

# 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。

- 阿里数据中心内景(阿里、百度、腾讯这样的互联网巨头，数据量据说已经接近EB级)

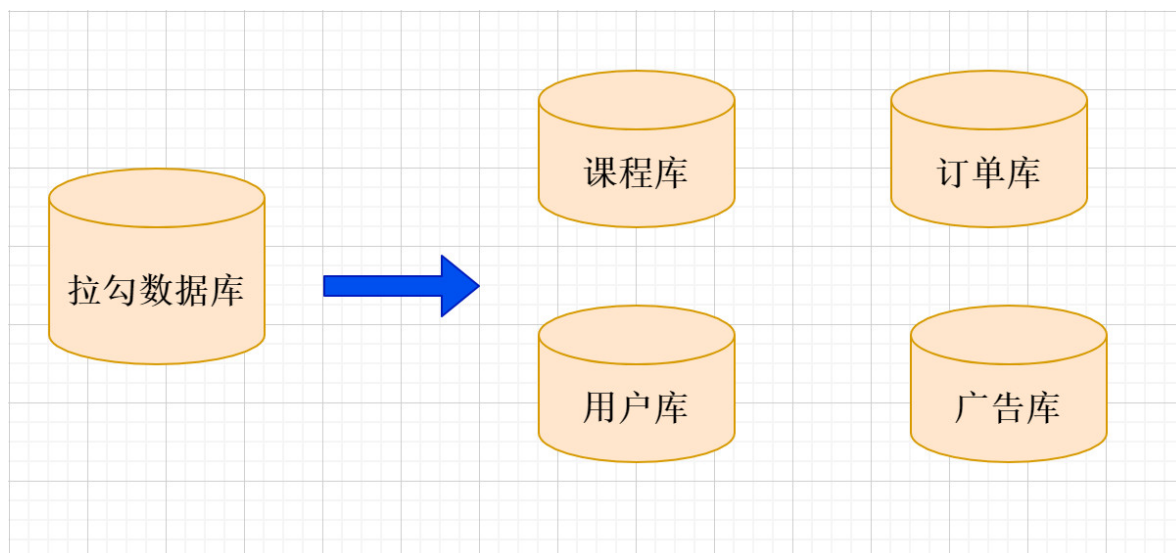


- **解决方案1:** 使用NoSQL数据库, 通过降低数据的安全性, 减少对事务的支持, 减少对复杂查询的支持, 来获取性能上的提升。
- **解决方案2:** NoSQL并不是万能的, 就比如有些使用场景是绝对要有事务与安全指标的, 所以还是要用关系型数据库, 这时候就需要搭建MySQL数据库集群, 为了提高查询性能, 将一个数据库的数据分散到不同的数据库中存储, 通过这种数据库拆分的方法来解决数据库的性能问题。

### 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

- 单机存储容量遇到瓶颈.
- 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.

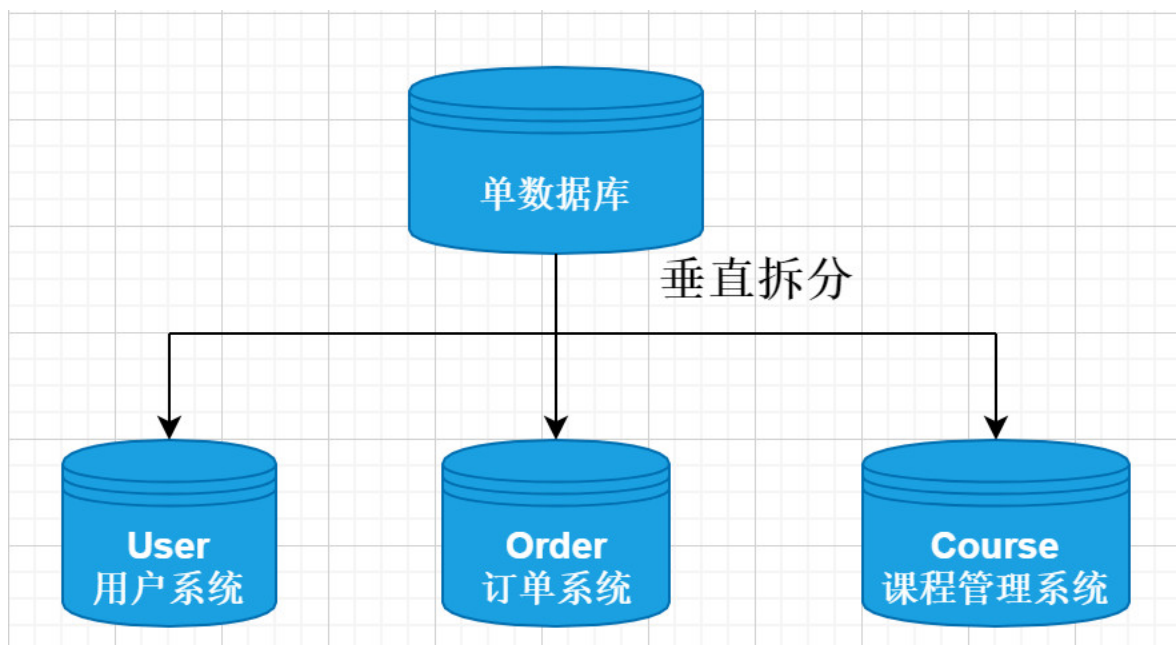
数据量不够就不要分表,单表数据量超过1000万或100G的时候, 速度就会变慢(官方测试),

