

《JavaEE 平台技术》实验报告

实验名称:	基于 MyBatis 的关联实现方案
实验日期:	2021.10.18
实验地点:	四号楼 A404
提交日期:	2021.10.21

组号:	1-07
组名:	这队更是重量级
专业年级:	软件工程 2019 级
学年学期:	21-22 学年上半学期

一、 实验目的

- 1、掌握 MyBatis 的 ResultSet 的使用方法
- 2、掌握用 SpringBoot 应用 Dao 层的实现方法
- 3、对在 MyBatis 中用不同方式实现数据表关联的效率比较

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、 实验设计

总体是在实验二的基础上进行简单修改

(1) 对于使用 SQL 进行表的关联的 Dao 层代码:

```
public ReturnObject<List<Orders>> findOrders(OrdersPo ordersPo) {  
  
    //long startTime = System.currentTimeMillis();  
    List<OrdersPo> ordersPos = ordersMapper.findOrders(ordersPo);  
    //long endTime = System.currentTimeMillis();  
    //System.out.println("OrdersDao: findOrders: find order spend " + (endTime - startTime));  
  
    List<Orders> retOrders = new ArrayList<>(ordersPos.size());  
  
    for (OrdersPo o : ordersPos) {  
        List<OrderItemPo> orderItemPos = o.getOrderItemPoList();  
        Orders orders = new Orders(o);  
        List<OrderItem> orderItemList = new ArrayList<>(orderItemPos.size());  
        for (OrderItemPo orderItemPo : orderItemPos) {  
            OrderItem orderItem = new OrderItem(orderItemPo);  
            orderItemList.add(orderItem);  
        }  
        orders.setOrderItemList(orderItemList);  
        retOrders.add(orders);  
    }  
    //long endTime = System.currentTimeMillis();  
    //System.out.println("ordersDao : findOrders: left join total spend " + (endTime - startTime));  
    return new ReturnObject<>(retOrders);  
}
```

对应的 Mapper 中的代码如下，直接在 SQL 中进行左连接，返回一个 ResultMap 类型

```

<select id="findOrders" parameterType="OrdersPo" resultMap="OrdersWithItemMap">
    select
        os.id as id,
        os.order_sn,
        os.order_type,
        os.state,
        os.substate,
        os.gmt_create,
        os.gmt_modified,
        os.confirm_time,
        os.origin_price,
        os.discount_price,
        os.freight_price,
        os.rebate_num,
        os.message,
        os.region_id,
        os.address,
        os.mobile,
        os.consignee,
        os.coupon_id,
        os.groupon_id,
        os.presale_id,
        os.shipment_sn,
        oi.id as order_item_id,
        oi.goods_sku_id as order_item_goods_sku_id,
        oi.order_id as order_item_order_id,
        oi.quantity as order_item_quantity,
        oi.price as order_item_price,
        oi.discount as order_item_discount,
        oi.coupon_activity_id as order_item_coupon_activity_id,
        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
        os.state != 2
        <if test="id!=null">and os.id = #{id}</if>
        <if test="consignee!=null and consignee!=''">and consignee = #{consignee}</if>
</select>

```

对应的 ResultMap 定义:

```

<resultMap id="OrdersWithItemMap" type="OrdersPo" autoMapping="true">
    <id property="id" column="id"/>
    <collection property="orderItemPoList" ofType="com.example.exp_3.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>

```

(2) 对于先查询一个对象，然后在 Dao 层完成对象关联：

对应的 Dao 层代码：

先选出一个不带 OrderItem 的 Orders，然后使用 for 循环，按照 Orders 的 id 逐个找出对应的 OrderItem，再进行拼接

```
public ReturnObject<List<Orders>> findOrdersWithItems(OrdersPo ordersPo) {  
    List<OrdersPo> ordersPos = ordersMapper.findOrdersWithoutItems(ordersPo);  
  
    List<Orders> retOrders = new ArrayList<>(ordersPos.size());  
  
    for (OrdersPo o : ordersPos) {  
        List<OrderItemPo> orderItemPos = ordersMapper.findOrderItems(o.getId());  
  
        Orders orders = new Orders(o);  
  
        List<OrderItem> orderItemList = new ArrayList<>(orderItemPos.size());  
        for (OrderItemPo orderItemPo : orderItemPos) {  
            OrderItem orderItem = new OrderItem(orderItemPo);  
            orderItemList.add(orderItem);  
        }  
        orders.setOrderItemList(orderItemList);  
        //System.out.println(orders.getOrderItemList());  
        retOrders.add(orders);  
    }  
    return new ReturnObject<>(retOrders);  
}
```

