

基于 Spring Data JPA 的关联实现方案

摘要: 本实验旨在基于 Spring Data JPA 实现 MySQL 数据库的关联查询，并与 MyBatis 不同关联方式的查询效率进行比较。通过 RESTful API /products?name=xxxx 查询产品完整信息，并使用 JMeter 对查询接口进行性能测试，分析不同关联查询方式在高并发环境下的响应时间和服务器负载情况。实验结果显示，Spring Data JPA 方案在查询效率和资源占用方面表现良好，尤其在高并发请求下具有较高的稳定性和较低的资源消耗。

关键词: Spring Data JPA, MyBatis, 关联查询, RESTful API, 性能测试, JMeter

问题描述:

在数据库应用开发中，表与表之间的关联查询是常见需求。传统的 MyBatis 框架提供了多种实现关联查询的方式（ResultMap 关联、Dao 层关联、Join 查询），但这些方式在效率和易用性上存在差异。为了进一步探索在高并发环境下的关联查询性能，本实验引入 Spring Data JPA，利用其自动生成查询功能，实现数据库表关联，并与 MyBatis 三种关联方式进行对比，分析各方案在不同并发量下的查询效率和资源使用情况。

实验设计:

1. 实验环境

- 服务器 A: Ubuntu 18.04, 2 核 1G 内存, 作为管理机, 运行 Docker、Maven、Git。
- 服务器 B: Ubuntu 18.04, 2 核 2G 内存, 运行 Docker, 部署 MySQL 数据库。
- 服务器 C: Ubuntu 18.04, 2 核 1G 内存, 运行 Docker, 部署 Spring 应用。
- 服务器 D: Ubuntu 18.04, 2 核 1G 内存, 安装 JMeter 5.6.3 用于测试。

2. 实验步骤

- (1) 实现了一个基于 JPA 查询数据库的 RESTful API 接口
 - a. GET /products?name=xxxx 通过商品名称查询查询产品完整信息
- (2) 设计并运行了 JMeter 测试，分别对 API 进行读写压力测试
- (3) 将编写好的代码打包成 docker 镜像，部署到 OOMALL-node1 服务器上
- (4) 使用华为云云监控服务 CES 监控了 MySQL 数据库服务器和 OOMALL-node1 服务器的 CPU 使用率、内存使用率和运行中的进程数
- (5) 分别对基于 JPA 查询数据库的接口和 MyBatis 自动生成的接口进行测试，并对比两种方法速度差异与服务器负载

结果分析与讨论:

1.代码部分

基于 JPA 实现查询的代码详见: <https://github.com/YUK1PEDIA/XMU-JavaEE-exp5-JPA.git>

2.分析部分

为了模拟**高并发、大负载**环境，设计下面五组 JMeter 测试参数涵盖从中等负载到极限负载的不同场景

测试组	Threads	Ramp-up Time	Loop Count
测试组 1	200	60s	10 次
测试组 2	400	90s	15 次
测试组 3	600	120s	20 次
测试组 4	1000	180s	25 次
测试组 5	1500	240s	30 次

参数说明：

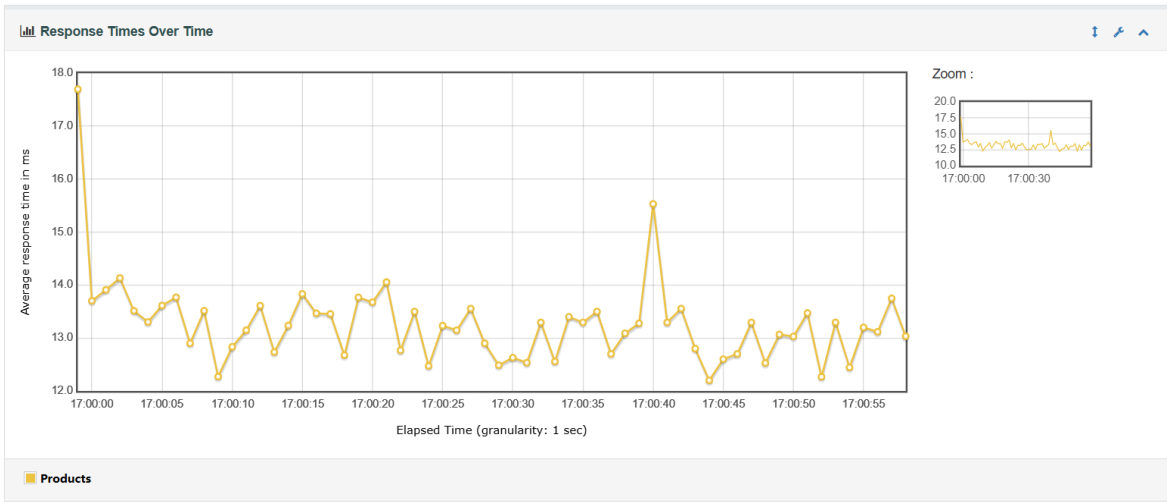
- (1) 线程数（Threads）
- a) 测试组 1 和测试组 2：分别使用 200 和 400 个线程，模拟中等并发场景，适合初期性能测试和比较。
 - b) 测试组 3：使用 600 个线程，模拟较高负载，开始测试系统在高并发下的表现。
 - c) 测试组 4 和测试组 5：使用 1000 和 1500 个线程，模拟极高并发和极限负载，评估系统性能极限和瓶颈。
- (2) Ramp-up 时间（Ramp-up Time）
- a) 为了避免瞬时高负载导致系统崩溃，随着线程数增加，Ramp-up 时间从 60 秒逐渐增加到 240 秒。
 - b) Ramp-up 时间较长，保证每秒启动的线程数量适中，帮助系统平稳进入高负载状态。
- (3) 循环次数（Loop Count）
- a) 从 10 次到 30 次逐步增加，确保每个线程都能多次执行请求，模拟长时间高负载环境。
 - b) 更高的循环次数可以帮助收集更多的数据样本，分析系统在长时间运行时的稳定性。

具体测试情况如下：

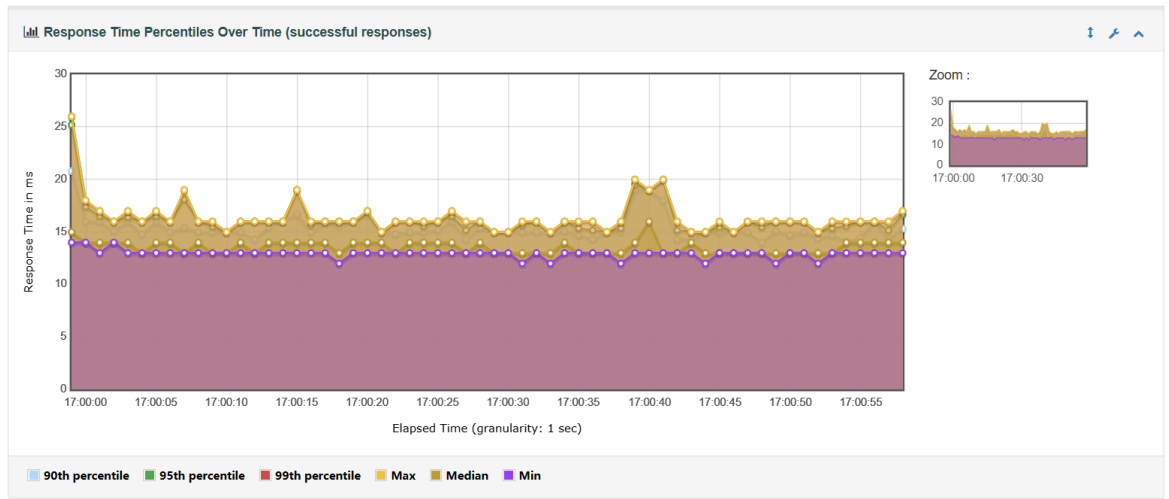
1. 基于 JPA 实现的数据库查询方案

(1) 测试组 1

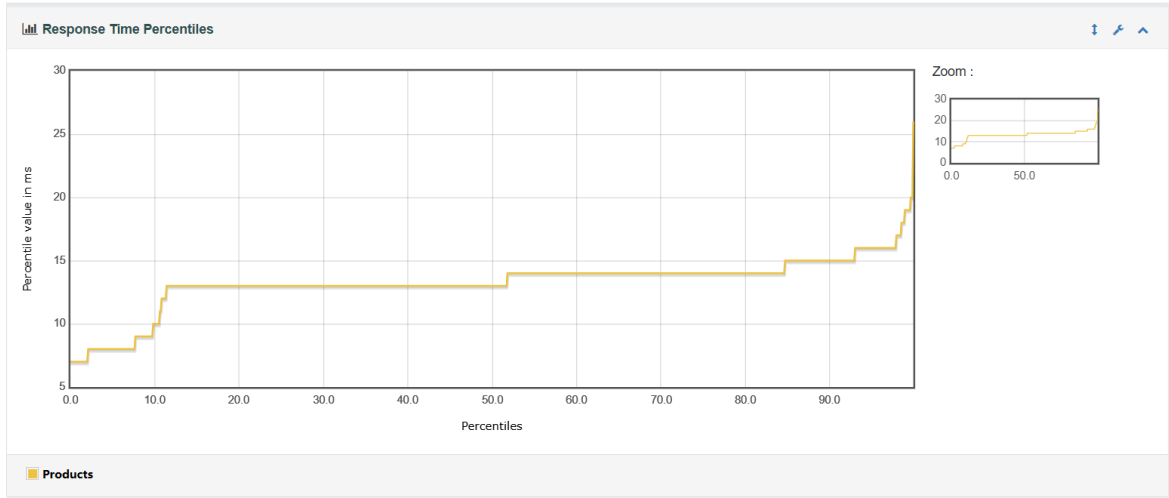
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

