# 10.mycat概述

## 10.1.mycat概述

### 10.1.1.介绍

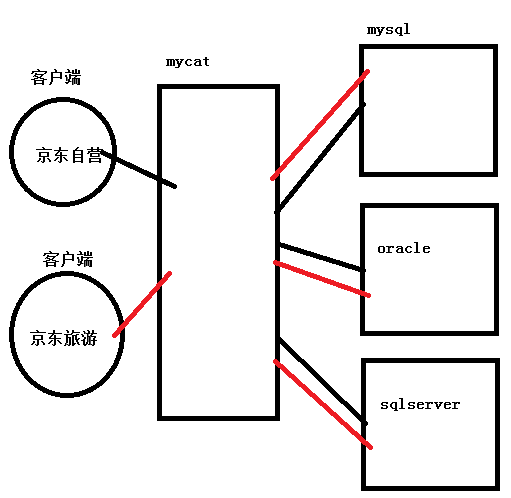
mycat号称中国第一开源数据库中间件，京东，双11都在使用

### 10.1.2.特点

支持高性能的读写分离

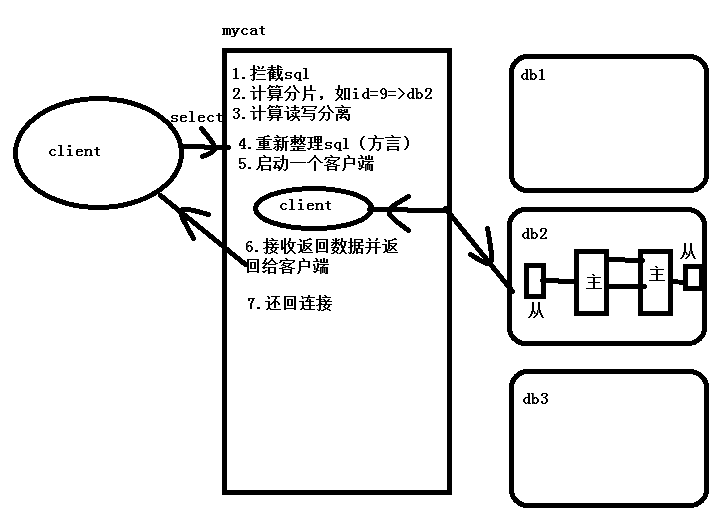
最大支持百亿（官方宣称千亿）级别的大表分片并行计算

支持数据整合输入输出



### 10.1.3.mycat原理

客户端发起sql，mycat进行拦截，之后开启后端数据库的连接，重新组织sql（方言）。



mycat执行逻辑：

1.拦截sql

2.计算分片，获取分片位置

3.计算读写分离逻辑

4.重新整理sql

5.启动客户端，并获取一条连接（连接池）

6.执行sql，并获取返回数据

7.将数据返回到客户端

8.还回资源

### 10.1.4.同类产品比较

**同类产品**

amoeba：变形虫

cobar：mycat前身

mycat：11年左右出现的最新中间件

**amoeba**

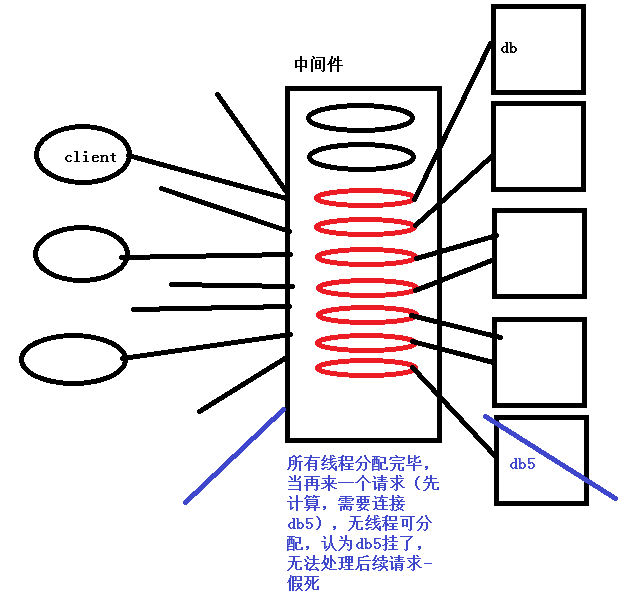
早期常用的数据库中间件，支持读写分离，支持跨库查询，但是一旦跨库，性能会受到极大影响。

停更了，团队几尽解散

**cobar**

非常受欢迎的数据库中间件，在mycat之前是常用的中间件，开发团队很多来自amoeba。

cobar早期版本（官方说已经解决）存在致命缺陷-后端数据库假死



由于高并发环境，由于后端线程是阻塞的，极易出现线程都被占用，没有闲置线程，当再来请求时，导致中间件无法连接数据库，认为数据库宕机--假死

## 10.2.mycat安装启动

1.环境要求

安装mycat,需要jdk支持，1.5版本的mycat需要jdk1.8支持,并且需要mysql的安装环境支持(mycat的命令,登录mycat就是使用mysql登录的);

2.安装启动mycat

2.1安装mycat(解压)

* 拷贝

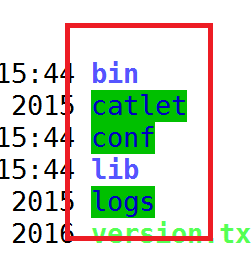
/home/resources/Mycat\*\*\*\*.tar.gz

[root@10-9-48-69 software]# cp /home/resources/Mycat-server-1.5.1-RELEASE-20161130213509-linux.tar.gz ./

* 解压

[root@10-9-48-69 software]# tar -xf Mycat-server-1.5.1-RELEASE-20161130213509-linux.tar.gz

* 目录结构



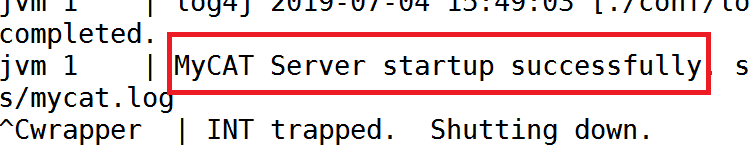
* bin mycat的命令脚本文件都在bin
* catlet 插件资源(默认是空)
* conf (配置文件所在路径,server.xml,schema.xml rule.xml等都是我们需要使用)
* logs:
  + wrapper.log:启动日志
  + mycat.log:运行日志
* mycat的启动

默认搭建安装的mycat支持启动测试,能够启动但是无法访问.