对应的 Mapper:

```
<select id="findOrderItems" parameterType="OrderItemPo" resultType="OrderItemPo">  
    select id,  
           order_id,  
           goods_sku_id,  
           quantity,  
           price,  
           name,  
           coupon_activity_id,  
           be_share_id,  
           gmt_create,  
           gmt_modified  
    from order_item  
    where order_id = #{orderId}  
</select>
```

```

<select id="findOrdersWithoutItems" parameterType="OrdersPo" resultType="OrdersPo">
    select
        id,
        customer_id,
        shop_id,
        order_sn,
        order_type,
        state,
        substate,
        gmt_create,
        gmt_modified,
        confirm_time,
        origin_price,
        discount_price,
        freight_price,
        rebate_num,
        message,
        region_id,
        address,
        mobile,
        consignee,
        coupon_id,
        groupon_id,
        presale_id,
        shipment_sn,
        be_deleted
    from orders
    where id = #{id}
</select>

```

以下语句用于测试，打包时记得删掉

因为我们在实验二中，将测试线程设置为 200，发生了严重的阻塞，影响正常数据的观测。因此，我们需要尽量排除一切可能产生阻塞的原因，以尽量观测到正常的数据

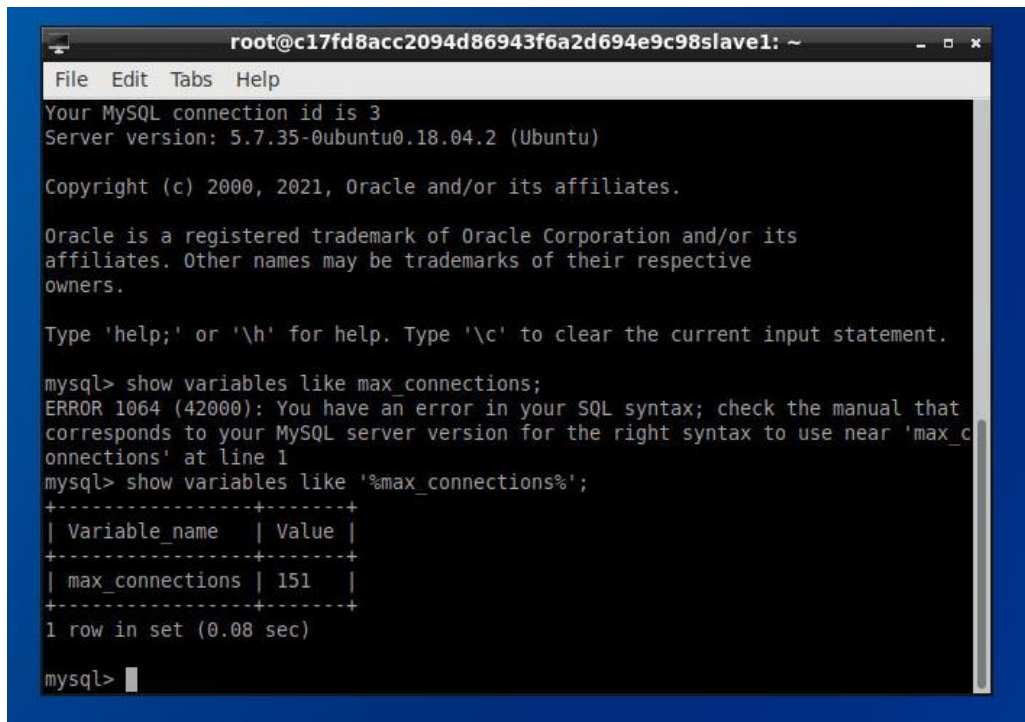
- (1) 设置 Druid 连接池的最大连接数

```

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/*
        allow: 172.16.1.99

