[wangdha@lenovo.com](mailto:wangdha@lenovo.com)

Mycat本身是一个数据库中间件，实现了数据库代理、数据库引擎功能；前端模拟MySQL协议与客户端交互，经过SQL解析器与后端通过MySQL协议、JDBC与各种关系型数据库交互；与其他中间见不同的是，MyCAT后端兼容多个类型的数据库，且能实现跨库条件下大部分数据库函数。

Mycat: 数据库中间件技术，前端使用中间件后端使用数据库(mysql oracle)支持，可以对数据库集群有效的管理方便扩容，对大型数据表横纵分片，分片后的表连接查询，对前端开发处于透明状态。

|  |
| --- |
| mycat对APP来说，可以理解成一个数据库，他实际上是一个数据库中间件(目前只支持mysql),提供  集群支持 |

Mycat和MySQL的区别：  
可以把上层看作是对下层的抽象，例如操作系统是对各类计算机硬件的抽象。那么我们什么时候需要抽象？假如只有一种硬件的时候，我们需要开发一个操作系统吗？  
再比如一个项目只需要一个人完成的时候不需要leader，但是当需要几十人完成时，就应该有一个管理者，发挥沟通协调等作用，而这个管理者对于他的上层来说就是对项目组的抽象。  
同样的，当我们的应用只需要一台数据库服务器的时候我们并不需要Mycat，而如果你需要分库甚至分表，这时候应用要面对很多个数据库的时候，这个时候就需要对数据库层做一个抽象，来管理这些数据库，而最上面的应用只需要面对一个数据库层的抽象或者说数据库中间件就好了，这就是Mycat的核心作用。  
所以可以这样理解：数据库是对底层存储文件的抽象，而Mycat是对数据库的抽象。

MySQL分布式集群搭建的一个东东，MyCAT

目前通过配置文件的方式来定义逻辑库和相关配置：

·         MYCAT\_HOME/conf/schema.xml中定义逻辑库，表、分片节点等内容；

·         MYCAT\_HOME/conf/rule.xml中定义分片规则；

·         MYCAT\_HOME/conf/server.xml中定义用户以及系统相关变量，如端口等。

### 水平切分

现在互联网非常火爆的web 2.0类型的网站，基本上大部分数据都能够通过会员用户信息关联上，可能很多核心表都非常适合通过会员ID来进行数据的水平切分。而像论坛社区讨论系统，就更容易切分了，非常容易按照论坛编号来进行数据的水平切分。切分之后基本上不会出现各个库之间的交互。

Hadoop（分布式计算）

Datanode是文件系统的工作节点，他们根据客户端或者是namenode的调度存储和检索数据，并且定期向namenode发送他们所存储的块(block)的列表。

集群中的每个服务器都运行一个DataNode后台程序，这个后台程序负责把HDFS数据块读写到本地的文件系统。当需要通过客户端读/写某个 数据时，先由NameNode告诉客户端去哪个DataNode进行具体的读/写操作，然后，客户端直接与这个DataNode服务器上的后台程序进行通 信，并且对相关的数据块进行读/写操作

简单的MyCAT搭建大致上就包括这些内容，现在讲讲使用一段时间以后，对MyCAT的一些总结；  
1.MyCAT的性能表现还是不错的，这几天一直对MyCAT的各方面进行测试，发现MyCAT作为一个代理，虽然是在JAVA虚拟机上面运行，但是面对接近9K的QPS的峰值的时候，本身并没有出现无响应或者丢失连接的问题；  
2.MyCAT对前端显示的所有的库，表，全部由schema来配置，但是本身不定义表结构，这使得后端的表结构如果出现不一致，MyCAT前端是察觉不到的，不太方便吧；  
3.第二点的不方便，也反映了一点，没有配置到schema的表，完全无法通过MyCAT去操作，这也算是安全性良好的一个表现吧；  
4.之前说SQL Executor没感觉到，也是因为在一些测试中，发现MyCAT更像一个提供转发和结果合并功能的代理，只是对SQL和结果进行了process，不过这个需要去看源代码才知晓细节了。

