# Mycat

### 第一章 Mycat开源宣言



如果我有一个32核心的服务器，我就可以实现1个亿的数据分片，我有32核心的服务器么？没有，所以我至今无法实现1个亿的数据分片。——**Mycat’s Plan**

上面这句话是Mycat 1.0快要完成时候的一段感言，而当发展到Mycat 1.3的时候，我们又有了一个新的Plan：

如果我们有10台物理机，我们就可以实现1000亿的数据分片，我们有10台物理机么？没有，所以，Mycat至今没有机会验证1000亿大数据的支撑能力——**Mycat’s Plan 2.0**

“每一个成功的男人背后都有一个女人”。自然Mycat也逃脱不了这个法则。Mycat背后是阿里曾经开源的知名产品——Cobar。Cobar的核心功能和优势是MySQL数据库分片，此产品曾经广为流传，据说最早的发起者对Mysql很精通，后来从阿里跳槽了，阿里随后开源的Cobar，并维持到2013年年初，然后，就没有然后了。

#### Cobar的十个秘密

##### 第一个秘密：Cobar会假死？

是的，很多人遇到这个问题。如何来验证这点呢？可以做个简单的小实验，假如你的分片表中配置有表company,则打开mysql终端，执行下面的SQL：

select sleep(500) from company;

此SQL会执行等待500秒，你再努力以最快的速度打开N个mysql终端，都执行相同的SQL，确保N>当前Cobra的执行线程数：

show @@threadpool

的所有Processor1-E的线程池的线程数量总和，然后你再执行任何简单的SQL，或者试图新建立连接，都会无法响应，此时

show @@threadpool

里面看到TASK\_QUEUE\_SIZE已经在积压中。

不可能吧，据说Cobra是NIO的非阻塞的，怎么可能阻塞！别激动，去看看代码，Cobra前端是NIO的，而后端跟Mysql的交互，是阻塞模式，其NIO代码只给出了框架，还未来得及实现。真相永远在代码里，所以，为了发现真相，还是转行去做码农吧！貌似码农也像之前的技术工人，越来越稀罕了。

##### 第二个秘密：高可用的陷阱？

每一个秘密的背后，总是隐藏着更大的秘密。Cobra假死的的秘密背后，还隐藏着一个更为“强大”的秘密，那就是假死以后，Cobra的频繁主从切换问题。我们看看Cobra的一个很好的优点——“高可用性”的实现机制，下图解释了Cobra如何实现高可用性：

分片节点dn2\_M1配置了两个dataSource，并且配置了心跳检测(heartbeat)语句，在这种配置下，每个dataNode会定期对当前正在使用的dataSource执行心跳检测，默认是第一个，频率是10秒钟一次，当心跳检测失败以后，会自动切换到第二个dataSource上进行读写，假如Cobra发生了假死，则在假死的1分钟内，Cobra会自动切换到第二个节点上，因为假死的缘故，第二个节点的心跳检测也超时。于是，1分钟内Cobra频繁来回切换，懂得MySQL主从复制机制的人都知道，在两个节点上都执行写操作意味着什么？——可能数据一致性被破坏，谁也不知道那个机器上的数据是最新的。

还有什么情况下，会导致心跳检测失败呢？这是一个不得不说的秘密：当后端数据库达到最大连接后，会对新建连接全部拒绝，此时，Cobar的心跳检测所建立的新连接也会被拒绝，于是，心跳检测失败，于是，一切都悄悄的发生了。

幸好，大多数同学都没有配置高可用性，或者还不了解此特性，因此，这个秘密，一直在安全的沉睡。

##### 第三个秘密：看上去很美的自动切换

Cobar很诱人的一个特性是高可用性，高可用性的原理是数据节点DataNode配置引用两个DataSource，并做心跳检测，当第一个DataSource心跳检测失败后，Cobar自动切换到第二个节点，当第二个节点失败以后，又自动切换回第一个节点，一切看起来很美，无人值守，几乎没有宕机时间。

在真实的生产环境中，我们通常会用至少两个Cobar实例组成负载均衡，前端用硬件或者HAProxy这样的负载均衡组件，防止单点故障，这样一来，即使某个Cobar实例死了，还有另外一台接手，某个Mysql节点死了，切换到备节点继续，至此，一切看起来依然很美，喝着咖啡，听着音乐，领导视察，你微笑着点头——No problem，Everything is OK!直到有一天，某个Cobar实例果然如你所愿的死了，不管是假死还是真死，你按照早已做好的应急方案，优雅的做了一个不是很艰难的决定——重启那个故障节点，然后继续喝着咖啡，听着音乐，轻松写好故障处理报告发给领导，然后又度过了美好的一天。

  你忽然被深夜一个电话给惊醒，你来不及发火，因为你的直觉告诉你，这个问题很严重，大量的订单数据发生错误很可能是昨天重启cobar导致的数据库发生奇怪的问题。你努力排查了几个小时，终于发现，主备两个库都在同时写数据，主备同步失败，你根本不知道那个库是最新数据，紧急情况下，你做了一个很英明的决定，停止昨天故障的那个cobar实例，然后你花了3个通宵，解决了数据问题。

  这个陷阱的代价太高，不知道有多少同学中枪过，反正我也是躺着中枪过了。若你还不清楚为何会产生这个陷阱，现在我来告诉你：

1. Cobar启动的时候，会用默认第一个Datasource进行数据读写操作；
2. 当第一个Datasource心跳检测失败，会切换到第二个Datasource；
3. 若有两个以上的Cobar实例做集群，当发生节点切换以后，你若重启其中任何一台Cobar，就完美调入陷阱；

