一、 实验目的

1、掌握 SpringBoot 应用读写 MyBatis 数据库的方法
2、掌握用 SpringMVC 实现 RESTful API 的方法
3、掌握用 JMeter 测试 RESTful API 应用的方法
4、验证通过 MyBatis 读写数据库的效率

二、实验环境

1、服务器 A: Ubuntu 18.04 服务器 2 核 4G 内存虚拟机一台，图形界面，安装 JDK 11， Maven、git
2、服务器 B: Ubuntu 18.04 服务器 2 核 2G 内存虚拟机一台，命令行界面，安装 JDK 11，Maven、git，JMeter 5.4.1
3、服务器 C: Ubuntu 18.04 服务器 2 核 2G 内存虚拟机一台，命令行界面，安装 JDK 11，Maven、git，MySQL 8.0

三、实验内容及要求

在基于数据库的应用中，数据库的访问是系统的主要瓶颈之一。设计一个实验对比使用 MyBatis 读写数据库的效率。要求实现以下两个 RESTful API:

API	API 描述链接

查询订单完整信息	GET /orders/{id}
	返回值:
	{
	"errno": 0,
	"errmsg": "成功",
	"data": {
	"id": 0,
	"orderSn": "string",
	"pid": 0,
	"orderType": 0,
	"state": 0,
	"subState": 0,
	"gmtCreate": "string",
	"gmtModified": "string",
	"confirmTime": "string",
	"originPrice": 0,
	"discountPrice": 0,
	"freightPrice": 0,
	"rebateNum": 0,
	"message": "string",
	"regionId": 0,

<table>
<tr><td></td><td>

```
    "address": "string",
    "mobile": "string",
    "consignee": "string",
    "couponId": 0,
    "grouponId": 0,
    "presaleId": 0,
    "shipmentSn": "string",
    "orderItems": [
      {
        "skuId": 0,
        "orderId": 0,
        "name": "string",
        "quantity": 0,
        "price": 0,
        "discount": 0,
        "couponActId": 0,
        "beSharedId": 0
      }
    ]
  }
}
```

</td></tr>
<tr><td>新建订单</td><td>POST /orders
参数:
Orderinfo:</td></tr>
<tr><td></td><td>

```
{
  "orderItems": [
    {
      "skuId": 0,
      "quantity": 0,
      "couponActId": 0
    }
  ],
  "consignee": "string",
  "regionId": 0,
  "address": "string",
  "mobile": "string",
  "message": "string",
  "couponId": 0,
  "presaleId": 0,
  "grouponId": 0
}
```

</td></tr>
</table>

四、具体设计

1. 总体工程结构:


Project
Exp_2 E:\lesson\JavaEE_Exp\Exp_2
.idea
.mvn
logs
src
main
java
com
example
exp_2
config
CorsConfig
LocalDateTimeSerializerConfig
Swagger2Configuration
controller
OrdersController
dao
OrdersDao
mapper
OrdersMapper
model
GoodsSKU
GoodsSKUPo
GoodsSKURetVo
GoodsSKUVo
OrderItem
OrderItemPo
OrderItemRetVo
OrderItemVo
Orders
OrdersPo
OrdersRetVo
OrdersVo
VoObject
service
OrdersService
util
Common
JacksonUtil
ResponseCode
ResponseUtil
ReturnObject
Exp2Application
resources
com
example
exp_2
mapper
OrdersMapper.xml
application.yaml
logback.xml
test
.gitignore
Exp_2.iml
pom.xml
External Libraries
Scratches and Consoles

数据库中总共提供了三章表：order、order_item、goods_sku，因此我们在 model 包中分别为这三章表建立对应的 po 对象，并且针对要求的 API 建立需要的 VO 和 RetVo。另外，我们采用代理设计模式，为每个 PO 类设计对应的 BO 类，以便把 PO 中一些不需要或者没有的属性在 BO 中实现。

为方便查看系统具体运行情况，也为了方便调试，我们配置了 logback，在控制台和 logs 文件夹下进行日志输出。

为方便工程最后的打包，我们将 OrdersMapper.xml 文件放在 resources 的同路径目录下，最后打包时会和生产代码打包在一起。

2. 设计思路

实验文档中给出两个需要实现的 API，一个是 GET 的查找订单，一个是 POST 的新建订单，很容易联想到要用 SQL 语句中的 SELECT 和 INSERT 去对应实现。

根据实验文档中的 E-R 图可以看出，orders 和 order_item 是一对多的关系，很明显不是单个的 SELECT 语句可以实现的，需要设计一对多的映射，因此需要在 OrdersPo 对象中建立对应的 OrderItemPoList，在 Mapper 中定义对应的 resultMap，同时要考虑两表的连接查询。

```
    private LocalDateTime gmtModified;

    private List<OrderItemPo> orderItemPoList;

}
```

```
<resultMap id="OrdersWithItemMap" type="OrdersPo" autoMapping="true">
    <id property="id" column="id"/>
    <collection property="orderItemPoList" ofType="com.example.exp_2.model.OrderItemPo">
        <id property="id" column="order_item_id"/>
        <result property="goodsSkuId" column="order_item_goods_sku_id"/>
        <result property="orderId" column="order_item_order_id"/>
        <result property="quantity" column="order_item_quantity"/>
        <result property="price" column="order_item_price"/>
        <result property="discount" column="order_item_discount"/>
        <result property="couponActivityId" column="order_item_coupon_activity_id"/>
        <result property="beShareId" column="order_item_be_share_id"/>
    </collection>
</resultMap>
```

```
    oi.be_share_id as order_item_be_share_id
    from orders os left join order_item oi
    on os.id = oi.order_id
    where
```

在定义 resultMap 时，如果映射的两方有相同名称的属性，需要慎重使用 autoMapping 属性。比如 order 表和 order_item 表中都有 gmt_create 和 gm_modified 属性，如果我们在 collection 标签中也定义了 autoMapping，就会把前端不需要的某些属性也筛选出来。因此，如果有重复名称的属性，最好手动去一一对应

第二个 API 相对简单，插入一个 order，并插入对应的 orderItem，不涉及两表连接，只需要在插入 orderItem 时注意设置对应的 order_id 即可

```java
public ReturnObject<Orders> createOrders(Orders orders){
    OrdersPo ordersPo = orders.getOrdersPo();
    String seqNum = genSeqNum();
    ordersPo.setOrderSn(seqNum);

    int ret = ordersMapper.createOrders(ordersPo);
    if(orders.getOrderItemList()!=null){
        for (OrderItem orderItem : orders.getOrderItemList()){
            OrderItemPo orderItemPo = orderItem.getOrderItemPo();
            orderItemPo.setOrderId(orders.getId());
            orderItemPo.setPrice(5);
            ret = ordersMapper.createOrderItem(orderItemPo);
        }
    }
    ReturnObject<Orders> returnObject = new ReturnObject<>(orders);
    return returnObject;
}
```

