

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。

```

/**
 * 获得商品信息
 * @author Ming Qiu
 * <p>
 * date: 2022-12-04 19:19
 * @param id
 * @return
 */
@GetMapping("/{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：

```

5) DEBUG cn.edu.xmu.javaee.core.aop.ControllerAspect - doAround: method = findProductById, paramNames = [id, type]
5) DEBUG cn.edu.xmu.javaee.core.aop.ControllerAspect - checkPageTimeLimit: paramNames[0] = id
5) DEBUG cn.edu.xmu.javaee.core.aop.ControllerAspect - checkPageTimeLimit: paramNames[1] = type
5) DEBUG cn.edu.xmu.oomall.product.service.ProductService - findProductById: id = 1558
5) DEBUG cn.edu.xmu.oomall.product.dao.ProductDao - findValidById: id = 1558
5) DEBUG cn.edu.xmu.javaee.core.aop.DaoAspect - doAround: obj = cn.edu.xmu.oomall.product.dao.ProductDao@a83ac89, method = findValidById
5) DEBUG cn.edu.xmu.oomall.product.dao.bo.Product - getStatus: id = 1558
5) DEBUG cn.edu.xmu.oomall.product.dao.bo.Product - getValidOnSale: onSaleExecutor = cn.edu.xmu.oomall.product.dao.onSale.ValidOnSaleExecutor@a6308c4f
5) DEBUG cn.edu.xmu.oomall.product.dao.onSale.ValidOnSaleExecutor - execute: productId = 1558
5) DEBUG cn.edu.xmu.oomall.product.dao.onSale.OnSaleDao - findLatestValidOnSale: id = 1558
5) DEBUG cn.edu.xmu.oomall.product.dao.onSale.OnSaleDao - findById: id = 9
5) DEBUG cn.edu.xmu.javaee.core.aop.DaoAspect - doAround: obj = cn.edu.xmu.oomall.product.dao.onSale.OnSaleDao@5c142659, method = findLatestValidOnSale
5) DEBUG cn.edu.xmu.oomall.product.dao.bo.Product - getValidOnSale: validOnSale = OnSale(super=OOMallObject{id=9, creatorId=1, creatorName=admin, mod
5) DEBUG cn.edu.xmu.oomall.product.dao.bo.Product - getValidOnSale: validOnSale = OnSale(super=OOMallObject{id=9, creatorId=1, creatorName=admin, mod
5) DEBUG cn.edu.xmu.oomall.product.dao.bo.Product - getValidOnSale: validOnSale = OnSale(super=OOMallObject{id=9, creatorId=1, creatorName=admin, mod
lue='67690477f3a3f266052d1852', description='null')-mongo1:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo2:27
lue='67690477f3a3f266052d1852', description='null')-mongo3:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo1:27
lue='67690477f3a3f266052d1852', description='null')-mongo3:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo3:27017 with
lue='67690477f3a3f266052d1852', description='null')-mongo2:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo3:27017 with
lue='67690477f3a3f266052d1852', description='null')-mongo2:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo2:27017 with
lue='67690477f3a3f266052d1852', description='null')-mongo1:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo2:27
lue='67690477f3a3f266052d1852', description='null')-mongo1:27017] INFO org.mongodb.driver.cluster - Rediscovering type of existing primary mongo1:27017 with

```

The screenshot displays an IDE environment for a Java project named 'oomal-2023'. The left sidebar shows a project tree with 'Spring Boot' and 'ProductApplication' highlighted. The main editor shows the 'product.getShop()' method in 'product_controller.java', which returns a new 'ReturnObject' containing a 'product'. The bottom console shows a log of HTTP requests and responses, including headers like 'Host: localhost:8080', 'Connection: keep-alive', 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.5006.64 Safari/537.36', and 'Accept: application/json;charset=UTF-8'.

The screenshot displays an IDE environment with a project named 'oomal-2023'. The project structure on the left includes a 'Spring Boot' application with various components like 'AtipPayApplication', 'CoreApplication', 'FreightApplication', 'GatewayApplication', 'JexpressApplication', 'OrderApplication', 'PaymentApplication', 'ProductApplication', 'RegionApplication', 'SfExpressApplication', 'ShopApplication', 'WeChatPayApplication', and 'ZTOExpressApplication'. The code editor shows a REST controller with a GET endpoint. The console/actuator log on the right shows a series of DEBUG messages from Spring Boot, including application startup logs and a list of headers being added to the response. A red box highlights the 'addHeader' calls in the log, and a red arrow points to the 'addHeader' method in the code. A red circle highlights the 'addHeader' method in the code, and a red arrow points to the 'addHeader' method in the code.