### 数据切分

垂直切分和水平切分

垂直切分：

按照不同的表切分到不同的数据库上，这种切分可以称之为数据的垂直切分。

水平切分：根据表中的数据逻辑关系，将同一个表中的数据按照某种条件拆分到多台数据库（主机）上面，这种切分称之为数据库的水平切分

Mycat配置：

## Schema.xml

## Server.xml

## Rule.xml

# Schema.xml

Schema.xml管理着mycat的逻辑库，表，分片规则，dataNode和DataSource

Schema.xml标签

<schema name=”” checkSQLschema=”false” sqlMaxlimit=”100” >

<table name=”” dataNode=”dn1,dn2,dn3” rule=”auto-sharding-long”></table>

</schema>

Schema标签用来定义mycat实例中的逻辑库，一个mycat实例可以有多个逻辑库

如果不配置schema标签，所有表的配置都属于同一个逻辑库

在system.xml的user标签中配置访问的schema，多个用“,”隔开

## Schema标签的相关属性：

## DataNode（String） checkSQLschema（boolean） sqlMaxlimit（Integer）

## DataNode

该属性用于绑定逻辑库到某个固定的database上，如果定义了这个属性，那么这个逻辑库就不能工作在分库分表模式下，这个逻辑库的所有操作会直接作用在绑定的 DataNode上，这个schema就可以用作读写分离和主从切换

<schema name=”USERDB” checkSQLschema=”false” sqlMaxlimit=”100” dataNode=”db1”>

<!-- 这里不能配置任何信息 -->

</schema>

那么userdb现在就绑定到了db1所配置的具体database上，可以直接访问这个database，当然属性只能配置绑定到一个database上，不能绑定到多个dn上

## checkSQLschema

当该值设置为true时，当我们执行：

Select \* from USERDB.travelrecord ;

Mycat会把语句修改为：

Select \* from travelrecord;

即把表示schema的字符去掉，避免\*\*（ERROR 1146 (42S02): Table

‘testdb.travelrecord’ doesn’t exist）报错

不过，即使设置语句为true,如果语句所带的并非表示schema的字符，例如：

\*\*select \* from db1.travelrecord;\*\*

那么，mycat并不会删除db1这个字段，如果没有定义该库的话会报错，所以在提供sql语句的最好不要带这个字段

## sqlMaxLimit

当该值设置为某个数值时，每条执行的sql语句，如果没有加上limit语句，mycat会自动的加上所对应的值，例如设置值为100，执行：

\*\*select \* from TESTDB.travelrecord;\*\*

和 执行

\*\*select \* from TESTDB.travelrecord limit 100;\*\*

效果相同

不设置该值时，mycat会默认把查询到的信息都显示出来，造成过多的输出。

建议加，用于减少数据返回

当然sql语句也显式的制定limit大小，不收该属性的约束

需要注意：如果运行的schema为非拆分库，那么该属性不会生效。需要手动添加limit语句

Table标签：

<table name="travelrecord" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" ></table>

Table标签定义了mycat中的逻辑表，所有拆分的表都需要在这个标签中定义

Name(String) 必须唯一

DataNode定义这个逻辑表所属的DataNode，该属性值需要和dataNode中的name属性相对应，如果定义的dn过多可以使用如下减法减少配置：

<table name="travelrecord" dataNode="multipleDn$0-99,multipleDn2$100-199" rule="auto-sharding-long" ></table>

<dataNode name="multipleDn" dataHost="localhost1" database="db$0-99" ></dataNode>

<dataNode name="multipleDn2" dataHost="localhost1" database=" db$0-99" ></dataNode>