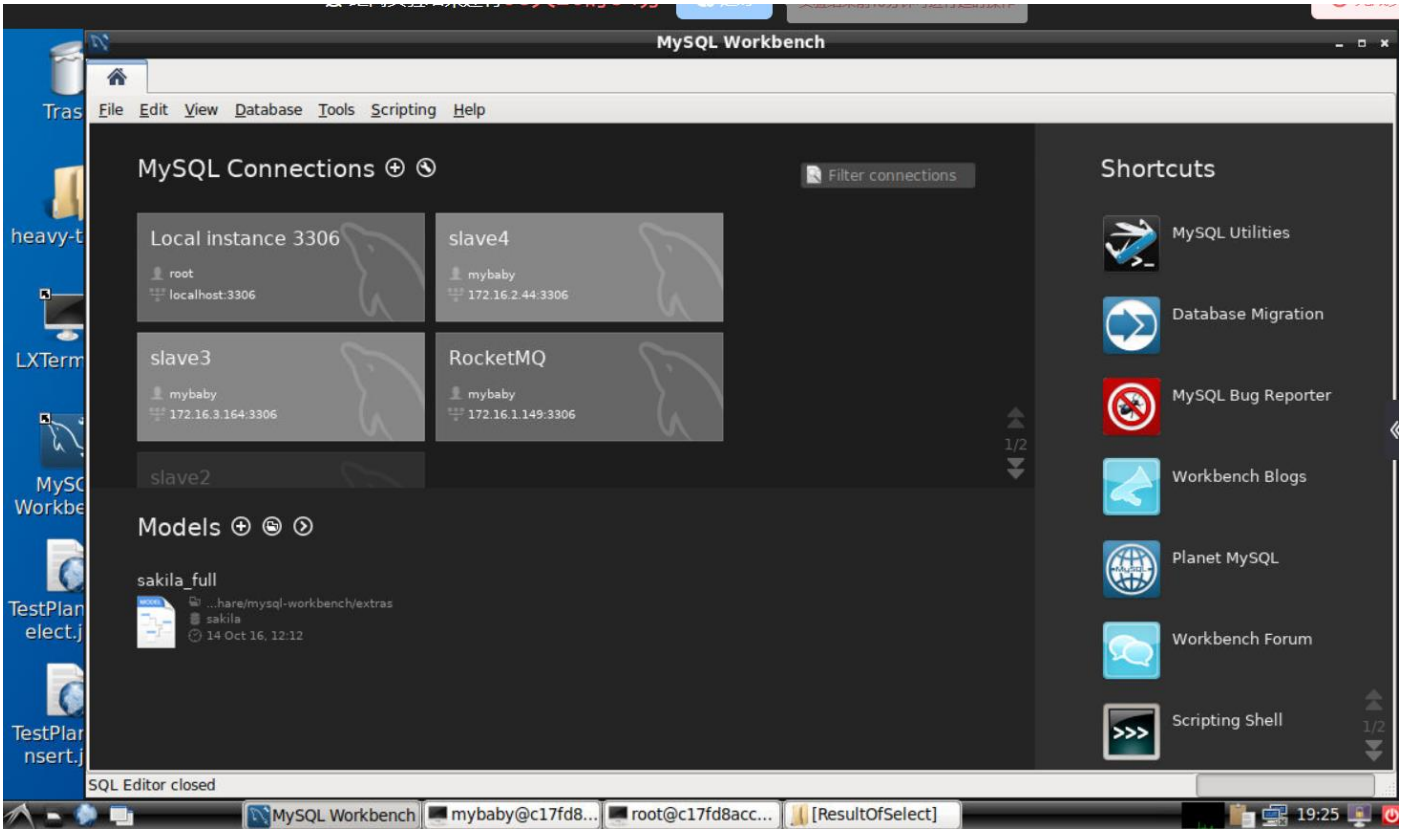
3. 部分配置信息

```yaml
datasource:
  driver-class-name: com.mysql.cj.jdbc.Driver
  url: jdbc:mysql://localhost:3306/databaseforexp2?serverTimezone=GMT%2B8
  username: mybaby
  password: 12345678
  type: com.alibaba.druid.pool.DruidDataSource
  initialization-mode: always
  druid:
    initial-size: 3
    min-idle: 3
    max-active: 300
    max-wait: 60000
    stat-view-servlet:
      login-username: admin
      login-password: 123456
      enabled: true
      url-pattern: /druid/*
```

