单体应用与微服务应用的效率对比

摘　要:

关键词:

问题描述：

实验设计：

1. 实验环境

- 服务器 A：Ubuntu 18.04，2 核 2G 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 4G 内存，运行 Docker、Maven、Git，部署 Product 应用与 Nacos 服务。
- 服务器 D：Ubuntu 18.04，2 核 2G 内存，安装 JMeter 5.6.3 用于测试。
- 服务器 E：Ubuntu 18.04，2 核 4G 内存，运行 Docker、Maven、Git，部署 Shop 应用与 Rocketmq-broker。
- 服务器 F：Ubuntu 18.04，2 核 4G 内存，运行 Docker、Git，部署 Redis 服务。
- 服务器 G：Ubuntu 18.04，2 核 2G 内存，运行 Docker、Git，部署 Rocketmq 服务。
- 服务器 H：Ubuntu 18.04，2 核 2G 内存，运行 Docker、Git，部署 mongo 服务。
- 服务器 I：Ubuntu 18.04，2 核 2G 内存，运行 Docker、Git，部署 mongo 服务。
- 服务器 J：Ubuntu 18.04，2 核 2G 内存，运行 Docker、Git，部署 mongo 服务。

2. 实验步骤

(1) 实现了一个基于微服务体系结构查询数据库的 RESTful API 接口

a. GET /products/1551 通过商品 id 查询**产品完整信息或不完整信息**

(2) 设计并运行了 JMeter 测试，分别对 API 的**不同参数**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 服务器与 OOMALL-node2 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

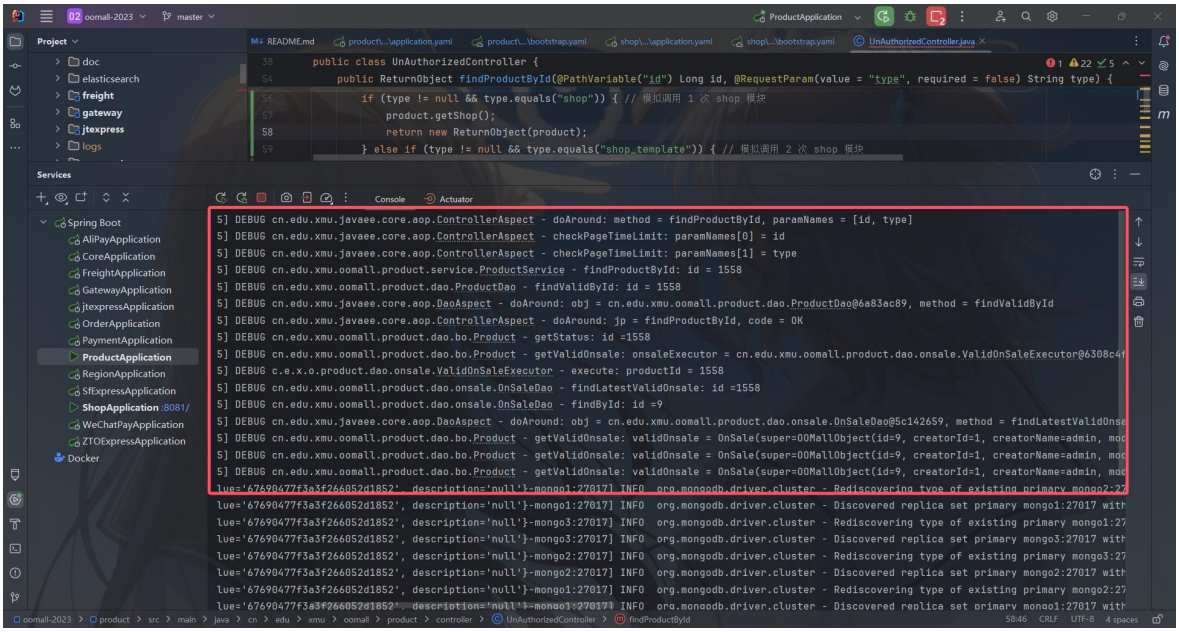
(5) 分别对三种情况：product 模块不调用 shop 模块、product 模块调用一次 shop 模块、product 模块调用两次 shop 模块进行记录，并对比两种方法速度差异与服务器负载

结果分析与讨论：

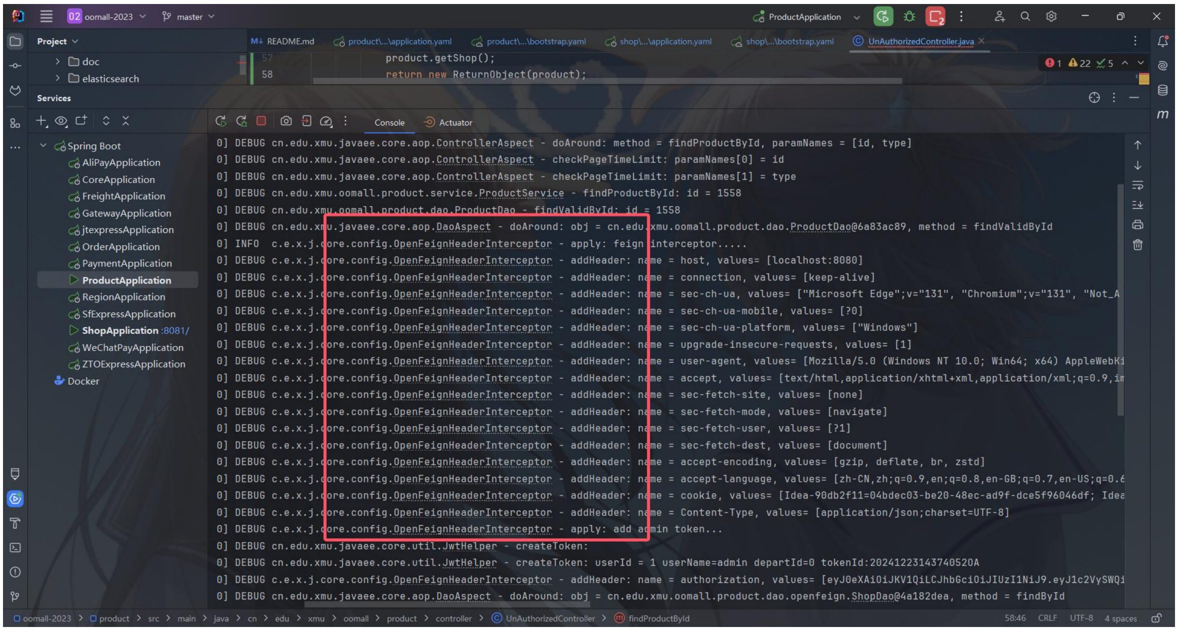
1.根据实验要求，需要实现 product 模块对 shop 模块的调用，具体分为：0 次调用、1 次调用、2 次调用，我们需要对三种情况进行分析。首先对原来的代码进行修改，在原来 controller 的基础上添加一个 type 参数，用来区分查询的类别。当没有传入 type 时，表示调用 0 次 shop 模块；传入的 type 为 shop 时，表示调用 1 次 shop 模块；传入的 type 为 shop_template 时，表示调用 2 次 shop 模块。在调用 2 次 shop 模块的代码中，直接使用了 ProductVo 的构造方法，因为此构造方法已经调用了 product.getShop()和 product.getTemplate()，因此不需要再进行额外调用。其他两个方案均对调用方式进行模拟。

```java
/**
 * 获得商品信息
 * @author Ming Qiu
 * <p>
 * date: 2022-12-04 19:19
 * @param id
 * @return
 */
@GetMapping("/products/{id}")
@Transactional(propagation = Propagation.REQUIRED)
public ReturnObject findProductById(@PathVariable("id") Long id, @RequestParam(value = "type", required = false) String type) {
    Product product = this.productService.findProductById(PLATFORM, id);
    if (type != null && type.equals("shop")) { // 模拟调用 1 次 shop 模块
        product.getShop();
        return new ReturnObject(product);
    } else if (type != null && type.equals("shop_template")) { // 模拟调用 2 次 shop 模块
        return new ReturnObject(new ProductVo(product));
    }
    return new ReturnObject(product); // 模拟调用 0 次 shop 模块
}
```

