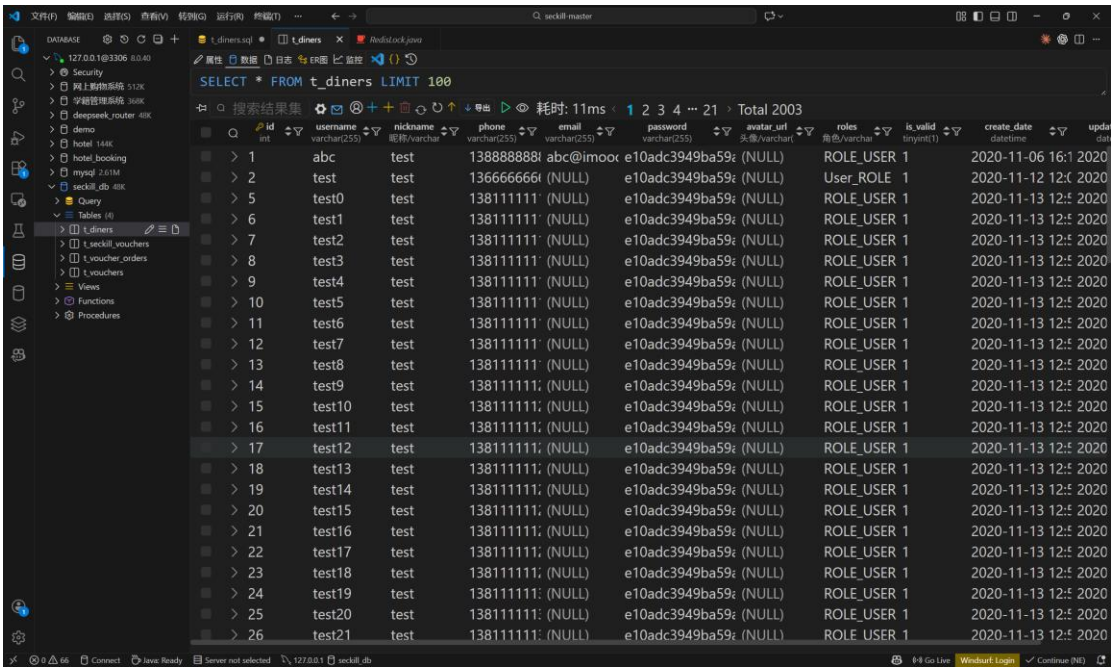
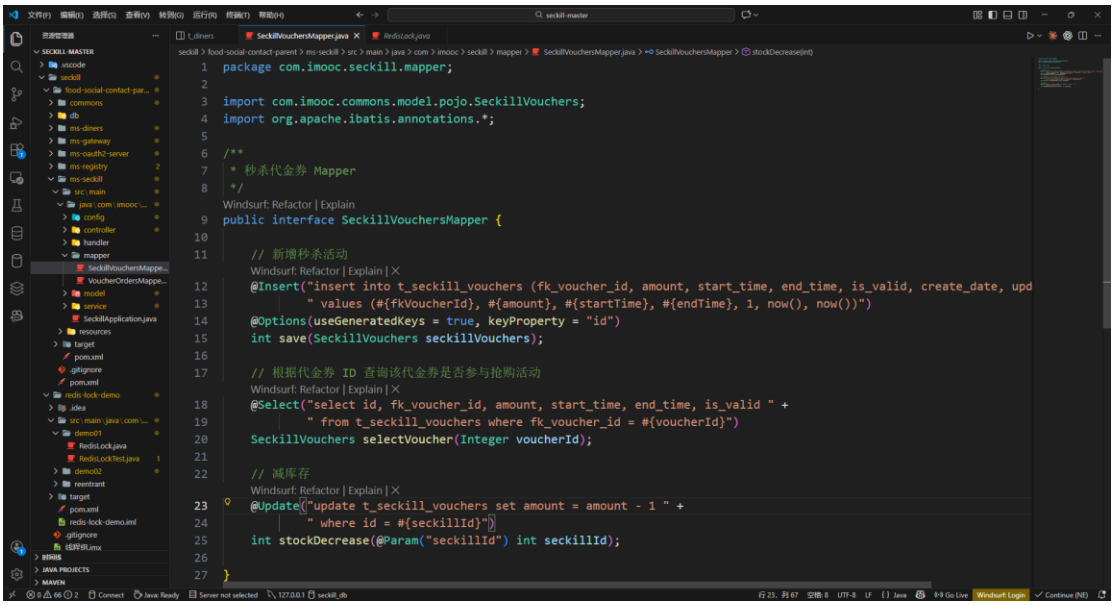
项目截图：