## 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。

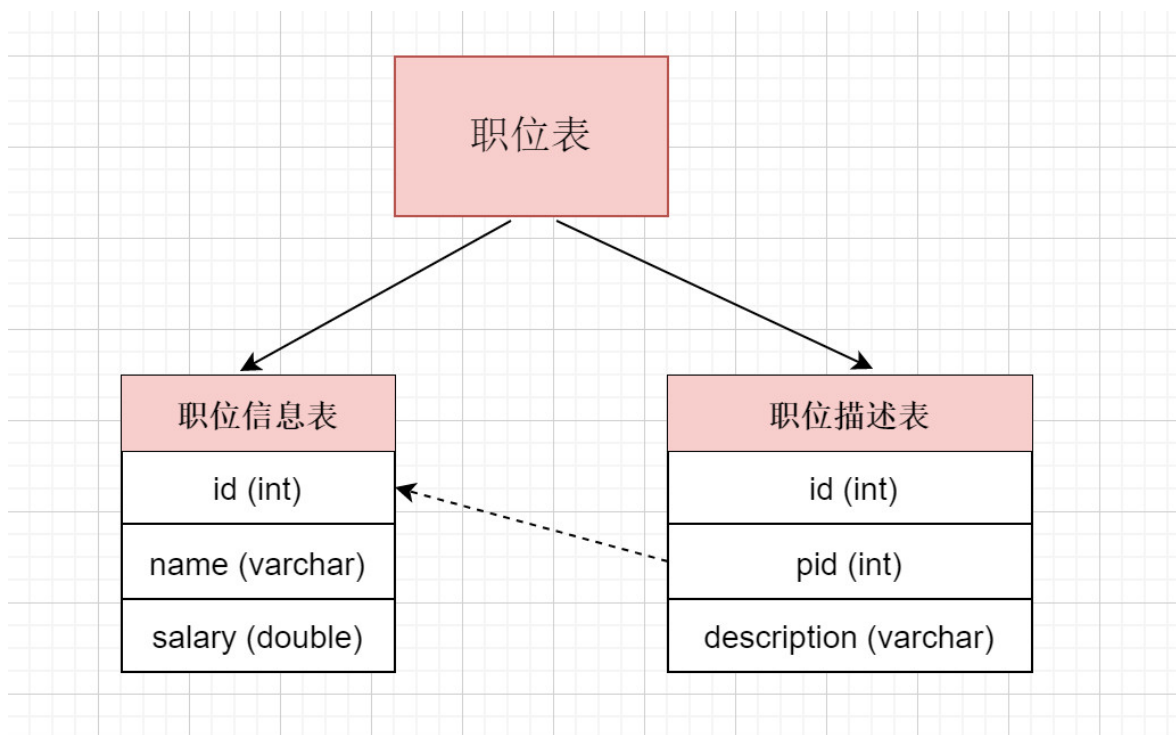
### 1.2.2.1 垂直分库

- 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

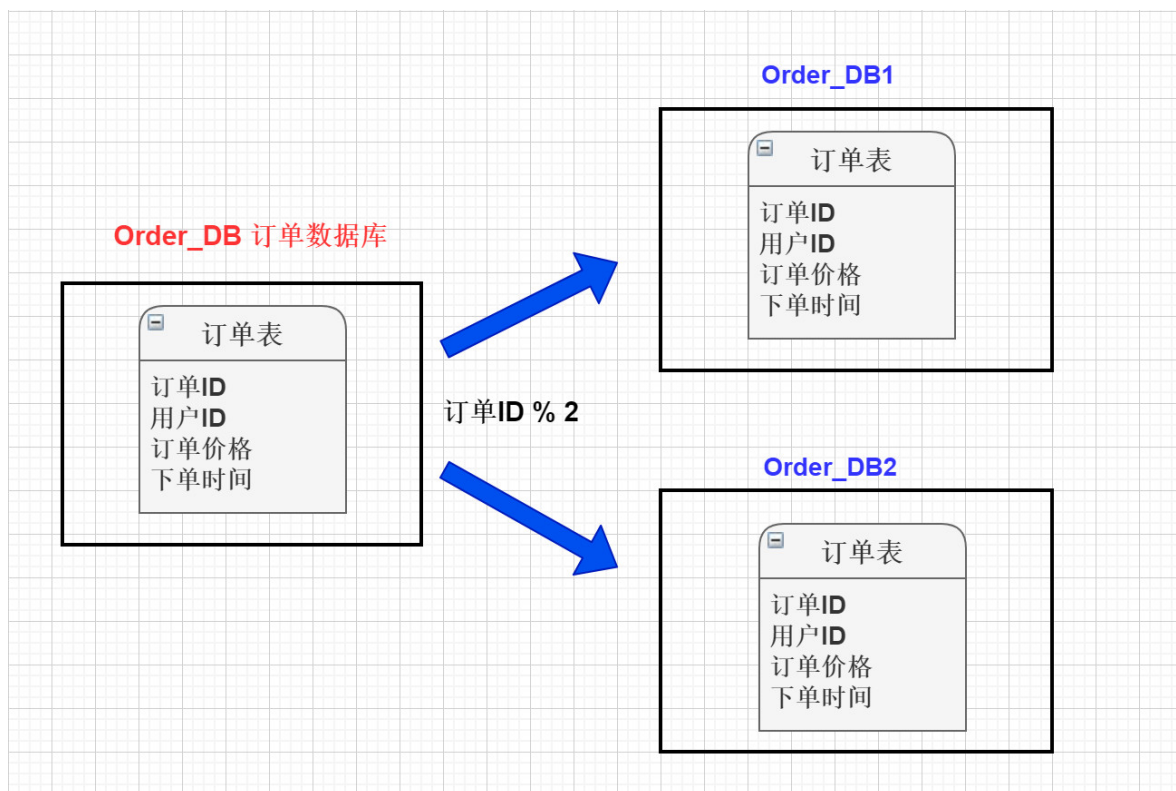
- 将一个表按照字段分成多表，每个表存储其中一部分字段。
  - 对职位表进行垂直拆分，将职位基本信息放在一张表，将职位描述信息存放在另一张表



- 垂直拆分带来的一些提升
  - 解决业务层面的耦合，业务清晰
  - 能对不同业务的数据进行分级管理、维护、监控、扩展等
  - 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