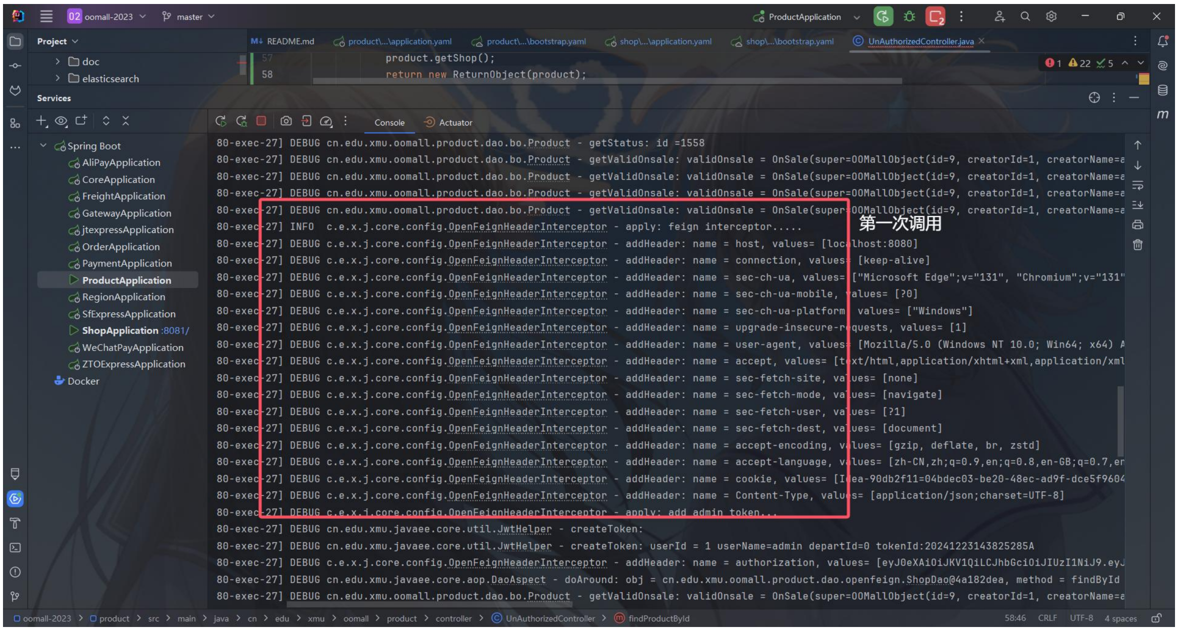
调用 0 次 shop 模块日志输出如下：

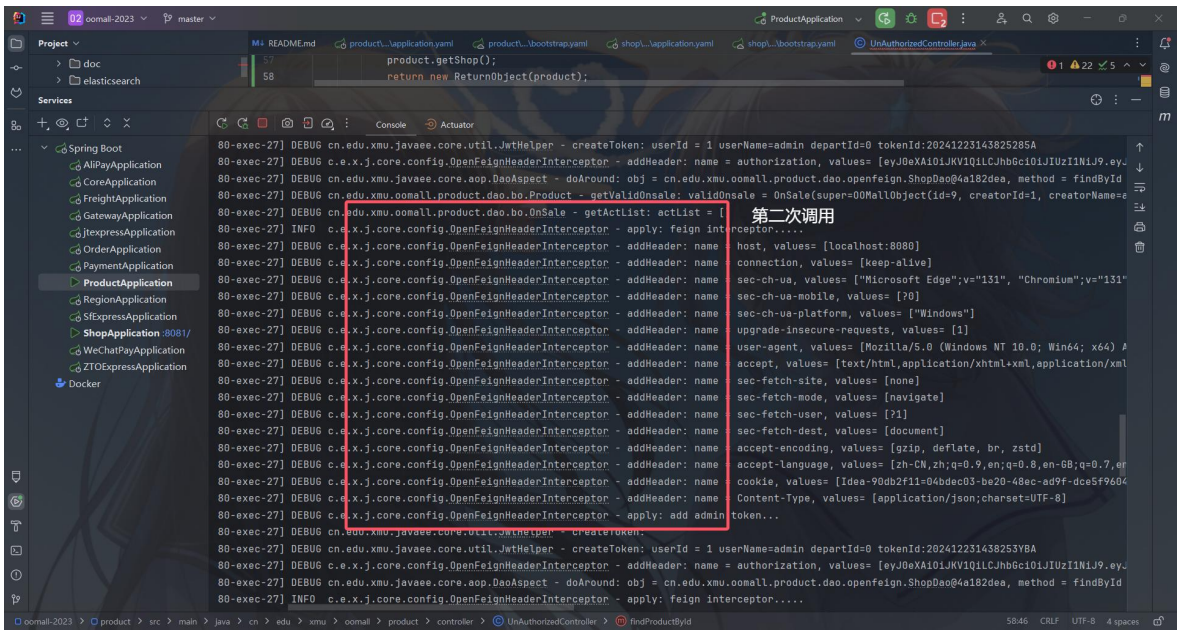


调用 1 次 shop 模块日志输出如下：

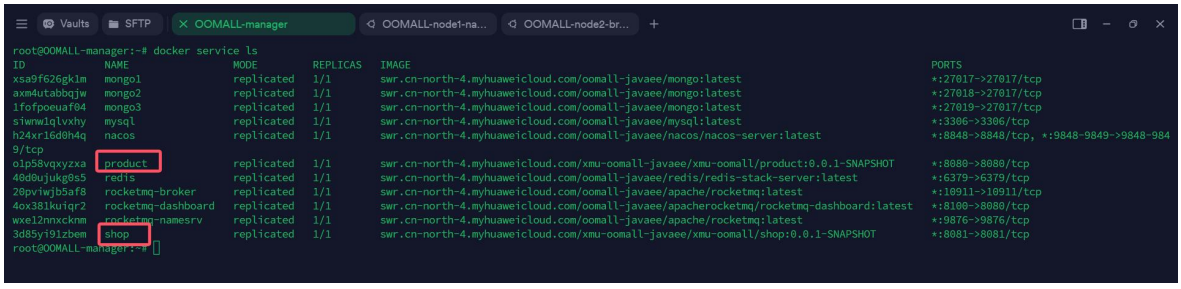


调用 2 次 shop 模块日志输出如下：

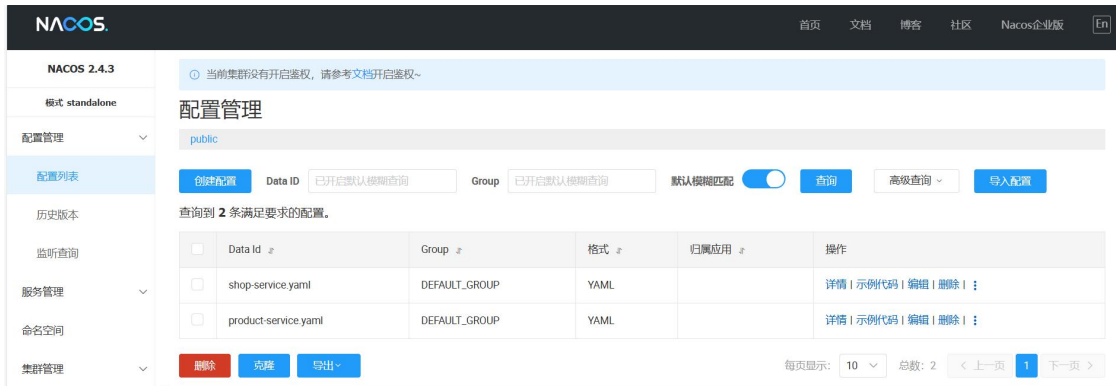




2.本地测试均通过，接下来将两个模块部署到服务器上，其中，product 服务部署在 nacos 节点上，shop 服务部署在 rocketmq 节点上，在管理机上运行结果如下，product 运行在 8080 端口，shop 运行在 8081 端口