NACOS

首页文档博客社区Nacos企业版

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

```
1 {
2   "errmsg": "成功",
3   "data": {
4     "id": 1558,
5     "creatorId": 1,
6     "creatorName": "admin",
7     "gmtCreate": "2021-11-11T13:12:48",
8     "name": "奥利奥树莓蓝莓",
9     "originalPrice": 65283,
10    "weight": 106,
11    "barcode": "6901668054029",
12    "unit": "包",
13    "originPlace": "江苏",
14    "commissionRatio": 3,
15    "status": 1,
16    "goodsId": 29,
17    "categoryId": 219,
18    "shopId": 2,
19    "templateId": 16,
20    "freeThreshold": -1
21  },
22  "errno": 0
23 }
```



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

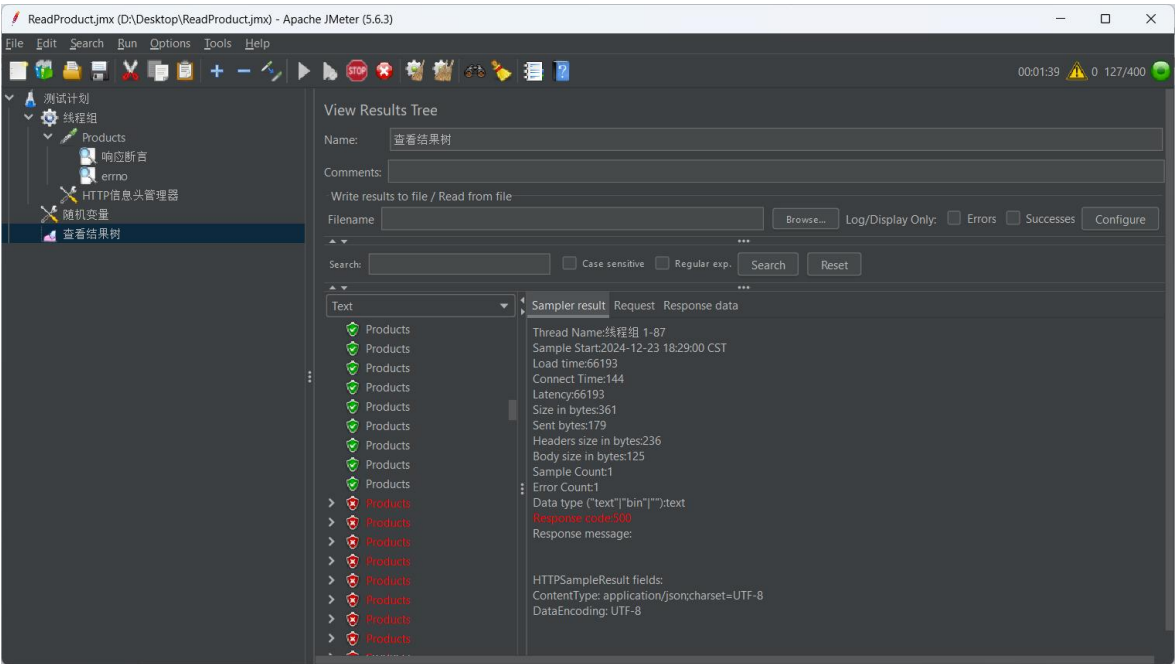
```
1 {
2   "errmsg": "成功",
3   "data": {
4     "id": 1558,
5     "creatorId": 1,
6     "creatorName": "admin",
7     "gmtCreate": "2021-11-11T13:12:48",
8     "name": "奥利奥树莓蓝莓",
9     "originalPrice": 65283,
10    "weight": 106,
11    "barcode": "6901668054029",
12    "unit": "包",
13    "originPlace": "江苏",
14    "commissionRatio": 3,
15    "status": 1,
16    "goodsId": 29,
17    "categoryId": 219,
18    "shopId": 2,
19    "templateId": 16,