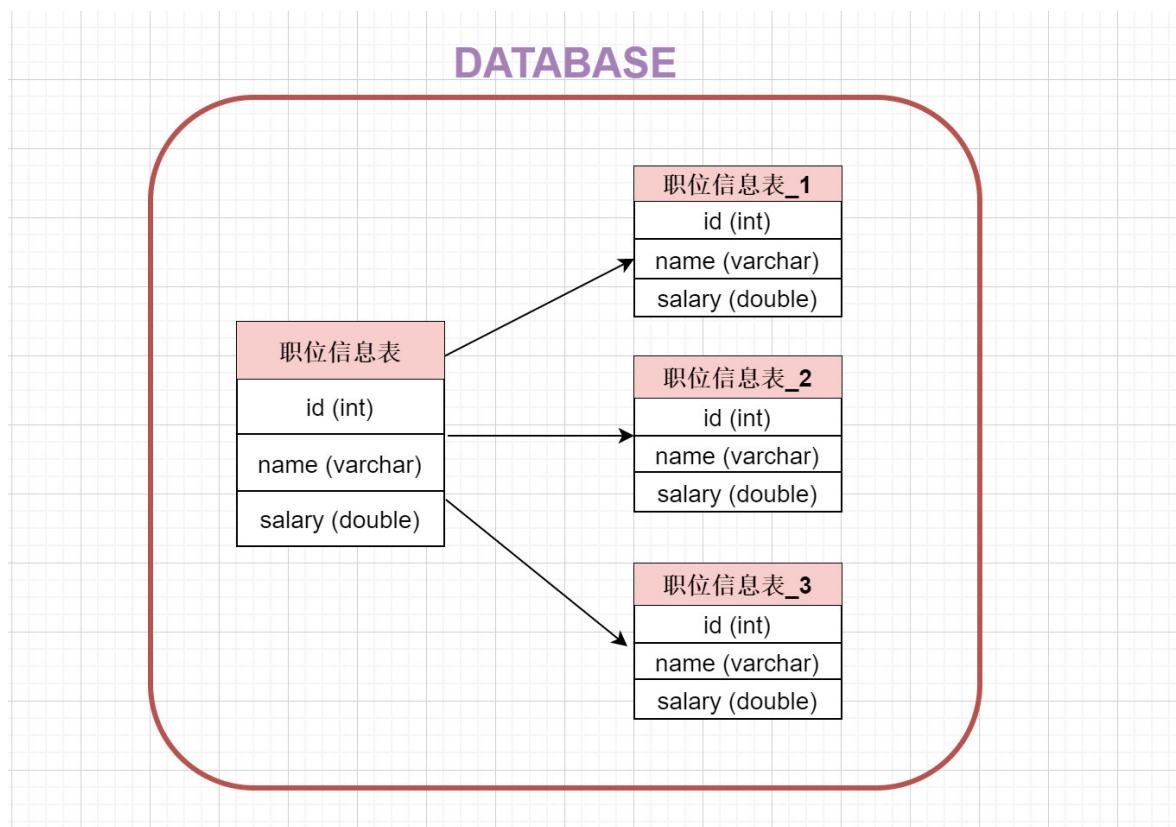
### 1.2.2.3 水平分库

- 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。接下来了解一下什么是**mycat**。

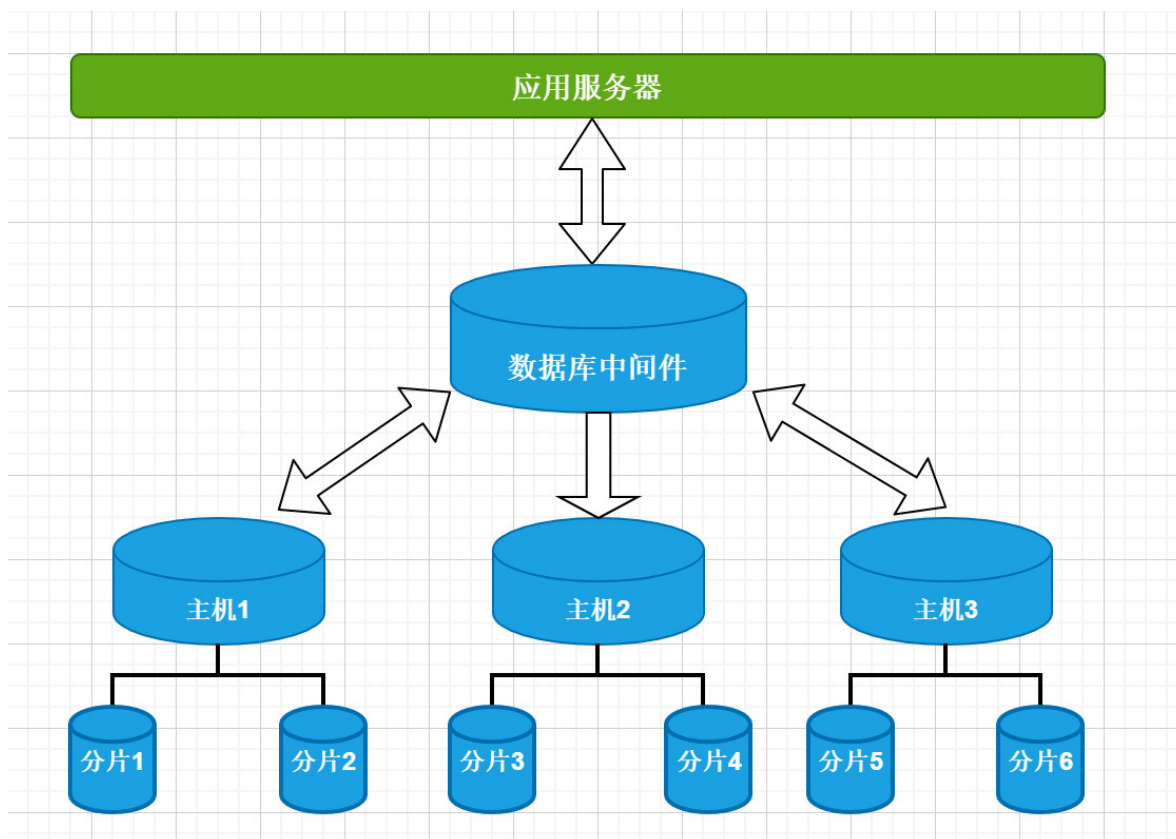
## 2. MyCat

### 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 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)

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

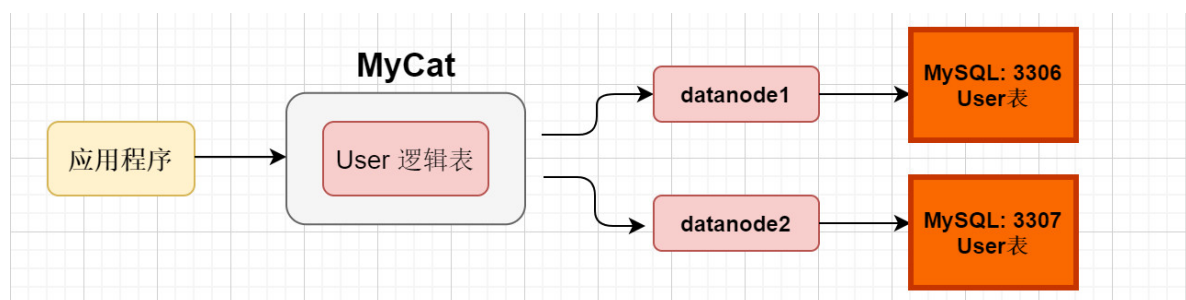
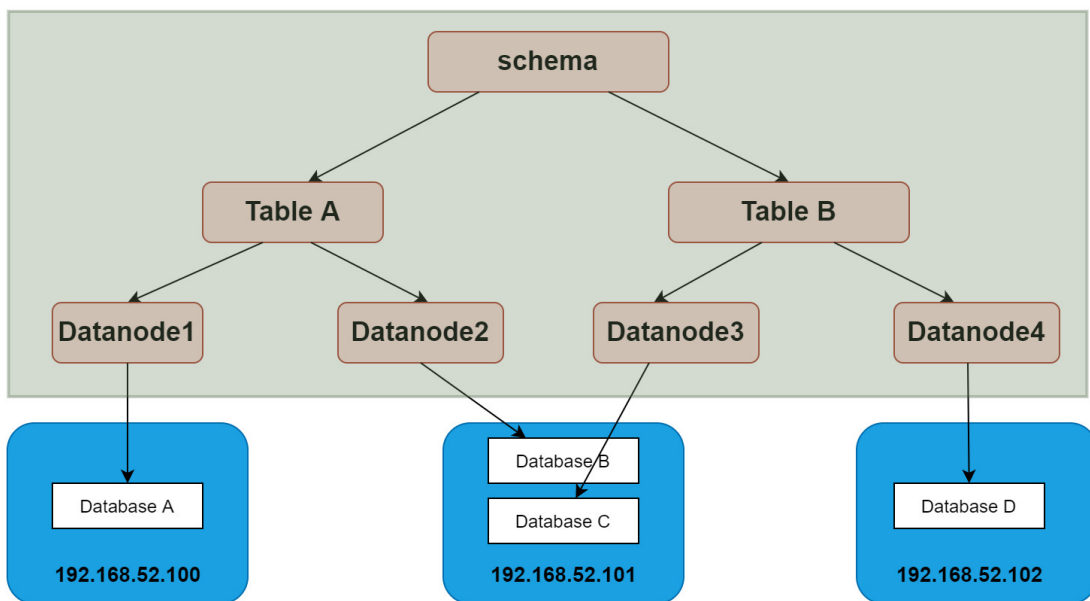


- mongoDB
- SQLServer

## 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
  - 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。



- **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**

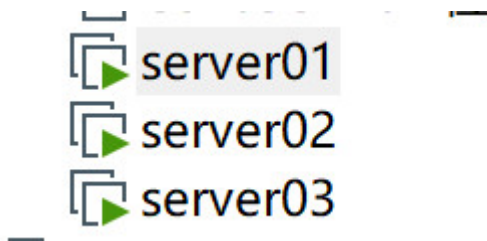
前面讲了数据切分，一个大表被分成若干个分片表，就需要一定的规则rule，这样按照某种业务规则把数据分到 某个分片的规则就是分片规则，数据切分选择合适的分片规则非常重要，将极大的避免后续数据处理的难度。

## 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/>

第一步: 搭建3台虚拟机

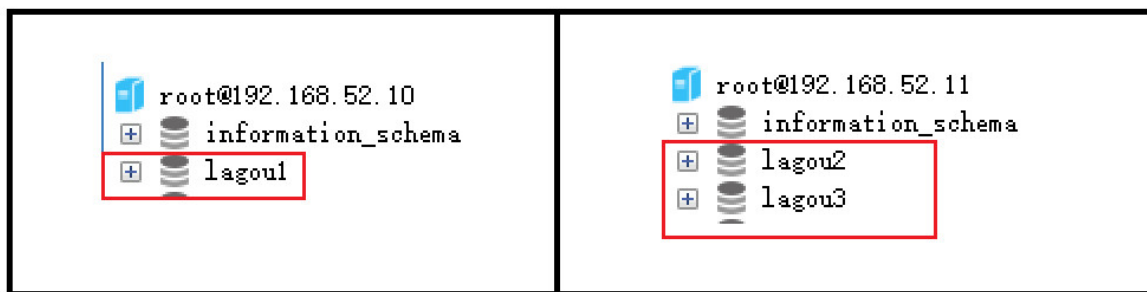


第二步: server01与server02 安装MySQL数据库服务器,保证版本一致

```
server01 192.168.52.10
server02 192.168.52.11
```

第三步: 创建数据库


- 192.168.52.10 创建 **lagou1** 数据库
- 192.168.52.11 创建 **lagou2** 和 **lagou3** 数据库



## 2.4.2 MyCat安装

注意: 提前安装好DK

- 第一步: 下载MyCat

 [Mycat-server-1.6.7.5-release-20200410174409-linux.tar.gz](#)

- 第二步: 上传MyCat 到 **server03** 服务器 ,并解压

```
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标签

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。

```
<!-- 逻辑库 -->
<schema name="lagou" checkSQLschema="true" sqlMaxLimit="100" >
</schema>
```

属性名	值	数量限制	说明
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