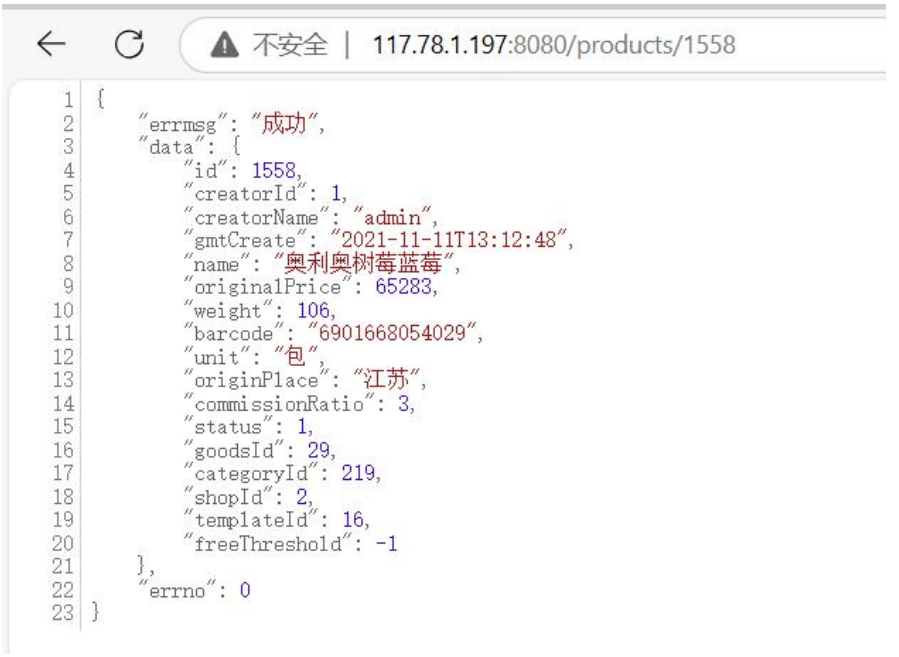


然后在 nacos 中配置两个服务：




NACOS.
首页
文档
博客
社区
Nacos企业版
En
NACOS 2.4.3
模式 standalone
配置管理
服务管理
服务列表
订阅者列表
命名空间
集群管理
当前集群没有开启鉴权，请参考文档开启鉴权~
服务列表
public
创建服务
服务名称
请输入服务名称
分组名称
请输入分组名称
隐藏空服务
查询
服务名
分组名称
集群数目
实例数
健康实例数
触发保护阈值
操作
product-service
DEFAULT_GROUP
1
1
1
false
详情 | 示例代码 | 订阅者 | 删除
shop-service
DEFAULT_GROUP
1
1
1
false
详情 | 示例代码 | 订阅者 | 删除
每页显示: 10
总数: 2
上一页
1
下一页

