mysql高级

1.读写分离环境搭建(一)

这是松哥之前一个零散的笔记，整理出来分享给大伙！

MySQL 读写分离在互联网项目中应该算是一个非常常见的需求了。受困于 Linux 和 MySQL 版本问题，很多人经常会搭建失败，今天松哥就给大伙举一个成功的例子，后面有时间再和大家分享下使用 Docker 搭建环境，那样就 100% 成功了。

CentOS 安装 MySQL

自己玩 Linux 松哥一般首选 Ubuntu，不过公司里边使用一般还是 CentOS 为主，因此这里松哥就以 CentOS 为例来向大家演示整个过程，今天这篇文章主要来看看 MySQL 的安装。

环境：

- CentOS7
- MySQL5.7

具体的安装步骤如下：

- 检查是否安装了 mariadb，如果已经安装了则卸载：

```
yum list installed | grep mariadb
```

如果执行结果如下，表示已经安装了 mariadb，将之卸载：

```
mariadb-libs.x86_64                 1:5.5.52-1.el7                  @anaconda
```

卸载命令如下：

```
yum -y remove mariadb*
```

- 接下来下载官方提供的 rpm 包

如果 CentOS 上没有 wget 命令，首先通过如下命令安装 wget：

```
yum install wget
```

然后执行如下操作下载 rpm 包：

```
wget https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
```

- 下载完成后，安装rpm包：

```
rpm -ivh mysql57-community-release-el7-11.noarch.rpm
```

- 检查 MySQL 的 yum 源是否安装成功：

```
yum repolist enabled | grep "mysql.*-community.*"
```

执行结果如下表示安装成功：

```
[root@localhost ~]# yum repolist enabled | grep "mysql.*-community.*"
mysql-connectors-community/x86_64       MySQL Connectors Community          45
mysql-tools-community/x86_64            MySQL Tools Community               59
mysql56-community/x86_64                MySQL 5.6 Community Server          378
```

- 安装 MySQL

```
yum install mysql-server
```

- 安装完成后，启动MySQL：

```
systemctl start mysqld.service
```

- 停止MySQL：

```
systemctl stop mysqld.service
```

- 登录 MySQL：

```
mysql -u root -p
```

默认无密码。有的版本有默认密码，查看默认密码，首先去 /etc/my.cnf 目录下查看 MySQL 的日志位置，然后打开日志文件，可以看到日志中有一个提示，生成了一个临时的默认密码，使用这个密码登录，登录成功后修改密码即可。

- 改密码

首先修改密码策略(这一步不是必须的，如果不修改密码策略，需要取一个比较复杂的密码，松哥这里简单起见，就修改下密码策略)：

```
set global validate_password_policy=0;
```

然后重置密码：

```
set password=password("123");
flush privileges;
```

- 授权远程登录同方式一：

```
grant all privileges on *.* to 'root'@'%' identified by '123' with grant option;
flush privileges;
```

- 授权远程登录同方式二：

修改 mysql 库中的 user 表，将 root 用户的 Host 字段的值改为 `%` ，然后重启 MySQL 即可。

- 关闭防火墙
  MySQL 要能远程访问，还需要关闭防火墙：

```
systemctl stop firewalld.service
```

禁止firewall开机启动:

```
systemctl disable firewalld.service
```

总结

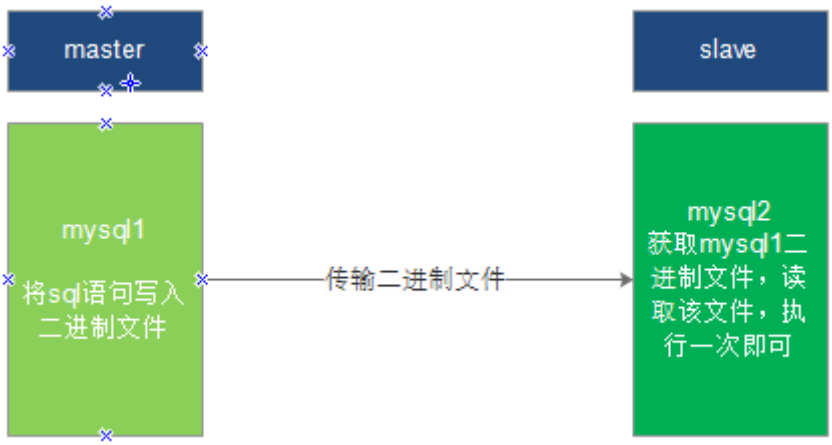
装了这么多 MySQL，还是 Ubuntu 下 MySQL 最好弄，其他系统多多少少总有点麻烦，本文主要和大家分享了 CentOS7 中 MySQL 的安装步骤，大伙有问题欢迎留言讨论。下篇文章和大伙分享 MySQL 读写分离环境搭建。

2. 读写分离环境搭建(二)

上篇文章和大家聊了 CentOS7 安装 MySQL5.7，这个大家一般装在虚拟机里边，装好了，把虚拟拷贝一份，这样我们就有两个 MySQL，就可以开始今天的主从搭建了。

准备工作

我这里有一张简单的图向大伙展示 MySQL 主从的工作方式:



这里，我们准备两台机器:

- 主机：192.168.248.128
- 从机：192.168.248.139

主机配置

主机的配置就三个步骤，比较容易:

1.授权给从机服务器

```
GRANT REPLICATION SLAVE ON *.* to 'rep1'@'192.168.248.139' identified by '123';
FLUSH PRIVILEGES;
```

这里表示配置从机登录用户名为 rep1，密码为 123，并且必须从 192.168.248.139这个地址登录，登录成功之后可以操作任意库中的任意表。其中，如果不需要限制登录地址，可以将 IP 地址更换为一个 `%`。

2.修改主库配置文件，开启 binlog ， 并设置 server-id ， 每次修改配置文件后都要重启 MySQL 服务才会生效

```
vi /etc/my.cnf
```

修改的文件内容如下：

```
[mysqld]
log-bin=/var/lib/mysql/binlog
server-id=128
binlog-do-db = cmdb
```

如下图：

```
[mysqld]
log-bin=/var/lib/mysql/binlog
server-id=128
binlog-do-db=cmdb
binlog-do-db=db1
binlog-do-db=db2
binlog-do-db=db3
```

- log-bin：同步的日志路径及文件名，一定注意这个目录要是 MySQL 有权限写入的（我这里是偷懒了，直接放在了下面那个datadir下面）。
- binlog-do-db：要同步的数据库名，当从机连上主机后，只有这里配置的数据库才会被同步，其他的不会被同步。
- server-id: MySQL 在主从环境下的唯一标志符，给个任意数字，注意不能和从机重复。

配置完成后重启 MySQL 服务端：

```
systemctl restart mysqld
```

3.查看主服务器当前二进制日志名和偏移量，这个操作的目的是为了在从数据库启动后，从这个点开始进行数据的恢复：

```
show master status;
```

```
mysql> show master status;
+---------------+----------+------------------+------------------+-------------------+
| File          | Position | Binlog_Do_DB     | Binlog_Ignore_DB | Executed_Gtid_Set |
+---------------+----------+------------------+------------------+-------------------+
| binlog.000129 |      480 | cmdb,db1,db2,db3 |                  |                   |
+---------------+----------+------------------+------------------+-------------------+
1 row in set (0.00 sec)
```

至此，主机配置完成。

从机配置

从机的配置也比较简单，我们一步一步来看：

1.在/etc/my.cnf 添加下面配置：

```
[mysqld]
server-id=139
```

注意从机这里只需要配置一下 server-id 即可。

注意：如果从机是从主机复制来的，即我们通过复制 CentOS 虚拟机获取了 MySQL 实例，此时两个 MySQL 的 uuid 一样（正常安装是不会相同的），这时需要手动修改，修改位置在 `/var/lib/mysql/auto.cnf`，注意随便修改这里几个字符即可，但也不可太过于随意，例如修改了 uuid 的长度。

2.使用命令来配置从机：

```
change master to
master_host='192.168.248.128',master_port=3306,master_user='rep1',master_passwor
d='123',master_log_file='binlog.000001',master_log_pos=120;
```