那么，怎么避免这个陷阱？目前只有一个办法，节点切换以后，尽快找个合适的时间，全部集群都同时重启，避免隐患。为何是重启而不是用节点切换的命令去切换？想象一下32个分片的数据库，要多少次切换？

  MyCAT怎么解决这个问题的？很简单，节点切换以后，记录一个properties文件（ conf目录下），重启的时候，读取里面的节点index，真正实现了无故障无隐患的高可用性。

##### 第四个秘密：只实现了一半的NIO

  NIO技术用作JAVA服务器编程的技术标准，已经是不容置疑的业界常规做法，若一个Java程序员，没听说过NIO，都不好意思说自己是Java人。所以Cobar采用NIO技术并不意外，但意外的是，只用了一半。

  Cobar本质上是一个“数据库路由器”，客户端连接到Cobar，发生SQL语句，Cobar再将SQL语句通过后端与MySQL的通讯接口Socket发出去，然后将结果返回给客户端的Socket中。下面给出了SQL执行过程简要逻辑：

SQL->FrontConnection->Cobar->MySQLChanel->MySQL

  FrontConnection 实现了NIO通讯，但MySQLChanel则是同步的IO通讯，原因很简单，指令比较复杂，NIO实现有难度，容易有BUG。后来最新版本Cobar尝试了将后端也NIO化，大概实现了80%的样子，但没有完成，也存在缺陷。

  由于前端NIO，后端BIO，于是另一个有趣的设计产生了——两个线程池，前端NIO部分一个线程池，后端BIO部分一个线程池。各自相互不干扰，但这个设计的结果，导致了线程的浪费，也对性能调优带来很大的困难。

  由于后端是BIO，所以，也是Cobar吞吐量无法太高、另外也是其假死的根源。

  MyCAT在Cobar的基础上，完成了彻底的NIO通讯，并且合并了两个线程池，这是很大一个提升。从1.1版本开始，MyCAT则彻底用了JDK7的AIO，有一个重要提升。

##### 第五个秘密：阻塞、又见阻塞

  Cobar本质上类似一个交换机，将后端Mysql 的返回结果数据经过加工后再写入前端连接并返回，于是前后端连接都存在一个“写队列”用作缓冲，后端返回的数据发到前端连接FrontConnection的写队列中排队等待被发送，而通常情况下，后端写入的的速度要大于前端消费的速度，在跨分片查询的情况下，这个现象更为明显，于是写线程就在这里又一次被阻塞。

  解决办法有两个，增大每个前端连接的“写队列”长度，减少阻塞出现的情况，但此办法只是将问题抛给了使用者，要是使用者能够知道这个写队列的默认值小了，然后根据情况进行手动尝试调整也行，但Cobar的代码中并没有把这个问题暴露出来，比如写一个告警日志，队列满了，建议增大队列数。于是绝大多数情况下，大家就默默的排队阻塞，无人知晓。

  MyCAT解决此问题的方式则更加人性化，首先将原先数组模式的固定长度的队列改为链表模式，无限制，并且并发性更好，此外，为了让用户知道是否队列过长了（一般是因为SQL结果集返回太多，比如1万条记录），当超过指定阀值（可配）后，会产生一个告警日志。

<system><property name="frontWriteQueueSize">1024</property></system>

##### 第六个秘密：又爱又恨的SQL 批处理模式

  正如一枚硬币的正反面无法分离，一块磁石怎样切割都有南北极，爱情中也一样，爱与恨总是纠缠着，无法理顺，而Cobar的 SQL 批处理模式，也恰好是这样一个令人又爱又恨的个性。

  通常的SQL 批处理，是将一批SQL作为一个处理单元，一次性提交给数据库，数据库顺序处理完以后，再返回处理结果，这个特性对于数据批量插入来说，性能提升很大，因此也被普遍应用。JDBC的代码通常如下：

String sql = "insert into travelrecord (id,user\_id,traveldate,fee,days) values(?,?,?,?,?)";ps = con.prepareStatement(sql);for (Map<String, String> map : list) {  ps.setLong(1, Long.parseLong(map.get("id")));  ps.setString(2, (String) map.get("user\_id"));  ps.setString(3, (String) map.get("traveldate"));  ps.setString(4, (String) map.get("fee"));  ps.setString(5, (String) map.get("days"));  ps.addBatch();}ps.executeBatch();con.commit();ps.clearBatch();

  但Cobar的批处理模式的实现，则有几个地方是与传统不同的：

* 提交到cobar的批处理中的每一条SQL都是单独的数据库连接来执行的
* 批处理中的SQL并发执行

并发多连接同时执行，则意味着Batch执行速度的提升，这是让人惊喜的一个特性，但单独的数据库连接并发执行，则又带来一个意外的副作用，即事务跨连接了，若一部分事务提交成功，而另一部分失败，则导致脏数据问题。看到这里，你是该“爱”呢还是该“恨”？

先不用急着下结论，我们继续看看Cobar的逻辑，SQL并发执行，其实也是依次获取独立连接并执行，因此还是有稍微的时间差，若某一条失败了，则cobar会在会话中标记”事务失败，需要回滚“，下一个没执行的SQL就抛出异常并跳过执行，客户端就捕获到异常，并执行rollback，回滚事务。绝大多数情况下，数据库正常运行，此刻没有宕机，因此事务还是完整保证了，但万一恰好在某个SQL commit指令的时候宕机，于是杯具了，部分事务没有完成，数据没写入。但这个概率有多大呢？一条insert insert 语句执行commit指令的时间假如是50毫秒，100条同时提交，最长跨越时间是5000毫秒，即5秒中，而这个C指令的时间占据程序整个插入逻辑的时间的最多20%，假如程序批量插入的执行时间占整个时间的20%（已经很大比例了），那就是20%×20%=4%的概率，假如机器的可靠性是99.9%，则遇到失败的概率是0.1%×4%=十万分之四。十万分之四，意味着99.996%的可靠性，亲，可以放心了么？

