1.分库分表技术之MyCat

**1.海量存储问题**

**1.1 背景描述**

如今随着互联网的发展，数据的量级也是成指数的增长，从GB到TB到PB。对数据的各种操作也是愈加的困难，传统的关系性数据库已经无法满足快速查询与插入数据的需求。

解决方案1: 使用NoSQL数据库, 通过降低数据的安全性，减少对事务的支持，减少对复杂查询的支持，来获取性能上的提升。

解决方案2: NoSQL并不是万能的，就比如有些使用场景是绝对要有事务与安全指标的, 所以还是要用关系型数据库, 这时候就需要搭建MySQL数据库集群,为了提高查询性能, 将一个数据库的数据分散到不同的数据库中存储, 通过这种数据库拆分的方法来解决数据库的性能问题。

**1.2 分库分表**

**1.2.1 什么是分库分表**

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。

**分库分表解决的问题：**

分库分表的目的是为了**解决由于数据量过大而导致数据库性能降低的问题**，将原来单体服务的数据库进行拆分.将数据大表拆分成若干数据表组成，使得单一数据库、单一数据表的数据量变小，从而达到提升数据库性能的目的。

**什么情况下需要分库分表：**

**（1）单机存储容量遇到瓶颈.**

**（2）连接数,处理能力达到上限.**

**注意:**

**分库分表之前,要根据项目的实际情况 确定我们的数据量是不是够大,并发量是不是够大,来决定是否分库分表；数据量不够就不要分表,单表数据量超过1000万或100G的时候, 速度就会变慢(官方测试)**

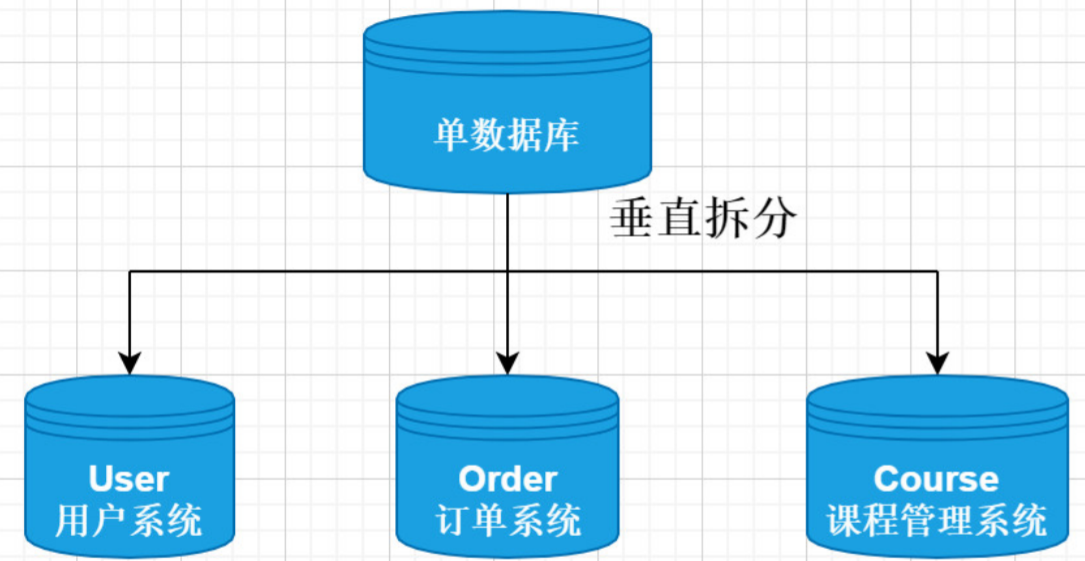
**1.2.2 分库分表的方式**

分库分表包括：**垂直分库、垂直分表、水平分库、水平分表**四种方式。

**1.2.2.1 垂直分库**

数据库中不同的表对应着不同的业务，垂直切分是指按照业务的不同将表进行分类,分布到不同的数据库上面

将数据库部署在不同服务器上，从而达到多个服务器共同分摊压力的效果



**1.2.2.2 垂直分表**

将一个表按照字段分成多表，每个表存储其中一部分字段。

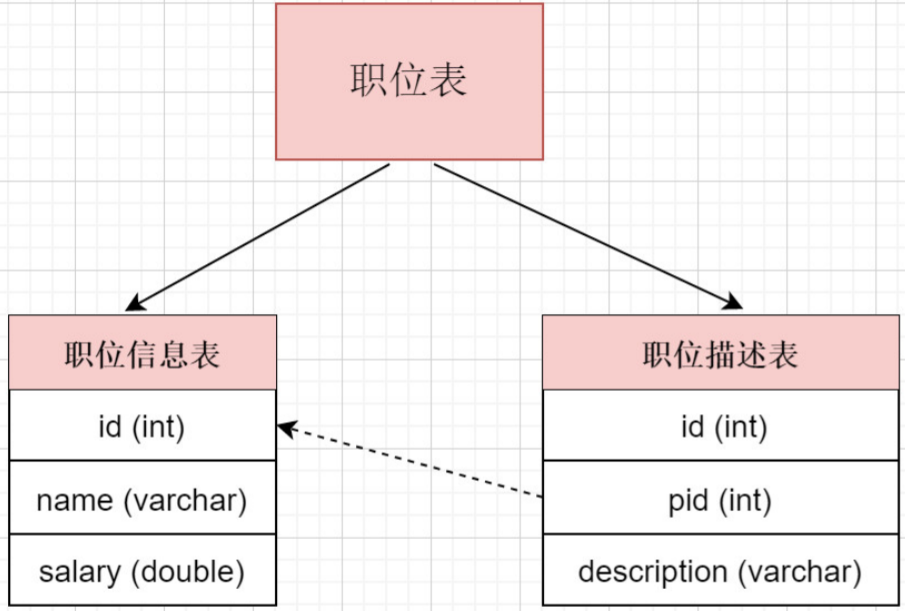
垂直拆分带来的一些提升：

（1）解决业务层面的耦合，业务清晰

（2）能对不同业务的数据进行分级管理、维护、监控、扩展等

（3）高并发场景下，垂直分库一定程度的提高访问性能

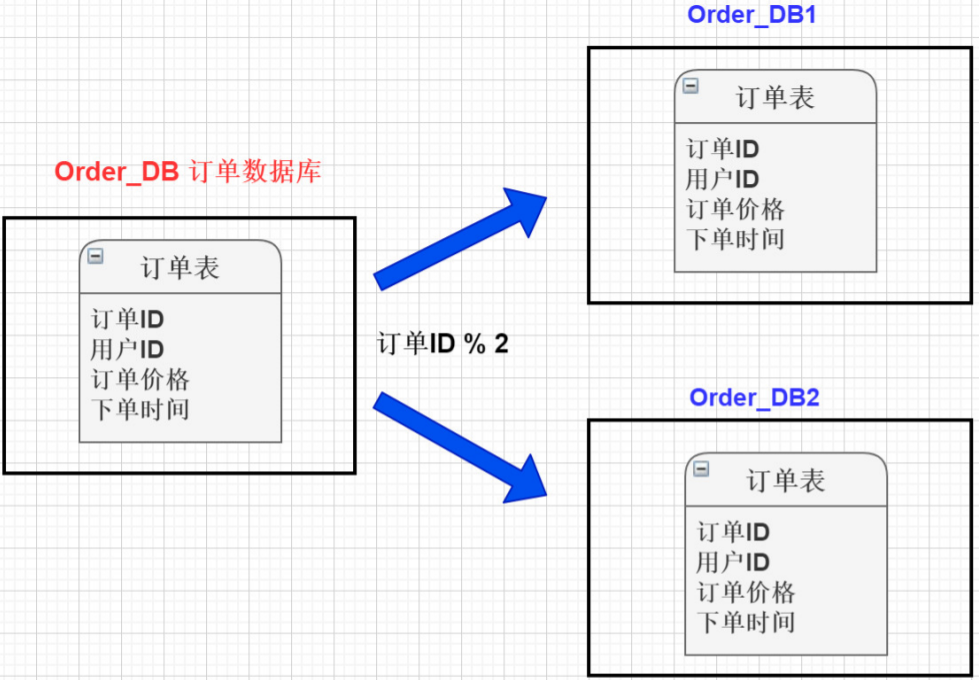
**垂直拆分没有彻底解决单表数据量过大的问题**



**1.2.2.3 水平分库**

将单张表的数据切分到**不同的数据库**中，每个数据库具有相同的库与表，只是表中数据集合不同。

简单讲就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, 例如将订单表按照id是奇数还是偶数, 分别存储在不同的库中。



**1.2.2.4 水平分表**

针对数据量巨大的单张表（比如订单表），按照规则把一张表的数据切分到多张表里面去。 但是这些表还是在**同一个库中**，所以库级别的数据库操作还是有IO瓶颈。



总结

**垂直分表**: 将**一个表按照字段分成多表**，每个表存储其中一部分字段。

**垂直分库**: **根据表的业务不同,分别存放在不同的库中**,这些库分别部署在不同的服务器. **水平分库**: **把一张表的数据按照一定规则,分配到不同的数据库**,每一个库只有这张表的部分数据.

**水平分表**: **把一张表的数据按照一定规则,分配到同一个数据库的多张表中**,每个表只有这个表的部分数据.

**1.3 如何实现分库分表**

当数据库进行分库分表后，数据由一个数据库分散到多个数据库中。此时系统要查询时需要切换不同的数据库进行查询，那么系统如何知道要查询的数据在哪个数据库中？当添加一条记录时要向哪个数据库中插入呢？这些问题处理起来都是非常的麻烦。

这种情况下可以使用一个数据库中间件mycat来解决相关的问题。

**2. MyCat**

**2.1 什么是MyCat**

MyCat 是目前最流行的基于java语言编写的**数据库中间件**，是一个实现了MySQL协议的服务器，前端用户可以把它看作是一个数据库代理，用MySQL客户端工具和命令行访问，而其后端可以用MySQL原生协议与多个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和读写分离**，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对于我们Java程序员来说,**就是一个近似等于MySQL的数据库服务器，你可以用连接MySQ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)

我们可以像使用MySQL一样使用MyCat,Mycat可以管理若干MySQL数据库,同时实现数据的存储和操作

**2.2 MyCat支持的数据库**

Oracle；MySQL；mongoDB；SQlServer

**2.3 MyCat 概念说明**

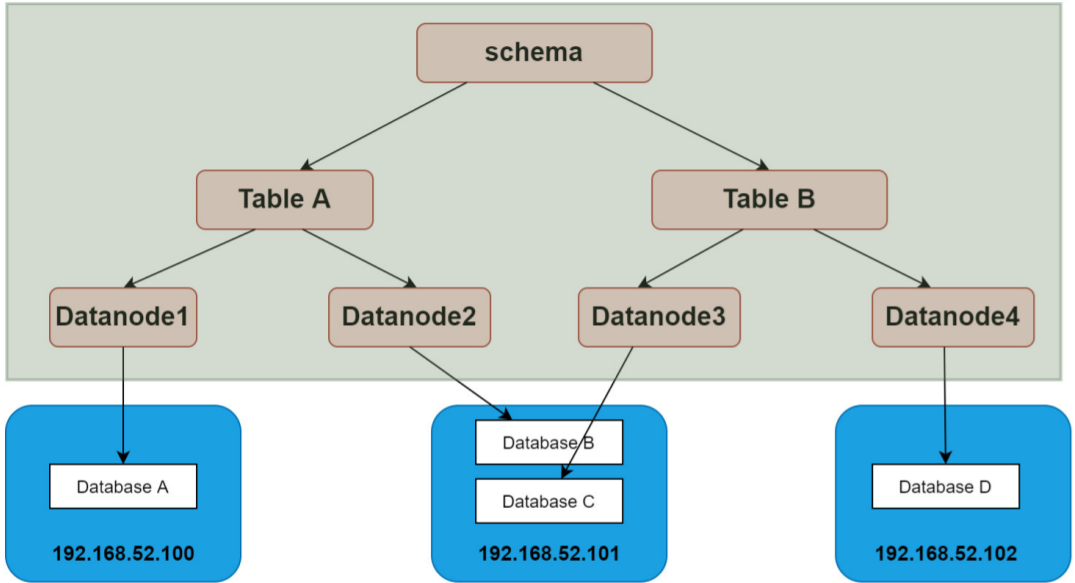
**2.3.1 MyCat的分片策略**

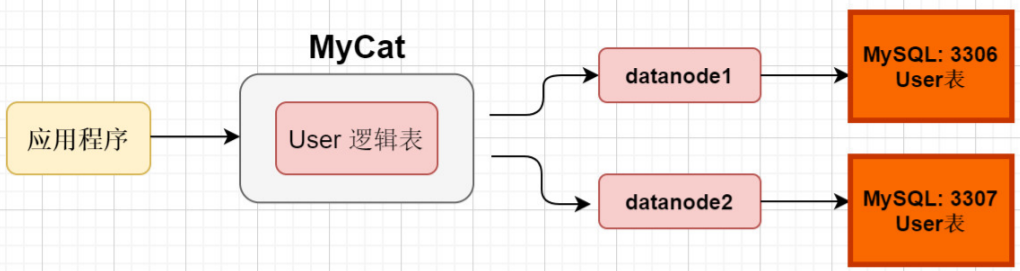
什么是分片：通过某种特定的条件，将我们存放在同一个数据库中的数据分散存放到多个数据库（主机）上面，以达到分散单台设备负载的效果。