```

- (2) 查看 mysql 默认最大连接数，mysql 8.0 默认为 151



```
root@c17fd8acc2094d86943f6a2d694e9c98slave1: ~
File Edit Tabs Help
Your MySQL connection id is 3
Server version: 5.7.35-0ubuntu0.18.04.2 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

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

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

mysql> show variables like max_connections;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near 'max_c
onnections' at line 1
mysql> show variables like '%max_connections%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_connections | 151 |
+-----+-----+
1 row in set (0.08 sec)

mysql>
```

- (3) 将 jmeter、jvm、mysql 分在三台服务器上部署，减少其他因素的干扰
我们在图形界面 slave1 中运行 jmeter、在 slave4 中使用 runjava.sh 运行 jar 文件，在 slave3 中部署 mysql 以及添加我们需要的库与数据
- (4) 控制测试计划线程数，逐步找到阻塞临界值，之后在临界值之内进行测试，并查看最终测试数据

四、 具体实验过程

1. 找到阻塞临界值

我们将两种方式简记为 **Left Join** 和 **Single Select**

为了找到临界值，我们在 jmeter 的 GUI 模式下中从 20 个线程开始测试，每个线程运行 5 次，并统计响应时间的平均值、最大值、最小值、90%响应时间、95%响应时间
最终得到的统计表如下：

线程数		单				avg	多				avg
20	avg	39	32	32	43	36.5	25	28	26	26	26.25
	max	65	69	51	71	64	41	44	38	40	40.75
	min	33	25	26	30	28.5	15	20	19	21	18.75
	90%	46	41	44	58	47.25	32	34	31	30	31.75
	95%	46	47	46	58	49.25	38	41	35	31	36.25
30	avg	34	41	31	31	34.25	21	20	22	25	22
	max	69	100	52	65	71.5	51	39	39	65	48.5
	min	25	25	24	24	24.5	15	14	14	15	14.5
	90%	60	79	52	53	61	30	30	30	41	32.75
	95%	61	95	52	58	66.5	31	31	30	43	33.75
35	avg	59	69			64	19	20			19.5
	max	163	193			178	59	52			55.5
	min	24	24			24	14	14			14
	90%	120	146			133	29	29			29
	95%	147	162			154.5	31	37			34
40	avg	146	50	34	50	70	18	16	18	19	17.75
	max	260	124	73	116	143.25	37	35	64	40	44
	min	24	24	23	24	23.75	14	13	14	14	13.75
	90%	232	112	61	103	127	28	22	19	33	25.5
	95%	248	116	68	112	136	31	24	38	35	32
50	avg	586					18				
	max	814					51				
	min	256					14				
	90%	742					31				
	95%	757					43				
60	avg						22				
	max						79				
	min						13				
	90%						50				
	95%						66				
70	avg						48	30	30	84	
	max						127	113	132	264	
	min						12	12	13	14	
	90%						108	78	68	154	
	95%						115	90	101	169	
100	avg						279				
	max						484				
	min						138				
	90%						383				
	95%						409				

从总体来看，Left Join 的速度快于 Single Select（在正常范围内，快大约 10ms）

逐步加大测试的线程数，可以看到 max 响应时间增加较快，我们把响应时间明显增加看作阻塞边界，可以看出：

对于 Single Select，更快迎来阻塞边界，约 35-40 个线程即发生较大程度的阻塞，到 50 个线程几乎完全阻塞

对于 Left Join，阻塞边界到来的更晚，在 60 个线程仍可保持较短的响应时间，在 70 逐渐发生阻塞，100 几乎完全阻塞

因此，为了保持测试数据的正常，以进行正确的查询效率对比，我们选择 20 个线程作为我们最终的测试线程数

2. 使用 jmeter 命令行测试，并导出最终实验结果

Left Join 的最终结果表格：

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	20	0	0.00%	39.45	15	134	17.00	128.00	133.70	134.00	23.64	21.89	4.04
HTTP Request	20	0	0.00%	39.45	15	134	17.00	128.00	133.70	134.00	23.64	21.89	4.04

