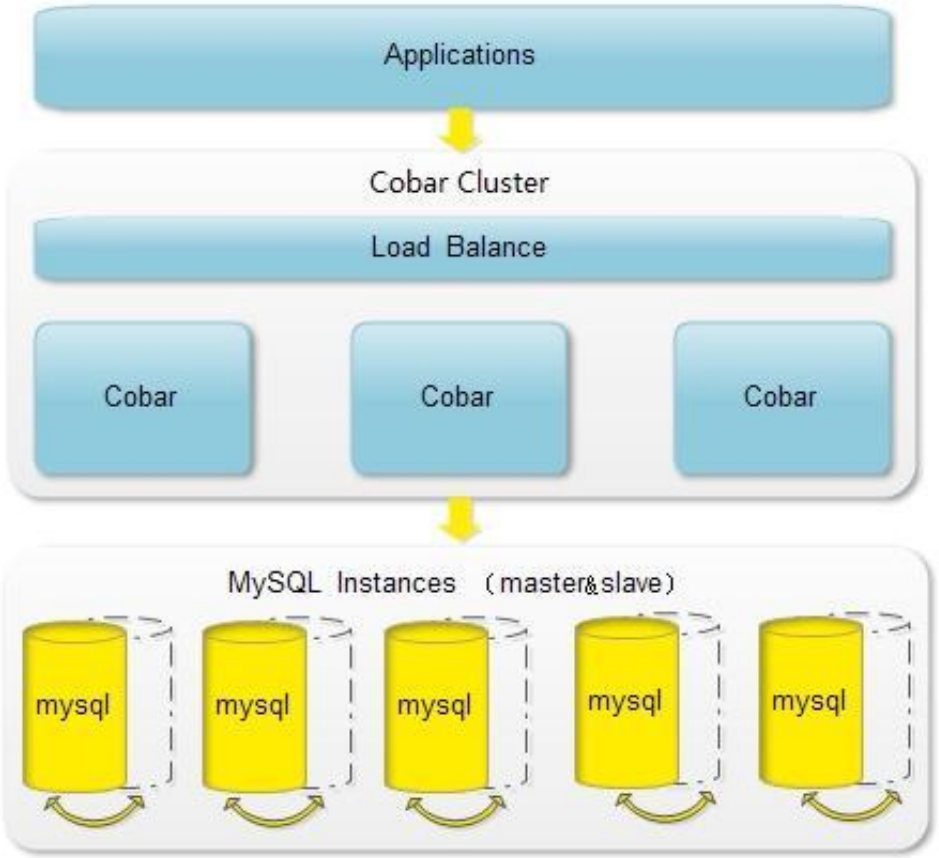
产品文档

1.产品约束

- 不支持跨库情况下的join、分页、排序、子查询操作。
- SET语句执行会被忽略，事务和字符集设置除外。
- 分库情况下，insert语句必须包含拆分字段列名。
- 分库情况下，update语句不能更新拆分字段的值。
- 不支持SAVEPOINT操作。
- 暂时只支持MySQL数据节点。
- 使用JDBC时，不支持rewriteBatchedStatements=true参数设置(默认为false)。
- 使用JDBC时，不支持useServerPrepStmts=true参数设置(默认为false)。
- 使用JDBC时，BLOB, BINARY, VARBINARY字段不能使用setBlob()或setBinaryStream()方法设置参数。

2.功能概述

Cobar是关系型数据的分布式处理系统，它可以在分布式的环境下看上去像传统数据库一样为您提供海量数据服务。

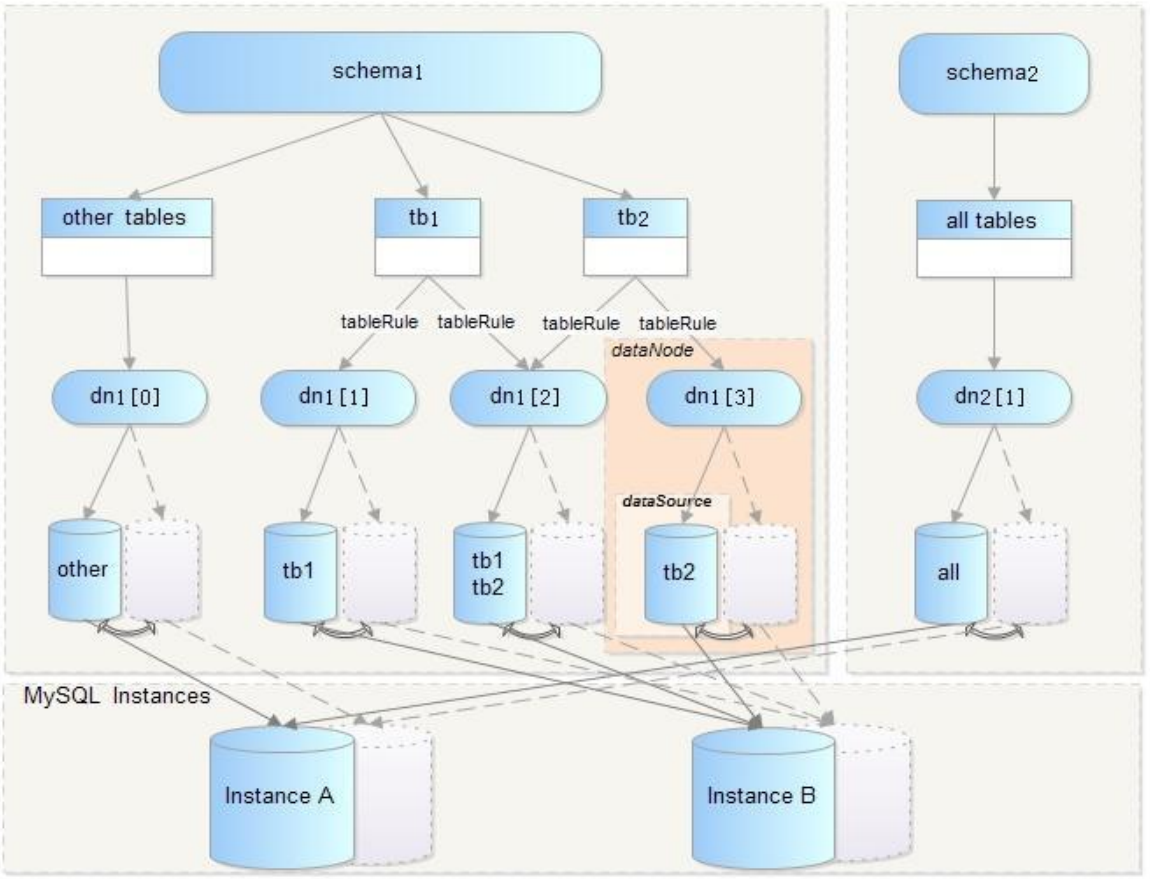


2.1 Cobar解决的问题

- 分布式：Cobar的分布式主要是通过将表放入不同的库来实现：
  1. Cobar支持将一张表水平拆分成多份分别放入不同的库来实现表的水平拆分
  2. Cobar也支持将不同的表放入不同的库
  3. 多数情况下，用户会将以上两种方式混合使用
  这里需要强调的是，Cobar不支持将一张表，例如test表拆分成test_1, test_2, test_3.....放在同一个库中，必须将拆分后的表分别放入不同的库来实现分布式。

- HA：
  在用户配置了MySQL心跳的情况下，Cobar可以自动向后端连接的MySQL发送心跳，判断MySQL运行状况，一旦运行出现异常，Cobar可以自动切换到备机工作。但需要强调的是：
  1. Cobar的主备切换有两种触发方式，一种是用户手动触发，一种是Cobar的心跳语句检测到异常后自动触发。那么，当心跳检测到主机异常，切换到备机，如果主机恢复了，需要用户手动切回主机工作，Cobar不会在主机恢复时自动切换回主机，除非备机的心跳也返回异常。
  2. Cobar只检查MySQL主备异常，不关心主备之间的数据同步，因此用户需要在使用Cobar之前在MySQL主备上配置双向同步，详情可以参阅MySQL参考手册。

2.2 Cobar逻辑层次图



- dataSource：数据源，表示一个具体的数据库连接，与一个物理存在的schema一一对应。
- dataNode：数据节点，由主、备数据源，数据源的HA以及连接池共同组成，可以将一个dataNode理解为一个分库。
- table：表，包括拆分表(如tb1，tb2)和非拆分表。
- tableRule：路由规则，用于判断SQL语句被路由到具体哪些datanode执行。
- schema：cobar可以定义包含拆分表的schema(如schema1)，也可以定义无拆分表的schema(如schema2)。
- 以上层次关系具有较强的灵活性，用户可以将表自由放置不同的datanode，也可将不同的datasource放置在同一MySQL实例上。

2.3 Cobar使用注意事项

1.请注意表的拆分方式，一张表水平拆分多份到不同的库中，而不是放入同一个库中。
2.如果使用HA功能，请在MySQL主备之间配置双向同步

3.使用手册

3.1 目录结构

- 基本目录
  如果您还没有下载Cobar，请先进入Cobar Release下载最新Cobar压缩包。解压后，进入cobar-server-1.2.4目录，可以看到Cobar的主要目录如下：

```
bin    #包含Cobar的启动、重启、停止等脚本文件
conf   #包含Cobar所有配置文件
lib    #包含Cobar及其依赖的jar文件
logs   #包含Cobar所有日志文件
```

- 启动脚本
  Cobar的所有启动停止脚本全部放在bin目录中，进入bin目录，可以看到：

```
startup.sh   #Linux环境启动脚本
startup.bat  #Windows环境启动脚本
restart.sh   #Linux环境重启脚本
shutdown.sh  #Linux环境停止脚本
```