MyCat支持两种切分模式：

（1）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

（2）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。





**逻辑库(schema)**：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。Mycat在操作时，使用逻辑库来代表这个完整的数据库集群，便于对整个集群操作。

**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：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。 总而言之就是需要进行分片的表。

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。

**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所在的机器就是节点主机，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rule，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。

**2.4 MyCat的下载和安装**

**2.4.1 安装环境**

1. jdk: 要求jdk必须是1.7及以上版本

2. MySQL: 推荐mysql5.5 版本以上

3. MyCat: Mycat的官方网站：http://www.mycat.org.cn/

**2.4.2 MyCat安装**

**注意: 提前安装好JDK**

第一步: 下载MyCat



第二步: 上传MyCat到服务器 ,并解压

解压：tar -zxvf ….tar.gz；

进入mycat目录：cd mycat,，使用一下命令：

启动命令：./mycat start

停止命令：./mycat stop

重启命令：./mycat restart

查看状态：./mycat status

带控制台启动：./mycat console

**2.5 MyCat核心配置**

**2.5.1 schema.xml配置**

Schema.xml作为MyCat中重要的配置文件之一，管理着MyCat的逻辑库、表、分片规则、DataNode以及DataSource。弄懂这些配置，是正确使用MyCat的前提。这里就一层层对该文件进行解析。

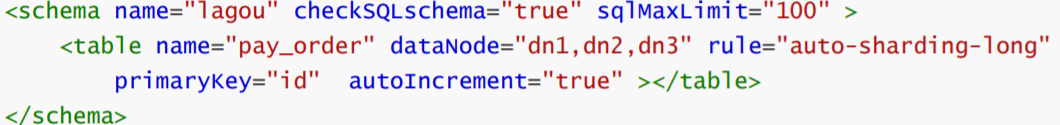
**schema标签**



|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 值 | 数量限制 | 说明 |
| name | string | (1) | 逻辑库名称 |
| dataNode | 任意String | (0..1) | 分片节点 |
| sqlMaxLimit | Integer | (1) | 查询返回的记录数限制limit |
| checkSQLschema | Boolean | (1) | 执行SQL时,是否去掉表所属的库名 |

**table标签**

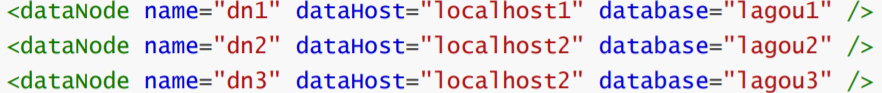
table标签定义了Mycat中的逻辑表，所有需要拆分的表都需要在这个标签中定义



|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 值 | 数量限制 | 说明 |
| name | String | (1) | 逻辑表名 |
| dataNode | String | (1..\*) | 分片节点 |
| rule | String | (0..1) | 分片规则 |
| ruleRequired | Boolean | (0..1) | 是否强制绑定分片规则 |
| primaryKey | String | (1) | 主键 |
| type | String | (0..1) | 逻辑表类型，全局表、普通表 |
| autoIncrement | Boolean | (0..1) | 自增长主键 |
| subTables | String | (1) | 分表 |
| needAddLimit | Boolean | (0..1) | 是否为查询SQL自动加limit限制 |

**dataNode标签**

dataNode标签定义了MyCat中的分片节点，也就是我们通常说所的数据分片。



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

dataHost: 用于定义该分片属于哪个分片主机，属性值是引用dataHost标签上定义的name属性。

database：用于定义该分片节点属于哪个具体的库。

**dataHost标签**

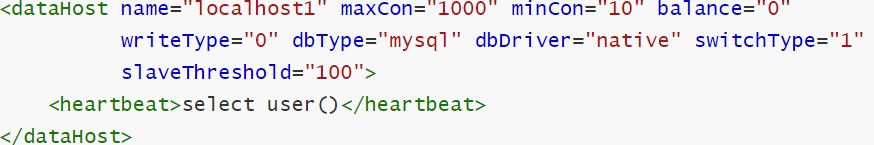
dataHost标签在Mycat逻辑库中也是作为最底层的标签存在，直接定义了具体的数据库实例、读写分离配置和心跳语句



|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 值 | 数量限制 | 说明 |
| name | String | (1) | 节点主机名 |
| maxCon | Integer | (1) | 最大连接数 |
| minCon | Integer | (1) | 最小连接数 |
| balance | Integer | (1) | 读操作负载均衡类型 |
| writeType | Integer | (1) | 写操作负载均衡类型 |
| dbType | String | (1) | 数据库类型 |
| dbDriver | String | (1) | 数据库驱动 |
| switchType | String | (1) | 主从切换类型 |

**heartbeat标签**

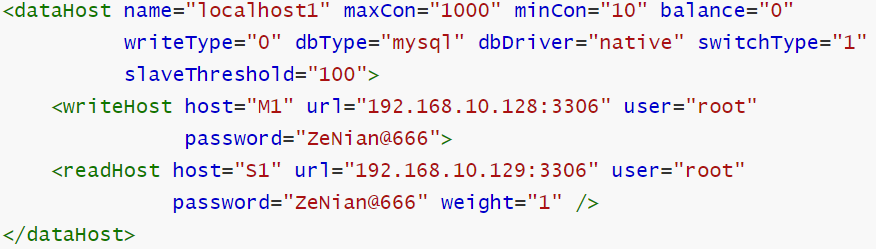
heartbeat标签内指明用于和后端数据库进行心跳检查的语句。例如：MySQL可以使用 select user()、Oracle可以使用select 1 from dual等



**writeHost和readHost标签**

writeHost和readHost标签都指定后端数据库的相关配置给mycat，用于实例化后端连接池。

writeHost指定写实例 , readHost指定读实例. 在一个dataHost内可以定义多个 writeHost和readHost



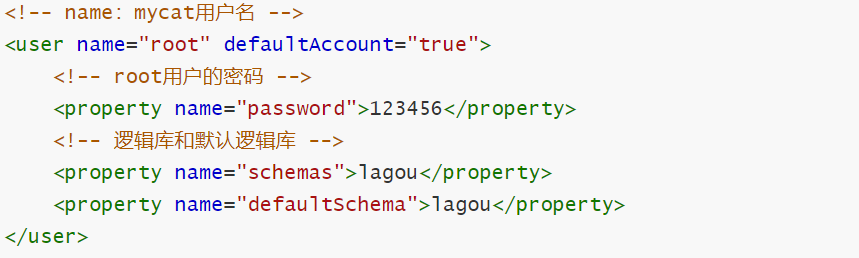
|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 值 | 数量限制 | 说明 |
| host | String | (1) | 主机名 |
| url | String | (1) | 连接字符串 |
| password | String | (1) | 密码 |
| user | String | (1) | 用户名 |
| weight | String | (1) | 权重 |
| usingDecrypt | String | (1) | 是否对密码加密，默认0 |

**2.5.2 server.xml配置**

server.xml几乎保存了所有mycat需要的系统配置信息。

2.5.2.1 user标签

这个标签主要用于定义登录mycat的用户和权限。



2.5.2.2 连接MyCat

**mysql -uroot -p123456 -h127.0.0.1 -P8066**

**2.5.3 rule.xml配置**

rule.xml里面就定义了我们对表进行拆分所涉及到的规则定义。我们可以灵活的对表使用不同的分片算法，或者对表使用相同的算法但具体的参数不同。

这个文件里面主要有tableRule和function这两个标签。在具体使用过程中可以按照需求添加tableRule和function。

此配置文件可以不用修改，使用默认即可。

**2.5.1 tableRule标签**



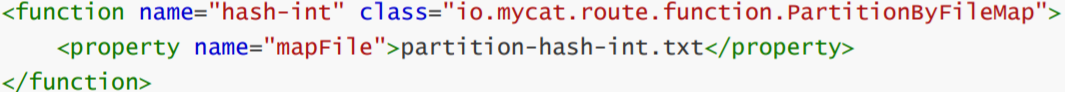
name属性：指定唯一的名字，用于标识不同的表规则。

rule标签: 指定对物理表中的哪一列进行拆分和使用什么路由算法

columns标签：指定要拆分的列名字。

algorithm标签：使用function标签中的name属性，连接表规则和具体路由算法。

**2.5.2 function标签**



name属性：指定算法的名字。

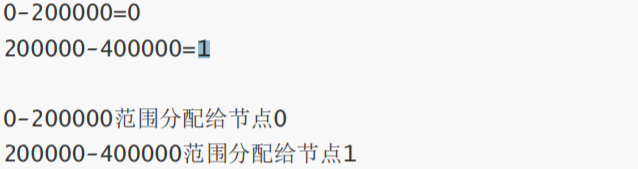
class属性：制定路由算法具体的类名字。

property标签：为具体算法需要用到的一些属性。文件中定义分片策略

**2.5.4 常用的分片规则**

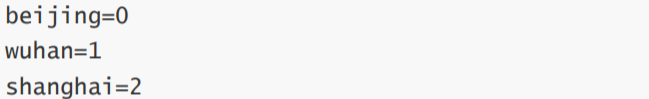
自动分片(auto-sharding-long)：根据指定的列的范围进行分片.默认从0节点开始

autopartition-long.txt文件：



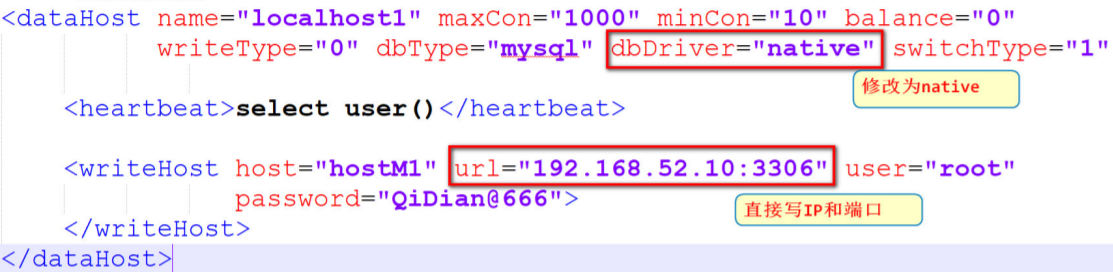
枚举分片(sharding-by-intfile)：把数据分类存储, 这种方法适用于取值固定的场合，例如性别和省份

partition-hash-int.txt文件内容如下：



取模分片(mod-long)：根据配置中的count值进行分片，将数据分成配置的count份，然后将数据均匀的分布在各个节点上

**解决MyCat乱码问题**



**配置全局序列号**

在实现分库分表的情况下，数据库自增主键已无法保证自增主键的全局唯一。为此，Mycat提供了全局sequence，并且提供了包含本地配置和数据库配置等多种实现方式。

在server.xml文件中，将下面标签中的值从1改为0



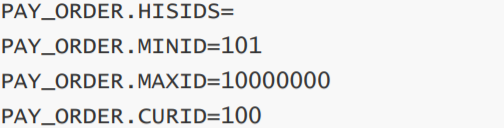
0 表示是表示使用本地文件方式。

1 表示的是根据数据库来生成

2 表示时间戳的方式ID= 64位二进制(42(毫秒)+5(机器 ID)+5(业务编码)+12(重复累加)

本地文件：

此方式Mycat将sequence配置到文件中，当使用到sequence中的配置后，Mycat会更新classpath中的sequence\_conf.properties文件中sequence当前的值。



其中HISIDS表示使用过的历史分段(一般无特殊需要可不配置)，MINID表示最小ID值，MAXID表示最大ID值，CURID表示当前ID值

**2.7 MyCat读写分离**

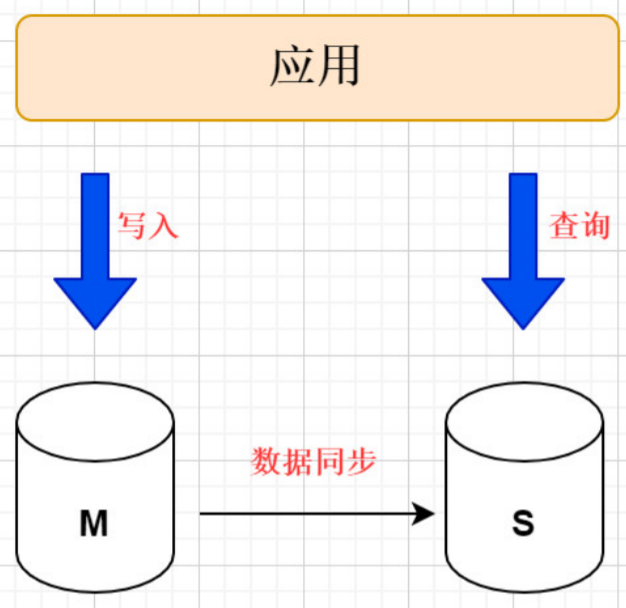
**2.7.1 什么是读写分离**

在实际的生产环境中, 数据的读写操作如果都在同一个数据库服务器中进行, 当遇到大量的并发读或者写操作的时候,是没有办法满足实际需求的,数据库的吞吐量将面临巨大的瓶颈压力.

