## Mycat介绍

### 什么是MYCAT

 一个彻底开源的，面向企业应用开发的大数据库集群

 支持事务、ACID、可以替代MySQL的加强版数据库

 一个可以视为MySQL集群的企业级数据库，用来替代昂贵的Oracle集群

 一个融合内存缓存技术、NoSQL技术、HDFS大数据的新型SQL Server

 结合传统数据库和新型分布式数据仓库的新一代企业级数据库产品

 一个新颖的数据库中间件产品

### 关键特性

 支持SQL92标准

 支持MySQL、Oracle、DB2、SQL Server、PostgreSQL等DB的常见SQL语法

 遵守Mysql原生协议，跨语言，跨平台，跨数据库的通用中间件代理。

 基于心跳的自动故障切换，支持读写分离，支持MySQL主从，以及galera cluster集群。

 支持Galera for MySQL集群，Percona Cluster或者MariaDB cluster

 基于Nio实现，有效管理线程，解决高并发问题。

 支持数据的多片自动路由与聚合，支持sum,count,max等常用的聚合函数,支持跨库分页。

 支持单库内部任意join，支持跨库2表join，甚至基于caltlet的多表join。

 支持通过全局表，ER关系的分片策略，实现了高效的多表join查询。

 支持多租户方案。

 支持分布式事务（弱xa）。

 支持XA分布式事务（1.6.5）。

 支持全局序列号，解决分布式下的主键生成问题。

 分片规则丰富，插件化开发，易于扩展。

 强大的web，命令行监控。

 支持前端作为MySQL通用代理，后端JDBC方式支持Oracle、DB2、SQL Server 、 mongodb 、巨杉。

 支持密码加密

 支持服务降级

 支持IP白名单

 支持SQL黑名单、sql注入攻击拦截

 支持prepare预编译指令（1.6）

 支持非堆内存(Direct Memory)聚合计算（1.6）

 支持PostgreSQL的native协议（1.6）

 支持mysql和oracle存储过程，out参数、多结果集返回（1.6）

 支持zookeeper协调主从切换、zk序列、配置zk化（1.6）

 支持库内分表（1.6）

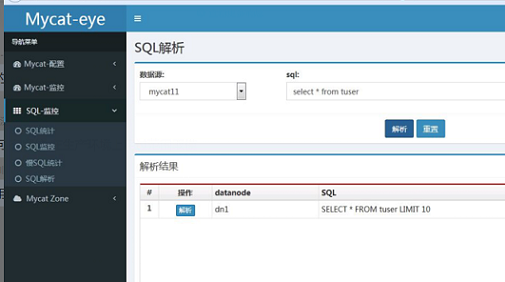
 集群基于ZooKeeper管理，在线升级，扩容，智能优化，大数据处理（2.0开发版）。

### MYCAT监控

 支持对Mycat、Mysql性能监控

 支持对Mycat的JVM内存提供监控服务

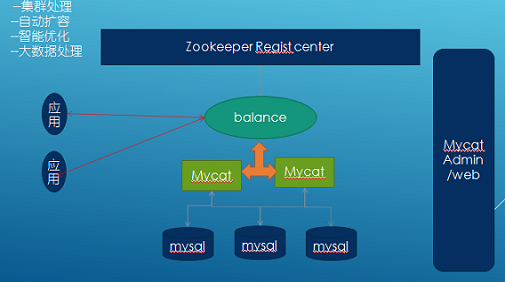
 支持对线程的监控

 支持对操作系统的CPU、内存、磁盘、网络的监控

### 目标

低成本的将现有的单机数据库和应用平滑迁移到“云”端，解决数据存储和业务规模迅速增长情况下的数据瓶颈问题。

### 1.5版本架构



MYCAT官方论坛地址已更新为：[i.mycat.io](http://i.mycat.io/) 欢迎大家访问！MYCAT社区所打造的IT新生活！

### 长期规划2.0

 完全实现分布式事务，完全的支持分布式。

 通过Mycat web（eye）完成可视化配置，及智能监控，自动运维。

 通过mysql 本地节点，完整的解决数据扩容难度，实现自动扩容机制，解决扩容难点。

 支持基于zookeeper的主从切换及Mycat集群化管理。

 通过Mycat Balance 替代第三方的Haproxy，LVS等第三方高可用，完整的兼容Mycat集群节点的动态上下线。

 接入Spark等第三方工具，解决数据分析及大数据聚合的业务场景。

 通过Mycat智能优化，分析分片热点，提供合理的分片建议，索引建议，及数据切分实时业务建议。

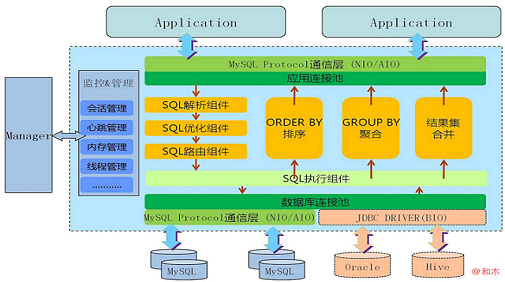
### 优势

基于阿里开源的Cobar产品而研发，Cobar的稳定性、可靠性、优秀的架构和性能以及众多成熟的使用案例使得MYCAT一开始就拥有一个很好的起点，站在巨人的肩膀上，我们能看到更远。业界优秀的开源项目和创新思路被广泛融入到MYCAT的基因中，使得MYCAT在很多方面都领先于目前其他一些同类的开源项目，甚至超越某些商业产品。

MYCAT背后有一支强大的技术团队，其参与者都是5年以上资深软件工程师、架构师、DBA等，优秀的技术团队保证了MYCAT的产品质量。

MYCAT并不依托于任何一个商业公司，因此不像某些开源项目，将一些重要的特性封闭在其商业产品中，使得开源项目成了一个摆设。

### MYCAT架构



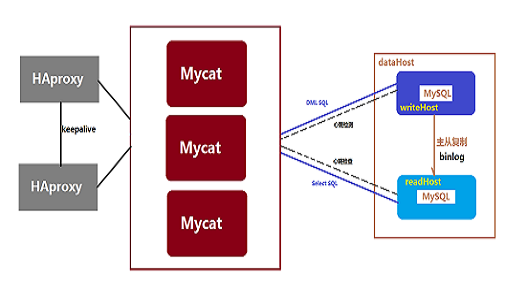
### 企业技术支持服务

随着中国电子商务等互联网业务的不断增长，企业对于数据库集群的需求越来越多，这为MYCAT的发展带来了机遇，同时也带来了很多挑战。由于MYCAT开源社区的开放性，目前已经吸引了来自全国各地2000多名开发志愿者，并在Github上贡献各种开发说明文档上百篇，均可免费下载。另外为了帮助使用者更好更快的掌握MYCAT使用技巧和方法，开源社区特别撰写了 [《MYCAT权威指南》](http://item.taobao.com/item.htm?spm=a1z10.3-c.w4002-8793307026.9.fKuC9x&id=44263828402)。虽然MYCAT在很大程度上简化了数据库分片集群的开发与部署难度，但考虑到仍然存在一些特殊情况，比如项目工期紧，企业缺乏足够的时间和人力来全面研究MyCAT，因此为了更好的支持各类企业使用MYCAT实施项目，加速项目开发进度，如今MYCAT开源社区整合了一支强大的技术团队，以每月5000元的赞助合作费用，向社会提供公益性的技术支持服务，以此加速MYCAT开源项目的发展，并且成为Apache第一个来自中国的顶级开源项目。

[Mycat赞助商合同](http://www.mycat.io/document/Mycat%E8%B5%9E%E5%8A%A9%E5%95%86%E5%90%88%E5%90%8C.doc)

联系MYCAT官方合作请发邮件到：market@mycat.io

### Mycat高可用方案



### MYCAT典型案例



### 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：

  a.单纯的读写分离，此时配置最为简单，支持读写分离，主从切换

  b.分表分库，对于超过1〇〇〇万的表进行分片，最大支持1 〇〇〇亿的单表分片

  c.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

  d.报表系统，借助于Mycat的分表能力，处理大规模报表的统计

  e.代替Hbase,分析大数据

  f.作为海量数据实时查询的一种简单有效方案，比如 1〇〇亿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

—单纯的读写分离，此时配置最为简单，支持读写分离，主从切换分表分库，对于超过000万的表进行分片，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ycat可能是最简单有效的选择

### Mycat 介绍配置详解

http://blog.csdn.net/linuxlsq/article/details/52606225

**Mycat 前生今世**

1. 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个亿的数据库分片。---Mycat's Plan

**Mycat 简介**

#### 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Server，前端用户可以把它看做是一个数据库代理，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，其核心功能是分库分表，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发展到目前版本，已经不在是一个单纯的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，可以大幅度降低开发难度，提升开发速度，在测试阶段，可以将一表定义为任何一种Mycat支持的存储方式，比如MySQL的MyASM表、内存表、或者MongoDB、LeveIDB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。

#### 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：

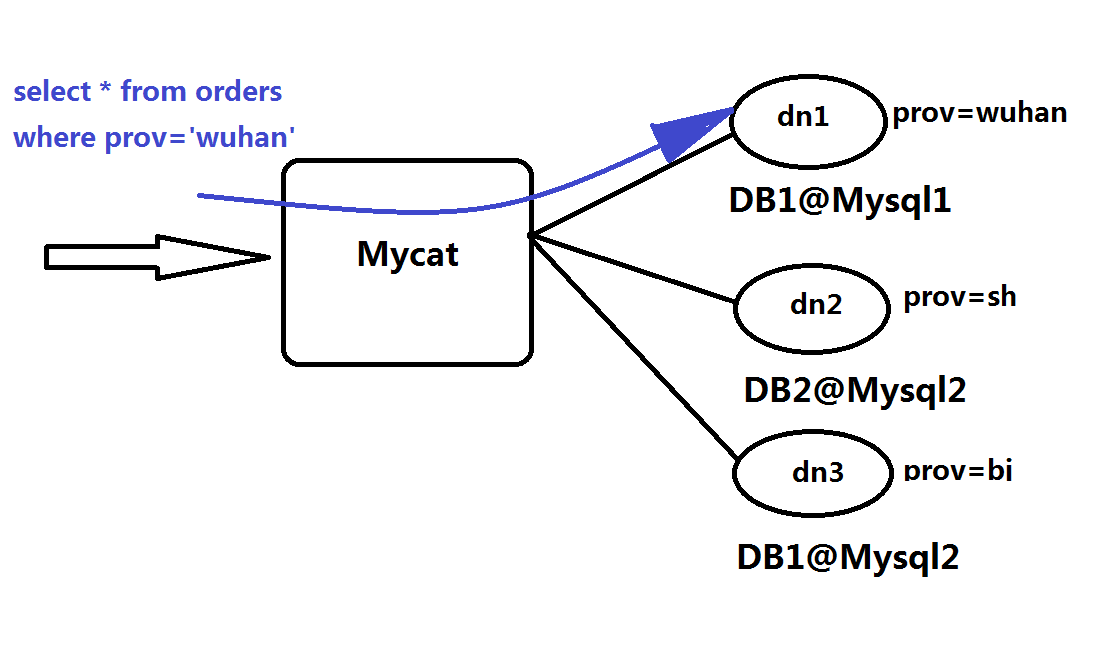
Mycat是一个强大的数据库中间件，不仅仅可以用作读写分离、以及分表分库、容灾备份，而且可以用于多租户应用开发、云平台基础设施、让你的架构具备很强的适应性和灵活性，借助于即将发布的Mycat智能优化模块，系统的数据访问瓶颈和热点一目了然，根据这些统计分析数据，你可以自动或手工调整后端存储，将不同的表映射到不同存储引擎上，而整个应用的代码一行也不用改变。

    当前是个大数据的时代，但究竟怎样规模的数据是和数据库系统呢？对此，国外有一个数据库领域的权威人士说了一个结论：**干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，更适合的是干亿以上的规模**，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！

#### 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rov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QL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elect \* from Orders where prov in (‘wuhan’,‘beijing’)，那么，SQL就会发给ySQL1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！

#### 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：**

  a.单纯的读写分离，此时配置最为简单，支持读写分离，主从切换

  b.分表分库，对于超过1〇〇〇万的表进行分片，最大支持1 〇〇〇亿的单表分片

  c.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

  d.报表系统，借助于Mycat的分表能力，处理大规模报表的统计

  e.代替Hbase,分析大数据

  f.作为海量数据实时查询的一种简单有效方案，比如 1〇〇亿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

—单纯的读写分离，此时配置最为简单，支持读写分离，主从切换分表分库，对于超过000万的表进行分片，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ycat可能是最简单有效的选择

#### 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。

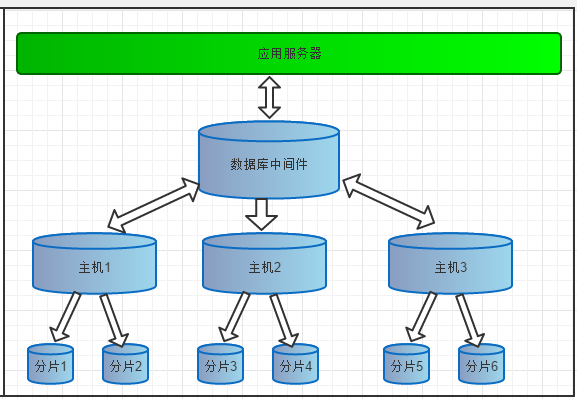
依托Mycat社区，聚集100个CXO级别的精英，众筹建设亲亲山庄，Mycat社区+亲亲山庄=中国最大IT O2O社区