- 配置文件
  Cobar的所有配置文件全部放在conf目录中，进入conf目录，可以看到：

```
server.xml   #Cobar系统、用户、集群等相关配置
schema.xml   #schema，dataNode，dataSource相关配置
rule.xml     #分布式规则定义
log4j.xml    #日志相关配置
```

3.2 配置详解

- **schema.xml**
  schema.xml中定义了schema逻辑层次图中的所有元素，并利用这些元素以及rule.xml中定义的规则组建分布式数据库系统
  - dataSource

```
<!-- 数据源定义，数据源是一个具体的后端数据连接的表示。-->
  <!--数据源的名称与类型，Cobar暂时只支持mysql这种类型-->
  <dataSource name="ds_single_master" type="mysql">
    <!--连接的地址，端口和schema名称-->
    <property name="location">
      <location>192.168.0.4:3306/single</location>
    </property>

     <!--连接用户名，密码-->
    <property name="user">test</property>
    <property name="password"></property>

    <!--连接的SQL模式-->
    <property name="sqlMode">STRICT_TRANS_TABLES</property>
  </dataSource>
```

上面给出的是单个数据源的配置，名称为ds_single_master, Cobar支持把多个用户名密码相同的数据源合并配置在同一个标签中,如下：

```
<dataSource name="ds_shard_master" type="mysql">
    <property name="location">
      <location>192.168.0.4:3306/shard</location>
      <!--shard$1-3是shard1、shard2、shard3的缩写，在后续介绍的Cobar配置中，我们也会经常看到类似的缩写形式-->
      <location>192.168.0.4:3306/shard$1-3</location>
    </property>
    <property name="user">test</property>
    <property name="password"></property>
    <property name="sqlMode">STRICT_TRANS_TABLES</property>
  </dataSource>
```

上例中配置了4个数据源，数据源名称分别为ds_shard_master[0]、ds_shard_master[1]、ds_shard_master[2]、ds_shard_master[3]，按照用户location中配置的顺序，对应关系如下:

数据源名称	数据源地址
ds_shard_master[0]	192.168.0.4:3306/shard
ds_shard_master[1]	192.168.0.4:3306/shard1
ds_shard_master[2]	192.168.0.4:3306/shard2
ds_shard_master[3]	192.168.0.4:3306/shard3

- dataNode

```
<!--数据节点由主、备数据源，心跳，连接池等配置组成。-->
  <!--数据节点名称-->
  <dataNode name="dn_single">
    <property name="dataSource">
      <!--第一行dataSourceRef表示主数据源-->
      <dataSourceRef>ds_single_master</dataSourceRef>
      <!--第二行dataSourceRef表示备数据源-->
      <dataSourceRef>ds_single_slave</dataSourceRef>
      <!--如果需要一主多备的情况，可以将第二备数据源配置在第三行，以此
类推-->
      <dataSourceRef>ds_single_slave2</dataSourceRef>
    </property>

    <!--Cobar与后端数据源连接池大小设置-->
    <property name="poolSize">256</property>

    <!--Cobar通过心跳来实现后端数据源HA，一旦主数据源心跳失败，便切换到
备数据源上工作-->
    <!--Cobar心跳是通过向后端数据源执行一条SQL语句，根据该语句的返回结
果判断数据源的运行情况-->
    <property name="heartbeatSQL">select user()</property>    （1.2.7版
本用这个配置）
    <property name="heartbeat">select user()</property>    （1.2.6及以前
版本用这个配置）
  </dataNode>
```

注意：心跳SQL语句可以由用户自定义，数据源心跳配置介绍了一种我们更推荐的配置方式, 后面的例子中，我们也将会用这种推荐的方式配置心跳。
同数据源配置类似，可以将多个数据节点配置合并,只需在每行dataSourceRef中配置多个数据源即可

```
<dataNode name="dn_shard">
    <property name="dataSource">
      <!--三个数据节点的主数据源，可用逗号分隔，支持$1-3的缩写形式， 表
示ds_shard_master[1], ds_shard_master[2], ds_shard_master[3]-->
      <dataSourceRef>ds_shard_master[0], ds_shard_master$1-
3</dataSourceRef>
      <!--三个数据节点的备数据源，必须与主数据源一一对应(个数，顺序都要
对应)-->
      <dataSourceRef>ds_shard_slave$0-3</dataSourceRef>
    </property>
    <property name="poolSize">256</property>
    <property name="heartbeatSQL">update xdual set x=now() where
id=${(1,10)}</property>
  </dataNode>
```

上例中配置了三个数据节点，按照顺序，名称分别为dn_shard[0], dn_shard[1], dn_shard[2]，dn_shard[3]三个节点的连接池大小与心跳语句都相同，数据节点的主备对应关系如下

节点名称	主数据源	备数据源
dn_shard[0]	ds_shard_master[0]	ds_shard_slave[0]
dn_shard[1]	ds_shard_master[1]	ds_shard_slave[1]
dn_shard[2]	ds_shard_master[2]	ds_shard_slave[2]
dn_shard[3]	ds_shard_master[3]	ds_shard_slave[3]

注意：Cobar根据每行dataSourceRef中数据源的顺序来对应主备，因此每行dataSourceRef里的数据源不仅个数必须相同，顺序也必须一一对应。如果用户把上例中dataSourceRef的顺序调整一下

```
<dataSourceRef>ds_shard_master[1], ds_shard_master[0],
ds_shard_master$2-3</dataSourceRef>
<dataSourceRef>ds_shard_slave$0-3</dataSourceRef>
```

主备对应关系立即改变，因此配置时请注意顺序

节点名称	主数据源	备数据源
dn_shard[0]	**ds_shard_master[1]**	**ds_shard_slave[0]**
dn_shard[1]	**ds_shard_master[0]**	**ds_shard_slave[1]**
dn_shard[2]	ds_shard_master[2]	ds_shard_slave[2]
dn_shard[3]	ds_shard_master[3]	ds_shard_slave[3]

- schema
  schema定义了Cobar展示给用户的schema，schema由dataNode以及rule.xml中定义的路由规则共同组成。

```
<!--定义schema db_single，该schema未做分布式存储，类似于5.1节中schema逻
辑层次图中的db2-->
<!--对schema db_single所有表的访问均路由到dn_single上执行-->
<schema name="db_single" dataNode="dn_single"/>

<!--定义schema db_shard-->
<!--所有除tb1，tb2，tb3，tb4之外表的访问都路由到dn_shard[0]去执行-->
<schema name="db_shard" dataNode="dn_shard[0]">
 <!--对tb1的访问会根据规则tb1Rule路由到dn_shard$0-3的某一个或某几个
datanode执行-->
 <table name="tb1" dataNode="dn_shard$0-3" rule="tb1Rule" />
 <!--对tb2的访问会根据规则string_val_2路由到dn_shard[0]，dn_shard[3]的
某一个或两个datanode执行-->
 <table name="tb2" dataNode="dn_shard[0], dn_shard[3]" rule="tb2Rule"
/>
 <!--对tb3的访问会根据规则tb3Rule路由到dn_shard$0-3的某一个或某几个
datanode执行-->
 <table name="tb3" dataNode="dn_shard$0-3" rule="tb3Rule" />
 <!--对tb4的访问会直接路由到dn_shard[2]执行，如果datanode中只有一个
node，可以不用配置rule-->
 <table name="tb4" dataNode="dn_shard[2]" ruleRequired="false" />
</schema>
```

```
<!--tb1Rule, tb2Rule, tb3Rule将会在rule.xml的tableRule中定义-->
```

- **rule.xml**
  - tableRule
    tableRule主要作用是用来判断SQL语句路由到哪些datanode执行，Cobar是通过在SQL中提取一个或多个字段的值，并根据这些字段的值来决定路由到哪个库执行。因此，tableRule定义两个要素：
    1)按表中的哪个字段路由？------下文中我们称此字段为路由字段
    2)有了字段值，如何路由？------即路由函数

```
<!--tableRule名称-->
<tableRule name="tb1Rule">
    <rule>
      <!--id为路由字段， id是int型字段-->
      <columns>id</columns>
      <!--int_4是路由函数，参数为id，int_4在function中定义，稍后介绍-->
      <algorithm><![CDATA[int_4(${id})]]></algorithm>
    </rule>
</tableRule>
<tableRule name="tb2Rule">
    <rule>
      <!--val为路由字段， val是varchar型字段-->
      <columns>val</columns>
      <!--string_2是路由函数，参数为val-->
      <algorithm><![CDATA[string_2(${id})]]></algorithm>
    </rule>
</tableRule>
<tableRule name="tb3Rule">
    <rule>
      <!--id和val共同组成路由字段， id是int型字段，val是varchar型字段--
>
      <columns>id, val</columns>
      <algorithm><![CDATA[ twoDimensionFunc(${id},${val})]]>
</algorithm>
    </rule>
    <!--按多个字段路由时需要考虑SQL中只有一个字段的情况-->
    <rule>
      <!--当SQL语句中只有id，无val字段时，匹配此规则，val参数设置为
null-->
      <columns>id</columns>
      <algorithm><![CDATA[ twoDimensionFunc(${pid},null)]]></algorithm>
    </rule>
    <rule>
      <!--当SQL语句中只有val，无id字段时，匹配此规则，id参数设置为null-
->
      <columns>val</columns>
      <algorithm><![CDATA[ twoDimensionFunc(null,${uid})]]></algorithm>
    </rule>
</tableRule>
```

