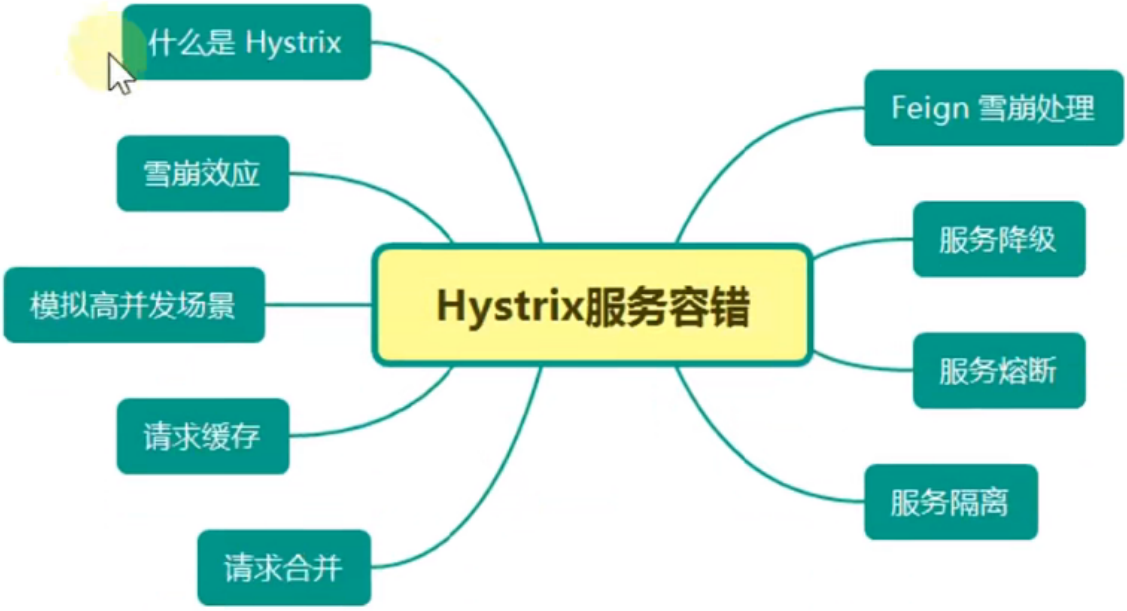
Spring Boot Hystrix服务容错

HyStrix服务容器

1. 技术介绍

2. 学习目标



3. 什么是Netflix

Hystrix 源自 Netflix 团队于 2011 年开始研发。2012年 Hystrix 不断发展和成熟，Netflix 内部的许多团队都采用了它。如今，每天在 Netflix 上通过 Hystrix 执行数百亿个线程隔离和数千亿个信号量隔离的调用。极大地提高了系统的稳定性。

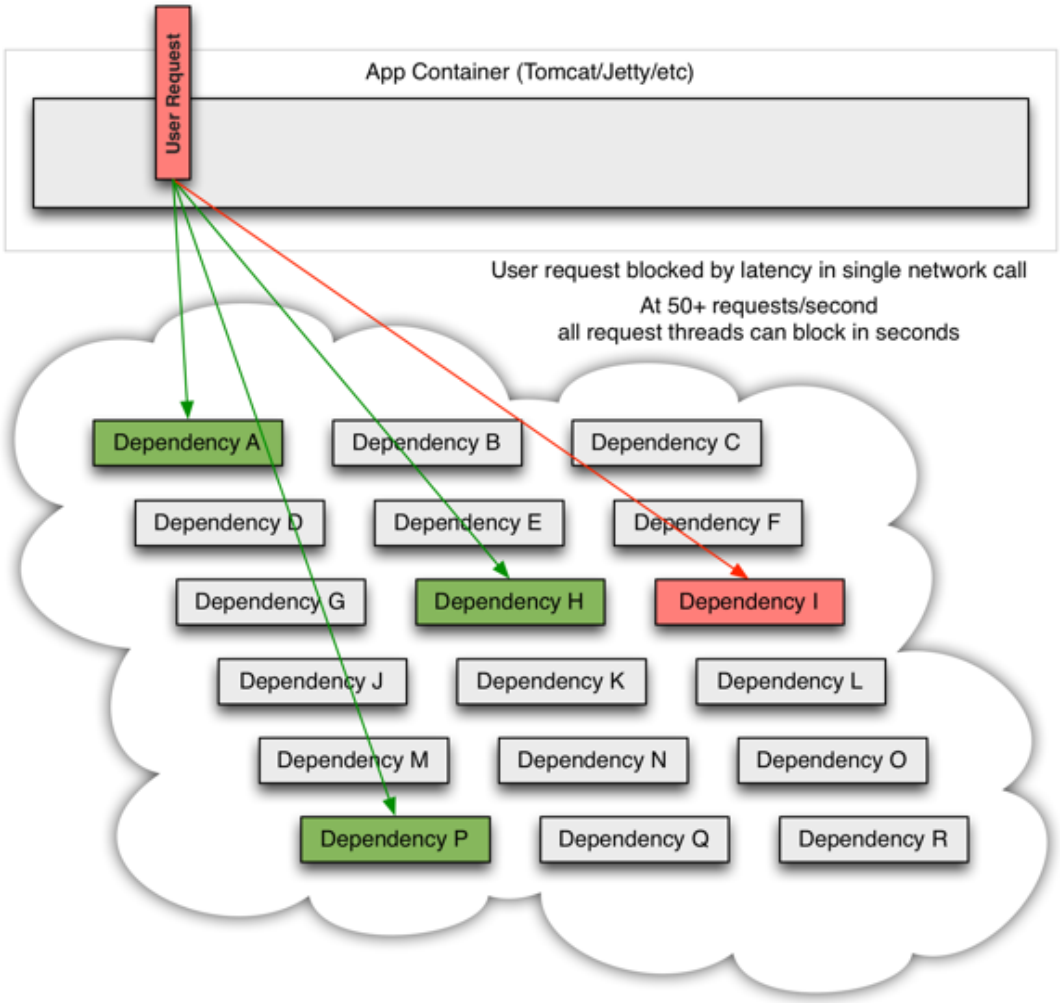
在分布式环境中，不可避免地会有许多服务依赖项中的某些服务失败而导致**雪崩效应**。Hystrix 是一个库，可通过添加等待时间容限和容错逻辑来帮助您控制这些分布式服务之间的交互。Hystrix 通过隔离服务之间的访问点，停止服务之间的级联故障并提供后备选项来实现此目的，所有这些都可以提高系统的整体稳定性。

4. 雪崩效应

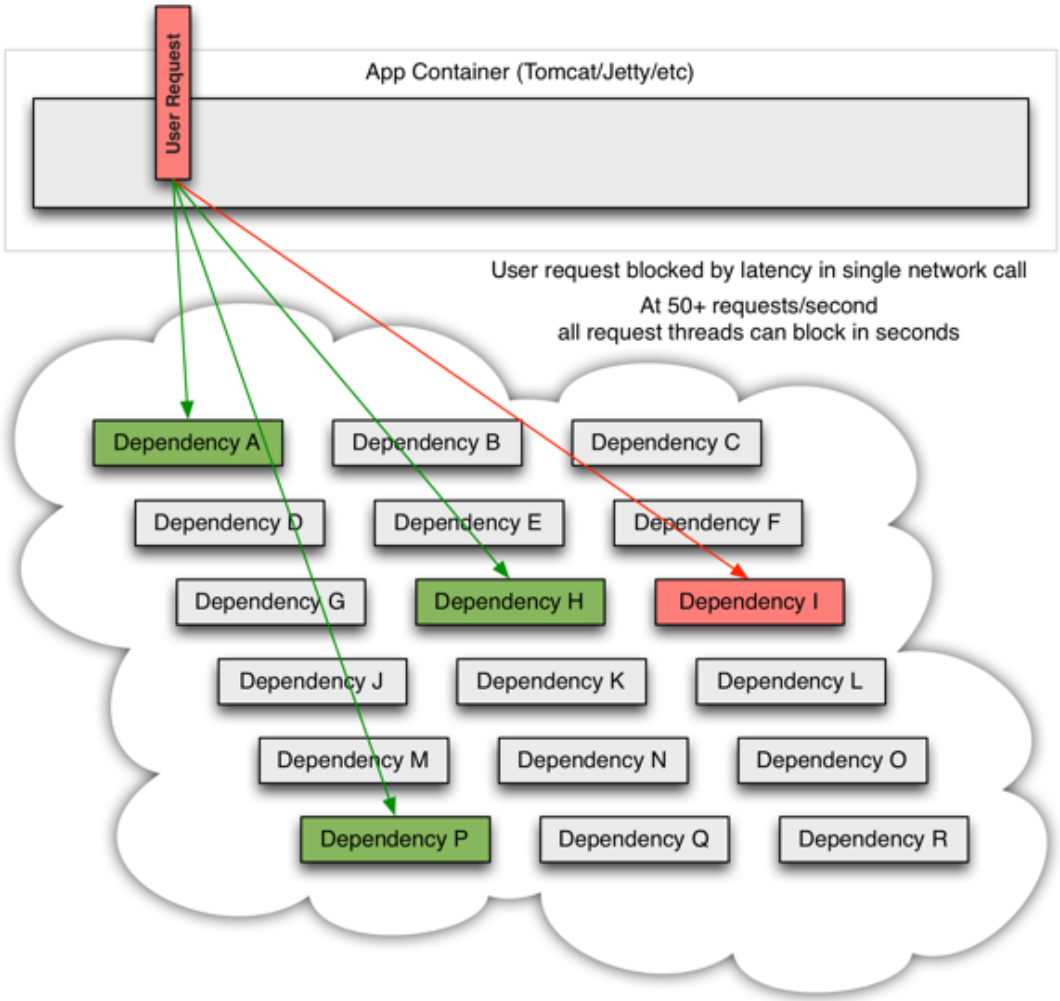
在微服务架构中，一个请求需要调用多个服务是非常常见的。如客户端访问 A 服务，而 A 服务需要调用 B 服务，B 服务需要调用 C 服务，由于网络原因或者自身的原因，如果 B 服务或者 C 服务不能及时响应，A 服务将处于阻塞状态，直到 B 服务 C 服务响应。此时若有大量的请求涌入，容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，造成连锁反应，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。以下图示完美解释了什么是雪崩效应。