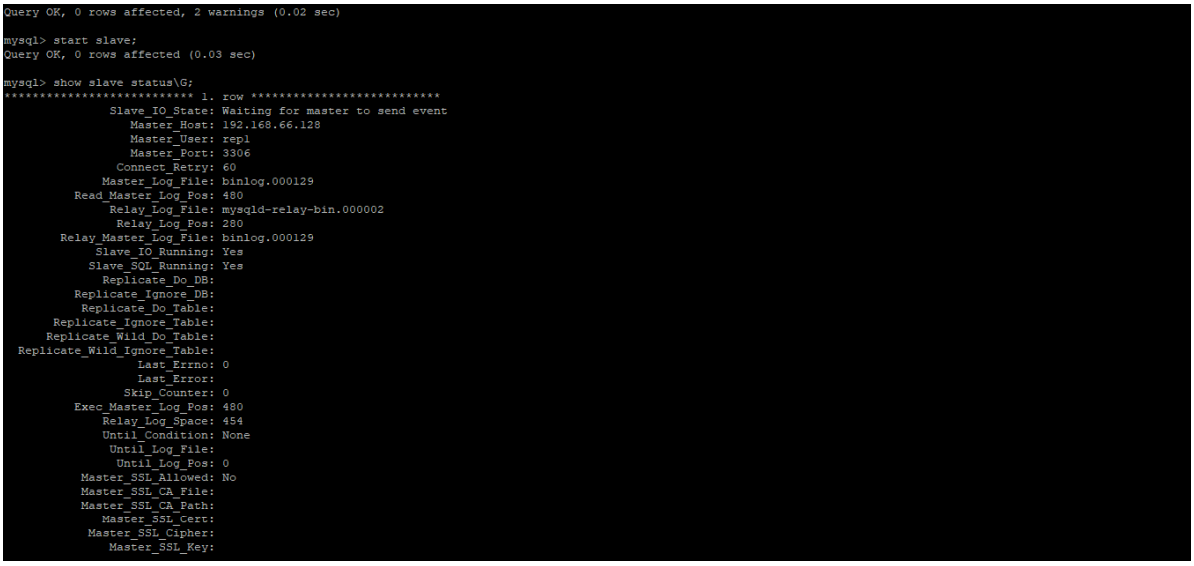
这里配置了主机地址、端口以及从机登录主机的用户名和密码，注意最后两个参数要和 master 中的保持一致。