  - function
    Cobar支持按1-2个字段做路由，因此路由算法分为单维路由和多维路由（2维），关于路由算法参见路由算法。

- **server.xml**
  - system

```
<!--system中包含系统参数定义，服务端口、管理端口，处理器个数、线程池等
定义-->
```

```
<!--这些配置都有默认值，一般情况下用户可以完全不用配置-->
<system>
    <!--端口定义，如果没有端口冲突，可不用配置-->
    <!--Cobar服务端口，通过此端口执行SQL语句，默认值8066-->
    <property name="serverPort">8066</property>
    <!--Cobar管理端口，通过此端口执行Cobar管理命令， 默认值9066-->
    <property name="managerPort">9066</property>

    <!--Cobar内部处理器个数，线程池等定义，默认值为Cobar所在机器处理器
个数-->
    <!--这些配置主要影响Cobar内部处理性能，可在做性能优化时调整-->
    <!--initExecutor:处理初始化任务的线程-->
    <property name="initExecutor">16</property>
    <!--timerExecutor:处理定时任务的线程-->
    <property name="timerExecutor">4</property>
    <!--managerExecutor:处理来自9066端口任务的线程-->
    <property name="managerExecutor">4</property>
    <!--processors:Cobar内部处理器个数，默认与系统cpu个数相同-->
    <property name="processors">4</property>
    <!--processorHandler:前端处理线程，负责处理所有8066端口前端连接-->
    <property name="processorHandler">8</property>
    <!--processorExcutor:后端处理线程，负责处理Cobar与MySQL之间的连接，
可以适当设置大一些-->
    <property name="processorExecutor">8</property>

    <!--Cobar与Cobar间心跳的用户名和密码，默认值即是_HEARTBEAT_USER_和
_HEARTBEAT_PASS_-->
    <!--如果两台Cobar之间需要心跳，这两项配置必须相同,一般不建议自行配
置，使用默认值即可-->
    <property name="clusterHeartbeatUser">_HEARTBEAT_USER_</property>
    <property name="clusterHeartbeatPass">_HEARTBEAT_PASS_</property>
  </system>
```

- user

```
<!--Cobar的用户定义，包括用户名，密码以及访问权限设置，可以配置任意多个
用户-->
  <!--普通用户配置-->
  <user name="test">                                <!--用户名-->
    <property name="password">test</property>   <!--密码-->
    <!--Cobar对用户进行了简单的权限控制，可定义用户对某些schema是否有访
问权限-->
    <!--这里不区分读写权限，一旦拥有访问权限，便是读写权限-->
    <!--以逗号分隔多个schema，test用户只能访问db_single和db_shard两个
Cobar定义的schema-->
    <property name="schemas">db_single，db_shard</property>
  </user>

  <!--超级用户配置，超级用户是指对所有schema都有访问权限的用户-->
  <user name="root1">
    <property name="password">root</property>
    <!--不配置任何访问权限，表示对所有schema都有访问权限-->
  </user>
  <user name="root2">
    <property name="password">root</property>
    <!--配置为空同样表示对所有schema都有访问权限-->
    <property name="schemas"></property>
  </user>
```

- cluster

在实际应用中，经常需要部署一个Cobar集群，我们称集群中的一台Cobar为一个Cobar节点。

```
<!--组建一个Cobar集群，只需在cluster配置中把所有Cobar节点（注意：包括当前Cobar自身）都配置上便可-->
<cluster>
    <!--node名称，一个node表示一个Cobar节点，一旦配置了node，当前Cobar便会向此节点定期发起心跳，探测节点的运行情况-->
    <node name="cobar1">
      <!--Cobar节点IP，表示当前Cobar将会向192.168.0.1上部署的Cobar发送心跳-->
      <property name="host">192.168.0.1</property>
      <!--节点的权重，用于客户端的负载均衡，用户可以通过命令查询某个节点的运行情况以及权重-->
      <property name="weight">1</property>
    </node>
    <!--当前Cobar将会向192.168.0.2上部署的Cobar发送心跳-->
    <node name="cobar2">
      <property name="host">192.168.0.2</property>
      <property name="weight">2</property>
    </node>
    <!--当前Cobar将会向192.168.0.3上部署的Cobar发送心跳-->
    <node name="cobar3">
      <property name="host">192.168.0.3</property>
      <property name="weight">3</property>
    </node>
    <!--用户还可以将Cobar节点分组，以便实现schema级别的细粒度负载均衡-->
    <group name="group12">
      <property name="nodeList">cobar1,cobar2</property>
    </group>
     <group name="group23">
      <property name="nodeList">cobar2,cobar3</property>
    </group>
</cluster>
```

用户只需登录Cobar的服务端口(8066)，运行Cobar自带的查询命令show cobar_cluster，便可查询集群中所有节点的运行情况以及权重，并根据查询结果做负载均衡。

```
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>show cobar_cluster;  #查询cluster配置中正常的Cobar节点
+---------------+--------+
| HOST          | WEIGHT |
+---------------+--------+
| 192.168.0.1   |      1 |
| 192.168.0.2   |      2 |
| 192.168.0.3   |      3 |
+---------------+--------+
```

注意：

1）如果需要配置Cobar集群，当前Cobar自身也需要作为一个节点配置在cluster中，Cobar不会默认向自己发心跳；

2）show cobar_cluster只显示cluster配置中得正常Cobar节点，如果节点异常（如超时或错误），结果中便不会包含此节点。

3）3.4节会介绍更多关于show cobar_cluster以及group配置的用处，详见3.4节->cobar自定义语句->show cobar_status/show cobar_cluster。

在了解show cobar_cluster使用后，用户可以根据如何在多个cobar节点之间做负载均衡选择适合的负载均衡策略。

- quarantine

```
<!-- 隔离区定义，出于安全角度考虑，限定某个主机上只允许某个用户登录。 -->
 <quarantine>
    <!--从192.168.0.110这台主机连接当前Cobar，只能使用test， test1这两个普通用户登录，不可用root1, root2登录-->
    <host name="192.168.0.110">
      <!--注意：此处配置的用户必须为普通用户，不可以配置超级用户，多个用户用逗号分隔-->
      <property name="user">test， test1</property>
    </host>
    <!--从192.168.0.120这台主机连接当前Cobar，只能使用test用户登录，不可用root1，root2登录-->
    <host name="192.168.0.120">
      <property name="user">test</property>
    </host>
  </quarantine>
```

注意：隔离区内不允许配置当前集群中任何Cobar节点的IP。

3.3 示例场景

典型场景

3.4 访问Cobar

- 访问方式
  由于Cobar遵循MySQL协议，访问Cobar的方式与访问MySQL数据库完全相同。
  - 支持MySQL命令行方式访问

```
#命令行
mysql -h192.168.0.1 -utest -ptest -P8066 -Ddb_shard
```

  - 支持JDBC方式访问，支持用户使用JDBC连接池

```
#JDBC(建议5.1以上的mysql driver版本)
Class.forName("com.mysql.jdbc.Driver");
Connection conn =
DriverManager.getConnection("jdbc:mysql://192.168.0.1:8066/db_shard",
"test", "test");
......
```

- **SQL**语句
  - SQL中的路由字段
    在有分库的情况下，Cobar会提取SQL中的路由字段值判断此SQL被路由到哪个一分库执行，假如用户的SQL语句中没有路由字段值，Cobar将会把SQL分发到所有分库执行。
    假设表tb1包含id(INT)，val(VARCHAR)和gmt(DATE)三列，如果用户直接使用MySQL，下面两句SQL语句的执行结果完全相同。

```
mysql>insert into tb1 values (1, "part1", now())
mysql>insert into tb1(id, val, gmt) values (1, "part2", now())
```

如果用户将tb1按id拆分成两份放在不同的MySQL实例中，然后通过Cobar执行上述两句SQL，结果便不同。

```
#由于id为路由字段，Cobar在SQL语句中无法提取id这个字段，因此将会把此SQL分发到两个分库执行，这样后端两个MySQL分库都会新增一条记录
mysql>insert into tb1 values (1, "part1", now())
#Cobar提取出路由字段id的值为1，便能够根据路由函数做出判断，这样后端只有一个MySQL分库新增记录
mysql>insert into tb1(id, val, gmt) values (1, "part2", now())
```

因此,SQL中显示指定拆分字段非常重要，而且一条记录的拆分字段值不希望被更改

```
# 如果用户修改拆分字段值，cobar将会报错
mysql>update t1 set id=2 where id =1;
ERROR 1064 (HY000): partition key cannot be changed
```

- SQL中显示指定Schema
  默认情况下：
  1） 如果当前连接的schema没有拆分表，如上例中的db_single，则SQL语句中显示指定的schema会原样发送给后端MySQL数据库；
  2） 如果当前连接的schema包含拆分表，如上例中的db_shard, 则SQL语句中显示指定的schema都将被删除掉。
  如果用户需要将SQL中显示指定的schema带入到后端MySQL数据库，需要在schema.xml的schema标签中这样配置

```
<!--比原有配置增加一个keepSqlSchema配置-->
<schema name="db_single" dataNode="dn_single" keepSqlSchema="true"/>

<!--比原有配置增加一个keepSqlSchema配置-->
<schema name="db_shard" dataNode="dn_shard[0]" keepSqlSchema="true">
 <table name="tb1" dataNode="dn_shard$0-3" rule="tb1Rule" />
 <table name="tb2" dataNode="dn_shard[0], dn_shard[3]" rule="tb2Rule"
/>
 <table name="tb3" dataNode="dn_shard$0-3" rule="tb3Rule" />
 <table name="tb4" dataNode="dn_shard[2]" ruleRequired="false" />
 </schema>
```