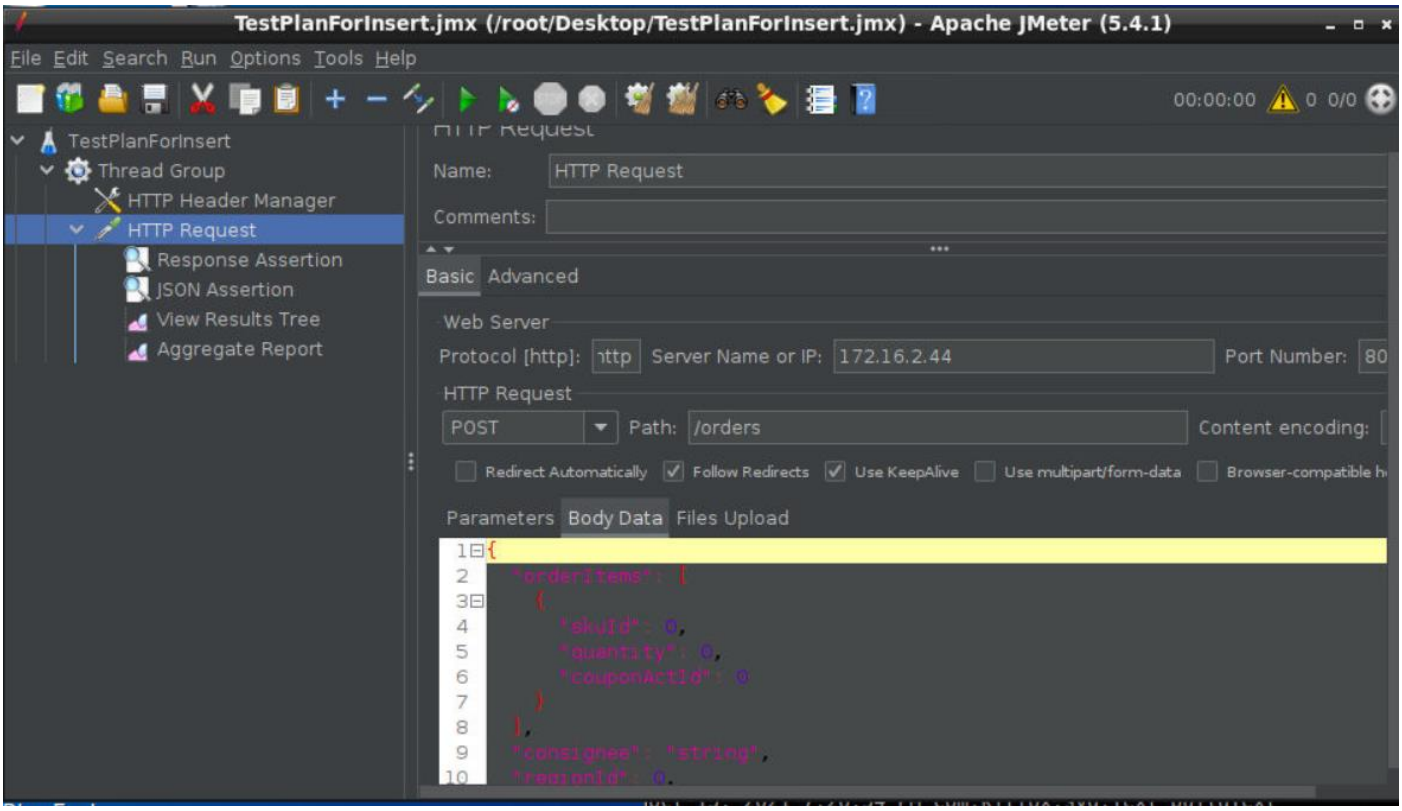
考虑到给出的 sql 文件只有建表和建数据，没有指定 database 名称，因此需要在建表之前先建立 schema。并且要保证在每台运行系统的服务器上都有对应名称的 database。同时，为了安全性考虑，不使用 root 用户，而是新建 mybaby 用户用于访问数据库

考虑到简单的 insert 和 select 不需要大量线程去对比差异，我们在制定测试计划时也不需要过多的线程数，200 左右即可，因此我们设置 Druid 连接池的最大数量为 300，以适应线程数