这里需要注意的是database属性所指定的真实database name需要在后面添加一个，例如上面的例子中，我需要在真实的mysql

上建立名称为dbs0到dbs99的database。

Rule属性

该属性需要指定逻辑表中使用的规则名字，规则名字在rule.xml中定义，必须与tableRule中的标签中的name 属性属性值一一对应

ruleRequired属性

该属性用于指定是否绑定分片规则，如果配置为TRUE，但没有具体配置TRUE,会报错

primaryKey属性

该逻辑表对应真实表的主键，例如：分片的规则是使用非主键进行分片的，那么在使用主键查询的时候，就会发送查询语句到所

有配置的DN上，如果使用该属性配置真实表的主键。那么MyCat会缓存主键与具体DN的信息，那么再次使用非主键进行查询的

时候就不会进行广播式的查询，就会直接发送语句给具体的DN，但是尽管配置该属性，如果缓存并没有命中的话，还是会发送语

句给具体的DN，来获得数据

Type属性：

该属性定义了逻辑表中的类型，分为全局表和普通表两种

全局表：global

普通表：不指定为global

autoIncrement属性

mysql对非自增长主键，使用last\_insert\_id()是不会返回结果的，只会返回0。所以，只有定义了自增长主键的表才可以用

last\_insert\_id()返回主键值。

mycat目前提供了自增长主键功能，但是如果对应的mysql节点上数据表，没有定义auto\_increment，那么在mycat层调用

last\_insert\_id()也是不会返回结果的。

由于insert操作的时候没有带入分片键，mycat会先取下这个表对应的全局序列，然后赋值给分片键。这样才能正常的插入到数据

库中，最后使用last\_insert\_id()才会返回插入的分片键值。

如果要使用这个功能最好配合使用数据库模式的全局序列。

使用autoIncrement=“true” 指定这个表有使用自增长主键，这样mycat才会不抛出分片键找不到的异常。

使用autoIncrement=“false” 来禁用这个功能，当然你也可以直接删除掉这个属性。默认就是禁用的。

# mycat怎样在分库分表的情况下保证主键的全局唯一方法，接下来就来分析三种方法各自的优缺点。

http://blog.csdn.net/WuLex/article/details/52495256

needAddLimit属性

指定表是否需要自动的在每个语句后面加上limit限制。由于使用了分库分表，数据量有时会特别巨大。这时候执行查询语句，如

果恰巧又忘记了加上数量限制的话。那么查询所有的数据出来，也够等上一小会儿的。

所以，mycat就自动的为我们加上LIMIT 100。当然，如果语句中有limit，就不会在次添加了。

这个属性默认为true,你也可以设置成false`禁用掉默认行为。

childTable标签

childTable标签用于定义E-R分片的子表。通过标签上的属性与父表进行关联。

### Name

### joinKey

### ParentKey

### PrimaryKey

### needAndLimit

name属性

定义子表的表名。

joinKey属性

插入子表的时候会使用这个列的值查找父表存储的数据节点。

parentKey属性

属性指定的值一般为与父表建立关联关系的列名。程序首先获取joinkey的值，再通过\*\*parentKey\*\*属性指定的列名产生查询语

句，通过执行该语句得到父表存储在哪个分片上。从而确定子表存储的位置。

primaryKey属性

同table标签所描述的。

needAddLimit属性

同table标签所描述的。

dataNode标签

<dataNode name="dn1" dataHost="lch3307" database="db1" ></dataNode>

dataNode 标签定义了MyCat中的数据节点，也就是我们通常说所的数据分片。一个\*\*dataNode\*\* 标签就是一个独立的数据分

片。

例子中所表述的意思为：使用名字为lch3307数据库实例上的db1物理数据库，这就组成一个数据分片，最后，我们使用名字dn1

标识这个分片。

dataNode标签的相关属性：

### Name

### dataHost

### dataBase

name属性

定义数据节点的名字，这个名字需要是唯一的，我们需要在table标签上应用这个名字，来建立表与分片对应的关系。

dataHost属性

该属性用于定义该分片属于哪个数据库实例的，属性值是引用dataHost标签上定义的name属性。

database属性

该属性用于定义该分片属性哪个具体数据库实例上的具体库，因为这里使用两个纬度来定义分片，就是：实例+具体的库。因为

每个库上建立的表和表结构是一样的。所以这样做就可以轻松的对表进行水平拆分。

## dataHost标签

该标签在mycat逻辑库中也是作为最底层的标签存在，直接定义了具体的数据库实例、读

写分离配置和心跳语句

dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"writeType="0" dbType="mysql" dbDriver="native">