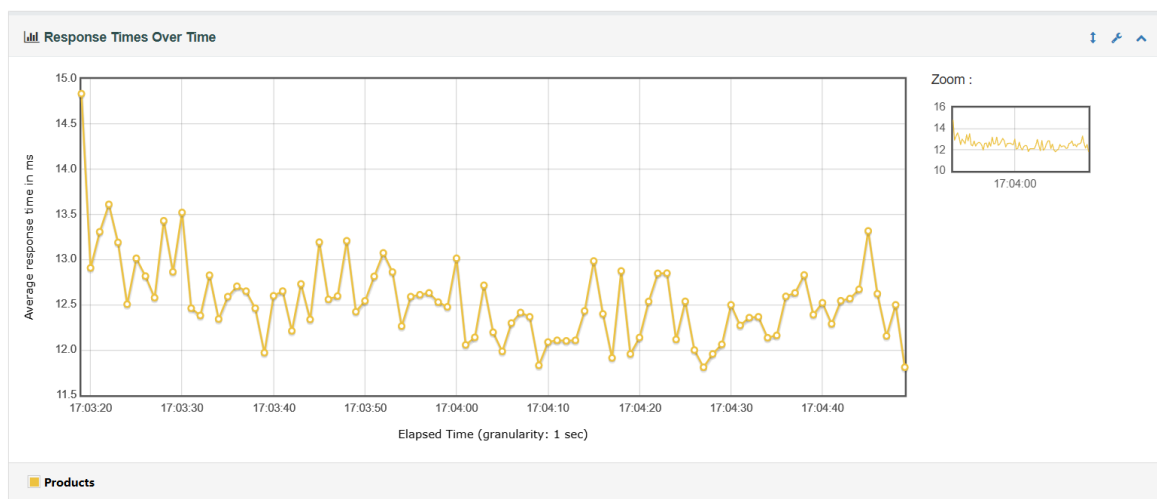


响应时间百分位数

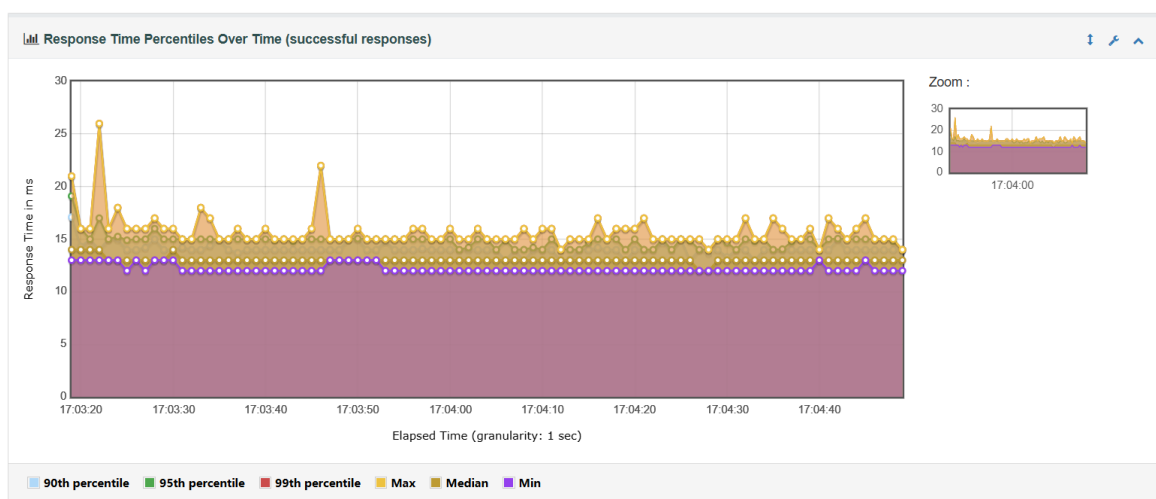


(2) 测试组 2

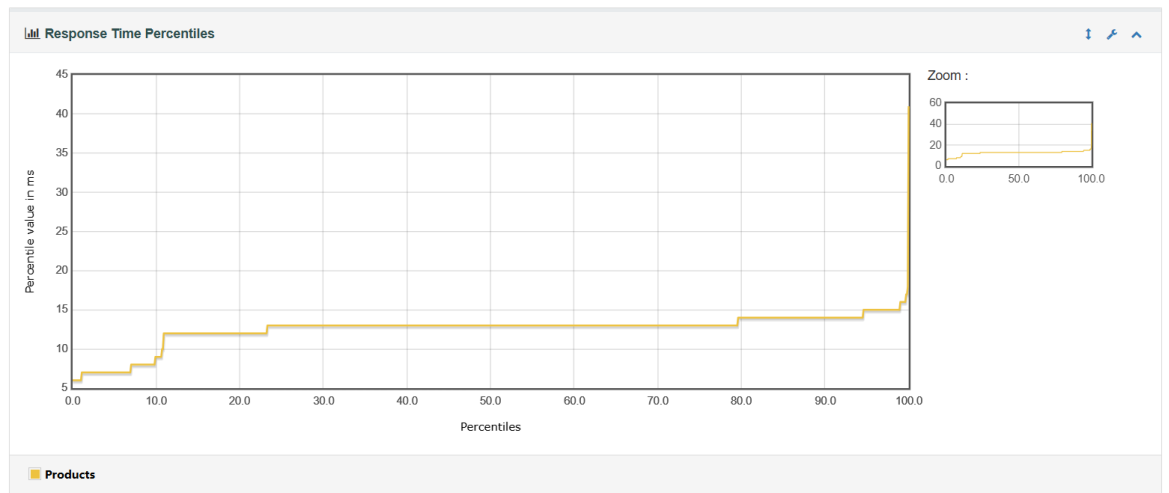
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

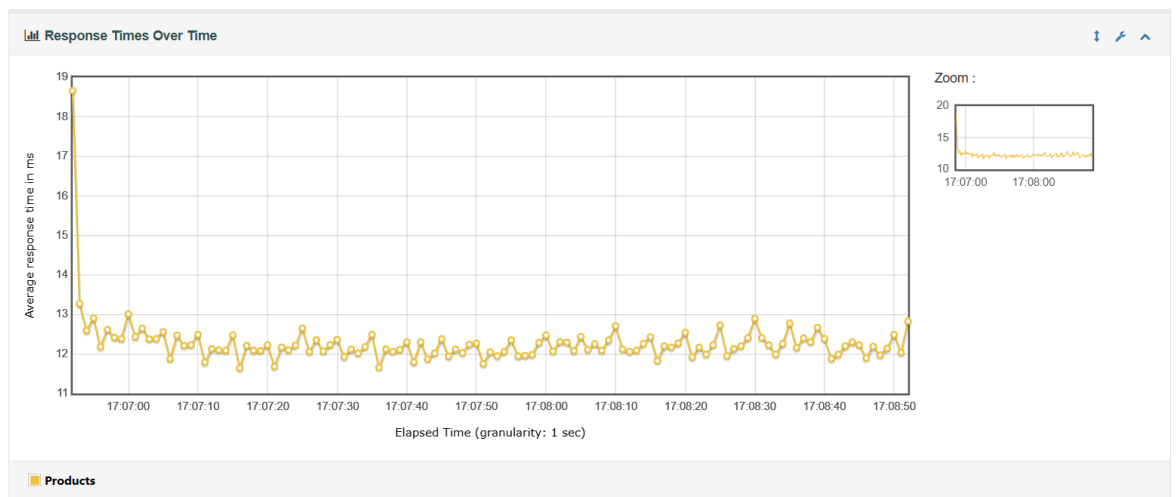


响应时间百分位数

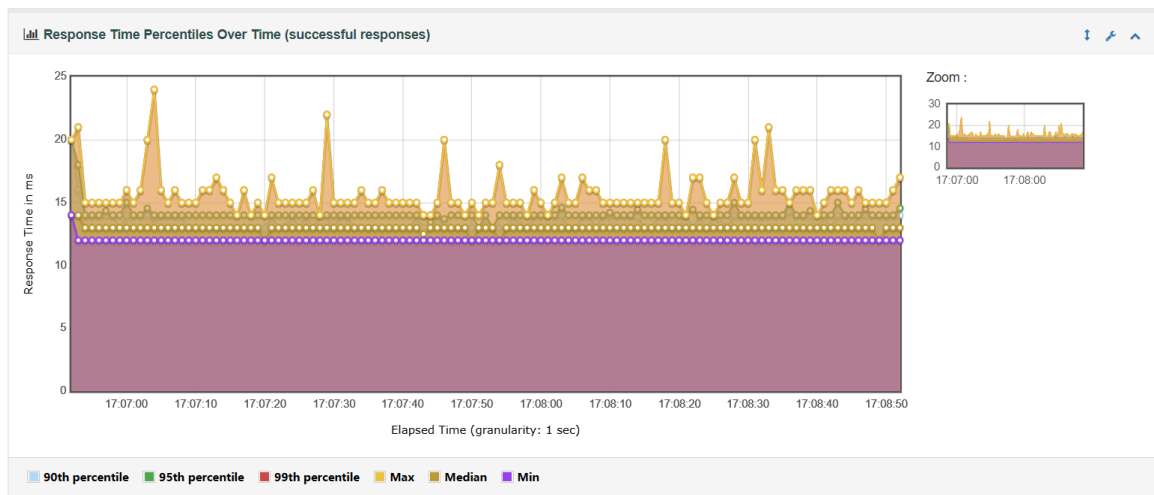


(3) 测试组 3

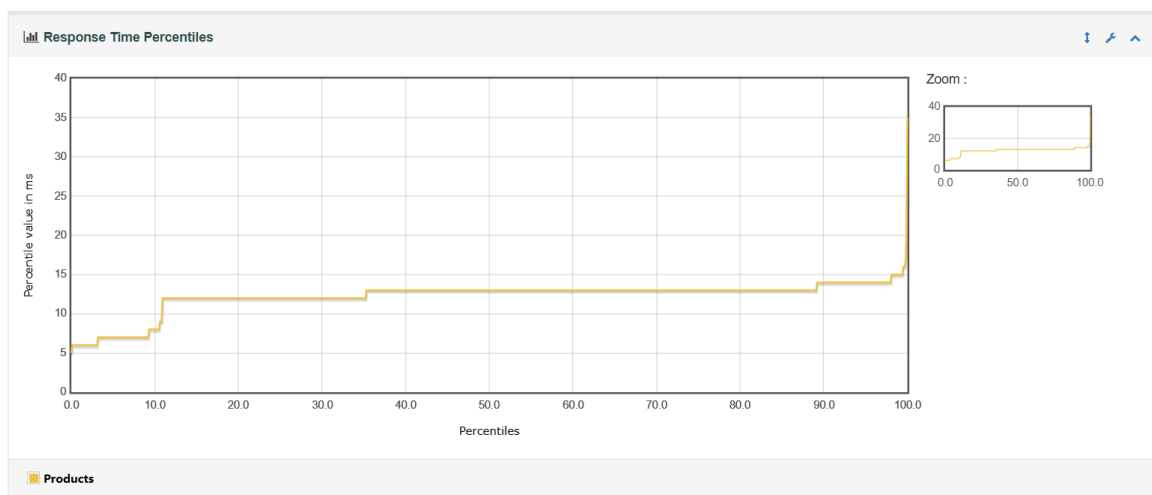
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