在bin下执行

#mycat start /后台启动,可以查看wrapper.log启动日志

#mycat console /控制台输出日志启动,wrapper.log依然会记录日志内容



## 10.3.macat配置文件与核心概念

修改之前，将配置文件复制一份

server.xml、schema.xml、rule.xml

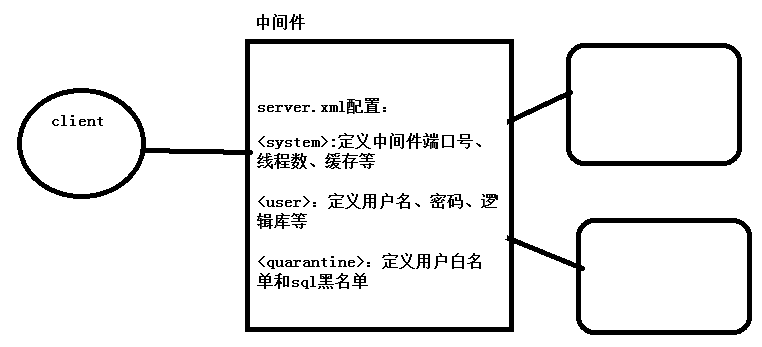
[root@localhost conf]# cp server.xml server.xml.bck

[root@localhost conf]# cp schema.xml schema.xml.bck

[root@localhost conf]# cp rule.xml rule.xml.bck

### 10.3.1.配置文件server.xml

**（1）标签与作用**



**（2）属性及作用**

system标签属性

defaultSqlParser：默认使用阿里德鲁伊的sql解析器，也可以自定义其他配置

serverPort：默认8066，不用改

user标签属性

name：登录mycat的用户名

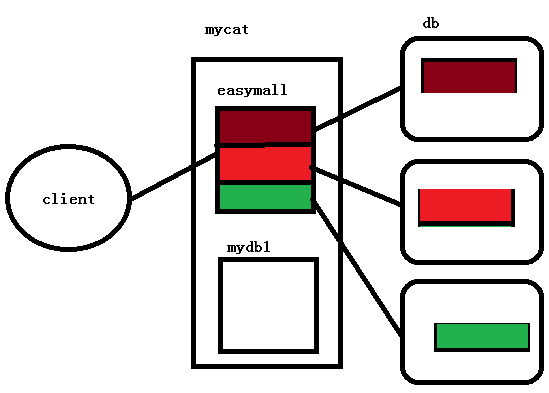
property

password：登录的密码

schemas：逻辑库，可以配多个，以“,”隔开

逻辑库：

不是真正的数据库，但是对于用户，使用逻辑库和真实库没有区别，逻辑库的使用对用户来说是透明的



quarantine标签属性

whitehost：用户白名单，如果在这里定义了此标签，只有定义的这些用户可以使用mycat，别人不能使用。

blacklist：sql黑名单，可以在这里定义一些sql，使之无法使用

<blacklist check="true">

<property name="selelctAllow">false</property>

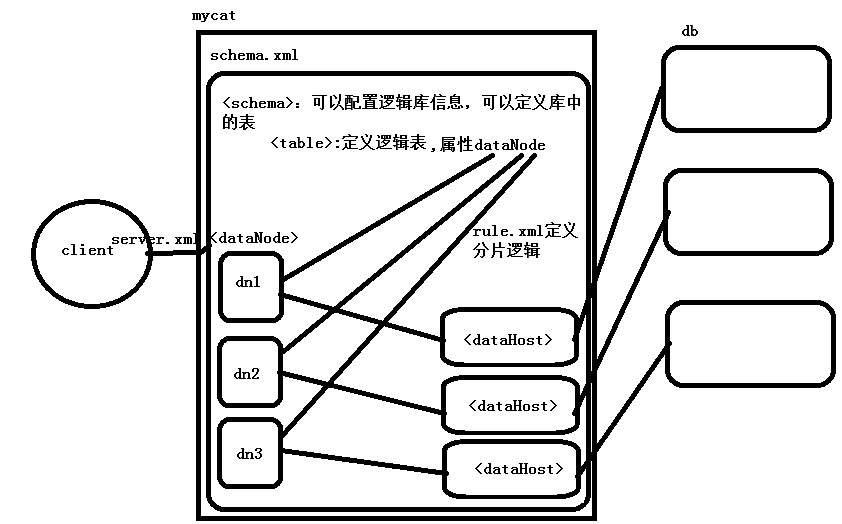
</blacklist>

**（3）server.xml配置案例**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mycat:server SYSTEM "server.dtd">  <mycat:server xmlns:mycat="http://org.opencloudb/">  <system>  <property name="defaultSqlParser">druidparser</property>  </system>  <user name="root">  <property name="password">root</property>  <property name="schemas">mstest</property>  </user>  </mycat:server> |