Home · Netflix/Hystrix Wiki (github.com)

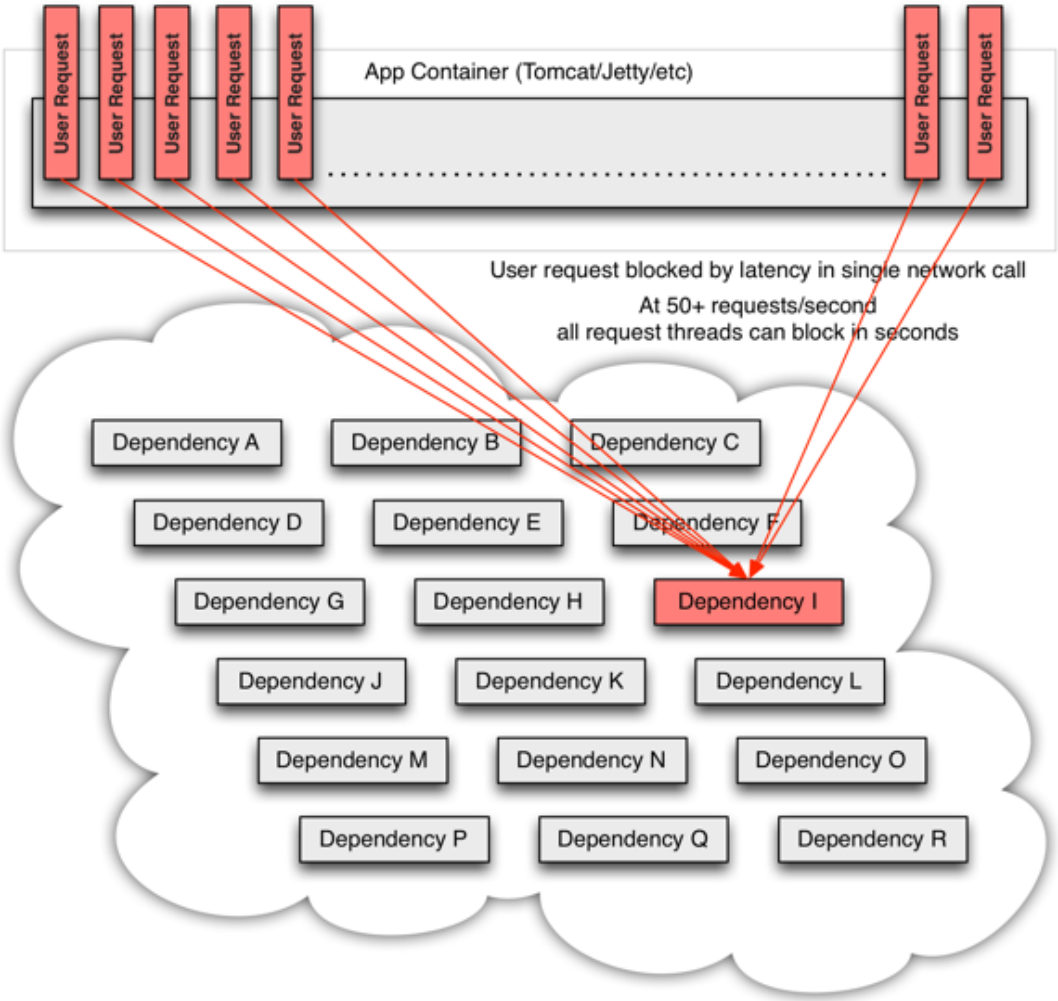
当一切服务正常时，请求看起来是这样的：



当其中一个服务有延迟时，它可能阻塞整个用户请求：



在高并发的情况下，一个服务的延迟可能导致所有服务器上的所有资源在数秒内饱和。比起服务故障，更糟糕的是这些应用程序还可能导致服务之间的延迟增加，从而备份队列，线程和其他系统资源，从而导致整个系统出现更多级联故障。



总结

造成雪崩的原因可以归结为以下三点：

- 服务提供者不可用（硬件故障，程序 BUG，缓存击穿，用户大量请求等）
- 重试加大流量（用户重试，代码逻辑重试）
- 服务消费者不可用（同步等待造成的资源耗尽）

最终的结果就是：一个服务不可用，导致一系列服务的不可用

5. 解决方案

雪崩是系统中的蝴蝶效应导致，其发生的原因多种多样，从源头我们无法完全杜绝雪崩的发生，但是雪崩的根本原因来源于服务之间的强依赖，所以我们可以提前评估做好服务容错。解决方案大概可以分为以下几种：

- 请求缓存：支持将一个请求与返回结果做缓存处理；
- 请求合并：将相同的请求进行合并然后调用批处理接口；
- 服务隔离：限制调用分布式服务的资源，某一个调用的服务出现问题不会影响其他服务调用；
- 服务熔断：牺牲局部服务，保全整体系统稳定性的措施；

- 服务降级：服务熔断以后，客户端调用自己本地方法返回缺省值。

6. 环境准备

`hystrix-demo` 聚合工程。`SpringBoot 2.2.4.RELEASE`、`Spring Cloud Hoxton.SR1`。

- `eureka-server`：注册中心
- `eureka-server02`：注册中心
- `product-service`：商品服务，提供了 `/product/{id}` 接口，`/product/list` 接口，`/product/listByIds` 接口
- `order-service-rest`：订单服务，基于 `Ribbon` 通过 `RestTemplate` 调用商品服务
- `order-server-feign`：订单服务，基于 `Feign` 通过声明式服务调用商品服务

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - 192.168.126.1:8762 , 192.168.126.1:8761
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.126.1:8080
SERVICE-CONSUMER	n/a (1)	(1)	UP (1) - 192.168.126.1:8081

https://www.cnblogs.com/mrhelloworld/p/hystrix1.html

6.1创建项目

忽略

6.2 添加依赖

具体依赖包，可见Github上pom.xm文件内容

6.3 注册中心Eureka

忽略

6.4 商品微服务

6.4.1 创建服务提供者

忽略

6.4.2 添加依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>hystrix</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>product-service</artifactId>
  <packaging>jar</packaging>

  <name>product-service</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!--spring boot web 依赖-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--netflix eureka client 依赖-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
      <version>3.0.2</version>
    </dependency>

    <!--lombok 依赖包-->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
    </dependency>

    <!--mysql 依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.18</version>
    </dependency>
    <!--mybatis-plus 依赖-->
    <dependency>
      <groupId>com.baomidou</groupId>
      <artifactId>mybatis-plus-boot-starter</artifactId>
      <version>3.4.0</version>
    </dependency>

    <!--swagger2 依赖-->
    <dependency>
```

```
      <groupId>io.springfox</groupId>
      <artifactId>springfox-swagger2</artifactId>
      <version>2.7.0</version>
    </dependency>
    <!--swagger 第三方ui依赖-->
    <dependency>
      <groupId>com.github.xiaoymin</groupId>
      <artifactId>swagger-bootstrap-ui</artifactId>
      <version>1.9.6</version>
    </dependency>