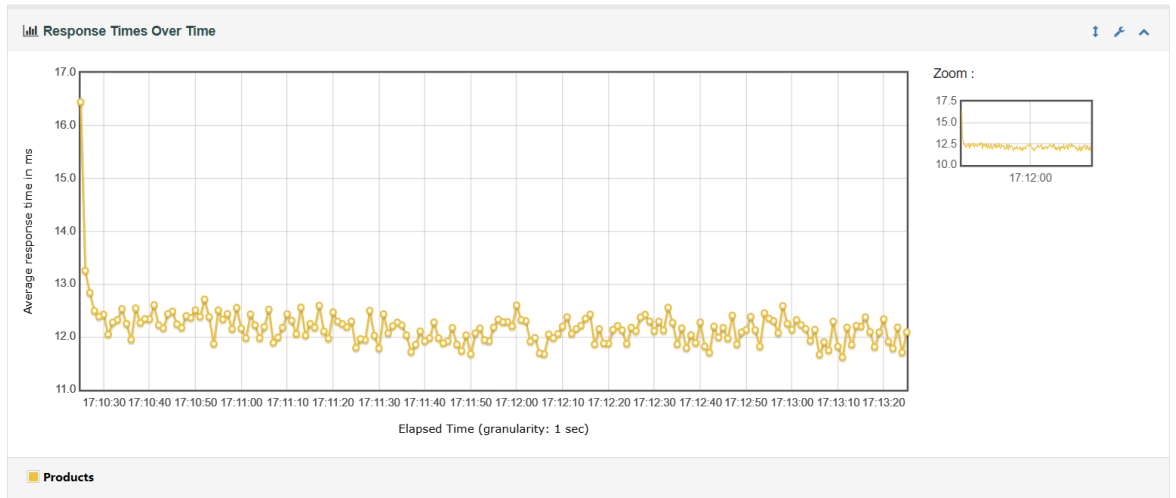


响应时间百分位数

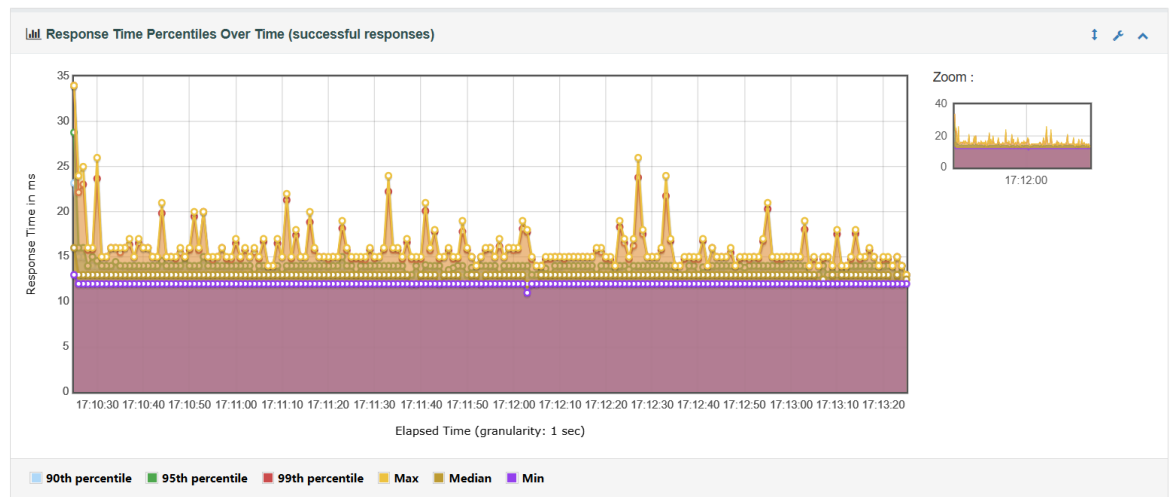


(4) 测试组 4

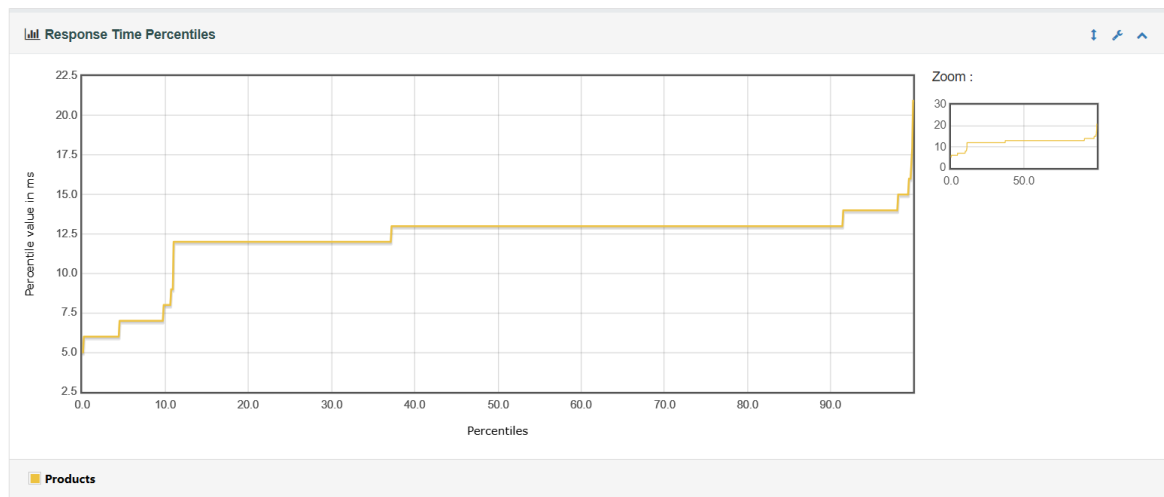
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

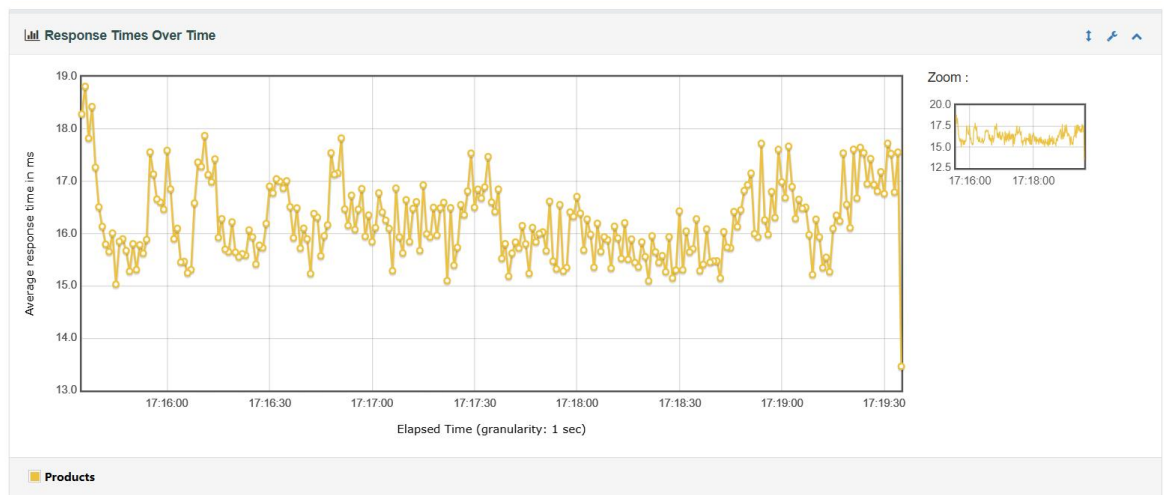


响应时间百分位数

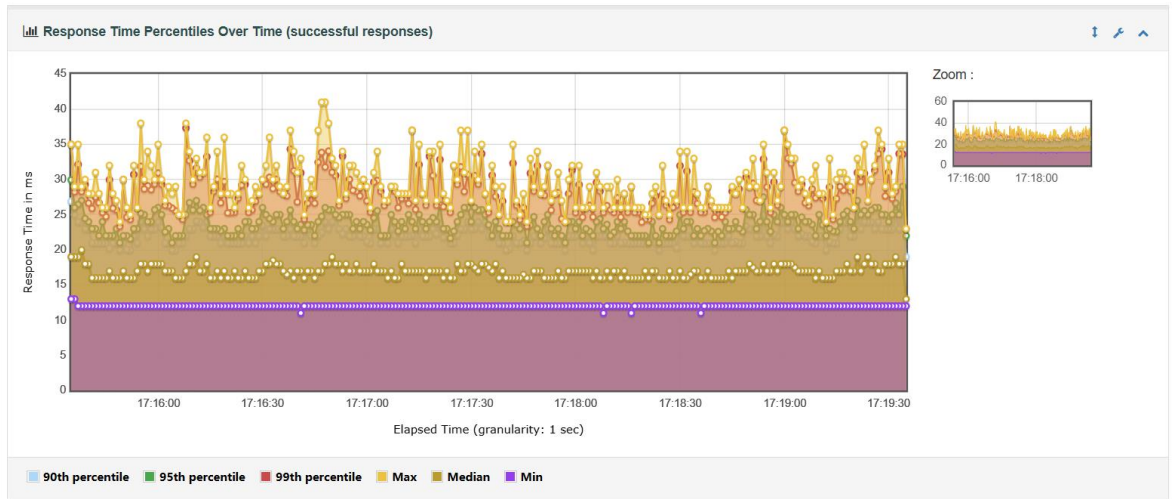


(5) 测试组 5

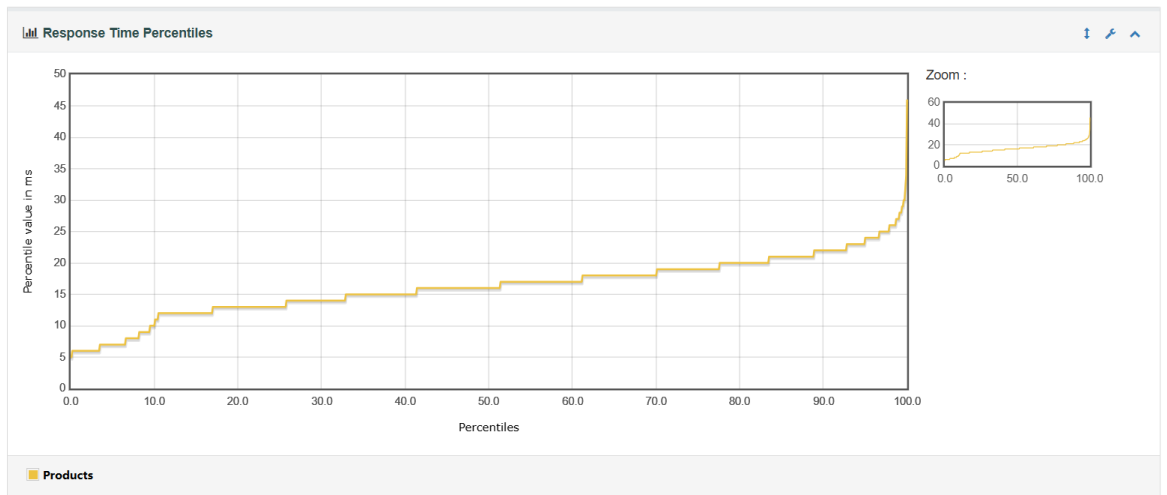
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）