另外一个问题，即批量执行的SQL，通常都是insert的，插入成功就OK，失败的怎么办？通常会记录日志，重新找机会再插入，因此建议主键是能日志记录的，用于判断数据是否已经插入。

最后，假如真要多个SQL使用同一个后端MYSQL连接并保持事务怎么办？就采用通常的事务模式，单条执行SQL，这个过程中，Cobar会采用Session中上次用过的物理连接执行下一个SQL语句，因此，整个过程是与通常的事务模式完全一致。

##### 第六个秘密：庭院深深锁清秋

说起死锁，貌似我们大家都只停留在很久远的回忆中，只在教科书里看到过，也看到过关于死锁产生的原因以及破解方法，只有DBA可能会偶尔碰到数据库死锁的问题。但很多用了Cobar的同学后来经常发现一个奇怪的问题，SQL很久没有应答，百思不得其解，无奈之下找DBA排查后发现竟然有数据库死锁现象，而且比较频繁发生。要搞明白为什么Cobar增加了数据库死锁的概率，只能从源码分析，当一个SQL需要拆分为多条SQL去到多个分片上执行的时候，这个执行过程是并发执行的，即N个SQL同时在N个分片上执行，这个过程抽象为教科书里的事务模型，就变成一个线程需要锁定N个资源并执行操作以后，才结束事务。当这N个资源的锁定顺序是随机的情况下，那么就很容易产生死锁现象，而恰好Cobar并没有保证N个资源的锁定顺序，于是我们再次荣幸“中奖”。

##### 第七个秘密：出乎意料的连接池

数据库连接池，可能是仅次于线程池的我们所最依赖的“资源池”，其重要性不言而喻，业界也因此而诞生了多个知名的开源数据库连接池。我们知道，对于一个MySQL Server来说，最大连接通常是1000-3000之间，这些连接对于通常的应用足够了，通常每个应用一个Database独占连接，因此足够用了，而到了Cobar的分表分库这里，就出现了问题，因为Cobar对后端MySQL的连接池管理是基于分片——Database来实现的，而不是整个MySQL的连接池共享，以一个分片数为100的表为例，假如50个分片在Server1上，就意味着Server1上的数据库连接被切分为50个连接池，每个池是20个左右的连接，这些连接池并不能互通，于是，在分片表的情况下，我们的并发能力被严重削弱。明明其他水池的水都是满的，你却只能守着空池子等待。。。

##### 第八个秘密：无奈的热装载

Cobar有一个优点，配置文件热装载，不用重启系统而热装载配置文件，但这里存在几个问题，其中一个问题是很多人不满的，即每次重载都把后端数据库重新断连一次，导致业务中断，而很多时候，大家改配置仅仅是为了修改分片表的定义，规则，增加分片表或者分片定义，而不会改变数据库的配置信息，这个问题由来已久，但却不太好修复。

##### 第九个秘密：不支持读写分离

不支持读写分离，可能熟悉相关中间件的同学第一反应就是惊讶，因为一个MySQL Proxy最基本的功能就是提供读写分离能力，以提升系统的查询吞吐量和查询性能。但的确Cobar不支持读写分离，而且根据Cobar的配置文件，要实现读写分离，还很麻烦。可能有些人认为，因为无法保证读写分离的时延，因此无法确定是否能查到之前写入的数据，因此读写分离并不重要，但实际上，Mycat的用户里，几乎没有不使用读写分离功能的，后来还有志愿者增加了强制查询语句走主库（写库）的功能，以解决刚才那个问题。

##### 第十个秘密：不可控的主从切换

Cobar提供了MySQL主从切换能力，这个功能很实用也很方便，但你无法控制它的切换开启或关闭，有时候我们不想它自动切换，因为到目前为止，还没有什么好的方法来确认MySQL写节点宕机的时候，备节点是否已经100%完成数据同步，因此存在数据不一致的风险，如何更可靠的确定是否能安全切换，这个问题比较复杂，Mycat也一直在努力完善这个特性。