3.启动 slave 进程

```
start slave;
```

启动之后查看从机状态：

```
show slave status\G;
```



4.查看 slave 的状态

主要是下面两项值都要为为 YES，则表示配置正确：

```
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

至此，配置完成，主机创建库，添加数据，从机会自动同步。

如果这两个有一个不为 YES，表示主从环境搭建失败，此时可以阅读日志，查看出错的原因，再具体问题具体解决。

总结

本文主要和大伙说了 MySQL 主从环境搭建，这几个步骤松哥反反复复操作过很多遍，小伙伴只要按照松哥的步骤一般来说都能成功，有问题欢迎留言讨论。

3.MySQL 只能做小项目?

松哥上学那会，很多人对 MySQL 有一些偏见，偏见主要集中在以下几方面：

1. MySQL 不支持事务（事实上 MyISAM 有表锁，但是效率比较低）
2. MySQL 存储的数据量比较小，适合小项目，大项目还是得上 Oracle、DB2 等

这么多年过去了，松哥自己在开发中一直是以 MySQL 为主，我觉得我有必要说两句公道话了。

公道话

第一个问题

关于第一个不支持事务的问题，这有一定的历史原因。MySQL 从设计之初，存储引擎就是可插拔的，允许公司或者个人按照自己的需求定义自己的存储引擎（当然，普通的公司或者个人其实是没有这个实力的）。MySQL 自研的使用较广的存储引擎是 MyISAM，MyISAM 支持表锁，不支持行锁，所以在处理高并发写操作时效率要低一些，另外 MyISAM 也不支持外键（虽然现在实际项目中外键已经用的比较少了）。

但是这个问题并非无解。这就不得不说 MySQL 中另外一个大名鼎鼎的存储引擎 InnoDB 了。

InnoDB 存储引擎是由一家位于芬兰赫尔辛基的名为 Innobase Oy 的公司开发的，InnoDB 存储引擎的历史甚至比 MySQL 还要悠久。

InnoDB 刚刚开发的时候，就是作为一个完整的数据库来开发的，因此功能很完备。开发出来之后，创始人是想将这个数据库卖掉的，但是没有找到买家。

后来 MySQL2.0 推出后，这种可插拔的存储引擎吸引了 Innobase Oy 公司创始人 Heikki Tuuri 的注意，在和 MySQL 沟通之后，决定将 InnoDB 作为一个存储引擎引入到 MySQL 中，MySQL 虽然支持 InnoDB，但是实际上还是主推自家的 MyISAM。

但是 InnoDB 实在太优秀了，最终在 2006 年的时候，成功吸引到大魔王 Oracle 的注意，大手一挥，就把 InnoDB 收购了。

MySQL 主推自家的 MyISAM，日子过得也很惨淡，最终在 2008 年被 sun 公司以 10 亿美元拿下，这个操作巩固了 sun 在开源领域的领袖的地位，可是一直以来 sun 公司的变现能力都比较弱，最终 sun 自己在 2009 年被 Oracle 收入囊中。那会松哥还在读高中，某一天吃午饭的时候，餐厅的电视机上播放央视的午间新闻，看到了这条消息，现在还有一些印象。

Oracle 收购 sun 之后，InnoDB 和 MySQL 就都成了 Oracle 的产品了，这下整合就变得非常容易了，在后来发布的版本中，InnoDB 慢慢就成为了 MySQL 的默认存储引擎。在最新的 MySQL8 中，元数据表也使用了 InnoDB 作为存储引擎。

InnoDB 存储引擎主要有如下特点：

1. 支持事务
2. 支持 4 个级别的事务隔离
3. 支持多版本读
4. 支持行级锁
5. 读写阻塞与事务隔离级别相关
6. 支持缓存，既能缓存索引，也能缓存数据
7. 整个表和主键以 Cluster 方式存储，组成一颗平衡树
8. ...

当然也不是说 InnoDB 一定就是好的，在实际开发中，还是要根据具体的场景来选择到底是使用 InnoDB 还是 MyISAM 。

所以第一个问题不攻自破。

第二个问题

第二个问题确实是一个硬伤。

你要是拿 MySQL 和 Oracle 比，肯定是要差一点点感觉。毕竟一个免费一个收费，而且收费的还很贵。但是这个问题并非无解。

相信很多小伙伴都听过国内很多大厂都使用了 MySQL 来存储数据。大厂用 MySQL ，是因为他们有能力研发出自己的存储引擎，小厂一般没有这个实力，没法去研发出自己的存储引擎，但是 Oracle 又用不起，那么怎么办呢?

这几年兴起的分布式数据库中间件刚好可以很好的解决这个问题。Java 领域，类似的工具很多，例如 Sharding-JDBC 、MyCat 等，通过这些工具，可以很好的实现数据库分库分表，以及数据表的动态扩展、读写分离、分布式事务解决等。有了这些工具，极大的提高了 MySQL 的应用场景。

另一方面，近些年流行微服务，这不是单纯的炒概念，微服务架构将一个大的项目拆分成很多个小的微服务，各个微服务处理自己很小的一部分事情，这更符合人类分工协作的特点。在微服务架构中，我们对大表的需求、对多表联合查询的需求都会有所降低，MySQL 也更具用武之地。

因此，第二个问题也是可以解决的。

据松哥了解，互联网公司使用 MySQL 还是比较多的，传统软件公司，可能会更青睐 Oracle 等数据库。

不过话说回来，云计算，也是未来一个方向。

结语

为什么要写这篇文章呢？因为松哥打算出几篇文章给大家介绍一下分布式数据库中间件 MyCat 和 Sharding-JDBC 的用法，有了这些分布式数据库中间件，就可以让你的 MySQL 真正具备可以媲美大型数据库的能力。本文算是一个引子吧。

后面松哥就先更新 MyCat 。

4.存千万级数据

千万量级的数据，用 MySQL 要怎么存?

初学者在看到这个问题的时候，可能首先想到的是 MySQL 一张表到底能存放多少条数据?

根据 MySQL 官方文档的介绍，MySQL 理论上限是 (232)2 条数据，然而实际操作中，往往还受限于下面两条因素：

1. myisam_data_pointer_size，MySQL 的 myisam_data_pointer_size 一般默认是 6，即 48 位，那么对应的行数就是 248-1。
2. 表的存储大小 256TB

那有人会说，只要我的数据大小不超过上限，数据行数也不超过上限，是不是就没有问题了？其实不尽然。

在实际项目中，一般没有哪个项目真的触发到 MySQL 数据的上限了，因为当数据量变大了之后，查询速度会慢的吓人，而一般这个时候，你的数据量离 MySQL 的理论上限还远着呢!

传统的企业应用一般数据量都不大，数据也都比较容易处理，但是在互联网项目中，上千万、上亿的数据量并不鲜见。在这种时候，还要保证数据库的操作效率，我们就不得不考虑数据库的分库分表了。

那么接下来就和大家简单聊一聊数据库分库分表的问题。

数据库切分

看这个名字就知道，就是把一个数据库切分成 N 多个数据库，然后存放在不同的数据库实例上面，这样做有两个好处：

1. 降低单台数据库实例的负载
2. 可以方便的实现对数据库的扩容

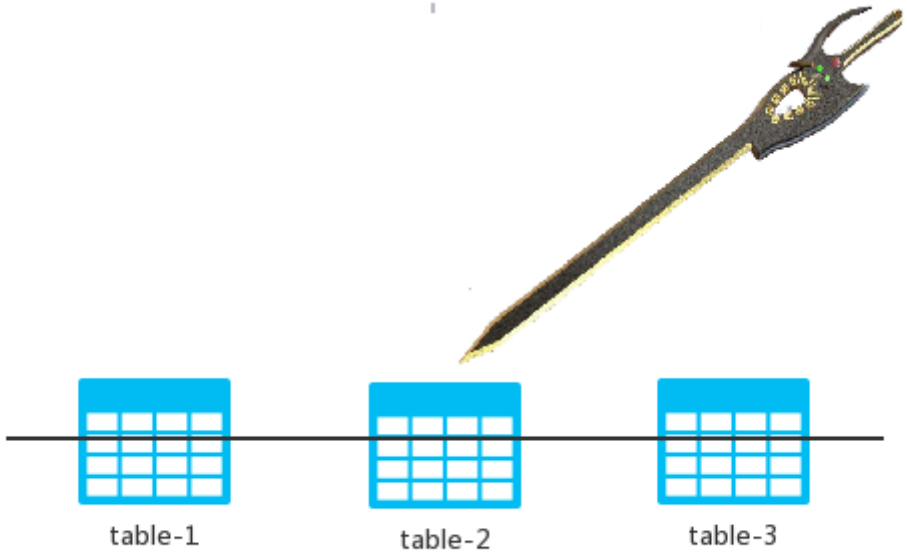
一般来说，数据库的切分有两种不同的切分规则：

1. 水平切分
2. 垂直切分

接下来我们就对这两种不同的切分规则分别进行介绍。

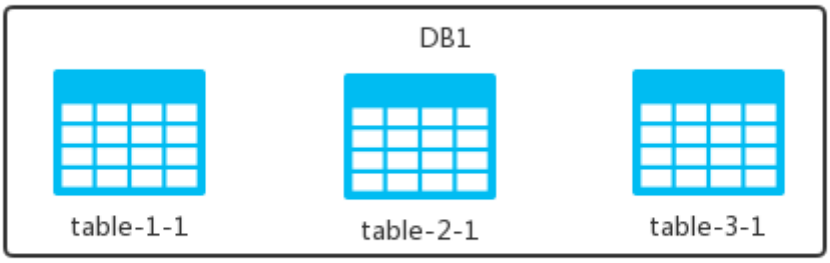
水平切分

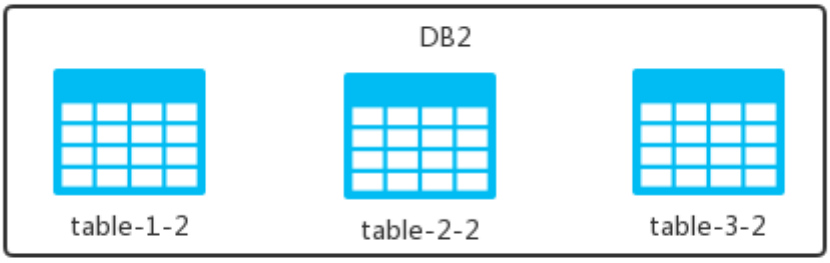
先来一张简单的示意图，大家感受一下什么是水平切分：



假设我的 DB 中有 table-1、table-2 以及 table-3 三张表，水平切分就是拿着我的绝世好剑，对准黑色的线条，砍一剑或者砍 N 剑！

砍完之后，将砍掉的部分放到另外一个数据库实例中，变成下面这样：





这样，原本放在一个 DB 中的 table 现在放在两个 DB 中了，观察之后我们发现：

1. 两个 DB 中表的个数都是完整的，就是原来 DB 中有几张表，现在还是几张。
2. 每张表中的数据是不完整的，数据被拆分到了不同的 DB 中去了。

这就是数据库的水平切分，也可以理解为按照数据行进行切分，即按照表中某个字段的**某种规则**来将表数据分散到多个库之中，每个表中包含一部分数据。

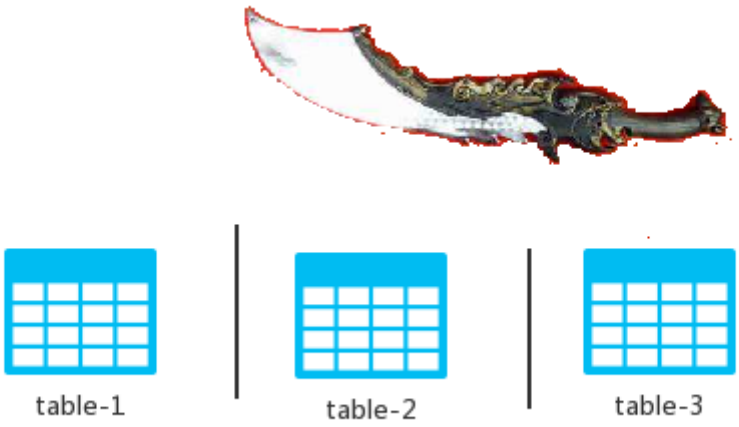