**主从复制：**通过搭建主从架构, 将数据库拆分为主库和从库，主库负责处理事务性的增删改操作，从库负责处理查询操作，能够有效的避免由数据更新导致的行锁，使得整个系统的查询性能得到极大的改善。

**读写分离：**读写分离就是让主库处理事务性操作，从库处理select查询。数据库复制被用来把事务性查询导致的数据变更同步到从库，同时主库也可以select查询。

**读写分离的数据节点中的数据内容是一致。**



**2.7.2 MySQL主从复制(同步)**

MyCat的读写分离是建立在MySQL主从复制基础之上实现的，所以必须先搭建MySQL的主从复制架构。

**主从复制的用途：**

（1）实时灾备，用于故障切换

（2）读写分离，提供查询服务

（3）备份，避免影响业务

**主从部署必要条件：**

（1）主库开启binlog日志（设置log-bin参数）

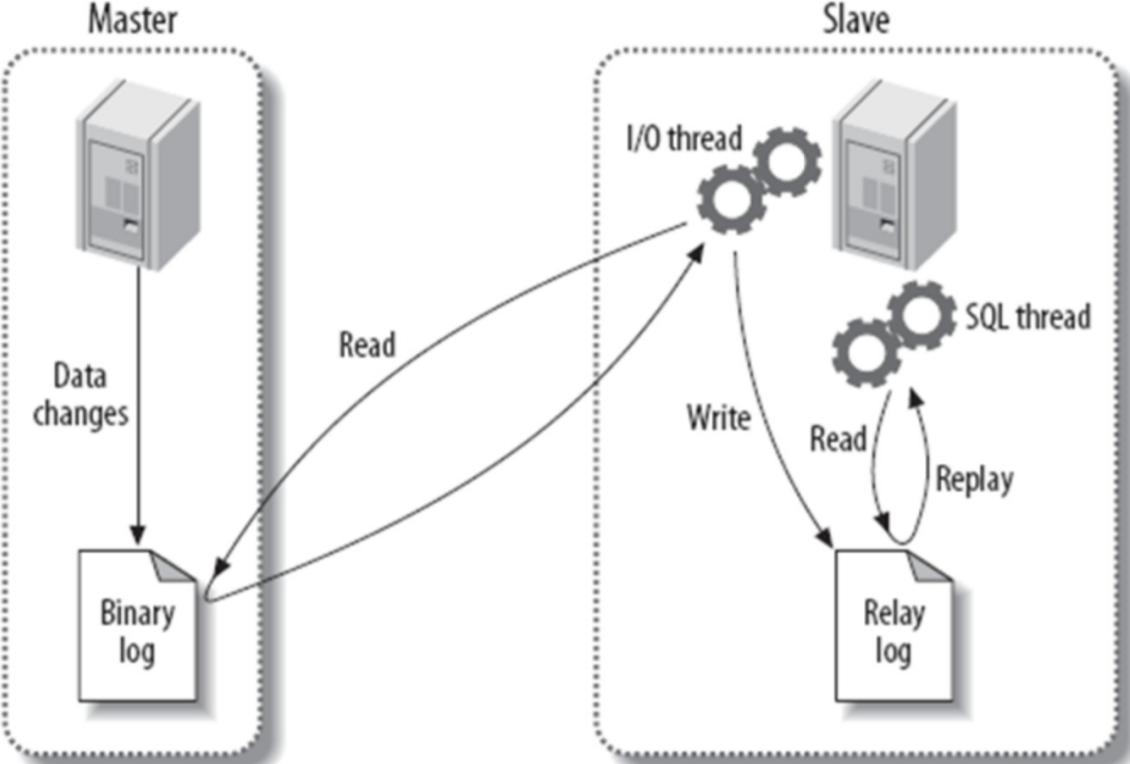
（2）主从server-id不同

（3）从库服务器能连通主库

**主从复制的原理：**

Mysql中有一种日志叫做bin日志(二进制日志)。这个日志会记录下所有修改了数据库的SQL语句（insert,update,delete,create/alter/drop table, grant 等等）。

主从复制的原理其实就是把主服务器上的bin日志复制到从服务器上执行一遍，这样从服务器上的数据就和主服务器上的数据相同了。



1. 主库db的更新事件(update、insert、delete)被写到binlog

2. 主库创建一个binlog dump thread，把binlog的内容发送到从库

3. 从库启动并发起连接，连接到主库

4. 从库启动之后，创建一个I/O线程，读取主库传过来的binlog内容并写入到relay log

5. 从库启动之后，创建一个SQL线程，从relay log里面读取内容，执行读取到的更新事件，将更新内容写入到slave的db

**2.7.3 主从复制架构搭建**

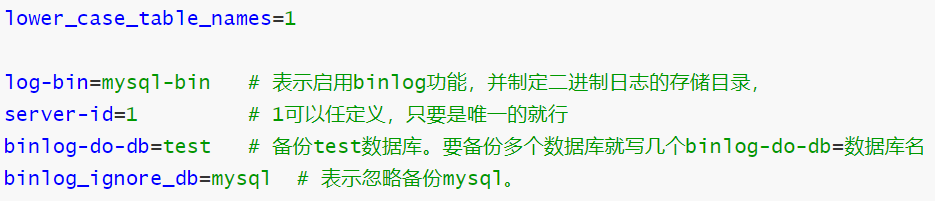
Mysql的主从复制至少是需要两个Mysql的服务，当然Mysql的服务是可以分布在不同的服务器上，也可以在一台服务器上启动多个服务。

1) 第一步 master中创建数据库和表

2) 第二步 修改主数据库的配置文件my.cnf

vim /etc/my.cnf

插入下面的内容



lower\_case\_table\_names=1

log-bin=mysql-bin

server-id=1

binlog-do-db=test

binlog\_ignore\_db=mysql

3) 第三步 重启MySQL

4) 第四步 在主数据库上, 创建一个允许从数据库来访问的用户账号.

用户: slave；密码：123456；主从复制使用 REPLICATION SLAVE 赋予权限

**GRANT REPLICATION SLAVE ON \*.\* TO 'slave'@'192.168.10.129' IDENTIFIED BY 'Qwer@1234';**

5) 第五步 停止主数据库的更新操作, 并且生成主数据库的备份

**FLUSH TABLES WITH READ LOCK;** 执行以下命令锁定数据库以防止写入数据

6) 导出数据库,恢复写操作

**unlock tables;**

7) 将刚才主数据库备份的test.sql导入到从数据库

导入后, 主库和从库数据会追加相平，保持同步！此过程中，若主库存在业务，并发较高，在同步的时候要先锁表，让其不要有修改！等待主从数据追平，主从同步后在打开锁！

8) 接着修改从数据库的 my.cnf

**server-id=2** 增加server-id参数,保证唯一.

9) 在从数据库设置相关信息

执行以下SQL：

STOP SLAVE; 停止主从

CHANGE MASTER TO MASTER\_HOST='192.168.10.128',

MASTER\_USER='slave',

MASTER\_PASSWORD='Qwer@1234',

MASTER\_PORT=3306,

MASTER\_LOG\_FILE='mysql-bin.000001',

MASTER\_LOG\_POS=0,

MASTER\_CONNECT\_RETRY=10;

10) 修改auto.cnf中的UUID,保证唯一

vim /var/lib/mysql/auto.cnf

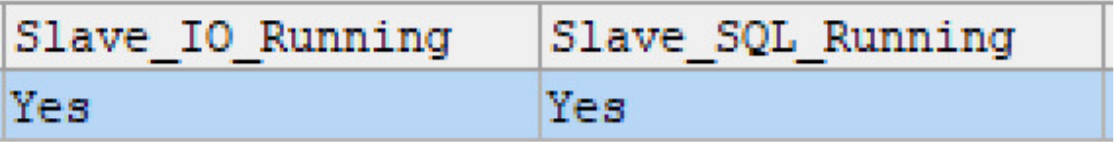
**server-uuid=a402ac7f-c392-11ea-ad18-000c2980a208**

service mysqld restart

11) 在从服务器上,启动slave 进程

start slave;

SHOW SLAVE STATUS; 查看状态



**注意: 这两个参数的值,必须是 Yes,否则就要进行错误的排查.**

**命令行下使用如下命令查看错误信息状态**

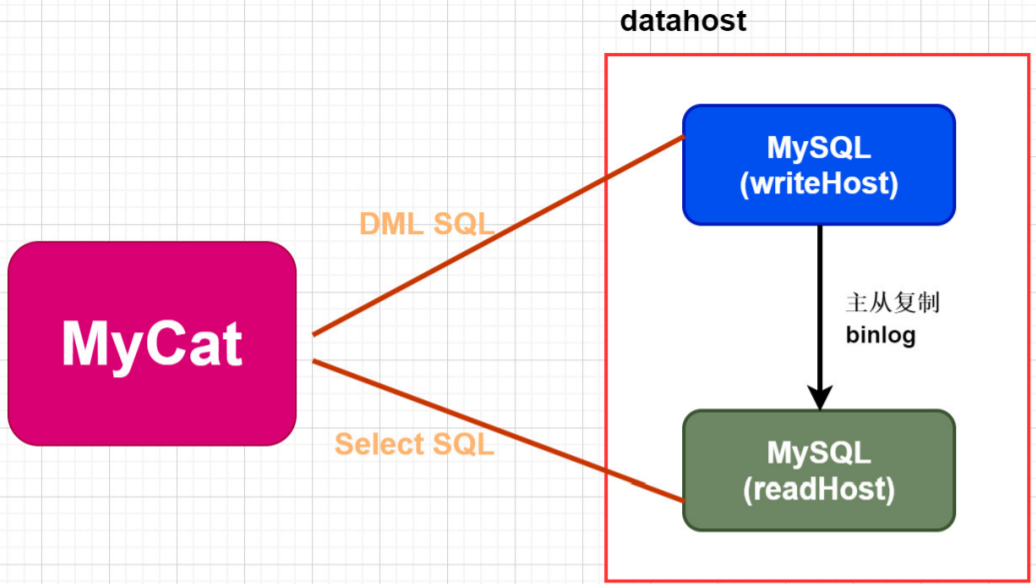
SHOW SLAVE STATUS \G;