```
<schema name="lagou" checkSQLSchema="true" sqlMaxLimit="100" >
  <table name="pay_order" dataNode="dn1,dn2,dn3" rule="auto-sharding-long"
    primaryKey="id" autoIncrement="true" ></table>
</schema>
```

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，也就是我们通常所说的数据分片。

```
<dataNode name="dn1" dataHost="localhost1" database="lagou1" />
<dataNode name="dn2" dataHost="localhost2" database="lagou2" />
<dataNode name="dn3" dataHost="localhost2" database="lagou3" />
```

- 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

```
<!-- 节点主机 -->
<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
  writeType="0" dbType="mysql" dbDriver="native" switchType="1"
  slaveThreshold="100">

  <heartbeat>select user()</heartbeat>

  <writeHost host="hostM1" url="192.168.52.10:3306" user="root"
    password="QiDian@666">
  </writeHost>
</dataHost>

<dataHost name="localhost2" maxCon="1000" minCon="10" balance="0"
```

```

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

    <heartbeat>select user()</heartbeat>

    <writeHost host="hostM2" url="192.168.52.11:3306" user="root"
        password="QiDian@666">
    </writeHost>
</dataHost>

```

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

```

<heartbeat>select user()</heartbeat>

```

### 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

```

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

    <heartbeat>select user()</heartbeat>

    <writeHost host="hostM1" url="192.168.52.10:3306" user="root"
password="QiDian@666">