4. 具体测试

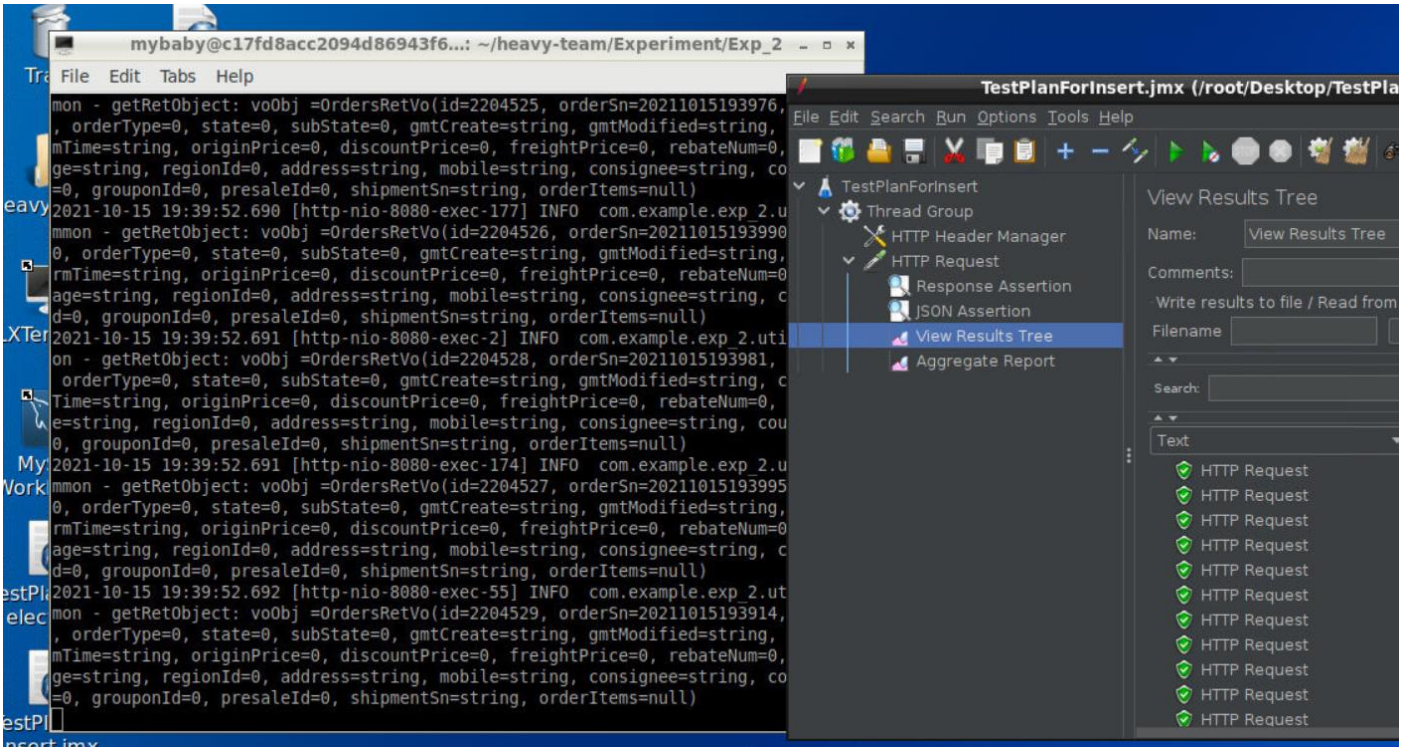


为图形化界面服务器加装 MySQL WorkBench，同时远程连接其他四台服务器，方便运行 sql 脚本、观察无界面服务器数据库

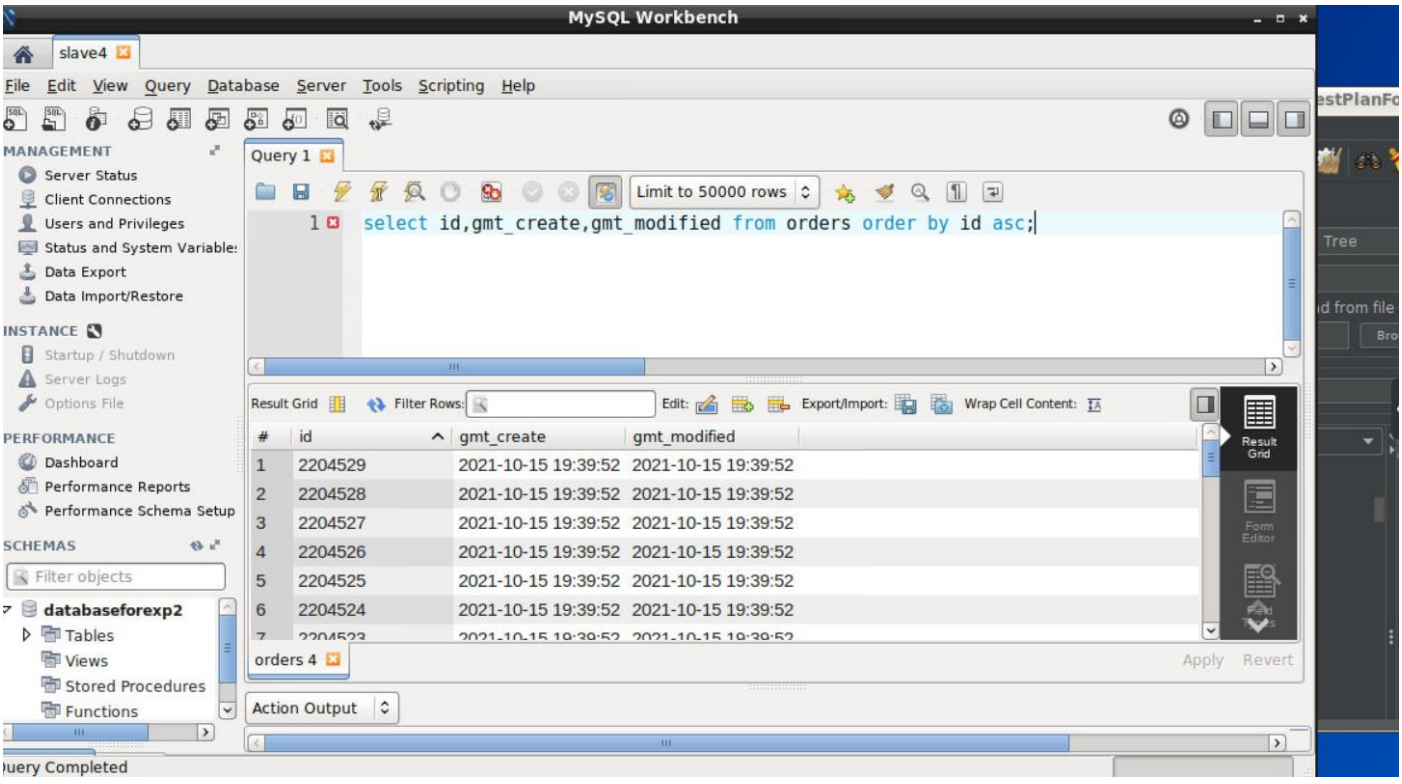


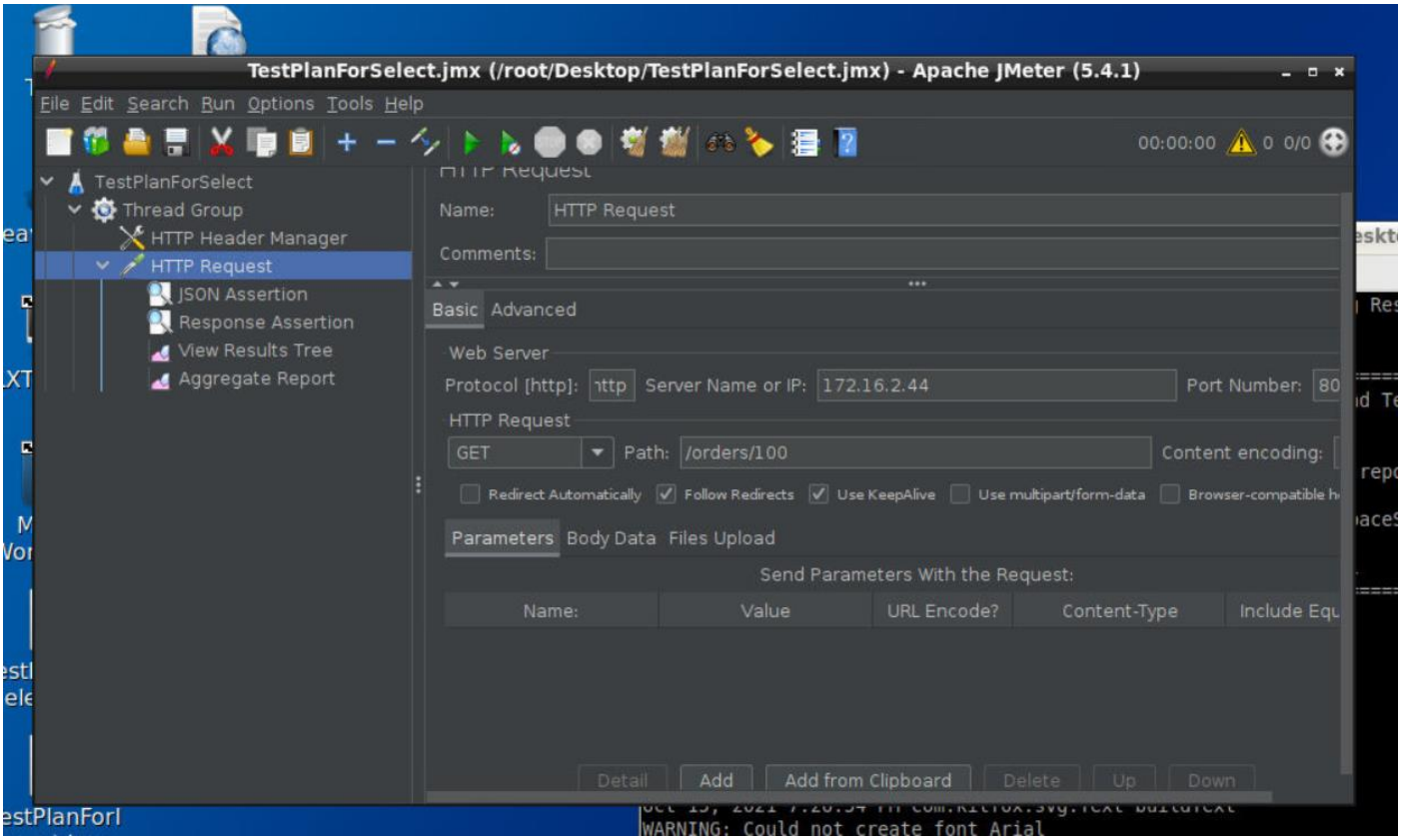
这是测试 post 的测试计划，我们只开 200 个线程，在 Body 中填入要求的 json 数据