**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。



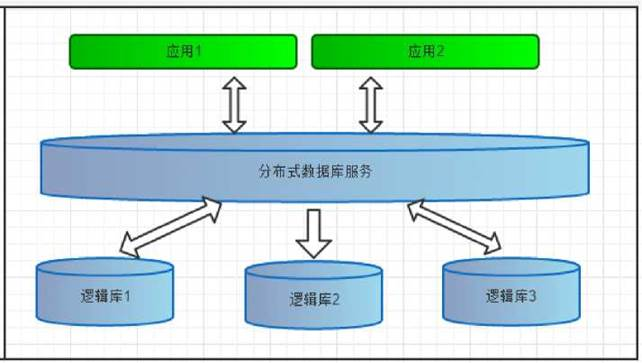
  如上图所表示，数据被分到多个分片数据库后，应用如果需要读取数据，就要需要处理多个数据源的数据。如果没有数据库中间件，那么应用将直接面对分片集群，数据源切换、事务处理、数据聚合都需要应用直接处理，原本该是专注于业务的应用，将会花大量的工作来处理分片后的问题，最重要的是每个应用处理将是完全的重复造轮子。

  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。

#### 逻辑库(schema)

    通常对实际应用来说，并不需要知道中间件的存在，开发人员只需要知道数据库的概念，所以数据库中间件可以被看做是一个或多个数据库集群构成的逻辑库。

    在云计算时代，数据库中间件可以以多租户的形式给一个或多个应用提供服务，每个应用访问的可能是一个独立或者是共享的物理库，常见的如阿里云数据库服务器RDS。



#### 逻辑表（table)

    既然有逻辑库，那么就会有逻辑表，分布式数据库中，对应用来说，读写数据的表就是逻辑表。逻辑表，可 以是数据切分后，分布在一个或多个分片库中，也可以不做数据切分，不分片，只有一个表构成。

#### 分片表

  分片表，是指那些原有的很大数据的表，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

  例如在mycat配置中的t\_node就属于分片表，数据按照规则被分到dn1,dn2两个分片节点(dataNode) 上。

1. <table name=nt\_noden primaryKey=nvidn autoincrement=ntruen dataNode=ndn1,dn2n rule=nrule1n />

#### 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。

  如下配置中t\_node ,只存在于分片节点（dataNode ) dn1上。

1. <table name=nt\_noden primaryKey=nvidn autoincrement=ntruen dataNode=ndn1" />

#### ER 表

    关系型数据库是基于实体关系模型（Entity-Relationship Model)之上，通过其描述了真实世界中事物与关 系，Mycat中的ER表即是来源于此。根据这一思路，提出了基于E-R关系的数捤分片策略，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，即子表依赖于父表，通过表分组（Table Group )保证数据Join不会跨库操作。

    表分组（Table Group )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重要一条规则。

#### 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，字典表具有以下几个特性：

1.变动不频繁

2.数据量总体变化不大

3.数据规模不大，很少有超过数十万条记录。

#### 分片节点(dataNode)

  数据切分后，一个大表被分到不同的分片数据库上面，每个表分片所在的数据库就是分片节点 (dataNode )。

节点主机(dataHost)

  数据切分后，每个分片节点（dataNode )不一定都会独占一台机器，同一机器上面可以有多个分片数据库， 这样一个或多个分片节点（dataNode )所在的机器就是节点主机（dataHost),为了规避单节点主机并发数限 制，尽量将读写压力高的分片节点（dataNode )均衡的放在不同的节点主机（dataHost )。

分片规则(rule)

  前面讲了数据切分，1个大表被分成若干个分片表，就需要一定的规则，这样按照某种业务规则把数据分到 某个分片的规则就是分片规则，数据切分选择合适的分片规则非常重要，将极大的避免后续数据处理的难度。

#### 多租户

多租户技术或称多重租赁技术，是一种软件架构技术，它是在探讨与实现如何于多用户的环境下共用相同的系统或程序组件，并且仍可确保各用户间数据的隔离性。在云计算时代，多租户技术在共用的数据中心以单一系统架构与服务提供多数客户端相同甚至可定制化的服务，并且仍然可以保障客户的数据隔离。目前各种各样的云计算服务就是这类技术范畴，例如阿里云数据库服务（RDS )、阿里云服务器(ECS)等等。

多租户在数据存储上存在三种主要的方案，分别是：

##### 1.1 独立数据库

这是第一种方案，即一个租户一个数据库，这种方案的用户数据隔离级别最高，安全性最好，但成本也高。

**优点：**

1、为不同的租户提供独立的数据库，有助于简化数据模型的扩展设计，满足不同租户的独特需求；

2、如果出现故障，恢复数据比较简单。

**缺点：**

1、增大了数据库的安装数量，随之带来维护成本和购置成本的增加。

    这种方案与传统的一个客户、一套数据、一套部署类似，差别只在于软件统一部署在运营商那里。如果面对的是银行、医院等需要非常高数据隔离级别的租户，可以选择这种模式，提高租用的定价。如果定价较低，产品 走低价路线，这种方案一般对运营商来说是无法承受的。

##### 1.2共享数据库，隔离数据架构

这是第二种方案，即多个或所有租户共享Database，但是每个租户一个Schema。

**优点：**

1、为安全性要求较高的租户提供了一定程度的逻辑数据隔离，并不是完全隔离；每个数据库可以支持更多的租户数量。

**缺点：**

1、如果出现故障，数据恢复比较困难，因为恢复数据库将牵扯到其它租户的数据

2、如果需要跨租户统计数据，存在一定困难。

#### 共享数据库，共享数据架构

这是第三种方案，即租户共享同一个Database、同一个Schema，但在表中通过TenantID区分租户的数据。这是共享程度最高、隔离级别最低的模式。

**优点：**

1、三种方案比较，第三种方案的维护和购置成本最低，允许每个数据库支持的租户数量最多。

**缺点：**

1、隔离级别最低，安全性最低，需要在设计开发时加大对安全的开发量；

2、数据备份和恢复最困难，需要逐表逐条备份和还原。

3、如果希望以最少的服务器为最多的租户提供服务，并且租户接受以牺牲隔离级别换取降低成本，这种方案最适合

官方网站：http://www.mycat.org.cn/

#### 何为数据切分？

简单来说，就是指通过某种特定的条件，将我们存放在同一个数据库中的数据分散存放到多个数据库（主 机）上面，以达到分散单台设备负载的效果。

数据的切分(Sharding )根据其切分规则的类型，可以分为两种切分模式。一种是按照不同的表（或者 Schema )来切分到不同的数据库(主机）之上，这种切可以称之为数据的垂直（纵向）切分；另外一种则是根据表中的数据的逻辑关系，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的水平（横向）切分。

**垂直切分的最大特点就是规则简单，实施也更为方便，尤其适合各业务之间的耦合度非常低，相互影响很小，业务逻辑非常清晰的系统**。在这种系统中，可以很容易做到将不同业务模块所使用的表分拆到不同的数据库中。根据不同的表来进行拆分，对应用程序的影响也更小，拆分规则也会比较简单清晰。

水平切分于垂直切分相比，相对来说稍微复杂一些。因为要将同一个表中的不同数据拆分到不同的据库中，对于应用程序来说，拆分规则本身就较根据表名来拆分更为复杂，后期的数据维护也会更为复杂一些。

#### 垂直切分

一个数据库由很多表的构成，每个表对应着不同的业务，垂直切分是指按照业务将表进行分类，分布到不同 的数据库上面，这样也就将数据或者说压力分担到不同的库上面，如下图：

|  |
| --- |
| http://www.abcdocker.com/wp-content/uploads/2016/07/afe3fb8d8fa5221dc5f54d3532c44f2b_1e623fee-b3e8-4427-b835-4484bf983dab.png |

    一个架构设计较好的应用系统，其总体功能肯定是由很多个功能模块所组成的，而每一个功能模块所需要的 数据对应到数据库中就是一个或者多个表。而在架构设计中，各个功能模块相互之间的交互点越统一越少，系统 的耦合度就越低，系统各个模块的维护性以及扩展性也就越好。这样的系统，实现数据的垂直切分也就越容易。

      但是往往系统之有些表难以做到完全的独立，存在这扩库join的情况，对于这类的表，就需要去做平衡，是数据库让步业务，共用一个数据源，还是分成多个库，业务之间通过接口来做调用。在系统初期，数据量比较少，或者资源有限的情况下，会选择共用数据源，但是当数据发展到了一定的规模，负载很大的情况，就需要必须去做分割。

  一般来讲业务存在着复杂join的场景是难以切分的，往往业务独立的易于切分。如何切分，切分到何种 程度是考验技术架构的一个难题。

##### 下面来分析下垂直切分的优缺点：

**优点：**

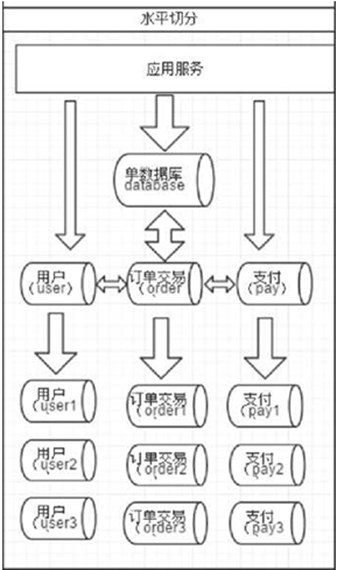
1. 拆分后业务清晰，拆分规则明确。
2. 系统之间整合或扩展容易。
3. 数据维护简单。

**缺点：**

1. 部分业务表无法join ,只能通过接口方式解决，提高了系统复杂度。
2. 受每种业务不同的限制存在单库性能瓶颈，不易扩展跟性能提高。
3. 事务处理复杂。由于垂直切分是按照业务的分类将表分散到不同的库，所以有些业务表会过于庞大，存在单库读写与存储瓶颈，所以就需要水平拆分来做解决。

##### 水平切分

    相对于垂直拆分，水平拆分不是将表做分类，而是按照某个字段的某种规则来分散到多个库之中，每个表中包含一部分数据。简单来说，我们可以将数据的水平切分理解为是按照数据行的切分，就是将表中的某些行切分 到一个数据库，而另外的某些行又切分到其他的数据库中，如图：



    拆分数据就需要定义分片规则。关系型数据库是行列的二维模型，拆分的第一原则是找到拆分维度。比如: 从会员的角度来分析，商户订单交易类系统中查询会员某天期某个订单，那么就需要按照会员结合日期来拆分，不同的数据按照会员ID做分组，这样所有的数据查询join都会在单库内解决；如果从商户的角度来讲，要查询某个商家某天所有的订单数，就需要按照商户ID做拆分；但是如果系统既想按会员拆分，又想按商家数据，则会有一定的困难。如何找到合适的分片规则需要综合考虑衡量。

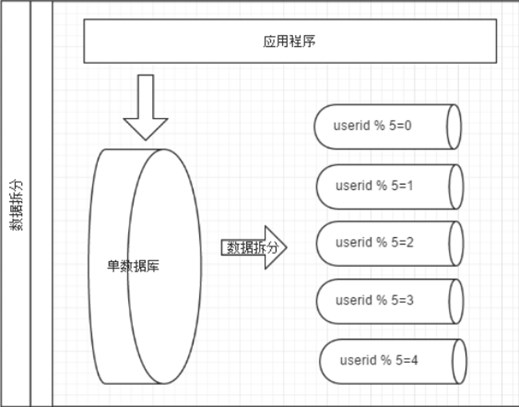
#### 几种典型的分片规则包括：

  a.按照用户ID求模，将数据分散到不同的数据库，具有相同数据用户的数据都被分散一个库中。

  b.按照日期，将不同月甚至日的数据分散到不同的库中。

  c.按照某个特定的字段求模，或者根据特定范围段分散到不同的库中。

如图，切分原则都是根据业务找到适合的切分规则分散到不同的库，下面用用户 ID 求模举例：



既然数据做了拆分有优点也就优缺点。

**优点：**

A.拆分规则抽象好，join 操作基本可以数据库做

B.不存在单库大数据，高并发的性能瓶颈。

C.应用端改造较少。

D.提高了系统的稳定性跟负载能力。

**缺点：**

a.拆分规则难以抽象。

b.分片亊务一致性难以解决。

c.数捤多次扩展难度跟维护量极大。

d.跨库 join 性能较差。

#### 前面讲了垂直切分跟水平切分的不同跟优缺点，会发现每种切分都有缺点，但共同的特点缺点有：

1.引入分布式亊务的问题。

2.跨节点 Join 的问题。

3.跨节点合并排序分页问题。

4.多数据源管理问题。

#### 针对数据源管理，目前主要有两种思路：

  A. 客户端模式，在每个应用程序模块中配置管理自己需要的一个（或者多个）数据源，直接访问各个数据库，在模块内完成数据的整合；

  B. 通过中间代理层来统一管理所有的数据源，后端数据库集群对前端应用程序透明；

可能 90%以上的人在面对上面这两种解决思路的时候都会倾向于选择第二种，尤其是系统不断变得庞大复杂的时候。确实，这是一个非常正确的选择，虽然短期内需要付出的成本可能会相对更大一些，但是对整个系统的扩展性来讲，是非常有帮助的数据切分的原则：