服务器上测试结果如下：


不安全 | 117.78.1.197:8080/products/1558
{
"errmsg": "成功",
"data": {
"id": 1558,
"creatorId": 1,
"creatorName": "admin",
"gmtCreate": "2021-11-11T13:12:48",
"name": "奥利奥树莓蓝莓",
"originalPrice": 65283,
"weight": 106,
"barcode": "6901668054029",
"unit": "包",
"originPlace": "江苏",
"commissionRatio": 3,
"status": 1,
"goodsId": 29,
"categoryId": 219,
"shopId": 2,
"templateId": 16,
"freeThreshold": -1
},
"errno": 0
}

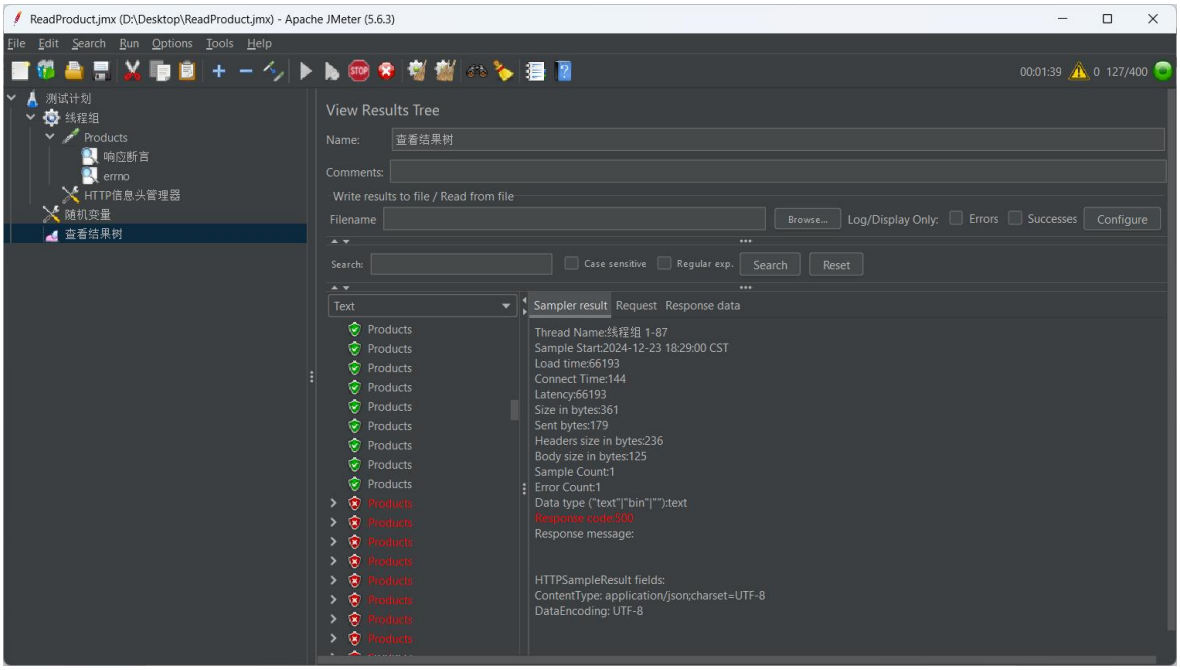
不安全 | 117.78.1.197:8080/products/1558?type=shop


```
{
    "errmsg": "成功",
    "data": {
        "id": 1558,
        "creatorId": 1,
        "creatorName": "admin",
        "gmtCreate": "2021-11-11T13:12:48",
        "name": "奥利奥树莓蓝莓",
        "originalPrice": 65283,
        "weight": 106,
        "barcode": "6901668054029",
        "unit": "包",
        "originPlace": "江苏",
        "commissionRatio": 3,
        "status": 1,
        "goodsId": 29,
        "categoryId": 219,
        "shopId": 2,
        "templateId": 16,
        "freeThreshold": -1
    },
    "errno": 0
}
```


不安全 | 117.78.1.197:8080/products/1558?type=shop_template


```
{
    "errmsg": "成功",
    "data": {
        "id": 1558,
        "name": "奥利奥树莓蓝莓",
        "status": 1,
        "originalPrice": 65283,
        "weight": 106,
        "barcode": "6901668054029",
        "unit": "包",
        "originPlace": "江苏",
        "otherProduct": [
            {
                "id": 1558,
                "name": "奥利奥树莓蓝莓",
                "price": 6985,
                "status": 1,
                "quantity": 78
            },
            {
                "id": 3015,
                "name": "味好美椒盐20",
                "price": 37684,
                "status": 1,
                "quantity": 95
            },
            {
                "id": 3162,
                "name": "咖喱羊肉串(320)",
                "price": 3851,
                "status": 1,
                "quantity": 31
            },
            {
                "id": 3301,
                "name": "上海风味火腿(360)",
                "price": 986,
                "status": 1,
                "quantity": 89
            },
            {
                "id": 4598,
                "name": "180美洁2粒纯木浆纸",
                "price": 4619,
                "status": 1,
                "quantity": 27
            },
            {
                "id": 5045,
                "name": "110金苑大骨面",
                "price": 16991,
                "status": 1,
                "quantity": 45
            }
        ],
        "price": 6985
```