<heartbeat>select user()</heartbeat>

<!-- can have multi write hosts -->

<writeHost host="hostM1" url="localhost:3306" user="root" password="123456">

<!-- can have multi read hosts -->

<!-- <readHost host="hostS1" url="localhost:3306" user="root" password="123456"/> -->

</writeHost>

<!-- <writeHost host="hostM2" url="localhost:3316" user="root" password="123456"/> -->

</dataHost>

### Name

### maxCon

### mixCon

### Balance

### writeType

### dbType

### dbDriver

name属性

唯一标识dataHost标签，供上层的标签使用。

maxCon属性

指定每个读写实例连接池的最大连接。也就是说，标签内嵌套的writeHost、readHost标签都会使用这个属性的值来实例化出连

接池的最大连接数。

minCon属性

指定每个读写实例连接池的最小连接，初始化连接池的大小。

balance属性

负载均衡类型，目前的取值有3种：

1. balance=“0”, 所有读操作都发送到当前可用的writeHost上。

2. balance=“1”，所有读操作都随机的发送到readHost。

3. balance=“2”，所有读操作都随机的在writeHost、readhost上分发。

writeType属性

负载均衡类型，目前的取值有3种：

1. writeType=“0”, 所有写操作都发送到可用的writeHost上。

2. writeType=“1”，所有写操作都随机的发送到readHost。

3. writeType=“2”，所有写操作都随机的在writeHost、readhost分上发.

dbType属性

指定后端连接的数据库类型，目前支持二进制的mysql协议，还有其他使用JDBC连接的数据库。例如：mongodb、oracle、

spark等

dbDriver属性

指定连接后端数据库使用的Driver，目前可选的值有native和JDBC。使用native的话，因为这个值执行的是二进制的mysql协

议，所以可以使用mysql和maridb。其他类型的数据库则需要使用JDBC驱动来支持。

如果使用JDBC的话需要将符合JDBC 4标准的驱动JAR包放到MYCAT\lib目录下，并检查驱动JAR包中包括如下目录结构的文

件：META-INF\services\java.sql.Driver。在这个文件内写上具体的Driver类名，例如：com.mysql.jdbc.Driver。

heartbeat标签

这个标签内指明用于和后端数据库进行心跳检查的语句。例如,MYSQL可以使用select user()，Oracle可以使用select 1 from

dual等。

这个标签还有一个connectionInitSql属性，主要是当使用Oracla数据库时，需要执行的初始化SQL语句就这个放到这里面来。例

如：alter session set nls\_date\_format='yyyy-mm-dd hh24:mi:ss'

writeHost标签、readHost标签

这两个标签都指定后端数据库的相关配置给mycat，用于实例化后端连接池。唯一不同的是，writeHost指定写实例、readHost

指定读实例，组着这些读写实例来满足系统的要求。

在一个dataHost内可以定义多个writeHost和readHost。但是，如果writeHost指定的后端数据库宕机，那么这个writeHost绑

定的所有readHost都将不可用。另一方面，由于这个writeHost宕机系统会自动的检测到，并切换到备用的writeHost上去。

### Host

### Url

### User

### Password

host属性

用于标识不同实例，一般writeHost我们使用\*M1，readHost我们用\*S1。

url属性

后端实例连接地址，如果是使用native的dbDriver，则一般为address:port这种形式。用JDBC或其他的dbDriver，则需要特殊

指定。当使用JDBC时则可以这么写：jdbc:mysql://localhost:3306/。

user属性

后端存储实例需要的用户名字

password属性

后端存储实例需要的密码

# Server.xml

优化配置

server.xml几乎保存了所有mycat需要的系统配置信息。其在代码内直接的映射类为SystemConfig类。

## user标签

<user name="test">

<property name="password">test</property>

<property name="schemas">TESTDB</property>

<property name="readOnly">true</property>

</user>

这个标签主要用于定义登录mycat的用户和权限。例如上面的例子中，我定义了一个用户，用户

名为test、密码也为test，可访问的schema也只有TESTDB一个。

如果我在schema.xml中定义了多个schema，那么这个用户是无法访问其他的schema。在mysql客户端看来则是无法使用use切

换到这个其他的数据库。如果使用了use命令，则mycat会报出这样的错误提示：

ERROR 1044 (HY000): Access denied for user 'test' to database 'xxx'