#### 数据切分的原则：

第一原则：能不切分尽量不要切分。

第二原则：如果要切分一定要选择合适的切分规则，提前规划好。

第三原则：数据切分尽量通过数据冗余或者表分组（Table Group）来降低跨库 Join 的可能。

第四原则：由于数据库中间件对数据 Join 实现的优劣难以把握，而且实现高性能难度极大，业务读取尽量少使用多表 Join。

**安装Mycat**

#### 安装Mycat

1. wget  https://github.com/MyCATApache/Mycat-download/raw/master/1.5-RELEASE/Mycat-server-1.5.1-RELEASE-20160622153300-linux.tar.gz
2. tar zxf Mycat-server-1.5.1-RELEASE-20160622153300-linux.tar.gz
3. mv mycat  /application/
4. echo "export PATH=/application/mycat/bin:$PATH">>/etc/profile
5. source /etc/profile
6. echo $PATH

##### 下载安装JDK

1. wget http://download.oracle.com/otn-pub/java/jdk/8u91-b14/jdk-8u91-linux-x64.rpm
2. rpm -ivh jdk-8u91-linux-x64.rpm

##### 配置环境变量

1. echo "export JAVA\_HOME=/usr/java/jdk1.8.0\_91/"   >>/etc/profile
2. echo "export PATH=$JAVA\_HOME/bin:/application/mycat/bin/:$PATH">>/etc/profile
3. echo "export CLASSPATH=$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar">>/etc/profile
4. source  /etc/profile
5. [root@Linuxdemo3 ~]# java -version
6. java version "1.8.0\_91"
7. Java(TM) SE RuntimeEnvironment(build 1.8.0\_91-b14)
8. JavaHotSpot(TM)64-BitServer VM (build 25.91-b14, mixed mode)

##### 创建管理用户

1. 主库上对rep和rep\_r用户授权如下：
2. 用户：rep  密码：oldboy 端口：3306
3. 权限：all
4. 命令：
5. grant replication slave on test.\* to 'rep'@'10.0.0.%' identified by 'oldboy';
6. grant replication slave on test.\* to 'rep\_r'@'10.0.0.%' identified by 'oldboy';
7. flush privileges;
8. 注：为了方便下面的主从切换，两个用户都授予了所有的权限，生产环境尽量不要这样子授权！

修改mycat配置文件

1. [root@Linuxdemo3 mycat]# pwd
2. /application/mycat
3. [root@Linuxdemo3 mycat]# ll
4. total 24
5. drwxr-xr-x 2 root root 4096Jul  403:13 bin
6. drwxrwxrwx 2 root root 4096Dec13  2015 catlet
7. drwxrwxrwx 2 root root 4096Jul  403:13 conf
8. drwxr-xr-x 2 root root 4096Jul  403:13 lib
9. drwxrwxrwx 2 root root 4096Dec13  2015 logs
10. -rwxrwxrwx 1 root root  219Jun2215:33 version.txt

**目录解释如下：**

1.bin程序目录，存放了 window版本和linux版本，除了提供封装服务的版本之外，也提供了 nowrap的 shell脚本命令，方便大家选择和修改，进入到bin目录：

2.Linux 下运行：./mycat console,首先要 chmod +x \*

注：mycat 支持的命令{ console | start | stop | restart | status | dump }

3.conf目录下存放配置文件，其中：

4.server.xm丨是Mycat服务器参数调整和用户授权的配置文件，

5.schema.xm丨是逻 辑库定义和表仌及分片定义的配置文件，

6.rule.xml是分片规则的配置文件，分片规则的具体一些参数信息单独存 放为文件，也在这个目录下，配置文件修改，需要重启Mycat或者通过9066端口 reload.

7.lib目录下主要存放mycat依赖的一些jar文件.

9.日志存放在logs/mycat.log中，每天一个文件，日志的配置是在conf/log4j.xml中，根据自己的需要，可 以调整输出级别为debug , debug级别下，会输出更多的信息，方便排查问题

**注意：Linux下部署安装MySQL ,默认不忽略表名大小写，需要手动到/etc/my.cnf下配置 lower\_case\_table\_names=1使Linux环境下MySQL忽略表名大小写，否则使用MyCAT的时候会提示找不到 表的**

##### 修改server.xml文件

1. [root@Linuxdemo3 conf]#  vim server.xml
2. </system>
3. <user name="rep">
4. <property name="password">oldboy</property>
5. <property name="schemas">test</property>
6. </user>
7. <user name="rep\_r">
8. <property name="password">oldboy</property>
9. <property name="schemas"> test</property>
10. <property name="readOnly">true</property>
11. </user>

**注意：**

1、这里配置的是可以连接主库的两个用户

用户：rep   密码：oldboy 给予此用户test数据库增删改查的权限。

用户：rep\_r 密码：oldboy 给予此用户test数据库读的权限。

2、这里的test，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。

##### 修改schema.xml文件

1. [root@Linuxdemo3 conf]#  vim schema.xml
2. <?xml version="1.0"?>
3. <!DOCTYPE mycat:schema SYSTEM "schema.dtd">
4. <mycat:schema xmlns:mycat="http://org.opencloudb/">
5. <schema name="test" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
6. </schema>
7. <dataNode name="dn1" dataHost="localhost1" database="test"/>
8. <dataNode name="dn2" dataHost="localhost1" database=" test "/>
9. <dataNode name="dn3" dataHost="localhost1" database=" test "/>
10. <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
11. writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
12. <heartbeat>select user()</heartbeat>
13. <writeHost host="hostM1" url="10.0.0.202:3306" user="rep"
14. password="oldboy">
15. </writeHost>
16. <readHost host="hostS1" url="10.0.0.202:3307" user="rep\_r"
17. password="oldboy"/>
18. </readHost>
19. </dataHost>
20. </mycat:schema>

(1)<schema name=”test” 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=”test” />

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)

1. <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
2. writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

##### 需要配置的位置：

1. 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writehost了 .

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(M1àS1,M2àS2，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)

1. <writeHost host="hostM1" url="10.0.0.202:3306" user="rep"
2. password="oldboy">
3. <!-- can have multi read hosts -->
4. <readHost host="hostS1" url="10.0.0.202:3306" user="rep\_r"
5. password="oldboy"/>

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。

##### 启动Mycat

1. [root@Linuxdemo3 conf]#/application/mycat/bin/mycat start
2. StartingMycat-server...
3. [root@Linuxdemo3 logs]#

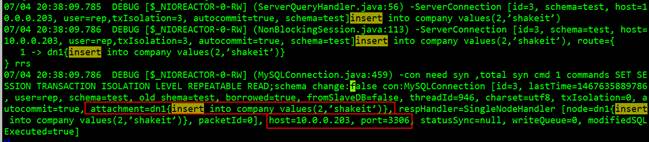
出现这个界面表示mycat已经成功启动了，如果出现错误的话请查看日志wrapper.log

**读写分离测试1、在客户端连接mysql主库服务器：**

1. mysql -urep -poldboy -h10.0.0.202 -P8066
2. mysql> explain create table company(id int not null primary key,name varchar(100));
3. +-----------+---------------------------------------------------------------------+
4. | DATA\_NODE | SQL                                                                 |
5. +-----------+---------------------------------------------------------------------+
6. | dn1       | create table company(id int not null primary key,name varchar(100)) |
7. | dn2       | create table company(id int not null primary key,name varchar(100)) |
8. | dn3       | create table company(id int not null primary key,name varchar(100)) |
9. +-----------+---------------------------------------------------------------------+
10. 3 rows in set (0.06 sec)
12. mysql> create table company(id int not null primary key,name varchar(100));
13. Query OK, 0 rows affected (1.82 sec)
15. mysql> explain insert into company(id, name) values (100, 'abc');
16. +-----------+---------------------------------------------------+
17. | DATA\_NODE | SQL                                               |
18. +-----------+---------------------------------------------------+
19. | dn1       | insert into company(id, name) values (100, 'abc') |
20. | dn2       | insert into company(id, name) values (100, 'abc') |
21. | dn3       | insert into company(id, name) values (100, 'abc') |
22. +-----------+---------------------------------------------------+
23. 3 rows in set (0.00 sec)
25. mysql> insert into company(id, name) values (100, 'abc');
26. Query OK, 1 row affected (0.24 sec)

##### 查看日志

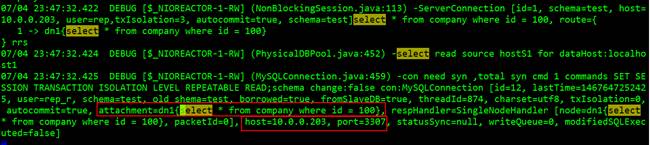
日志中有如下信息



##### 分别在主从库中执行以下命令：

1. mysql> select \* from company where id =100;
2. +-----+------+
3. | id  | name |
4. +-----+------+
5. |100| abc  |
6. +-----+------+
7. 1 row in set(0.00 sec)

日志中有如下



在从库中执行insert命令会得到以下错误：

1. mysql> insert into company(id, name) values (4,'abcwq');
2. ERROR 1495(HY000):User readonly
3. #从库只读

从上面的测试中可以看出来，向数据库中读写数据，走的是不同的数据库，证明，读写分离测试成功！

##### 主从切换测试

##### 将schema.xml文件做如下修改：

1. [root@Linuxdemo3 conf]#  vim schema.xml
2. <?xml version="1.0"?>
3. <!DOCTYPE mycat:schema SYSTEM "schema.dtd">
4. <mycat:schema xmlns:mycat="http://org.opencloudb/">
5. <schema name="test" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
6. </schema>
7. <dataNode name="dn1" dataHost="localhost1" database="test"/>
8. <dataNode name="dn2" dataHost="localhost1" database=" test "/>
9. <dataNode name="dn3" dataHost="localhost1" database=" test "/>
10. <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
11. writeType="0" dbType="mysql" dbDriver="native" switchType="2" slaveThreshold="100">
12. <heartbeat>show slave status</heartbeat>
13. <writeHost host="hostM1" url="10.0.0.202:3306" user="rep"
14. password="oldboy">
15. </writeHost>
16. <readHost host="hostS1" url="10.0.0.202:3307" user="rep\_r"
17. password="oldboy"/>
18. </readHost>
19. <writeHost host="hostM2" url="10.0.0.202:3307" user="rep"
20. password="oldboy">
21. </writeHost>
22. </dataHost>
23. </mycat:schema>

##### 重启mycat服务

1. [root@Linuxdemo2 conf]#/application/mycat/bin/mycat restart
2. StoppingMycat-server...
3. StoppedMycat-server.
4. StartingMycat-server...

此时，停止主库，再进行测试，会发现mycat服务依然可以正常读写数据库。mysql已切换到从库。

1. [root@Linuxdemo2 conf]#/data/3306/mysql stop

##### Mycat-Web

    MyCAT-WEB就是基于mycat的一个性能监控工具，方便大家更有效的使用mycat管理mycat监控mycat，让大家的mycat工作更加高效。mycat-web的运行依赖 zookpeer 所以事先要安装好。

    Zookeeper 作为一个分布式的服务框架，主要用来解决分布式集群中应用系统的一致性问题, 它能提供基于类似于文件系统的目录节点树方式的数据存储,但是 Zookeeper 并不是用来专门存储数据的，它的作用主要是用来维护和监控你存储的数据的状态变化，通过监控这些数据状态的变化，从而可以达到基于数据的集群管理。

    ZooKeeper是一个开源的分布式的，为分布式应用提供协调服务的Apache项目。ZooKeeper提供一个简单的原语集合，以便于分布式应用可以在它之上构建更高层次的同步服务。ZooKeeper的设计非常易于编程，它使用的是类似于文件系统那样的树形数据结构。

##### 安装zookpeer

1. wget  http://mirrors.hust.edu.cn/apache/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz
2. tar -xzvf  zookeeper-3.4.6.tar.gz
3. cd zookeeper-3.4.6/conf
4. cp zoo\_sample.cfg zoo.cfg
5. cd zookeeper-3.4.6/bin
6. ./zkServer.sh start
7. 出现一下信息说明启动成功
8. JMX enabled by default
9. Using config:/usr1/zookeeper/bin/../conf/zoo.cfg
10. Starting zookeeper ... STARTED

##### 安装mycat-web

1. wget https://github.com/MyCATApache/Mycat-download/raw/master/mycat-web-1.0/Mycat-web-1.0-SNAPSHOT-20160617163048-linux.tar.gz
2. tar zxf Mycat-web-1.0-SNAPSHOT-20160617163048-linux.tar.gz
3. cd mycat-web/mycat-web/WEB-INF/classes
4. vim mycat.properties
5. zookeeper=127.0.0.1:2181
6. sqlonline.server=10.0.0.202
7. cd mycat-web
8. #将start.sh文件中的JVM调整到合适的大小
9. ./start.sh &
10. #8082端口是web端口
11. 访问10.0.0.202:8082/mycat即可进入web页面

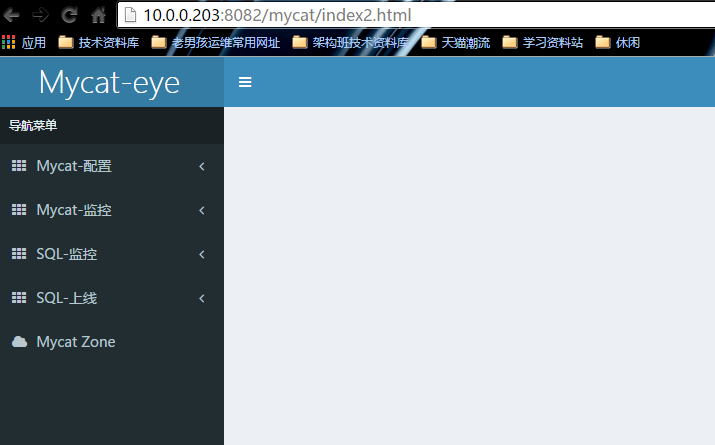
配置Mycat(此步骤可省略)