简单的在 jmeter 中的测试结果如下：

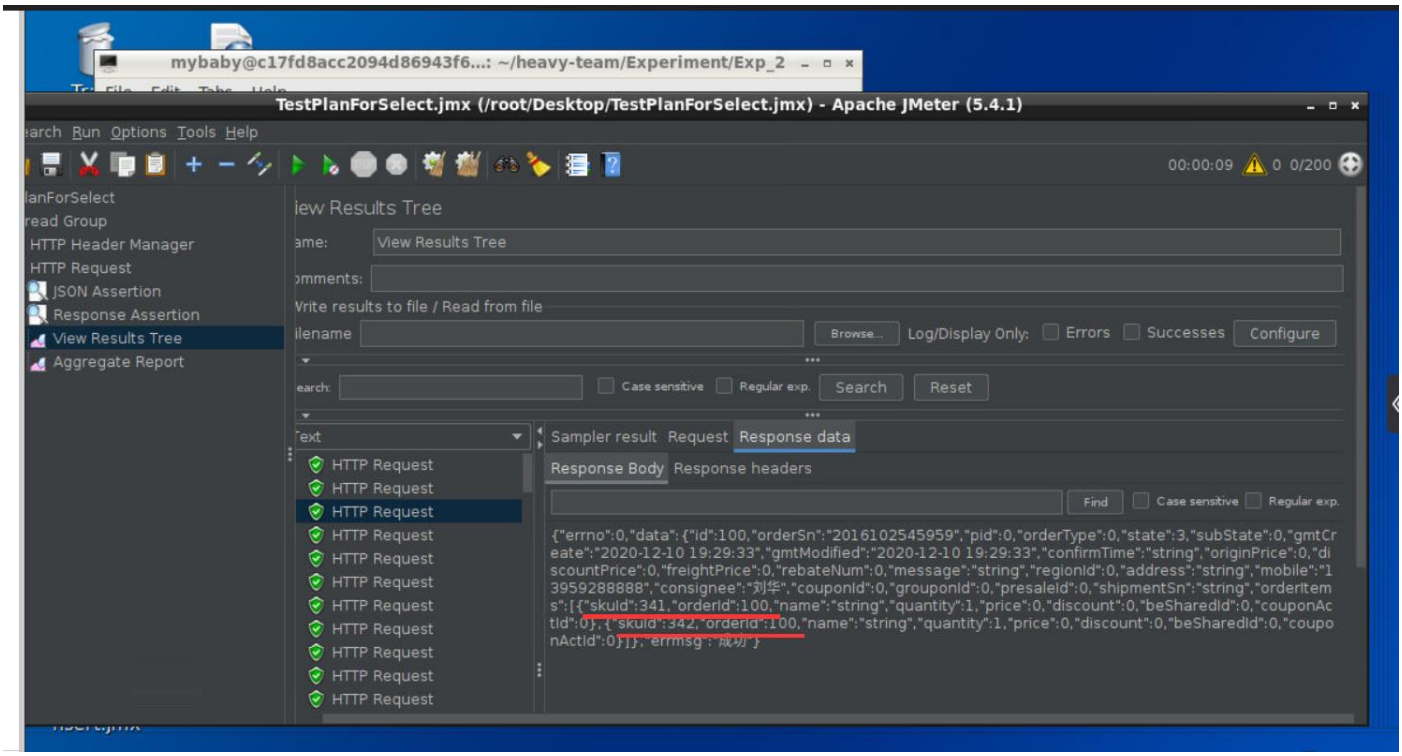


在 workbench 中，我们使用 select id,gmt_create,gmt_modified from orders order by id asc;可以看到新添加的数据





这是测试 get 的测试计划，我们选择 id 为 100 的 order 记录，因为通过观察，id 为 100 的 order 对应有两个 orderItem，可以测试 resultMap 以及一对多的映射，因此我们使用这一个 id 进行测试。同样开 200 个线程



Get 的测试计划执行结果，可以看到在结果中进行了一对多的映射

5. **<u>设计过程中的问题</u>**:

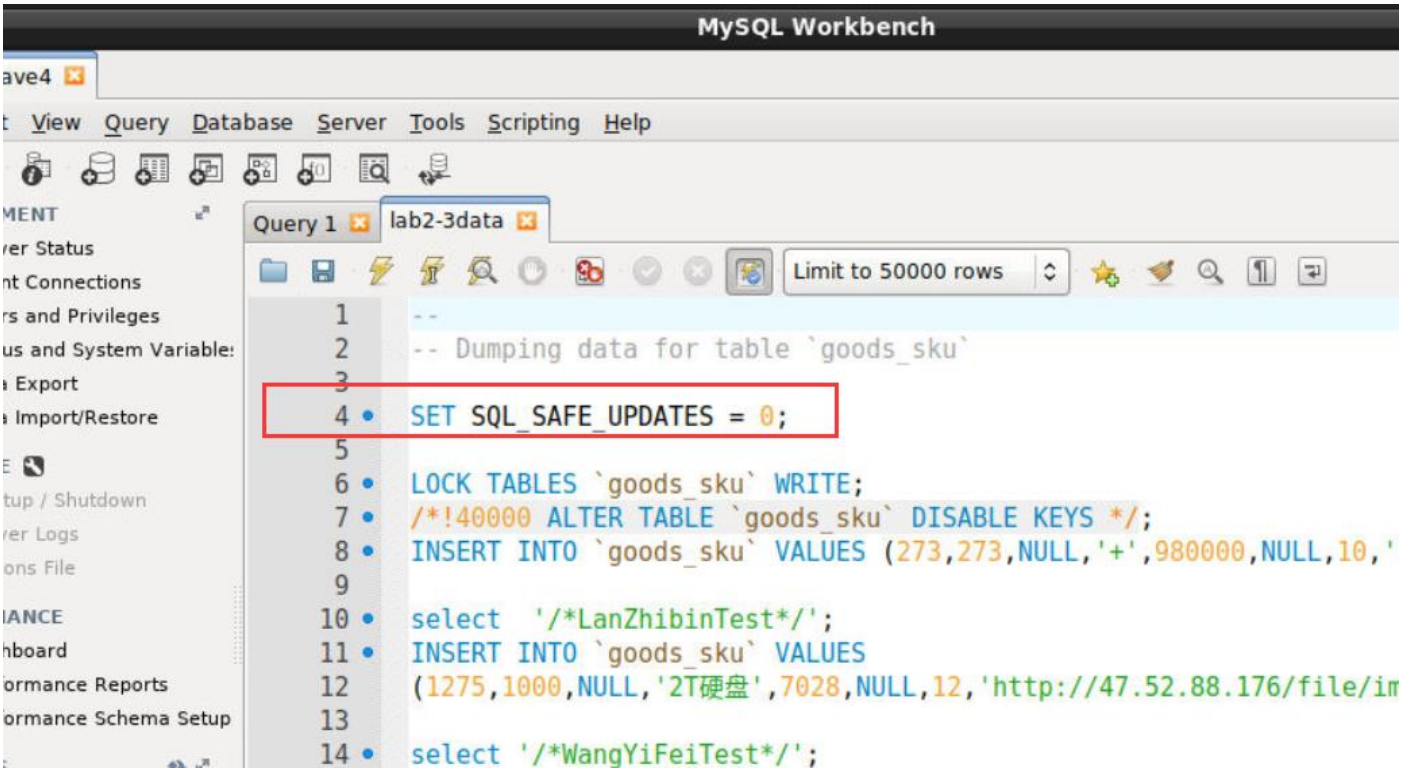
首先，遇到的第一个问题是日期时间类型数据转换的问题，根据观察数据库原有数据，可以发现 gmt_create 等时间类型的字段，或者是采用 CURRENT_STAMP 数据类型的字段，对应到 java 中是 LocalDateTime 类型。这个类型使用时挺方便，但如果涉及到转换字符串的时候，就会出现问题。LocalDateTime 的 toString()源码如下：

```
    Returns: a string representation of this date-time, not null
    @Override
    public String toString() { return date.toString() + 'T' + time.toString(); }
```