  这样配置后，Cobar处理方式变为：
  1） 如果SQL中指定的schema与当前连接的schema相同，则Cobar首先删除SQL中的schema，再路由至后端，如果不同，则原样发送到后端MySQL执行。
  2） 如果当前连接的schema包含拆分表，如上例中的db_shard, 且按照第一条判断需要原样发送到后端MySQL执行，则发送到default datanode中，即上例中的dn_shard[0]中。
- 切换schema
  Cobar不允许再同一个连接中切换schema。如果在db_shard的连接中执行use db_single，Cobar会报错

```
mysql -h192.168.0.1 -utest -ptest -P8066 -Ddb_shard
mysql>use db_single
ERROR 1044 (HY000): Not allowed to change the database!
```

- 数据库管理语句

- 数据库定义语句
  不推荐用户通过Cobar执行数据库定义语句。
- **Cobar**自定义语句
  - hint
  - show cobar_status/cobar_cluster
    show cobar_status查询单台Cobar的状态。show cobar_cluster查询集群中所有Cobar节点状态。

```
# Cobar启动默认状态为ON
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>show cobar_status;
+--------+
| STATUS |
+--------+
| ON     |
+--------+
```

用户通过Cobar提供的管理命令online和offline来改变Cobar本身的状态

```
# 管理命令通过9066端口执行，管理手册将会详细介绍各种命令
mysql -h192.168.0.1 -utest -ptest -P9066
#offline将Cobar的状态由ON变为OFF
mysql>offline;

# 通过8066端口查看cobar状态，变为OFF，结果返回为shutdown
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>show cobar_status;
ERROR 1053 (HY000): The server has been shutdown
```

在产品文档我们介绍了可以在server.xml中定义一个Cobar集群和多个组，且一旦Cobar节点被定义在集群中，当前Cobar便会向用户所配置的Cobar节点发送心跳语句探知节点的运行情况。这里的心跳语句便是上面介绍的show cobar_status。当集群中每个Cobar都向互相发送心跳语句，用户访问任何一台Cobar，并执行show cobar_cluster命令，便可查看整个集群中节点的运行情况。server.xml配置详解中我们也曾有介绍：

```
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>show cobar_cluster;  #查询cluster配置中正常的Cobar节点
+---------------+--------+
| HOST          | WEIGHT |
+---------------+--------+
| 192.168.0.1   |      1 |
| 192.168.0.2   |      2 |
| 192.168.0.3   |      3 |
+---------------+--------+
```

show cobar_cluster查询出的是状态为ON的Cobar节点，假如某Cobar节点的状态为OFF或该Cobar未启动或该Cobar执行心跳语句超时等情况下，show cobar_cluster的结果中不会显示该cobar节点。用户可以尝试登录192.168.0.2节点，并利用offline语句将该节点的状态设置为OFF，一个心跳周期后，其他Cobar节点便可探知该节点的状态。再次登录192.168.0.1，将会看到如下结果：

```
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>show cobar_cluster;  #查询cluster配置中正常的Cobar节点
+---------------+--------+
| HOST          | WEIGHT |
```

```
+---------------+--------+
| 192.168.0.1   |      1 |
| 192.168.0.3   |      3 |
+---------------+--------+
```

利用show cobar_cluster这种特性，用户便可以自定义适合自己项目的负载均衡策略。除此之外，cobar还提供了更细粒度的负载均衡方法。用户可定义某些Cobar节点只为部分schema提供服务。在3.2配置详解中，server.xml的cluster配置中我们配置group，以group12为例，group12包含两个cobar节点cobar1和cobar2。

```
#server.xml中cluster配置截取
<cluster>
     ......
    <group name="group12">
      <property name="nodeList">cobar1,cobar2</property>
    </group>
    ......
</cluster>
```

此时我们在schema.xml的schema标签中增加schema与group的对应配置

```
# schema.xml中schema配置截取,配置db_single对应的group为group12
......
<schema name="db_single" dataNode="dn_single" group="group12"/>
......
```

假设所有的Cobar节点状态均为ON，如果我们在db_single的连接中执行show cobar_cluster，结果将只显示gourp12中的Cobar节点

```
# 通过db_single查询，结果中只包含了gourp12中的cobar1和cobar2节点
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>use db_single
mysql>show cobar_cluster
+---------------+--------+
| HOST          | WEIGHT |
+---------------+--------+
| 192.168.0.1   |      1 |
| 192.168.0.2   |      2 |
+---------------+--------+

#由于db_shard没有配置group，通过db_shard查询，便可查询到所有节点
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>use db_shard
mysql>show cobar_cluster
# 结果中只包含了gourp12中的cobar1和cobar2节点
+---------------+--------+
| HOST          | WEIGHT |
+---------------+--------+
| 192.168.0.1   |      1 |
| 192.168.0.2   |      2 |
| 192.168.0.3   |      3 |
+---------------+--------+
```

只要系统中还存在状态为ON的Cobar节点，Cobar希望用户使用show cobar_cluster时总能查询到可

用Cobar节点，假设cobar1和cobar2这两个节点状态均为OFF，而cobar3为ON，此时group12中没有可用节点，而db_single对应的group正是group12，此时如果用户通过db_single查询show cobar_cluster，Cobar会返回给用户cobar3节点。

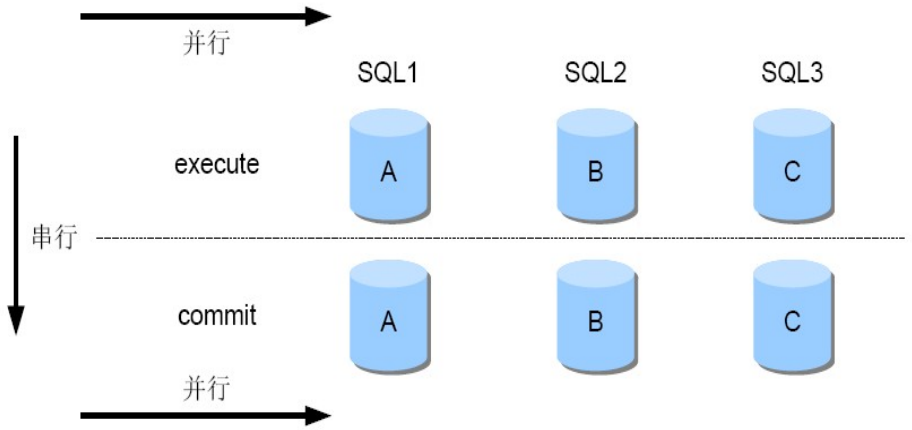
```
# 假设group12中无可用节点，通过db_single查询，结果中显示cobar3节点，即使cobar3并不在group12中
mysql -h192.168.0.1 -utest -ptest -P8066
mysql>use db_single
mysql>show cobar_cluster
+---------------+--------+
| HOST          | WEIGHT |
+---------------+--------+
| 192.168.0.3   |      3 |
+---------------+--------+
```

- explain
  如果用户想查看Cobar是如何路由自己的SQL语句，只需在explain后加上该SQL，Cobar便会向用户展示该SQL被路由到哪些datanode执行。

```
mysql -h192.168.0.1 -utest -ptest -P8066
# 语句按id拆分成两句，分别路由到不同的datanode执行
mysql>explain select * from tb1 where id in (1, 257);
+----------------+--------------------------------------+
| DATA_NODE      | SQL                                  |
+----------------+--------------------------------------+
| dn_shard[0]    |  select * from tb1 where id in (1)   |
| dn_shard[1]    |  select * from tb1 where id in (257) |
+----------------+--------------------------------------+
```

  explain的SQL并未真正的发送到后端MySQL执行，Cobar仅仅向用户展示该语句会被发送到后端哪些datanode执行以及在这些datanode上执行的实际SQL语句。
- show datasource
  通过8066端口执行，展示该Cobar中所有的datasource配置，便于用户查看后端MySQL物理地址。

- **Cobar**事务
  - Cobar在单库的情况下保持事务的强一致性，分库的情况下保持事务的弱一致性。
    分库事务包含两个阶段，执行阶段和提交阶段



  执行阶段：SQL语句按照路由规则被路由到单个或者多个分库(如图A、B、C)并行执行，如果SQL语句在执行阶段发生错误，可以回滚，处理方式参见连接和执行异常处理。
  提交阶段：各个分库并行提交事务，提交阶段出错，事务将无法正确回滚。
  - Cobar的隐含事务
    当用户没有显示指定事务，且用户的SQL语句被路由到多个分库执行写操作，Cobar会默认处理成事务

```
mysql>update tb1 set val="newval" where pid in (1, 513, 1023); #假设这个update操作被分发到了后端3个不同的分库执行，则Cobar按照事务处理
```

4.管理手册

4.1 入口

Cobar通过9066端口向用户提供了一些管理和监控命令。

```
# 9066端口暂时未做权限控制，普通用户和超级用户均可登陆
mysql -h192.168.0.1 -utest -ptest -P9066
# 通过show @@help可以查看9066端口支持的所有命令,以及简单解释
mysql>show @@help;
```