**补：手动删除master中的binlog日志：reset master;**

**手动删除slave中的中继日志：reset slave;**

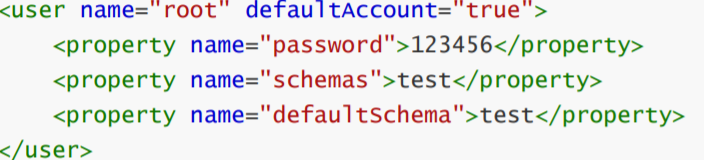
**重新开启主从：start slave;**

**2.7.4 mycat实现读写分离**



在schema.xml文件中配置Mycat读写分离。使用前需要搭建MySQL主从架构，并实现主从复制，Mycat不负责数据同步问题。

**server.xml：**修改用户可以访问的逻辑库为 test



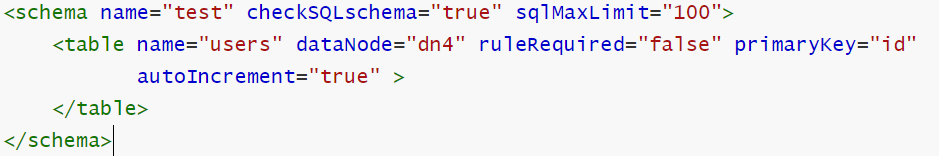
**schema**

逻辑库 name="test"

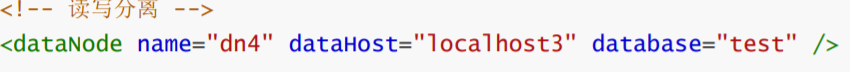
逻辑表 name="users"

读写分离 不设置分片规则 ruleRequired=false

分片节点 dataNode="dn4"



**dataNode**



**dataHost**



balance参数：

0 ：所有读操作都发送到当前可用的writeHost

1 ：所有读操作都随机发送到readHost和stand by writeHost

2 ：所有读操作都随机发送到writeHost和readHost

3 ：所有读操作都随机发送到writeHost对应的readHost上，但是writeHost不负担读压力

writeType参数：

0 ：所有写操作都发送到可用的writeHost

1 ：所有写操作都随机发送到readHost

2 ：所有写操作都随机发送到writeHost，readHost

2.分库分表技术之ShardingJDBC

**1. ShardingJDBC**

**1.1 分库分表方式回顾**

1) 分库分表的目的就是将我们的单库的数据控制在合理范围内,从而提高数据库的性能.

垂直拆分( 按照结构分 )

**垂直分表**: 将一张宽表(字段很多的表), 按照字段的访问频次进行拆分,就是按照表单结构进行拆

**垂直分库**: 根据不同的业务,将表进行分类, 拆分到不同的数据库. 这些库可以部署在不同的服务器,分摊访问压力.

水平拆分( 按照数据行分 )

**水平分库**: 将一张表的数据 ( 按照数据行) 分到多个不同的数据库.每个库的表结构相同. 每个库都只有这张表的部分数据,当单表的数据量过大,如果继续使用水平分库, 那么数据库的实例就会不断增加,不利于系统的运维. 这时候就要采用水平分表.

**水平分表**: 将一张表的数据 ( 按照数据行) , 分配到同一个数据库的多张表中,每个表都只有一部分数据.

2) 什么时候用分库分表

在系统设计阶段,就要完成垂直分库和垂直分表. 在数据量不断上升,数据库性能无法满足需求的时候, 首先要考虑的是缓存、读写分离、索引技术等方案.如果数据量不断增加,并且持续增长再考虑水平分库水平分表.

**1.2 分库分表带来的问题**

关系型数据库在单机单库的情况下,比较容易出现性能瓶颈问题,分库分表可以有效的解决这方面的问题,但是同时也会产生一些比较棘手的问题

**1.2.1 事务一致性问题**

当我们需要更新的内容同时分布在不同的库时, 不可避免的会产生跨库的事务问题. 原来在一个数据库操作, 本地事务就可以进行控制, 分库之后一个请求可能要访问多个数据库,如何保证事务的一致性,目前还没有简单的解决方案.

**1.2.2 跨节点关联的问题**

在分库之后, 原来在一个库中的一些表,被分散到多个库,并且这些数据库可能还不在一台服务器,无法关联查询。解决这种关联查询,需要我们在代码层面进行控制,将关联查询拆开执行,然后再将获取到的结果进行拼装.

**1.2.3 分页排序查询的问题**

分库并行查询时,如果用到了分页每个库返回的结果集本身是无序的, 只有将多个库中的数据先查出来,然后再根据排序字段在内存中进行排序,如果查询结果过大也是十分消耗资源的.

**1.2.4 主键避重问题**

在分库分表的环境中,表中的数据存储在不同的数据库, 主键自增无法保证ID不重复, 需要单独设计全局主键.

**1.2.5 公共表的问题**

不同的数据库,都需要从公共表中获取数据. 可以在每一个库都创建这个公共表, 所有对公共表的更新操作,都同时发送到所有分库执行. ShardingJDBC可以帮助我们解决这个问题.

**1.3 ShardingJDBC 简介**

ShardingSphere是一套开源的分布式数据库中间件解决方案组成的生态圈，它由Sharding-JDBC、Sharding-Proxy和Sharding-Sidecar（计划中）这3款相互独立的产品组成,我们只关注Sharding-JDBC即可.

官方地址：https://shardingsphere.apache.org/document/current/cn/overview/

Sharding-JDBC定位为轻量级Java框架，在Java的JDBC层提供的额外服务。它使用客户端直连数据库，以jar包形式提供服务，无需额外部署和依赖，可理解为增强版的JDBC驱动，完全兼容JDBC和各种ORM框架的使用。

适用于任何基于Java的ORM框架，如：JPA, Hibernate, Mybatis, Spring JDBC Template或直接使用JDBC。

基于任何第三方的数据库连接池，如：DBCP, C3P0, Druid等。

支持任意实现JDBC规范的数据库。目前支持MySQL，Oracle，SQLServer和PostgreSQL。

**1.3.2 Sharding-JDBC主要功能**

**数据分片；读写分离**

通过Sharding-JDBC，应用可以**透明**的使用jdbc访问已经分库分表、读写分离的多个数据源，而不用关心数据源的数量以及数据如何分布。

**1.3.3 Sharding-JDBC与MyCat的区别**

1) mycat是一个中间件的第三方应用，sharding-jdbc是一个jar包

2) 使用mycat时不需要修改代码，而使用sharding-jdbc时需要修改代码

3) Mycat是基于Proxy，它复写了MySQL协议，将Mycat Server伪装成一个MySQL数据库，而Sharding-JDBC是基于JDBC的扩展，是以jar包的形式提供轻量级服务的。

**1.4 Sharding-JDBC使用**

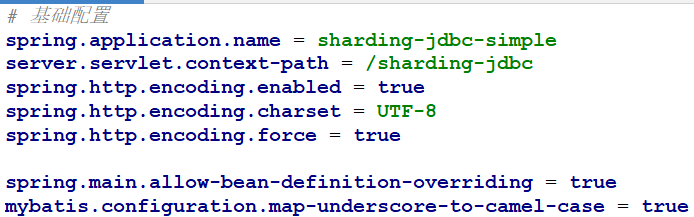
**1.4.1 搭建基础环境**

创建SpringBoot项目引入maven依赖sharding-jdbc-spring-boot-starter

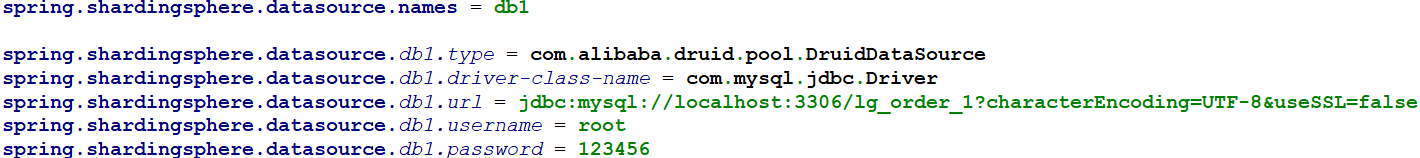
**1.4.2 分片规则配置(水平分表)**

使用sharding-jdbc 对数据库中水平拆分的表进行操作,通过sharding-jdbc对分库分表的规则进行配置,配置内容包括：数据源、主键生成策略、分片策略等。application.properties

基础配置：



数据源配置：



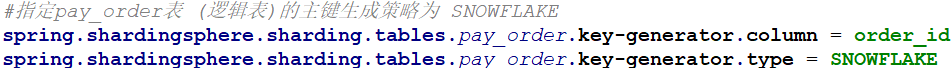
配置数据节点：



表达式.pay\_order\_$->{1..2} ，$会被大括号中的{1..2}所替换

会有两种选择: db1.pay\_order\_1和db1.pay\_order\_2

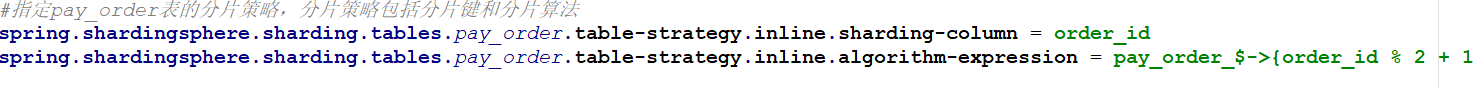
配置主键生成策略：



使用shardingJDBC提供的主键生成策略,全局主键

为避免主键重复, 生成主键采用**SNOWFLAKE分布式ID生成算法**

配置分片算法：



分表策略表达式: pay\_order\_$-> {order\_id % 2 + 1}

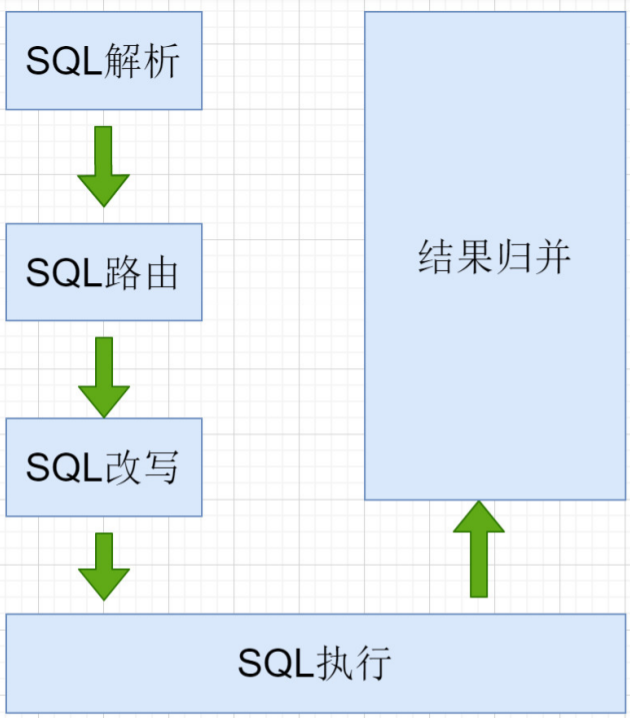
{order\_id % 2 + 1} 结果是偶数操作 pay\_order\_1表；{order\_id % 2 + 1} 结果是奇数操作 pay\_order\_2表