### 10.3.2.配置文件schema.xml

**（1）标签结构**



**（2）标签介绍**

schema标签：定义逻辑库，一个schema定义一个逻辑库

-table标签：定义逻辑表，一个schema中可以有多个table

-childTable标签：和ER分片表有关（后面讲）

dataNode标签：定义分片，table与dataNode绑定，dataNode与dataHost标签绑定

dataHost标签：定义某个分片绑定的真实数据库信息

-heartbeat标签：用来进行心跳检测的

-writeHost标签：定义一个分片用来负责写操作的节点（主节点）

--readHost标签：writeHost的子标签，是当前writeHost对应节点的从节点，用来读

**（3）定义一个逻辑库**

Schema标签定义

name属性：定义逻辑库的名称，给客户端看的

checkSQLschema属性：默认false，不检测，检测逻辑库名与真实库名是否一致

例如：select \* from user

如果是false（逻辑库名与真实库不一致能查到），则直接使用此sql

如果是true（必须逻辑库名与真实库一致才能查到），则会select \* from mstest.user

sqlMaxLimit属性：默认100，最大查询数量，如果sql没有limit字段，这里自动拼接，保护机制，防止客户端错误使用select语句造成mycat的性能损耗。如果sql有limit字段，此属性无效。

-table标签

逻辑表，分为：非分片表、分片表、ER分片表、全局表

非分片表：表的数据量较小，没有必要分片，直接存入一个真实数据库即可

分片表：表的数据量庞大，无法在单独一个表中存储，可以利用分片规则，将数据存入不同的真实数据库中。

注意：用户看到的就是一个逻辑表

--name属性：逻辑表的名称，逻辑表的名称必须和真实表的名称一致

--primaryKey属性：逻辑表的主键字段，也是和真实表保持一致

--dataNode属性：定义一个表是分片还是非分片，如果只对应一个dataNode就是非分片，对应多个就是分片

--rule属性：表示当前分片表的分片规则，例如auto-sharding-long：1-500万放入第一个分片，500-1000万放入第二个分片，以此类推

dataNode标签：定义分片计算内容

-name属性：分片名称，用来计算分片的关键字，table的dataNode属性和这个名字绑定

-dataHost属性：用来绑定dataHost标签

-database属性：分片数据对应的真实库名称

dataHost标签：定义一个分片对应的真实数据库

-name属性：绑定dataNode标签的

-maxCon属性：数据库连接池的最大连接

-minCon属性：数据库连接池的最小连接

-dbType属性：数据库类型：mysql、oracle

-dbDriver属性：连接数据库的Driver，如果是mysql，可以使用native，如果是oracle，只能写driver的全路径名

-heartbeat标签：心跳检测标签

-writeHost标签：定义写的数据库的真实节点信息

--host属性：名称

--url属性：真实数据库的ip:port

--user属性：用户名

--password属性：密码

--readHost标签：writeHost的子标签，定义读节点信息

---host属性：名称

---url属性：真实数据库的ip:port

---user属性：用户名

---password属性：密码

## 10.4.mycat配置

### 10.4.1.非分片表配置

**（1）准备数据库真实表**

两台数据库主从结构，准备真实库和真实表：mstest.t\_student(id int,name varchar)

**（2）配置server.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mycat:server SYSTEM "server.dtd">  <mycat:server xmlns:mycat="http://org.opencloudb/">  <system>  <property name="defaultSqlParser">druidparser</property>  </system>  <user name="root">  <property name="password">root</property>  <property name="schemas">mstest</property>  </user>  </mycat:server> |

**（3）配置schema.xml**

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE mycat:schema SYSTEM "schema.dtd">  <mycat:schema xmlns:mycat="http://org.opencloudb/" >  <schema name="mstest" checkSQLschema="false" sqlMaxLimit="100">  <table name="t\_student" dataNode="dn1" primaryKey="id"/>  </schema>  <dataNode name="dn1" dataHost="localhost1" database="mstest" />  <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="192.168.171.138:3306" user="root" password="root">  </writeHost>  </dataHost>  </mycat:schema> |

启动测试

在bin中执行：mycat console

可以在mycat中插入数据、查询数据等

### 10.4.2.配置单分片高可用

**（1）故障转移配置逻辑**

dataHost的switchType属性默认是1，表示开启故障转移，-1表示不开启

故障转移逻辑：

dataHost中可以配置多个writeHost。默认按照顺便编号，比如第一个writeHost编号是0，第二个是1。

默认使用0号writeHost，一旦心跳检测发现0号挂了，就顺序使用下一个编号的writeHost，以此类推，如果0号再启动，则编号顺序向下

**（2）配置高可用的schema.xml**

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE mycat:schema SYSTEM "schema.dtd">  <mycat:schema xmlns:mycat="http://org.opencloudb/" >  <schema name="mstest" checkSQLschema="false" sqlMaxLimit="100">  <table name="t\_student" dataNode="dn1" primaryKey="id"/>  </schema>  <dataNode name="dn1" dataHost="localhost1" database="mstest" />  <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="192.168.171.138:3306" user="root" password="root">  </writeHost>  <writeHost host="hostM2" url="192.168.171.142:3306" user="root" password="root">  </writeHost>  </dataHost>  </mycat:schema> |

**（3）验证高可用**

1.将第一个writeHost 干掉

mycat读写依然正常

2.将第二个writeHost 干掉（前提是启动第一个）

mycat读写依然正常

### 10.4.3.读写分离测试

**（1）准备工作**

1.将两个主从关系干掉

stop slave

2.连个数据库构造不同数据用来区分

**（2）属性配置**

writeType属性：默认是0，表示所有的写操作都在第一个（index=0）writeHost执行

1：老版本遗留（1.5以上的mycat不推荐配置），对所有的writeHost随机读写

balance属性：

0：不启动读写分离，所有操作都在第一个writeHost执行

1：开启读写分离：绝大部分读在除了第一个writeHost之外的其他节点读取（writeHost、readHost），当并发量超过一定标准时，第一个writeHost也分担读的压力。