### 第二章 Mycat的概述

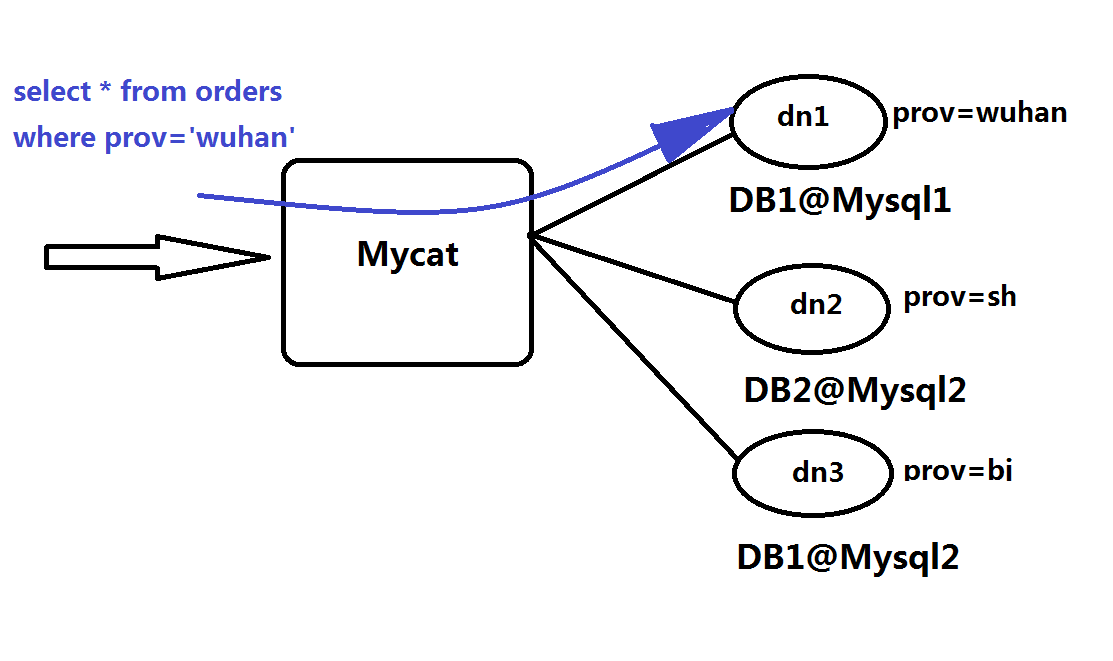
#### 2.1 功能介绍

Mycat是什么？从定义和分类来看，它是一个开源的分布式数据库系统，是一个实现了MySQL协议的的Server，前端用户可以把它看作是一个数据库代理，用MySQL客户端工具和命令行访问，而其后端可以用MySQL原生（Native）协议与多个MySQL服务器通信，也可以用JDBC协议与大多数主流数据库服务器通信，其核心功能是分表分库，即将一个大表水平分割为N个小表，存储在后端MySQL服务器里或者其他数据库里。  
 Mycat发展到目前的版本，已经不是一个单纯的MySQL代理了，它的后端可以支持MySQL、SQL Server、Oracle、DB2、PostgreSQL等主流数据库，也支持MongoDB这种新型NoSQL方式的存储，未来还会支持更多类型的存储。而在最终用户看来，无论是那种存储方式，在Mycat里，都是一个传统的数据库表，支持标准的SQL语句进行数据的操作，这样一来，对前端业务系统来说，可以大幅降低开发难度，提升开发速度，在测试阶段，可以将一个表定义为任何一种Mycat支持的存储方式，比如MySQL的MyASIM表、内存表、或者MongoDB、LevelDB以及号称是世界上最快的内存数据库MemSQL上。试想一下，用户表存放在MemSQL上，大量读频率远超过写频率的数据如订单的快照数据存放于InnoDB中，一些日志数据存放于MongoDB中，而且还能把Oracle的表跟MySQL的表做关联查询，你是否有一种不能呼吸的感觉？而未来，还能通过Mycat自动将一些计算分析后的数据灌入到Hadoop中，并能用Mycat+Storm/Spark Stream引擎做大规模数据分析，看到这里，你大概明白了，Mycat是什么？Mycat就是BigSQL，Big Data On SQL Database。  
对于DBA来说，可以这么理解Mycat：  
 **Mycat就是MySQL Server，而Mycat后面连接的MySQL Server，就好象是MySQL的存储引擎,如InnoDB，MyISAM等，因此，Mycat本身并不存储数据，数据是在后端的MySQL上存储的，因此数据可靠性以及事务等都是MySQL保证的，简单的说，Mycat就是MySQL最佳伴侣，它在一定程度上让MySQL拥有了能跟Oracle PK的能力。**  
对于软件工程师来说，可以这么理解Mycat：  
 **Mycat就是一个近似等于MySQL的数据库服务器，你可以用连接MySQL的方式去连接Mycat（除了端口不同，默认的Mycat端口是8066而非MySQL的3306，因此需要在连接字符串上增加端口信息），大多数情况下，可以用你熟悉的对象映射框架使用Mycat，但建议对于分片表，尽量使用基础的SQL语句，因为这样能达到最佳性能，特别是几千万甚至几百亿条记录的情况下。**

对于架构师来说，可以这么理解Mycat：  
 **Mycat是一个强大的数据库中间件，不仅仅可以用作读写分离、以及分表分库、容灾备份，而且可以用于多租户应用开发、云平台基础设施、让你的架构具备很强的适应性和灵活性，借助于即将发布的Mycat智能优化模块，系统的数据访问瓶颈和热点一目了然，根据这些统计分析数据，你可以自动或手工调整后端存储，将不同的表映射到不同存储引擎上，而整个应用的代码一行也不用改变。**

当前是个大数据的时代，但究竟怎样规模的数据适合数据库系统呢？对此，国外有一个数据库领域的权威人士说了一个结论：千亿以下的数据规模仍然是数据库领域的专长，而Hadoop等这种系统，更适合的是千亿以上的规模。所以，Mycat适合1000亿条以下的单表规模，如果你的数据超过了这个规模，请投靠Mycat Plus吧！

#### 2.2 Mycat 原理

Mycat的原理并不复杂，复杂的是代码，如果代码也不复杂，那么早就成为一个传说了。  
Mycat的原理中最重要的一个动词是“拦截”，它拦截了用户发送过来的SQL语句，首先对SQL语句做了一些特定的分析：如分片分析、路由分析、读写分离分析、缓存分析等，然后将此SQL发往后端的真实数据库，并将返回的结果做适当的处理，最终再返回给用户。  