打开SQL日志：



**1.4.3 ShardingJDBC执行流程**

当ShardingJDBC接收到发送的SQL之后,会执行下面的步骤,最终返回执行结果



1. SQL解析: 编写SQL查询的是逻辑表, 执行时 ShardingJDBC 要解析SQL ,解析的目的是为了找到需要改写的位置.

2. SQL路由: SQL的路由是指 将对逻辑表的操作,映射到对应的数据节点的过程. ShardingJDBC会获取分片键判断是否正确,正确 就执行分片策略(算法) 来找到真实的表.

3. SQL改写: 程序员面向的是逻辑表编写SQL, 并不能直接在真实的数据库中执行,SQL改写用于将逻辑SQL改为在真实的数据库中可以正确执行的SQL.

4. SQL执行: 通过配置规则pay\_order\_$->{order\_id % 2 + 1} ,可以知道当order\_id为偶数时, 应该向pay\_order\_1表中插入数据, 为奇数时向pay\_order\_2表插入数据.

5. 将所有真正执行sql的结果进行汇总合并，然后返回。

**1.5 Sharding-JDBC分库分表**

**1.5.1 水平分库分表**

1. 分库策略

分库策略 ,目的是将一个逻辑表 , 映射到多个数据源

**spring.shardingsphere.sharding.tables.逻辑表名称.database-strategy.分片策略.分 片策略属性名 = 分片策略表达式**

2. 分表策略

分表策略, 如何将一个逻辑表 , 映射为多个实际表

**spring.shardingsphere.sharding.tables.逻辑表名称.table-strategy.分片策 略.algorithm-expression = 分片策略表达式**

3. Sharding-JDBC支持以下几种分片策略：

**standard**：标准分片策略

**complex**：符合分片策略

**inline**：行表达式分片策略,使用Groovy的表达式.

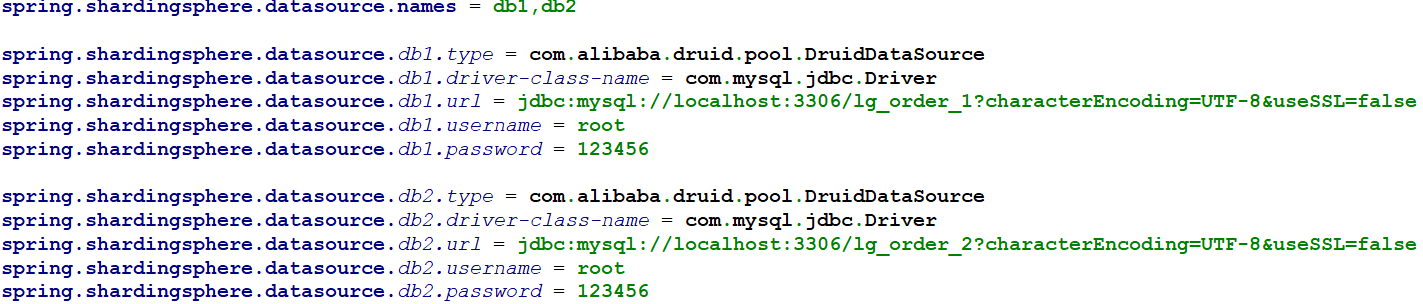
**hint**：Hint分片策略，对应HintShardingStrategy。

**none**：不分片策略，对应NoneShardingStrategy。不分片的策略。

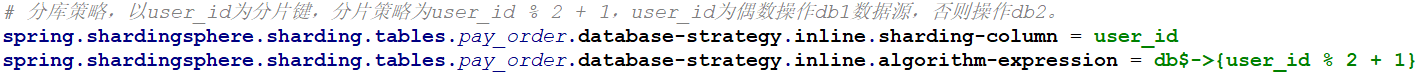
具体信息请查阅官方文档：https://shardingsphere.apache.org

4. 分片规则配置

1）配置两个数据库



2）配置分库策略



3）配置分表策略，哪个库中的哪个表

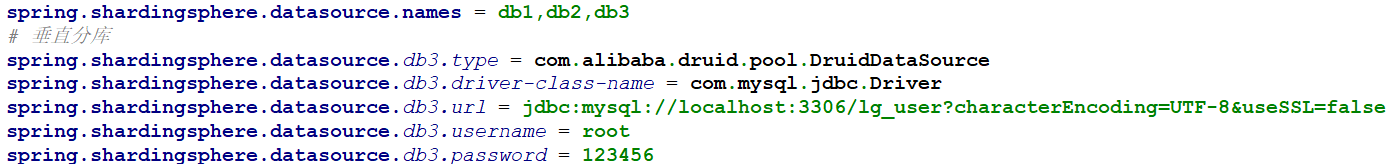


**1.5.2 垂直分库**

垂直分库是指按照业务将表进行分类，分布到不同的数据库上面，每个库可以放在不同的服务器上，它的核心理念是专库专用.

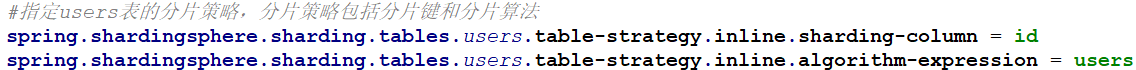
在使用微服务架构时，业务切割得足够独立，数据也会按照业务切分，保证业务数据隔离，大大提升了数据库的吞吐能力。

配置数据源信息：



配置数据节点：





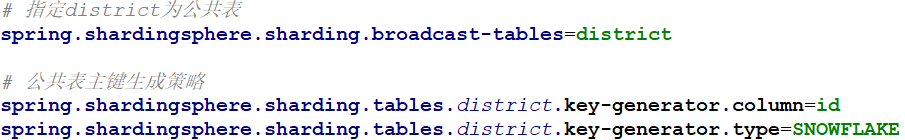
**1.6 Sharding-JDBC 操作公共表**

**1.6.1 什么是公共表**

公共表属于系统中数据量较小，变动少，而且属于高频联合查询的依赖表。参数表、数据字典表等属于此类型。

可以将这类表在每个数据库都保存一份，所有更新操作都同时发送到所有分库执行。

在Sharding-JDBC的配置文件中指定公共表

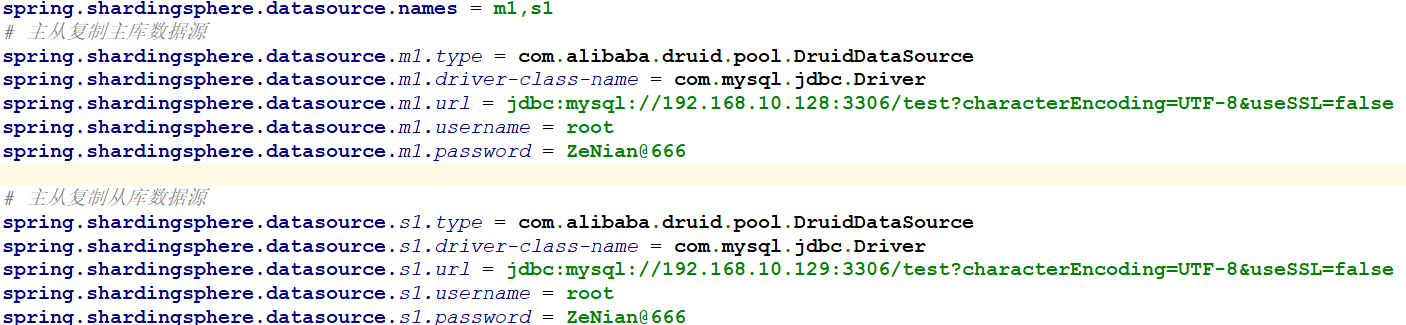


**1.7 Sharding-JDBC读写分离**

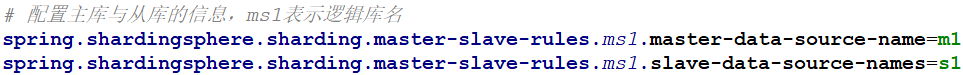
**Sharding-JDBC读写分离则是根据SQL语义的分析，将读操作和写操作分别路由至主库与从库**。它提供透明化读写分离，让使用方尽量像使用一个数据库一样使用主从数据库集群。

**1.7.1 sharding-jdbc实现读写分离**

1. 配置数据源



2. 配置主库与从库的相关信息



3. 配置数据节点



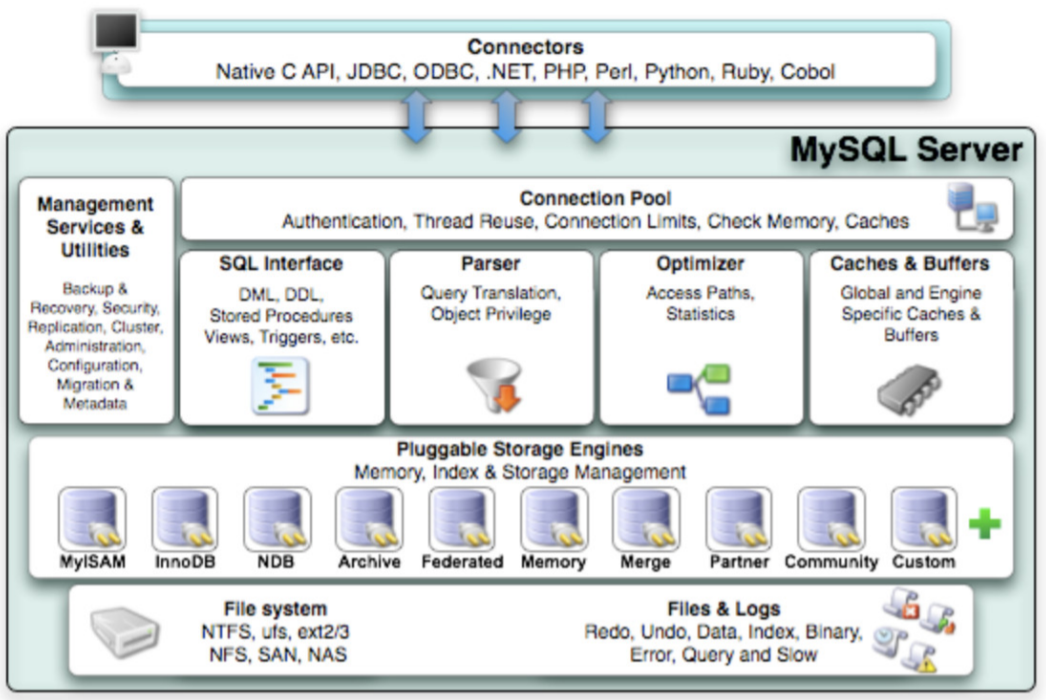
3.MySQL高级

**1.MySQL架构组成**

**1.1 MySQL逻辑架构**

1.1.1 MySQL架构体系介绍

MySQL由连接池、SQL接口、解析器、优化器、缓存、存储引擎等组成，可以分为四层，即连接层、服务层、引擎层和文件系统层。



1. 连接层：最上面是一些客户端和连接服务, 不是MySQL特有的，所有基于网络的C/S的网络应用程序都应该包括连接处理、认证、安全管理等。

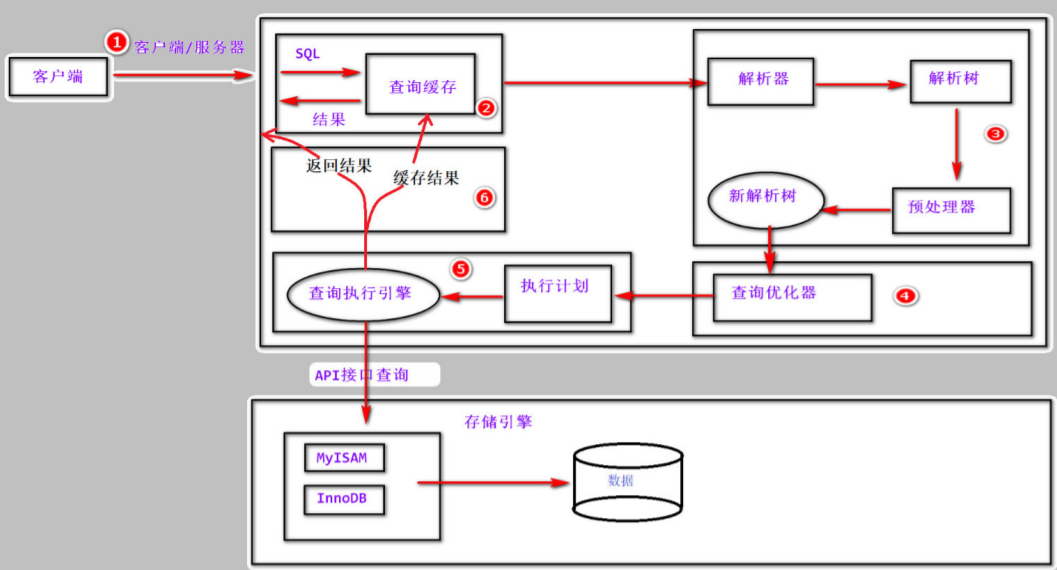
2. 服务层：中间层是MySQL的核心，包括查询解析、分析、优化和缓存等。同时它还提供跨存储引擎的功能，包括存储过程、触发器和视图等。

3. 引擎层：存储引擎层，它负责存取数据。服务器通过API可以和各种存储引擎进行交互。不同的存储引擎具有不同的功能,我们可以根据实际需求选择使用对应的存储引擎

4. 存储层：数据存储层,主要是将数据存储在运行于裸设备的文件系统之上,并完成与存储引擎的交互

1.1.2 SQL查询流程

用一条 SQL SELECT 语句的执行轨迹来说明客户端与 MySQL 的交互过程



1. 通过客户端/服务器通信协议与MySQL建立连接

2. 查询缓存，这是MySQL的一个可优化查询的地方，如果开启了Query Cache且在查询缓存过程中查询到完全相同的SQL语句，则将查询结果直接返回给客户端；如果没有开启Query Cache或者没有查询到完全相同的SQL语句则会由解析器进行语法语义解析，并生成解析树。