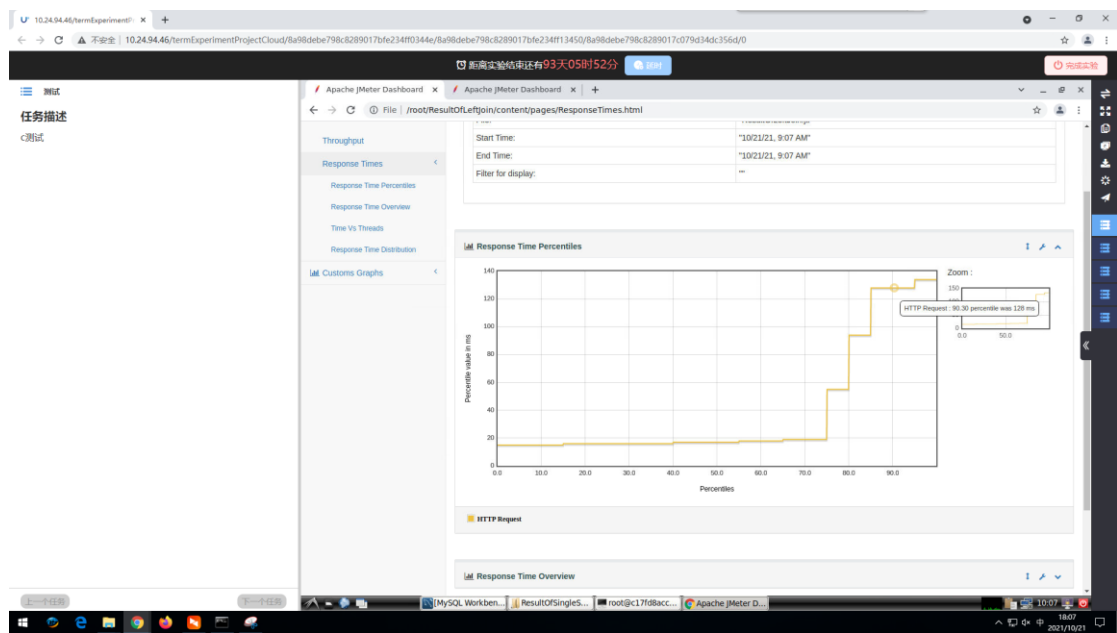
Single Select 的最终结果表格：

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	20	0	0.00%	63.65	25	181	34.00	173.00	180.60	181.00	23.09	21.38	4.15
HTTP Request	20	0	0.00%	63.65	25	181	34.00	173.00	180.60	181.00	23.09	21.38	4.15

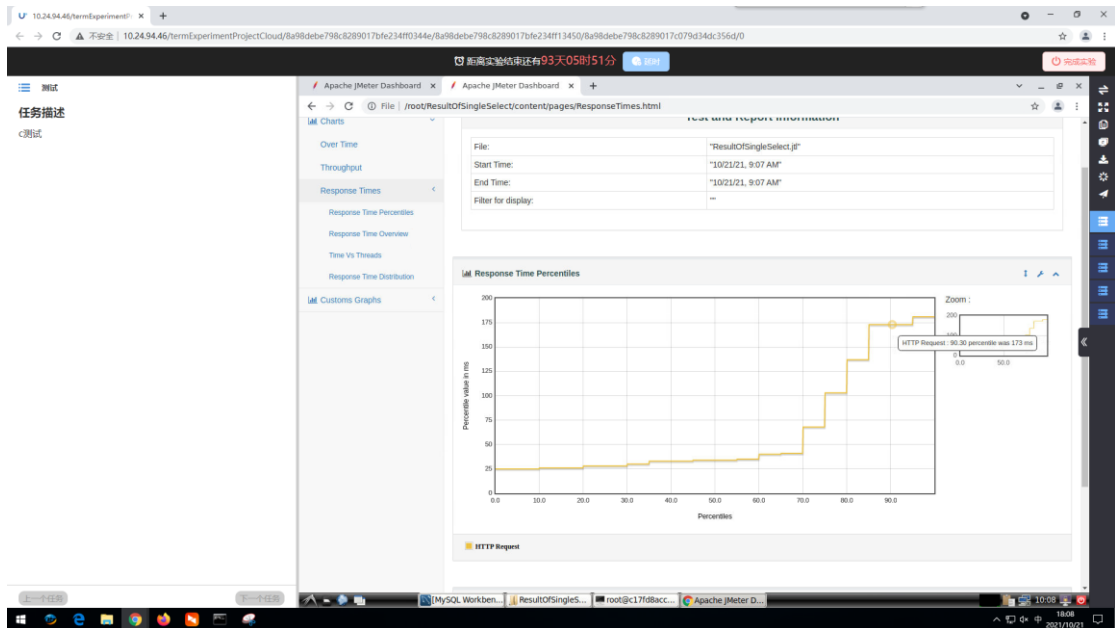
可以看到，jmeter 使用命令行进行测试与使用 GUI 进行测试，结果有较大的差异，经过查询得知，jmeter 图形化界面会耗费更多的 CPU 与内存，而且 jmeter 图形化界面不能进行大负载的压力测试，只能用于制定测试计划，然后用命令行方式执行