    <!--spring boot actuator 依赖-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <!--Junit 测试单元依赖-->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

  </dependencies>

</project>
```

6.4.3 配置文件

```
server:
  port: 8080

spring:
  application:
    # 应用名称(集群内相同)
    name: product-service
  main:
    allow-circular-references: true
  mvc:
    pathmatch:
      matching-strategy: ANT_PATH_MATCHER
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: "jdbc:mysql://localhost:3306/micro_service?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai"
    username: root
    password: password

# 配置Eureka Server注册中心
eureka:
  instance:
    # 主机名, 不配置的时候将根据操作系统的主机名称来获取
    hostname: localhost
    # 是否开启IP地址注册
```

```
    prefer-ip-address: true
    # 主机地址+端口号
    instance-id: ${spring.cloud.client.ip-address}:${server.port}
  client:
    serviceUrl:
      # 注册中心对外暴露的注册地址
      defaultZone:
http://root:123456@localhost:8761/eureka/,http://root:123456@localhost:8762/eureka/
    register-with-eureka: true
    fetch-registry: true

# 度量指标监控与健康检测
management:
  endpoints:
    web:
      exposure:
        # 开启shutdown端点访问
        include: shutdown
  endpoint:
    shutdown:
      # 开启shutdown实现优雅停止服务
      enabled: true
```

6.4.3 ProductController

```
package com.example.controller;


import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * <p>
 * 前端控制器
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@RestController
@RequestMapping("/product")
public class ProductController {

  @Autowired
  private IProductService productService;

  /**
```

```
     * 功能描述: 查询商品列表
     *
     * @return
     */
    @GetMapping("/list")
    public List<Product> selectProductList() {
        return productService.selectProductList();
    }

    /**
     * 功能描述: 根据多个主键查询商品
     *
     * @param ids
     * @return
     */
    @GetMapping("/{ids}/product/listByIds")
    public List<Product> selectProductListByIds(@PathVariable("ids") List<Integer> ids)
{
        return productService.selectProductListByIds(ids);
    }

    /**
     * 功能描述: 使用GET方法, 根据主键查询商品
     *
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Product selectProductById(@PathVariable("id") Integer id) {
//        try {
//            // 设置超时验证
//            Thread.sleep(2000L);
//        } catch (InterruptedException e) {
//            throw new RuntimeException(e);
//        }
        return productService.selectProductById(id);
    }

    /**
     * 功能描述: 使用POST方法,根据主键查询商品
     *
     * @param id
     * @return
     */
    @PostMapping("/single")
    public Product queryProductById(@RequestBody Integer id) {
        return productService.queryProductById(id);
    }


    /**
     * 功能描述: 使用POST方法,添加商品
     *
     * @param product
     * @return
     */
    @PostMapping("/save")
    public RespBean addProduct(@RequestBody Product product) {
```

```
    if (product.getProductName() == null || product.getProductNum() == null
        || product.getProductPrice() == null) {
      return RespBean.success("添加失败");
    }
    return productService.addProduct(product);
  }


  /**
   * 功能描述: 接收商品对象参数
   *
   * @param product
   * @return
   */
  @GetMapping("/pojo")
  public Product selectProductByPojo(@RequestBody Product product) {
    return productService.selectProductByPojo(product);
  }
}
```

6.4.4 IProductService

```
package com.example.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;

/**
 * <p>
 * 服务类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
public interface IProductService extends IService<Product> {


  /**
   * 功能描述: 查询商品列表
   *
   * @return
   */
  List<Product> selectProductList();

  /**
   * 功能描述: 根据多个主键查询商品
   *
   * @param ids
   * @return
   */
  List<Product> selectProductListByIds(List<Integer> ids);
```

```
    /**
     * 功能描述: 使用GET方法,根据主键查询商品
     *
     * @return
     */
    Product selectProductById(Integer id);


    /**
     * 功能描述: 使用POST方法,根据主键查询商品
     *
     * @param id
     * @return
     */
    Product queryProductById(Integer id);


    /**
     * 功能描述: 使用POST方法,添加商品
     *
     * @param product
     * @return
     */
    RespBean addProduct(Product product);

    /**
     * 功能描述: 接收商品对象参数
     *
     * @param product
     * @return
     */
    Product selectProductByPojo(Product product);

}
```

6.4.5 ProductServiceImpl

```
package com.example.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.mapper.ProductMapper;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import java.util.ArrayList;
import java.util.List;
import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
```

```
 * @since 2023-02-23
 */
@Service
public class ProductServiceImpl extends ServiceImpl<ProductMapper, Product> implements
    IProductService {

  @Autowired
  @Resource
  private ProductMapper productMapper;

  /**
   * 功能描述: 查询商品列表
   *
   * @return
   */
  @Override
  public List<Product> selectProductList() {

    //    return Arrays.asList(
    //        new Product(1, "华为手机", 1, 5800D),
    //        new Product(2, "联想电脑", 1, 29999D),
    //        new Product(3, "苹果手机", 1, 9899D)
    //    );
    return productMapper.selectProductList();
  }

  /**
   * 功能描述: 根据多个主键查询商品
   *
   * @param ids
   * @return
   */
  @Override
  public List<Product> selectProductListByIds(List<Integer> ids) {
    return productMapper.selectProductListByIds(ids);
  }

  /**
   * 功能描述: 根据主键查询商品
   *
   * @param id
   * @return
   */
  @Override
  public Product selectProductById(Integer id) {
    return productMapper.selectProductById(id);
  }

  /**
   * 功能描述: 使用POST方法,根据主键查询商品
   *
   * @param id
   * @return
   */
  @Override
  public Product queryProductById(Integer id) {
    return productMapper.queryProductById(id);
  }
```

```
  /**
   * 功能描述: 使用POST方法,添加商品
   *
   * @param product
   * @return
   */
  @Override
  public RespBean addProduct(Product product) {

    int result = productMapper.addProduct(product);
    System.out.println("添加商品 = " + product);
    if (result > 0) {
      return RespBean.success("添加成功!");
    } else {
      return RespBean.success("添加失败!");
    }
  }

  /**
   * 功能描述: 接收商品对象参数
   *
   * @param product
   * @return
   */
  @Override
  public Product selectProductByPojo(Product product) {
    System.out.println("接收商品对象参数 = " + product);
    return product;
  }
}
```

6.4.6 ProductMapper

```
package com.example.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.example.pojo.Product;
import java.util.List;
import org.apache.ibatis.annotations.Param;

/**
 * <p>
 * Mapper 接口
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
public interface ProductMapper extends BaseMapper<Product> {

  /**
   * 功能描述: 查询商品列表
   *
   * @return
   */
```

```
    List<Product> selectProductList();

    /**
     * 功能描述: 根据多个主键查询商品
     *
     * @param ids
     * @return
     */
    List<Product> selectProductListByIds(@Param("ids") List<Integer> ids);

    /**
     * 功能描述: 根据主键查询商品
     *
     * @param id
     * @return
     */
    Product selectProductById(@Param("id") Integer id);

    /**
     * 功能描述: 使用POST方法,根据主键查询商品
     *
     * @param id
     * @return
     */
    Product queryProductById(@Param("id") Integer id);

    /**
     * 功能描述: 使用POST方法,添加商品
     *
     * @param product
     * @return
     */
    int addProduct(Product product);


}
```

6.4.7 Mybatis Metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.mapper.ProductMapper">

  <!-- 通用查询映射结果 -->
  <resultMap id="BaseResultMap" type="com.example.pojo.Product">
    <id column="id" property="id"/>
    <result column="productName" property="productName"/>
    <result column="productNum" property="productNum"/>
    <result column="productPrice" property="productPrice"/>
  </resultMap>

  <!-- 通用查询结果列 -->
  <sql id="Base_Column_List">
    id
    , productName, productNum, productPrice
```

```
  </sql>

  <!--    查询商品列表-->
  <select id="selectProductList" resultType="com.example.pojo.Product">
    select *
    from t_product;
  </select>

  <!--  根据主键查询商品-->
  <select id="selectProductById" resultType="com.example.pojo.Product">
    select *
    from t_product
    where id = #{id};
  </select>

  <!--  功能描述: 使用POST方法,根据主键查询商品-->
  <select id="queryProductById" resultType="com.example.pojo.Product">
    select *
    from t_product
    where id = #{id};
  </select>

  <!--  根据多个主键查询商品-->
  <select id="selectProductListByIds" resultType="com.example.pojo.Product">
    select *
    from t_product
    <where>
      id in
      <foreach collection="ids" item="ids" index="index"
        open="(" separator="," close=")">#{ids}
      </foreach>
    </where>
  </select>

  <!--  使用POST方法,添加商品-->
  <insert id="addProduct">
    insert into t_product(id, productName, productNum, productPrice)
    values (#{id}, #{productName}, #{productNum}, #{productPrice});
  </insert>

</mapper>
```