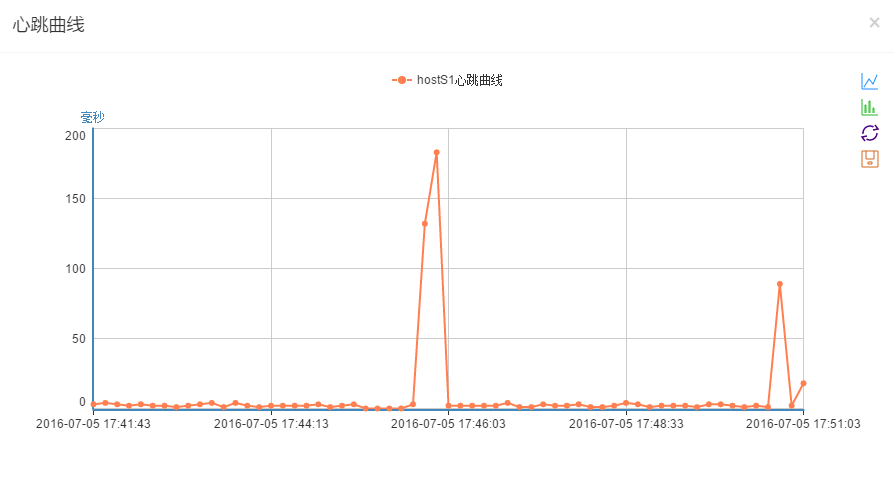
1. cd /application/mycat/bin

另外有四个相对重要的文件

        文件                                     用处

1. /application/mycat/bin/xml\_to\_yaml.sh 根据mycat配置生成 zookeeper yaml 配置工具
2. /application/mycat/bin/init\_zk\_data.sh 将生成的yaml配置文件导入远程zookeeper
3. /application/mycat/conf/myid.properties zookeeper 路径配置参数信息
4. /application/mycat/conf/zk-create.yaml 自动生成的zk配置
5. 1）确认安装好zookeeper 环境
6. 2）启动mycat服务，确认本地服务一切正常。
7. 3）执行xml\_to\_yaml.sh脚本（确认本地配置过 JAVA\_HOME 和 MYCAT\_HOME环境变量，）
8. [root@Linuxdemo3 bin]#./xml\_to\_yaml.sh
9. 4)执行 init\_zk\_data.sh 脚本,注意下默认情况下脚本会将zookeeper定位在同一台服务器上面，如有需要可以调整下 ZK 的 IP : PORT
10. [root@Linuxdemo3 bin]#./init\_zk\_data.sh





##### 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： 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：

1. mysql -urep -poldboy  -h10.0.0.202-P9066 -Dtest

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：

1. mysql> show @@help;
2. +--------------------------------------+-----------------------------------+
3. | STATEMENT                            | DESCRIPTION                       |
4. +--------------------------------------+-----------------------------------+
5. | clear @@slowwhere datanode =?      |Clear slow sql by datanode        |
6. | clear @@slowwhere schema =?        |Clear slow sql by schema          |
7. | kill @@connection id1,id2,...        |Kill the specified connections    |
8. | offline                              |ChangeMyCat status to OFF        |
9. | online                               |ChangeMyCat status to ON         |
10. | reload @@config                      |Reload basic config from file     |
11. | reload @@config\_all                  |Reload all config from file       |
12. | reload @@route                       |Reload route config from file     |
13. | reload @@user                        |Reload user config from file      |
14. | rollback @@config                    |Rollback all config from memory   |
15. | rollback @@route                     |Rollback route config from memory |
16. | rollback @@user                      |Rollback user config from memory  |
17. | show @@backend                       |Report backend connection status  |
18. | show @@cache                         |Report system cache usage         |
19. | show @@command                       |Report commands status            |
20. | show @@connection                    |Report connection status          |
21. | show @@connection.sql                |Report connection sql             |
22. | show @@database                      |Report databases                  |
23. | show @@datanode                      |Report dataNodes                  |
24. | show @@datanodewhere schema =?     |Report dataNodes                  |
25. | show @@datasource                    |Report dataSources                |
26. | show @@datasourcewhere dataNode =?|Report dataSources                |
27. | show @@heartbeat                     |Report heartbeat status           |
28. | show @@parser                        |Report parser status              |
29. | show @@processor                     |Report processor status           |
30. | show @@router                        |Report router status              |
31. | show @@server                        |Report server status              |
32. | show @@session                       |Report front session details      |
33. | show @@slowwhere datanode =?       |Report datanode slow sql          |
34. | show @@slowwhere schema =?         |Report schema slow sql            |
35. | show @@sqlwhere id =?              |Report specify SQL                |
36. | show @@sql.detail where id =?       |Report execute detail status      |
37. | show @@sql.execute                   |Report execute status             |
38. | show @@sql.slow                      |Report slow SQL                   |
39. | show @@threadpool                    |Report threadPool status          |
40. | show @@time.current                  |Report current timestamp          |
41. | show @@time.startup                  |Report startup timestamp          |
42. | show @@version                       |ReportMycatServer version       |
43. | stop @@heartbeat name:time           |Pause dataNode heartbeat          |
44. |switch@@datasource name:index       |Switch dataSource

###### 2、更新配置文件

1. mysql> reload @@config;
2. Query OK,1 row affected (0.13 sec)
3. 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

1. mysql> show @@database;
2. +----------+
3. | DATABASE |
4. +----------+
5. | oldboy   |
6. +----------+
7. 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

1. mysql> show @@datanode;
2. +------+-------------------+-------+-------+--------+------+------+---------+------------+----------+---------+---------------+
3. | NAME | DATHOST           | INDEX | TYPE  | ACTIVE | IDLE | SIZE | EXECUTE | TOTAL\_TIME |MAX\_TIME | MAX\_SQL | RECOVERY\_TIME |
4. +------+-------------------+-------+-------+--------+------+------+---------+------------+----------+---------+---------------+
5. | dn1  | localhost1/oldboy |     0| mysql |      0|   10|1000|     457|          0|        0|       0|            -1|
6. | dn2  | localhost1/oldboy |     0| mysql |      0|   10|1000|     457|          0|        0|       0|            -1|
7. | dn3  | localhost1/oldboy |     0| mysql |      0|   10|1000|     457|          0|        0|       0|            -1|
8. +------+-------------------+-------+-------+--------+------+------+---------+------------+----------+---------+---------------+
9. 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

1. mysql> show @@heartbeat;
2. +--------+-------+-------------+------+---------+-------+--------+---------+--------------+---------------------+-------+
3. | NAME   | TYPE  | HOST        | PORT | RS\_CODE | RETRY | STATUS | TIMEOUT | EXECUTE\_TIME |LAST\_ACTIVE\_TIME    | STOP  |
4. +--------+-------+-------------+------+---------+-------+--------+---------+--------------+---------------------+-------+
5. | hostM1 | mysql |10.0.0.202|3306|       1|     0| idle   |       0|1,1,1        |2015-12-2921:39:40|false|
6. | hostS1 | mysql |10.0.0.202|3307|       1|     0| idle   |       0|3,3,3        |2015-12-2921:39:40|false|
7. +--------+-------+-------------+------+---------+-------+--------+---------+--------------+---------------------+-------+
8. 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

1. mysql> show @@version;
2. +--------------------------------------+
3. | VERSION                              |
4. +--------------------------------------+
5. |5.5.8-mycat-1.4-alpha-20150520235658|
6. +--------------------------------------+
7. 1 row in set(0.00 sec)

###### 7、显示mycat前端连接状态

1. mysql> show @@connection;
2. +------------+------+-----------+------+------------+--------+---------+--------+---------+---------------+-------------+------------+---------+------------+
3. | PROCESSOR  | ID   | HOST      | PORT | LOCAL\_PORT | SCHEMA | CHARSET | NET\_IN | NET\_OUT |ALIVE\_TIME(S)| RECV\_BUFFER | SEND\_QUEUE | txlevel | autocommit |
4. +------------+------+-----------+------+------------+--------+---------+--------+---------+---------------+-------------+------------+---------+------------+
5. |Processor0|    6|127.0.0.1|9066|      46490| oldboy | utf8:33|    281|    6164|          1008|       4096|          0|         |            |
6. +------------+------+-----------+------+------------+--------+---------+--------+---------+---------------+-------------+------------+---------+------------+
7. 1 row in set(0.00 sec)

###### 8、显示mycat后端连接状态

1. mysql> show @@backend;
2. +------------+------+---------+-------------+------+--------+--------+---------+------+--------+----------+------------+--------+---------+---------+------------+
3. | processor  | id   | mysqlId | host        | port | l\_port | net\_in | net\_out | life | closed |borrowed | SEND\_QUEUE | schema | charset | txlevel | autocommit |
4. +------------+------+---------+-------------+------+--------+--------+---------+------+--------+----------+------------+--------+---------+---------+------------+
5. |Processor0|    1|      30|10.0.0.202|3306|  14881|   3554|    1068|5041|false  |false    |          0| oldboy | utf8:33|3       |true       |
6. |Processor0|    2|      32|10.0.0.202|3306|  14883|   3554|    1068|5041|false  |false    |          0| oldboy | utf8:33|3       |true       |
7. |Processor0|    3|      24|10.0.0.202|3306|  14875|   3515|    1068|5041|false  |false    |          0| oldboy | utf8:33|3       |true       |
8. |Processor0|    4|      28|10.0.0.202|3306|  14879|   3561|     986|5041|false  |false    |          0| oldboy | utf8:33|0       |true       |

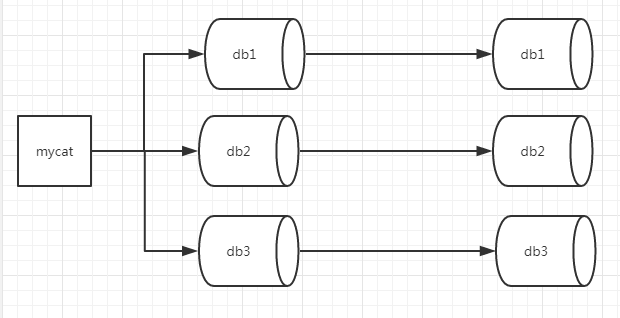
###### 9、显示数据源

1. mysql> show @@datasource;
2. +----------+--------+-------+-------------+------+------+--------+------+------+---------+
3. | DATANODE | NAME   | TYPE  | HOST        | PORT | W/R  | ACTIVE | IDLE | SIZE | EXECUTE |
4. +----------+--------+-------+-------------+------+------+--------+------+------+---------+
5. | dn1      | hostM1 | mysql |10.0.0.202|3306| W    |      0|   10|1000|     525|
6. | dn1      | hostS1 | mysql |10.0.0.202|3306| R    |      0|    8|1000|     522|
7. | dn3      | hostM1 | mysql |10.0.0.202|3306| W    |      0|   10|1000|     525|
8. | dn3      | hostS1 | mysql |10.0.0.202|3306| R    |      0|    8|1000|     522|
9. | dn2      | hostM1 | mysql |10.0.0.202|3306| W    |      0|   10|1000|     525|
10. | dn2      | hostS1 | mysql |10.0.0.202|3306| R    |      0|    8|1000|     522|
11. +----------+--------+-------+-------------+------+------+--------+------+------+---------+
12. 6 rows in set(0.01 sec)

特别说明：

# 多租户解决方案

## 1.注册拦截 指定数据库。



　　优点:支持进一步分片

　　缺点:schema配置繁琐

注解式  /\*!mycat:schema=[schemaName] \*/   注意:这在navicat 里面是会报错的,请用命令行登陆mycat 来测试

mysql> explain /\*!mycat:schema=TESTDB \*/ select \* from users;

可以在每个sql语句前面添加此注解，Mybatis 可以重写 MappedStatement  的 getBoundSql 来添加。

Mycat schema.xml 中添加很冗长的 配置。

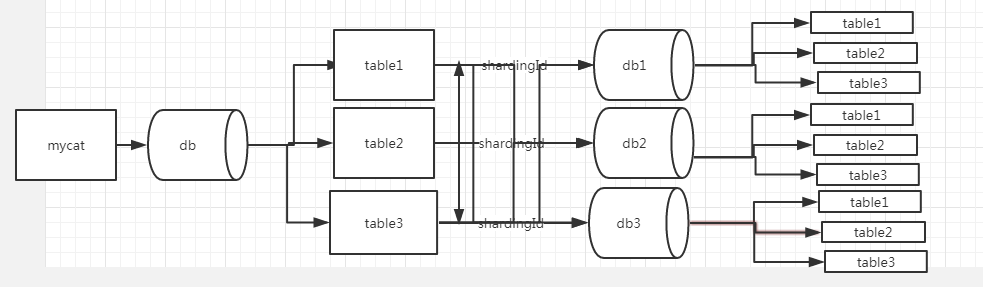
[复制代码](javascript:void(0);)

<mycat:schema xmlns:mycat="http://io.mycat/">

<schema name="USER1" checkSQLschema="false" sqlMaxLimit="10000">  
　　　　.....  
　　</schema>  
  