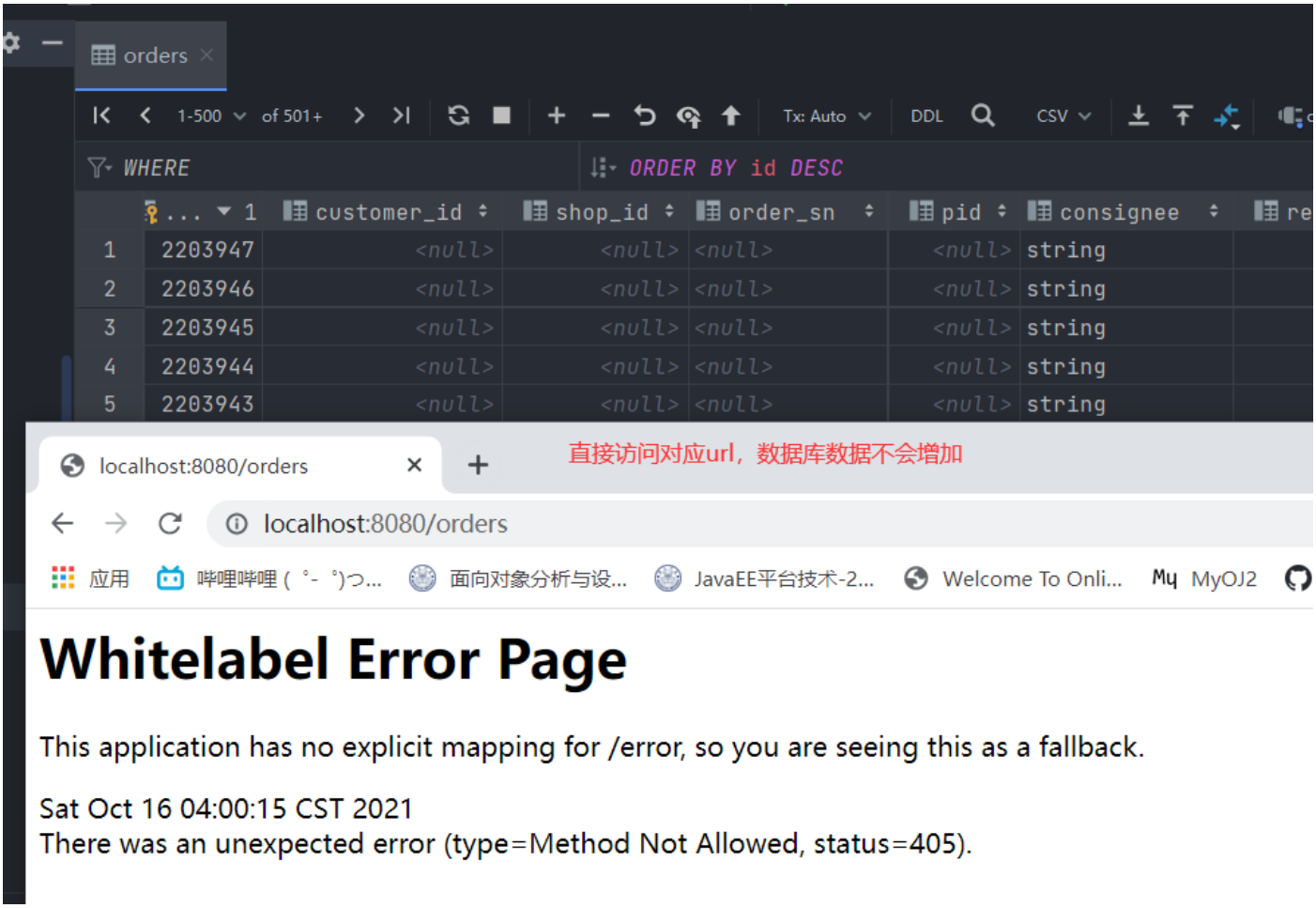
一个"2021-10-15 21:22:23"的时间类型的数据，如果使用 toString()，会变成"2021-10-15T21:22:23"，而我们在比对 JSON 字符串以及返回给前端时，不需要中间的 T 字符，因此需要想办法去掉。网络上大多数使用 jackson-datatype-jsr310，并通过一些全局配置进行实现，但效果不佳，因此，我们使用比较简单粗暴的办法，在所有涉及时间转换字符串的地方使用 replaceAll，将 T 替换为空格