这里的某种规则都包含哪些规则呢？这就涉及到数据库的分片规则问题了，这个松哥在后面的文章中也会和大家一一展开详述。这里先简单说几个常见的分片规则：

1. 按照日期划分：不容日期的数据存放到不同的数据库中。
2. 对 ID 取模：对表中的 ID 字段进行取模运算，根据取模结果将数据保存到不同的实例中。
3. 使用一致性哈希算法进行切分。

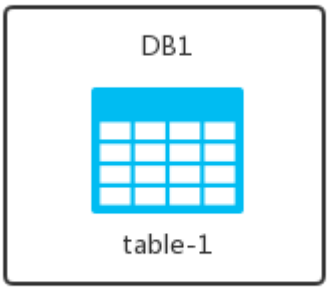
详细的用法，将在后面的文章中和大家仔细说。

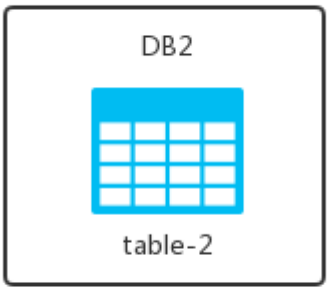
垂直切分

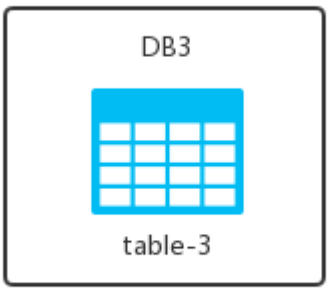
先来一张简单的示意图，大家感受一下垂直切分：



所谓的垂直切分就是拿着我的屠龙刀，对准了黑色的线条砍。砍完之后，将不同的表放到不同的数据库实例中去，变成下面这个样子：







这个时候我们发现如下几个特点：

1. 每一个数据库实例中的表的数量都是不完整的。
2. 每一个数据库实例中表的数据是完整的。

这就是垂直切分。一般来说，垂直切分我们可以按照业务来划分，不同业务的表放到不同的数据库实例中。

老实说，在实际项目中，数据库垂直切分并不是一件容易的事，因为表之间往往存在着复杂的跨库 JOIN 问题，那么这个时候如何取舍，就要考验架构师的水平了！

优缺点分析

通过上面的介绍，相信大家对于水平切分和垂直切分已经有所了解，优缺点其实也很明显了，松哥再来和大家总结一下。

水平切分

- 优点

1. 水平切分最大的优势在于数据库的扩展性好，提前选好切分规则，数据库后期可以非常方便的进行扩容。
2. 有效提高了数据库稳定性和系统的负载能力。拆分规则抽象好， join 操作基本可以数据库做。

- 缺点

1. 水平切分后，分片事务一致性不容易解决。
2. 拆分规则不易抽象，对架构师水平要求很高。
3. 跨库 join 性能较差。

垂直切分

- 优点

1. 一般按照业务拆分，拆分后业务清晰，可以结合微服务一起食用。
2. 系统之间整合或扩展相对要容易很多。
3. 数据维护相对简单。

- 缺点

1. 最大的问题在于存在单库性能瓶颈，数据表扩展不易。
2. 跨库 join 不易。
3. 事务处理复杂。

结语

虽然 MySQL 中数据存储的理论上限比较高，但是在实际开发中我们不会等到数据存不下的时候才去考虑分库分表问题，因为在那之前，你就会明显的感觉到数据库的各项性能在下降，就要开始考虑分库分表了。

好了，今天主要是向大家介绍一点概念性的东西，算是我们分布式数据库中间件正式出场前的一点铺垫。

5.Tomcat也算中间件

关于 MyCat 的铺垫文章已经写了两篇了：

1. MySQL 只能做小项目？松哥要说几句公道话！
2. 北冥有 Data，其名为鲲，鲲之大，一个 MySQL 放不下！

今天是最后一次铺垫，后面就可以迎接大 Boss 了！

本来今天就该讲 MyCat 了，但是我发现还有一个概念值得和大家聊一下，那就是 Java 中间件！

因为 MyCat 是一个**分布式数据库中间件**，要理解 MyCat ，那你就得先知道到底什么是中间件！

松哥去年在一次外训中专门讲过中间件，本来想直接和大家分享一下讲稿，但是没找到，所以又动手敲了下。

中间件简介

说起中间件，很多人首先想到的就是消息中间件，那么除了消息中间件呢？其实我们日常开发中，接触到的中间件太多了，我们来看维基百科上的一段介绍：

> 中间件（英语：Middleware），又译中间件、中介层，是提供系统软件和应用软件之间连接的软件，以便于软件各部件之间的沟通。在现代信息技术应用框架如 Web 服务、面向服务的体系结构等项目中应用比较广泛。如数据库、Apache 的 Tomcat ， IBM 公司的 WebSphere ,BEA 公司的 WebLogic 应用服务器，东方通公司的 Tong 系列中间件，以及 Kingdee 公司的等都属于中间件。

看到这个，你可能会大吃一惊，原来我们不知不觉不知不觉中已经用过这么多中间件了！甚至连 Tomcat 也是一个中间件！

中间件，顾名思义，就是连接在两个软件之间的东西，是软件之间的一个粘合剂，一个胶水一样的东西。它位于操作系统和我们的应用程序之间，可以让开发者方便地处理通信、输入和输出，使开发者能够专注于自己的业务逻辑开发。

这么一说，好像 Tomcat 确实还有点像中间件！位于我们的操作系统和应用程序之间！

中间件分类

中间件有很多，早在 1998 年 IDC 公司就将中间件分成了 6 大类，国内 2005 年之前出版的中间件相关的书上，很多都是按照这 6 大类来分的，分别是：

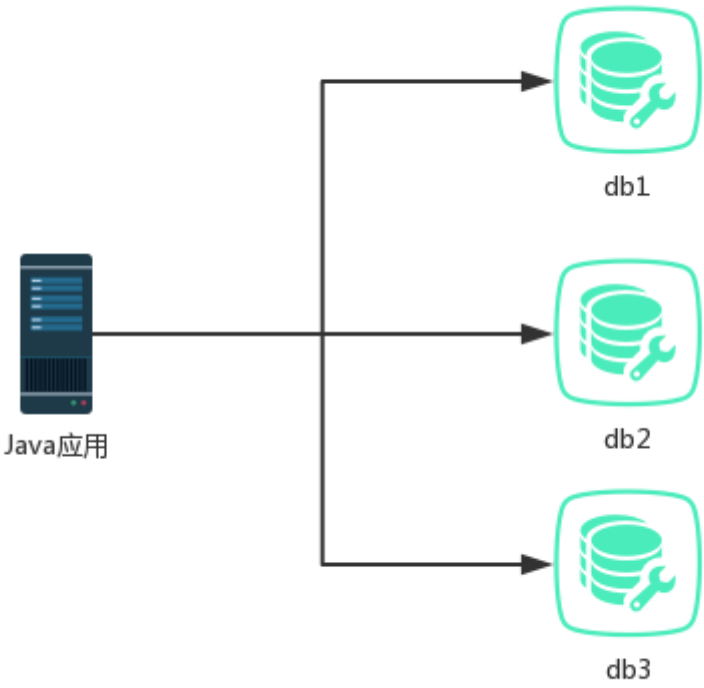
1. 终端仿真/屏幕转换
2. 数据访问中间件（UDA）
3. 远程过程调用中间件（RPC）
4. 消息中间件（MOM）
5. 交易中间件（TPM）
6. 对象中间件

这里边除了消息中间件和交易中间件大家可能听说过之外，其他的中间件估计都很少听说，这是因为时代在变化，有的中间件慢慢被淘汰了（例如 终端仿真/屏幕转换 中间件），有的则慢慢合并到其他框架中去了（例如 远程过程调用中间件）。

数据库中间件

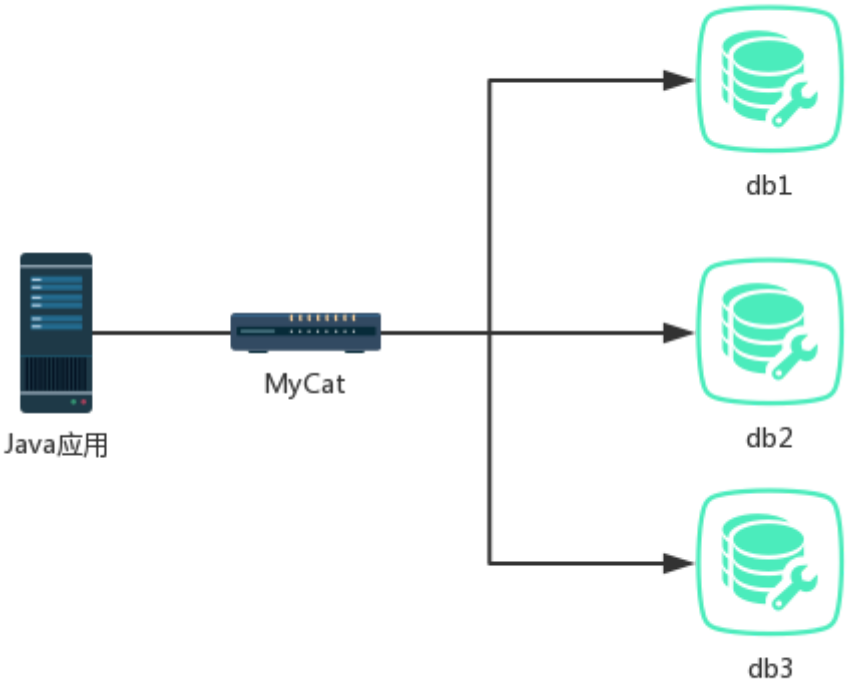
那么什么是数据库中间件呢？

前面文章我们提到，如果数据量比较大的话，我们需要对数据进行分库分表，分完之后，原本存在一个数据库中的数据，现在就存在多个数据库中了，那么我们的项目结构可能就是下面这个样子了：



我们要在 Java 代码中配置复杂的多数据源，配置读写分离，数据查询的时候还要进行数据的预处理，例如从多个 DB 上加载到的数据要先进行排序、过滤等等操作，这样我们的 Java 代码就参杂了很多业务无关的方法，而且这些参杂进来的代码，大多数都还是重复的。

为了使开发人员，将更多精力放到业务上，我们引入数据库中间件，像下面这样：



这张图非常形象的说明了什么是中间件！一个介于两个应用程序之间的东西。引入 MyCat 中间件之后，我们的应用程序将只需要连接 MyCat 就行了，再由 MyCat 去操作各种不同的 DB，各个分布式数据库的排序、结果集合并、数据过滤等操作都在 MyCat 中完成，这样我们的 Java 应用又可以专注于业务的开发了，那些繁琐的重复的操作，又交给 MyCat 去完成。

如果没有数据库中间件，那么我们的 Java 应用程序将直接面对分片集群，数据源切换、事务处理、数据聚合等等众多问题，这样原本该是专注于业务的 Java 应用程序，将会花大量的工作来处理分片后的问题，而且大部分的代码又都是重复的！

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，所以在项目中选择一款好的数据库中间件至关重要。

6.中间件 MyCat

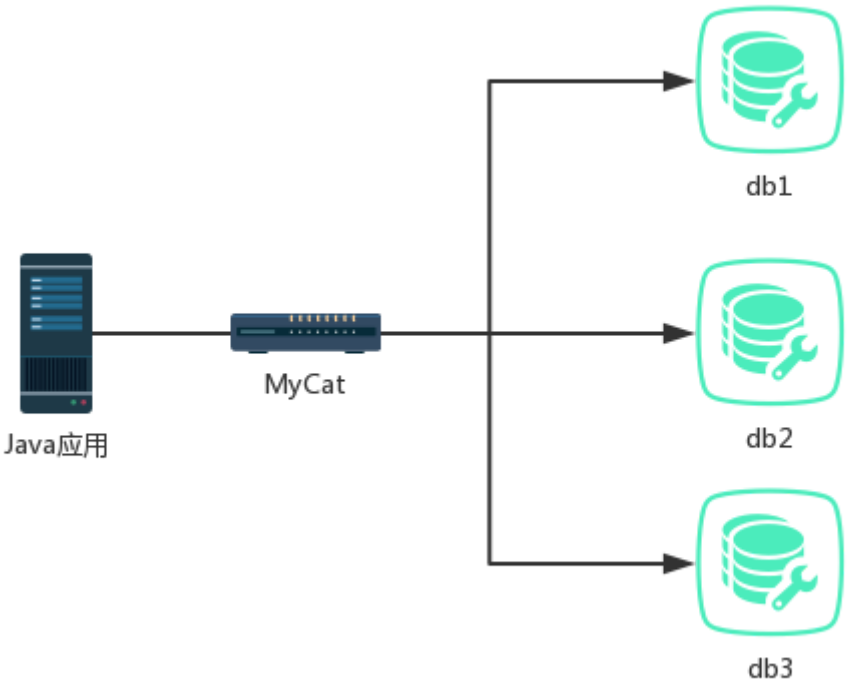
关于 MyCat 的铺垫文章已经写了三篇了：

1. MySQL 只能做小项目？松哥要说几句公道话！
2. 北冥有 Data，其名为鲲，鲲之大，一个 MySQL 放不下！
3. What？Tomcat 竟然也算中间件？

今天终于可以迎接我们的大 Boss 出场了！

MyCat 简介

前面文章我们提到，如果数据量比较大的话，我们需要对数据进行分库分表，分完之后，原本存在一个数据库中的数据，现在就存在多个数据库中了，就像下面这样：



那么此时 MyCat 所扮演的角色就是分布式数据库中间件！

MyCat 是一个开源的分布式数据库中间件，它实现了 MySQL 协议，在开发者眼里，他就是一个数据库代理，我们甚至可以使用 MySQL 的客户端工具以及命令行来访问 MyCat 。

MyCat 现在已经不仅仅只支持 MySQL 了，同时也支持 MSSQL、Oracle、DB2、以及 PostgreSQL等主流数据库。甚至像 MongoDB 这种 NoSQL 也支持。

快速入门

搭建读写分离

要搞 MyCat，一般要先搭建好 MySQL 的读写分离，MySQL 的读写分离可以参考松哥之前的这篇文章：

1. 提高性能，MySQL 读写分离环境搭建(二)

MyCat 安装

环境：

- CentOS7
- JDK1.8

MyCat 使用 Java 开发，因此，运行 MyCat，一定要具备 Java 环境，配置 Java 运行环境这个比较容易，网上资料也很多，我就不详细介绍了。

Java 环境安装好之后，首先下载 MyCat：