 <schema name="USER2" checkSQLschema="false" sqlMaxLimit="10000">

　　......　　  
　　</schema>  
以下省略无数个 schema

## 2.注册拦截 指定分片字段。



　优点:schema.xml 配置相对简洁

　　缺点:不可进一步分片

explain /\*!mycat:schema=TESTDB \*/ /\*!mycat:dataNode=dn1 \*/ select \* from users;

1. 将企业表作为全局表。因为企业ID 是作为shardingId 冗余在每个DB中。在schema.xml 中配置。

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

1. 配置表根据分片存储在各个分片中。例如User表

1. schema.xml

<table name="users" dataNode="dn1,dn2,dn3" rule="sharding-by-intfile-comp" />

2.rule.xml

<tableRule name="sharding-by-intfile-comp">

<rule>

<columns>comId</columns>

<algorithm>hash-int-comId</algorithm>

</rule>

</tableRule>

<function name="hash-int-comId"

class="io.mycat.route.function.PartitionByFileMap">

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

<property name="type">0</property>

</function>

3. partition-hash-int-comId.txt

1=0

2=1

3=2

3) 需要在每个表中加入shardingID 以免数据错乱。（繁的地方）

# [Mycat水平拆分之十种分片规则](http://www.cnblogs.com/756623607-zhang/p/6656022.html)

**水平切分分片实现**

配置schema.xml  在同一个mysql数据库中，创建了三个数据库 testdb1，testdb2，testdb3。并在每个库中都创建了user表

<?xml version="1.0"?>

<!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://io.mycat/">

   <schema name="**testdb**" checkSQLschema="false" sqlMaxLimit="100”  >

       <!——指定rule 分片规则-->

      <table name="user" dataNode="dn1,dn2,dn3" rule="sharding-by-intfile" />

   </schema>

    <dataNode name="dn1" dataHost="host" database="testdb1" />

    <dataNode name="dn2" dataHost="host" database="testdb2" />

    <dataNode name="dn3" dataHost="host" database="testdb3" />

    <dataHost name="host" maxCon="1000" minCon="10" balance="0"

       writeType="0" dbType="mysql" dbDriver="native">

       <heartbeat>select 1</heartbeat>

       <writeHost host="hostM1" url="localhost:3306" user="root" password="123" />

    </dataHost>

</mycat:schema>

配置server.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mycat:server SYSTEM "server.dtd">

<mycat:server xmlns:mycat="http://io.mycat/">

     <system>

          <property name="defaultSqlParser">druidparser</property>

     </system>

     <user name="mycat">

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

          <property name="schemas">**testdb**</property>

     </user>

</mycat:server>

配置rule.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mycat:rule SYSTEM "rule.dtd">

<mycat:rule xmlns:mycat="http://io.mycat/“>

    <tableRule name="sharding-by-intfile">

          <rule>

               <columns>sharding\_id</columns>

               <algorithm>hash-int</algorithm>

          </rule>

     </tableRule>

    <function name="hash-int"

              class="io.mycat.route.function.PartitionByFileMap">

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

     </function>

</mycat:rule>

**常用的分片规则：总共十个（基本够用）**

## 一、枚举法

<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">partition-hash-int.txt</property>

    <property name="type">0</property>

    <property name="defaultNode">0</property>

  </function>

partition-hash-int.txt 配置：

10000=0

10010=1

上面columns 标识将要分片的表字段，algorithm 分片函数，

其中分片函数配置中，mapFile标识配置文件名称，type默认值为0，0表示Integer，非零表示String，

所有的节点配置都是从0开始，及0代表节点1

/\*\*  
\*  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,结点为指定的值  
\*   
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      
\*/

## 二、固定分片hash算法

<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：

上面columns 标识将要分片的表字段，algorithm 分片函数，

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：

@Test  
public void testPartition() {  
 // 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%。（故本例非均匀分区）  
 // |<---------------------1024------------------------>|  
 // |<----256--->|<----256--->|<----------512---------->|  
 // | partition0 | partition1 | partition2 |  
 // | 共2份,故count[0]=2 | 共1份，故count[1]=1 |  
 int[] count = new int[] { 2, 1 };  
 int[] length = new int[] { 256, 512 };  
 PartitionUtil pu = new PartitionUtil(count, length);  
  
 // 下面代码演示分别以offerId字段或memberId字段根据上述分区策略拆分的分配结果  
 int DEFAULT\_STR\_HEAD\_LEN = 8; // cobar默认会配置为此值  
 long offerId = 12345;  
 String memberId = "qiushuo";  
  
 // 若根据offerId分配，partNo1将等于0，即按照上述分区策略，offerId为12345时将会被分配到partition0中  
 int partNo1 = pu.partition(offerId);  
  
 // 若根据memberId分配，partNo2将等于2，即按照上述分区策略，memberId为qiushuo时将会被分到partition2中  
 int partNo2 = pu.partition(memberId, 0, DEFAULT\_STR\_HEAD\_LEN);  
  
 Assert.assertEquals(0, partNo1);  
 Assert.assertEquals(2, partNo2);  
}

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

**<function name="func1" class="org.opencloudb.route.function.PartitionByLong">**

**<property name="partitionCount">4</property>**

**<property name="partitionLength">256</property>**

**</function>**

## 三、范围约定

**<tableRule name="auto-sharding-long">**

**<rule>**

**<columns>user\_id</columns>**

**<algorithm>rang-long</algorithm>**

**</rule>**

**</tableRule>**

**<function name="rang-long" class="io.mycat.route.function.AutoPartitionByLong">**

**<property name="mapFile">autopartition-long.txt</property>**

**</function>**

**# range start-end ,data node index**

**# K=1000,M=10000.**

**0-500M=0**

**500M-1000M=1**

**1000M-1500M=2**

**或**

**0-10000000=0**

**10000001-20000000=1**

配置说明：

上面columns 标识将要分片的表字段，algorithm 分片函数，

rang-long 函数中mapFile代表配置文件路径

所有的节点配置都是从0开始，及0代表节点1，此配置非常简单，即预先制定可能的id范围到某个分片

## 四、求模法

**<tableRule name="mod-long">**

**<rule>**

**<columns>user\_id</columns>**

**<algorithm>mod-long</algorithm>**

**</rule>**

**</tableRule>**

**<function name="mod-long" class="io.mycat.route.function.PartitionByMod">**

**<!-- how many data nodes  -->**

**<property name="count">3</property>**

**</function>**

配置说明：

上面columns 标识将要分片的表字段，algorithm 分片函数，

此种配置非常明确即根据id与count（你的结点数）进行求模预算，相比方式1，此种在批量插入时需要切换数据源，id不连续

## 五、日期列分区法

**<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">2014-01-01</property>**

**<property name="sPartionDay">10</property>**

**</function>**

配置说明：

上面columns 标识将要分片的表字段，algorithm 分片函数，

配置中配置了开始日期，分区天数，即默认从开始日期算起，分隔10天一个分区

还有一切特性请看源码

Assert.assertEquals(true, 0 == partition.calculate("2014-01-01"));  
Assert.assertEquals(true, 0 == partition.calculate("2014-01-10"));  
Assert.assertEquals(true, 1 == partition.calculate("2014-01-11"));  
Assert.assertEquals(true, 12 == partition.calculate("2014-05-01"));

## 六、通配取模

<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

# id partition range start-end ,data node index

###### first host configuration

1-32=0

33-64=1

65-96=2

97-128=3

######## second host configuration

129-160=4

161-192=5

193-224=6

225-256=7

0-0=7

配置说明：

上面columns 标识将要分片的表字段，algorithm 分片函数，patternValue 即求模基数，defaoultNode 默认节点，如果不配置了默认，则默认是0即第一个结点

mapFile 配置文件路径

配置文件中，1-32 即代表id%256后分布的范围，如果在1-32则在分区1，其他类推，如果id非数字数据，则会分配在defaoultNode 默认节点

String idVal = "0";

Assert.assertEquals(true, 7 == autoPartition.calculate(idVal));  
idVal = "45a";  
Assert.assertEquals(true, 2 == autoPartition.calculate(idVal));

## 七、ASCII码求模通配

<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.txt

# range start-end ,data node index

# ASCII

# 48-57=0-9

# 64、65-90=@、A-Z

# 97-122=a-z

###### first host configuration

1-4=0

5-8=1

9-12=2

13-16=3

###### second host configuration

17-20=4

21-24=5

25-28=6

29-32=7

0-0=7

配置说明：

上面columns 标识将要分片的表字段，algorithm 分片函数，patternValue 即求模基数，prefixLength ASCII 截取的位数

mapFile 配置文件路径

配置文件中，1-32 即代表id%256后分布的范围，如果在1-32则在分区1，其他类推

此种方式类似方式6只不过采取的是将列种获取前prefixLength位列所有ASCII码的和进行求模sum%patternValue ,获取的值，在通配范围内的

即 分片数，

/\*\*  
\* ASCII编码：  
\* 48-57=0-9阿拉伯数字  
\* 64、65-90=@、A-Z  
\* 97-122=a-z  
\*  
\*/

如

String idVal="gf89f9a";  
Assert.assertEquals(true, 0==autoPartition.calculate(idVal));  
  
idVal="8df99a";  
Assert.assertEquals(true, 4==autoPartition.calculate(idVal));  
  
idVal="8dhdf99a";  
Assert.assertEquals(true, 3==autoPartition.calculate(idVal));

## 八、编程指定

<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：

上面columns 标识将要分片的表字段，algorithm 分片函数

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

## 九、字符串拆分hash解析

**<tableRule name="sharding-by-stringhash">**

**<rule>**

**<columns>user\_id</columns>**

**<algorithm>sharding-by-stringhash</algorithm>**

**</rule>**

**</tableRule>**

**<function name="sharding-by-substring" class="io.mycat.route.function.PartitionByString">**

**<property name=length>512</property> <!-- zero-based -->**

**<property name="count">2</property>**

**<property name="hashSlice">0:2</property>**

**</function>**

配置说明：

上面columns 标识将要分片的表字段，algorithm 分片函数

函数中length代表字符串hash求模基数，count分区数，hashSlice hash预算位

即根据子字符串 hash运算

hashSlice ： 0 means str.length(), -1 means str.length()-1

/\*\*  
     \* "2" -&gt; (0,2)<br/>  
     \* "1:2" -&gt; (1,2)<br/>  
     \* "1:" -&gt; (1,0)<br/>  
     \* "-1:" -&gt; (-1,0)<br/>  
     \* ":-1" -&gt; (0,-1)<br/>  
     \* ":" -&gt; (0,0)<br/>  
     \*/

public class PartitionByStringTest {  
  
 @Test  
 public void test() {  
 PartitionByString rule = new PartitionByString();  
 String idVal=null;  
 rule.setPartitionLength("512");  
 rule.setPartitionCount("2");  
 rule.init();  
 rule.setHashSlice("0:2");  
// idVal = "0";  
// Assert.assertEquals(true, 0 == rule.calculate(idVal));  
// idVal = "45a";  
// Assert.assertEquals(true, 1 == rule.calculate(idVal));  
  
   
   
 //last 4  
 rule = new PartitionByString();  
 rule.setPartitionLength("512");  
 rule.setPartitionCount("2");  
 rule.init();  
 //last 4 characters  
 rule.setHashSlice("-4:0");  
 idVal = "aaaabbb0000";  
 Assert.assertEquals(true, 0 == rule.calculate(idVal));  
 idVal = "aaaabbb2359";  
 Assert.assertEquals(true, 0 == rule.calculate(idVal));  
 }

## 十、一致性hash

<tableRule name="sharding-by-murmur">

      <rule>

        <columns>user\_id</columns>

        <algorithm>murmur</algorithm>

      </rule>

   </tableRule>

<function name="murmur" class="io.mycat.route.function.PartitionByMurmurHash">

      <property name="seed">0</property><!-- 默认是0-->

      <property name="count">2</property><!-- 要分片的数据库节点数量，必须指定，否则没法分片—>

      <property name="virtualBucketTimes">160</property><!-- 一个实际的数据库节点被映射为这么多虚拟节点，默认是160倍，也就是虚拟节点数是物理节点数的160倍-->

      <!--

      <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>

**一致性hash预算有效解决了分布式数据的扩容问题，前1-9中id规则都多少存在数据扩容难题，而10规则解决了数据扩容难点**

# [MyCat集群部署（HAProxy + MyCat）](http://www.iteye.com/topic/1146264)

一、本节课程的依赖课程

《高可用架构篇--第13节--MySQL源码编译安装（CentOS-6.6+MySQL-5.6）》

《高可用架构篇--第14节--MySQL主从复制的配置（CentOS-6.6+MySQL-5.6）》

《高可用架构篇--第15节--MyCat在MySQL主从复制基础上实现读写分离》

二、软件版本

操作系统：CentOS-6.6-x86\_64

JDK版本：jdk1.7.0\_72

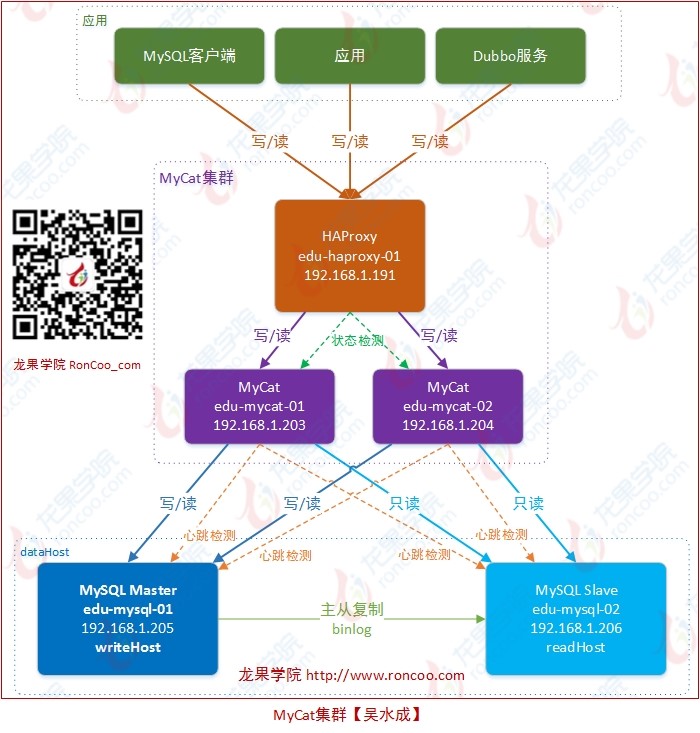
HAProxy版本：haproxy-1.5.16.tar.gz

MyCat版本：Mycat-server-1.4-release-20151019230038-linux.tar.gz

MySQL版本：mysql-5.6.26.tar.gz

三、部署环境规划



四、MyCat集群部署架构图如下：                                               


图解说明：

HAProxy负责将请求分发到MyCat上，起到负载均衡的作用，同时HAProxy也能检测到MyCat是否存活，HAProxy只会将请求转发到存活的MyCat上。如果一台MyCat服务器宕机，HAPorxy转发请求时不会转发到宕机的MyCat上，所以MyCat依然可用。