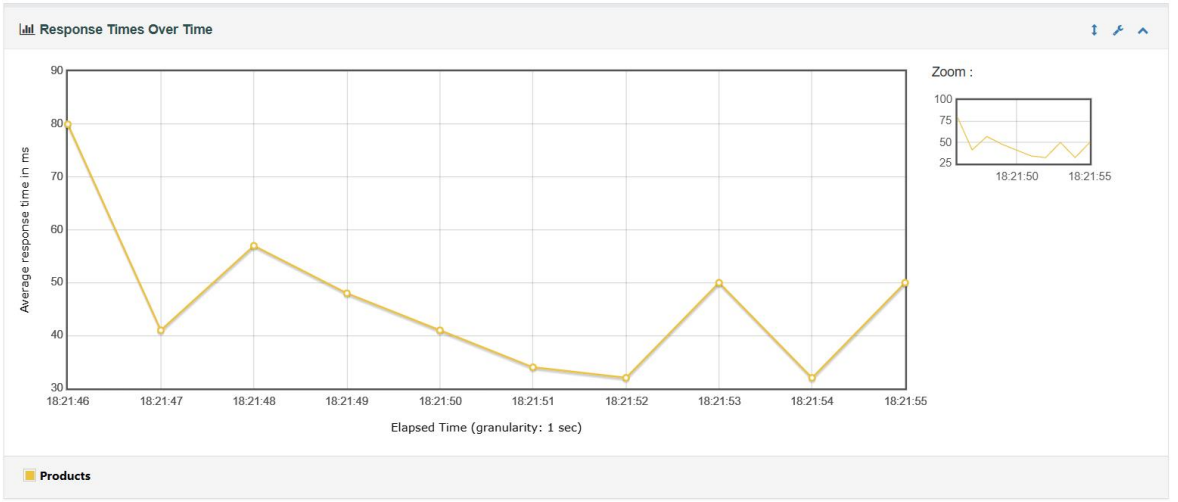
3.两个服务在服务器上已经部署完成，接下来使用 JMeter 进行模拟并发测试，由于在本机测试时发现 JMeter 模拟并发条件时，线程阻塞非常严重，故此处只采取较小规模的并发量



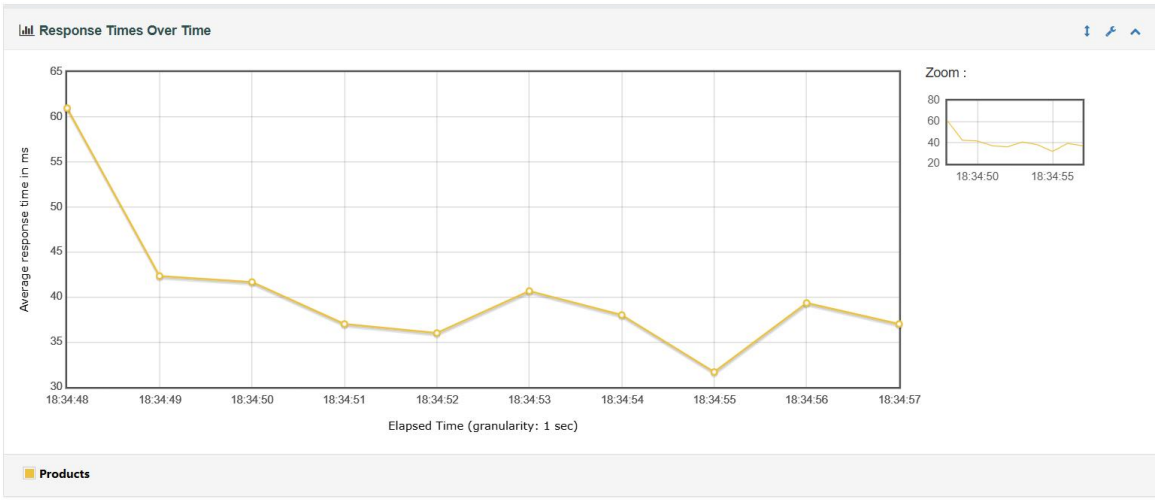
可以看出并发访问时出现请求连接超时的情况，微服务下的并发能力严重不足。

一、不调用 shop 模块

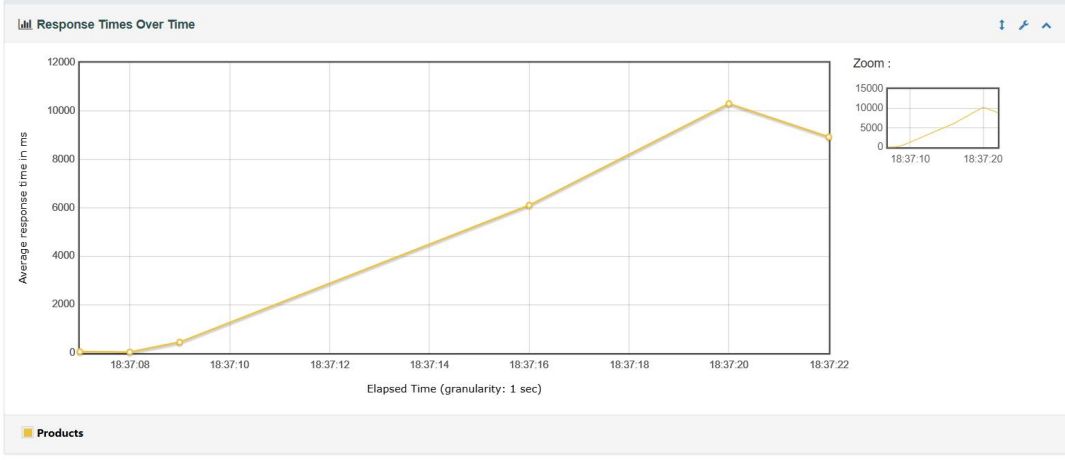
(1)thread-rampTime-loop 为 10-10-1 时，并没有出现线程阻塞问题，响应时间稳定在 40ms 左右