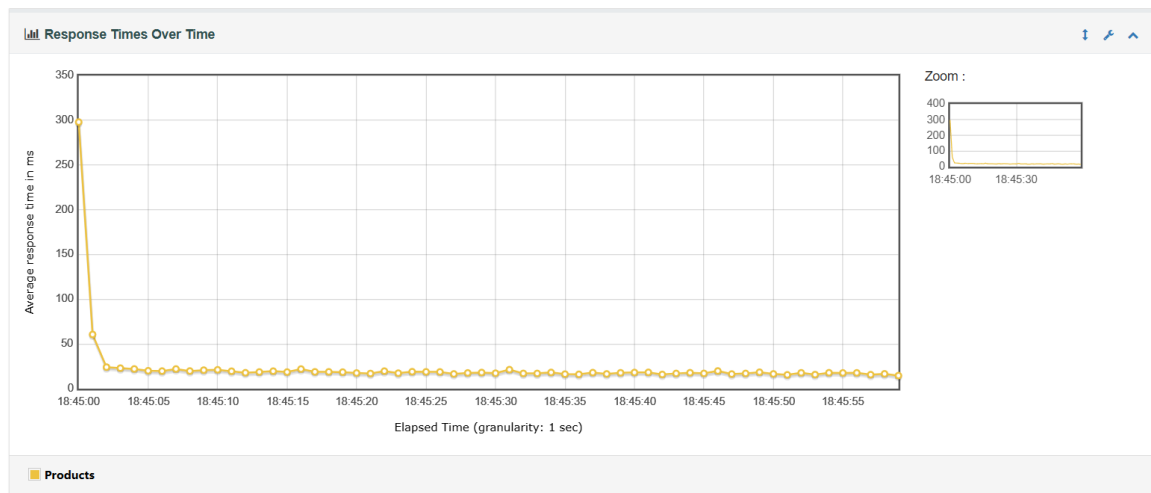
响应时间百分位数



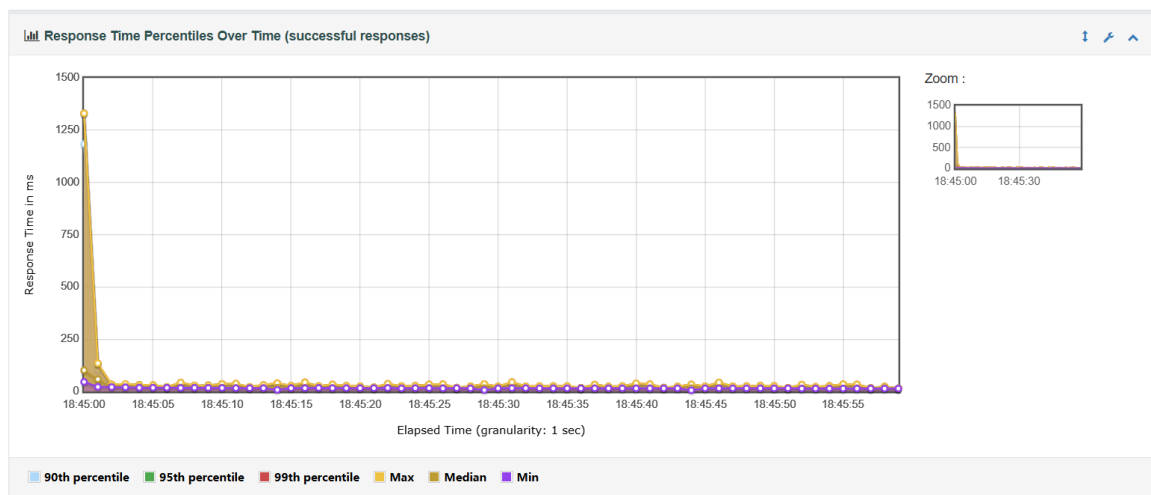
2. 基于 MyBatis 自动生成实现的数据库查询方案

(1) 测试组 1

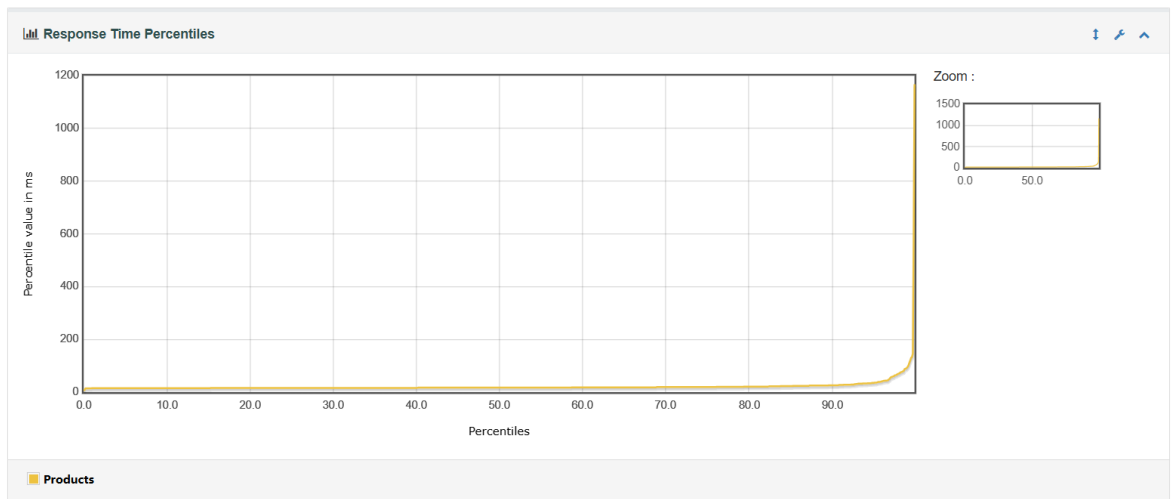
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

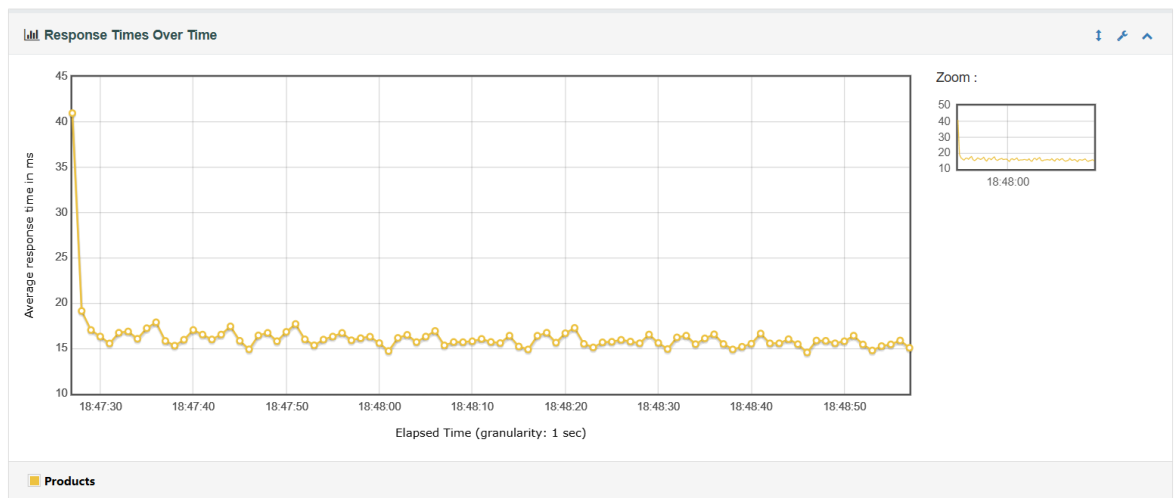


响应时间百分位数

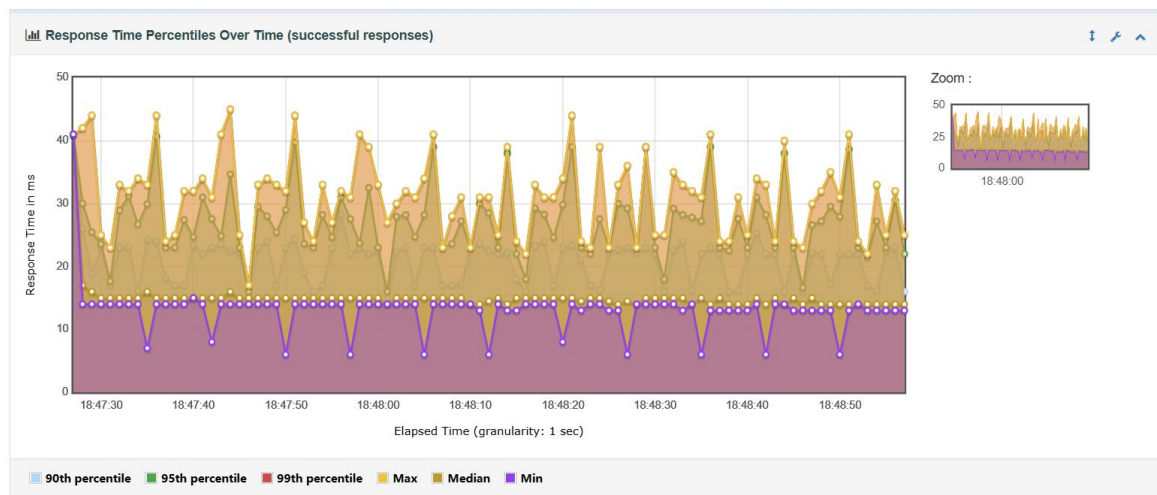


(2) 测试组 2

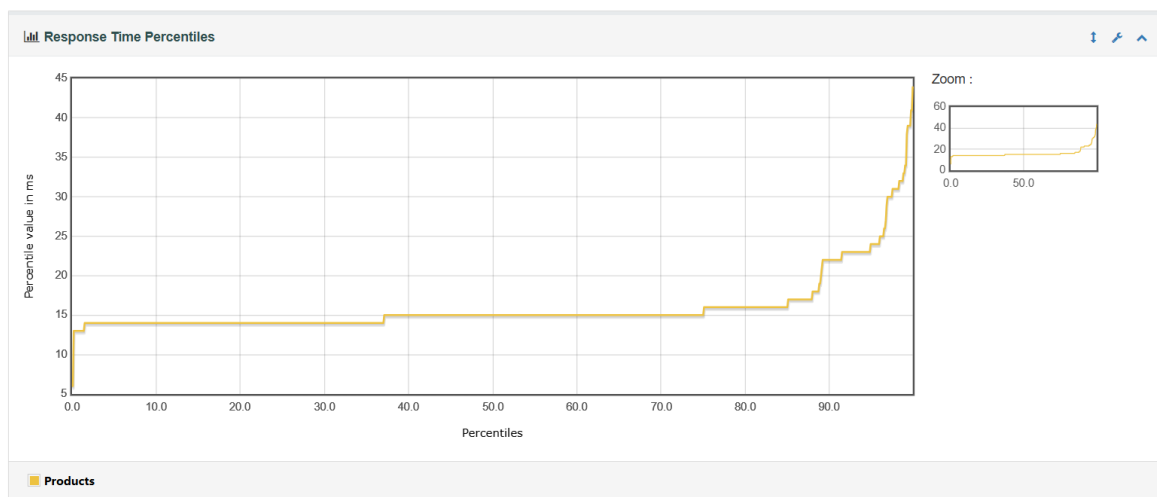
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