五、MyCat节点2的部署

MyCat主机2（edu-mycat-02，192.168.1.204）请参考《高可用架构篇--第15节--MyCat在MySQL主从复制基础上实现读写分离》进行对等部署和做相应配置。

注意：edu-mycat-01（192.168.1.203）和 edu-mycat-02（192.168.1.204） 中都要加上（或更新）主机名映射配置。

# vi /etc/hosts

192.168.1.203 edu-mycat-01

192.168.1.204 edu-mycat-02

192.168.1.205 edu-mysql-01

192.168.1.206 edu-mysql-02

六、配置MyCat状态检查服务（在MyCat节点主机上配置）

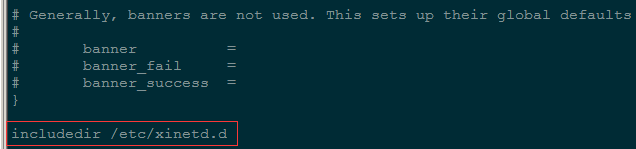
MyCat服务主机（edu-mycat-01、edu-mycat-02）上需要增加mycat服务的状态检测脚本，并开放相应的检测端口，以提供给HAProxy对MyCat的服务状态进行检测判断。可以使用xinetd来实现，通过xinetd，HAProxy可以用httpchk来检测MyCat的存活状态。（xinetd即extended internet daemon，xinetd是新一代的网络守护进程服务程序，又叫超级Internet服务器。经常用来管理多种轻量级Internet服务。xinetd提供类似于inetd+tcp\_wrapper的功能，但是更加强大和安全。xinetd为linux系统的基础服务）

1、如果xinetd还没有安装，可使用如下命令安装：

# yum install xinetd

2、检查/etc/xinetd.conf的末尾是否有 includedir /etc/xinetd.d ，没有就加上

# vi /etc/xinetd.conf



3、检查 /etc/xinetd.d 目录是否存在，不存在刚创建

# mkdir /etc/xinetd.d/

4、增加MyCat存活状态检测服务配置

# touch /etc/xinetd.d/mycat\_status

# vi /etc/xinetd.d/mycat\_status

增加以下内容：

service mycat\_status

{

        flags           = REUSE

## 使用该标记的socket\_type为stream，需要设置wait为no

        socket\_type     = stream ## 封包处理方式，Stream为TCP数据包

        port            = 48700  ## 服务监听端口

        wait            = no     ## 表示不需等待，即服务将以多线程的方式运行

        user            = root   ## 执行此服务进程的用户

        server          =/usr/local/bin/mycat\_status    ## 需要启动的服务脚本

        log\_on\_failure  += USERID    ## 登录失败记录的内容

        disable         = no     ## 要启动服务，将此参数设置为no

}

5、添加 /usr/local/bin/mycat\_status 服务脚本

# touch /usr/local/bin/mycat\_status

# vi /usr/local/bin/mycat\_status

增加以下内容：

#!/bin/bash

#/usr/local/bin/mycat\_status.sh

# This script checks if a mycat server is healthy running on localhost.

# It will return:

# "HTTP/1.x 200 OK\r" (if mycat is running smoothly)

# "HTTP/1.x 503 Internal Server Error\r" (else)

mycat=`/usr/local/mycat/bin/mycat status | grep 'not running' | wc -l`

if [ "$mycat" = "0" ]; then

    /bin/echo -e "HTTP/1.1 200 OK\r\n"

else

    /bin/echo -e "HTTP/1.1 503 Service Unavailable\r\n"

fi

6、给新增脚本赋予可执行权限

# chmod a+x /usr/local/bin/mycat\_status

7、在 /etc/services 中加入 mycat\_status 服务

# vi /etc/services

在末尾加入：

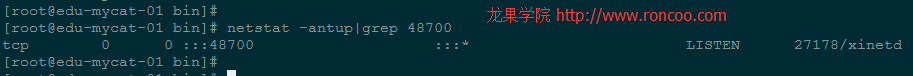
mycat\_status    48700/tcp               # mycat\_status

保存后，重启xinetd服务

# service xinetd restart

8、验证mycat\_status服务是否成功启动

# netstat -antup|grep 48700



能看到上图这样的信息，说明服务配置成功。

9、MyCat服务主机的防火墙上打开 48700端口

# vi /etc/sysconfig/iptables

增加：

-A INPUT -m state --state NEW -m tcp -p tcp --dport 48700 -j ACCEPT

保存后重启防火墙

# service iptables restart

脚本测试：

# /usr/local/bin/mycat\_status

七、HAProxy介绍

HAProxy官网：<http://www.haproxy.org/>

HAProxy各版本的官方文档：<http://cbonte.github.io/haproxy-dconv/index.html>

HAProxy 是一款提供高可用性、负载均衡以及基于TCP（第四层）和HTTP（第七层）应用的代理软件，支持虚拟主机，它是免费、快速并且可靠的一种解决方案。

HAProxy目前主要有三个版本：1.4、1.5、1.6，CentOS6.6自带的RPM包为1.5的。

HAProxy1.5版开始，支持SSL、DDoS防护等功能，可看官网说明：

version 1.5 : the most featureful version, supports SSL, IPv6, keep-alive, DDoS protection, etc...

MyCat官方推荐使用HAProxy做MyCat的高可用负载均衡代理。

八、HAProxy的安装（192.168.1.191）

1、下载（或上传） haproxy-1.5.16.tar.gz 到 /usr/local/src，解压安装

[root@edu-haproxy-01 src]# cd /usr/local/src/

#wget http://www.haproxy.org/download/1.5/src/haproxy-1.5.16.tar.gz

[root@edu-haproxy-01 src]# tar -zxvf haproxy-1.5.16.tar.gz

[root@edu-haproxy-01 src]# cd haproxy-1.5.16

2、如需了解安装注意点，可查看HAProxy的软件说明

[root@edu-haproxy-01 haproxy-1.5.16]# less README

3、安装编译所需的依赖包

# yum install gcc gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel

4、编译

# make  TARGET=linux2628  ARCH=x86\_64  USE\_PCRE=1  USE\_OPENSSL=1  USE\_ZLIB=1  PREFIX=/usr/local/haproxy

## TARGET是指定内核版本，高于2.6.28的建议设置为linux2628，Linux操作系统内核版本查看命令# uname -r， ARCH指定系统架构，openssl  pcre  zlib 这三个包需要安装不然不支持

5、创建安装目录 /usr/local/haproxy

# mkdir /usr/local/haproxy

6、执行安装

[root@edu-haproxy-01 haproxy-1.5.16]# make install PREFIX=/usr/local/haproxy

install -d "/usr/local/haproxy/sbin"

install haproxy  "/usr/local/haproxy/sbin"

install -d "/usr/local/haproxy/share/man"/man1

install -m 644 doc/haproxy.1 "/usr/local/haproxy/share/man"/man1

install -d "/usr/local/haproxy/doc/haproxy"

for x in configuration architecture haproxy-en haproxy-fr; do \

                install -m 644 doc/$x.txt "/usr/local/haproxy/doc/haproxy" ; \

        done

7、创建配置文件目录

# mkdir -p /usr/local/haproxy/conf

# mkdir -p /etc/haproxy/

8、从配置文件模版复制配置文件，并添加配置文件软连接

# cp /usr/local/src/haproxy-1.5.16/examples/haproxy.cfg /usr/local/haproxy/conf/

# ln -s /usr/local/haproxy/conf/haproxy.cfg /etc/haproxy/haproxy.cfg

9、拷贝错误页面，并添加目录软连接（HTTP模式选配）

# cp -r /usr/local/src/haproxy-1.5.16/examples/errorfiles /usr/local/haproxy/

# ln -s /usr/local/haproxy/errorfiles /etc/haproxy/errorfiles

10、拷贝开机启动文件，并赋予可执行权限

# cp /usr/local/src/haproxy-1.5.16/examples/haproxy.init  /etc/rc.d/init.d/haproxy

# chmod +x /etc/rc.d/init.d/haproxy

11、添加haproxy命令脚本软连接

# ln -s /usr/local/haproxy/sbin/haproxy /usr/sbin

12、设置HAProxy开机启动

# chkconfig --add haproxy

# chkconfig haproxy on

九、HAProxy配置MyCat负载均衡集群

     HAProxy支持TCP（第四层）和HTTP（第七层）应用的代理，本节课程我们使用HAProxy来做MyCat的负载均衡代理使用的是TCP模式。在4层模式下HAProxy仅在客户端和服务器之间转发双向流量。HAProxy配置简单，拥有非常不错的服务器健康检查功能，当其代理的后端服务器出现故障，HAProxy会自动将该服务器摘除，故障恢复后会自动将该服务器加入进来｡

1、修改haproxy.cfg 配置文件

具体参数说明可参考官方配置文档 /usr/local/haproxy/doc/haproxy/configuration.txt

或GitHub连接：<http://cbonte.github.io/haproxy-dconv/configuration-1.5.html>

# vi /usr/local/haproxy/conf/haproxy.cfg

## global配置中的参数为进程级别的参数，通常与其运行的操作系统有关

global

      log 127.0.0.1    local0     info ## 定义全局的syslog服务器，最多可以定义2个

      ### local0是日志设备，对应于/etc/rsyslog.conf中的配置，默认回收info的日志级别

      #log 127.0.0.1   local1 info

      chroot /usr/share/haproxy   ## 修改HAProxy的工作目录至指定的目录并在放弃权限之前执行

                                             ### chroot() 操作，可以提升 haproxy 的安全级别

      group haproxy    ## 同gid，不过这里为指定的用户组名

      user  haproxy    ## 同uid，但这里使用的为用户名

      daemon     ## 设置haproxy后台守护进程形式运行

      nbproc     1     ## 指定启动的haproxy进程个数，

                      ### 只能用于守护进程模式的haproxy；默认为止启动1个进程，

                      ### 一般只在单进程仅能打开少数文件描述符的场中中才使用多进程模式

      maxconn 4096     ## 设定每个haproxy进程所接受的最大并发连接数，

                            ### 其等同于命令行选项"-n"，"ulimit-n"自动计算的结果正式参照从参数设定的

      # pidfile  /var/run/haproxy.pid  ## 进程文件（默认路径 /var/run/haproxy.pid）

      node  edu-haproxy-01   ## 定义当前节点的名称，用于HA场景中多haproxy进程共享同一个IP地址时

      description  edu-haproxy-01  ## 当前实例的描述信息

## defaults：用于为所有其他配置段提供默认参数，这默认配置参数可由下一个"defaults"所重新设定

defaults

      log   global     ## 继承global中log的定义

      mode  http  ## mode:所处理的模式 (tcp:四层 , http:七层 , health:状态检查,只会返回OK)

      ### tcp: 实例运行于纯tcp模式，在客户端和服务器端之间将建立一个全双工的连接，

      #### 且不会对7层报文做任何类型的检查，此为默认模式

      ### http:实例运行于http模式，客户端请求在转发至后端服务器之前将被深度分析，

      #### 所有不与RFC模式兼容的请求都会被拒绝

      ### health：实例运行于health模式，其对入站请求仅响应“OK”信息并关闭连接，

      #### 且不会记录任何日志信息 ，此模式将用于相应外部组件的监控状态检测请求

      option     httplog

      retries    3

      option redispatch     ## serverId对应的服务器挂掉后,强制定向到其他健康的服务器

      maxconn    2000  ## 前端的最大并发连接数（默认为2000）

      ### 其不能用于backend区段，对于大型站点来说，可以尽可能提高此值以便让haproxy管理连接队列，

      ### 从而避免无法应答用户请求。当然，此最大值不能超过“global”段中的定义。

      ### 此外，需要留心的是，haproxy会为每个连接维持两个缓冲，每个缓存的大小为8KB，

      ### 再加上其他的数据，每个连接将大约占用17KB的RAM空间，这意味着经过适当优化后 ，

      ### 有着1GB的可用RAM空间时将维护40000-50000并发连接。

      ### 如果指定了一个过大值，极端场景中，其最终所占据的空间可能会超过当前主机的可用内存，

      ### 这可能会带来意想不到的结果，因此，将其设定一个可接受值放为明智绝对，其默认为2000

      timeout connect  5000ms     ## 连接超时(默认是毫秒,单位可以设置us,ms,s,m,h,d)

      timeout client   50000ms    ## 客户端超时

      timeout server   50000ms    ## 服务器超时

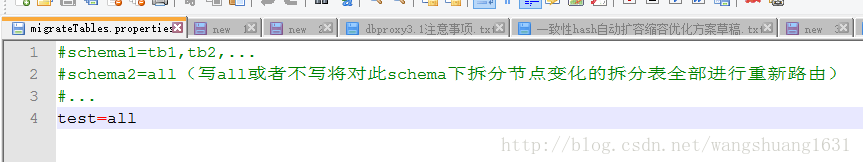
## HAProxy的状态信息统计页面

listen admin\_stats

<p style="color: #333333;