下面我们将分类介绍这些命令的作用

4.2 监控命令

- 数据源、数据节点查看
  - show @@datanode [where schema = ?] ------ 查看系统中配置的数据节点，支持查询某个schema对应的数据节点

```
mysql>show @@datanode;
+-------------+-----------------------------------------+-------+-------+-
-------+------+------+---------+-----------+----------+---------+-----
----------+
| NAME        | DATASOURCES                             | INDEX | TYPE  |
ACTIVE | IDLE | SIZE | EXECUTE | TOTAL_TIME | MAX_TIME | MAX_SQL |
RECOVERY_TIME |
+-------------+-----------------------------------------+-------+-------+-
-------+------+------+---------+-----------+----------+---------+-----
----------+
| dn_shard[0] | ds_shard_master[0],ds_shard_slave[0] |     0 | mysql
|      0 |    1 |  256 |       0 |          0 |        0 |       0
|           -1 |
| dn_shard[1] | ds_shard_master[1],ds_shard_slave[1] |     0 | mysql
|      0 |    1 |  256 |       0 |          0 |        0 |       0
|           -1 |
| dn_shard[2] | ds_shard_master[2],ds_shard_slave[2] |     0 | mysql
|      0 |    1 |  256 |       0 |          0 |        0 |       0
|           -1 |
| dn_shard[3] | ds_shard_master[3],ds_shard_slave[3] |     0 | mysql
|      0 |    1 |  256 |       0 |          0 |        0 |       0
|           -1 |
| dn_single   | ds_single_master,ds_single_slave     |     1 | mysql
|      0 |    1 |  256 |       0 |          0 |        0 |       0
|           -1 |
+-------------+-----------------------------------------+-------+-------+-
-------+------+------+---------+-----------+----------+---------+-----
----------+
# NAME:  datanode名称。
# DATASOURCES:  datanode对应的主备datasource，按照结果列出的顺序依次为主、备、第二备.....,Cobar依次为其编号为0、1、2.....依次类推。
# INDEX:  指示datanode正在使用的数据源，如dn_shard[0]的INDEX为0，就表示
```

```
此datanode正在使用ds_shard_master[0]这个数据源，而dn_single的INDEX为1，正在使用ds_single_slave数据源。
# TYPE：  数据源类型
# ACTIVE|IDLE|SIZE：  Cobar连接数据源的连接池的大小，以及Active和Idle的连接数
# EXECUTE ：通过该datanode执行的SQL语句条数
# TOTAL_TIME|MAX_TIME |MAX_SQL：  预留字段，暂时未使用
# RECOVERY_TIME：该datanode的心跳被停止后，还有多久会自动恢复，后面介绍停止心跳语句时，会再次介绍此字段
```

- show @@datasource [where schema = ?] ------ 查看系统中配置的数据源，支持查询某个schema对应的数据源, show @@datasource结果中各列的意义比较清晰，这里不再赘述。
- show @@database ------ 等同于在8066端口的show databases语句。

- **sql**执行状况查询
  - show @@sql.slow ------ 慢sql查询

```
mysql>show @@sql.slow;
```

  - show @@slow where datanode = ? /clear @@slow where datanode = ? ------ 根据datanode查询慢sql，利用clear清除该datanode当前的慢sql
  - show @@slow where schema = ? / clear @@slow where schema= ? ------ 根据schema查询慢sql，利用clear清除该schema当前慢sql
  - show @@sql where id = ?
  - show @@sql.detail where id = ?
  - show @@sql.execute
- 内部状态查询
  - show @@command ------ Cobar执行语句的个数统计

```
mysql>show @@command;
+------------+---------+-------+--------------+--------------+---------
---+--------+------+------+-------+
| PROCESSOR  | INIT_DB | QUERY | STMT_PREPARE | STMT_EXECUTE |
STMT_CLOSE | PING   | KILL | QUIT | OTHER |
+------------+---------+-------+--------------+--------------+---------
---+--------+------+------+-------+
| Processor0 |       2 |    48 |            0 |            0 |
0 |      0 |    0 |    2 |     0 |
| Processor1 |       4 |   125 |            0 |            0 |
0 | 137408 |    0 |    6 |     0 |
+------------+---------+-------+--------------+--------------+---------
---+--------+------+------+-------+
# PROCESSOR ： 处理器名称
#
INIT_DB|QUERY|STMT_PREPARE|STMT_EXECUTE|STMT_CLOSE|PING|KILL|QUIT|OTHER
： cobar执行语句的类型
```

  - show @@processor ------ Cobar处理器状态查询

```
mysql>show @@processor;
+------------+---------+----------+-------------+---------+---------+--
----------+--------------+----------+----------+
| NAME       | NET_IN  | NET_OUT  | REACT_COUNT | R_QUEUE | W_QUEUE |
FREE_BUFFER | TOTAL_BUFFER | FC_COUNT | BC_COUNT |
+------------+---------+----------+-------------+---------+---------+--
----------+--------------+----------+----------+
```

```
| Processor0 |    1591 |    13819 |     1105376 |       0 |       0
|        4096 |         4096 |        2 |        1 |
| Processor1 |  863060 |  3684375 |     1257779 |       0 |       0
|        4091 |         4096 |        2 |        2 |
+------------+---------+----------+-------------+---------+---------+--
----------+-------------+----------+----------+
# NAME : 处理器名称
# NET_IN | NET_OUT : processor已处理的读入和写出的数据量
# REACT_COUNT : processor处理的网络时间个数
# R_QUEUE | W_QUEUE : 注册队列与写队列的大小
# FREE_BUFFER | TOTAL_BUFFER : 空闲buffer的数量与buffer总数
# FC_COUNT | BC_COUNT : 前端连接与后端连接的个数
```

- show @@threadpool ------ Cobar内部线程池状态查询

```
mysql>show @@threadpool;
+-----------------+----------+-------------+-----------------+-------
---------+------------+
| NAME            | POOL_SIZE | ACTIVE_COUNT | TASK_QUEUE_SIZE |
COMPLETED_TASK | TOTAL_TASK |
+-----------------+----------+-------------+-----------------+-------
---------+------------+
| InitExecutor    |         8 |            0 |               0
|             47 |         47 |
| TimerExecutor   |         8 |            0 |               0
|         426707 |     426707 |
| ManagerExecutor |         8 |            1 |               0
|            175 |        176 |
| Processor0-H    |         8 |            0 |               0
|             24 |         24 |
| Processor0-E    |         8 |            0 |               0
|             94 |         94 |
| Processor1-H    |         8 |            0 |               0
|         137529 |     137529 |
| Processor1-E    |         8 |            0 |               0
|          15784 |      15784 |
+-----------------+----------+-------------+-----------------+-------
---------+------------+
# NAME : 线程池名称
# POOL_SIZE : 线程池大小
# ACTIVE_COUNT : 活动线程数
# TASK_QUEUE_SIZE | COMPLETED_TASK | TOTAL_TASK: 任务队列大小，已完成任
务数及任务总数
```

- show @@parser
- show @@router

- 连接状况查询
  - show @@connection ------ cobar前端连接状态查询

```
mysql>show @@connection;
+------------+------+---------------+-------+-----------+--------+----
-----+---------+---------+---------------+---------------+-----------
-+------------+----------+
| PROCESSOR  | ID   | HOST          | PORT  | LOCAL_PORT | SCHEMA |
CHARSET | NET_IN  | NET_OUT | ALIVE_TIME(S) | WRITE_ATTEMPTS |
RECV_BUFFER | SEND_QUEUE | CHANNELS |
+------------+------+---------------+-------+-----------+--------+----
-----+---------+---------+---------------+---------------+-----------
-+------------+----------+
```

```
| Processor0 |    2 | 192.168.0.1   | 45646 |       8066 | NULL    |
utf8    | 1131370 | 7228028 |       1131953 |              0 |
4096 |          0 |        0 |
| Processor1 |    3 | 192.168.0.2   | 37744 |       8066 | NULL    |
utf8    | 1131340 | 7227842 |       1131952 |              0 |
4096 |          0 |        0 |
| Processor0 |  679 | 192.168.0.3   | 52201 |       9066 | NULL    |
utf8    |     179 |    1386 |         12955 |              0 |
4096 |          0 |     NULL |
| Processor1 |    4 | 10.20.153.17  | 54892 |       8066 | NULL    |
utf8    | 1131695 | 7230123 |       1131952 |              0 |
4096 |          0 |        0 |
+------------+------+---------------+-------+------------+--------+----
-----+---------+----------+---------------+---------------+------------
-+------------+----------+
# PROCESSOR :  处理器名称
# ID : 连接的id
# HOST | PORT| LOCAL_PORT | SCHEMA | CHARSET: 连接的信息，前端连接是指
其他客户端连接Cobar的连接，因此LOCAL_PORT为8066或9066
# NET_IN  | NET_OUT : 该前端连接的读入写出数据量
# ALIVE_TIME(S) : 连接的存活时间
# WRITE_ATTEMPTS  : 尝试重写的次数
# RECV_BUFFER | SEND_QUEUE : 读缓存的大小和写队列的长度
```

- show @@backend ------ 后端连接状态查询

```
mysql>show @@backend;
+------------+------+---------------+------+--------+---------+--------
-+--------+--------+------+-------+----------+-------+--------+
| processor  | id   | host          | port | l_port | net_in  | net_out
| life   | closed | auth | quit  | checking | stop  | status |
+------------+------+---------------+------+--------+---------+--------
-+--------+--------+------+-------+----------+-------+--------+
| Processor2 |   10 | 10.20.153.179 | 8066 |  39376 | 3645758 |  444670
| 444890 | false  | true | false | false    | false |      2 |
| Processor3 |   11 | 10.20.153.182 | 8066 |  54781 |  978188 |  444670
| 444890 | false  | true | false | false    | false |      1 |
| Processor4 |   12 | 10.20.153.17  | 8066 |  44619 |  978188 |  444670
| 444890 | false  | true | false | false    | false |      1 |
+------------+------+---------------+------+--------+---------+--------
-+--------+--------+------+-------+----------+-------+--------+
# processor|id|host|port|l_port|net_in|net_out|life: 后端连接只Cobar内
部主动创建的连接，如Cobar至MySQL的连接，Cobar心跳连接等。参数意义可参考
前端连接
# closed :
# auth :
# quit :
# checking :
# stop :
# status :
```