2：所有的读操作，都随机在所有节点执行（包括所有writeHost和readHost）

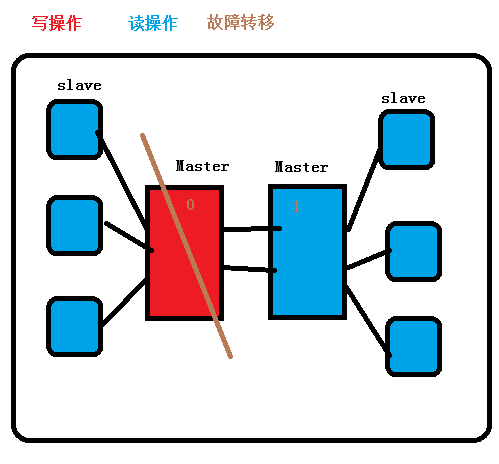
3：只在readHost中完成读操作，如果没有配置readHost，读的操作会到第一个writeHost完成。

配置一个readHost标签

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE mycat:schema SYSTEM "schema.dtd">  <mycat:schema xmlns:mycat="http://org.opencloudb/" >  <schema name="mstest" checkSQLschema="false" sqlMaxLimit="100">  <table name="t\_student" dataNode="dn1" primaryKey="id"/>  </schema>  <dataNode name="dn1" dataHost="localhost1" database="mstest" />  <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="192.168.171.138:3306" user="root" password="root">  <readHost host="hostMs1" url="192.168.171.142:3306" user="root" password="root"></readHost>  </writeHost>  </dataHost>  </mycat:schema> |

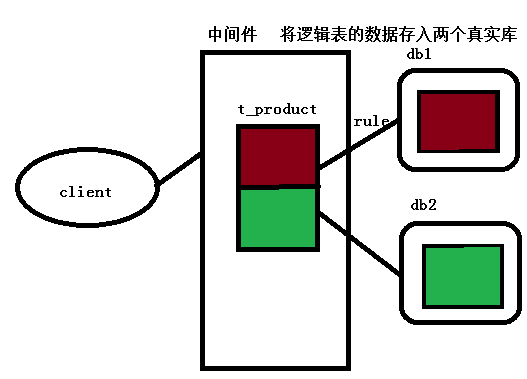
### 10.4.4.一个合理的dataHost配置

writeType：0 balance：1 switchType：1



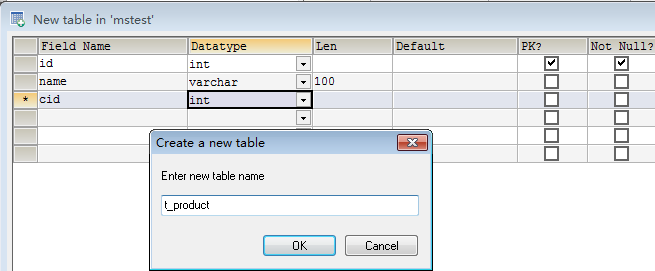
### 10.4.5.多分片配置

**（1）分片结构**



**（2）数据库准备**

两个数据库都准备表：mstest.t\_product: id int,name varchar,cid int



**（3）配置schema.xml**

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE mycat:schema SYSTEM "schema.dtd">  <mycat:schema xmlns:mycat="http://org.opencloudb/" >  <schema name="mstest" checkSQLschema="false" sqlMaxLimit="100">  <table name="t\_student" dataNode="dn1" primaryKey="id"/>  <table name="t\_product" dataNode="dn1,dn2" primaryKey="id" rule="auto-sharding-long"/>  </schema>  <dataNode name="dn1" dataHost="localhost1" database="mstest" />  <dataNode name="dn2" dataHost="localhost2" database="mstest" />  <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="192.168.171.138:3306" user="root" password="root">  </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="hostM1" url="192.168.171.142:3306" user="root" password="root">  </writeHost>  </dataHost>  </mycat:schema> |

**（4）测试**

新增三条数据

INSERT INTO t\_product(id,NAME,cid) VALUES(1,'打印机',1)

INSERT INTO t\_product(id,NAME,cid) VALUES(5000001,'phone',2)

INSERT INTO t\_product(id,NAME,cid) VALUES(10000001,'apple',3)