上述图片里，Orders表被分为三个分片datanode（简称dn)，这三个分片是分布在两台MySQL Server上(DataHost)，即datanode=database@datahost方式，因此你可以用一台到N台服务器来分片，分片规则为（sharding rule)典型的字符串枚举分片规则，一个规则的定义是分片字段（sharding column)+分片函数(rule function)，这里的分片字段为prov而分片函数为字符串枚举方式。  
 当Mycat收到一个SQL时，会先解析这个SQL，查找涉及到的表，然后看此表的定义，如果有分片规则，则获取到SQL里分片字段的值，并匹配分片函数，得到该SQL对应的分片列表，然后将SQL发往这些分片去执行，最后收集和处理所有分片返回的结果数据，并输出到客户端。以select \* from Orders where prov=?语句为例，查到prov=wuhan，按照分片函数，wuhan返回dn1，于是SQL就发给了MySQL1，去取DB1上的查询结果，并返回给用户。  
 如果上述SQL改为select \* from Orders where prov in (‘wuhan’,‘beijing’)，那么，SQL就会发给MySQL1与MySQL2去执行，然后结果集合并后输出给用户。但通常业务中我们的SQL会有Order By 以及Limit翻页语法，此时就涉及到结果集在Mycat端的二次处理，这部分的代码也比较复杂，而最复杂的则属两个表的Jion问题，为此，Mycat提出了创新性的ER分片、全局表、HBT（Human Brain Tech)人工智能的Catlet、以及结合Storm/Spark引擎等十八般武艺的解决办法，从而成为目前业界最强大的方案，这就是开源的力量！

#### 2.3 Mycat 应用场景

Mycat发展到现在，适用的场景已经很丰富，而且不断有新用户给出新的创新性的方案，以下是几个典型的应用场景：

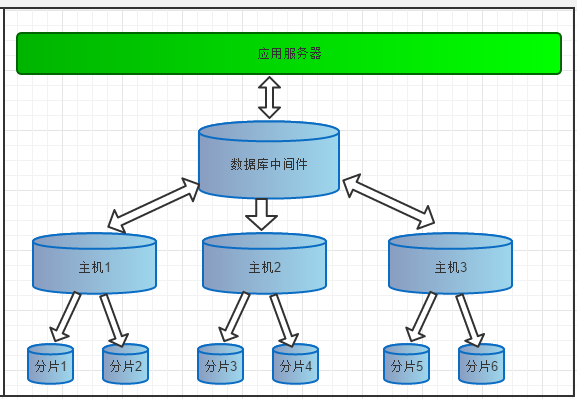
* 单纯的读写分离，支持读写分离，主从切换
* 分表分库，对于超过1000万的表进行分片，最大支持1000亿的单表分片
* 多租户应用，每个应用一个库，但应用程序只连接Mycat，从而不改造程序本身，实现多租户化
* 报表系统，借助于Mycat的分表能力，处理大规模报表的统计
* 替代Hbase，分析大数据
* 作为海量数据实时查询的一种简单有效方案，比如100亿条频繁查询的记录需要在3秒内查询出来结果，除了基于主键的查询，还可能存在范围查询或其他属性查询，此时Mycat可能是最简单有效的选择

#### 2.4 Mycat长期路线图

* **强化分布式数据库中间件**的方面的功能，使之具备丰富的插件、强大的数据库智能优化功能、全面的系统监控能力、以及方便的数据运维工具，实现在线数据扩容、迁移等高级功能
* 进一步挺进大数据计算领域，深度结合Spark Stream和Storm等分布式实时流引擎，能够完成快速的巨表关联、排序、分组聚合等 OLAP方向的能力，并集成一些热门常用的实时分析算法，让工程师以及DBA们更容易用Mycat实现一些高级数据分析处理功能。
* 不断强化Mycat开源社区的技术水平，吸引更多的IT技术专家，使得Mycat社区成为中国的Apache，并将Mycat推到Apache基金会，成为国内顶尖开源项目，最终能够让一部分志愿者成为专职的Mycat开发者，荣耀跟实力一起提升。
* 依托Mycat社区，聚集100个CXO级别的精英，众筹建设亲亲山庄，Mycat社区+亲亲山庄=中国最大IT O2O社区

### 第三章 Mycat中的概念

#### 数据库中间件

* 前面讲了Mycat是一个开源的分布式数据库系统，但是由于真正的数据库需要存储引擎，而Mycat并没有存储引擎，所以**并不是完全意义的分布式数据库系统**。  
   那么Mycat是什么？Mycat是数据库中间件，就是介于数据库与应用之间，进行数据处理与交互的中间服务。由于前面讲的对数据进行分片处理之后，从原有的一个库，被切分为多个分片数据库，所有的分片数据库集群构成了整个完整的数据库存储。
* 
* 如上图所表示，数据被分到多个分片数据库后，应用如果需要读取数据，就要需要处理多个数据源的数据。如果没有数据库中间件，那么应用将直接面对分片集群，数据源切换、事务处理、数据聚合都需要应用直接处理，原本该是专注于业务的应用，将会花大量的工作来处理分片后的问题，最重要的是每个应用处理将是完全的重复造轮子。  
  所以有了数据库中间件，应用只需要集中与业务处理，大量的通用的数据聚合，事务，数据源切换都由中间件来处理，中间件的性能与处理能力将直接决定应用的读写性能，所以一款好的数据库中间件至关重要。