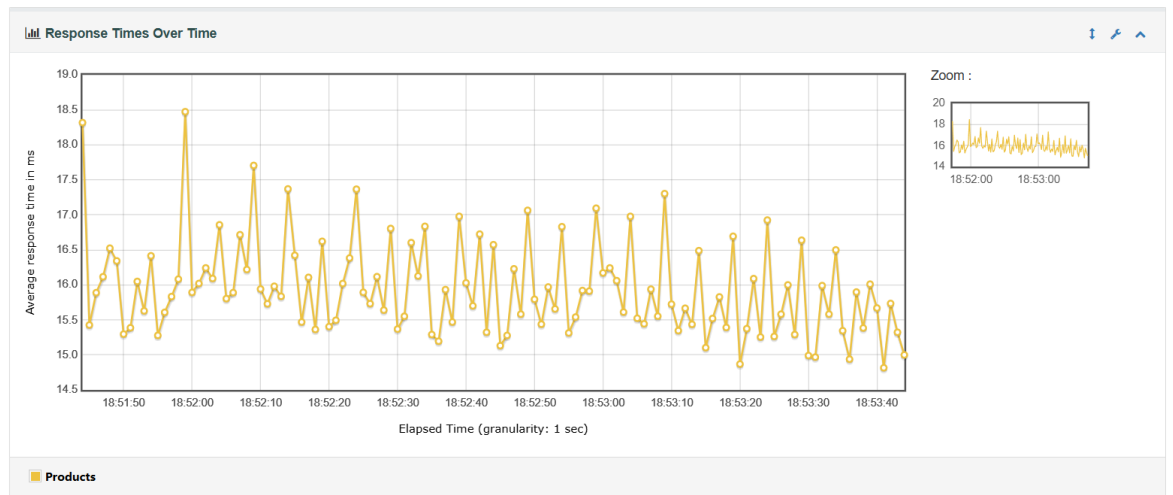


响应时间百分位数

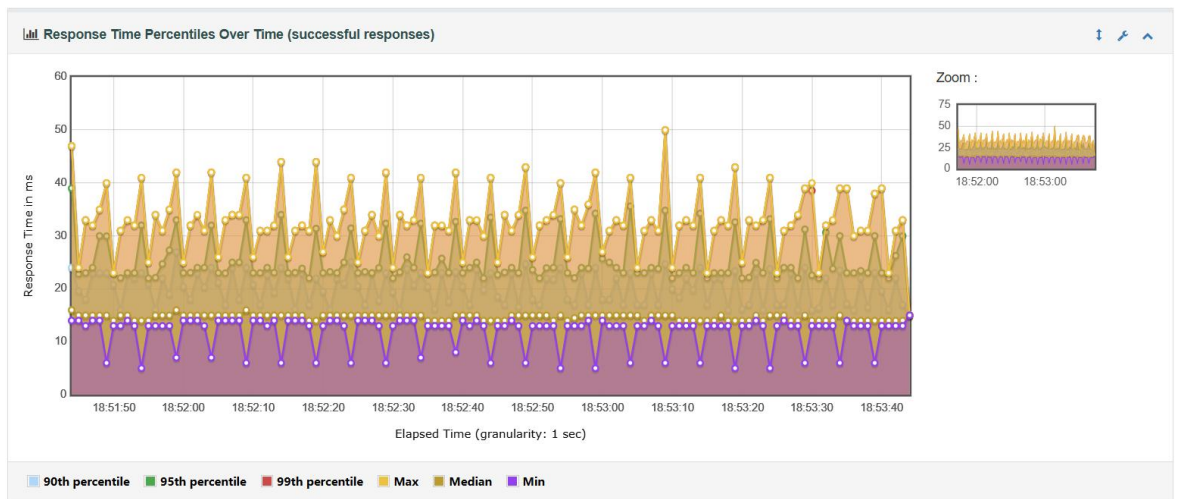


(3) 测试组 3

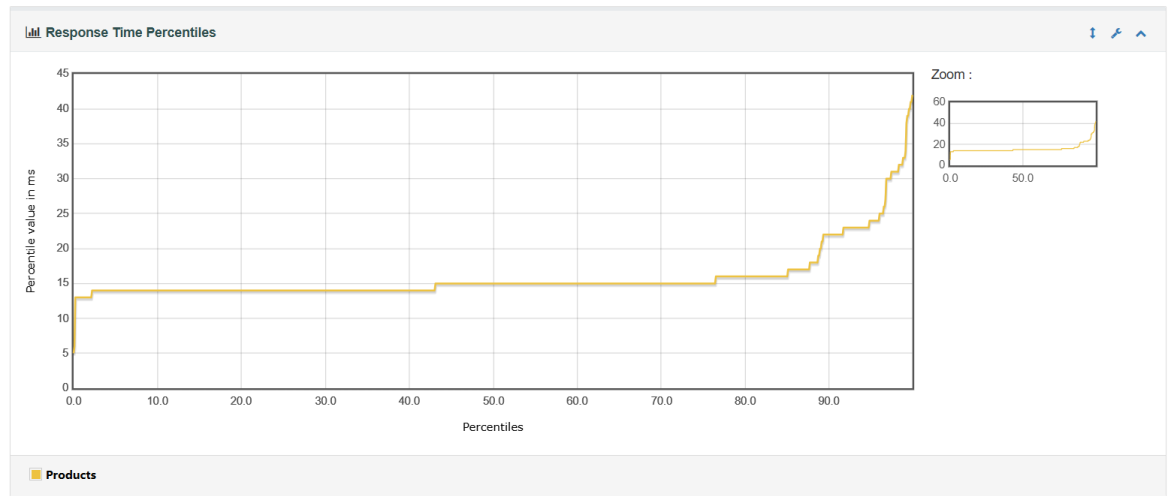
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