```
wget http://dl.mycat.io/1.6.7.1/Mycat-server-1.6.7.1-release-20190213150257-linux.tar.gz
```

下载完成后，对下载文件进行解压。

```
tar -zxvf Mycat-server-1.6.7.1-release-20190213150257-linux.tar.gz
```

解压成功后，会出现一个 `mycat` 目录，进入到 `mycat/conf` 目录，对 `mycat` 进行配置：

首先来配置 `schema.xml` 文件：

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
                          writeType="0" dbType="mysql" dbDriver="native" switchType="1"  slaveThreshold="100">
                <heartbeat>select user()</heartbeat>
                <!-- can have multi write hosts -->
                <writeHost host="hostM1" url="localhost:3306" user="root"
                                   password="123">
                        <!-- can have multi read hosts -->
                        <readHost host="hostS2" url="192.168.66.131:3306" user="repl" password="123" />
                </writeHost>
        </dataHost>
```

1. 首先在 `schema` 中指定逻辑库的名字，逻辑库是指 `MyCat` 中的库，这个库不存储数据，数据存储在 `MySQL` 中的物理库中。
2. 逻辑库中配置逻辑表，配置逻辑表时，需要指定 `dataNode` 节点， `dataNode` 就是指数据库存储的位置
3. 配置 `dataNode` ， `dataNode` 指定 `dataHost` 和物理库的名字。
4. `dataHost` 则配置 `MySQL` 的主机和从机的位置，登录密码等。主机和从机都可以配置多个。

配置完 schema.xml 后，接下来配置 server.xml。

server.xml 中主要配置 MyCat 的登录用户名和密码，以及需要操作的逻辑库。

```
<user name="root" defaultAccount="true">
        <property name="password">123456</property>
        <property name="schemas">javaboy</property>

        <!-- 表级 DML 权限设置 -->
        <!--
        <privileges check="false">
                <schema name="TESTDB" dml="0110" >
                        <table name="tb01" dml="0000"></table>
                        <table name="tb02" dml="1111"></table>
                </schema>
        </privileges>
         -->
</user>

<user name="user">
        <property name="password">user</property>
        <property name="schemas">javaboy</property>
        <property name="readOnly">true</property>
</user>
```

配置完成后，接下来就可以启动 MyCat 了 。

执行 MyCat 解压目录下的 bin 目录下的 mycat 命令，可以启动 MyCat

```
./bin/mycat start
```

如果启动后，提示无法创建 `mycat.pid` 文件，就自己手动创建一个 `mycat.pid` 文件。启动成功之后，就可以在本地连接 `MyCat` 了，连接方式和 `MySQL` 一样，唯一的区别在于端口号不同。

在连接 `MyCat` 之前，先在 `MySQL` 物理库中创建 `db1`、`db2` 以及 `db3` 三个数据库。

使用 `SQLyog` 连接：

也可以在 `cmd` 命令行登录 `MyCat`：

```
C:\Users\sang>mysql -u root -h 192.168.66.128 -P 8066 -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.29-mycat-1.6.7.1-release-20190213150257 MyCat Server (OpenCloudDB)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

登录成功后，在 `MyCat` 的窗口中，执行如下命令，创建表：

```
create table t_user (id integer primary key,username varchar(255))
```

执行成功后，我们会发现物理库中出现了相应的表。
接下来，手动往各个物理库的物理表中存储一条数据，然后在 MyCat 窗口中查询：