这个标签嵌套的property标签则是具体声明的属性值，正如上面的例子。我们可以修改user标签的name属性来指定用户名；修

改password内的文本来修改密码；修改readOnly为true 或false来限制用户是否只是可读的；修改schemas内的文本来控制用户可

放问的schema；修改schemas内的文本来控制用户可访问的schema，同时访问多个schema的话使用 , 隔开，例如:

<property name="schemas">TESTDB,db1,db2</property>

## system标签

这个标签内嵌套的所有property标签都与系统配置有关

defaultSqlParser属性

由于mycat最初是时候Foundation DB的sql解析器，而后才添加的Druid的解析器。所以这个属性用来指定默认的解析器。目前

的可用的取值有：druidparser和 fdbparser。使用的时候可以选择其中的一种，目前一般都使用druidparser。

Druid 阿里数据库连接池

Foundation DB 是新一代的数据库，用来支持多样化的应用程序和数据模型

processors属性

这个属性主要用于指定系统可用的线程数，默认值为Runtime.getRuntime().availableProcessors()方法返回的值。主要影响

processorBufferPool、processorBufferLocalPercent、processorExecutor属性。NIOProcessor的个数也是由这个属性定义的，

所以调优的时候可以适当的调高这个属性。

processorBufferChunk属性

这个属性指定每次分配Socket Direct Buffer的大小，默认是4096个字节。这个属性也影响buffer pool的长度。

processorBufferPool属性

这个属性指定bufferPool计算 比例值。由于每次执行NIO读、写操作都需要使用到buffer，系统初始化的时候会建立一定长度的

buffer池来加快读、写的效率，减少建立buffer的时间。

Mycat中有两个主要的buffer池:

- BufferPool

- ThreadLocalPool

BufferPool由ThreadLocalPool组合而成，每次从BufferPool中获取buffer都会优先获取ThreadLocalPool中的buffer，未命中之

后才会去获取BufferPool中的buffer。也就是说ThreadLocalPool是作为BufferPool的二级缓存，每个线程内部自己使用的。当

然，这其中还有一些限制条件**需要线程的名字是由$\_开头**。然而，BufferPool上的buffer则是每个NIOProcessor都共享的。

默认这个属性的值为： 默认bufferChunkSize(4096) \* processors属性 \*　1000

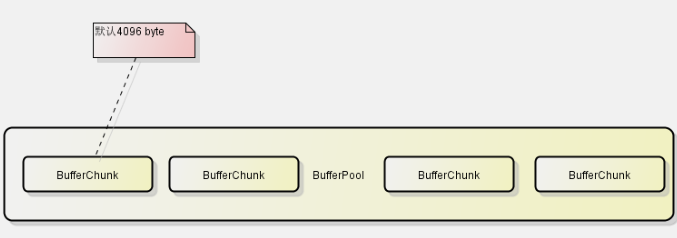
BufferPool的总长度 = bufferPool / bufferChunk。

若bufferPool不是bufferChunk的整数倍，则总长度为前面计算得出的商 + 1

假设系统线程数为4，其他都为属性的默认值，则：

bufferPool ＝　4096 \*　4 \* 1000

BufferPool的总长度 : 4000 = 16384000 / 4096



## ......

# Rule.xml

Rule.xml中定义了对表进行拆分所涉及到的规则定义。这个文件夹里主要有tableRule和function两个标签

tableRule标签

这个标签定义表规则。

定义的表规则，在schema.xml：

<tableRule name="rule1">

<rule>

<columns>id</columns>

<algorithm>func1</algorithm>

</rule>

</tableRule>

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。

标签内使用。让逻辑表使用这个规则进行分片。

function标签

<function name="hash-int"

class="org.opencloudb.route.function.PartitionByFileMap">

<property name="mapFile">partition-hash-int.txt</property>

</function>

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