发现第一条进入第一个分片，第二条进入第二个分片

但是第三条报错

**因为auto-sharding-long分片规则为：**

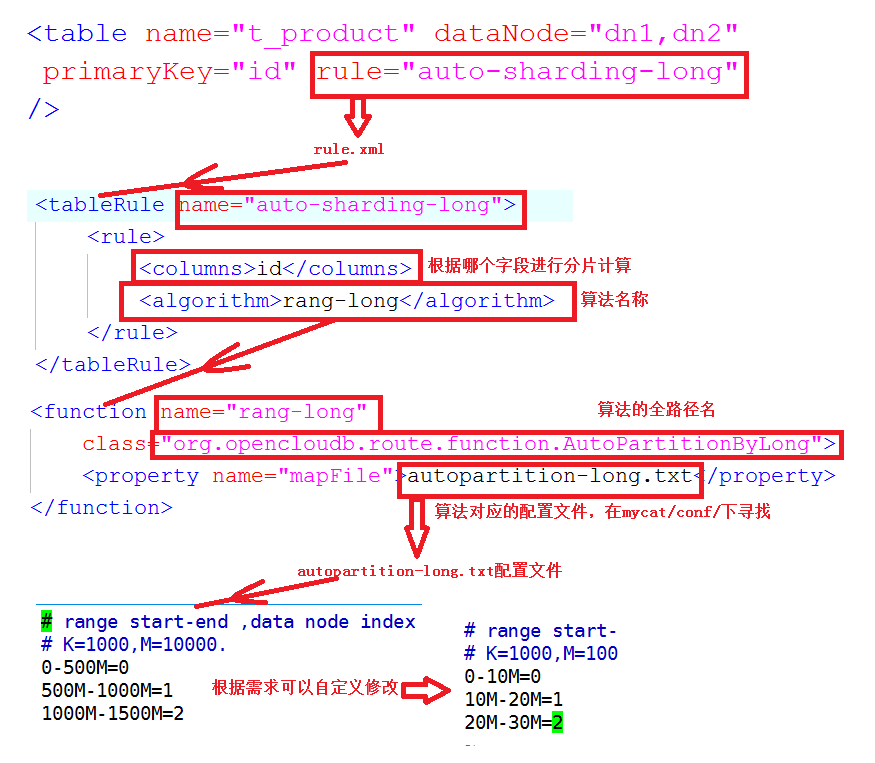
0-500万进入第一个分片

500-1000万进入第二个分片

以此类推

由于之前配置只有两个分片，所以如果id>1000万，则无法找到第三个分片，就会报错

### 10.4.6.分片规则rule.xml



### 10.4.7.hash一致性分片

如果计算分片的是其他类型，如字符串、时间等，可以使用hash一致性

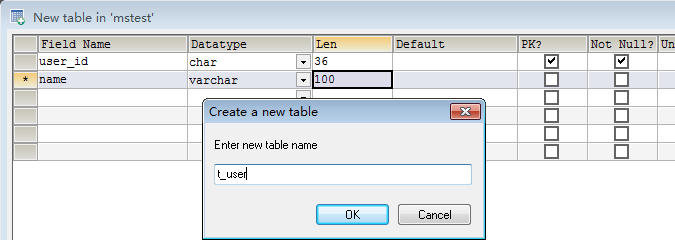
**（1）配置hash一致性**

修改schema.xml的rule属性

<table name="t\_product" dataNode="dn1,dn2" primaryKey="id" rule="sharding-by-murmur"/>

**（2）自定义hash一致性分片主键**

1.两个数据库创建t\_user表：mstest.t\_user user\_id char(36)主键,name varchar(100)



2.schema.xml修改

<table name="t\_user" dataNode="dn1,dn2" primaryKey="user\_id" rule="user-murmur"/>

3.rule.xml修改

增加一个tableRule 标签

<tableRule name="user-murmur">

<rule>

<columns>user\_id</columns>

<algorithm>murmur</algorithm>

</rule>

</tableRule>

4.murmur的function解释（无需改动）

<function name="murmur"

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

<property name="seed">0</property><!-- 默认是0，hash底层计算hash桶的种子值，和底层计算有关，不用动 -->

<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，和数据库数量保持一致 -->

<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，即使配置，可能也找不到 -->

</function>

**（3）测试**

插入4条数据

INSERT INTO t\_user(user\_id,NAME)

VALUES('153b198e-228d-435b-91ab-58e6498c00f9','zhangsan')

INSERT INTO t\_user(user\_id,NAME)

VALUES('3bca1c96-2a0f-4b0d-a684-106924823aa2','lisi')

INSERT INTO t\_user(user\_id,NAME)

VALUES('6f9378f3-c400-4a7a-9e96-5162bc25c842','wangwu')

INSERT INTO t\_user(user\_id,NAME)

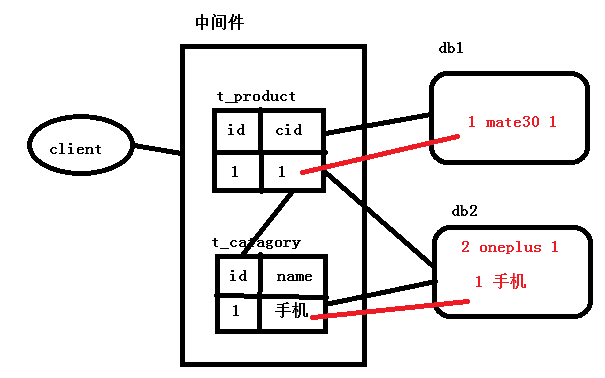
VALUES('da57776f-9fae-431d-9359-18492ae13858','laoliu')

发现部分在第一个分片，部分在第二个分片

## 10.5.全局表

### 10.5.1.全局表概念

mycat不支持分库关联查询（分库关联查询效率很低）



问题描述：商品表是分片表，会存入不同数据库，但是商品分类表数据量很少，只需要存入一个数据库即可，但是由于mycat不支持分库查询，所以，如果商品表和分类表的数据不在同一个库，那么将无法关联查询到完整数据

解决思路：由于分类表数据量特别少，可以给每个分片都保存一份，这样，在关联查询时，每个分片都可以拿到完整数据--全局表

**全局表概念：**

当某个表数据量不大，且比较稳定，本来可以作为非分片表存储，但是经常需要和其他分片表关联查询，而由于mycat不支持分库查询，如果此表和其他表不在同一个分片，将无法查询到完整数据，就可以在每个分片中都保存这个表，这样的表就称为全局表。