### 第四章 Mycat支持的读写分离

#### 4.1 安装mycat

[root@db01 tools]# tar xf Mycat-Server-1.4-RC-Linux-RW-20150724.tar.gz

[root@db01 tools]# mv mycat /application/

[root@db01 ~]# tail -1 /etc/profile

**export PATH=application/mycat/bin:$PATH**

#### 4.2 安装java环境

##### 4.2.1 下载JDK

1、登录Sun的JDK官方下载网址:http://java.sun.com/javase/downloads/index.jsp

2、下载jdk-8u11-linux-x64.tar.gz

##### 4.2.2 配置环境变量

**配置环境变量PATH,JAVA\_HOME,CLASSPATH**

[root@db01 mycat]# **tail -3 /etc/profile**

export JAVA\_HOME=/application/jdk1.5.0\_06

export PATH=$JAVA\_HOME/bin:/application/mycat/bin/:$PATH

export CLASSPATH=$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

[root@db01 mycat]# **. /etc/profile**  **#<=让配置文件生效**

[root@db01 bin]# **java -version**   **#<=查看java版本**

java version "1.8.0\_11"

Java(TM) SE Runtime Environment (build 1.8.0\_11-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.11-b03, mixed mode)

#### 4.3 创建管理用户

主库上对web用户授权如下：

用户：web 密码：oldboy123 端口：3306

权限：**insert,delete,update,select**

命令：grant **insert,delete,update,select** on oldboy.\* to web@'172.16.1.%' identified by 'oldboy123';

从库上对web用户授权如下：

用户：web 密码：oldboy123 端口：3306

权限： **select**

**提示：由于主库和从库是同步复制的，所以从库上的web用户会自动和主库的一致，即无法实现只读select的授权**

**revoke insert,update,delete on oldboy.\* from web@'172.16.1.%'; #<=回收insert update delete 权限**

#### 4.4 修改mycat配置文件

##### 4.4.1 server.xml

[root@db01 ~]# vim server.xml

</system>

<user name="**web**">

<property name="password">**oldboy123**</property>

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

</user>

<user name="**web\_r**">

<property name="password">**oldboy123**</property>

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

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

注意：

1、这里配置的是可以连接**主库**的两个用户

用户：web 密码：oldboy123 给予此用户oldboy数据库**增删改查**的权限。

用户：web\_r 密码：oldboy123 给予此用户oldboy数据库**读**的权限。

2、这里的oldboy，不一定是你数据库上的真实库名，可以任意指定，只要接下来和schema.xml的配置文件的库名统一即可。

##### 4.4.1 schema.xml

[root@db01 conf]# vim schema.xml

<schema name="oldboy" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">

<dataNode name="dn1" dataHost="localhost1" database="oldboy" />

<dataNode name="dn2" dataHost="localhost1" database="oldboy" />

<dataNode name="dn3" dataHost="localhost1" database="oldboy" />

<dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select user()</heartbeat>

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

**<writeHost host="hostM1" url="172.16.1.51:3306" user="web"**

**password="oldboy123">**

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

**<readHost host="hostS1" url="172.16.1.53:3306" user="web"**

**password="oldboy123"/>**

(1)<schema name="oldboy" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">

这里的oldboy就是我们所宣称的数据库名称，必须和server.xml中的用户指定的数据库名称一致。添加一个dataNode=”dn1”，是指定了我们这个库只有在dn1上，没有分库。

(2) <dataNode name="dn1" dataHost="localhost1" database="oldboy" />

这里只需要改database的名字，db1就是你真是的数据库上的数据库名，可根据自己的数据库名称修改。

(3) <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

**需要配置的位置：**

**balance="1" writeType="0" switchType="1"**

**balance**

1、balance=0 不开启读写分离机制，所有读操作都发送到当前可用的writehostle .

2、balance=1 全部的readhost与stand by writeHost 参与select语句的负载均衡。简单的说，双主双从模式(M1🡪S1,M2🡪S2，并且M1和M2互为主备)，正常情况下，M1，S1，S2都参与select语句的复杂均衡。

3、balance=2 所有读操作都随机的在readhost和writehost上分发

**writeType**   
负载均衡类型，目前的取值有3种：  
1、writeType="0", 所有写操作发送到配置的第一个writeHost。  
2、writeType="1"，所有写操作都随机的发送到配置的writeHost。  
3、writeType="2"，不执行写操作。

**switchType**   
1、switchType=-1 表示不自动切换  
2、switchType=1 默认值，自动切换  
3、switchType=2 基于MySQL 主从同步的状态决定是否切换

(4)

<writeHost host="hostM1" url="172.16.1.51:3306" user="web"

password="oldboy123">

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

<readHost host="hostS1" url="172.16.1.53:3306" user="web"

password="oldboy123"/>

注意：

这里配置的是读写服务器的IP地址和端口访问，访问的用户名和密码；而且一定要先在客户端测试OK了以后在进行配置。

#### 4.5 启动mycat

[root@db01 ~]# mycat console  **#<=启动mycat**

Running Mycat-server...

wrapper | --> Wrapper Started as Console

wrapper | Launching a JVM...

jvm 1 | Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=64M; support was removed in 8.0

jvm 1 | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org

jvm 1 | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.

jvm 1 |

jvm 1 | log4j 2015-12-29 20:02:18 [./conf/log4j.xml] load completed.

jvm 1 | MyCAT Server startup successfully. see logs in logs/mycat.log

**出现这个界面表示mycat已经成功启动了，如果出现错误的话请查看日志mycat.log**

#### 4.6 读写分离测试

**1、在客户端连接mysql主库服务器：**

[root@old-boy ~]# mysql -uweb -poldboy123 -h172.16.1.51 -P8066

mysql> use oldboy

mysql> select \* from company;