20    "freeThreshold": -1
21  },
22   "errno": 0
23 }
```



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

```
1 {
2   "errmsg": "成功",
3   "data": {
4     "id": 1558,
5     "name": "奥利奥树莓蓝莓",
6     "status": 1,
7     "originalPrice": 65283,
8     "weight": 106,
9     "barcode": "6901668054029",
10    "unit": "包",
11    "originPlace": "江苏",
12    "otherProduct": [
13      {
14        "id": 1558,
15        "name": "奥利奥树莓蓝莓",
16        "price": 6985,
17        "status": 1,
18        "quantity": 78
19      },
20      {
21        "id": 3015,
22        "name": "味好美椒盐20",
23        "price": 37684,
24        "status": 1,
25        "quantity": 95
26      },
27      {
28        "id": 3162,
29        "name": "咖喱羊肉串(320)",
30        "price": 3851,
31        "status": 1,
32        "quantity": 31
33      },
34      {
35        "id": 3301,
36        "name": "上海风味火腿(360)",
37        "price": 988,
38        "status": 1,
39        "quantity": 89
40      },
41      {
42        "id": 4598,
43        "name": "180美洁2粒纯木浆纸",
44        "price": 4619,
45        "status": 1,
46        "quantity": 27
47      },
48      {
49        "id": 5045,