```
2
3     select  * from t_user
```

1 结果 | 2 个配置文件 | 3 信息

（只读）

id	username
3	lisi
1	javaboy
2	zhangsan

这样就可以查询到 三个库中的三个表中的数据。

问题分析

整个过程不难，但是有的小伙伴在第一次配置的过程中还是容易出错，因此我这里还是来说两句，出错了要如何定位。

一般来说，配置 MyCat 出错，问题可能发生在两个阶段。第一个阶段就是客户端连接 MyCat 出错，第二个阶段就是 MyCat 连接 MySQL 出错。

无论你是使用 SQLyog 还是 Navicat，我们在连接数据库的过程中，都可以先测试连接，很多人卡在这一步。

如果在测试连接的时候就连接不通，说明是 MyCat 的问题，这个时候检查步骤如下：

1. 首先当然是查看日志信息，看能不能找出端倪
2. 通过 jps 命令查看 mycat 是否成功启动
3. 检查 server.xml 中配置是否正确，用户名密码是否输入正确

这是第一种可能的问题，第二种问题就是测试连接没问题，但是测试完后，却连接不上。反映到 Navicat 上，就是测试连接没问题，测完之后，点击连接名要打开连接时，Navicat 就崩了，出现这个问题一般是 MyCat 在连接 MySQL 出问题了，这个时候就要去检查 schema.xml 文件中关于 MySQL 主机和从机的配置是否正确，数据库地址是否正确，用户名密码是否正确。

结语

好了，本文主要简单介绍了下 MyCat 的安装问题，下篇文章我们来看 MyCat 中的分片规则问题。

7.分库分表的分片规则

上次和大伙聊了 MyCat 的安装，今天来说一个新的话题，就是数据库的分片。

当我们把 MyCat + MySQL 的架构搭建完成之后，接下来面临的一个问题就是，数据库的分片规则：有那么多 MySQL ，一条记录通过 MyCat 到底要插入到哪个 MySQL 中？这就是我们今天要讨论的问题。

关于数据库分库分表的问题，我们前面还有几篇铺垫的文章，阅读前面的文章有助于更好的理解本文：

---

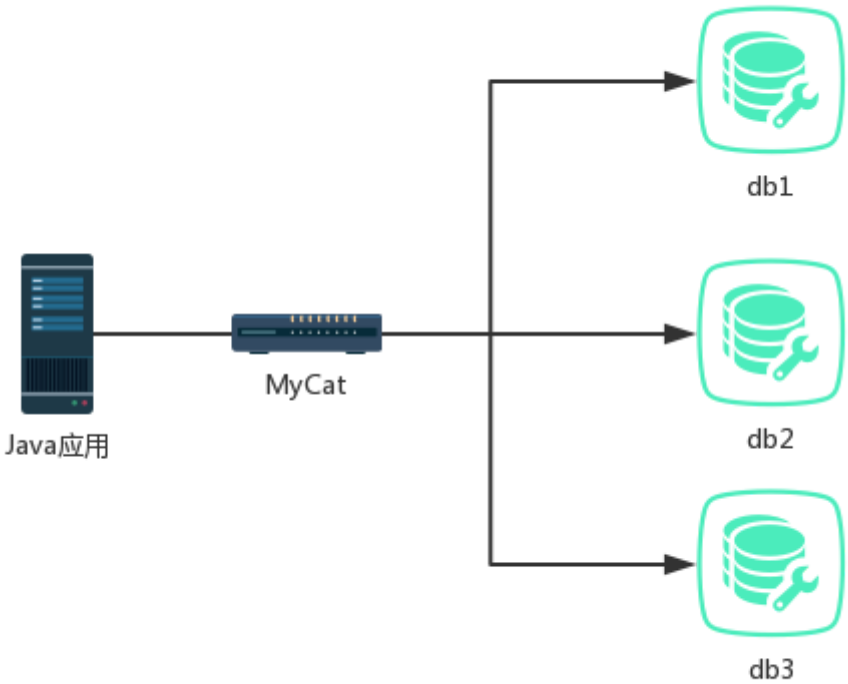
1. 提高性能，MySQL 读写分离环境搭建(一)
2. 提高性能，MySQL 读写分离环境搭建(二)
3. MySQL 只能做小项目？松哥要说几句公道话！
4. 北冥有 Data，其名为鲲，鲲之大，一个 MySQL 放不下！
5. What？Tomcat 竟然也算中间件？
6. 分布式数据库中间件 MyCat 搞起来！

基本概念

逻辑库

一般来说，对于应用而言，数据库中间件是透明的，应用并不需要去了解中间件复杂的运作过程，中间件对应用来说就是透明的，我们操作中间件就像操作一个普通的 MySQL 一样，这就是 MyCat 的优势之一。

但是我们毕竟操作的不是 MySQL ，而是 MyCat ，MyCat 中的数据库并不真正存储数据，数据还是存储在 MySQL 中，因此，我们可以将 MyCat 看作是一个或者多个数据库集群构成的逻辑库。



逻辑表

逻辑表又有几种不同的划分：

- 逻辑表

既然有逻辑库，那么就会有逻辑表。

因为数据库分片之后，本来存储在一张表中的数据现在被分散到 N 张表中去了，但是在应用程序眼里，还是只有一张表，它也只操作这一张表，这张表并不真正存储数据，数据存储在 N 张物理表中，这个并不真正存储数据的表称之为逻辑表。

- 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。

- 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。

- ER 表

关系型数据库是基于实体关系模型之上，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 关系的数据分片策略，子表的记录与所关联的父表记录存放在同一个数据分片上，即子表依赖于父表，通过表分组保证数据 join 不会跨库操作。

表分组是解决跨分片数据 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：

- 变动不频繁
- 数据量总体变化不大
- 数据规模不大，很少有超过数十万条记录

对于这类的表，在分片的情况下，当业务表因为规模而进行分片以后，业务表与这些附属的字典表之间的关联，就成了比较棘手的问题，所以 MyCat 中通过数据冗余来解决这类表的 join ，即所有的分片都有一份数据的拷贝，所有将字典表或者符合字典表特性的一些表定义为全局表。

数据冗余是解决跨分片数据 join 的一种很好的思路，也是数据切分规划的另外一条重要规则。

分片节点

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

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

前面讲了数据切分，一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。

MyCat 提供的分片规则有如下几种：

- 分片枚举
- 固定分片 hash 算法
- 范围约定
- 取模
- 按日期（天）分片
- 取模范围约束
- 截取数字做 hash 求模范围约束
- 应用指定
- 截取数字 hash 解析
- 一致性 hash
- 按单月小时拆分
- 范围求模分片
- 日期范围 hash 分片
- 冷热数据分片
- 自然月分片

实践

这里向大家简单介绍 5 种规则。

global

有一些表，数据量不大，也不怎么修改，主要是查询操作，例如系统配置表，这一类表我们可以使用 global 这种分片规则。global 的特点是，该表会在所有的库中都创建，而且每一个库中都保存了该表的完整数据。具体配置方式，就是在 schema.xml 的 table 节点中添加一个 type 属性，值为 global：

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" type="global" rule="auto-sharding-long" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
```

配置完成后，重启 mycat

```
./bin/mycat restart
```

重启完成后，要删除之前已经创建的 `t_user` 表，然后重新创建表，创建完成后，向表中插入数据，可以看到，db1、db2 以及 db3 中都有数据了。

```
自动完成： [Tab]-> 下一个标签， [Ctrl+Space]-> 列出所有标签， [Ctrl+Enter]-> 列出匹配标签
1     drop table if exists t_user;
2     create table t_user (id integer primary key,username varchar(255));
3     insert into t_user(id,username) values(1,'www.javaboy.org');
4     select  * from t_user;
```

这里 虽然查询出来的记录只有一条，实际上 db1、db2 以及 db3 中都有该条记录。

总结：global 适合于 数据量不大、以查询为主、增删改较少的表。

sharding-by-intfile

sharding-by-intfile 这个是枚举分片，就是在数据表中专门设计一个字段，以后根据这个字段的值来决定数据插入到哪个 dataNode 上。

注意，在配置 sharding-by-intfile 规则时，一定要删除 type=”global”，否则配置不会生效。具体配置如下：

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="sharding-by-intfile" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
```

配置完成后，还需要指定枚举的数据。枚举的数据可以在 rule.xml 中查看。