3. 实验结果分析

Left Join 的 90%响应时间——128ms



SingleSelect 的 90%响应时间——173ms



从以上两图可以看出，Left Join 的响应时间明显快于 Single Select，差别约为 50ms
(两种方式都是在大约 70% 出现明显阻塞)

对于出现差别的原因，我们猜测是连表查询与单表查询拼接的效率差异，为了验证猜想，我们在代码中添加 `currentTimeMillis` 来获取指定位置段的时间差

```
public ReturnObject<List<Orders>> findOrders(OrdersPo ordersPo) {
    long startTime = System.currentTimeMillis();
    List<OrdersPo> ordersPos = ordersMapper.findOrders(ordersPo);
    long endTime = System.currentTimeMillis();
    System.out.println("OrdersDao: findOrders: find order spend " + (endTime - startTime));
}
```

我们分别测试 Dao 层、Service 层、Controller 层以及测试代码中指定代码段的时间差，这种手动测试的方法会受到多方面的干扰，与真实环境有较大差异，因此结果并不可完全参考，但我们仍可以从中获取到一些结果：

Dao 层——Service 层——Controller 层等层次结构之间的切换耗时并不明显，时间差别主要在于 Dao——Mapper——SQL 这个层级（即需要访问数据库的部分）

经查询得知，在 MyBatis 原理中，Mapper 每一次请求访问数据库，都会新建一个 SqlSession 与数据库进行交互

所以，出现上述差异的原因就比较明朗了：

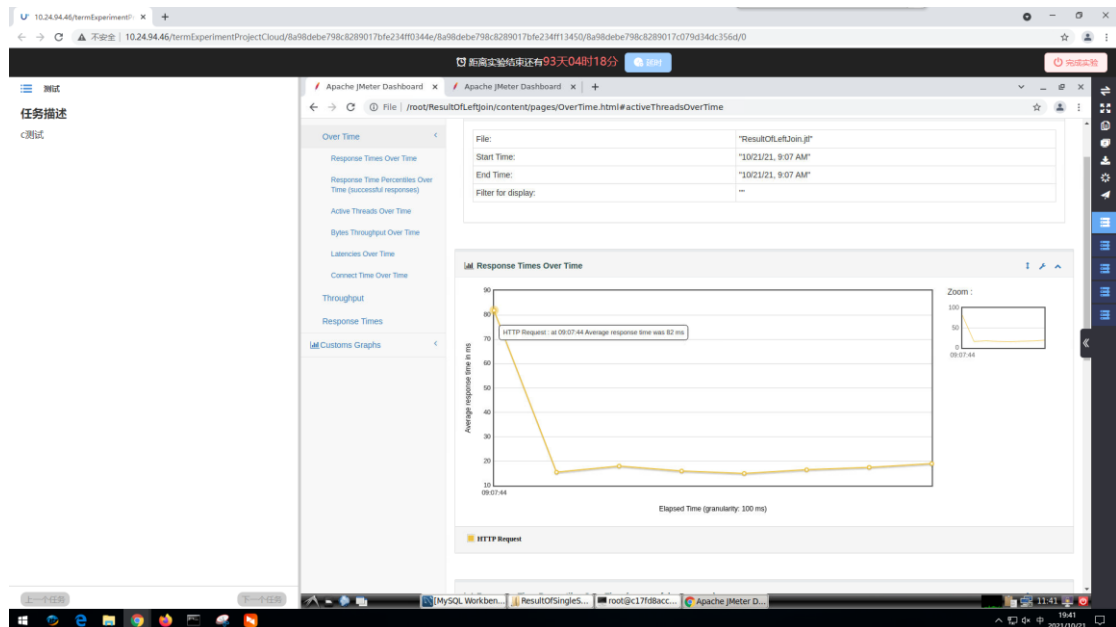
对于 Left Join，虽然我们一般认知中，进行外连接操作都慢于正常的查询语句，但在本次实验中，我们进行一个 Orders 的查询，只访问了一次数据库，就将 Orders 和对应的 OrderItem 全部选出，后续不再需要访问数据库。