6.5 订单微服务

6.5.1 创建服务提供者

忽略

6.5.2 添加依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
```

```
    <artifactId>hystrix</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>order-service-rest</artifactId>
  <packaging>jar</packaging>

  <name>order-service-rest</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!--spring boot web 依赖-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--netflix eureka client 依赖-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
      <version>3.0.2</version>
    </dependency>

    <!--spring cloud openfeign 依赖-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-openfeign</artifactId>
    </dependency>

    <!--lombok 依赖包-->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
    </dependency>

    <!--mysql 依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.18</version>
    </dependency>
    <!--mybatis-plus 依赖-->
    <dependency>
      <groupId>com.baomidou</groupId>
      <artifactId>mybatis-plus-boot-starter</artifactId>
      <version>3.4.0</version>
    </dependency>

    <!--swagger2 依赖-->
    <dependency>
      <groupId>io.springfox</groupId>
```

```
      <artifactId>springfox-swagger2</artifactId>
      <version>2.7.0</version>
    </dependency>
    <!--swagger 第三方ui依赖-->
    <dependency>
      <groupId>com.github.xiaoymin</groupId>
      <artifactId>swagger-bootstrap-ui</artifactId>
      <version>1.9.6</version>
    </dependency>

    <!--spring boot actuator 依赖-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <!--Junit 测试单元依赖-->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!--Ribbon 依赖包-->
    <dependency>
      <groupId>com.netflix.ribbon</groupId>
      <artifactId>ribbon-loadbalancer</artifactId>
      <version>2.3.0</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/io.github.openfeign/feign-httpclient -->
    <dependency>
      <groupId>io.github.openfeign</groupId>
      <artifactId>feign-httpclient</artifactId>
      <version>11.10</version>
    </dependency>

  </dependencies>
</project>
```

6.5.3 配置文件

```
server:
  port: 8081
  compression:
    # 是否开启压缩请求
    enabled: true
    # 配置支持类型
    mime-types: application/json,application/xml,text/html,text/html,text/plain

spring:
  application:
    # 应用名称
    name: service-consumer
  main:
```

```yaml
    allow-circular-references: true
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: "jdbc:mysql://localhost:3306/micro_service?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai"
    username: root
    password: password

# 配置Eureka Server注册中心
eureka:
  instance:
    # 主机名, 不配置的时候将根据操作系统的主机名称来获取
    hostname: localhost
    # 是否开启IP地址注册
    prefer-ip-address: true
    # 主机地址+端口号
    #instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
    instance-id: ${spring.cloud.client.ip-address}:${server.port}
  client:
    serviceUrl:
      # 注册中心对外暴露的注册地址
      defaultZone:
http://root:123456@localhost:8761/eureka/,http://root:123456@localhost:8762/eureka/
    # 是否将自己注册到注册中心,默认为true
    register-with-eureka: true
    # 表示Eureka Client 间隔多长时间服务器来取注册信息,默认为30秒
    registry-fetch-interval-seconds: 10

# 局部负载均衡策略
# service-provider是指被调用者的微服务名称
service-provider:
  ribbon:
    NFLoadBalancerRuleClassName: com.netflix.loadBalancer.RandomRule
#    ribbon:
#      OkToRetryOnAllOperations: false #对所有操作请求都进行重试,默认false
#      ReadTimeout: 5000   #负载均衡超时时间，默认值5000
#      ConnectTimeout: 3000 #ribbon请求连接的超时时间，默认值2000
#      MaxAutoRetries: 3     #对当前实例的重试次数，默认0
#      MaxAutoRetriesNextServer: 1 #对切换实例的重试次数，默认1

feign:
  httpclient:
    enabled: true # 开启httpclient
  client:
    config:
      service-provider: # 需要调用的服务名称
        logger-level: Full
#ribbon:
#  ConnectionTimeout: 5000 # 请求连接超时时间,默认为1秒
#  ReadTimeout: 5000 # 请求处理的超时时间
```

6.5.4 OrderController

```
package com.example.controller;


import com.example.pojo.Order;
import com.example.service.IOrderService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * <p>
 * 前端控制器
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-24
 */
@RestController
@RequestMapping("/order")
public class OrderController {

  @Autowired
  private IOrderService orderService;


  /**
   * 根据主键查询订单-->调用商品服务/product/list
   *
   * @param id
   * @return
   */
  @GetMapping("/{id}/product/list")
  public Order selectOrderById(@PathVariable("id") Integer id) {
    return orderService.selectOrderById(id);
  }

  /**
   * 根据主键查询订单-->调用商品服务/product/listByIds
   *
   * @param id
   * @return
   */
  @GetMapping("/{id}/product/listByIds")
  public Order queryOrderById(@PathVariable("id") Integer id) {
    return orderService.queryOrderById(id);
  }

  /**
   * 根据主键查询订单-->调用商品服务/product/{id}
   *
   * @param id
   * @return
   */
  @GetMapping("/{id}/product")
```

```
    public Order searchOrderById(@PathVariable("id") Integer id) {
        return orderService.searchOrderById(id);
    }

}
```

6.5.5 IOrderService

```
package com.example.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.example.pojo.Order;
import io.swagger.models.auth.In;

/**
 * <p>
 * 服务类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-24
 */
public interface IOrderService extends IService<Order> {

    /**
     * 根据主键查询订单-->调用商品服务/product/list
     *
     * @param id
     * @return
     */
    Order selectOrderById(Integer id);

    /**
     * 根据主键查询订单-->调用商品服务/product/listByIds
     *
     * @param id
     * @return
     */
    Order queryOrderById(Integer id);

    /**
     * 根据主键查询订单-->调用商品服务/product/{id}
     *
     * @param id
     * @return
     */
    Order searchOrderById(Integer id);
}
```

6.5.6 OrderServiceImpl

```
package com.example.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.mapper.OrderMapper;
import com.example.pojo.Order;
import com.example.pojo.Product;
import com.example.service.IOrderService;
import com.example.service.IProductService;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
@Service
public class OrderServiceImpl extends ServiceImpl<OrderMapper, Order> implements
IOrderService {

  @Autowired
  @Resource
  private OrderMapper orderMapper;
  @Autowired
  @Resource
  private RestTemplate restTemplate;
  @Autowired
  private IProductService productService;
  @Autowired
  private LoadBalancerClient loadBalancerClient; //Ribbon负载均衡器

  /**
   * 根据主键查询订单-->调用商品服务/product/list
   *
   * @param id
   * @return
   */
  @Override
  public Order selectOrderById(Integer id) {

    //查询商品列表
    List<Product> productList = productService.selectProductList();
```

```
    System.out.println("商品信息 = " + productList);

    //获取订单信息
    Order mapperOrderById = orderMapper.getOrderById(id);
    Order order = new Order();
    order.setId(mapperOrderById.getId());
    order.setOrderNo(mapperOrderById.getOrderNo());
    order.setOrderAddress(mapperOrderById.getOrderAddress());
    order.setTotalPrice(mapperOrderById.getTotalPrice());
    order.setProductList(productList);
    return order;
  }

  /**
   * 根据主键查询订单-->调用商品服务/product/listByIds
   *
   * @param id
   * @return
   */
  @Override
  public Order queryOrderById(Integer id) {
    //查询商品列表
    List<Product> productList =
productService.selectProductListByIds(Collections.singletonList(1));

    System.out.println("商品信息 = " + productList);

    //获取订单信息
    Order mapperOrderById = orderMapper.getOrderById(id);
    Order order = new Order();
    order.setId(mapperOrderById.getId());
    order.setOrderNo(mapperOrderById.getOrderNo());
    order.setOrderAddress(mapperOrderById.getOrderAddress());
    order.setTotalPrice(mapperOrderById.getTotalPrice());
    order.setProductList(productList);
    return order;
  }

  /**
   * 根据主键查询订单-->调用商品服务/product/{id}
   *
   * @param id
   * @return
   */
  @Override
  public Order searchOrderById(Integer id) {
    //查询商品列表
    Product productList =productService.selectProductById(1);

    System.out.println("商品信息 = " + productList);

    //获取订单信息
    Order mapperOrderById = orderMapper.getOrderById(id);
    Order order = new Order();
    order.setId(mapperOrderById.getId());
    order.setOrderNo(mapperOrderById.getOrderNo());
    order.setOrderAddress(mapperOrderById.getOrderAddress());
```