    </writeHost>
</dataHost>

```

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。

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

### 2.5.2.2 连接MyCat

- 重启myCat,查看状态

```
./mycat start
./mycat status
```

- 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

```
<mycat:rule xmlns:mycat="http://io.mycat/">
  <tableRule name="sharding-by-intfile">
    <rule>
      <columns>sharding_id</columns>
      <algorithm>hash-int</algorithm>
    </rule>
  </tableRule>
</mycat:rule>
```

name: 指定唯一的名字，用于标识不同的表规则。

rule: 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: 指定要拆分的列名字。

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

### 2.5.2 function 标签

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

name: 指定算法的名字。

class: 制定路由算法具体的类名字。

property: 为具体算法需要用到的一些属性。

### 2.5.4 常用的分片规则

Mycat常用分片配置示例：

- 自动分片

根据指定的列的范围进行分片.默认从0节点开始

```
<tableRule name="auto-sharding-long">
  <rule>
    <columns>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>
```

autopartition-long.txt文件：

```
0-200000=0
200000-400000=1
```

```
0-200000范围分配给节点0
200000-400000范围分配给节点1
```

- 枚举分片

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

```
<!-- 枚举分片 -->
<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>
```

- mapFile 中是自定义的分片策略文件, 需要自己编写
- partition-hash-int.txt文件内容如下:

```
beijing=0
wuhan=1
shanghai=2
```

- 取模分片

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

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

## 2.6 MyCat分库分表

### 2.6.1 分片规则配置(水平分库)

- **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.
- 在rule.xml配置, 自动分片
  - 每个datanode中保存一定数量的数据。根据id进行分片

```
<!-- schema 逻辑库 -->
<schema name="lagou" checkSQLSchema="true" sqlMaxLimit="100" >
  <table name="pay_order" dataNode="dn1,dn2,dn3" rule="auto-sharding-long"
    primaryKey="id" autoIncrement="true" >
```



```
</table>
</schema>
```

```
=====
=====
```

```
<!-- 自动分片 -->
<tableRule name="auto-sharding-long">
  <rule>
    <columns>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>
```

- autopartition-long.txt

```
# range start-end ,data node index
# K=1000,M=10000.
0-1k=0
1k-2k=1
2k-3k=2
```

- 对分片规则进行测试id 范围为:

**Datanode1:** 1~1000

**Datanode2:** 1000~2000

**Datanode3:** 2000~3000

## 2.6.2 启动MyCat 进行测试

- 重启MyCat

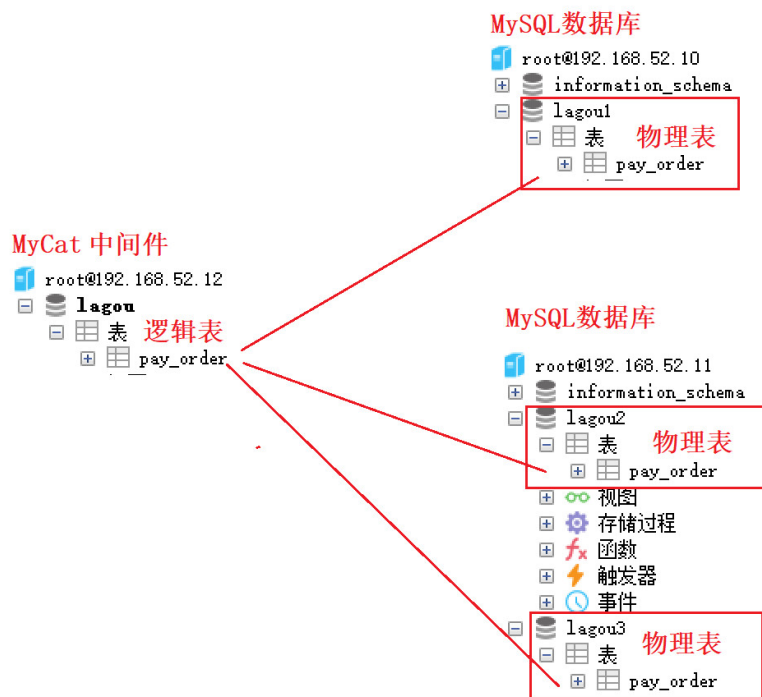
```
停止命令: ./mycat stop
重启命令: ./mycat restart
```

- 在MyCat中创建逻辑表

```
DROP TABLE IF EXISTS pay_order;

CREATE TABLE pay_order (
  id BIGINT(20) PRIMARY KEY,
  user_id INT(11) NOT NULL ,
  product_name VARCHAR(128) ,
  amount DECIMAL(12,2)
);
```

- MyCat中创建好表之后,我们的MySQL节点中也会对应的创建表



- 插入数据,观察数据被插入到哪张表中.

```
INSERT INTO pay_order(id,user_id,product_name,amount) VALUES(2001,1,"面试宝典",15.8);
```

- 注意: 解决MyCat乱码问题

```
<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
  writeType="0" dbType="mysql" dbDriver="native" switchType="1"
  <heartbeat>select user()</heartbeat>
  <writeHost host="hostM1" url="192.168.52.10:3306" user="root"
    password="QiDian@666">
  </writeHost>
</dataHost>
```

修改为native

直接写IP和端口

### 2.6.3 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。

- server.xml文件中

```
<system>
  <property name="sequenceHandlerType">0</property>
</system>
```

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)

### 2.6.3.1 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。

```
PAY_ORDER.HISIDS=  
PAY_ORDER.MINID=101  
PAY_ORDER.MAXID=10000000  
PAY_ORDER.CURID=100
```

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

重启MyCat, 插入一条数据,不用指定id.