- show @@connection.sql

- 心跳状态查询
  - show @@heartbeat ------ 心跳状态查询，心跳包括Cobar之间的心跳，Cobar到MySQL数据库的心跳

```
mysql>show @@heartbeat;
+-------------+-------+---------------+------+---------+-------+-------
-+---------+--------------+---------------------+-------+
| NAME        | TYPE  | HOST          | PORT | RS_CODE | RETRY | STATUS
```

```
| TIMEOUT | EXECUTE_TIME | LAST_ACTIVE_TIME     | STOP  |
+-------------+-------+---------------+------+---------+-------+-------
-+---------+-------------+---------------------+-------+
| cobar1      | COBAR | 192.168.0.1   | 8066 |       2 |     0 | idle
|   10000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| cobar2      | COBAR | 192.168.0.2   | 8066 |       1 |     0 | idle
|   10000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| cobar3      | COBAR | 192.168.0.3   | 8066 |       1 |     0 | idle
|   10000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| dn_shard[0] | MYSQL | 192.168.0.4   | 3306 |       1 |     0 | idle
|   30000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| dn_shard[1] | MYSQL | 192.168.0.4   | 3306 |       1 |     0 | idle
|   30000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| dn_shard[2] | MYSQL | 192.168.0.4   | 3306 |       1 |     0 | idle
|   30000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| dn_shard[3] | MYSQL | 192.168.0.4   | 3306 |       1 |     0 | idle
|   30000 | 0,0,0        | 2012-07-26 16:31:00 | false |
| dn_single   | MYSQL | 192.168.0.4   | 3306 |       1 |     0 | idle
|   30000 | 0,0,0        | 2012-07-26 16:31:00 | false |
+-------------+-------+---------------+------+---------+-------+-------
-+---------+-------------+---------------------+-------+
# NAME：节点的名称，如果是Cobar心跳，则为Cobar节点的名称，如果是MySQL心
跳，则为datanode的名称。
# TYPE：心跳类型
# HOST | PORT ：心跳连接的host与port
# RS_CODE：心跳结果，0为初始状态，1为OK，-1为ERROR，2为OFF， -2为
TIMEOUT
# RETRY：当心跳发生错误时，已经重试的心跳次数
# STATUS：如果当前连接正在心跳，则status为checking，否则为idle
# TIMEOUT：系统设定的心跳超时时间
# EXECUTE_TIME：最近1分钟，10分钟，30分钟心跳时间的平均值
# LAST_ACTIVE_TIME：上次心跳结束时间
# STOP：心跳是否被停止
```

- **Server**信息查询
  - show @@server ------ 查询server信息

```
mysql>show @@server;
+-------------+-------------+--------------+-----------+--------------
-+---------------+---------+--------+
| UPTIME      | USED_MEMORY | TOTAL_MEMORY | MAX_MEMORY | RELOAD_TIME
| ROLLBACK_TIME | CHARSET | STATUS |
+-------------+-------------+--------------+-----------+--------------
-+---------------+---------+--------+
| 18d 0h 32m  |   212161240 |   1046937600 | 1046937600 | 1342769220441
|            -1 | utf8    | OFF    |
+-------------+-------------+--------------+-----------+--------------
-+---------------+---------+--------+
# UPTIME ：  Server启动时间
# USED_MEMORY | TOTAL_MEMORY |MAX_MEMORY ： Server所占用的内存
# RELOAD_TIME ：  Server最近一次重载时间，初始值为-1
# ROLLBACK_TIME ： Server最近一次回滚时间，初始值为-1
# CHARSET ： Server所使用的字符集
# STATUS ： Server状态
```

  - show @@time.current ------ 当前时间
  - show @@time.startup ------ 启动时间
  - show @@version ------ 版本

4.3 管理命令

- 配置重载与回滚
  - reload @@config ------ 配置文件热加载，用户配置文件不需要重启cobar，只需运行reload命令即可。
    注意：当用户如果修改server.xml中<system>标签下的配置时，需要重启cobar才能生效，其余配置均可热加载。
  - rollback @@config ------ 配置文件回滚
    每次重载，Cobar会将旧的配置备份，并载入新的配置，如果重载后发现配置有误，可以运行rollback回滚到重载之前的配置。Cobar只备份最近一次重载之前的配置，因此rollback只能回归到最近一次重载之前的配置，更早版本的配置不可通过多次rollback找回。如果两次rollback之间没有reload操作，第二次rollback将不成功。
  - reload @@route
  - rollback @@route
  - reload @@user
  - rollback @@user

- 节点切换
  - switch @@datasource name:index ------ 切换数据节点对应的datasource，用于主、备之间的切换

```
# Cobar中的datanode运行过程中都要对应一个具体的datasource，datanode下的所有datasource在Cobar中都有编号，0代表主，1代表备，2代表第二备，以此类推，利用show @@datanode可以查看一个datanode对应的所有datasource以及当前其所连接的datasource
# switch命令可以切换一个datanode当前连接的datasource，以实现主备切换，切换后的结果可利用show @@datanode查看。

# 使用方式一：直接指定要切换的目的index,示例中，将dn_single直接切换到1号数据源上。
mysql>switch @@datasource dn_single:1;

# 使用方式二：不指定index，cobar将循环轮询切换，即如果当前为0则切换到1（没有1，保持0不变），如果当前为1则切换到2(没有2的话，切回到0)；
mysql>switch @@datasource dn_single;

# 使用方式三：支持同时切换多个节点；示例中，将dn_single和dn_shard[0]两个数据节点都切换到1号数据源
mysql>switch @@datasource dn_single, dn_shard[0]:1;
# 同理，切换多个节点的数据源也可以不指定index，由cobar的轮询策略来决定切换到哪个数据源
mysql>switch @@datasource dn_single, dn_shard[0];
```

- 心跳
  - stop @@heartbeat name:time ------ 暂停数据节点心跳，主要用于要暂时禁用某节点HA功能的场合

```
# 暂停dn_single节点的心跳100秒，100秒后，心跳自动恢复。通过show @@datanode命令可以查看被暂停的心跳多久以后可以自动恢复。
mysql>stop @@heartbeat dn_single:100;

# 通过将暂停心跳的时间设置为-1，立即恢复暂停的心跳
mysql>stop @@heartbeat dn_single:-1;
```

- 连接断开
  - kill @@connection id1,id2,... ------ 手工断开Cobar的前端连接

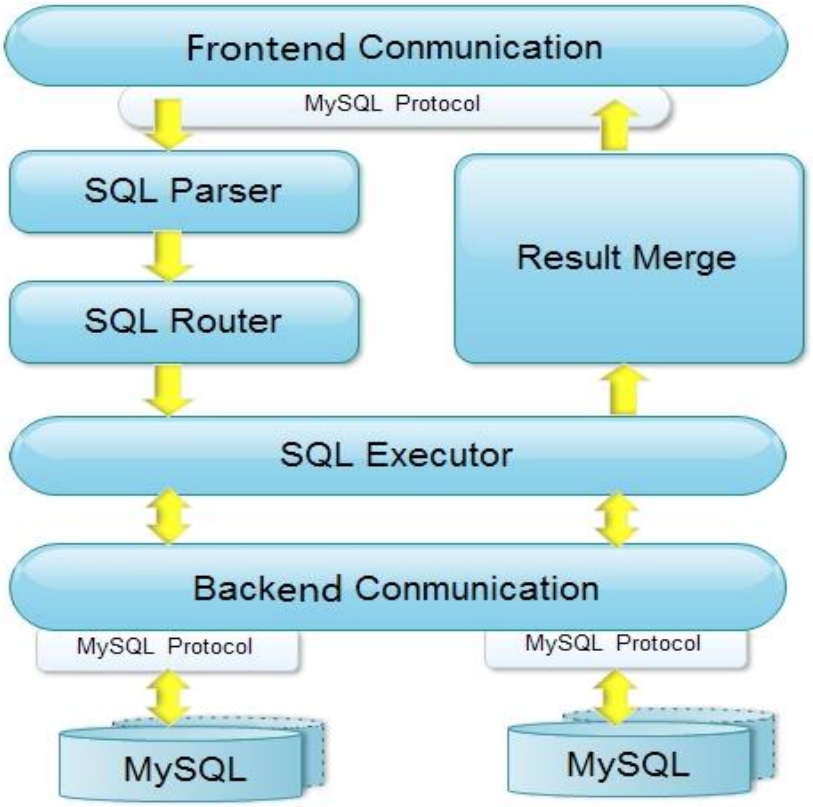
```
# show @@connection命令会列出Cobar所有的前端连接，利用kill @@connection
可以根据connection id关闭前端连接。示例中关闭了id为1,2的两个前端连接
mysql>kill @@connection 1, 2;

# kill @@connection不能关闭后端连接（后端连接利用show @@backend查看）
```