```
<tableRule name="sharding-by-intfile">
        <rule>
                <columns>sharding_id</columns>
                <algorithm>hash-int</algorithm>
        </rule>
</tableRule>
```

在 rule.xml 文件中，首先找到 tableRule 的名字为 sharding-by-intfile 的节点，这个节点中定义了两个属性，一个是 columns 表示一会在数据表中定义的枚举列的名字（数据表中一会需要创建一个名为 sharding_id 的列，这个列的值决定了该条数据保存在哪个数据库实例中），这个名字可以自定义；另外一个属性叫做 algorithm ，这是指 sharding-by-intfile 所对应的算法名称。根据这个名称，可以找到具体的算法：

```
<function name="hash-int"
        class="io.mycat.route.function.PartitionByFileMap">
        <property name="mapFile">partition-hash-int.txt</property>
</function>
```

还是在 rule.xml 文件中，我们找到了 hash-int ， class 表示这个算法对应的 Java 类的路径。第一个属性 mapFile 表示相关的配置文件，从这个文件名可以看出，这个文件 就在 conf 目录下。

打开 conf 目录下的 partition-hash-int.txt 文件，内容如下：

```
0=0
1=1
2=2
```

前面的数字表示枚举的值，后面的数字表示 dataNode 的下标，所以前面的数字可以自定义，后面的数字不能随意定义。

配置完成后，重启 MyCat，然后进行测试：

```
drop table if exists t_user;
create table t_user (id integer primary key,username varchar(255),sharding_id
integer);
insert into t_user(id,username,sharding_id) values(1,'www.javaboy.org',0);
insert into t_user(id,username,sharding_id) values(1,'www.javaboy.org',1);
insert into t_user(id,username,sharding_id) values(1,'www.javaboy.org',2);
select  * from t_user;
```

执行完后，sharding_id 对应值分别为 0、1、2 的记录分别插入到 db1、db2 以及 db3 中。

auto-sharding-long

auto-sharding-long 表示按照既定的范围去存储数据。就是提前规划好某个字段的值在某个范围时，相应的记录存到某个 dataNode 中。

配置方式，首先修改路由规则：

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
```

然后去 rule.xml 中查看对应的算法了规则相关的配置：

```
<tableRule name="auto-sharding-long">
        <rule>
                <columns>id</columns>
                <algorithm>rang-long</algorithm>
        </rule>
</tableRule>
```

可以看到，默认是按照 id 的范围来划分数据的存储位置的，对应的算法就是 rang-long 。

继续查看，可以找到算法对应的类，以及相关的配置文件，这个配置文件也在 conf 目录下，打开该文件：

```
# range start-end ,data node index
# K=1000,M=10000.
0-5=0
5-10=1
10-1500M=2
```

如上配置，表示 当 id 的取值在 0-5之间时，将数据存储到 db1 中，当 id 在 5-10 之间时，存储到 db2 中，当 id 的取值在 10-1500W 之间时，存储到 db3 中。

配置完成后，重启 MyCat，测试：

```
自动完成：[Tab]-> 下一个标签，[Ctrl+Space]-> 列出所有标签，[Ctrl+Enter]-> 列出匹配标签
1     drop table if exists t_user;
2     create table t_user (id integer primary key,username varchar(255));
3     insert into t_user(id,username) values(5,'www.javaboy.org');
4     insert into t_user(id,username) values(9,'www.javaboy.org');
5     insert into t_user(id,username) values(10000,'www.javaboy.org');
6     select  * from t_user;
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

（只读）

id	username
5	www.javaboy.org
9	www.javaboy.org
10000	www.javaboy.org

mod-long

取模：根据表中的某一个字段，做取模操作。根据取模的结果将记录存放在不同的 dataNode 上。这种方式不需要再添加额外字段。

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="mod-long" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
```

然后去 rule.xml 中配置一下 dataNode 的个数。

```
<tableRule name="mod-long">
        <rule>
                <columns>id</columns>
                <algorithm>mod-long</algorithm>
        </rule>
</tableRule>
```

可以看到，取模的字段是 id ，取模的算法名称是 mod-long ，再看具体的算法：

```
<function name="mod-long" class="io.mycat.route.function.PartitionByMod">
        <!-- how many data nodes -->
        <property name="count">3</property>
</function>
```

在具体的算法中，配置了 dataNode 的个数为 3。

然后保存退出，重启 MyCat，进行测试：

```
自动完成：[Tab]-> 下一个标签，[Ctrl+Space]-> 列出所有标签，[Ctrl+Enter]-> 列出匹配标签
1     drop table if exists t_user;
2     create table t_user (id integer primary key,username varchar(255));
3     insert into t_user(id,username) values(5,'www.javaboy.org');
4     insert into t_user(id,username) values(9,'www.javaboy.org');
5     insert into t_user(id,username) values(10000,'www.javaboy.org');
6     select  * from t_user;
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

（只读）

id	username
5	www.javaboy.org
9	www.javaboy.org
10000	www.javaboy.org

sharding-by-murmur

前面介绍的几种方式，都存在一个问题，如果数据库要扩容，之前配置会失效，可能会出现数据库查询紊乱。因此我们要引入一致性 hash 这样一种分片规则，可以解决这个问题。具体配置和前面一样：

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="sharding-by-murmur" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
```

另外需要注意，在 rule.xml 中修改默认 dataNode 的数量：

```
        <function name="murmur"
                class="io.mycat.route.function.PartitionByMurmurHash">
                <property name="seed">0</property><!-- 默认是0 -->
                <property name="count">3</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，
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
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>

        <function name="crc32slot"
```

修改完后，重启 MyCat ，进行测试。

好了，本文主要向大家介绍了 MyCat 的五种不同的切片规则。有问题欢迎留言讨论。

8.分布式主键局部自增

前面和大家介绍了 MyCat 中数据库不同的分片规则，从留言中看出大家对分布式数据库中间件还挺感兴趣，因此今天就再来一篇，聊一聊主键全局自增要如何实现。

关于数据库分库分表的问题，我们前面还有几篇铺垫的文章，阅读前面的文章有助于更好的理解本文：

1. 提高性能，MySQL 读写分离环境搭建(一)
2. 提高性能，MySQL 读写分离环境搭建(二)
3. MySQL 只能做小项目？松哥要说几句公道话！
4. 北冥有 Data，其名为鲲，鲲之大，一个 MySQL 放不下！
5. What？Tomcat 竟然也算中间件？
6. 分布式数据库中间件 MyCat 搞起来！
7. 数据库分库分表，分片配置轻松入门！

问题

主键自增这应该算是一个非常常见的需求，在单机数据库中，这个需求一个 `auto_increment` 就能实现，但是在数据库集群中，这个需求却变复杂了，因为存在多个数据库实例，各自都是主键自增，合在一起就不是主键自增了。

最简单的思路

最简单的办法莫过于通过设置主键自增的步长和起始偏移量来处理这个问题。默认情况下，主键自增步长为 1，如果我们有三个数据库实例，我们可以将主键自增步长设置为 3，这样对于第一个数据库实例而言，主键自增就是 1、4、7、10...，对于第二个数据库实例而言，主键自增就是 2、5、8、11...，对于第三个数据库实例而言，主键自增就是 3、6、9、12....。