```
INSERT INTO pay_order(user_id,product_name,amount) VALUES(1,"xiao",12.8);
```

## 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.

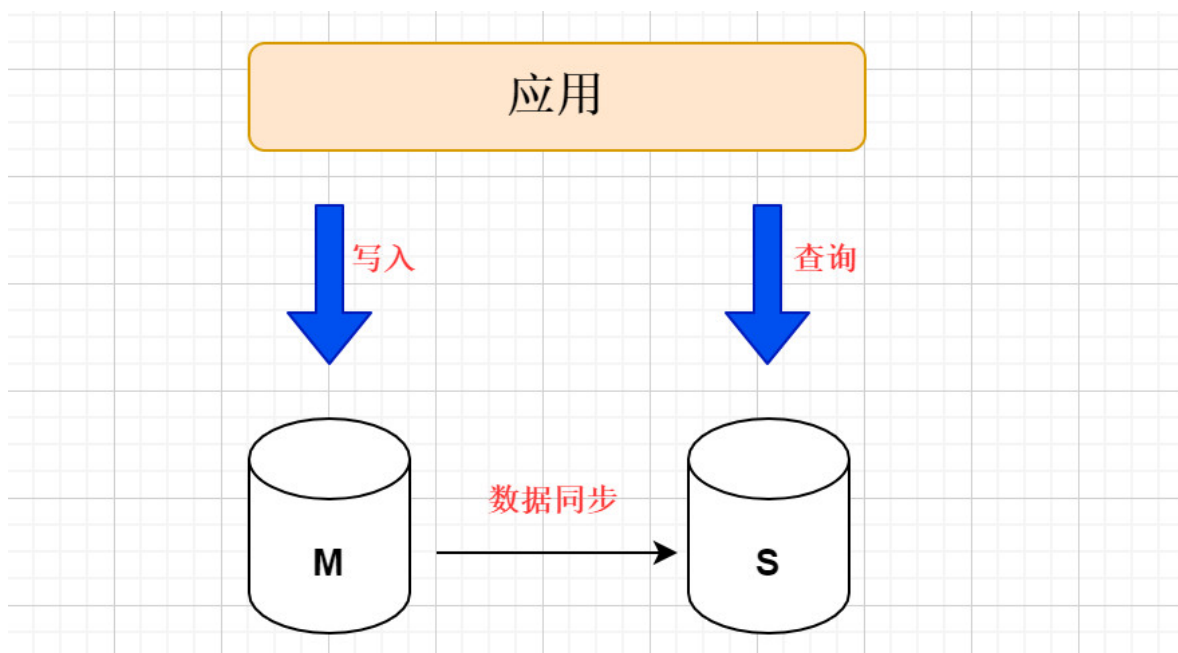
- 主从复制

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

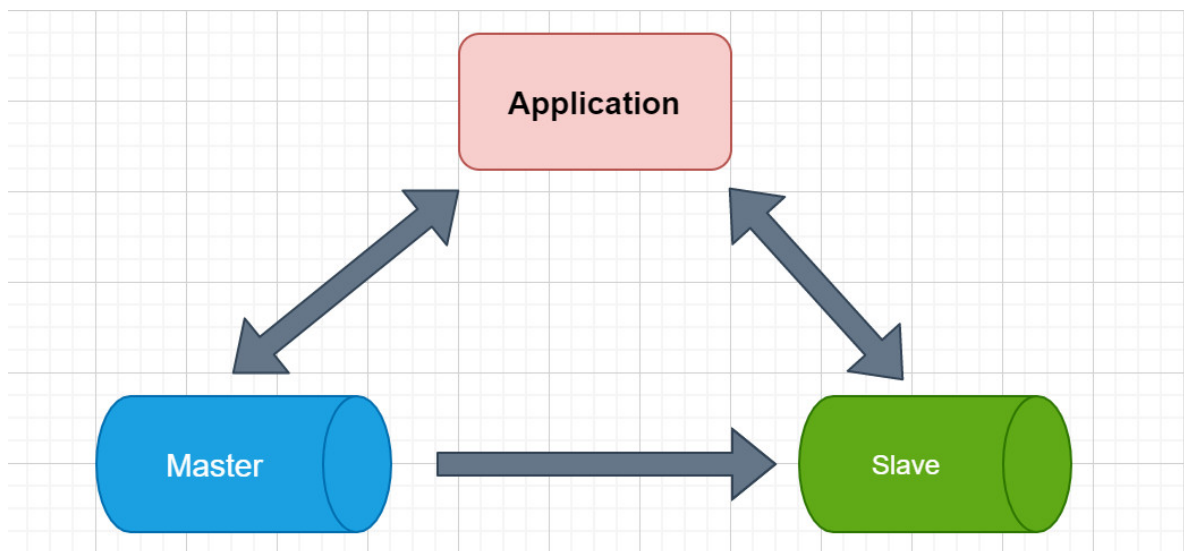
读写分离就是让主库处理事务性操作, 从库处理select查询。数据库复制被用来把事务性查询导致的数据变更同步到从库, 同时主库也可以select查询。

**读写分离的数据节点中的数据内容是一致的。**



### 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

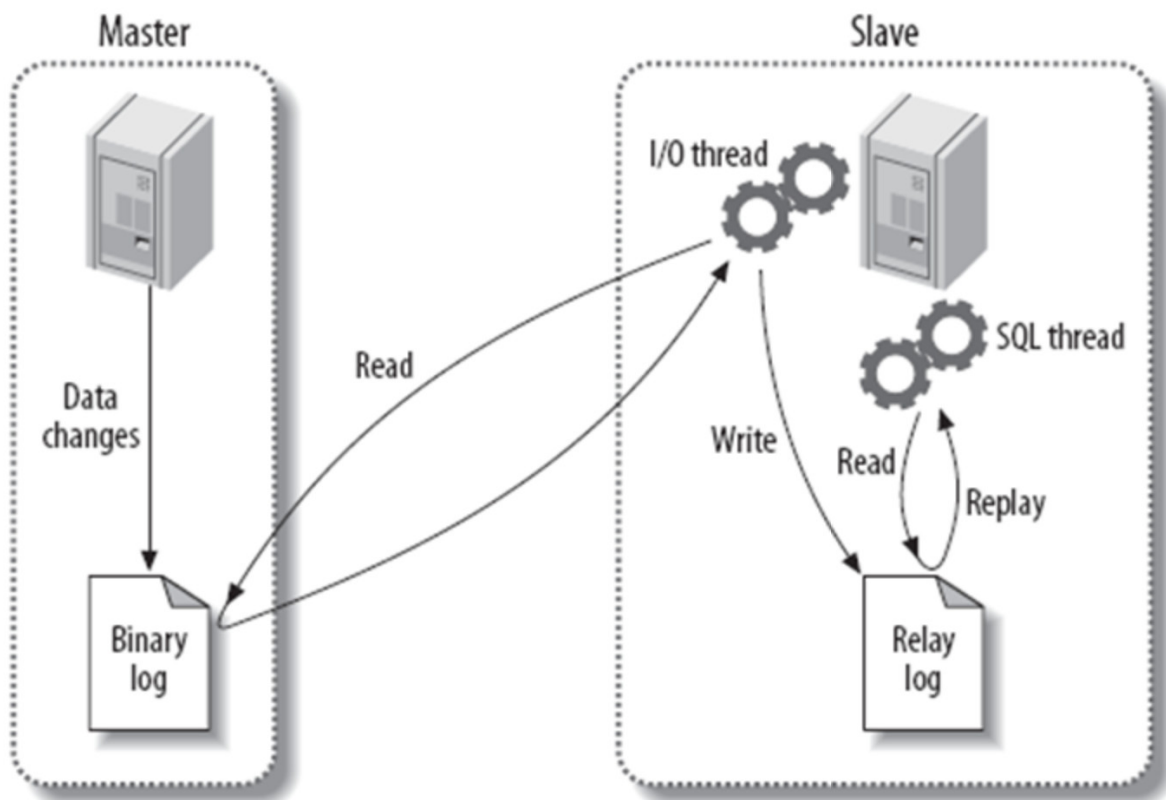
- 实时灾备，用于故障切换
- 读写分离，提供查询服务
- 备份，避免影响业务

### 主从部署必要条件

- 主库开启binlog日志（设置log-bin参数）
- 主从server-id不同
- 从库服务器能连通主库

### 主从复制的原理

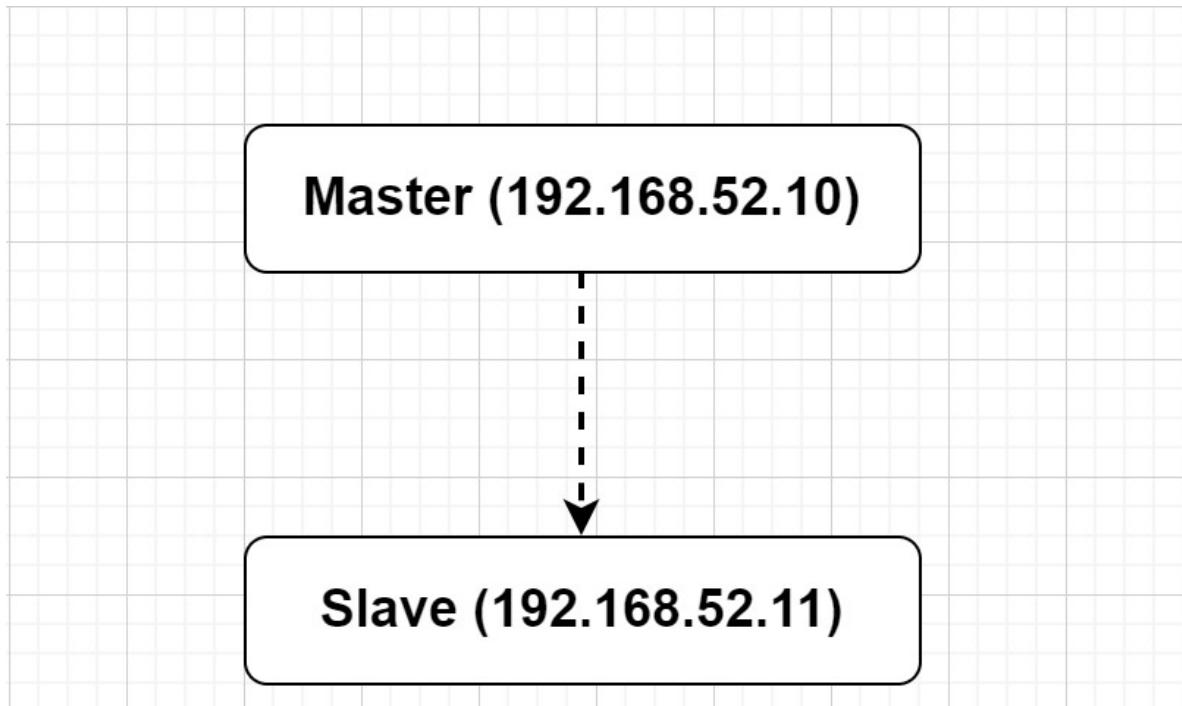
- Mysql 中有一种日志叫做 bin 日志（二进制日志）。这个日志会记录下所有修改了数据库的SQL语句（insert,update,delete,create/alter/drop table, grant 等等）。
- 主从复制的原理其实就是把主服务器上的 bin 日志复制到从服务器上执行一遍，这样从服务器上的数据就和主服务器上的数据相同了。



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