常见的全局表：工具字典表（http中404=找不到路径，500=后台代码错误，12352=分拨器）

### 10.5.2.配置全局表

在配置table表时，dataNode对应多个分片，直接删除rule属性

1.数据库准备t\_category（id、name）

2.修改schema.xml

<table name="t\_category" dataNode="dn1,dn2" primaryKey="id" type="global"/>

type="global"为默认配置，可以不加

## 10.6.ER分片表

### 10.6.1.ER分片表概念

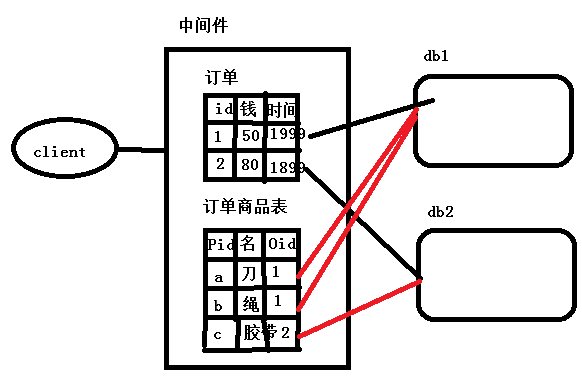
问题描述：

例如有订单表和订单商品表，两个表都需要分片

如果两个表的关联查询数据分到不同分片，则mycat关联查询时，无法拿到完整数据（不支持分库查）

解决思路：将关联的商品和订单数据放入同一个分片，如何保证？

可以利用外键字段（两个表的外键和主键字段内容相同）进行hash一致性计算，计算结果一致。



**ER分片表逻辑：**

er分片表是mycat首创

先定义父级表格（table），再定义子级表格（childTable），所有表格的数据切分由父级表格维护

### 10.6.ER分片表案例

1.2个数据库准备order表和order\_item表

CREATE TABLE `t\_order` (

`id` CHAR(36) NOT NULL,

`money` INT(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=INNODB DEFAULT CHARSET=latin1

CREATE TABLE `order\_item` (

`id` CHAR(36) NOT NULL,

`name` VARCHAR(100) DEFAULT NULL,

`oid` CHAR(36) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=INNODB DEFAULT CHARSET=latin1

2.mycat中配置schema.xml

<table name="t\_order" dataNode="dn1,dn2" primaryKey="id" rule="sharding-by-murmur">

<childTable name="order\_item" joinKey="oid" primaryKey="id" parentKey="id">

</childTable>

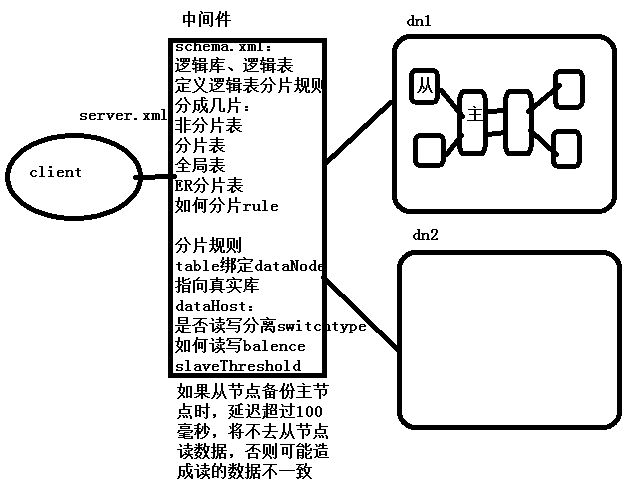
</table>

3.测试

给父表中插入一些数据，再向子表中插入一些数据，查看相关数据是否在同一个分片中

经测试，没有问题

## 10.7.mycat总结图



## 10.8.easymall订单系统

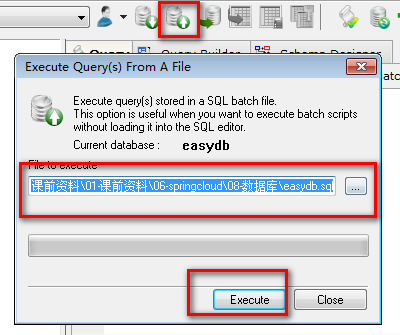
### 10.8.1.数据库表

订单业务并发量极高，需要mycat中间件的支持

**（1）数据库导入表格**

课前资料\01-课前资料\06-springcloud\08-数据库\easydb.sql

将两个数据库都导入以上数据



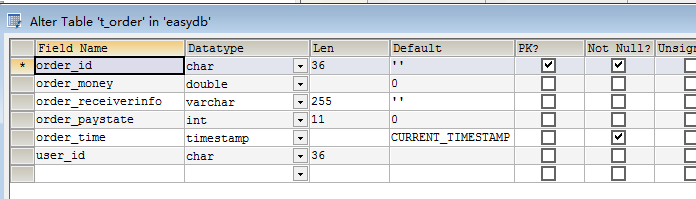
两个库清理数据：

DELETE FROM t\_order

DELETE FROM t\_order\_item

**（2）相关表格**

t\_order表：



order\_id：主键UUID

order\_money：订单金额

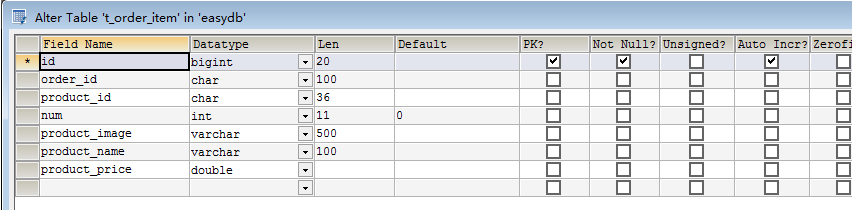
order\_receiverinfo：收货人信息

order\_paystate：支付状态

order\_time：订单生成时间

user\_id：用户id

t\_order\_item表：



id：主键自增

order\_id：订单id

product\_id：商品id

num：购买数量

product\_image、product\_name、product\_price冗余字段（反范式设计）

**（3）mycat配置**

server.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mycat:server SYSTEM "server.dtd">  <mycat:server xmlns:mycat="http://org.opencloudb/">  <system>  <property name="defaultSqlParser">druidparser</property>  </system>  <user name="root">  <property name="password">root</property>  <property name="schemas">easydb</property>  </user>  </mycat:server> |