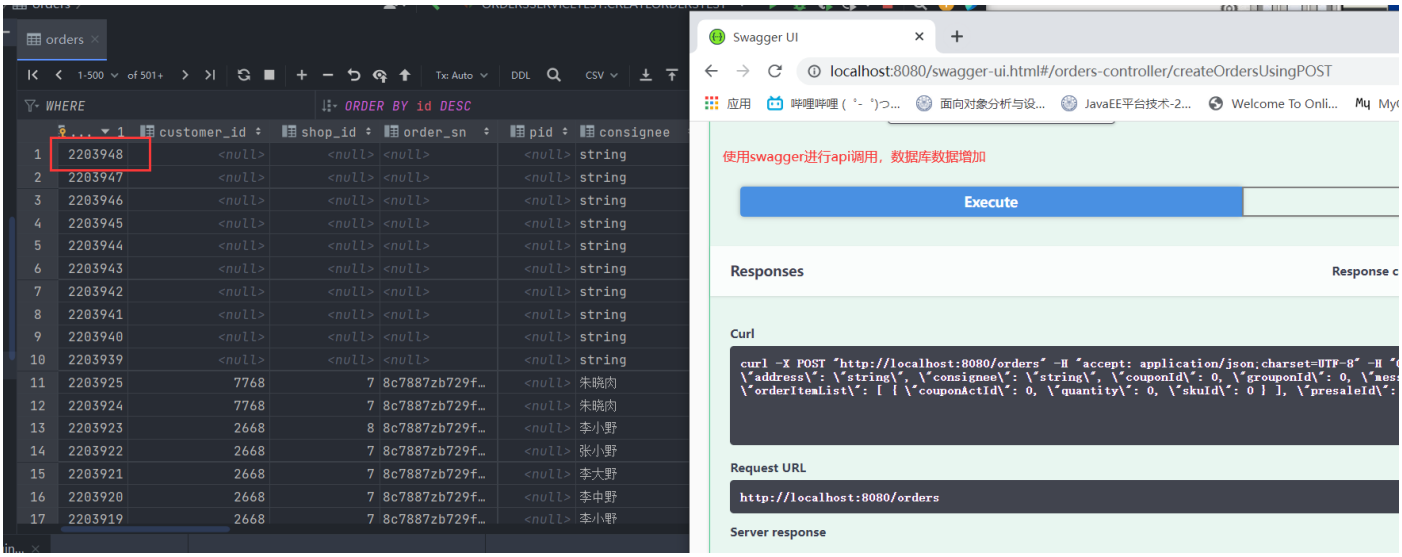
```
if (null != orders.getGmtCreate()) {
    if (orders.getGmtCreate().toString().contains("T"))
    {
        this.gmtCreate = orders.getGmtCreate().toString().replaceAll( regex: "T",  replacement: " ");
    } else {
        this.gmtCreate = orders.getGmtCreate().toString();
    }
} else {
```

第二个遇到的问题是，在使用提供的 Lab2-3data.sql 时，会出现"You are using safe update mode..."的错误提示，原因是 MySQL 为了提高数据库的安全性能，默认使用了 safe-updates 模式，在该模式下，要想执行更新操作（update/delete）,要使用 where 语句，且必须带有该表的主键约束。但提供的 sql 文件中部分语句的 where 没有使用主键约束，如"UPDATE orders SET shop_id = 1 WHERE shop_id IS NULL;"，才导致了这个错误。解决方法是在脚本文件开头添加 SET SQL_SAFE_UPDATES = 0;



第三个遇到的问题是，在本机上使用 springboot:run 将系统运行，然后在浏览器地址栏直接输入 localhost:8080/orders 进行 post 的测试，数据库中数据并不会增加；但如果使用 swagger 的 API 调试功能执行 post，数据库中数据就会增加。在服务器中进行 jmeter 测试时也会增加。**具体原因未知，暂未解决**