# [Mycat生产实践---数据迁移与扩容实践](http://blog.csdn.net/kimva/article/details/78875154)

## 1 离线扩容缩容

工具目前从mycat1.6开始支持。   
一、准备工作   
1、mycat所在环境安装mysql客户端程序   
2、mycat的lib目录下添加mysql的jdbc驱动包   
3、对扩容缩容的表所有节点数据进行备份，以便迁移失败后的数据恢复   
二、扩容缩容步骤   
1、复制schema.xml、rule.xml并重命名为newSchema.xml、newRule.xml放于conf目录下   
   
2、修改newSchema.xml和newRule.xml配置文件为扩容缩容后的mycat配置参数（表的节点数、数据源、路由规则）   
3、修改conf目录下的migrateTables.properties配置文件，告诉工具哪些表需要进行扩容或缩容,没有出现在此配置文件的schema表不会进行数据迁移，格式：   
  
4、修改bin目录下的dataMigrate.sh脚本文件，参数如下：

tempFileDir 临时文件路径,目录不存在将自动创建

isAwaysUseMaster默认true:不论是否发生主备切换，都使用主数据源数据，false：使用当前数据源

mysqlBin：mysql bin路径

cmdLength mysqldump命令行长度限制 默认110k 110\*1024。在LINUX操作系统有限制单条命令行的长度是128KB，也就是131072字节，这个值可能不同操作系统不同内核都不一样，如果执行迁移时报Cannot run program "sh": error=7, Argument list too long 说明这个值设置大了，需要调小此值。

charset导入导出数据所用字符集 默认utf8

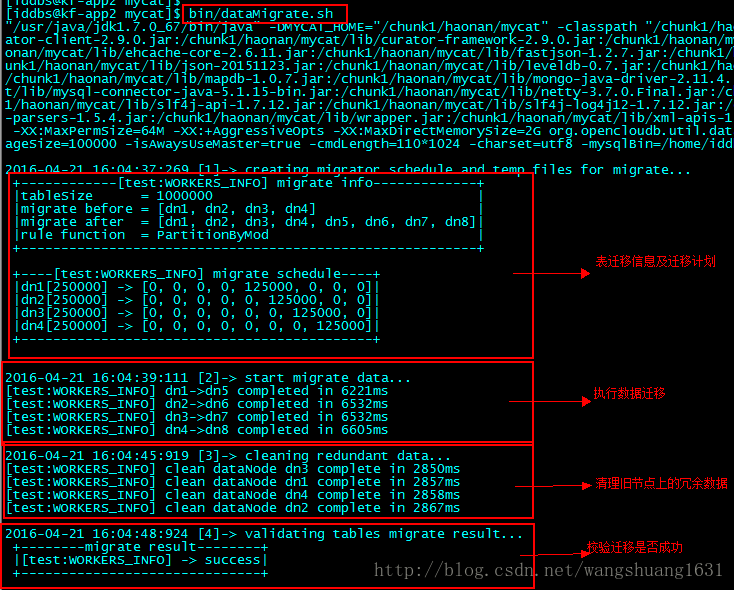
deleteTempFileDir完成扩容缩容后是否删除临时文件 默认为true

threadCount并行线程数（涉及生成中间文件和导入导出数据）默认为迁移程序所在主机环境的cpu核数\*2

delThreadCount每个数据库主机上清理冗余数据的并发线程数，默认为当前脚本程序所在主机cpu核数/2

queryPageSize 读取迁移节点全部数据时一次加载的数据量 默认10w条

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

5、停止mycat服务（如果可以确保扩容缩容过程中不会有写操作，也可以不停止mycat服务）   
6、通过crt等工具进入mycat根目录，执行bin/ dataMigrate.sh脚本，开始扩容/缩容过程：   
   
7、扩容缩容成功后，将newSchema.xml和newRule.xml重命名为schema.xml和rule.xml并替换掉原文件，重启mycat服务，整个扩容缩容过程完成。   
三、注意事项：   
1) 保证拆分表迁移数据前后路由规则一致   
2) 保证拆分表迁移数据前后拆分字段一致   
3) 全局表将被忽略   
4) 不要将非拆分表配置到migrateTables.properties文件中   
5) 暂时只支持拆分表使用mysql作为数据源的扩容缩容   
四、优化   
dataMigrate.sh脚本中影响数据迁移速度的有4个参数，正式迁移数据前可以先进行一次测试，通过调整以下参数进行优化获得一个最快的参数组合

threadCount脚本执行所在主机的并行线程数（涉及生成中间文件和导入导出数据）默认为迁移程序所在主机环境的cpu核数\*2

delThreadCount每个数据库主机上清理冗余数据的并发线程数，默认为当前脚本程序所在主机cpu核数/2，同一主机上并发删除数据操作线程数过多可能会导致性能严重下降，可以逐步提高并发数，获取执行最快的线程个数。

queryPageSize 读取迁移节点全部数据时一次加载的数据量 默认10w条

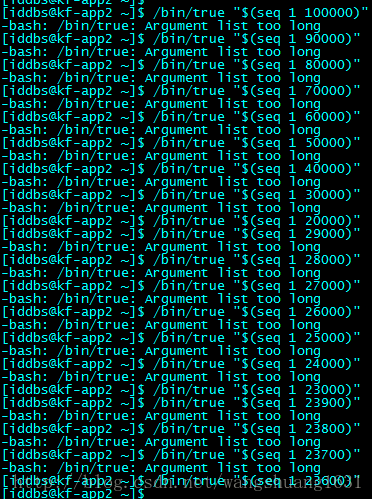
cmdLength mysqldump命令行长度限制 默认110k 110\*1024。尽量让这个值跟操作系统命令长度最大值一致，可以通过以下过程确定操作系统命令行最大长度限制：

逐步减少100000，直到不再报错

/bin/sh -c "/bin/true $(seq 1 100000)"

获取不报错的值，通过wc –c统计字节数，结果即操作系统命令行最大长度限制（可能稍微小一些）

* 1
* 2
* 3
* 4
* 5
* 6
* 7

   
这里写图片描述

## 2 案例一：使用一致性Hash进行分片

当使用一致性Hash进行路由分片时，假设存在节点宕机/新增节点这种情况，那么相对于使用其他分片算法(如mod)，就能够尽可能小的改变已存在key映射关系，尽可能的减少数据迁移操作。当然一致性hash也有一个明显的不足，假设当前存在三个节点A,B,C，且是使用一致性hash进行分片，如果你想对当前的B节点进行扩容，扩容后节点为A,B,C,D，那么扩容完成后数据分布就会变得不均匀。A,C节点的数据量是大于B,D节点的。   
据测试，分布最均匀的是mod，一致性哈希只是大致均匀。数据迁移也是，迁移量最小的做法是mod，每次扩容后节点数都是2的N次方，这样的迁移量最小。但是mod需要对每个节点都进行迁移，这也是mod的不足之处。总之，还得酌情使用，根据业务选择最适合自己系统的方案。   
2.1 配置使用   
rule.xml：定义分片规则

<tableRule name="sharding-by-murmur">

<rule>

<columns>SERIAL\_NUMBER</columns> <algorithm>murmur</algorithm>

</rule> </tableRule>

<function name="murmur" class="org.opencloudb.route.function.PartitionByMurmurHash">

<property name="seed">0</property>

<property name="count">2</property>

<property name="virtualBucketTimes">160</property>

</function>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

tableRule定义分片规则

* name：分片规则的名字。在schema.xml文件中调用。
* columns：根据数据库中此字段进行分片。
* algorithm：值是分片算法定义处的name属性。比如：murmur。

function定义一致性Hash的参数

* seed：计算一致性哈希的对象使用的数值，默认是0。
* count：待分片的数据库节点数量，必须指定，否则没法分片。
* virtualBucketTimes：虚拟节点。默认是160倍，也就是虚拟节点数是物理节点数的160倍。指定virtualBucketTimes可以使一致性hash分片更加均匀。
* bucketMapPath：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。必须是绝对路径，且可读写。   
  schema.xml：定义逻辑库，表、分片节点等内容

<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://org.opencloudb/">

<schema name="mycat" checkSQLschema="false" sqlMaxLimit="100">

<table name="T\_CMS\_ORDER" primaryKey="ORDER\_ID" dataNode="dn202\_3316"

rule="sharding-by-murmur" />

</schema>

<dataNode name="dn202\_3316" dataHost="lh202\_1" database="poc" />

<dataHost name="lh202\_1" maxCon="2000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native">

<heartbeat>select user()</heartbeat>

<writeHost host="master\_host-m1" url="10.21.17.202:3316"

user="usr" password="pwd"></writeHost>

<writeHost host="savle\_host-m1" url="10.21.17.201:3317"

user="usr" password="pwd"></writeHost>

</dataHost>

</mycat:schema>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

server.xml：定义用户以及系统相关变量，如端口等。没有太高要求的可以只修改数据库部分。

<user name="mycat"> <property name="password">usr</property> <property name="schemas">pwd</property> </user>

* 1

经过以上配置就可以使用一致性hash了。   
2.2 一致性Hash的数据迁移   
开始迁移   
进行一致性hash进行迁移的时候，假设你新增加一个节点，需要修改以下两个配置文件：   
rule.xml

<function name="murmur" class="org.opencloudb.route.function.PartitionByMurmurHash"> <property name="seed">0</property>

<property name="count">3</property> <property name="virtualBucketTimes">160</property> <!-- <property name="weightMapFile">weightMapFile</property> <property name="bucketMapPath">/home/usr/mycat/bucketMapPath</property> --> </function>

* 1
* 2

需要把节点的数量从2个节点扩为3个节点。   
schema.xml

<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://org.opencloudb/">

<schema name="mycat" checkSQLschema="false" sqlMaxLimit="100">

<table name="T\_CMS\_ORDER" primaryKey="ORDER\_ID" dataNode="dn202\_3316,dn201\_3316"

rule="sharding-by-murmur" />

</schema>

<dataNode name="dn202\_3316" dataHost="lh202\_1" database="poc" />

<dataNode name="dn201\_3316" dataHost="lh201\_1" database="poc" />

<dataHost name="lh202\_1" maxCon="2000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native">

<heartbeat>select user()</heartbeat>

<writeHost host="master\_host-m1" url="10.21.17.202:3316"

user="usr" password="pwd"></writeHost>

<writeHost host="savle\_host-m1" url="10.21.17.201:3317"

user="usr" password="pwd"></writeHost>

</dataHost>

<dataHost name="lh201\_1" maxCon="2000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native">

<heartbeat>select user()</heartbeat>

<writeHost host="master\_host-m1" url="10.21.17.201:3316"

user="usr" password="pwd"></writeHost>

<writeHost host="savle\_host-m1" url="10.21.17.202:3317"

user="usr" password="pwd"></writeHost>

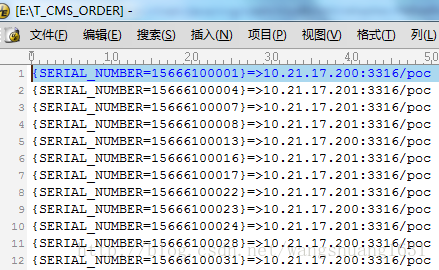
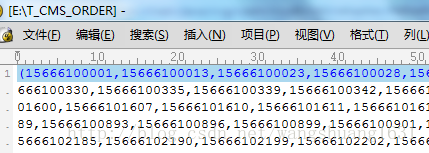
</dataHost>

</mycat:schema>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

需要添加新节点的dataNode和dataHost信息，以及在schema中的table标签下把新增节点的dataNode的name增加到dataNode的值中。   
2.3 开始迁移   
使用org.opencloudb.util.rehasher.RehashLauncher类进行数据迁移。参数以命令行的形式进行载入。如

-jdbcDriver=xxxxx -jdbcUrl=.... -host=192.168.1.1:3316 -user=xxxx -password=xxxx -database=xxxx