schema.xml

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE mycat:schema SYSTEM "schema.dtd">  <mycat:schema xmlns:mycat="http://org.opencloudb/" >  <schema name="easydb" checkSQLschema="false" sqlMaxLimit="100">  <table name="t\_order" dataNode="dn3,dn4" primaryKey="order\_id" rule="order-by-murmur">  <childTable name="t\_order\_item" joinKey="order\_id" primaryKey="id" parentKey="order\_id">  </childTable>  </table>  </schema>  <dataNode name="dn3" dataHost="localhost1" database="easydb" />  <dataNode name="dn4" dataHost="localhost2" database="easydb" />  <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"  writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">  <heartbeat>select user()</heartbeat>  <writeHost host="hostM1" url="192.168.171.138:3306" user="root" password="root">  </writeHost>  </dataHost>  <dataHost name="localhost2" maxCon="1000" minCon="10" balance="1"  writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">  <heartbeat>select user()</heartbeat>  <writeHost host="hostM1" url="192.168.171.142:3306" user="root" password="root">  </writeHost>  </dataHost>  </mycat:schema> |

rule.xml

|  |
| --- |
| <tableRule name="order-by-murmur">  <rule>  <columns>order\_id</columns>  <algorithm>murmur</algorithm>  </rule>  </tableRule> |

**（4）mycat启动测试**

不报错、能通过mycat查看到真实库的数据即可

### 10.8.2.系统搭建

1.quickstart（easymall-microservice-order）

2.pom文件继承easymall-parent，依赖repository、resources

3.配置文件

|  |
| --- |
| server.port=10005  spring.datasource.driverClassName=com.mysql.jdbc.Driver  spring.datasource.url=jdbc:mysql://192.168.171.138:8066/easydb  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  spring.datasource.initialSize=5  spring.datasource.maxActive=50  spring.datasource.maxIdle=10  spring.datasource.minIdle=5  mybatis.typeAliasesPackage=com.jt.common.pojo  mybatis.mapperLocations=classpath:mapper/\*.xml  mybatis.configuration.mapUnderscoreToCamelCase=true  mybatis.configuration.cacheEnables=false  spring.application.name=orderservice  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka |

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  @MapperScan("cn.tedu.order.mapper")  **public** **class** StarterOrderCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterOrderCenter.**class**, args);  }  } |

### 10.8.3.订单新增业务

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/order/save |
| 后台接收 | /order/manage/save |
| 请求方式 | Post |
| 请求参数 | Order order,其中缺少order的id值 |
| 返回数据 | 返回SysResult对象的json,其结构:  Integer status; 200表示成功,其他表示失败  String msg;成功返回 “ok”,失败返回其他信息  Object data;根据需求携带其他数据 |

OrderController

|  |
| --- |
| **package** cn.tedu.order.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.jt.common.pojo.Order;  **import** com.jt.common.vo.SysResult;  @RestController  @RequestMapping("/order/manage")  **public** **class** OrderController {  @Autowired  **private** OrderService orderService;  @RequestMapping("/save")  **public** SysResult saveOrder(Order order){  **try** {  orderService.saveOrder(order);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  **return** SysResult.*build*(202, e.getMessage(), **null**);  }  }  } |

OrderService

|  |
| --- |
| **package** cn.tedu.order.service;  **import** java.util.Date;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.jt.common.pojo.Order;  **import** com.jt.common.utils.UUIDUtil;  @Service  **public** **class** OrderService {  @Autowired  **private** OrderMapper orderMapper;  **public** **void** saveOrder(Order order) {  //补充数据:  //orderId\orderTime创建时间\orderPaystate支付状态，0表示未支付  order.setOrderId(UUIDUtil.*getUUID*());  order.setOrderTime(**new** Date());  order.setOrderPaystate(0);  orderMapper.saveOrder(order);  }  } |

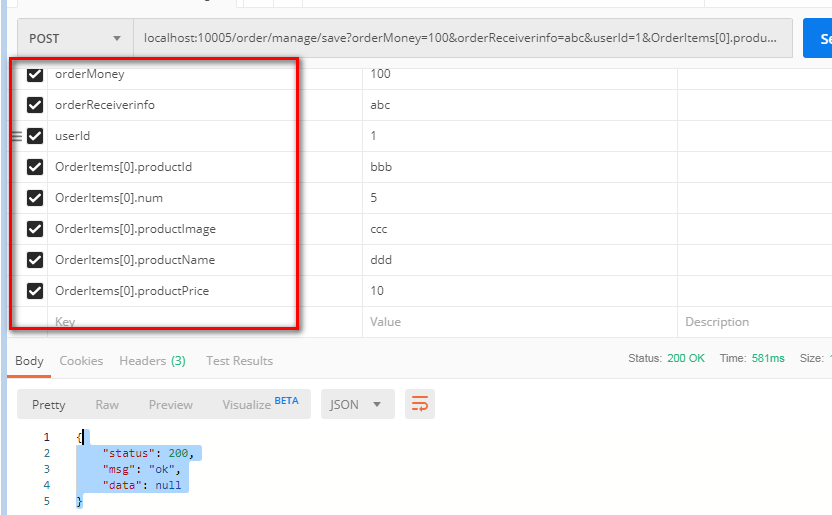
OrderMapper

|  |
| --- |
| **package** cn.tedu.order.mapper;  **import** com.jt.common.pojo.Order;  **public** **interface** OrderMapper {  **void** saveOrder(Order order);  } |