+----+----------+

| id | name |

+----+----------+

| 1 | artpop |

| 2 | ladygaga |

| 3 | artpop |

+----+----------+

mysql> insert into company values(4,'shakeit');

mysql> select \* from company;

+----+----------+

| id | name |

+----+----------+

| 1 | artpop |

| 2 | ladygaga |

| 3 | artpop |

| 4 | shakeit |

+----+----------+

**2、关闭从库的sql线程**

mysql> show slave status\G

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

mysql> **stop slave sql\_thread;**

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: No

mysql> insert into company values(5,'shakeit');

mysql> select \* from company;

+----+----------+

| id | name |

+----+----------+

| 1 | artpop |

| 2 | ladygaga |

| 3 | artpop |

| 4 | shakeit |

+----+----------+

**3、登录到主库上查看company信息**

mysql -uroot -p123456

mysql> select \* from company;

+----+----------+

| id | name |

+----+----------+

| 1 | artpop |

| 2 | ladygaga |

| 3 | artpop |

| 4 | shakeit |

**| 5 | shakeit |**

+----+----------+

**4、在从库上查看company信息**

mysql> select \* from company;

+----+----------+

| id | name |

+----+----------+

| 1 | artpop |

| 2 | ladygaga |

| 3 | artpop |

| 4 | shakeit |

+----+----------+

**5、打开从库的sql线程**

mysql> start slave sql\_thread;

mysql> show slave status\G

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

**6、在客户端查找company表内容**

mysql> select \* from company; **#<=读写分离验证成功！**

+----+----------+

| id | name |

+----+----------+

| 1 | artpop |

| 2 | ladygaga |

| 3 | artpop |

| 4 | shakeit |

**| 5 | shakeit |**

+----+----------+

5 rows in set (0.01 sec)

### 第五章 管理命令与监控

mycat自身有类似其他数据库的管理监控方式，可通过mysql命令行，登陆端口9066执行相应的SQL操作，也可通过jdbc的方式进行远程连接管理。

登录：目前mycat有两个端口，8066数据端口，9066管理端口。命令行登录时通过9066管理端口来执行：

mysql -uweb -poldboy123 -h127.0.0.1 -P9066 -Doldboy

选项：

-h 后面接主机

-u mycat server.xml配置的逻辑库用户

-p mycat server.xml配置的逻辑库密码

-P 后面接的端口9066，注意P大写

-D Mycat server.xml中配置的逻辑库

**1、查看所有的命令，如下：**

mysql> **show @@help;**

+--------------------------------------+-----------------------------------+

| STATEMENT | DESCRIPTION |

+--------------------------------------+-----------------------------------+

| clear @@slow where datanode = ? | Clear slow sql by datanode |

| clear @@slow where schema = ? | Clear slow sql by schema |

| kill @@connection id1,id2,... | Kill the specified connections |

| offline | Change MyCat status to OFF |

| online | Change MyCat status to ON |

| reload @@config | Reload basic config from file |

| reload @@config\_all | Reload all config from file |

| reload @@route | Reload route config from file |

| reload @@user | Reload user config from file |

| rollback @@config | Rollback all config from memory |

| rollback @@route | Rollback route config from memory |

| rollback @@user | Rollback user config from memory |

| show @@backend | Report backend connection status |

| show @@cache | Report system cache usage |

| show @@command | Report commands status |

| show @@connection | Report connection status |

| show @@connection.sql | Report connection sql |

| show @@database | Report databases |

| show @@datanode | Report dataNodes |

| show @@datanode where schema = ? | Report dataNodes |

| show @@datasource | Report dataSources |

| show @@datasource where dataNode = ? | Report dataSources |

| show @@heartbeat | Report heartbeat status |

| show @@parser | Report parser status |

| show @@processor | Report processor status |

| show @@router | Report router status |

| show @@server | Report server status |

| show @@session | Report front session details |

| show @@slow where datanode = ? | Report datanode slow sql |

| show @@slow where schema = ? | Report schema slow sql |

| show @@sql where id = ? | Report specify SQL |

| show @@sql.detail where id = ? | Report execute detail status |

| show @@sql.execute | Report execute status |

| show @@sql.slow | Report slow SQL |

| show @@threadpool | Report threadPool status |

| show @@time.current | Report current timestamp |

| show @@time.startup | Report startup timestamp |

| show @@version | Report Mycat Server version |

| stop @@heartbeat name:time | Pause dataNode heartbeat |

| switch @@datasource name:index | Switch dataSource

**2、更新配置文件**

mysql> **reload @@config;**

Query OK, 1 row affected (0.13 sec)

Reload config success

**3、显示mycat数据库的列表，对应的在****scehma.xml配置的逻辑库**

mysql> **show @@database;**

+----------+

| DATABASE |

+----------+

| oldboy |

+----------+

1 row in set (0.00 sec)