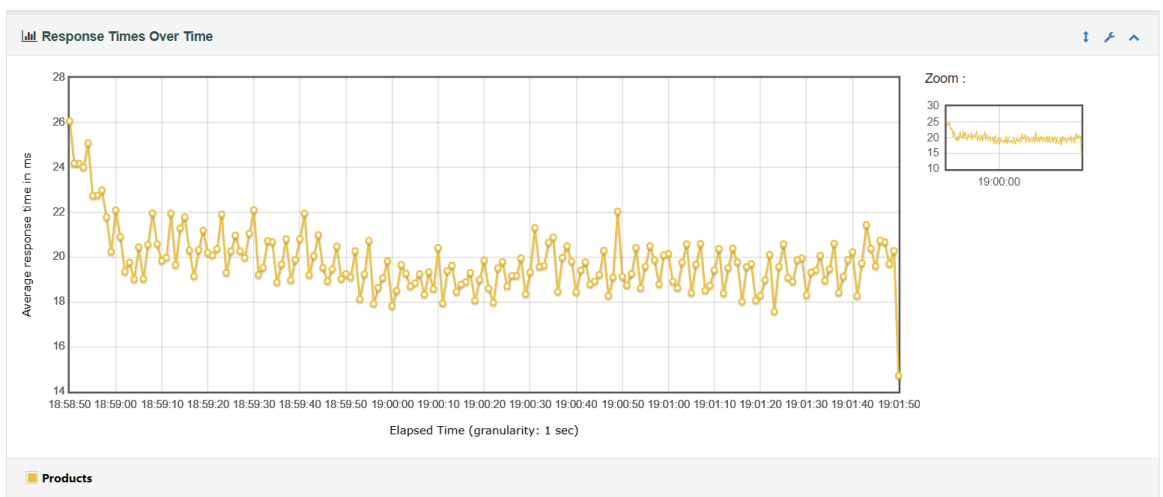


响应时间百分位数

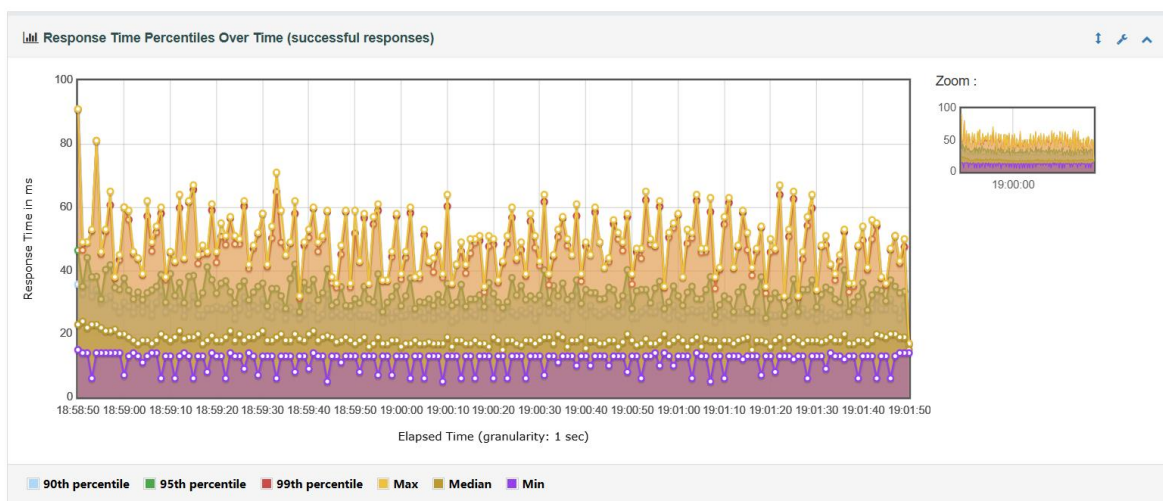


(4) 测试组 4

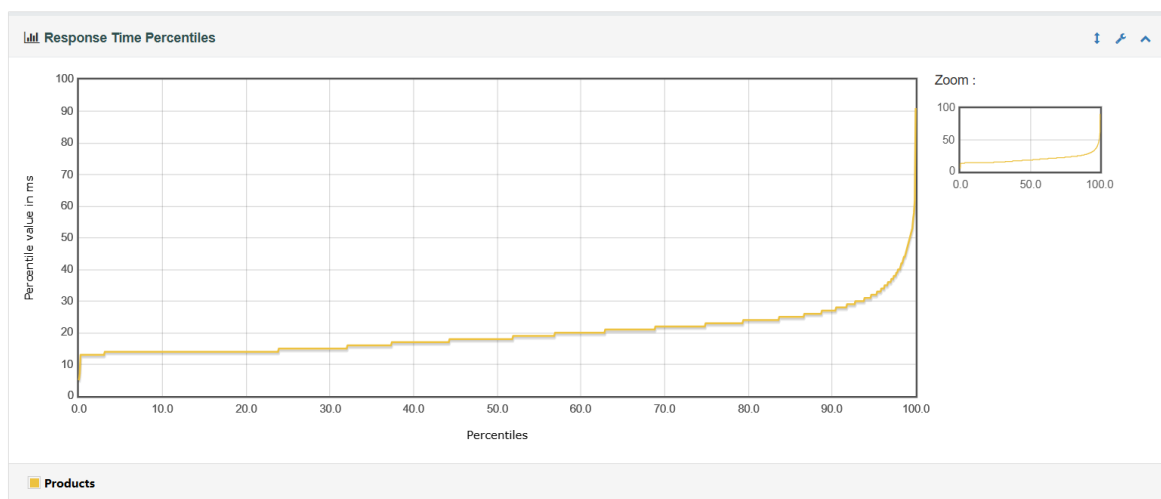
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）

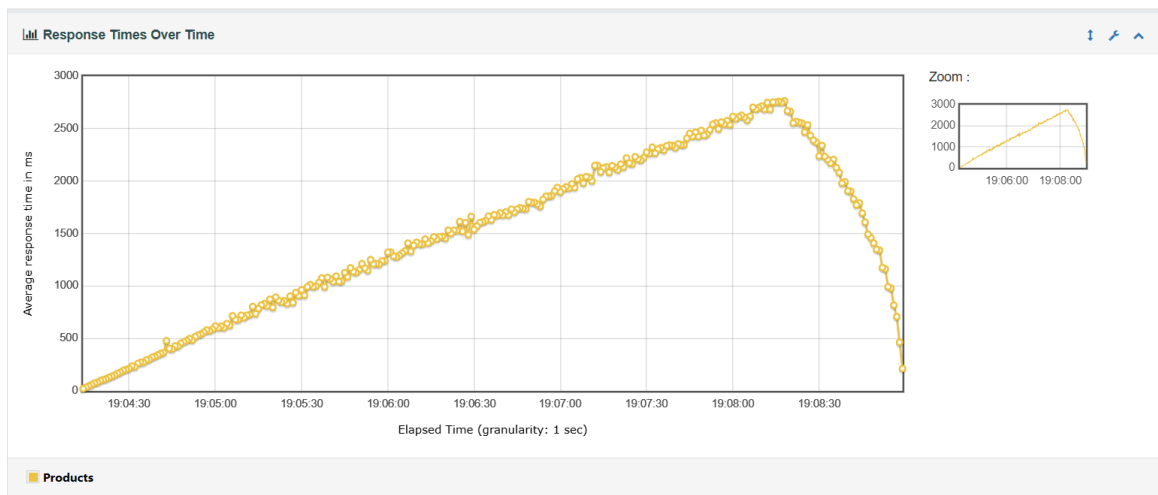


响应时间百分位数

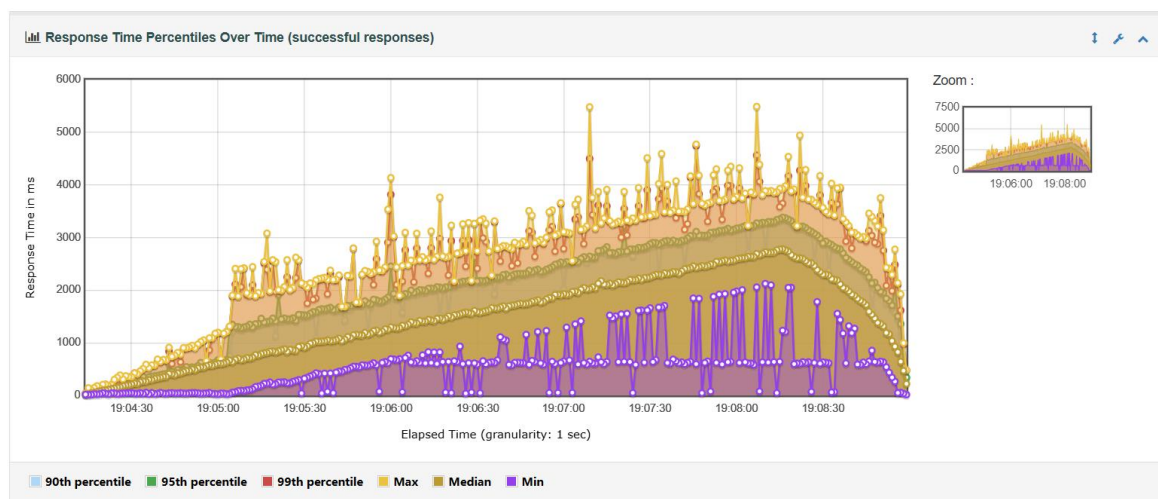


(5) 测试组 5

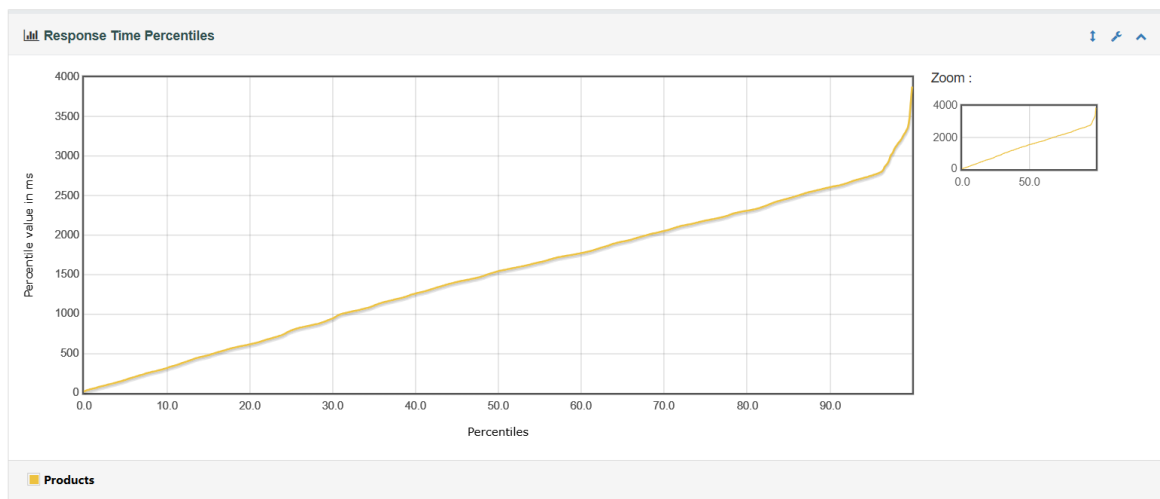
随时间变化的响应时间



随时间变化的响应时间百分比（成功的响应）



响应时间百分位数



通过 JMeter 测试数据，对比了 Spring Data JPA 和 MyBatis 三种关联查询方案的性能，具体分析结果如下：

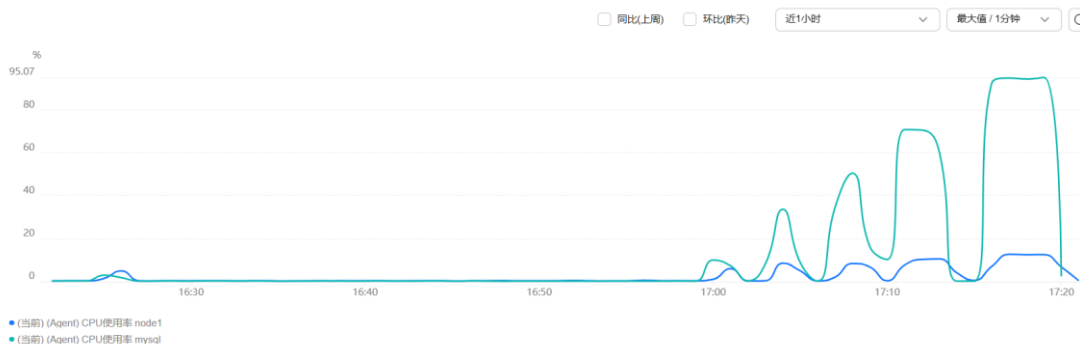
1. 在低并发环境下，Spring Data JPA 与 MyBatis 的 ResultMap 关联和 Dao 层关联方案性能相当，但略优于 MyBatis 自动生成查询方案。
2. 在中高并发环境下，Spring Data JPA 方案的响应时间增长较为平缓，表现出良好的可扩展性，而 MyBatis 的 ResultMap 和 Dao 层关联方案响应时间增加明显，出现瓶颈。

下面再来看华为云检测服务监测到的服务器负载情况：

1.JPA 实现的查询：

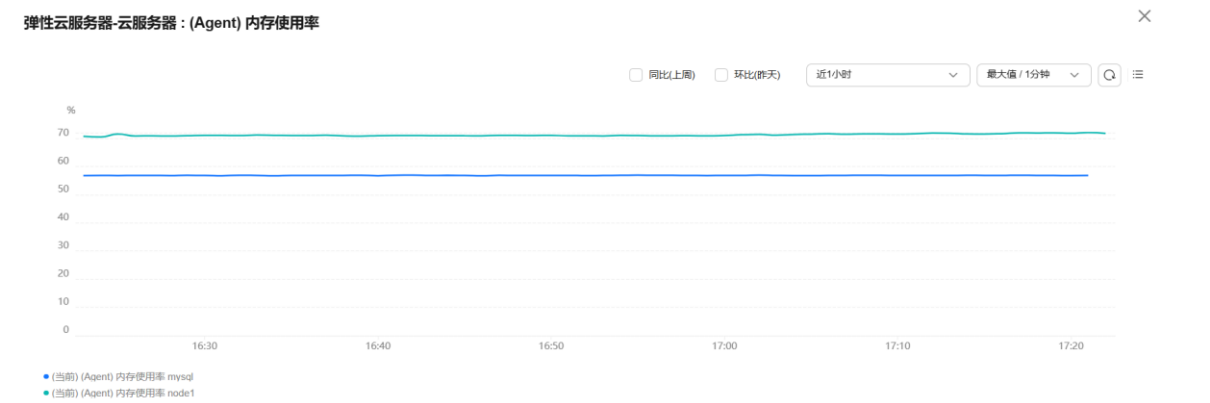
(1) CPU 负载情况

弹性云服务器-云服务器：(Agent) CPU使用率



可以看出，从低并发逐渐向高并发过渡的过程中，基于 JPA 实现的查询使得 CPU 使用率的增加较为平缓，具体的 CPU 使用率按照 10%—35%—50%—70%—95% 递增