50        "name": "110金苑大骨面",
51        "price": 16991,
52        "status": 1,
53        "quantity": 45
54      }
55    ],
56    "errno": 6095
57  }
```

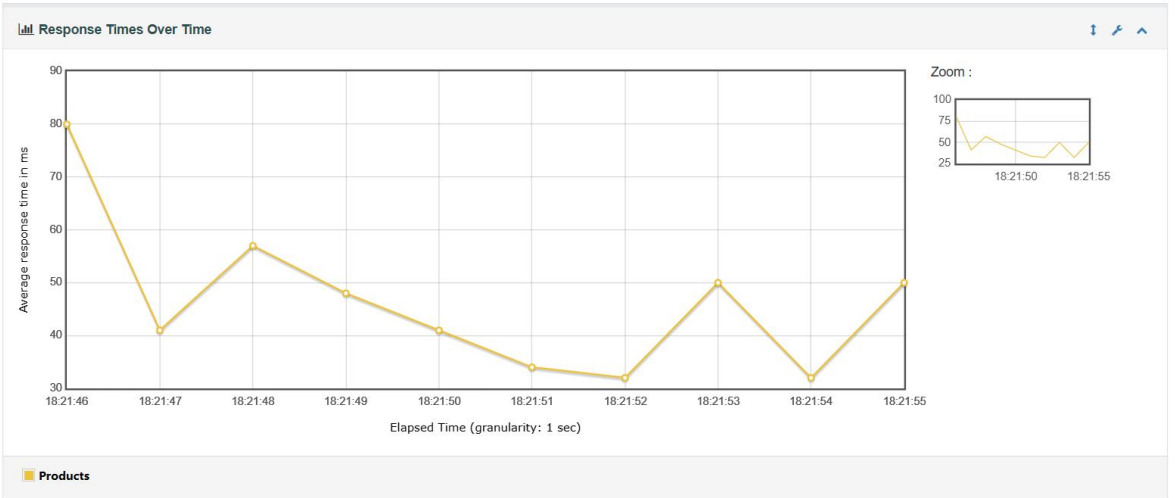
3.两个服务在服务器上已经部署完成，接下来使用 JMeter 进行模拟并发测试，由于在本机测试时发现 JMeter 模拟并发条件时，线程阻塞非常严重，故此只采取较小规模的并发量



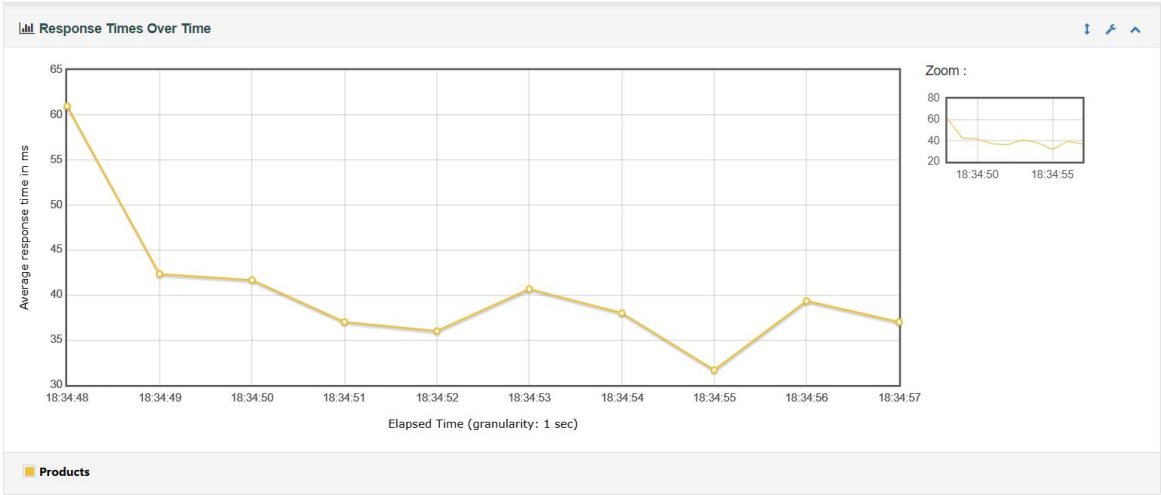
可以看出并发访问时出现请求连接超时的情况，微服务下的并发能力严重不足。

一、不调用 shop 模块

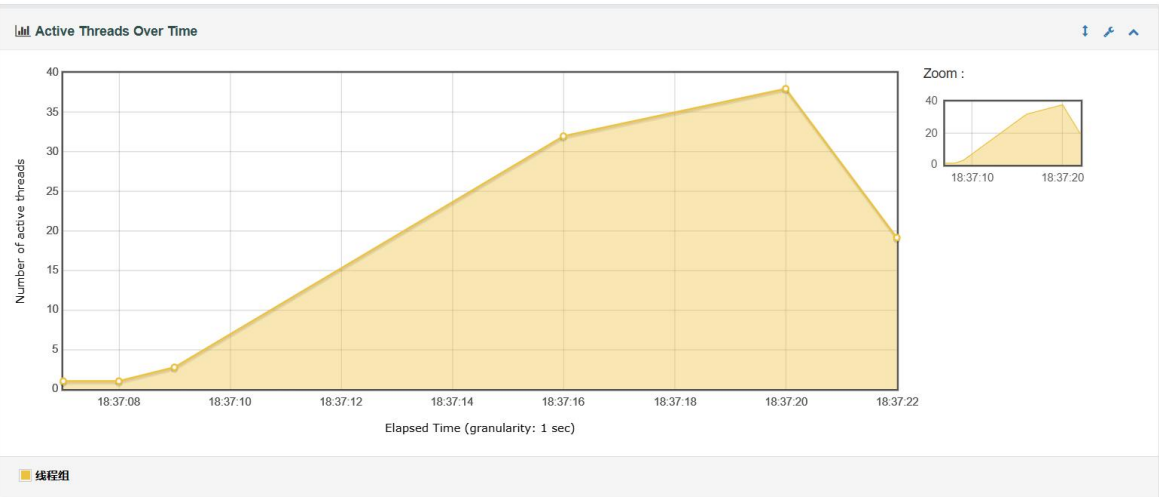
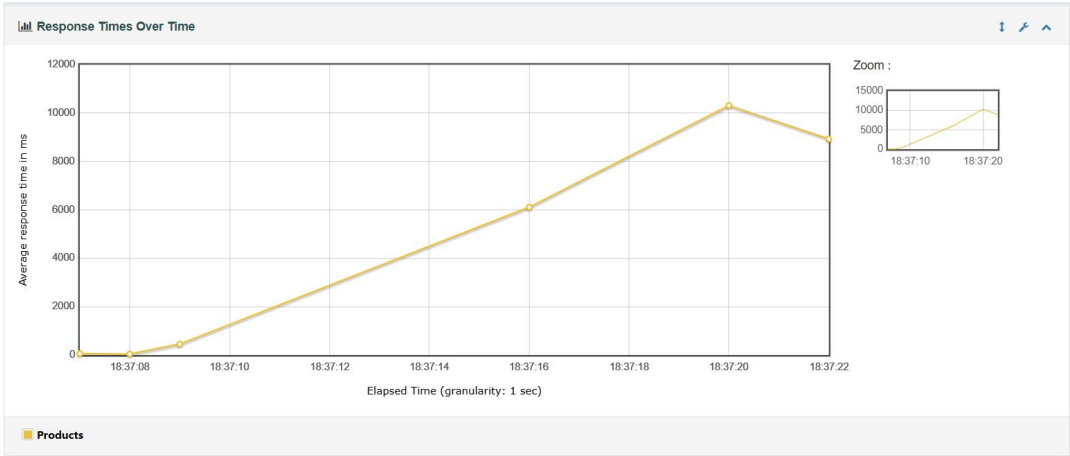
(1)thread-rampTime-loop 为 10-10-1 时，并没有出现线程阻塞问题，响应时间稳定在 40ms 左右



(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