```
    order.setTotalPrice(mapperOrderById.getTotalPrice());
    order.setProductList(Collections.singletonList(productList));
    return order;
  }

}
```

6.5.7 OrderMapper

```
package com.example.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.example.pojo.Order;

/**
 * <p>
 * Mapper 接口
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-24
 */
public interface OrderMapper extends BaseMapper<Order> {

  /**
   * 获取订单
   *
   * @param id
   * @return
   */
  Order getOrderById(Integer id);

  /**
   * 功能描述: 根据主键查询订单
   *
   * @param id
   * @return
   */
  Order searchOrderById(Integer id);
}
```

6.5.8 Mybatis Metatdata

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.mapper.OrderMapper">

  <!-- 通用查询映射结果 -->
  <resultMap id="BaseResultMap" type="com.example.pojo.Order">
    <id column="id" property="id"/>
    <result column="orderNo" property="orderNo"/>
    <result column="orderAddress" property="orderAddress"/>
    <result column="totalPrice" property="totalPrice"/>
```

```
  </resultMap>

  <!-- 通用查询结果列 -->
  <sql id="Base_Column_List">
    id
    , orderNo, orderAddress, totalPrice
  </sql>

  <!--    根据主键查询订单-->
  <select id="searchOrderById" resultType="com.example.pojo.Order">
    select *
    from t_order
    where id = #{id}
  </select>

  <!--    根据主键查询订单-->
  <select id="getOrderById" resultType="com.example.pojo.Order">
    select *
    from t_order
    where id = #{id}
  </select>

</mapper>
```

6.6 Ribbon负载均衡(核心代码)

使用loadBalancerClient的ribbon负载均衡调用微服务接口

6.6.1 IProductService

创建被调用商品微服务接口

```
package com.example.service;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * 功能描述: 调用服务生产者的微服务名称,例如：service-provider
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
public interface IProductService {

  /**
   * 功能描述: 查询商品列表
   *
   * @return
```

```
     */
    //配置需要调用的微服务地址和参数
    List<Product> selectProductList();

    /**
     * 功能描述: 根据主键查询商品
     *
     * @param ids
     * @return
     */
    List<Product> selectProductListByIds(List<Integer> ids);


    /**
     * 功能描述: 使用POST方法,根据主键查询商品
     *
     * @param id
     * @return
     */
    Product selectProductById(Integer id);


}
```

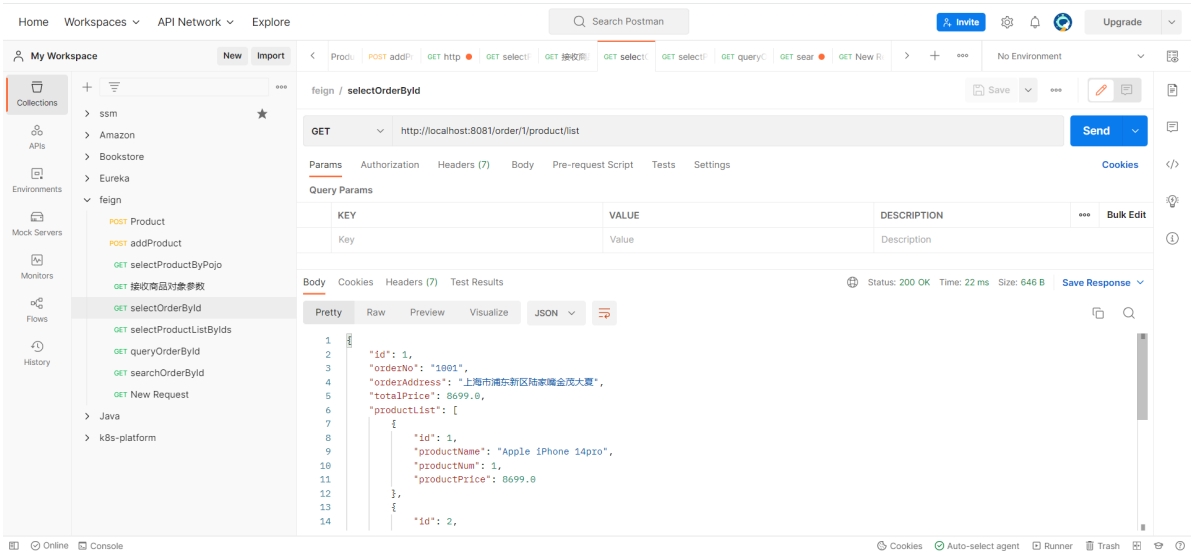
6.6.2 ProductServiceImpl

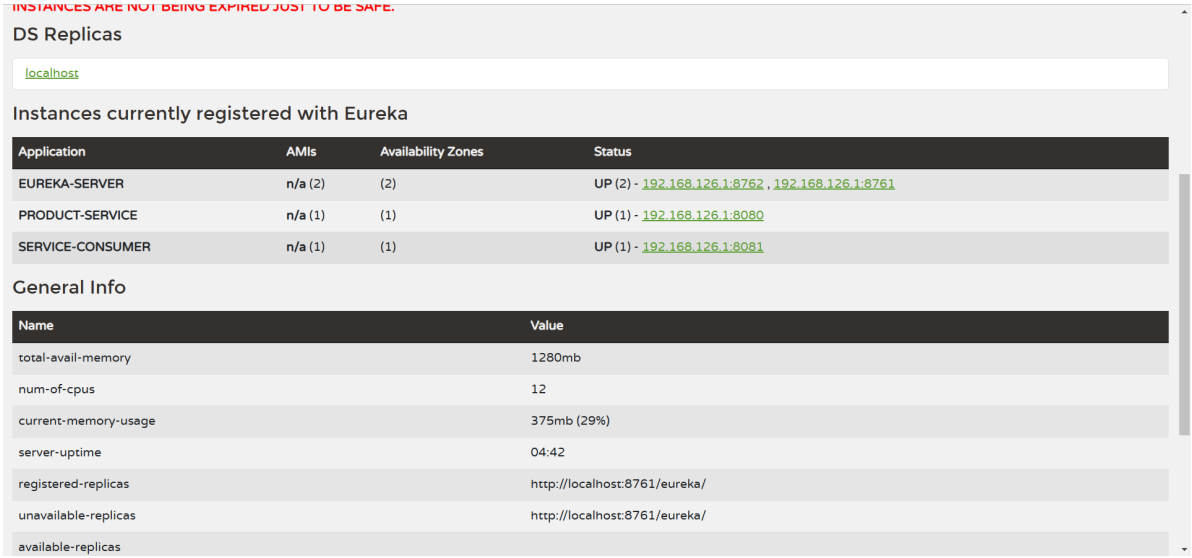
实现接口调用方法，具体通过RestTemplate调用微服务的接口

```
package com.example.service.impl;

import com.example.pojo.Product;
import com.example.service.IProductService;
import java.util.List;
import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@Service
public class ProductServiceImpl implements IProductService {

    @Autowired
    private RestTemplate restTemplate;
    @Autowired
```

```
  private LoadBalancerClient loadBalancerClient; //Ribbon负载均衡器

  /**
   * 功能描述: 查询商品列表
   *
   * @return
   */

  @Override
  public List<Product> selectProductList() {
    return restTemplate.exchange("http://product-service/product/list",
        HttpMethod.GET,
        null,
        new ParameterizedTypeReference<List<Product>>() {
        }).getBody();
  }

  /**
   * 功能描述: 根据多个主键查询商品
   *
   * @param ids
   * @return
   */
  @Override
  public List<Product> selectProductListByIds(List<Integer> ids) {
    System.out.println("-----orderService-----selectProductListByIds-----");
    StringBuffer sb = new StringBuffer();
    ids.forEach(id -> sb.append("id=" + id + "&"));
    return restTemplate.getForObject("http://product-service/product/listByIds?" +
sb.toString(), List.class);
  }

  /**
   * 功能描述: 根据主键查询商品
   *
   * @param id
   * @return
   */
  @Override
  public Product selectProductById(Integer id) {
    return restTemplate.getForObject("http://product-service/product/" + id,
Product.class);
  }

}
```

6.6.3 Restful API Test





6.7 模拟搞并发场景

6.7.1 设置线程超时

在商品微服务提供者的/product/list接口设置等待2秒钟

```
/**
 * 功能描述: 查询商品列表
 *
 * @return
 */
@GetMapping("/list")
public List<Product> selectProductList() {
    try {
        // 设置超时验证
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
```