3. 预处理器生成新的解析树。

4. 查询优化器生成执行计划。

5. 查询执行引擎执行SQL语句，此时查询执行引擎会根据SQL语句中表的存储引擎类型，以及对应的API接口与底层存储引擎缓存或者物理文件的交互情况，得到查询结果，由MySQL Server过滤后将查询结果缓存并返回给客户端。若开启了Query Cache，这时也会将SQL语句和结果完整地保存到Query Cache中，以后若有相同的SQL语句执行则直接返回结果。

**2.MySQL的备份与恢复**

**2.1 为什么要进行数据备份**

我们试着想一想, 在生产环境中什么最重要？如果我们服务器的硬件坏了可以维修或者换新, 软件问题可以修复或重新安装, 但是如果数据没了呢 对于一些网站、系统来说，数据库就是一切，所以做好数据库的备份是至关重要的！

**2.1.1 数据库备份的应用场景**

**数据丢失应用场景：**

（1）系统硬件或软件故障

（2）自然灾害,比如水灾火灾地震等

（3）黑客攻击,非法访问者故意破坏

（4）误操作 , 人为的误操作占比最大

**非数据丢失应用场景：**

（1）开发测试环境数据库搭建

（2）数据库或者数据迁移

**2.2 数据备份的类型**

**2.2.1 按照业务方式分**

**完全备份**：将数据库的全部信息进行备份，包括数据库的数据文件、日志文件，还需要备份文件的存储位置以及数据库中的全部对象和相关信息。

**差异备份**：备份从最近的完全备份后对数据所做的修改，备份完全备份后变化了的数据文件、日志文件以及数据库中其他被修改的内容。

**增量备份**：增量备份是指在一次全备份或上一次增量备份后，以后每次的备份只需备份与前一次相比增加或者被修改的文件。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 完全备份 | 差异备份 | 增量备份 |
| 备份方法 | 备份所有文件 | 一次全备份后,备份与全备份差异的部分 | 一次全备份后,备份与上次备份的差异部分 |
| 备份速度 | 最慢 | 较快 | 最快 |
| 恢复速度 | 最快 | 较快 | 最慢 |
| 空间要求 | 最多 | 较多 | 最少 |
| 优势 | 最快的恢复速度, 只需要上一次完全备份就能恢复 | 相比增量,更快也更简单 并且只需要最近一次的完全备份和最后一次的差异备份就能恢复 | 备份速度快,较少的空间需求,没有重复的备份文件 |
| 劣势 | 最多的空间需求大量重复的备 | 较慢的备份速度,仍然会存在许多的备份文件 | 最慢的恢复速度，恢复需要最近一次完全备份和全部增量备份 |

**2.2.2 备份的组合方式**

**完全备份与差异备份**：以每周数据备份为例，可以在星期一进行完全备份，在星期二至星期六进行差异备份。如果在星期六数据被破坏了，则只需要还原星期一完全的备份和星期五的差异备份。

这种策略备份数据需要较多的时间，但还原数据使用较少的时间。

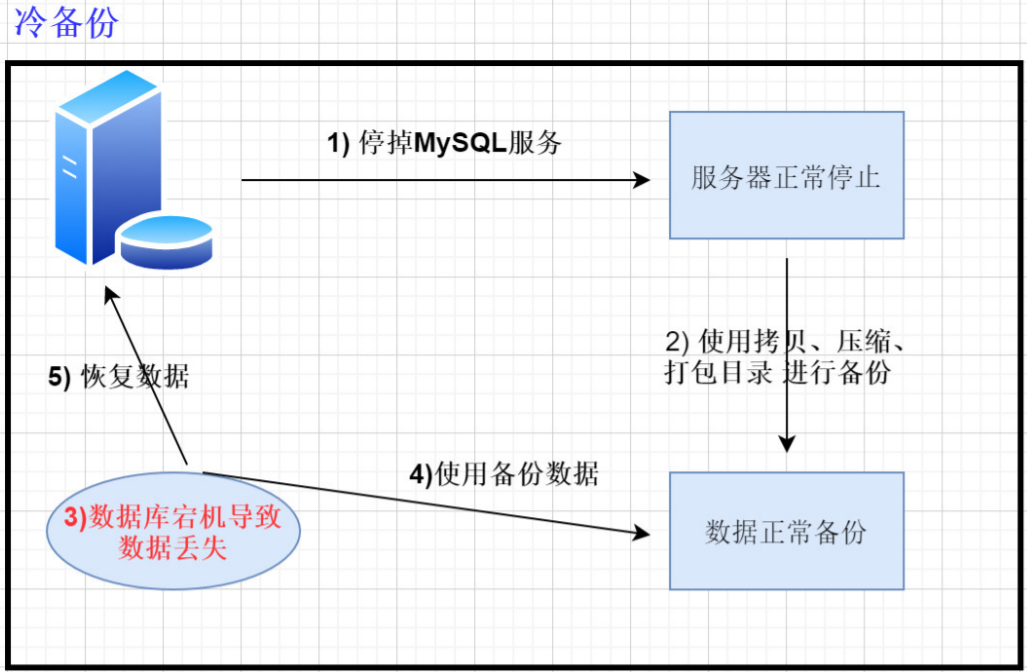
**完全备份与增量备份**：以每周数据备份为例，在星期一进行完全备份，在星期二至星期六进行增量备份。如果在星期六数据被破坏了，则需要还原星期一正常的备份和从星期二至星期五的所有增量备份。

这种策略备份数据需要较少的时间，但还原数据使用较长的时间。

**2.3 MySQL冷备份和热备份**

**冷备份和热备份指的是, 按照数据库的运行状态分类**

**2.3.1 冷备份**



**冷备份的优点：**

（1）是操作比较方便的备份方法（只需拷贝文件）

（2）低度维护，高度安全。

**冷备份的缺点:**

（1）在实施备份的全过程中，数据库必须要作备份而不能作其它工作。

（2）若磁盘空间有限，只能拷贝到磁带等其它外部存储设备上，速度比较慢慢。

（3）不能按表或按用户恢复。

2.3.1.1 冷备份实战

**1) 关闭SELinux**：**修改 selinux 配置文件,将SELINUX=enforcing改为SELINUX=disabled，保存后退出**

vim /etc/selinux/config SELINUX=disabled

修改后需要重启：reboot # 重启命令

**2) 找到MySQL数据文件位置,停止MySQL服务**

**SHOW VARIABLES LIKE '%dir%';** 结果显示, 数据目录就是datadir的所在位置，即 /var/lib/mysql/

**service mysqld stop** -- 停止mysql

**3) 进入到/mysql的上级目录, 执行打包命令将数据文件打包备份**

cd /var/lib/ # 进入其上级目录

tar jcvf /root/backup.tar.bz2 mysql/ # 打包压缩到 root目录下

**4) 删除掉数据目录下的所有数据**

rm -rf /var/lib/mysql/

**5)恢复数据 (使用tar命令)**

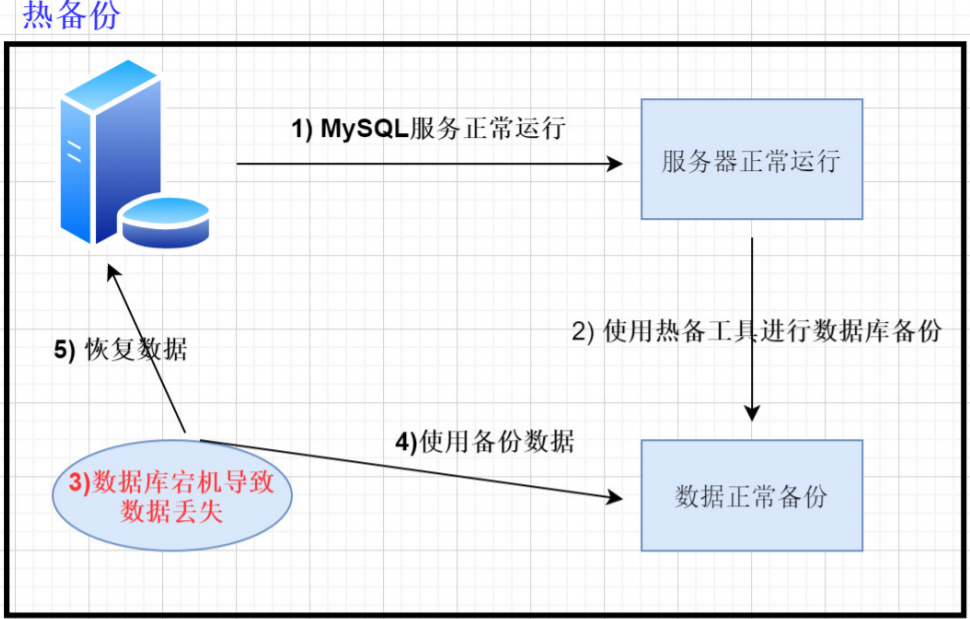
tar jxvf backup.tar.bz2 mysql/ -- 解压

mv /root/mysql/ /var/lib/ -- 把备份的文件移动到/var/lib/里面去替代原来的mysql

**6) 启动MySQL, 然后登陆MySQL,查看数据是否丢失, 如果数据正常代表冷备成功**

**2.3.2 热备份**

热备份是在数据库运行的情况下，备份数据库操作的sql语句，当数据库发生问题时，可以重新执行一遍备份的sql语句。



**热备份的优点:**

（1）可在表空间或数据文件级备份，备份时间短。

（2）备份时数据库仍可使用。

（3）可达到秒级恢复（恢复到某一时间点上）。

**热备份的缺点:**

（1）不能出错，否则后果严重。

（2）因难维护，所以要特别仔细小心，不允许“以失败而告终”。

2.3.1.2 热备份实战

**mysqldump备份工具**

mysqldump是MySQL数据库用来备份和数据转移的一个工具，一般在数据量很小的时候(几个G)可以用于备份。热备可以对多个库进行备份，可以对单张表或某几张表进行备份。

**备份单个数据库**

（1）创建文件夹 , 备份数据

[root@localhost ~]# mkdir databackup

[root@localhost databackup]# mysqldump -uroot -p lagou\_edu **>** lagou\_edu.sql

（2）模拟数据丢失，删除数据库,然后重新创建一个新的库.

DROP DATABASE lagou\_edu;

CREATE DATABASE lagou\_edu CHARACTER SET 'utf8';

（3）恢复数据

[root@localhost databackup]cd databackup

[root@localhost databackup]# mysql -uroot -p lagou\_edu **<** lagou\_edu.sql

**备份数据库的某些表**

（1）备份表数据

[root@localhost databackup]# mysqldump -uroot -p lagou\_edu course course\_lesson **>** backupTable.sql

（2）模拟数据丢失，删除数据表

DROP TABLE course;

DROP TABLE course\_lesson;

（3）恢复数据

mysql -uroot -p lagou\_edu < backupTable.sql

**直接将MySQL数据库压缩备份**

（1）备份数据

mysqldump -uroot -p lagou\_edu | gzip > lagou\_edu.sql.gz

（2）模拟数据丢失，删除数据库

DROP DATABASE lagou\_edu;

CREATE DATABASE lagou\_edu CHARACTER SET 'utf8';

（3）恢复数据

gunzip < lagou\_edu.sql.gz | mysql -uroot -p lagou\_edu

**3.MySQL查询和慢查询日志分析**

**3.1 SQL性能下降的原因**

**等待时间长：**

锁表导致查询一直处于等待状态，后续我们从MySQL锁的机制去分析SQL执行的原理

**执行时间长：**

（1）查询语句写的烂

（2）索引失效

（3）关联查询太多join

（4）服务器调优及各个参数的设置

**3.2 需要遵守的优化原则**

查询优化是一个复杂的工程，涉及从硬件到参数配置、不同数据库的解析器、优化器实现、SQL语句的执行顺序、索引以及统计信息的采集等等方面.