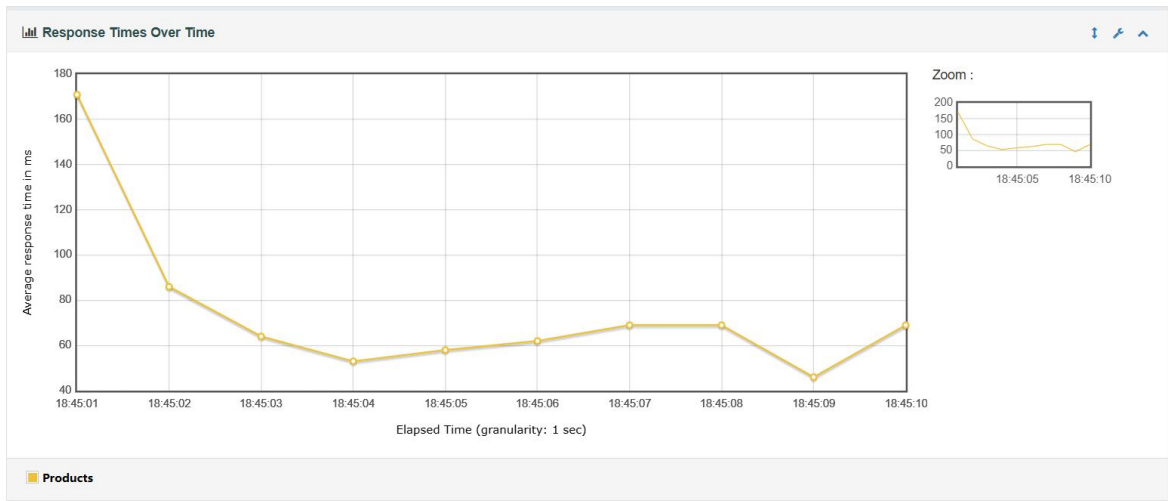


(3)thread-rampTime-loop 为 50-10-1 时，出现了非常严重的线程阻塞问题，最高时有超过 35 条线程被阻塞

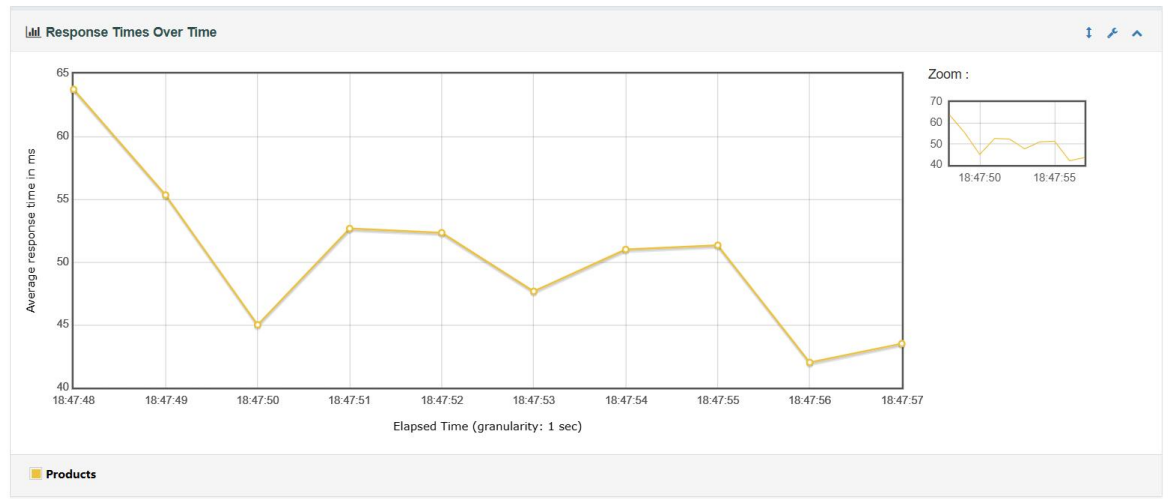


二、调用一次 shop 模块

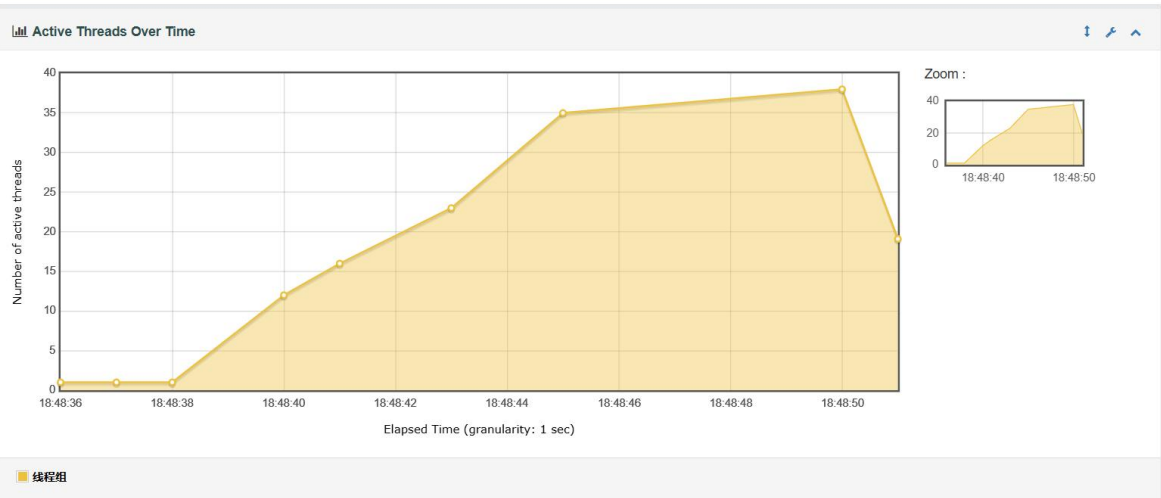
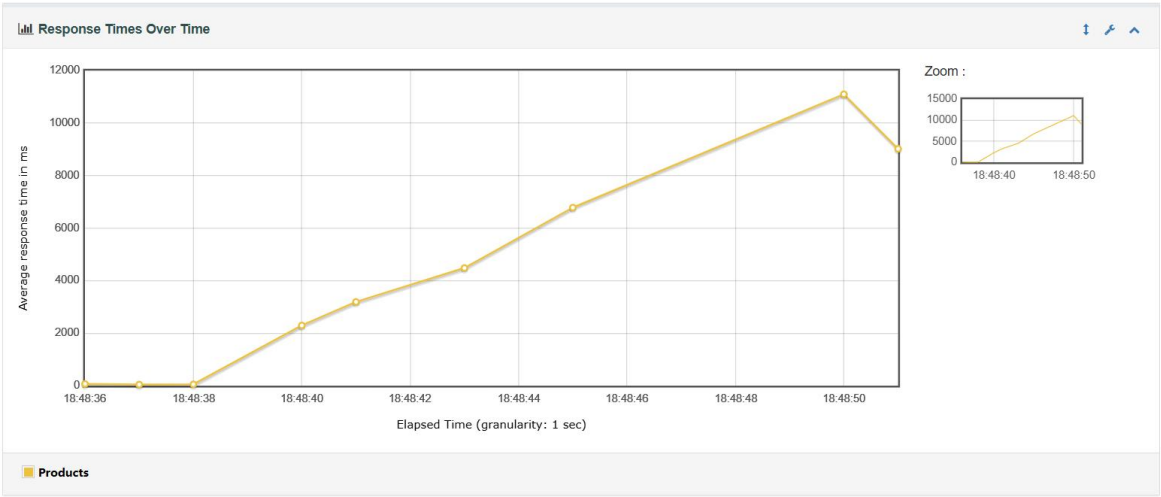
(1)thread-rampTime-loop 为 10-10-1 时，响应时间稳定在 60ms 左右，对比不调用 shop 模块的 40ms，响应速度明显变慢，但此时并没有出现线程阻塞



(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

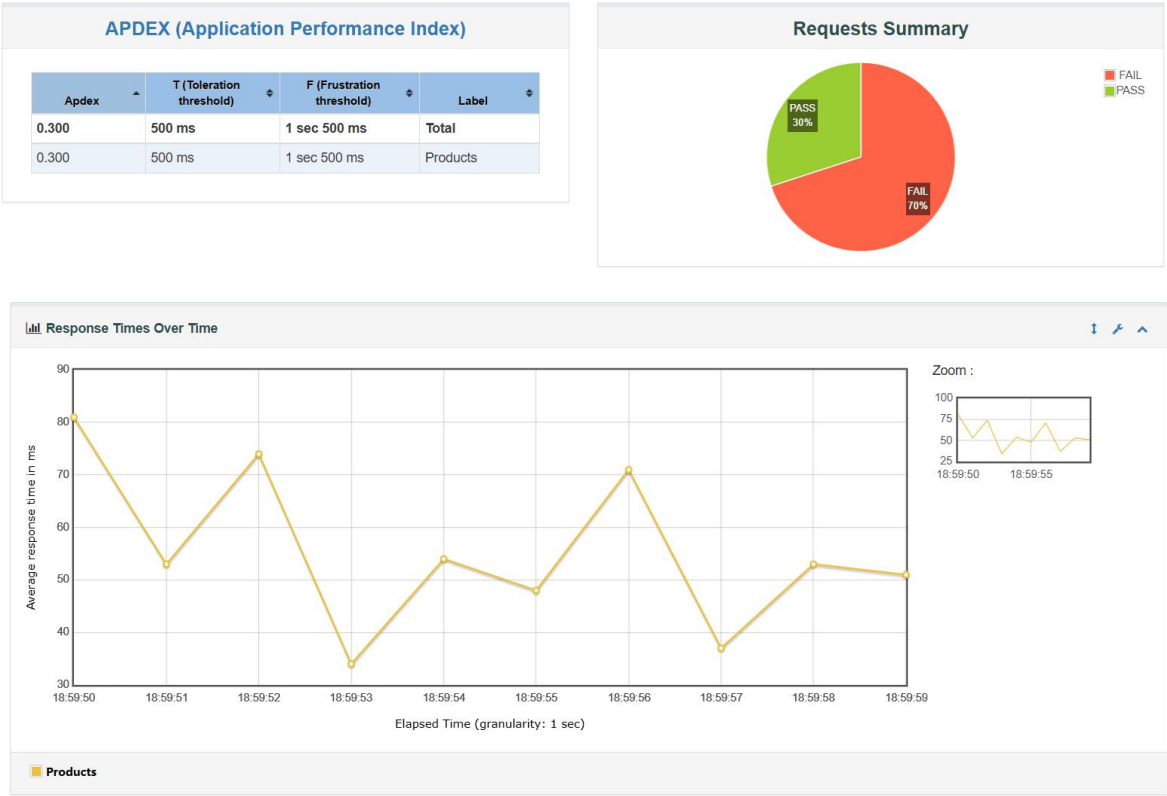


(3)thread-rampTime-loop 为 50-10-1 时，出现了非常严重的线程阻塞问题，相较于不调用 shop 模块，阻塞问题更严重