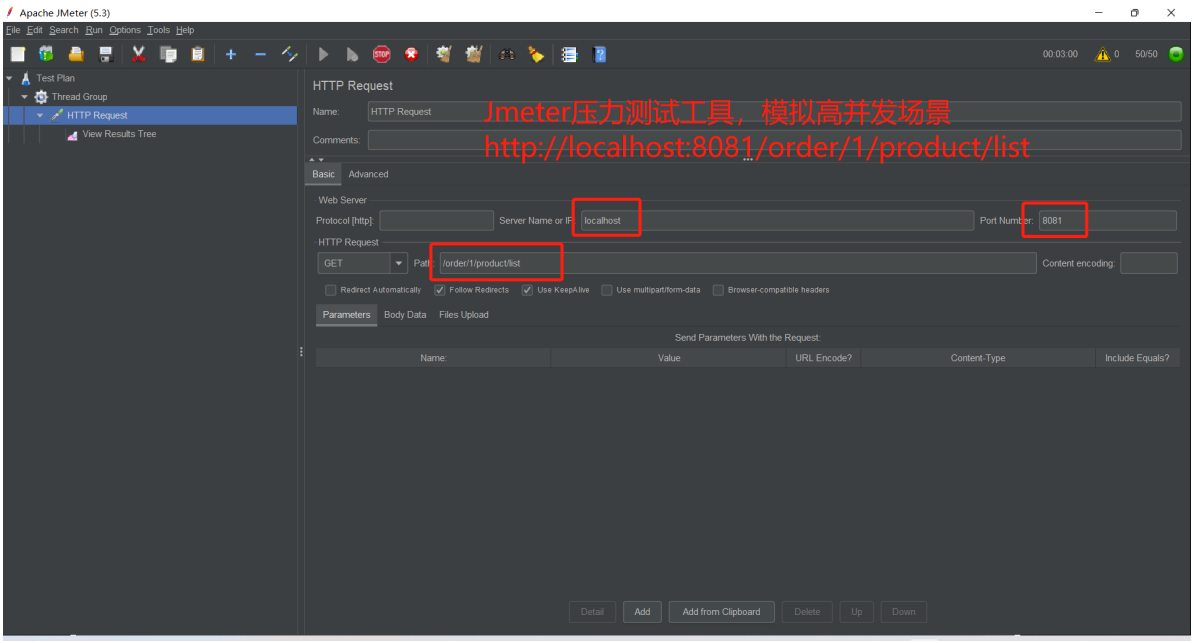
```
    return productService.selectProductList();
}
```

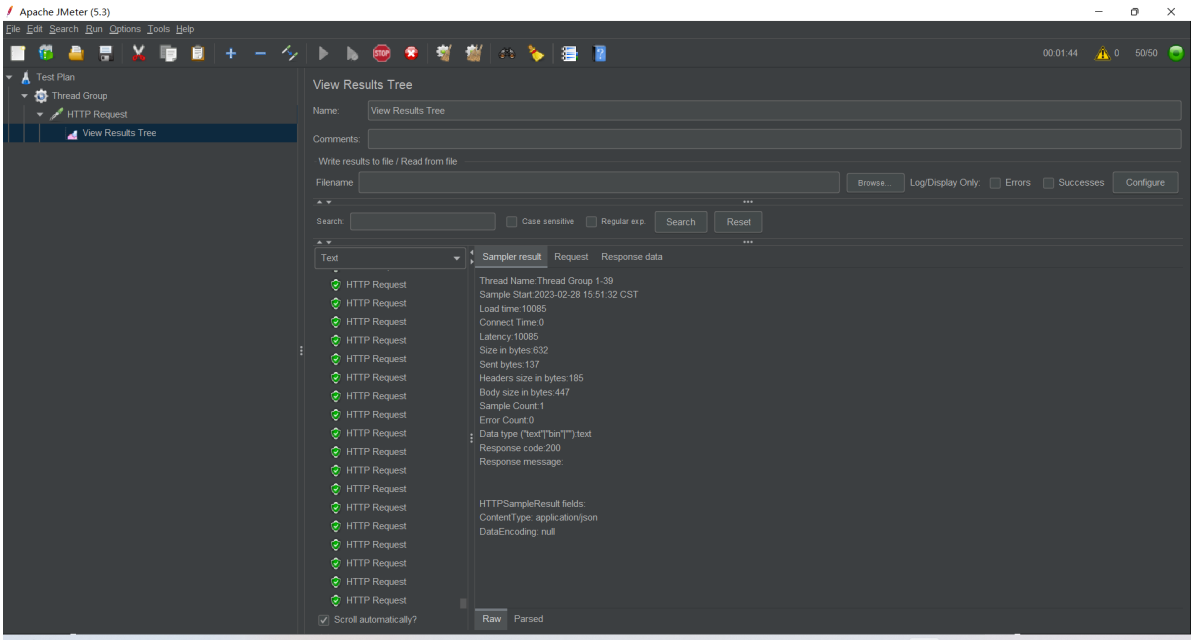
6.7.2 设置tomcat线程数

将tomcat默认最大线程数200，调整为10

```
server:
  port: 8081
  compression:
    # 是否开启压缩请求
    enabled: true
    # 配置支持类型
    mime-types: application/json,application/xml,text/html,text/html,text/plain
  tomcat:
    threads:
      max: 10 # 降低最大线程数方便模拟测试
```

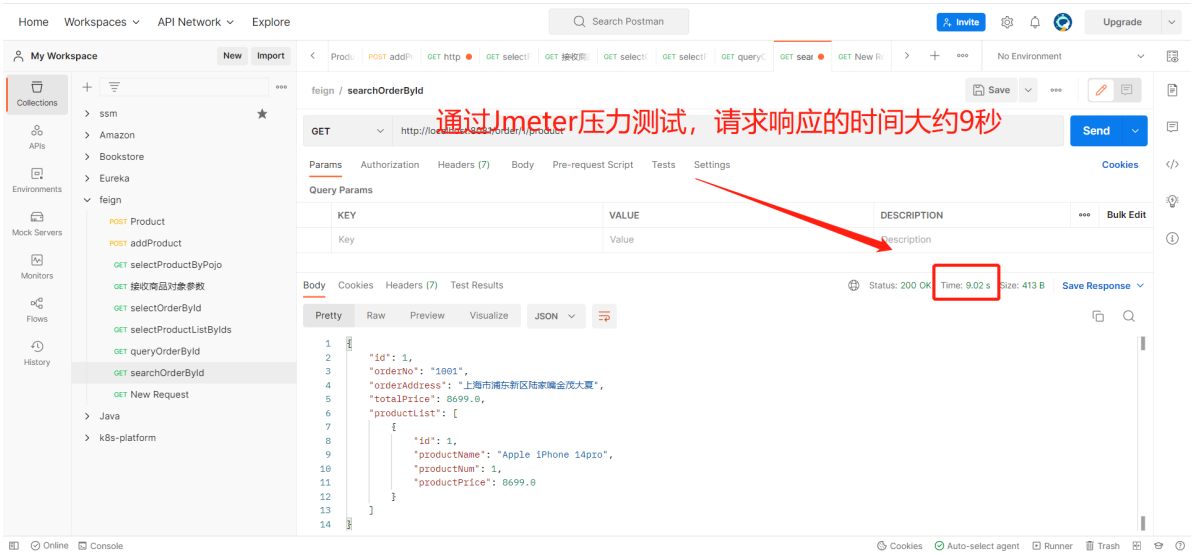
6.7.3 Jmeter压力测试

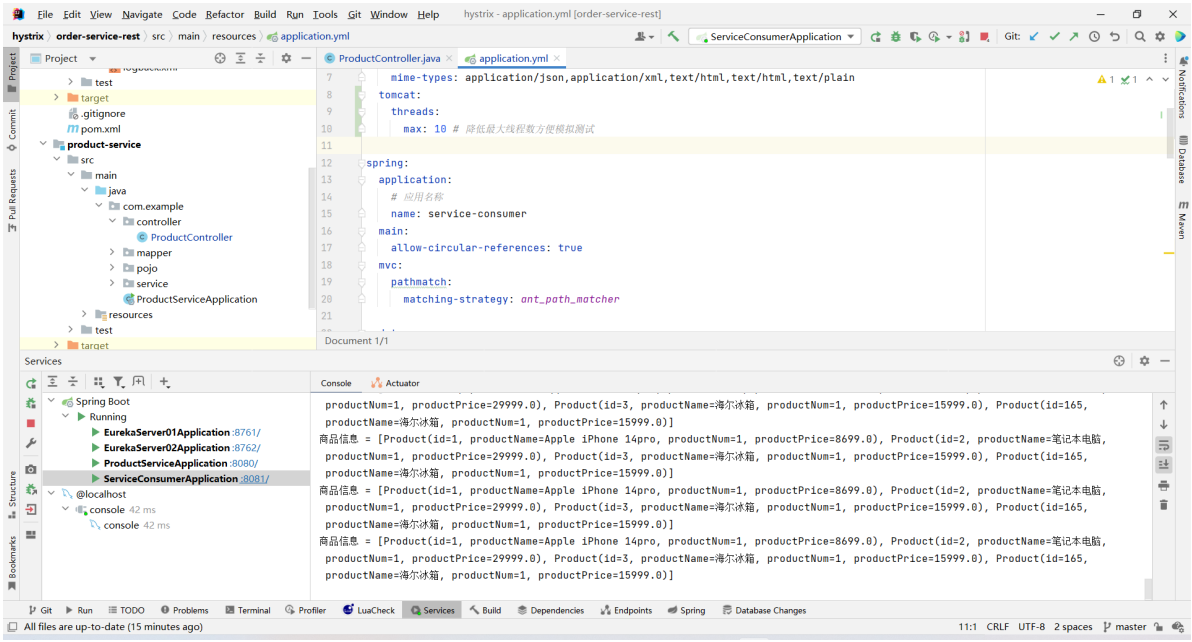




6.7.4 浏览器请求其它接口

通过浏览器访问 http://localhost:8081/order/1/product ， 由于Jmeter工具高并发调用接口 http://localhost:8081/order/1/product/listByIds 导致正在通过浏览器的访问接口耗时变成。