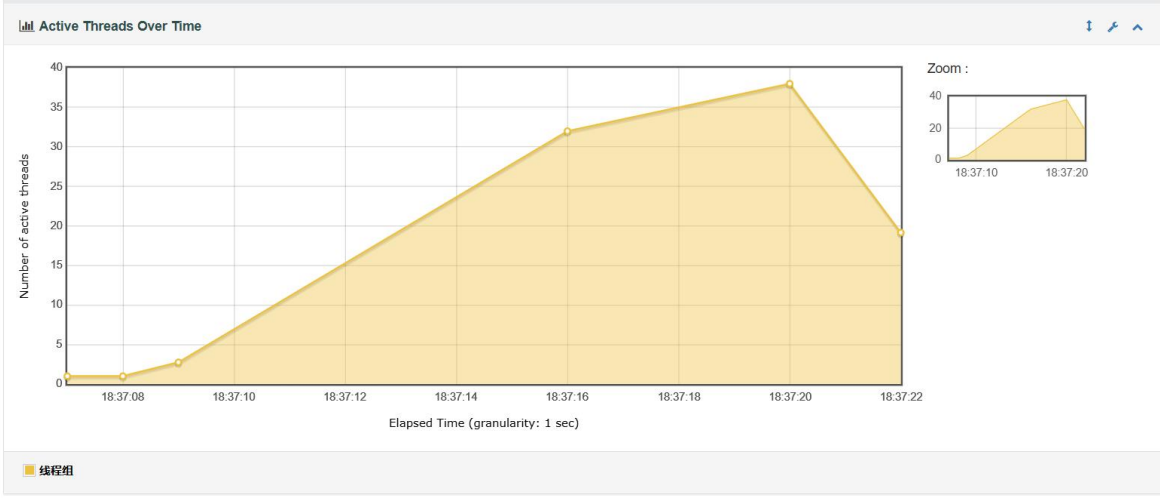


(2)thread-rampTime-loop 为 30-10-1 时，也没有出现线程阻塞问题，响应时间稳定在 40ms 左右



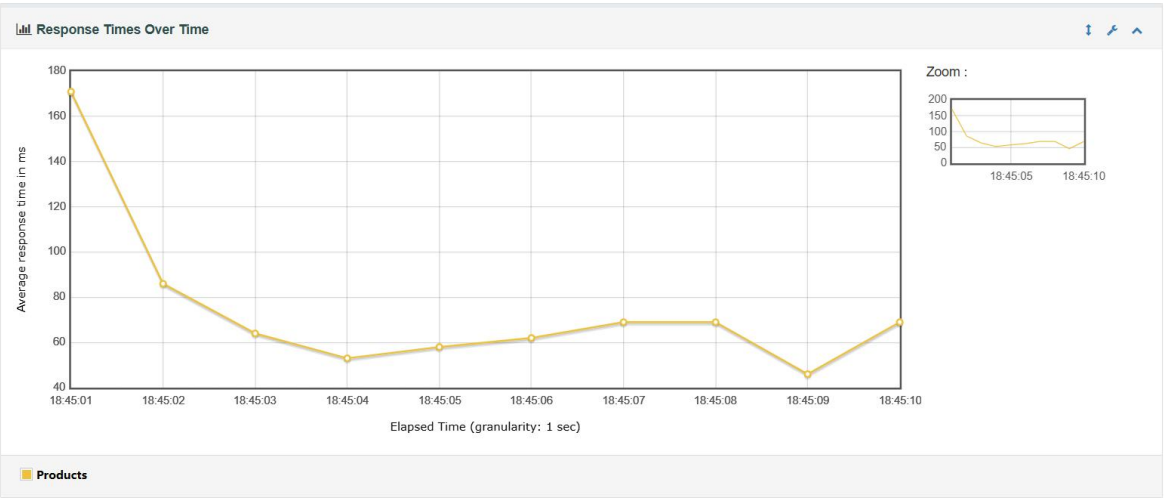
(3)thread-rampTime-loop 为 50-10-1 时，出现了非常严重的线程阻塞问题，最高时有超过 35 条线程被阻塞



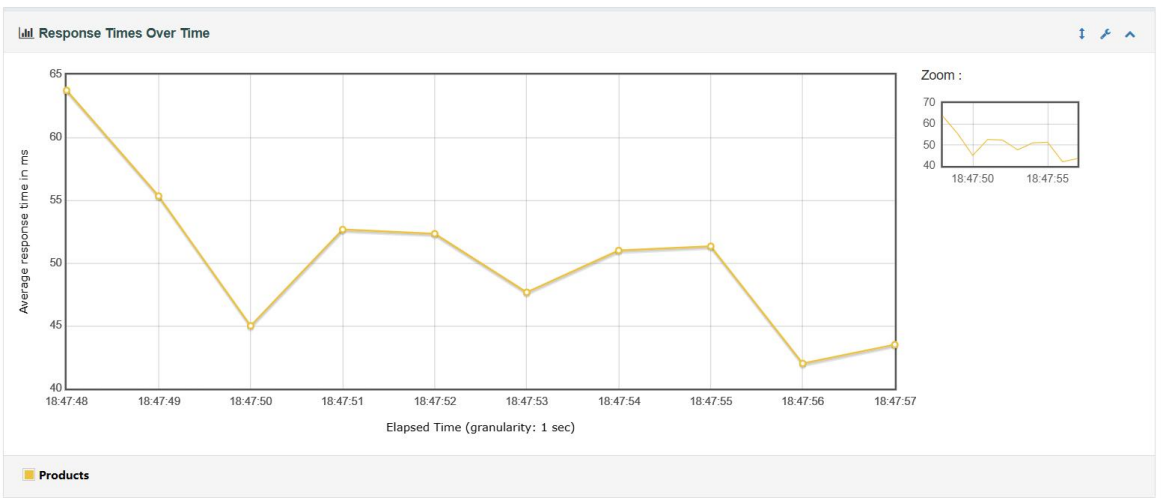


二、调用一次 shop 模块

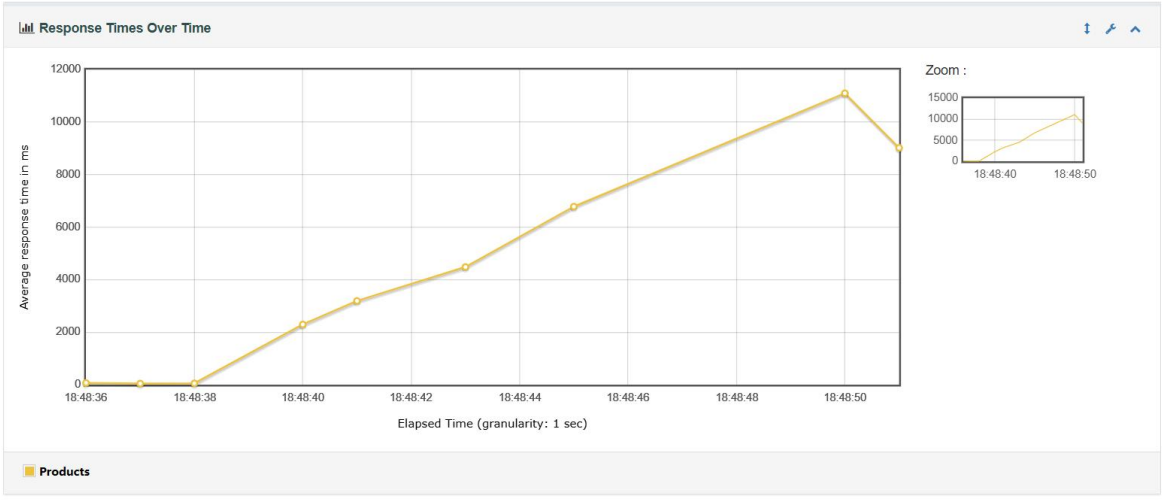
(1)thread-rampTime-loop 为 10-10-1 时，响应时间稳定在 60ms 左右，对比不调用 shop 模块的 40ms，响应速度明显变慢，但此时并没有出现线程阻塞