- 状态转换
  - offline
  - online

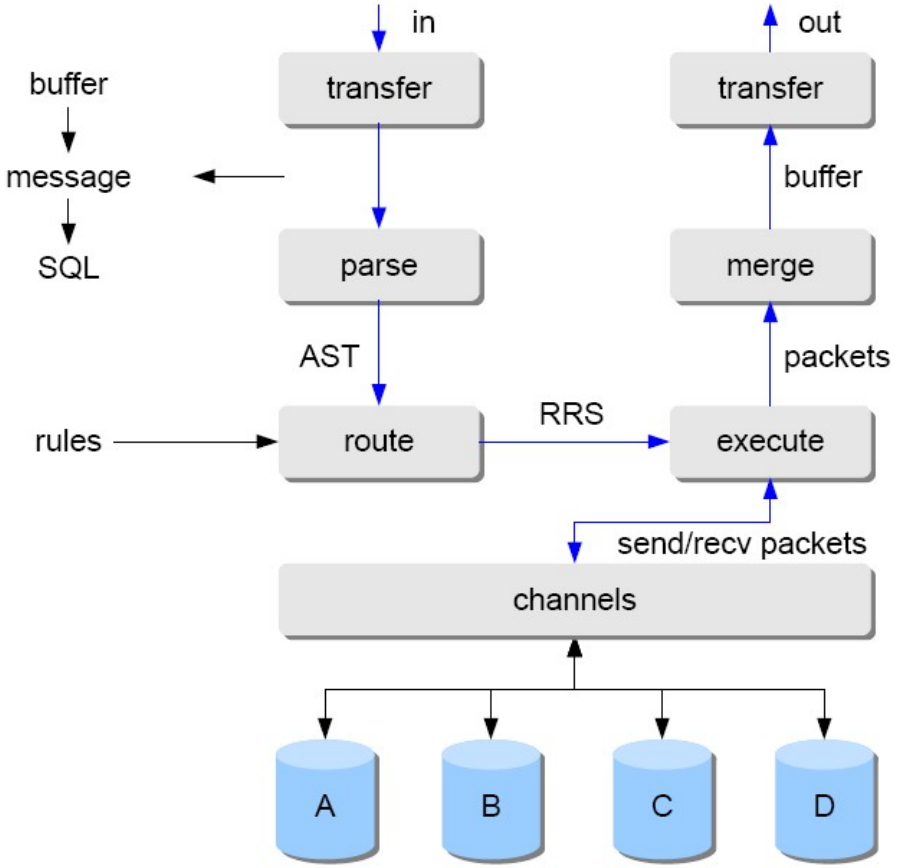
5.架构文档

5.1 系统架构

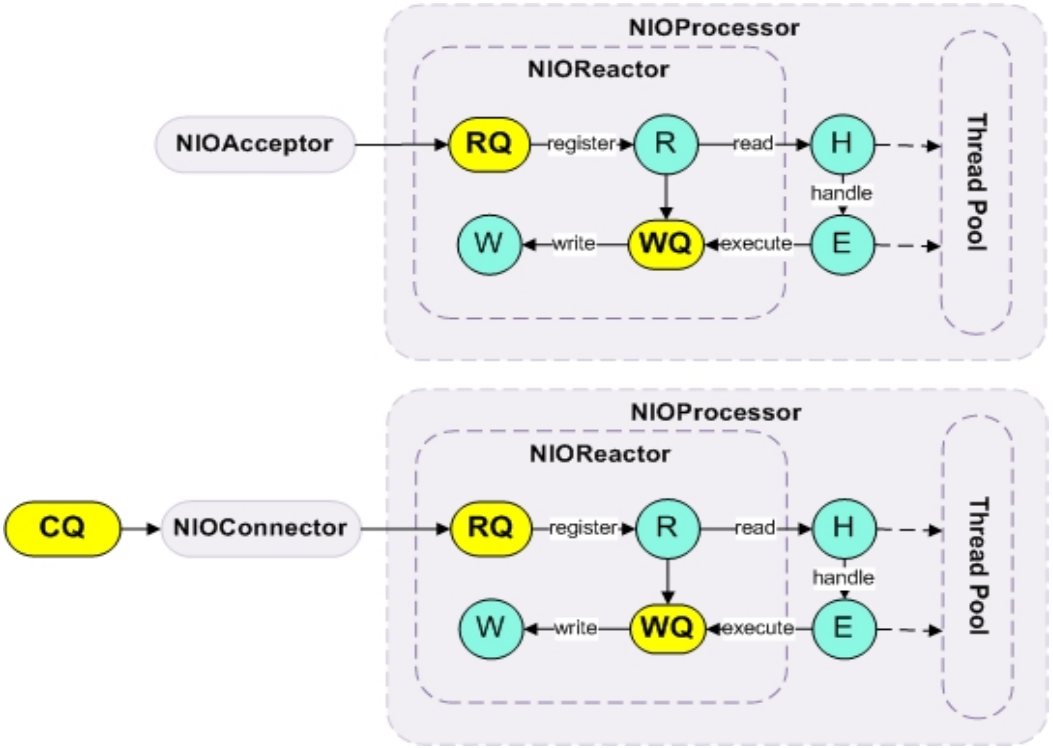
-
  - 系统模块架构图



  - 各个模块数据流图



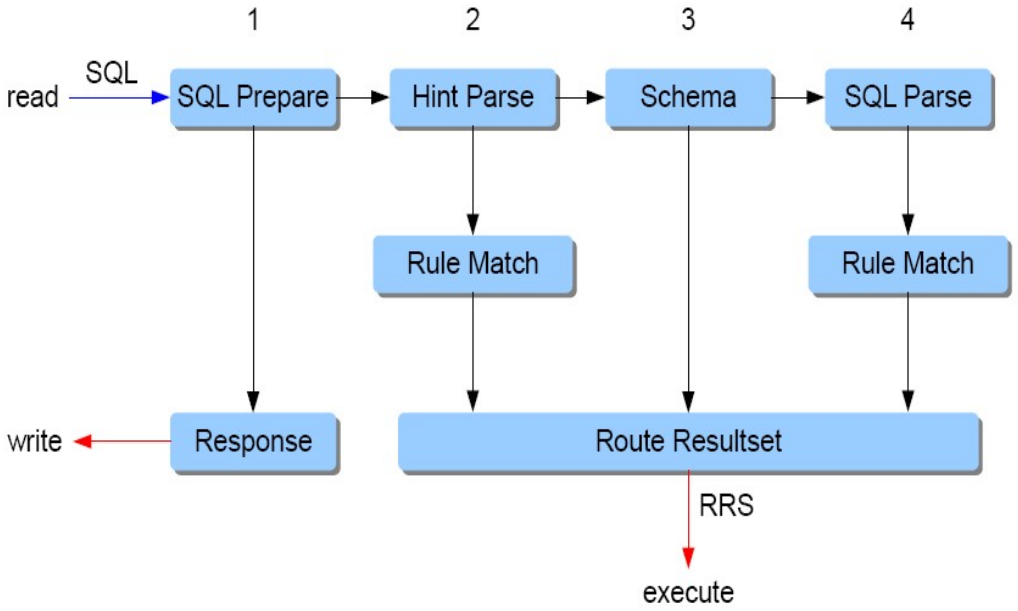
5.2 网络通讯模块



5.3 协议处理模块

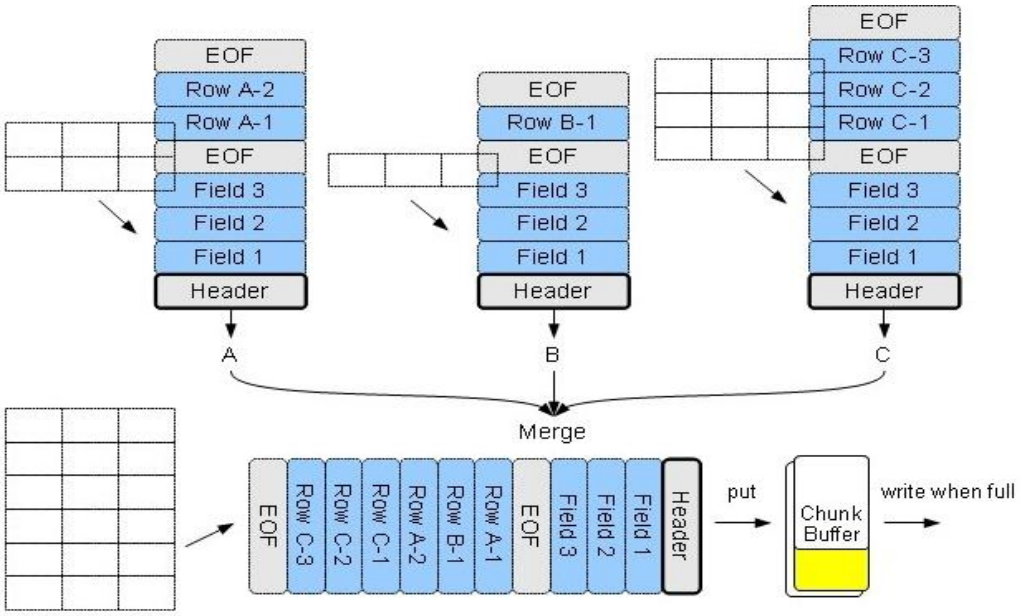
5.4 SQL解析模块

5.5 路由与数据分布模块



5.6 执行与优化模块

5.7 结果合并与返回模块



5.8 心跳模块

6.测试报告

- 6.1 性能测试报告

7.外部资源

PPT中主要介绍Cobar的原理及应用，设计范围较广，推荐大家下载阅读。注意，PPT中提到的部分内容并未在本版实现，未实现部分参照第一章产品约束。

(推荐)Cobar原理及应用.ppt

Labels: None

4 Child Pages

34 Comments

Anonymous 六月 19, 2012

请博主快补充应用场景部分，谢谢

郭 晓文 六月 19, 2012

http://code.alibabatech.com/wiki/display/cobar/rule 已经给出了一个典型的应用场景，可以先参考

Anonymous 七月 17, 2012

已经在实际项目中采用了，请问下，是否支持读写分离呢？

贺 贤懋 七月 17, 2012

读写分离目前版本不支持，以前的版本是支持的。
一个是公司内部不需要这样的需求，另外考虑到读写分离后的同步延迟不可控，所以就去掉了。
我们后续会考虑一下，这个作为一个需求，在现有基础上实现起来还是挺容易的。

Anonymous 八月 07, 2012

请问下那个版本支持读写分离呢?