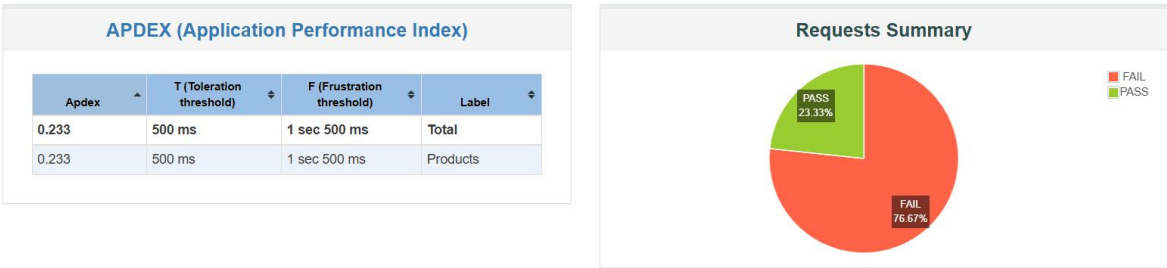


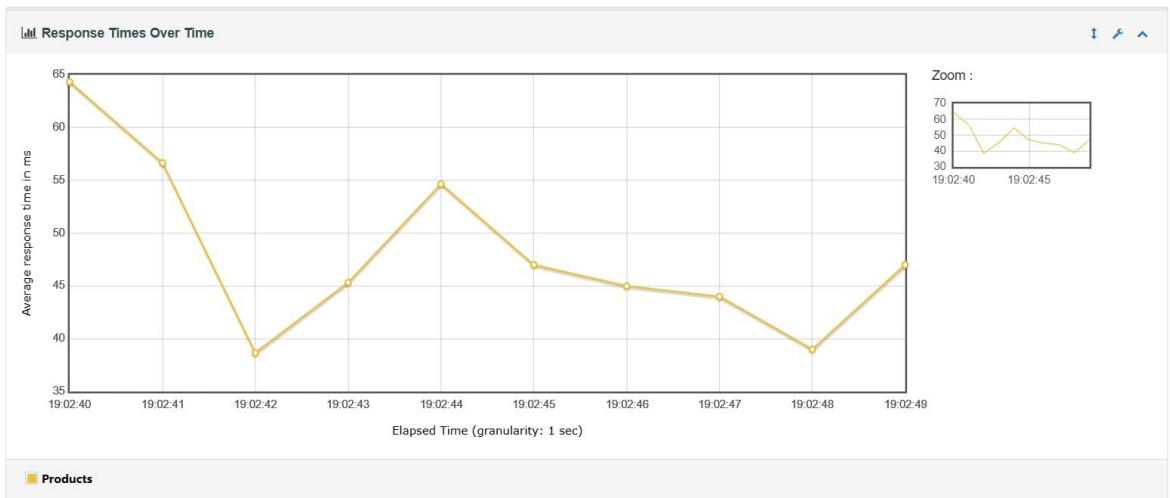
三、调用两次 shop 模块

(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

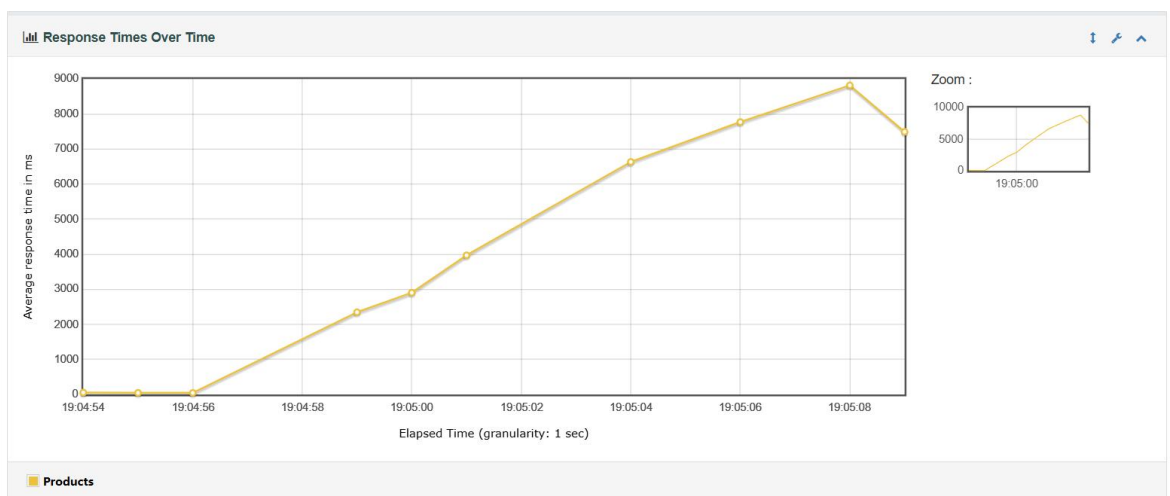
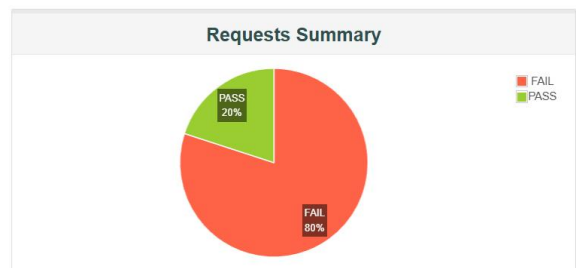




(3)thread-rampTime-loop 为 30-10-1 时，响应时间和调用一次相当，但是已经出现了大多数的线程运行失败的情况，由于某些 product 的 templateId 为空，所以出现大多数线程访问失败，响应时间相较于调用一次 shop 模块更短，应该是 templateId 为空，导致此次访问直接返回结果

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.040	500 ms	1 sec 500 ms	Total
0.040	500 ms	1 sec 500 ms	Products



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