7. Redis请求缓存

Hystrix 为了降低访问服务的频率，支持将一个请求与返回结果做缓存处理。如果再次请求的 URL 没有变化，那么 Hystrix 不会请求服务，而是直接从缓存中将结果返回。这样可以大大降低访问服务的压力。

7.1 Redis安装

忽略

Hystrix 自带缓存有两个缺点：

- 本地缓存，集群情况下缓存无法同步。
- 不支持第三方缓存容器，如：Redis，MemCache。

本文使用 Spring 的缓存集成方案，NoSql 使用 Redis 来实现，Redis 使用的是 5.0.7 版本。

7.2 添加依赖

服务消费者 pom.xml 添加 redis 和 commons-pool2 依赖。

```
<!-- spring boot data redis 依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- commons-pool2 对象池依赖 -->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
</dependency>
```

7.3 配置文件

```yaml
redis:
  #超时时间
  timeout: 10000ms
  #服务器地址
  host: 192.168.126.61
  #服务器端口
  port: 6379
  #数据库
  database: 0
  #密码
  password: Rational123
  lettuce:
    pool:
      #最大连接数，默认8
      max-active: 1024
      #最大连接阻塞等待时间，默认-1
      max-wait: 10000ms
      #最大空闲连接
      max-idle: 200
      #最小空闲连接
      min-idle: 5
```

7.4 Redis配置类

```java
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.cache.RedisCacheConfiguration;
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.cache.RedisCacheWriter;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.GenericJackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.RedisSerializationContext;
import org.springframework.data.redis.serializer.StringRedisSerializer;

import java.time.Duration;

/**
 * Redis 配置类
 */
@Configuration
public class RedisConfig {

    // 重写 RedisTemplate 序列化
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
redisConnectionFactory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        // 为 String 类型 key 设置序列化器
        template.setKeySerializer(new StringRedisSerializer());
        // 为 String 类型 value 设置序列化器
        template.setValueSerializer(new GenericJackson2JsonRedisSerializer());
```

```
        // 为 Hash 类型 key 设置序列化器
        template.setHashKeySerializer(new StringRedisSerializer());
        // 为 Hash 类型 value 设置序列化器
        template.setHashValueSerializer(new GenericJackson2JsonRedisSerializer());
        template.setConnectionFactory(redisConnectionFactory);
        return template;
    }

    // 重写 Cache 序列化
    @Bean
    public RedisCacheManager redisCacheManager(RedisTemplate redisTemplate) {
        RedisCacheWriter redisCacheWriter =
RedisCacheWriter.nonLockingRedisCacheWriter(redisTemplate.getConnectionFactory());
        RedisCacheConfiguration redisCacheConfiguration =
RedisCacheConfiguration.defaultCacheConfig()
                // 设置默认过期时间 30 min
                .entryTtl(Duration.ofMinutes(30))
                // 设置 key 和 value 的序列化

.serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(redisTem
plate.getKeySerializer()))

.serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(redisT
emplate.getValueSerializer()));
        return new RedisCacheManager(redisCacheWriter, redisCacheConfiguration);
    }

}
```

7.5 启动类

服务消费者启动类开启缓存注解

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

// 开启缓存注解
@EnableCaching
@SpringBootApplication
public class OrderServiceRestApplication {

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    public static void main(String[] args) {
        SpringApplication.run(OrderServiceRestApplication.class, args);
    }
```