OrderMapper.xml

|  |
| --- |
| <insert id=*"saveOrder"* parameterType=*"Order"*>  <!-- 插入t\_order表 -->  insert into t\_order(  order\_id,order\_money,order\_receiverinfo,  order\_paystate,order\_time,user\_id) values(  #{orderId},  #{orderMoney},  #{orderReceiverinfo},  #{orderPaystate},  #{orderTime},  #{userId});  <!-- 插入t\_order\_item表 -->  <foreach collection=*"orderItems"* item=*"item"*>  insert into t\_order\_item(  order\_id,product\_id,num,  product\_image,product\_name,product\_price) values(  #{orderId},  #{item.productId},  #{item.num},  #{item.productImage},  #{item.productName},  #{item.productPrice});  </foreach>  </insert> |

**单独测试：利用postman测试**



### 10.8.4.页面整合

**（1）nginx整合**

location /order {

proxy\_pass http://127.0.0.1:9005/zuul-order/order/manage;

add\_header 'Access-Control-Allow-Credentials' 'true';

add\_header 'Access-Control-Allow-Origin' '\*';

}

**（2）zuul整合**

zuul.routes.zuul-order.path=/zuul-order/\*\*

zuul.routes.zuul-order.serviceId=orderservice

**（3）启动测试**

启动nginx、redis、mycat、eureka、configserver、zuul、product、user、cart、order

### 10.8.5.订单查询业务

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/order/query/{userId} |
| 后台接收 | /order/manage/query/{userId} |
| 请求方式 | get |
| 请求参数 | 路径参数String userId |
| 返回数据 | List<Order>数据,根据长度判断查询是否成功,如果长度为0可能是未登录也可能是就是没有数据. |

由于一个用户对应多个订单

每个订单对应多个商品

可以在service中进行查询遍历，这样连接用的非常多（订单数\*每个订单的商品数）

将封装交给mysql

OrderController

|  |
| --- |
| @RequestMapping("/query/{userId}")  **public** List<Order> queryOrdery(@PathVariable String userId){  **return** orderService.queryOrdery(userId);  } |

OrderService

|  |
| --- |
| **public** List<Order> queryOrdery(String userId) {  **return** orderMapper.queryOrdery(userId);  } |

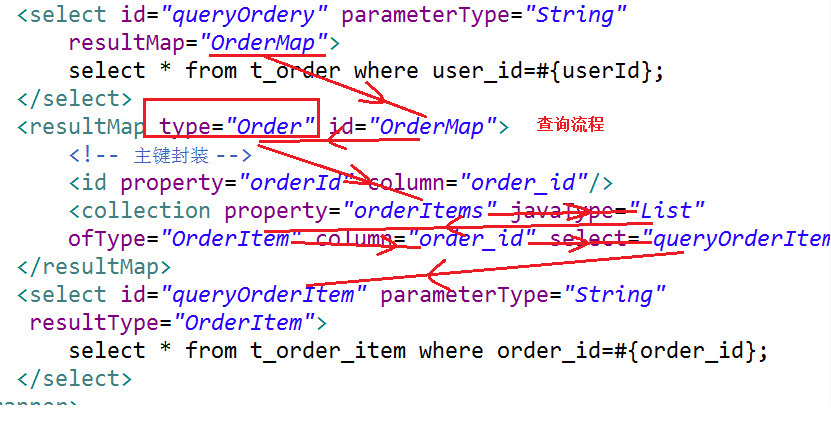
OrderMapper

|  |
| --- |
| List<Order> queryOrdery(String userId); |

OrderMapper.xml

|  |
| --- |
| <select id=*"queryOrdery"* parameterType=*"String"*  resultMap=*"OrderMap"*>  select \* from t\_order where user\_id=#{userId};  </select>  <resultMap type=*"Order"* id=*"OrderMap"*>  <!-- 主键封装 -->  <id property=*"orderId"* column=*"order\_id"*/>  <collection property=*"orderItems"* javaType=*"List"*  ofType=*"OrderItem"* column=*"order\_id"* select=*"queryOrderItem"*></collection>  </resultMap>  <select id=*"queryOrderItem"* parameterType=*"String"*  resultType=*"OrderItem"*>  select \* from t\_order\_item where order\_id=#{order\_id};  </select> |

查询流程



### 10.8.6.订单删除业务

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/order/delete/{orderId} |
| 后台接收 | /order/manage/delete/{orderId} |
| 请求方式 | Get |
| 请求参数 | 路径参数String orderId |
| 返回数据 | 返回SysResult对象的json,其结构:  Integer status; 200表示成功,其他表示失败  String msg;成功返回 “ok”,失败返回其他信息  Object data;根据需求携带其他数据 |

OrderController

|  |
| --- |
| @RequestMapping("/delete/{orderId}")  **public** SysResult deleteOrder(@PathVariable String orderId){  **try** {  orderService.deleteOrder(orderId);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  **return** SysResult.*build*(202, e.getMessage(), **null**);  }  } |

OrderService

|  |
| --- |
| **public** **void** deleteOrder(String orderId) {  orderMapper.deleteOrder(orderId);  } |

OrderMapper

|  |
| --- |
| **void** deleteOrder(String orderId); |

OrderMapper.xml

|  |
| --- |
| <delete id=*"deleteOrder"* parameterType=*"String"*>  delete from t\_order\_item where order\_id=#{orderId};  delete from t\_order where order\_id=#{orderId};  </delete> |