**4、显示mycat数据节点的列表，对应的是scehma.xml配置文件的dataNode节点**

**mysql> show @@datanode;**

+------+-------------------+-------+-------+--------+------+------+---------+------------+----------+---------+---------------+

| NAME | DATHOST | INDEX | TYPE | ACTIVE | IDLE | SIZE | EXECUTE | TOTAL\_TIME | MAX\_TIME | MAX\_SQL | RECOVERY\_TIME |

+------+-------------------+-------+-------+--------+------+------+---------+------------+----------+---------+---------------+

| dn1 | localhost1/oldboy | 0 | mysql | 0 | 10 | 1000 | 457 | 0 | 0 | 0 | -1 |

| dn2 | localhost1/oldboy | 0 | mysql | 0 | 10 | 1000 | 457 | 0 | 0 | 0 | -1 |

| dn3 | localhost1/oldboy | 0 | mysql | 0 | 10 | 1000 | 457 | 0 | 0 | 0 | -1 |

+------+-------------------+-------+-------+--------+------+------+---------+------------+----------+---------+---------------+

3 rows in set (0.00 sec)

**其中，NAME表示datanode的名称；dataHost 对应的是dataHost属性的值，数据主机的名称，ACTIVE表示活跃的连接数，IDIE表示闲置的连接数，SIZE对应的是总连接的数量。**

**5、报告心跳状态**

**mysql> show @@heartbeat;**

+--------+-------+-------------+------+---------+-------+--------+---------+--------------+---------------------+-------+

| NAME | TYPE | HOST | PORT | RS\_CODE | RETRY | STATUS | TIMEOUT | EXECUTE\_TIME | LAST\_ACTIVE\_TIME | STOP |

+--------+-------+-------------+------+---------+-------+--------+---------+--------------+---------------------+-------+

| hostM1 | mysql | 172.16.1.51 | 3306 | 1 | 0 | idle | 0 | 1,1,1 | 2015-12-29 21:39:40 | false |

| hostS1 | mysql | 172.16.1.53 | 3306 | 1 | 0 | idle | 0 | 3,3,3 | 2015-12-29 21:39:40 | false |

+--------+-------+-------------+------+---------+-------+--------+---------+--------------+---------------------+-------+

2 rows in set (0.01 sec)

**RS\_STATUS状态为1，正常状态**

**6、获取当前mycat的版本**

mysql> **show @@version;**

+--------------------------------------+

| VERSION |

+--------------------------------------+

| 5.5.8-mycat-1.4-alpha-20150520235658 |

+--------------------------------------+

1 row in set (0.00 sec)

**7、显示mycat前端连接状态**

mysql> **show @@connection;**

+------------+------+-----------+------+------------+--------+---------+--------+---------+---------------+-------------+------------+---------+------------+

| PROCESSOR | ID | HOST | PORT | LOCAL\_PORT | SCHEMA | CHARSET | NET\_IN | NET\_OUT | ALIVE\_TIME(S) | RECV\_BUFFER | SEND\_QUEUE | txlevel | autocommit |

+------------+------+-----------+------+------------+--------+---------+--------+---------+---------------+-------------+------------+---------+------------+

| Processor0 | 6 | 127.0.0.1 | 9066 | 46490 | oldboy | utf8:33 | 281 | 6164 | 1008 | 4096 | 0 | | |

+------------+------+-----------+------+------------+--------+---------+--------+---------+---------------+-------------+------------+---------+------------+

1 row in set (0.00 sec)

**8、显示mycat后端连接状态**

mysql> **show @@backend;**

+------------+------+---------+-------------+------+--------+--------+---------+------+--------+----------+------------+--------+---------+---------+------------+

| processor | id | mysqlId | host | port | l\_port | net\_in | net\_out | life | closed | borrowed | SEND\_QUEUE | schema | charset | txlevel | autocommit |

+------------+------+---------+-------------+------+--------+--------+---------+------+--------+----------+------------+--------+---------+---------+------------+

| Processor0 | 1 | 30 | 172.16.1.51 | 3306 | 14881 | 3554 | 1068 | 5041 | false | false | 0 | oldboy | utf8:33 | 3 | true |

| Processor0 | 2 | 32 | 172.16.1.51 | 3306 | 14883 | 3554 | 1068 | 5041 | false | false | 0 | oldboy | utf8:33 | 3 | true |

| Processor0 | 3 | 24 | 172.16.1.51 | 3306 | 14875 | 3515 | 1068 | 5041 | false | false | 0 | oldboy | utf8:33 | 3 | true |

| Processor0 | 4 | 28 | 172.16.1.51 | 3306 | 14879 | 3561 | 986 | 5041 | false | false | 0 | oldboy | utf8:33 | 0 | true |

**9、显示数据源**

mysql> **show @@datasource;**

+----------+--------+-------+-------------+------+------+--------+------+------+---------+

| DATANODE | NAME | TYPE | HOST | PORT | W/R | ACTIVE | IDLE | SIZE | EXECUTE |

+----------+--------+-------+-------------+------+------+--------+------+------+---------+

| dn1 | hostM1 | mysql | 172.16.1.51 | 3306 | W | 0 | 10 | 1000 | 525 |

| dn1 | hostS1 | mysql | 172.16.1.53 | 3306 | R | 0 | 8 | 1000 | 522 |

| dn3 | hostM1 | mysql | 172.16.1.51 | 3306 | W | 0 | 10 | 1000 | 525 |

| dn3 | hostS1 | mysql | 172.16.1.53 | 3306 | R | 0 | 8 | 1000 | 522 |

| dn2 | hostM1 | mysql | 172.16.1.51 | 3306 | W | 0 | 10 | 1000 | 525 |

| dn2 | hostS1 | mysql | 172.16.1.53 | 3306 | R | 0 | 8 | 1000 | 522 |

+----------+--------+-------+-------------+------+------+--------+------+------+---------+

6 rows in set (0.01 sec)

**特别说明：**

**reload @@config,这个命令在执行的时候，mycat服务不可用，防止提交的事物出错。**