```
-- 创建数据库
CREATE DATABASE test CHARACTER SET utf8;

-- 创建表
CREATE TABLE users (
  id INT(11) PRIMARY KEY AUTO_INCREMENT,
  NAME VARCHAR(20) DEFAULT NULL,
  age INT(11) DEFAULT NULL
);

-- 插入数据
INSERT INTO users VALUES(NULL, 'user1', 20);
INSERT INTO users VALUES(NULL, 'user2', 21);
INSERT INTO users VALUES(NULL, 'user3', 22);
```

#### 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
```

- server-id=1 中的1可以任定义，只要是唯一的就行。
- log-bin=mysql-bin 表示启用binlog功能，并制定二进制日志的存储目录，
- binlog-do-db=test 是表示只备份test 数据库。
- binlog\_ignore\_db=mysql 表示忽略备份mysql。
- 不加binlog-do-db和binlog\_ignore\_db，那就表示备份全部数据库。

### 3) 第三步 重启MySQL

```
service mysqld restart
```

### 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

```
-- 创建账号  
GRANT REPLICATION SLAVE ON *.* TO 'slave'@'192.168.52.11' 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

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

### 6) 导出数据库,恢复写操作

使用SQLYog导出,主数据库备份完毕，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

- 增加server-id参数,保证唯一.