MSSQL 可以直接在 SQL 中指定主键的自增步长和起始偏移量，但是 MySQL 则需要修改数据库配置才能实现，因此这里不推荐使用这种方式。

MyCat 的办法

MyCat 作为一个分布式数据库中间，屏蔽了数据库集群的操作，让我们操作数据库集群就像操作单机版数据库一样，对于主键自增，它有自己的方案：

1. 通过本地文件实现
2. 通过数据库实现
3. 通过本地时间戳实现
4. 通过分布式 ZK ID 生成器实现
5. 通过 ZK 递增方式实现

今天我们就先来看看看如何通过 ZK 递增的方式实现主键全局自增。

配置步骤如下：

- 首先修改主键自增方式为 4，4 表示使用 zookeeper 实现主键自增。

server.xml

```
<mycat:server xmlns:mycat="http://io.mycat/">
        <system>
        <property name="nonePasswordLogin">0</property> <!-- 0为需要密码登陆、1为不需要密码登陆 ,默认为0，设置为1则需要指定默认账户-->
        <property name="useHandshakeV10">1</property>
        <property name="useSqlStat">0</property>  <!-- 1为开启实时统计、0为关闭 -->
        <property name="useGlobleTableCheck">0</property>  <!-- 1为开启全加班一致性检测、0为关闭 -->

                <property name="sequnceHandlerType">4</property>
        <property name="subqueryRelationshipCheck">false</property> <!-- 子查询中存在关联查询的情况下,检查关联字段中是否有分片字段 .默认
-->
     <!--  <property name="useCompression">1</property>--> <!--1为开启mysql压缩协议-->
        <!--  <property name="fakeMySQLVersion">5.6.20</property>--> <!--设置模拟的MySQL版本号-->
        <!-- <property name="processorBufferChunk">40960</property> -->
        <!--
        <property name="processors">1</property>
        <property name="processorExecutor">32</property>
         -->
        <!--默认为type 0: DirectByteBufferPool | type 1 ByteBufferArena | type 2 NettyBufferPool -->
                <property name="processorBufferPoolType">0</property>
                <!--默认是65535 64K 用于sql解析时最大文本长度 -->
                <!--<property name="maxStringLiteralLength">65535</property>-->
                <!--<property name="sequnceHandlerType">0</property>-->
                <!--<property name="backSocketNoDelay">1</property>-->
```

- 配置表自增，并且设置主键

schema.xml

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

        <schema name="javaboy" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" autoIncrement="true" primaryKey="id" rule="sharding-by-murmur" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
                          writeType="0" dbType="mysql" dbDriver="native" switchType="1"  slaveThreshold="100">
                <heartbeat>select user()</heartbeat>
                <!-- can have multi write hosts -->
                <writeHost host="hostM1" url="localhost:3306" user="root"
                                   password="123">
```

设置主键自增，并且设置主键为 id 。

- 配置 zookeeper 的信息

在 myid.properties 中配置 zookeeper 信息：

```
loadZk=false
zkURL=127.0.0.1:21811,127.0.0.1:21812,127.0.0.1:21813
clusterId=mycat-cluster-1
myid=mycat_fz_01
clusterSize=3
clusterNodes=mycat_fz_01,mycat_fz_02,mycat_fz_04
#server  booster  ;  booster install on db same server,will reset all minCon to 2
type=server
boosterDataHosts=dataHost1
~
~
```

- 配置要自增的表

sequence_conf.properties

```
T_USER.MINID=1001
T_USER.MAXID=2000
T_USER.CURID=1000
```

注意，这里表名字要大写。

1. TABLE.MINID 某线程当前区间内最小值
2. TABLE.MAXID 某线程当前区间内最大值
3. TABLE.CURID 某线程当前区间内当前值
4. 文件配置的MAXID以及MINID决定每次取得区间，这个对于每个线程或者进程都有效
5. 文件中的这三个属性配置只对第一个进程的第一个线程有效，其他线程和进程会动态读取 ZK

- 重启 MyCat 测试

最后重启 MyCat ，删掉之前创建的表，然后创建新表进行测试即可。

9.数据库减压8个思路

传统的企业级应用，其实很少会有海量应用，因为企业的规模本身就摆在那里，能有多少数据？高并发？海量数据？不存在的！

不过在互联网公司中，因为应用大多是面向广大人民群众，数据量动辄上千万上亿，那么这些海量数据要怎么存储？光靠数据库吗？肯定不是。

今天松哥和大家简单的聊一聊这个话题。

海量数据，光用数据库肯定是没法搞定的，即使不读松哥这篇文章，相信大家也能凝聚这样的共识，海量数据，不是说一种方案、两种方案就能搞定，它是一揽子方案。那么这一揽子方案都包含哪些东西呢？松哥从以下八个方面来和大家聊聊。

1.缓存

首先第一种解决方案就是缓存了。

缓存，我们可以将数据直接缓存在内从中，例如 Map、也可以使用缓存框架如 Redis 等，将一些需要频繁使用的热点数据保存在缓存中，每当用户来访问的时候，就可以直接将缓存中的数据返回给用户，这样可以有效降低服务器的压力。

可以缓存起来使用的数据，一般都不能对实时性要求太高。

2.页面静态化

页面静态化其实可以算作是缓存的另外一种形式，相当于直接将相关的页面渲染结果缓存起来。首先大家知道，在我们的 Web 项目中，资源分为两大类：

- 静态资源
- 动态资源

静态资源就是我们常见的 HTML、CSS、JavaScript、图片等资源，这些资源可以不经过服务端处理，就可以直接返回给前端浏览器，浏览器就可以直接显示出来。

动态资源则是指我们项目中的 Servlet 接口、Jsp 文件、Freemarker 等，这些需要经过服务端渲染之后，才可以返回前端的资源。

在实际项目中，静态资源的访问速度要远远高于动态资源，动态资源往往很容易遇到服务器瓶颈、数据库瓶颈，因此，对于一些不经常更新的页面，或者说更新比较缓慢的页面，我们可以通过页面静态化，将一个动态资源保存为静态资源，这样当服务端需要访问的时候，直接将静态资源返回，就可以避免去操作数据库了，降低数据库的压力。

例如松哥以前做过的一个电商项目，系统根据大数据统计，自动统计出用户当前搜索的热点商品，这些热点商品，10 分钟更新一次，也就是说，在十分钟内，用户登录上来看到的热点商品都是相同的。那么就没有必要每次都去查询数据库，而是将热点数据的页面，通过输出流自动写到服务器上，写成一个普通的 HTML 文件，下次用户来访问，在 10 分钟有效期内，直接将 HTML 页面返回给用户，就不必操作数据库了。

一般来说，Freemarker、Velocity 等都有相关的方法可以帮助我们快速将动态页面生成静态页面。

这就是页面静态化。

3.数据库优化

很多时候程序跑得慢，不是因为设备落后，而是因为数据库 SQL 写的太差劲。

要解决海量数据的问题，数据库优化肯定也是不可避免的。一般来说，我们可以从 SQL 优化、表结构优化、以及数据库分区分表等多个方面来对数据库进行优化。数据库优化其实也是一门巨大的学问，松哥以后看有时间写个连载和大家仔细聊聊这个话题。

4.热点数据分离

数据库中的数据，虽然是海量数据，但是这些数据并不见得所有数据都是活跃数据，例如用户注册，有的用户注册完就消失的无影无踪了，而有的用户则在不停的登录，因此，对于这两种不同的用户，我们可以将活跃用户分离出来，在主要操作的数据表中只保存活跃用户数据。每次用户登录，先去主表中查看有没有记录，有的话，直接登录，没有的话，再去查看其他表。

通过判断用户在某一段时间内的登录次数，就可以很快分离出热点数据。

5.合并数据库操作

这个方案的宗旨其实是减少数据库操作的次数，例如多次插入操作，我们可以合并成一条 SQL 搞定。多个不同条件的查询，如果条件允许的话，也可以合并成为一个查询，尽量减少数据库的操作，减少在网络上消耗，同时也降低数据库的压力。

6.数据库读写分离

数据库的读写分离其实松哥在之前的 MyCat 中也和大伙聊过了（MyCat 系列），读写分离之后，一方面可以提高数据库的操作效率，另一方面也算是对数据库的一个备份。这一块的具体操作大家可以参考松哥前面的文章。

7.分布式数据库

数据库读写分离之后，无形中增大了代码的复杂度，所以一般还需要借助分布式数据库中间件，这样可以有效提高数据库的弹性，可以方便的随时为数据库扩容，同时也降低代码的耦合度。

8.NoSQL 和 Hadoop

另外，引入 NoSQL 和 Hadoop 也是解决方案之一。NoSQL 突破了关系型数据库中对表结构、字段等定义的条条框框，使用户可以非常灵活方便的操作，另外 NoSQL 通过多个存储块存储数据的特点，使得天然具备操作大数据的优势（快）。不过，老实说，NoSQL 目前还是在互联网项目中比较常见，在传统的企业级应用中还是比较少见。

Hadoop 就不必说了，大数据处理利器。

很多时候技术和架构只是一个工具，所有的东西都摆在你面前，关键是如何把这些东西组合在一起，使之产生最大化收益，这就需要大家慢慢琢磨，松哥后面也尽量和大家多分享一些这方面的经验。