**第一: 只返回需要的结果**

（1）一定要为查询语句指定WHERE条件，过滤掉不需要的数据行

（2）避免使用select \* from, 因为它表示查询表中的所有字段

**第二: 确保查询使用了正确的索引**

（1）经常出现在WHERE条件中的字段建立索引,可以避免全表扫描；

（2）将ORDER BY排序的字段加入到索引中，可以避免额外的排序操作；

（3）多表连接查询的关联字段建立索引，可以提高连接查询的性能；

（4）将GROUP BY分组操作字段加入到索引中，可以利用索引完成分组。

**第三: 避免让索引失效**

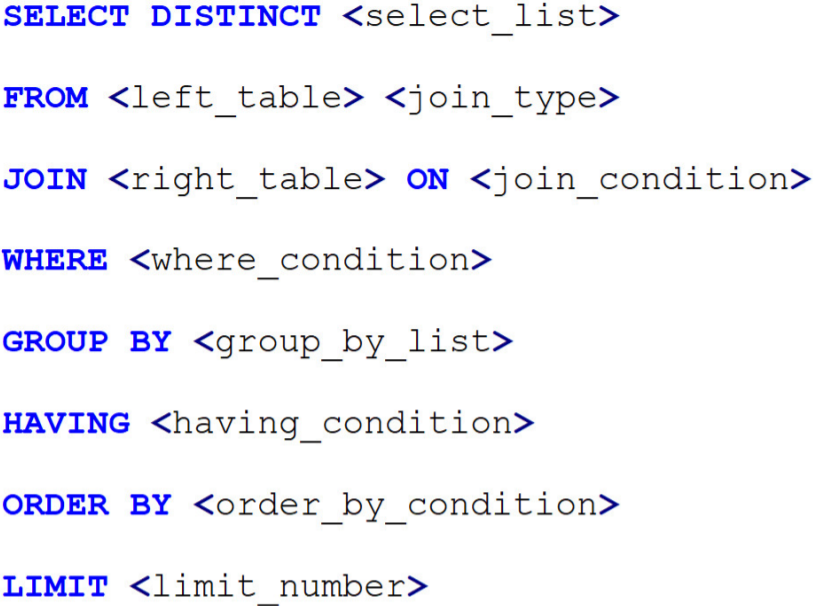
（1）在WHERE子句中对索引字段进行表达式运算或者使用函数都会导致索引失效

（2）使用LIKE匹配时，如果通配符出现在左侧无法使用索引

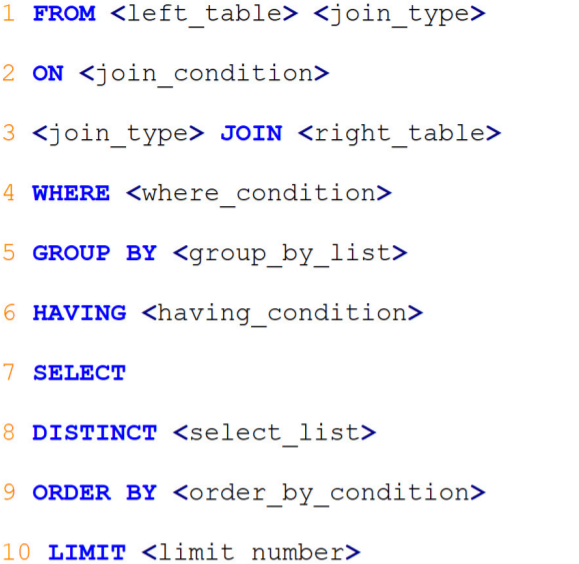
（3）如果WHERE条件中的字段上创建了索引，尽量设置为NOT NULL

**3.3 SQL的执行顺序**

程序员编写的SQL



MySQL执行的SQL



1. FORM子句: 左右两个表的笛卡尔积

2. ON: 筛选满足条件的数据

3. JOIN：如果是inner join那就正常,如果是outer join则会添加回来上面一步过滤掉的一些行

4. WHERE：对不满足条件的行进行移除, 并且不能恢复

5. GROUP BY: 分组后只能得到每组的第一行数据,或者聚合函数的数值

6. HAVING：对分组后的数据进行筛选

7. SELECT：执行select操作，获取需要的列。

8. DISTINCT：去重

9. ORDER BY: 排序

10. LIMIT：取出指定行的记录, 并将结果返回。

**3.4 JOIN查询的七种方式**

7中JOIN ,可以分为四类: **内连接、左连接、右连接、全连接**

**3.4.1 内连接**

SELECT \* FROM A a INNER JOIN B b ON a.id = b.id

**3.4.2 左连接**

SELECT \* FROM A a LEFT JOIN B b ON a.id = b.id

**3.4.3 左连接去重叠部分**

SELECT \* FROM A a LEFT JOIN B b ON a.id = b.id WHERE b.id IS NULL;

**3.4.4 右连接**

SELECT \* FROM A a RIGHT JOIN B b ON a.id = b.id

**3.4.5 右连接去重叠部分**

SELECT \* FROM A a RIGHT JOIN B b ON a.id = b.id WHERE a.id IS NULL;

**3.4.6 全连接**

SELECT \* FROM A a LEFT JOIN B b ON a.id = b.id

UNION

SELECT \* FROM A a RIGHT JOIN B b ON a.id = b.id

**3.4.7 全连接 各自独有**

SELECT \* FROM A a LEFT JOIN B b ON a.id = b.id WHERE b.id IS NULL

UNION

SELECT \* FROM A a RIGHT JOIN B b ON a.id = b.id WHERE a.id IS NULL

**4.MySQL存储引擎**

**4.1 存储引擎介绍**

**存储引擎：存储引擎就是如何存储数据、如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。就像汽车的发动机一样, 存储引擎好坏决定的数据库提供的功能和性能**

**存储引擎的作用：**并发性；事务支持；引用完整性；索引支持

**4.2 常见的3种存储引擎**

MySQL给用户提供了很多种类的存储引擎, 主要分两大类:

事务安全表: InnoDB

非事务安全表: MyISAM、MEMORY、MERGE、EXAMPLE、NDB Cluster、ARCHIVE、 CSV、BLACKHOLE、FEDERATED等。

查看MySQL数据的存储引擎有哪些：SHOW ENGINES;

查看当前的默认存储引擎 (MySQL5.7 默认使用 InnoDB)：SHOW VARIABLES LIKE '%default\_storage\_engine%';

在MySQL中,不需要整个服务器都是用同一种引擎,针对具体的需求,可以对**每一个表使用不同的存储引擎** .并且想要进一步优化, 还可以自己编写一个存储引擎.

**create table(...) engine=MyISAM;**

**4.2.1 InnoDB(推荐)**

InnoDB是一个健壮的事务型存储引擎，这种存储引擎已经被很多互联网公司使用，为用户操作非常大的数据存储提供了一个强大的解决方案。InnoDB还引入了行级锁定和外键约束，在以下场合下，使用 InnoDB是最理想的选择

**优点：**

（1）Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别

（2）支持多版本并发控制的行级锁，由于锁粒度小，写操作和更新操作并发高、速度快。

（3）支持自增长列。

（4）支持外键

（5）适合于大容量数据库系统，支持自动灾难恢复。

**缺点：**

它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表

**应用场景：**

（1）当需要使用数据库事务时，该引擎当然是首选。由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用Innodb引擎会提升效率

（2）更新密集的表, InnoDB存储引擎特别适合处理多重并发的更新请求。

**4.2.2 MyISAM**

MyISAM引擎, 不支持事务、也不支持外键，优势是访问速度快，对事务完整性没有要求或者以select，insert为主的应用基本上可以用这个引擎来创建表

**优点：**

（1）MyISAM表是独立于操作系统的，这说明可以轻松地将其从Windows服务器移植到Linux服务器。

（2）MyISAM存储引擎在查询大量数据时非常迅速，这是它最突出的优点

（3）另外进行大批量插入操作时执行速度也比较快。

**缺点：**

（1）MyISAM表没有提供对数据库事务的支持。

（2）不支持行级锁和外键。

（3）不适合用于经常UPDATE(更新)的表，效率低。

**应用场景：**

（1）以读为主的业务，例如：图片信息数据库，博客数据库，商品库等业务。

（2）对数据一致性要求不是非常高的业务（不支持事务）

（3）硬件资源比较差的机器可以用MyiSAM（占用资源少）

**4.2.3 MEMORY**

MEMORY的特点是 将表中的数据放在内存中，适用于存储临时数据的临时表和数据仓库中的纬度表

**优点：**

memory类型的表访问非常的快，因为它的数据是放在内存中的

**缺点：**

（1）一旦服务关闭，表中的数据就会丢失掉。

（2）只支持表锁，并发性能差，不支持TEXT和BLOB列类型，存储varchar时是按照char的方式

**应用场景：**

（1）目标数据较小，而且被非常频繁地访问。

（2）如果数据是临时的，而且要求必须立即可用，那么就可以存放在内存表中。

（3）存储在Memory表中的数据如果突然丢失，不会对应用服务产生实质的负面影响。

**4.3 如何选择存储引擎**

不同的存储引擎都有各自的特点，以适应不同的需求，如表所示。为了做出选择，首先要考虑每一个存储引擎提供了哪些不同的功能。

|  |  |  |  |
| --- | --- | --- | --- |
| 特性 | InnoDB | MyISAM | MEMORY |
| 存储限制(Storage limits) | 64TB | No | Yes |
| 支持事物(Transactions) | Yes | No | No |
| 锁机制(Locking granularity) | 行锁 | 表锁 | 表锁 |
| B树索引(B-tree indexes) | Yes | Yes | Yes |
| 哈希索引(Hash indexes) | Yes | No | Yes |
| 外键支持(Foreign key support) | Yes | No | No |
| 存储空间消耗(Storage Cost) | 高 | 低 | 低 |
| 内存消耗(Memory Cost) | 高 | 低 | 高 |
| 批量数据写入效率(Bulk insert speed) | 慢 | 快 | 快 |

提供几个选择标准，然后按照标准，选择对应的存储引擎：

（1）是否需要支持事务；

（2）崩溃恢复，能否接受崩溃；

（3）是否需要外键支持；

（4）存储的限制；

（5）对索引和缓存的支持；

**5.MySQL索引优化**

**5.1 索引简介**

**5.1.1 什么是索引**

索引就是排好序的,帮助我们进行快速查找的数据结构.

简单来讲，索引就是一种将数据库中的记录按照特殊形式存储的数据结构。通过索引，能够显著地提 数据查询的效率，从而提升服务器的性能.

专业一点来说呢，索引是一个排好序的列表，在这个列表中存储着索引的值和包含这个值的数据所在行的物理地址。在数据库十分庞大的时候，索引可以大大加快查询的速度，**这是因为使用索引后可以不用扫描全表来定位某行的数据，而是先通过索引表找到该行数据对应的物理地址然后访问相应的数据。**

1. 没有用索引时执行 select \* from where t.Col2 = , 数据从磁盘一条一条拿去最终找到结果，效率低下。

2. 为了加快查找,可以维护一个二叉树,左侧节点小于父节点, 右侧节点大于父节点,每个节点分别保存字段数据和一个指向对应数据记录物理地址的指针.