五、实验结果分析

将 jtl 导出为 html，方便观察

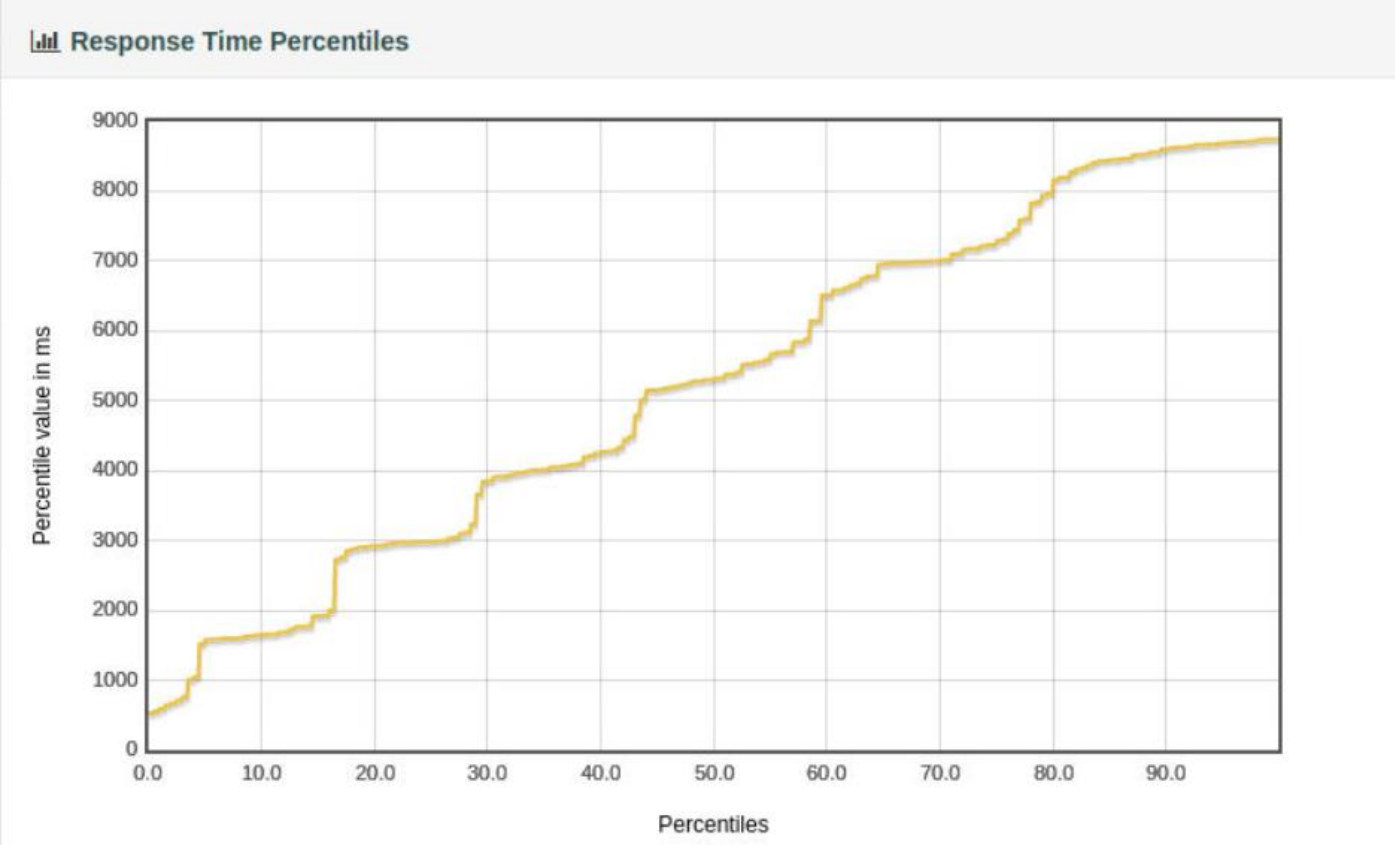
这是 GET /orders/100 的总数据表

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**5242.64**	**533**	**8749**	**5312.50**	**8612.70**	**8693.75**	**8747.99**	**22.85**	**21.16**	**3.91**
HTTP Request	200	0	0.00%	5242.64	533	8749	5312.50	8612.70	8693.75	8747.99	22.85	21.16	3.91

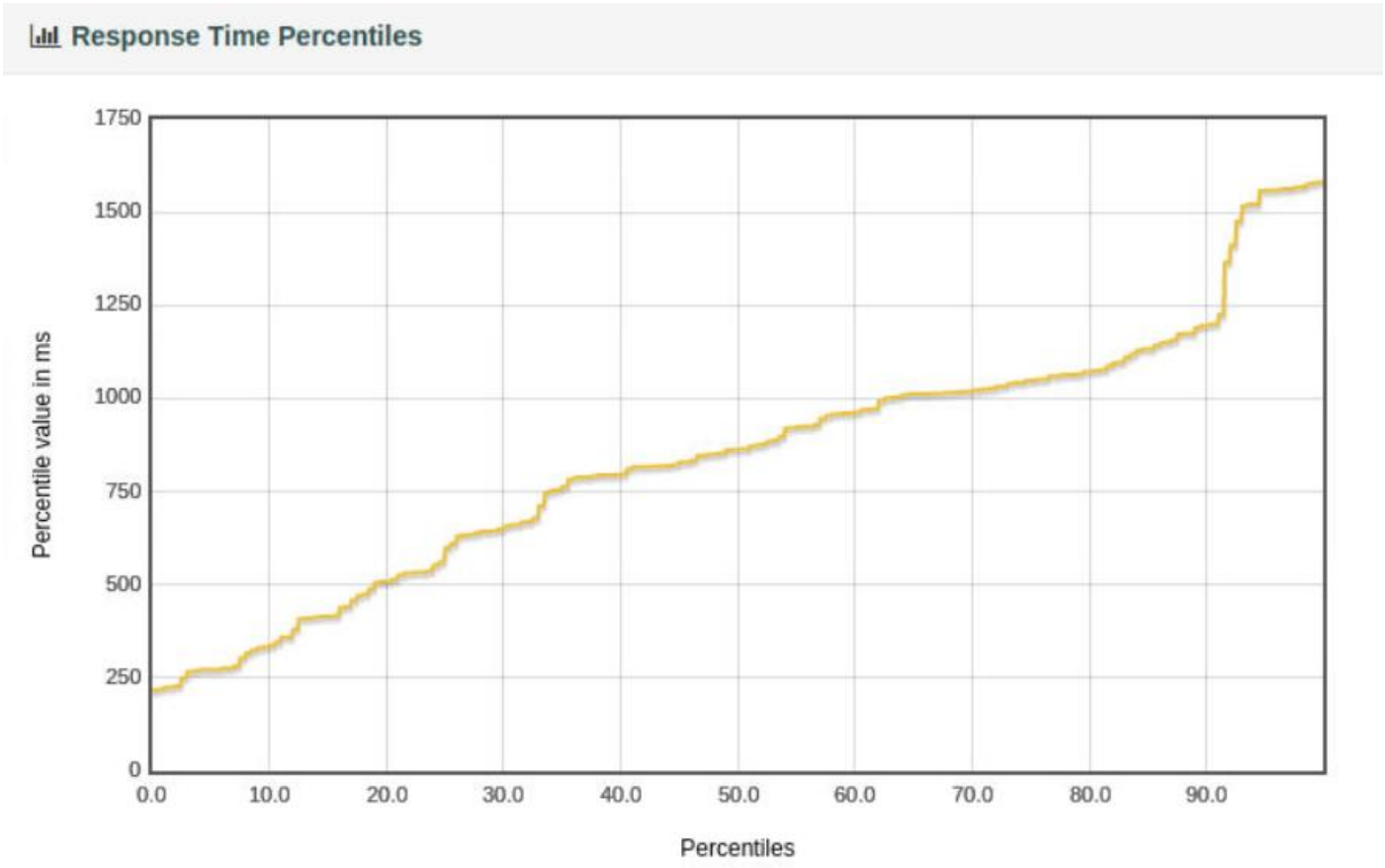
这是 POST /orders 的总数据表

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**843.34**	**217**	**1581**	**863.50**	**1197.80**	**1558.95**	**1578.97**	**124.22**	**85.28**	**58.84**
HTTP Request	200	0	0.00%	843.34	217	1581	863.50	1197.80	1558.95	1578.97	124.22	85.28	58.84

这是 GET /orders/100 的响应时间分布百分比



这是 POST /orders 的响应时间分布百分比



从这四张图我们可以看出，该实验中的 select 比 insert 要慢上很多，主要原因在于 insert 只是简单的向两个表中分别插入数据，而 select 要涉及到两张表的连接、组建、映射等，这样的复杂的操作直接造成了响应时间剧烈增加，甚至达到了 insert 的四倍左右
这样的对比告诉我们一个经验，在设计 E-R 图以及进行领域模型构建、类图设计等前期设计工作时，尽量减少一对多、多对多的映射这样的设计，也要避免使用中间表的使用，如果实在无法避免，要尽量减小这样操作的影响

六、实验代码 git 地址

https://github.com/529106896/HeavyTeam.git
本次实验位于 HeavyTeam/Experiment/Exp_2 目录下
（若需运行，需保持 application.yaml 中数据库 url 一致）

七、附加文件

1. 测试用 jtl 文件
2. 备用实验代码
3. 备用 sql 建库建表文件