```
}
```

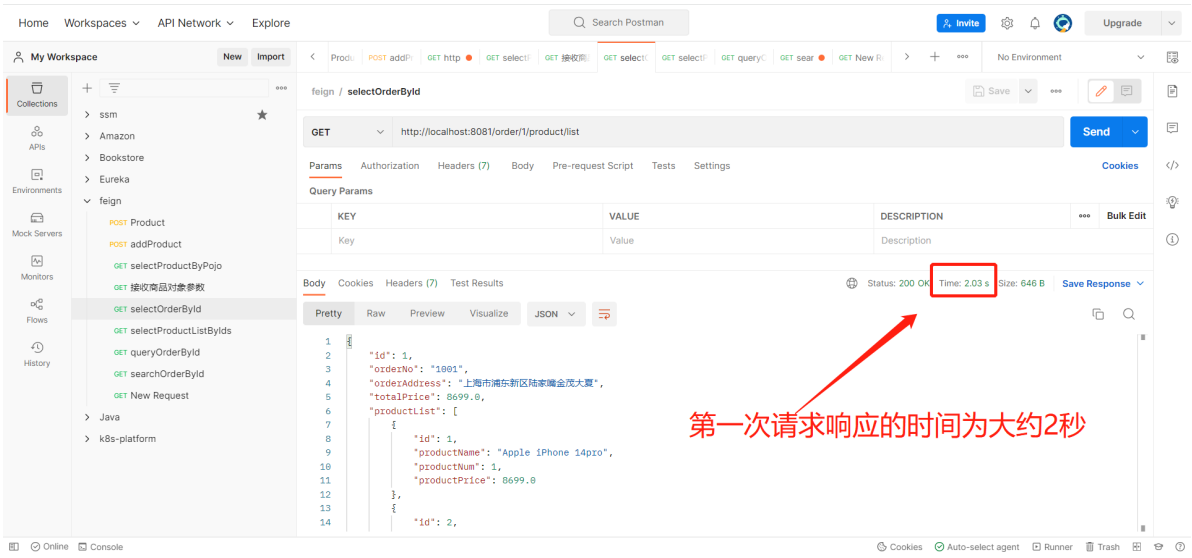
7.6 业务层

服务消费者业务层代码添加缓存规则

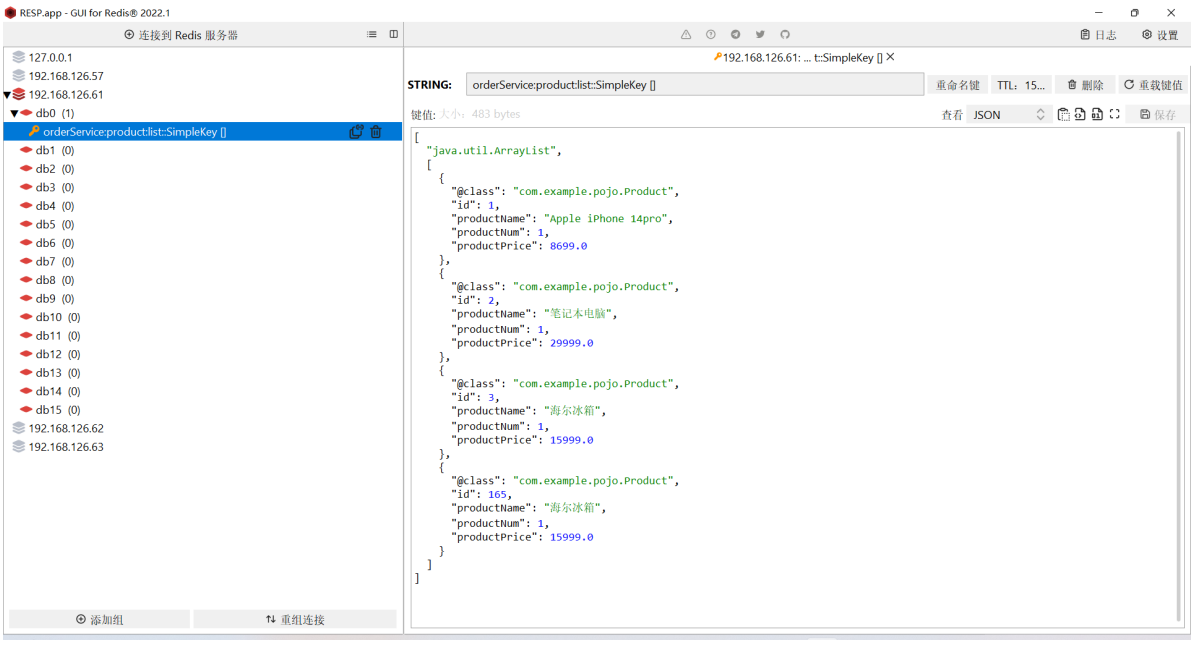
7.7 测试

7.7.1 Restful API Test

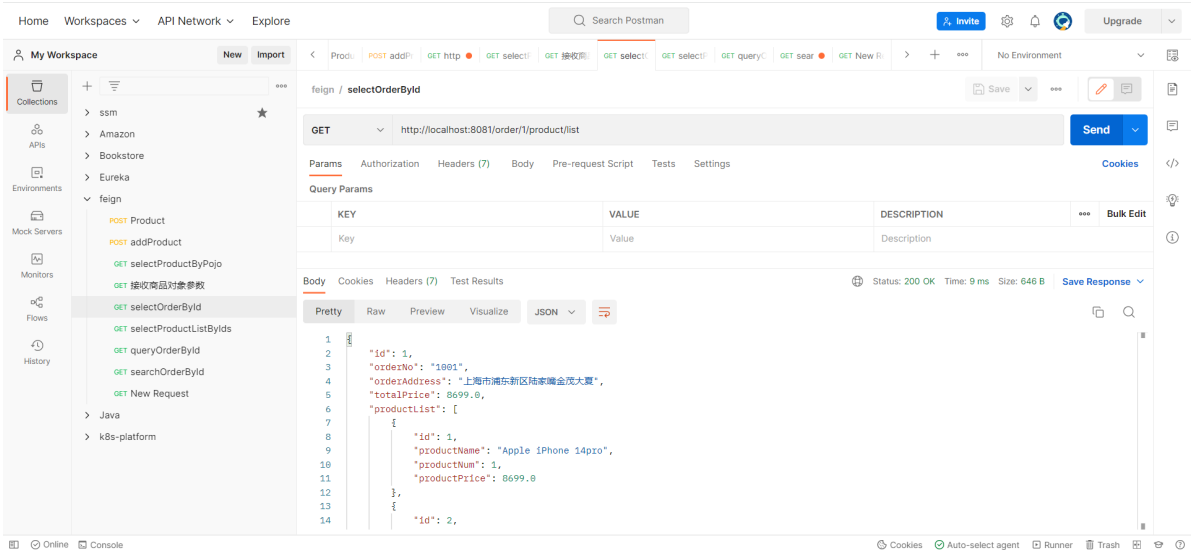
根据主键查询订单-->调用商品服务/product/list(第一次请求响应时间大约2秒钟)，由于消费者项目中引入Redis缓存数据库，所以在第一次调用接口之后自动将数据存储到redis中。



redis缓存中已存入订单微服务接口的数据。

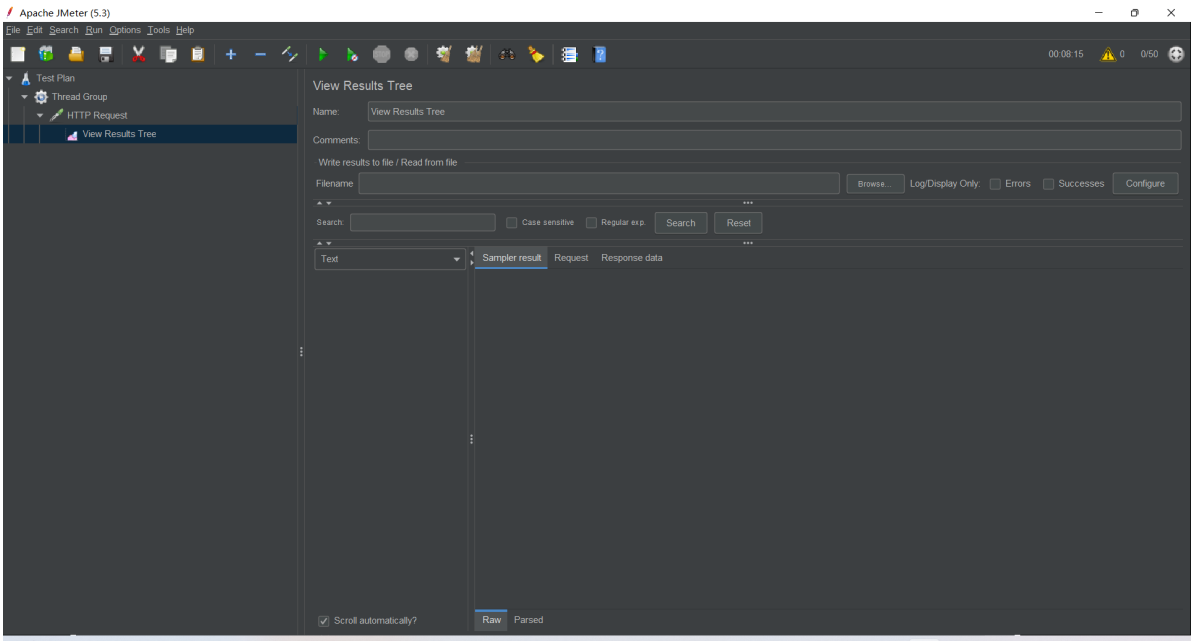


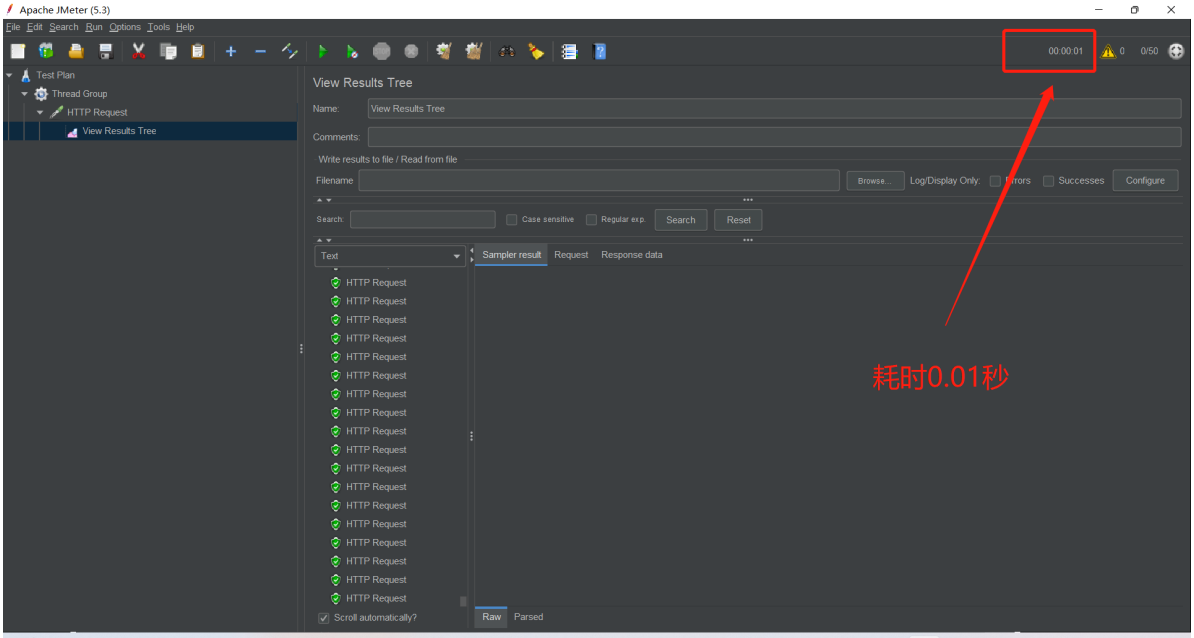
再次调用该接口时候，此时将从redis缓存数据库读取数据。



7.7.2 模拟高并发场景

通过Jmeter模拟高并发场景，如图所示：




Apache JMeter (5.3)
File Edit Search Run Options Tools Help
00:00:01
0
0/50
Test Plan
Thread Group
HTTP Request
View Results Tree
View Results Tree
Name:
View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Search:
Case sensitive
Regular exp.
Search
Reset
Text
Sampler result
Request
Response data
HTTP Request
Scroll automatically?
Raw
Parsed
耗时0.01秒