3. 查找时就可以使用二叉树查找获取相应的数据,从而快速检索出符合条件的记录

**一般来说索引本身也比较大,不可能全部保存在内存中,因此索引通常是以索引文件的形式存储在磁盘上**

**5.1.2 索引的种类**

**普通索引：**这是最基本的索引类型，基于普通字段建立的索引，没有任何限制。

CREATE INDEX <索引的名字> ON tablename (字段名);

ALTER TABLE tablename ADD INDEX [索引的名字] (字段名);

CREATE TABLE tablename ( [...], INDEX [索引的名字] (字段名) );

**唯一索引：**与"普通索引"类似，不同的就是：索引字段的值必须唯一，但允许有空值 。

CREATE UNIQUE INDEX <索引的名字> ON tablename (字段名);

ALTER TABLE tablename ADD UNIQUE INDEX [索引的名字] (字段名);

CREATE TABLE tablename ( [...], UNIQUE [索引的名字] (字段名) ;

**主键索引：**它是一种特殊的唯一索引，不允许有空值。在创建或修改表时追加主键约束即可，每个表只能有一个主键。

CREATE TABLE tablename ( [...], PRIMARY KEY (字段名) );

ALTER TABLE tablename ADD PRIMARY KEY (字段名);

**复合索引：**用户可以在多个列上建立索引，这种索引叫做组复合索引(组合索引)。复合索引可以代替多个单一索引，相比多个单一索引复合索引所需的开销更小。

CREATE INDEX <索引的名字> ON tablename (字段名1，字段名2...);

ALTER TABLE tablename ADD INDEX [索引的名字] (字段名1，字段名2...);

CREATE TABLE tablename ( [...], INDEX [索引的名字] (字段名1，字段名2...) );

**复合索引使用注意事项：**

1. 何时使用复合索引，要根据where条件建索引，注意不要过多使用索引，过多使用会对更新操作效率有很大影响。

2. 如果表已经建立了(col1，col2)，就没有必要再单独建立（col1）；如果现在有(col1)索 引，如果查询需要col1和col2条件，可以建立(col1,col2)复合索引，对于查询有一定提高。

**全文索引：**查询操作在数据量比较少时，可以使用like模糊查询，但是对于大量的文本数据检索，效率很低。如果使用全文索引，查询速度会比like快很多倍。在MySQL 5.6 以前的版本，只有MyISAM存储引擎支持全文索引，从MySQL 5.6开始MyISAM和InnoDB存储引擎均支持。

CREATE FULLTEXT INDEX <索引的名字> ON tablename (字段名);

ALTER TABLE tablename ADD FULLTEXT [索引的名字] (字段名);

CREATE TABLE tablename ( [...], FULLTEXT KEY [索引的名字] (字段名) ;

全文索引方式有自然语言检索IN NATURAL LANGUAGE MODE和布尔检索IN BOOLEAN MODE两种

**和常用的like模糊查询不同，全文索引有自己的语法格式，使用 match 和 against 关键字，比如**

**SELECT \* FROM users3 WHERE MATCH(NAME) AGAINST('aabb');**

**SELECT \* FROM users3 WHERE MATCH(NAME) AGAINST('aa\*' IN BOOLEAN MODE);**

**全文索引使用注意事项：**

（1）全文索引必须在字符串、文本字段上建立。

（2）全文索引字段值必须在最小字符和最大字符之间的才会有效。（innodb：3-84； myisam：4-84）

**5.1.3 索引的优势与劣势**

**优点：**

（1）提高数据检索的效率,降低数据库的IO成本

（2）通过索引列对数据进行排序,降低数据排序的成本,降低了CPU的消耗

**缺点：**

（1）创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加

（2）索引需要占物理空间，除了数据表占用数据空间之外，每一个索引还要占用一定的物理空间

（3）当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度

**创建索引的原则：**

（1）在经常需要搜索的列上创建索引，可以加快搜索的速度；

（2）在作为主键的列上创建索引，强制该列的唯一性和组织表中数据的排列结构；

（3）在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

（4）在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

（5）在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

（6）在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

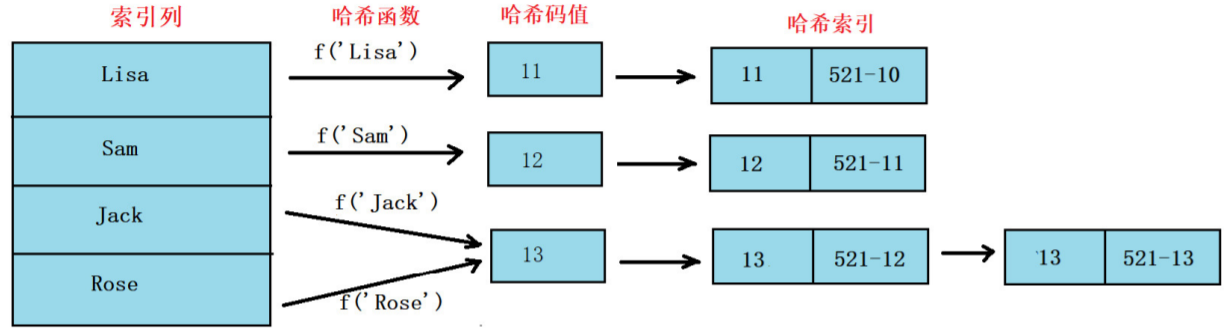
**5.2 索引原理**

**MySQL中索引的常用数据结构有两种，一种是Hash，另一种是BTree。**

**5.2.1 HASH结构**

Hash底层实现是由Hash表来实现的，是根据键值< key,value >存储数据的结构。非常适合根据 key查找value值，也就是单个key查询，或者说等值查询。

对于每一行数据，存储引擎都会对所有的索引列计算一个哈希码，哈希码是一个较小的值，并且不同键值的行计算出来的哈希码也不一样。



**Hash索引的缺点**

（1）哈希索引只包含哈希值和行指针，而不存储字段值，所以不能使用索引中的值来避免读取行。

（2）哈希索引数据并不是按照索引值顺序存储的，所以也就无法用于排序。

（3）哈希索引只支持等值比较查询。不支持任何范围查询和部分索引列匹配查找。

**Hsah索引的优点：**

（1）**只需要做等值比较查询，而不包含排序或范围查询的需求，都适合使用哈希索引**

（2）访问哈希索引的数据非常快，除非有很多哈希冲突。

**5.2.2 B+Tree结构**

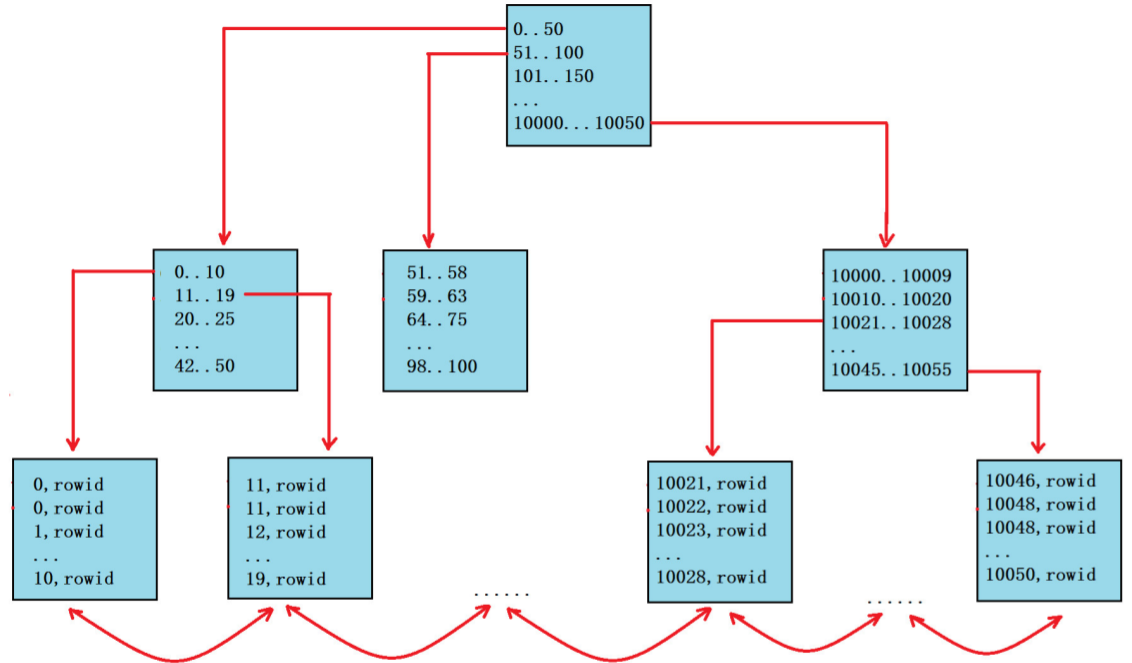
MySQL数据库索引采用的是B+Tree结构，在B-Tree结构上做了优化改造。

**B+Tree结构**

非叶子节点不存储data数据，只存储索引值，这样便于存储更多的索引值

叶子节点包含了所有的索引值和data数据

叶子节点用指针连接，提高区间的访问性能



**B树索引的应用**

全键值查询 where x=123

键值范围查询 where 45 < x < 123

**6.MySQL锁机制**

**6.1 MySQL锁概述**

数据库锁定机制简单来说，就是数据库为了保证数据的一致性，而使各种共享资源在被并发访问变得有序所设计的一种规则.

**6.2 MySQL的锁分类**

MySQL数据库由于其自身架构的特点,存在多种数据存储引擎, MySQL中不同的存储引擎支持不同的锁机制。

**MyISAM和MEMORY存储引擎采用的表级锁，**

**InnoDB存储引擎既支持行级锁，也支持表级锁，默认情况下采用行级锁。**

**BDB采用的是页面锁，也支持表级锁**

**按照数据操作的类型分：**

（1）**读锁(共享锁)**：针对同一份数据，多个读操作可以同时进行而不会互相影响。

（2）**写锁(排他锁)**：当前写操作没有完成前，它会阻断其他写锁和读锁。

**按照数据操作的粒度分：**

（1）**表级锁**：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。

（2）**行级锁**： 开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

（3）**页面锁**：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般

**按照操作性能可分为乐观锁和悲观锁：**

（1）**乐观锁**：一般的实现方式是对记录数据版本进行比对，在数据更新提交的时候才会进行冲突检测，如果发现冲突了，则提示错误信息。

（2）**悲观锁**：在对一条数据修改的时候，为了避免同时被其他人修改，在修改数据之前先锁定，再修改的控制方式。共享锁和排他锁是悲观锁的不同实现，但都属于悲观锁范畴。

**6.3 表级锁(偏读)**

表级锁是MySQL中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单，资源消耗较少，被大部分MySQL引擎支持。最常使用的MYISAM与INNODB都支持表级锁定。

表级锁定分为: **表共享读锁（共享锁）与表独占写锁（排他锁）。**

特点: 开销小，加锁快；不会出现死锁；锁定粒度大，发出锁冲突的概率最高，并发度最低。

**6.3.1 加锁语法**

**查看表中加过的锁：**

SHOW OPEN TABLES; -- 0表示没有加锁,当前的所有数据库表都没有加锁

SHOW OPEN TABLES WHERE In\_use > 0; -- 查询加锁的表,条件In\_use 大于0

**手动增加表锁：**

语法格式: **LOCK TABLE 表名 READ(WRITE), 表名2 READ(WRITE), 其他;**

lock table mylock01 read,mylock02 write; -- 为mylock01加读锁(共享锁) , 给mylock02加写锁(排他锁)

**释放锁, 解除锁定：**

unlock tables; -- 方式1

-- 方式2 找到锁进程,得到id SHOW PROCESSLIST; kill id

**总结: 对MyISAM表的读操作 (加读锁) ,不会阻塞其他进程对同一表的读请求,但是会阻塞对同一表的写请求. 只有当读锁释放后,才会执行其他进程的写操作.**

**总结: 对MyISAM表加写锁, 会阻塞其他进程对同一表的读和写操作,只有当写锁释放后,才会执行其他进程的操作**

**6.4 行级锁(偏写)**

**6.4.1 行级锁介绍**

行锁的是mysql锁中粒度最小的一种锁，因为锁的粒度很小，所以发生资源争抢的概率也最小，并发性能最大，但是也会造成死锁，每次加锁和释放锁的开销也会变大。

**使用MySQL行级锁的两个前提：**

（1）使用innoDB引擎

（2）开启事务(隔离级别为Repeatable Read)

SET autocommit=0; # 开启MySQL数据库手动提交

begin; # 开启事务

commit # 提交事务

**InnoDB行锁的类型：**

（1）**共享锁（S）**：当事务对数据加上共享锁后, 其他用户可以并发读取数据，但任何事务都不能对数据进行修改（获取数据上的排他锁），直到已释放所有共享锁。

（2）**排他锁（X）**：如果事务T对数据A加上排他锁后，则其他事务不能再对数据A加任何类型的封锁。获准排他锁的事务既能读数据，又能修改数据。

**加锁的方式：**

（1）InnoDB引擎默认更新语句，update,delete,insert只要开启事务都会自动给涉及到的数据加上排他锁

（2）select语句默认不会加任何锁类型，如果要加可以使用下面的方式:

加共享锁（S）：select \* from table\_name where ... **lock in share mode**;

加排他锁（x）：select \* from table\_name where ... **for update**;

**锁兼容：**

（1）共享锁只能兼容共享锁, 不兼容排它锁

（2）排它锁互斥共享锁和其它排它锁

|  |  |  |
| --- | --- | --- |
|  | 共享锁 | 排他锁 |
| 共享锁 | 兼容 | 冲突 |
| 排他锁 | 冲突 | 冲突 |

**行级锁都是基于索引的，如果一条SQL语句用不到索引是不会使用行级锁的，而会使用表级锁把整张表锁住，这点需要咱们格外的注意**