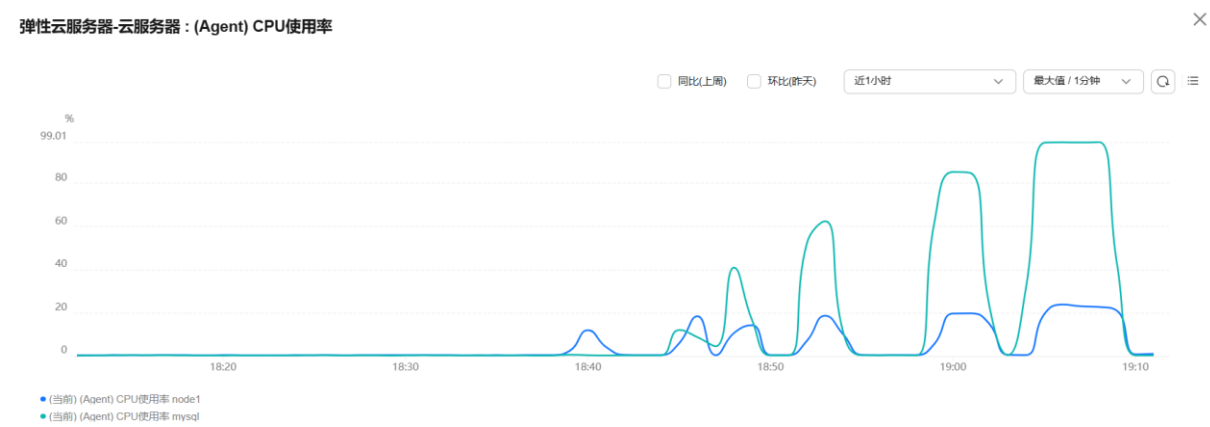
(2) 内存使用率



从图表中看出，基于 JPA 实现的查询方案整体内存使用率非常平缓，mysql 服务器的内存使用率一直维持在 50%—60% 之间。

2.MyBatis 自动生成的查询方案

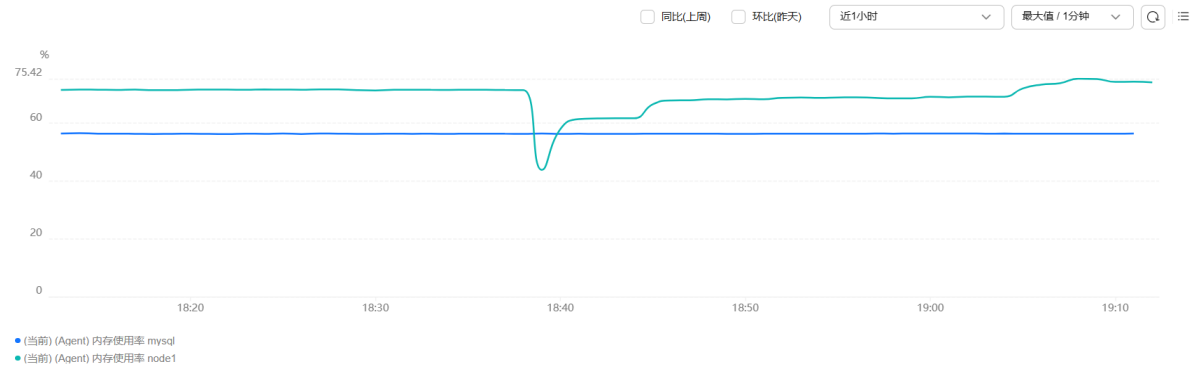
(1) CPU 负载情况



可以看出，这个方案实现的查询给服务器造成了较大的压力，同样并发水平的测试在这个方案里 CPU 的压力更大，并且部署服务的服务器也出现较大的波动。CPU 使用率：16%—40%—60%—82%—99%。

(2) 内存使用率

弹性云服务器-云服务器 : (Agent) 内存使用率



在测试时此方案给 mysql 造成的内存使用压力和 JPA 方案相当，但是部署服务的 node1 服务器出现较大的波动。

总结

实验结果表明，Spring Data JPA 在处理数据库关联查询时，尤其是在高并发环境下表现出较高的性能和稳定性。与 MyBatis 的 resultMap 关联和 Dao 层关联相比，Spring Data JPA 方案不仅减少了代码复杂度，还显著降低了服务器的资源占用。未来的研究可以进一步优化数据库连接池和缓存机制，以提升在极端高并发场景下的性能表现。

补充：jpa 调用 和 它对比方法 的 sql 语句输出 log

1. JPA 调用：

```
@Repository
public interface OnSaleRepository extends JpaRepository<OnSalePo, Long> {

    // 根据productId查询OnSale
    List<OnSalePo> findByProductId(Long productId);

}
```

```

@Repository
public interface ProductRepository extends JpaRepository<ProductPo, Long> {

    // 根据 name 字段查询 product
    ProductPo findByName(String name);

    // 根据 goodsId 查询 product
    List<ProductPo> findByGoodsId(Long goodsId);

}

```

The screenshot shows an IDE with the following components:

- Project View:** Shows the project structure for 'exp5-JPA' with subdirectories like 'idea', 'logs', and 'src'.
- Editor:** Displays the 'ProductRepository.java' file with the following code:


```

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

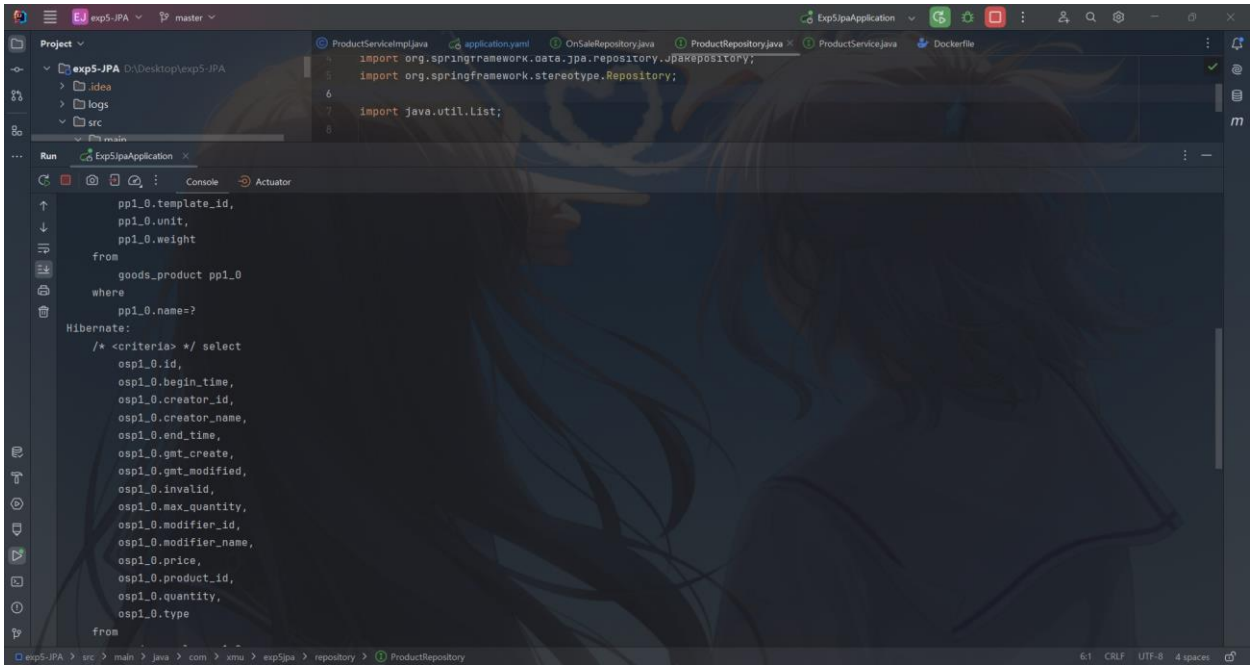
import java.util.List;

```
- Run View:** Shows the console output for 'Exp5JpaApplication'. The output includes:
 - Initialization of Spring DispatcherServlet 'dispatcherServlet'.
 - Initialization of Servlet 'dispatcherServlet'.
 - Completion of initialization in 1 ms.
 - A Hibernate SQL query:


```

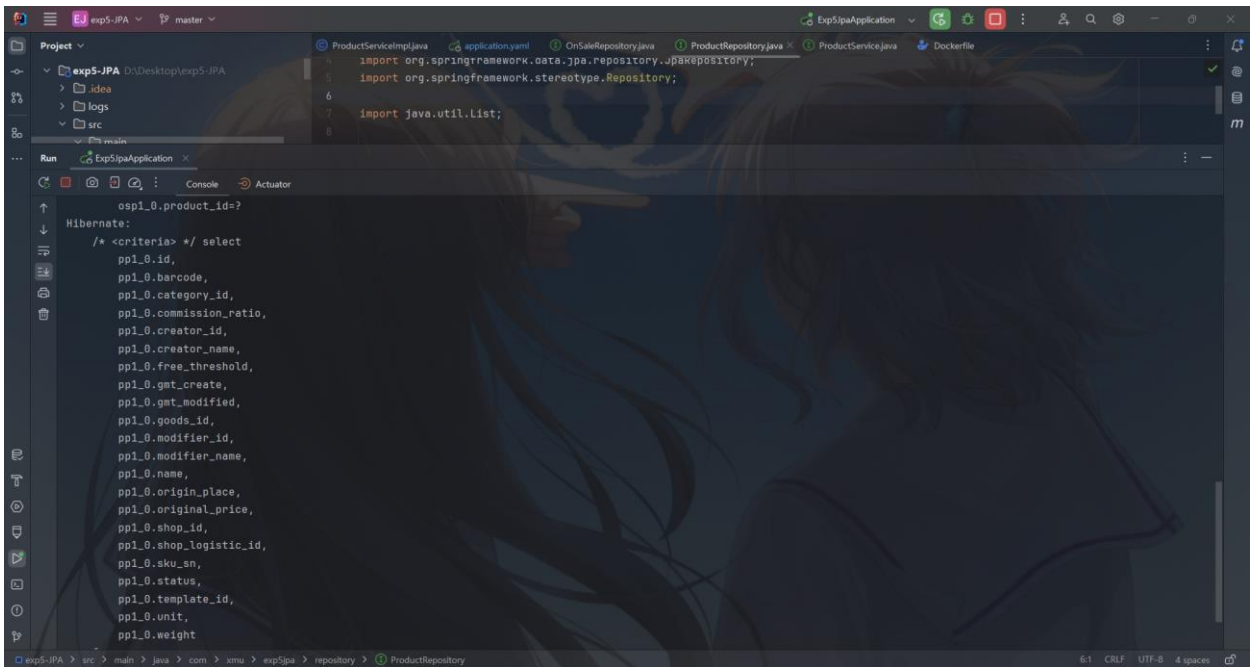
/* <criteria> */ select
  pp1.o.id,
  pp1.o.barcode,
  pp1.o.category_id,
  pp1.o.commission_ratio,
  pp1.o.creator_id,
  pp1.o.creator_name,
  pp1.o.free_threshold,
  pp1.o.gmt_create,
  pp1.o.gmt_modified,
  pp1.o.goods_id,
  pp1.o.modifier_id,
  pp1.o.modifier_name,
  pp1.o.name,
  pp1.o.origin_place,
  pp1.o.original_price,
  pp1.o.shop_id,
  pp1.o.shop_logistic_id,
  pp1.o.sku_sn,
  pp1.o.status,
  pp1.o.template_id,