* 1
* jdbcDriver：数据库驱动。如com.mysql.jdbc.Driver。
* jdbcUrl：连接数据库的url，不同数据库不一样。如jdbc:mysql://10.21.17.201:3316/mycat?rewriteBatchedStatements=true。
* host：包括主机名和端口，形如ip:port。如10.21.100.86:3316
* user：连接数据库的用户名。如usr
* database：数据库的名字。如mycat。
* password：连接数据库的密码。如pwd。
* tablesFile：记录数据表的文件，一个表一行。
* shardingField：数据库中进行分片的字段。
* rehashHostsFile：这个参数没有用到，按照当时的要求，这个类一次只处理一个节点，所以不需要配置
* hashType：是MURMUR hash还是mod hash。
* seed：生成一致性hash对象的参数。默认为0。
* virtualBucketTimes：虚拟节点的倍数。默认为160。
* weightMapFile：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。
* rehashNodeDir：一个linux目录，这个程序执行完了，把计算结果输出到这个目录，一个表一个文件存在这个目录里，文件名是表名。   
  如果你觉得使用命令行的方式去读取配置不是那么方便，你也可以自己定义读取配置文件的算法，只要能保证org.opencloudb.util.rehasher.RehashLauncher这个类能够读到所有的配置就可以了。比如使用properties文件保存配置文件(每次修改配置文件后都需要重新编译)，本着怎么方便怎么写代码的原则，就是这么任性。   
  运行org.opencloudb.util.rehasher.RehashLauncher后生成的文件格式如下：   
     
  为了方便进行迁移，我们可以对代码进行适当的修改，如   
     
  通过此种方式拼装，生成的文件如下：   
     
  形如(15666100001,15666100013,15666100023,15666100028)，这个就可以作为in条件了。 生成文件后，可以在linux环境下通过shell的方式进行数据迁移，当然前提是你得停机。   
  迁移脚本如下：

rehashNode=$1

expanNode=$2

order\_fn="$3"

if [ "$#" = "0" ]; then

echo "Please input parameter, for example:"

echo "ReRouter.sh 192.168.84.13 192.168.84.14 /home/mycat/T\_CMS\_ORDER "

echo " "

exit fi;

echo "需要进行迁移的主机总量为:$#, 主机IP列表如下:"

for i in "$@"

do

echo "$i"

done

echo " "

#取出rehash需要的SerNum(已经用in拼接好)

for n in `cat $order\_fn`

do

condOrder=$n

done echo "\*\*\*\*\*\*\*\*\*\*\*\*\* 导出 \*\*\*\*\*\*\*\*\*\*\*\*\*"

date

# 1) 首先调用mysqldump进行数据导出

echo "开始导出主机:$ 表:T\_CMS\_ORDER."

mysqldump -h$rehashNode -P3316 -upoc -ppoc123 poc T\_CMS\_ORDER --default-character-set=utf8 --extended-insert=false --no-create-info --add-locks=false --complete-insert --where=" SERIAL\_NUMBER in $condOrder " > ./T\_CMS\_ORDER\_temp.sql

echo "导出结束."

echo " "

echo "\*\*\*\*\*\*\*\*\*\*\*\*\* 导入 \*\*\*\*\*\*\*\*\*\*\*\*\*" date

# 2) 调用mycat接口进行数据导入

echo "开始导入T\_CMS\_ORDER表数据"

mysql -h$expanNode -P8066 -upoc -ppoc123 poc --default-character-set=utf8 < ./T\_CMS\_ORDER\_temp.sql echo "导入结束."

echo " "

echo "\*\*\*\*\*\*\*\*\*\*\*\*\* 删除数据 \*\*\*\*\*\*\*\*\*\*\*\*\*" date

# 3) 当前两步都无误的情况下,删除最初的导出数据.

echo "开始删除已导出的数据表:."

mysql -h$rehashNode -P3316 -upoc -ppoc123 -e "use poc; DELETE FROM T\_CMS\_ORDER WHERE SERIAL\_NUMBER in $condOrder ; commit; "

echo "删除结束."

echo " "

echo "\*\*\*\*\*\*\*\*\*\*\*\*\* 清空临时文件 \*\*\*\*\*\*\*\*\*\*\*\*\*" date

# 4) 清空临时文件 rm ./t\_cms\_order\_temp.sql

echo "清空临时文件"

echo "#####################主机:$rehashNode 处理完成#####################"

date

echo " " echo "ReHash运行完毕."

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42

假设文件名是：ReHashRouter.sh

* 授权：chmod +x ReHashRouter.sh
* 运行：./ReHashRouter.sh 10.21.17.200 10.21.17.201   
  /home/mycat/T\_CMS\_ORDER

## 3 案例二：使用范围分片

在使用范围分片算法进行路由分片时，配置非常简单。如下：   
3.1 配置使用   
rule.xml：定义分片规则

<tableRule name="auto-sharding-long">

<rule>

<columns>user\_id</columns>

<algorithm>rang-long</algorithm>

</rule>

</tableRule>

<function name="rang-long"

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

<property name="mapFile">autopartition-long.txt</property>

</function>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

tableRule定义分片规则

* name：分片规则的名字。在schema.xml文件中调用。
* columns：根据数据库中此字段进行分片。
* algorithm：值是分片算法定义处的name属性。比如：rang-long。

function定义范围分片的参数   
可以看到根据范围自动分片的配置文件非常简单，只有一个mapFile(要赋予读的权限),此mapFile文件定义了每个节点中user\_id的范围，如果user\_id的值超过了这个范围，那么则使用默认节点。当前版本代码中默认节点的值是-1，表示不配置默认节点，超过当前范围就会报错。当然你也可以在property中增加defaultNode的默认值，如：

<property name="defaultNode">0</property>

* 1

mapFile节点配置文件   
当前版本提供了一个mapFile配置文件供大家参考和使用，如下

# range start-end ,data node index

# K=1000,M=10000. 0-500M=0 500M-1000M=1

* 1
* 2

所有的节点配置都是从0开始，及0代表节点1，此配置非常简单，即预先制定可能的id范围到某个分片。   
(tips:K和M的定义是在org.opencloudb.route.function.NumberParseUtil中定义的,如果感兴趣的同学可以自己定义其他字母。)   
扩容   
如果业务需要或者数据超过当前定义的范围，需要新增节点，则可以在文件中追加 1000M-1500M=2 即可。当然新增的节点需要在schema.xml中进行定义。

# range start-end ,data node index

# K=1000,M=10000. 0-500M=0

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

* 1
* 2
* 3

## 4 数据迁移的注意点

4.1 迁移时间的确定   
在进行迁移之前，我们得先确定迁移操作发生的时间。停机操作需要尽可能的让用户感知不到，你可以观察每段时间系统的吞吐量，以此作为依据。一般来说，我们选择在凌晨进行升级操作。   
4.2 数据迁移前的测试   
需要做一些相关的性能测试，在条件允许的情况下在类似的环境中完全模拟，得到一些性能数据，然后不断的改进，看能够否有大的提升。   
我们在做数据迁移的时候，就是在备份库中克隆的一套环境，然后在上面做的性能测试，在生产上的步骤方式都一样，之后在正式升级的时候就能够做到心中有数。什么时候需要注意什么，什么时候需要做哪些相关的检查。   
4.3 数据备份   
热备甚至冷备，在数据迁移之前进行完整的备份，一定要是全量的。甚至在允许的情况下做冷备都可以。数据的备份越充分，出现问题时就有了可靠的保证。   
lob数据类型的备份，做表级的备份（create table nologging….)，对于lob的数据类型，在使用imp,impdp的过程中，瓶颈都在lob数据类型上了，哪怕表里的lob数据类型是空的，还是影响很大。自己在做测试的时候，使用Imp基本是一秒钟一千条的数据速度，impdp速度有所提升，但是parallle没有起作用，速度大概是1秒钟1万条的样子。   
如果在数据的导入过程中出了问题，如果有完整快速的备份，自己也有了一定的数据保证，要知道出问题之后再从备份库中导入导出，基本上都是很耗费时间的。   
4.4 数据升级前的系统级检查

* 内存检查。可以使用top,free -m来做一个检查，看内存的使用情况是否正常，是否有足够的内存空间。
* 检查cpu,io情况。查看iowait是否稳定，保持在较低的一个幅度。
* 检查进程的情况。检查是否有高cpu消耗的异常进程，检查是否有僵尸进程，排查后可以杀掉。
* 是否有crontab的设置。如果在升级的时候有什么例行的job在运行，会有很大的影响，可以使用crontab -l来查看crontab的情况。
* vxfs下的odm是否已经启用。如果使用的veritas的文件系统，需要检查一下odm是否正常启用。
* IO 简单测试。从系统角度来考虑，需要保证io的高效性。可以使用iostat,sar等来评估。
* 网络带宽。数据迁移的时候肯定会从别的服务器中传输大量的文件,dump等，如果网络太慢，无形中就是潜在的问题。可以使用scp来进行一个简单的测试。   
  4.5 异常情况   
  网络临时中断。网络的问题需要格外重视，可能在运行一些关键的脚本时，网络突然中断，那对于升级就是灾难，所以在准备脚本的时候，需要考虑到这些场景，保留完整的日志记录。   
  可以使用nohup来做外后台运行某些关键的脚本。这样网络断了以后，还有一线希望。在数据迁移，数据升级的时候，一定要保留完整的日志记录，这样如果稍候有问题，也可以及时查验，也可以避免很多不必要的纷争。如果有争议，可以找出日志来，一目了然。   
  当然，这样会有大量的日志产生，一定需要保证归档空间足够大，及时的转移归档文件。排除归档爆了以后数据的问题，使用sqlloader,impdp等数据迁移策略的时候，如果归档出了问题，是很头疼的问题。

## 5 load data批量导入

load data infile语句可以从一个文本文件中以很高的速度读入一个表中。性能大概是insert语句的几十倍。通常用来批量数据导入。目前只支持mysql数据库且dbDriver必须为native。Mycat支持load data自动路由到对应的分片。Load data和压缩协议mycat从1.4开始支持。   
5.1 语法和注意事项   
标准示例：

load DATA local INFILE 'd:\88\qq.txt' IGNORE INTO TABLE test CHARACTER SET 'gbk' FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n'(id,sid,asf) ;

* 1

注意：如果数据中可能包含一些特殊字符，比如分割符转义符等，建议用引号扩起来，通过OPTIONALLY ENCLOSED BY ‘”’指定。如果这样还不行，可以把字段值中的引号替换成\”。   
如果指定local关键词，则表明从客户端主机读文件。如果local没指定，文件必须位于mycat所在的服务器上。   
可以通过fields terminated by指定字符之间的分割符号，默认值为\t   
通过lines terminated by可以指定行之间的换行符。默认为\n,这里注意有些windows上的文本文件的换行符可能为\r\n，由于是不可见字符，所以请小心检查。   
character set 指定文件的编码，建议跟mysql的编码一致，否则可能乱码。其中字符集编码必须用引号扩起来，否则会解析出错。   
还可以通过replace | ignore指定遇到重复记录是替换还是忽略。   
目前列名必须指定，且必须包括分片字段，否则没办法确定路由。   
其他参数参考mysql的load data infile官方文档说明。   
注意其他参数的先后顺序不能乱，比如列名比较在最后的，顺序参考官方说明。   
标准load data语句： LOAD DATA语句,同样被记录到binlog,不过是内部的机制.   
例子： 导出：

select \* from tblog\_article into outfile '/test.txt' FIELDS TERMINATED BY '\t' OPTIONALLY ENCLOSED BY '' ESCAPED BY '\\' LINES TERMINATED BY '\n';

* 1

导入：

load data local infile '/var/lib/mysql/blog/test.txt' INTO TABLE tblog\_article FIELDS TERMINATED BY '\t' OPTIONALLY ENCLOSED BY '' ESCAPED BY '\\' LINES TERMINATED BY '\n' (id,title,level,create\_time,create\_user,create\_user,article\_type\_id,article\_content,istop,status,read\_count );

* 1

5.2 客户端配置   
如果是mysql命令行连接的mycat，则需要加上参数–local-infile=1。Jdbc则无需设置。   
Load data测试性能   
在一台win8下 ，jvm 1.7 参数默认，jdbc连接mycat。   
   
测试结果load data local导入1百万数据到5个分片耗时10秒，1千万数据到5个分片耗时145秒。

## 6 使用mysqldump进行数据迁移

mysqldump是mysql自带的命令行工具。可以用它完成全库迁移（从一个mysql库完整迁移到mycat），也可以迁移某几个表，还可以迁移某个表的部分数据。   
6.1 全库迁移   
迁移前准备   
迁移前确保mysql库和mycat库中的表名一样（mycat库中只需要有表名配置在schema.xml文件中即可）   
从mysql导出   
从mysql库上全库导出 mysqldump -c -–skip-add-locks databaseName> /root/databaseName.sql   
注意：（上面的语句没有-uroot -ppassword参数，是因为mysql服务器设置了本机免密码等。   
如果设置了密码:通过以下命名导出(用户名为root，密码为123456)： mysqldump -uroot -p123456 -c -–skip-add-locks databaseName> /root/databaseName.sql   
）   
说明:两个参数不可少，如下：   
-c参数不可少，-c, 全称为–complete-insert 表示使用完整的insert语句(用列名字)。   
-–skip-add-locks表示导数据时不加锁，如果加锁涉及多分片时容易导致死锁。   
导入到mycat   
将databaseName.sql拷贝到mycat集群中的一台mysql服务器上/root目录下。   
连接mycat：   
mysql -uusername -ppassword -h172.17.xxx.xxx -P8066   
切换到指定的数据库： use databaseName;   
导入脚本： source /root/databaseName.sql;   
6.2 迁移一个库中的某几个表   
只是导出命令不同，其他与全库迁移一样

mysqldump -c -–skip-add-locks databaseName table1 table2> /root/someTables.sql

* 1

## 7 迁移一个表中的部分数据

迁移一个表中的部分数据，加参数–where实现。   
命令如下：

mysqldump -c -–skip-add-locks databaseName tableName --where=" id > 900 " > /root/onetableDataWithCondition.sql

# 脚本

**登录Mycat mysql -h127.0.0.1 -P8066 -u root -p**

123456