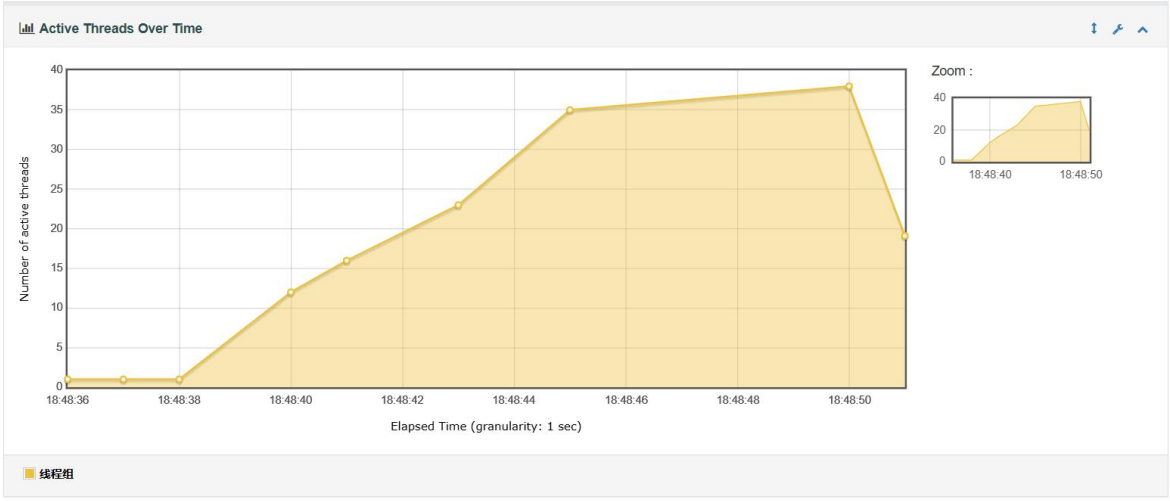


(2)thread-rampTime-loop 为 30-10-1 时，响应时间稳定在 50ms 左右，对比不调用 shop 模块的 40ms，响应速度明显变慢，但此时并没有出现线程阻塞



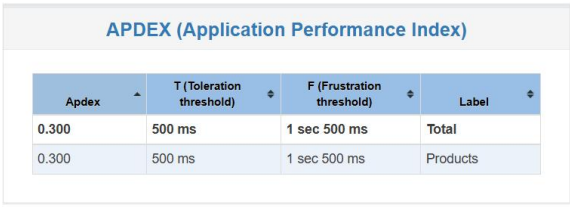
(3)thread-rampTime-loop 为 50-10-1 时，出现了非常严重的线程阻塞问题，相较于不调用 shop 模块，阻塞问题更严重

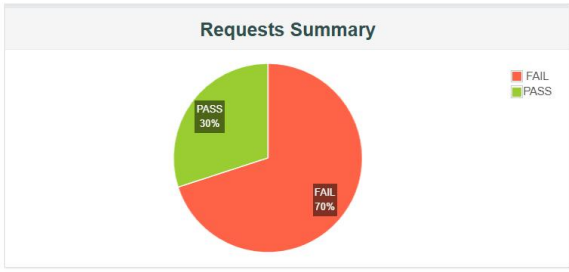


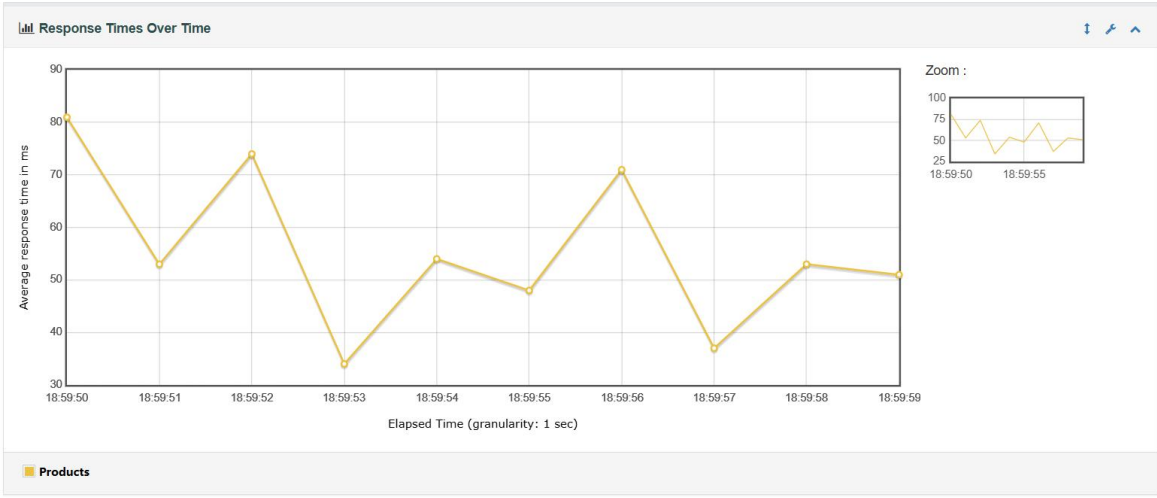


三、调用两次 shop 模块

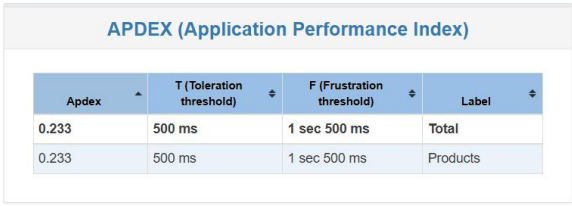
(1)thread-rampTime-loop 为 10-10-1 时，响应时间和调用一次相当，但是已经出现了大多数的线程运行失败的情况，**应该是由于数据库中某些 product 的 templateId 为空导致的访问失败**