```



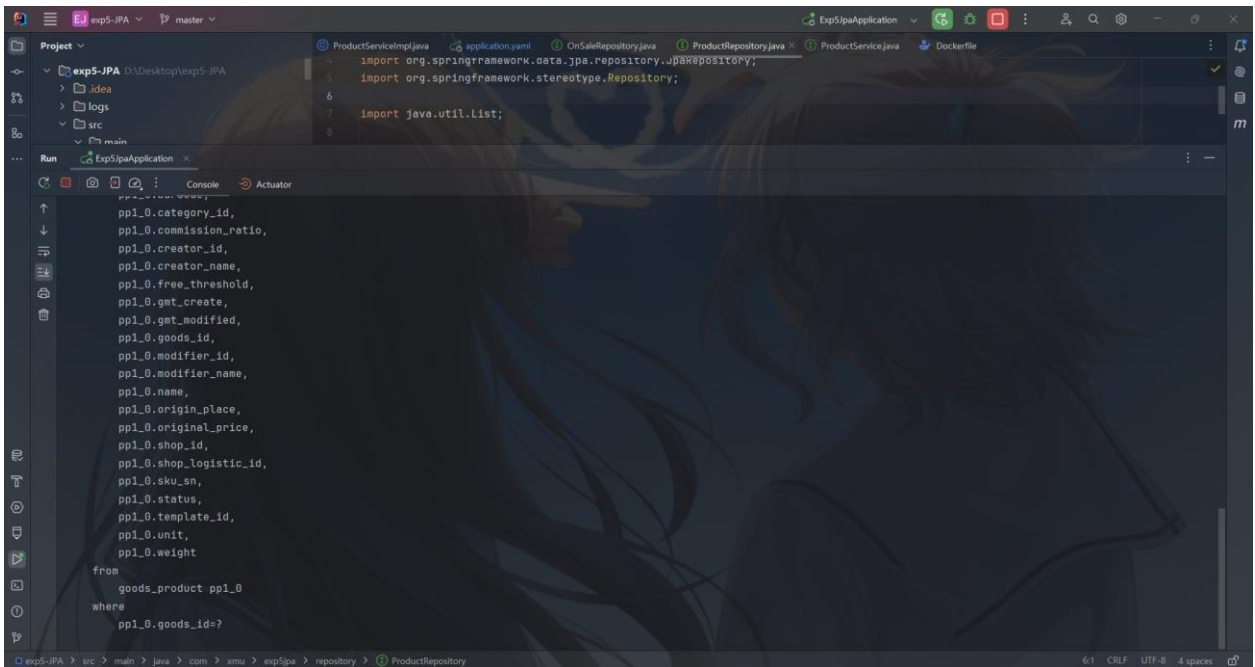
The screenshot shows an IDE with a project named 'exp5-JPA'. The console displays a SQL query generated by Hibernate. The query selects columns from the 'osp1_0' table, including 'osp1_0.id', 'osp1_0.begin_time', 'osp1_0.creator_id', 'osp1_0.creator_name', 'osp1_0.end_time', 'osp1_0.gmt_create', 'osp1_0.gmt_modified', 'osp1_0.invalid', 'osp1_0.max_quantity', 'osp1_0.modifier_id', 'osp1_0.modifier_name', 'osp1_0.price', 'osp1_0.product_id', 'osp1_0.quantity', and 'osp1_0.type'. The query is filtered by 'goods_product pp1_0' and 'pp1_0.name=?'. The IDE interface includes a Project view on the left, a Run tab at the bottom, and a code editor on the right showing imports for 'org.springframework.data.jpa.repository.JpaRepository' and 'java.util.List'.

```
pp1_0.template_id,  
pp1_0.unit,  
pp1_0.weight  
from  
  goods_product pp1_0  
where  
  pp1_0.name=?  
Hibernate:  
/* <criteria> */ select  
  osp1_0.id,  
  osp1_0.begin_time,  
  osp1_0.creator_id,  
  osp1_0.creator_name,  
  osp1_0.end_time,  
  osp1_0.gmt_create,  
  osp1_0.gmt_modified,  
  osp1_0.invalid,  
  osp1_0.max_quantity,  
  osp1_0.modifier_id,  
  osp1_0.modifier_name,  
  osp1_0.price,  
  osp1_0.product_id,  
  osp1_0.quantity,  
  osp1_0.type  
from
```

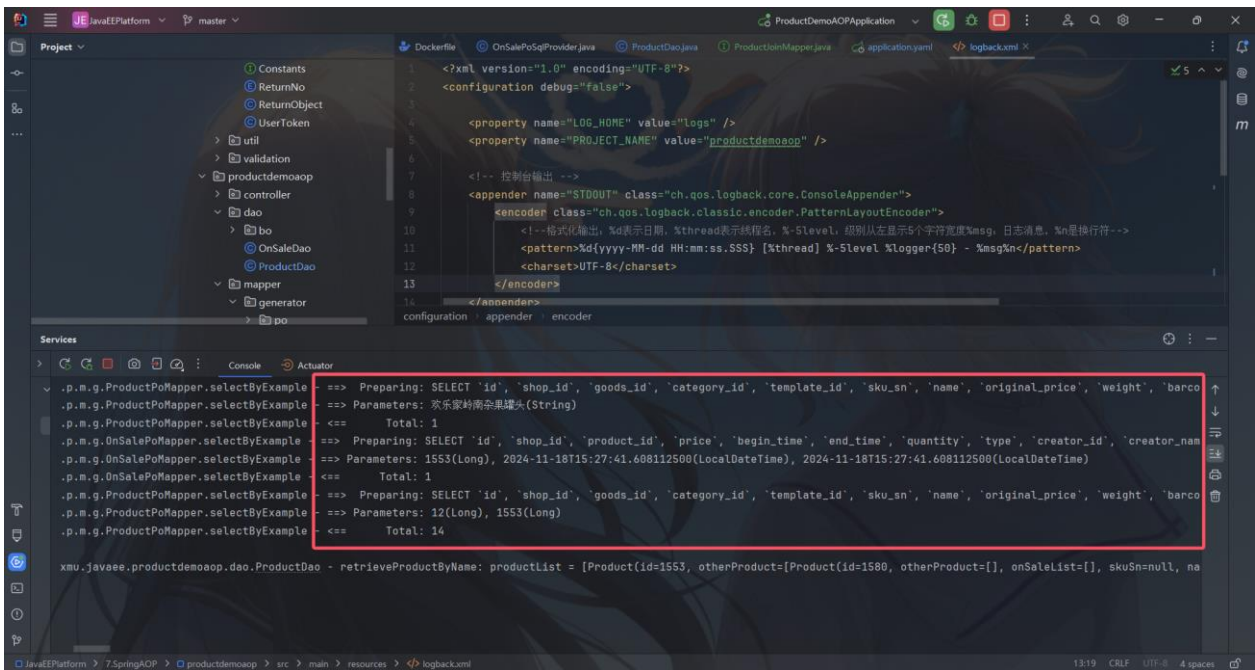


The screenshot shows the same IDE with a different SQL query in the console. This query selects columns from the 'pp1_0' table, including 'pp1_0.id', 'pp1_0.barcode', 'pp1_0.category_id', 'pp1_0.commission_ratio', 'pp1_0.creator_id', 'pp1_0.creator_name', 'pp1_0.free_threshold', 'pp1_0.gmt_create', 'pp1_0.gmt_modified', 'pp1_0.goods_id', 'pp1_0.modifier_id', 'pp1_0.modifier_name', 'pp1_0.name', 'pp1_0.origin_place', 'pp1_0.original_price', 'pp1_0.shop_id', 'pp1_0.shop_logistic_id', 'pp1_0.sku_sn', 'pp1_0.status', 'pp1_0.template_id', 'pp1_0.unit', and 'pp1_0.weight'. The query is filtered by 'osp1_0.product_id=?'. The IDE interface is consistent with the previous screenshot, showing the Project view, Run tab, and code editor with the same imports.

```
osp1_0.product_id=?  
Hibernate:  
/* <criteria> */ select  
  pp1_0.id,  
  pp1_0.barcode,  
  pp1_0.category_id,  
  pp1_0.commission_ratio,  
  pp1_0.creator_id,  
  pp1_0.creator_name,  
  pp1_0.free_threshold,  
  pp1_0.gmt_create,  
  pp1_0.gmt_modified,  
  pp1_0.goods_id,  
  pp1_0.modifier_id,  
  pp1_0.modifier_name,  
  pp1_0.name,  
  pp1_0.origin_place,  
  pp1_0.original_price,  
  pp1_0.shop_id,  
  pp1_0.shop_logistic_id,  
  pp1_0.sku_sn,  
  pp1_0.status,  
  pp1_0.template_id,  
  pp1_0.unit,  
  pp1_0.weight
```



2.对比方法（MyBatis 自动生成）



References:

- [1] 实验代码仓库: <https://github.com/YUKIPEdia/XMU-JavaEE-exp5-JPA.git>
- [2] JMeter 下载与使用文档: https://jmeter.apache.org/download_jmeter.cgi
- [3] Spring Data JPA 官方文档: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

[4] MyBatis 官方文档: <https://mybatis.org/mybatis-3/>