贺 贤懋 八月 07, 2012

1.0.x的版本，后面我们会把这个特性加到新版本中。

Anonymous 八月 10, 2012

能将集群的应用场景给个参考文档吗?

Anonymous 八月 14, 2012

执行聚合函数，或group by语句的查询不能产生正确的结果。
如：我做了两个shard， 这行select count(*) from table 返回两行。 group by 的查询结果也不对。

它只是把SQL在每个shard上执行，然后union到一起了。
这个如何解决？

贺贤懋 八月 14, 2012

目前跨库的group by，order by ，聚合函数等都不支持，类似count,sum等需要应用在本地处理。

Anonymous 八月 21, 2012

怎么从svn上checkout源码下来读啊

贺贤懋 八月 21, 2012

比如你要checkout cobar1.2.6的源代码：svn co http://code.alibabatech.com/svn/cobar/server/tags/1.2.6/ cobar-1.2.6

Anonymous 八月 27, 2012

多地址的dataSource 配置在哪里指定 机器名称？root@机器名称

贺贤懋 八月 27, 2012

具体指什么？
dataSource的配置，可以参考schema.xml里dataSource模块的配置。
进一步交流可以加入我们的QQ群：250345828

Anonymous 八月 27, 2012

Cobar与GreenPlum的区别在什么地方？

Anonymous 八月 27, 2012

我在schema.xml中配置来多个机器的ip地址作为数据源，启动时报 Access denied for user 'root'@'datanode1' (using password: NO)错误，其中，datanode1是我机器的ip，报错是因为配置的另一台机器名称不是datanode1
这种情况怎么解决，谢谢！

Anonymous 十一月 19, 2012

从你这个报错来看，其实是你的远端数据库没有配置远程连接许可，不是说机器名称错误。。。

Anonymous 八月 27, 2012

OceanBase的设计与Cobar有什么联系吗？

Anonymous

十月 29, 2012

我也想问下OceanBase的设计与Cobar有什么联系吗？
另外能给出上述两者在淘宝的具体应用场景么？
谢谢

Anonymous 十月 30, 2012

1、Cobar看起来跟Amoeba项目很相像，只是后者文档不是很完善，amoeba支持读写分离，请问cobar支持么？

2、还有就是水平切分表，比如将user表分为normal_user和enterprise_user两个表，但是还在同一个物理库中，
这种情况分库规则如何写？

Anonymous 十一月 14, 2012

2.3 Cobar使用注意事项
1.请注意表的拆分方式，一张表水平拆分多份到不同的库中，而不是放入同一个库中。

Anonymous 十一月 18, 2012

感谢分享 学习到很多

Anonymous 十一月 26, 2012

您好，我在第一次使用corba的时候，会报这个错误，但是后面的访问就正常了，只会在第一次的时候报time_out
17:50:47,419 WARN [thread=Processor0-
E0,class=ServerConnection,host=10.1.1.92,port=3658,schema=facebook_onlinegame_db]dn_shar
mysql-connector-java-5.1.21 (Revision: $

Unknown macro: {bzr.revision-id}

) */SHOW VARIABLES WHERE Variable_name ='language' OR Variable_name = 'net_write_timeout' OR Variable_name = 'interactive_timeout' OR Variable_name = 'wait_timeout' OR Variable_name = 'character_set_client' OR Variable_name = 'character_set_connection' OR Variable_name = 'character_set' OR Variable_name = 'character_set_server' OR Variable_name = 'tx_isolation' OR Variable_name = 'transaction_isolation' OR Variable_name = 'character_set_results' OR Variable_name = 'timezone' OR Variable_name = 'time_zone' OR Variable_name = 'system_time_zone' OR Variable_name = 'lower_case_table_names' OR Variable_name = 'max_allowed_packet' OR Variable_name = 'net_buffer_length' OR Variable_name = 'sql_mode' OR Variable_name = 'query_cache_type' OR Variable_name = 'query_cache_size' OR Variable_name = 'init_connect'}
java.util.concurrent.TimeoutException
at java.util.concurrent.FutureTask$Sync.innerGet(FutureTask.java:228)
at java.util.concurrent.FutureTask.get(FutureTask.java:91)
at com.alibaba.cobar.server.node.MySQLChannel.connect(MySQLChannel.java:185)
at com.alibaba.cobar.server.node.MySQLDataSource.getChannel(MySQLDataSource.java:156)
at com.alibaba.cobar.server.node.MySQLDataNode.getChannel(MySQLDataNode.java:115)
at com.alibaba.cobar.server.node.MySQLDataNode.getChannel(MySQLDataNode.java:107)
at com.alibaba.cobar.server.executor.SingleNodeExecutor$2.run(SingleNodeExecutor.java:142)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at java.lang.Thread.run(Thread.java:662)

17:50:48,427 INFO [thread=Processor1-
H1,class=ServerConnection,host=10.1.1.92,port=3661,schema=facebook_onlinegame_db]'test'
login success
17:50:52,714 WARN [thread=Processor1-
E14,class=ServerConnection,host=10.1.1.92,port=3657,schema=facebook_onlinegame_db]global_

Unknown macro: {UPDATE `sequence` SET `id` = LAST_INSERT_ID(`id` + 1) WHERE `name` =
'players'}

java.util.concurrent.TimeoutException
at java.util.concurrent.FutureTask$Sync.innerGet(FutureTask.java:228)
at java.util.concurrent.FutureTask.get(FutureTask.java:91)
at com.alibaba.cobar.server.node.MySQLChannel.connect(MySQLChannel.java:185)
at com.alibaba.cobar.server.node.MySQLDataSource.getChannel(MySQLDataSource.java:156)
at com.alibaba.cobar.server.node.MySQLDataNode.getChannel(MySQLDataNode.java:115)
at com.alibaba.cobar.server.node.MySQLDataNode.getChannel(MySQLDataNode.java:107)
at com.alibaba.cobar.server.executor.SingleNodeExecutor$2.run(SingleNodeExecutor.java:142)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at java.lang.Thread.run(Thread.java:662)
17:51:11,763 WARN [thread=Processor1-
E6,class=ServerConnection,host=10.1.1.92,port=3657,schema=facebook_onlinegame_db]dn_shar

Unknown macro: {select playerId, facebookId, userName, firstName, sex, accountType, currency, rewardAmount, membershipPoints, level, charm, email, thirdPathId, unsubscribeLike, unsubscribeEmail, isActive, isLiked, language, registerDate, birthday, lastLoginDate, rewardTime, sessionKey from player where playerId = null}

java.util.concurrent.TimeoutException
at java.util.concurrent.FutureTask$Sync.innerGet(FutureTask.java:228)
at java.util.concurrent.FutureTask.get(FutureTask.java:91)
at com.alibaba.cobar.server.node.MySQLChannel.connect(MySQLChannel.java:185)
at com.alibaba.cobar.server.node.MySQLDataSource.getChannel(MySQLDataSource.java:156)
at com.alibaba.cobar.server.node.MySQLDataNode.getChannel(MySQLDataNode.java:115)
at com.alibaba.cobar.server.node.MySQLDataNode.getChannel(MySQLDataNode.java:107)
at com.alibaba.cobar.server.executor.MultiNodeExecutor$3.run(MultiNodeExecutor.java:206)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at java.lang.Thread.run(Thread.java:662)

Anonymous 十一月 29, 2012

如何支持垂直切分， 在tableRule中如何设置？

Anonymous 十二月 12, 2012

•分库情况下，delete语句必须包含拆分字段列名。对不???

Anonymous 十二月 12, 2012

能否写个cobar-driver使用的简单说明？

Anonymous

二月 20, 2013

支持读写分离的新版本预计什么时候能推出？我在考虑能不能等的及,谢谢

Anonymous 二月 27, 2013

你好！我想问一下这个能否实现对于用户来说，下面是什么数据库都无所谓！但是所有操作都一样，语句也一样！ 比如说我现在用的是mysql，但是有一天我想用oracle了，我用户层面应用程序无需改动什么，也就是说对于用户来说，如果想要改变数据库，只要改变配置cobar的配置文件就可以！而我们用户层面的应用程序无需改动！

Anonymous 二月 27, 2013

你好！我想问一下这个有没有对外提供的接口！这个接口不用区分底层的数据库属于什么类型

Anonymous 三月 28, 2013

您好，想问一下这个cobar支持的最大连接是多少呢？ 怎么感觉连接在300个左右的时候就会出现查询出错呢？

Anonymous 四月 18, 2013

你好，在Rule配置里，看规则说必须保证count*length=1024。在实际应用中，假设以user_id为分区的，每个user_id可能很大，如1780001，这如何配置

Anonymous 五月 10, 2013

Corba后端有连接数据库限制或者连接数限制吗？

Anonymous 六月 26, 2013

您好，我想问下， cobar有类似 start transaction; insert into tb1; insert into tb2; commit;这样的事务支持么？ 我连cobar执行start transaction返回"unsupported statement"； 所说的支持事务是不是指的是update tb2 where id>5；这样的单条语句跨库的情况？

Anonymous 十一月 12, 2013

不知道现在的版本支持除了 MYSQL以外的其他数据库么？
以后会考虑增加对其他数据库的支持么？

Anonymous 十一月 14, 2013

cobar在主备数据源切换时，可能会出现原本在单库完成的一组sql语句，变成在多个库中执行，提交！如果主备数据库同步有问题的话，很容易产生数据不一致的情况！

Add Comment

友情链接:Taocode