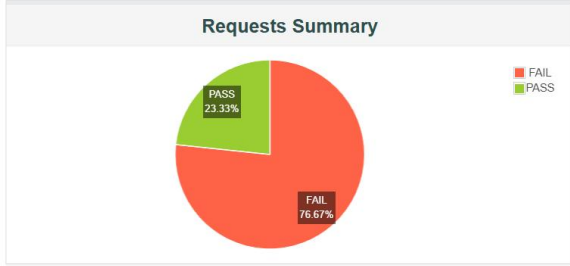


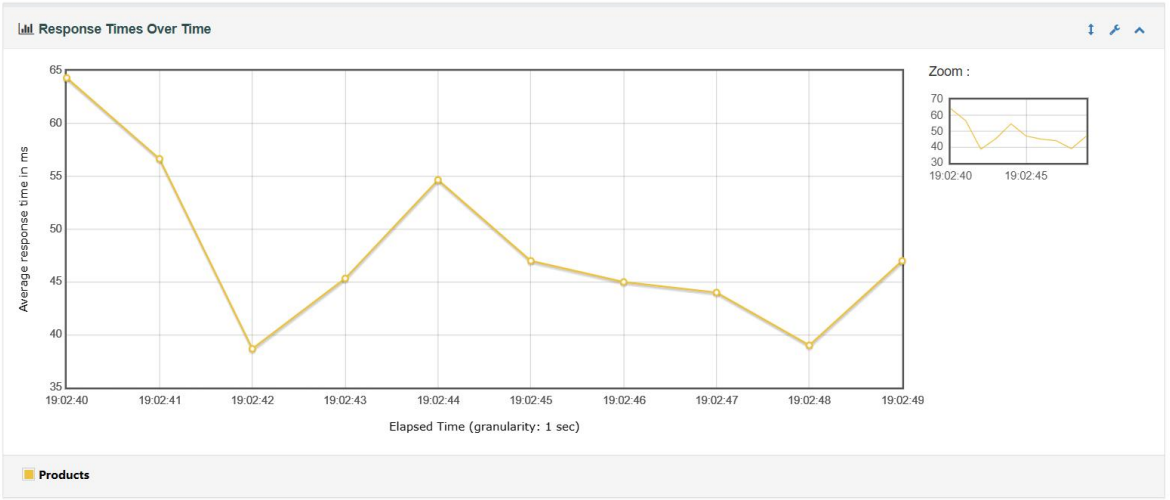




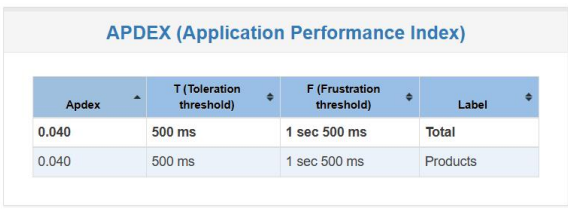
(2)thread-rampTime-loop 为 30-10-1 时，响应时间和调用一次相当，但是已经出现了大多数的线程运行失败的情况，**应该是由于数据库中某些 product 的 templateId 为空导致的访问失败**

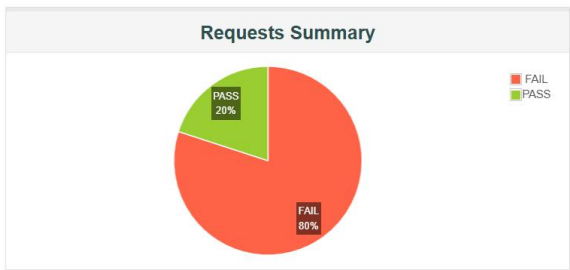


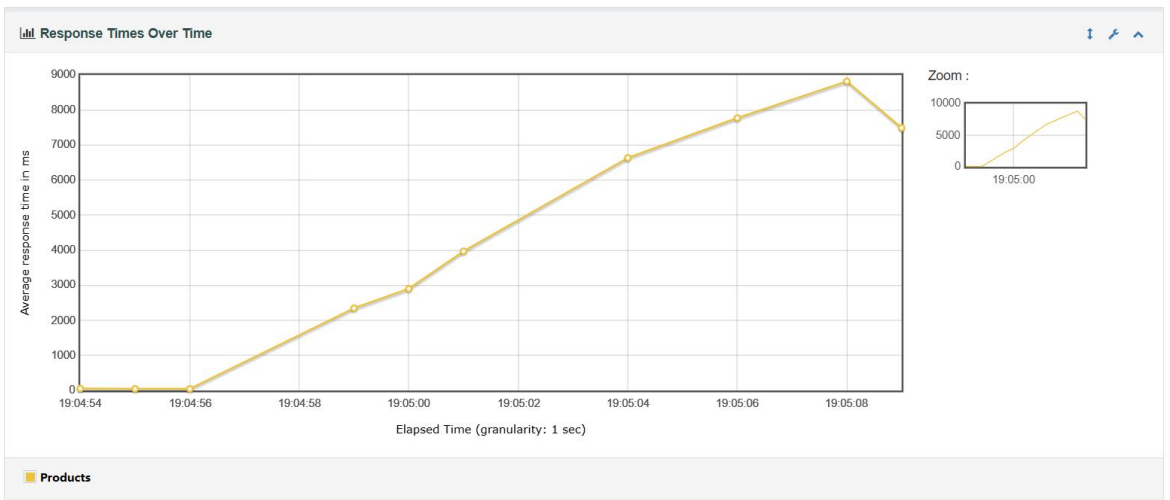




(3)thread-rampTime-loop 为 30-10-1 时，响应时间和调用一次相当，但是已经出现了大多数的线程运行失败的情况，由于某些 product 的 templateId 为空，所以出现大多数线程访问失败，**响应时间相较于调用一次 shop 模块更短，应该是 templateId 为空，导致此次访问直接返回结果**







4.由于并没有形成较高的并发度，服务器的 CPU 使用率并不是很高。本次实验结束后仍不太清楚为什么线程阻塞情况会如此严重