而对于 Single Select，我们先是访问数据库查询出了一个不带 OrderItem 的 Orders，之后在一个 for 循环中，根据 Orders 的 id 去多次访问数据库，次数多少决定于有多少 OrderItem 的 order_id 与 id 相等，因此，第二种方法的响应时间会被这种频繁访问数据库延缓，这也是之前测试中 Single Select 更早达到阻塞边界的原因

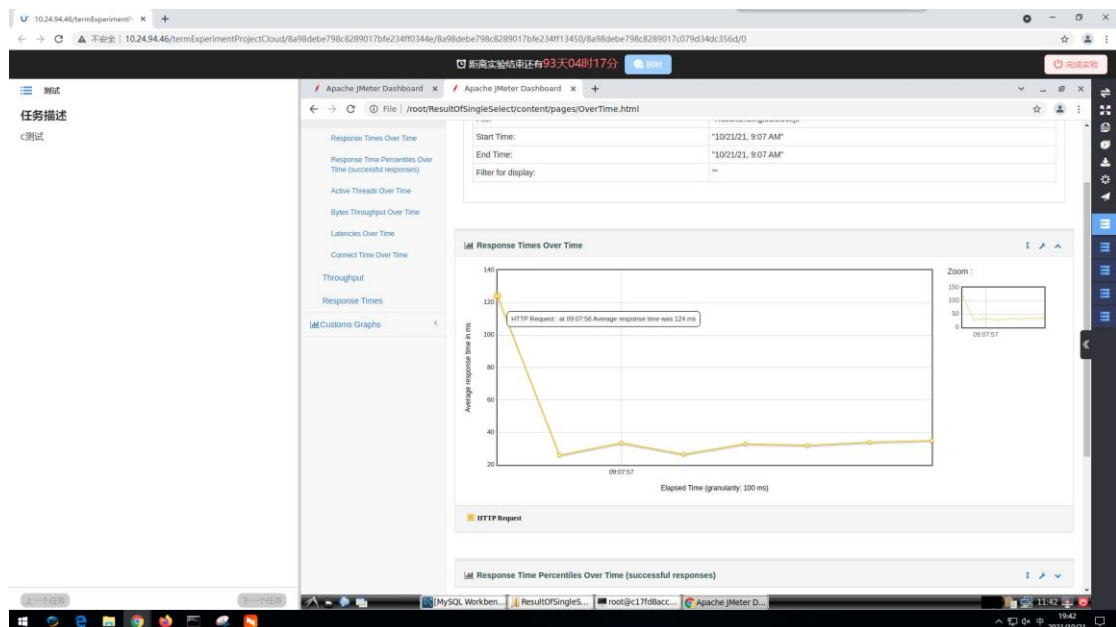
4. 其余发现

在进行反复测试的过程中，我们发现，不论是以上哪种方法、多少线程，开始测试后的最开始的平均响应时间几乎总是最长的，之后的响应时间则相对平稳

这是 Left Join 的 Response Time over time



这是 Single Select 的 Response Time Over Time



对此，我们的想法是：我们的请求都是针对同一请求，即 orders/100，MyBatis 针对同一请求，在第一次查询后建立缓存，之后的查询即可直接访问缓存，从而拥有更短的响应时间

为了验证此猜想，我们尝试在 jmeter 中为 get 请求添加变量，希望能在 20 个线程中查询不同的 orders，但因为 jmeter 的为 get 添加变量只能达到 order?id=100、order?id=101 这

样的效果，无法完成我们想要的 order/100、order/102、order/103 这样的请求。所以此猜想暂不做验证

五、 附加文件

Git 地址:

Gitee: <https://gitee.com/yjz6666774/heavy-team.git>

GitHub: <https://github.com/529106896/HeavyTeam.git>