```
server-id=2
```

```
-- 重启  
service mysqld restart
```

## 10) 在从数据库设置相关信息

- 执行以下SQL

```
STOP SLAVE;

CHANGE MASTER TO MASTER_HOST='192.168.52.10',
MASTER_USER='slave',
MASTER_PASSWORD='Qwer@1234',
MASTER_PORT=3306,
MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=0,
MASTER_CONNECT_RETRY=10;
```

## 11) 修改auto.cnf中的UUID,保证唯一

```
-- 编辑auto.cnf
vim /var/lib/mysql/auto.cnf

-- 修改UUID的值
server-uuid=a402ac7f-c392-11ea-ad18-000c2980a208

-- 重启
service mysqld restart
```

## 12) 在从服务器上,启动slave 进程

```
start slave;

-- 查看状态
SHOW SLAVE STATUS;

-- 命令行下查看状态 执行
SHOW SLAVE STATUS \G;
```

Slave_IO_Running	Slave_SQL_Running
Yes	Yes

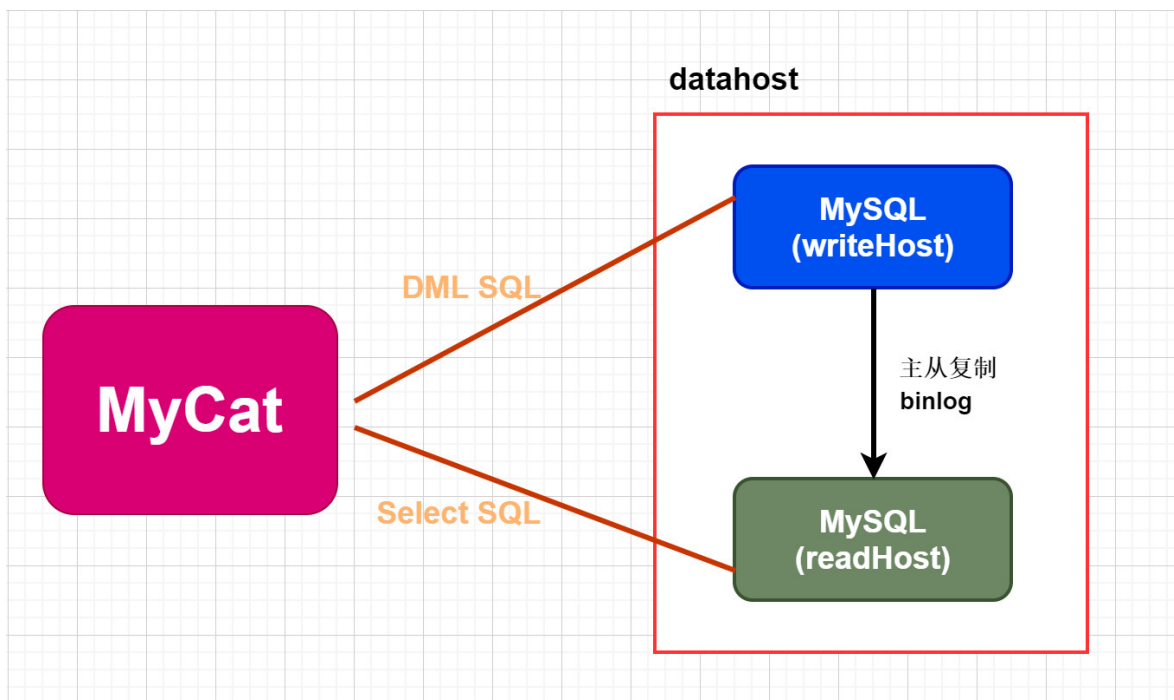
注意: 这两个参数的值,必须是 Yes, 否则就要进行错误的排查.

## 13) 现在可以在我们的主服务器做一些更新的操作,然后在从服务器查看是否已经更新

```
-- 在主库插入一条数据,观察从库是否同步
INSERT INTO users VALUES(NULL,'user4',23);
```

## 2.7.4 实现读写分离

数据库读写分离对于大型系统或者访问量很高的互联网应用来说，是必不可少的一个重要功能。对于MySQL来说，标准的读写分离是主从模式，一个写节点Master后面跟着多个读节点，读节点的数量取决于系统的压力，通常是1-3个读节点的配置



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表为 test

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

- **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"

```
<schema name="test" checkSQLSchema="true" sqlMaxLimit="100">
  <table name="users" dataNode="dn4" ruleRequired="false" primaryKey="id"
    autoIncrement="true" >
  </table>
</schema>
```

- **dataNode**

```
<!-- 读写分离 -->
<dataNode name="dn4" dataHost="localhost3" database="test" />
```

- dataHost

```
<!-- 读写分离 -->
<dataHost name="localhost3" maxCon="1000" minCon="10" balance="1" writeType="0"
dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
  <heartbeat>select user()</heartbeat>
  <!-- 主 -->
  <writeHost host="M1" url="192.168.52.10:3306" user="root"
password="QiDian@666">
  <!-- 从 -->
  <readHost host="S1" url="192.168.52.11:3306" user="root"
password="QiDian@666"
weight="1" />
</writeHost>
</dataHost>
```

balance参数:

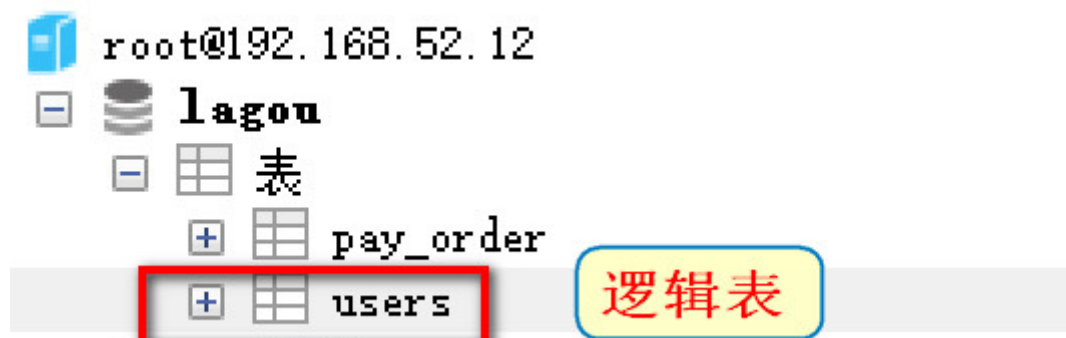
- 0 : 所有读操作都发送到当前可用的writeHost
- 1 : 所有读操作都随机发送到readHost和stand by writeHost
- 2 : 所有读操作都随机发送到writeHost和readHost
- 3 : 所有读操作都随机发送到writeHost对应的readHost上, 但是writeHost不负担读压力

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

- 重启MyCat

```
./mycat restart
```



- 执行查询和插入操作

1) 插入一条数据, 观察否两个表都同时新增了, 如果同时新增, 证明插入的是主库的表.

```
INSERT INTO users(NAME,age) VALUES('测试abc',26);
```

2) 在从库插入一条数据, 然后进行查询, 查询的是从库中的数据,证明查询操作在从库进行.

```
SELECT * FROM users;
```