数据库文件 t_diners.sql

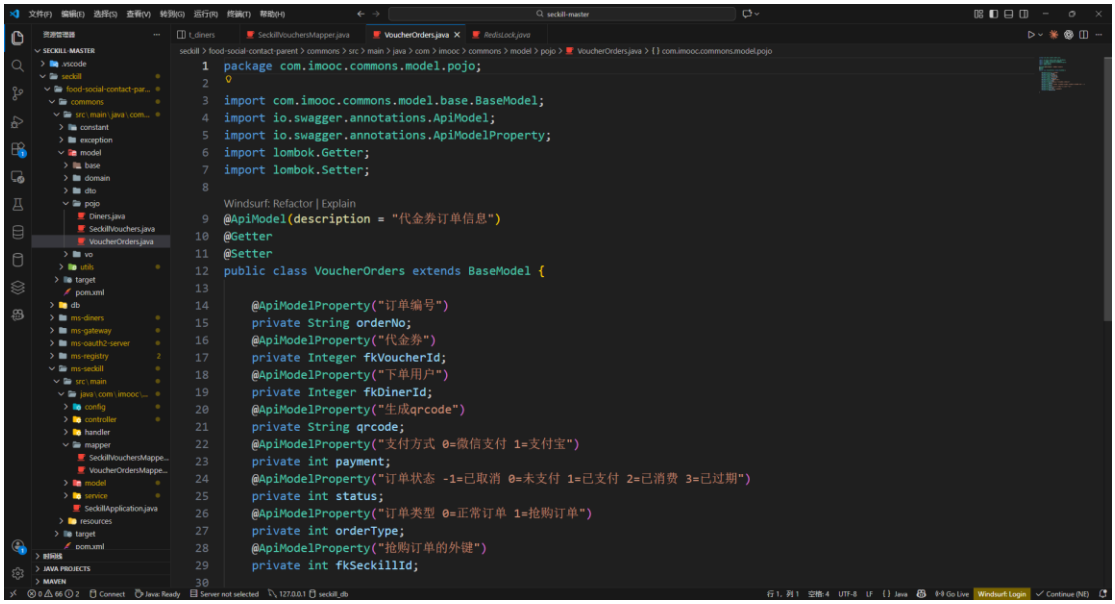


4.1 相关实体类

秒杀代金券实体

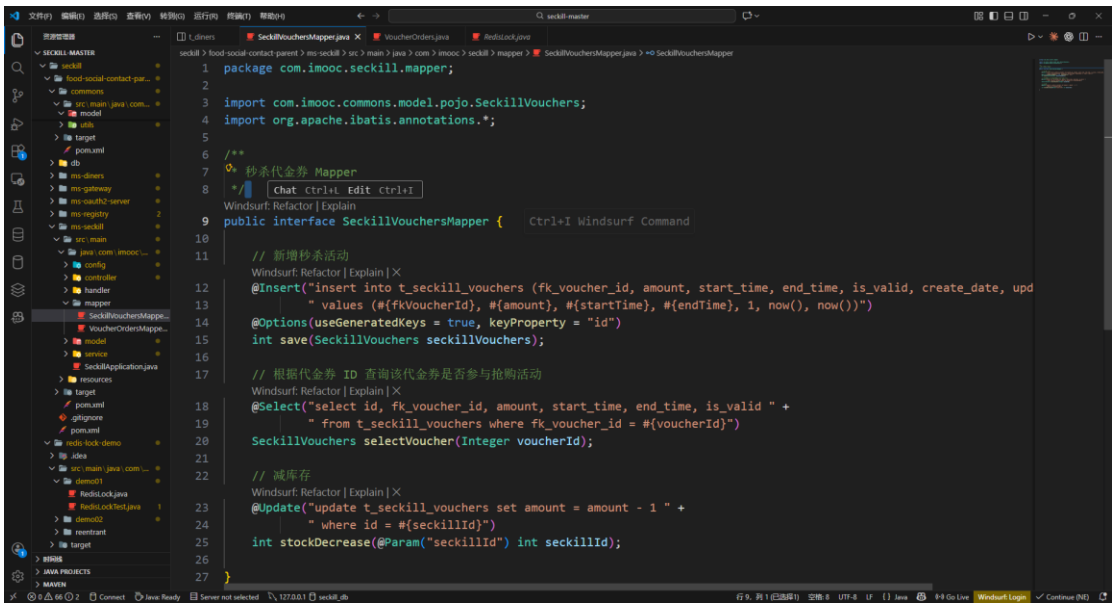


代金券订单实体

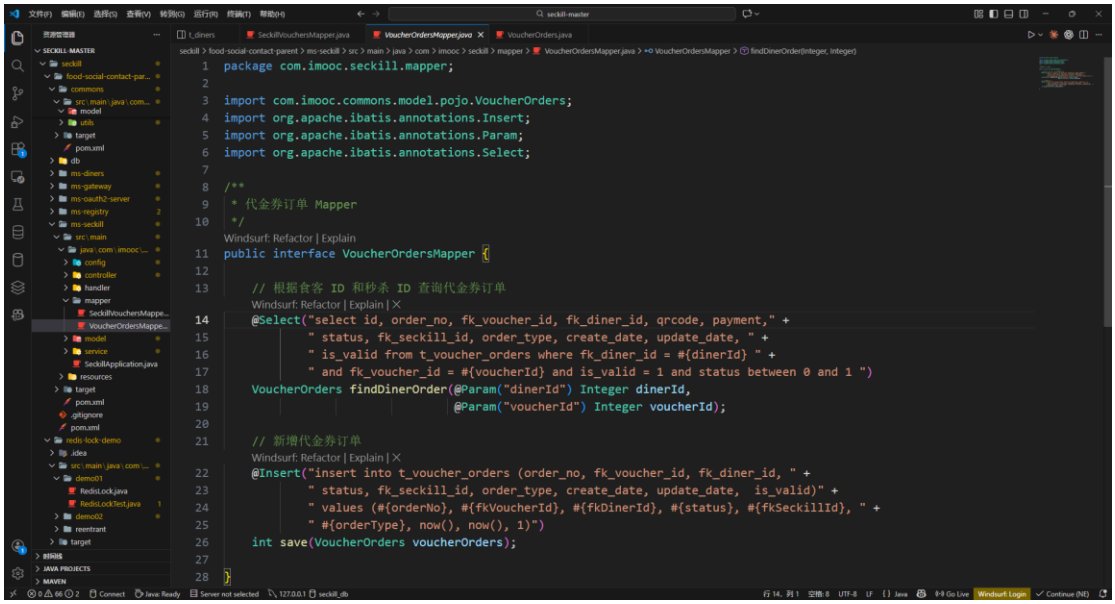


```java
package com.imooc.commons.model.pojo;

import com.imooc.commons.model.base.BaseModel;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Getter;
import lombok.Setter;

@ApiModel(description = "代金券订单信息")
@Getter
@Setter
public class VoucherOrders extends BaseModel {

    @ApiModelProperty("订单编号")
    private String orderNo;
    @ApiModelProperty("代金券")
    private Integer fkVoucherId;
    @ApiModelProperty("下单用户")
    private Integer fkDinerId;
    @ApiModelProperty("生成qrcode")
    private String qrcode;
    @ApiModelProperty("支付方式 0=微信支付 1=支付宝")
    private int payment;
    @ApiModelProperty("订单状态 -1=已取消 0=未支付 1=已支付 2=已消费 3=已过期")
    private int status;
    @ApiModelProperty("订单类型 0=正常订单 1=抢购订单")
    private int orderType;
    @ApiModelProperty("抢购订单的外键")
    private int fkSeckillId;
```

4..2 Maper 接口

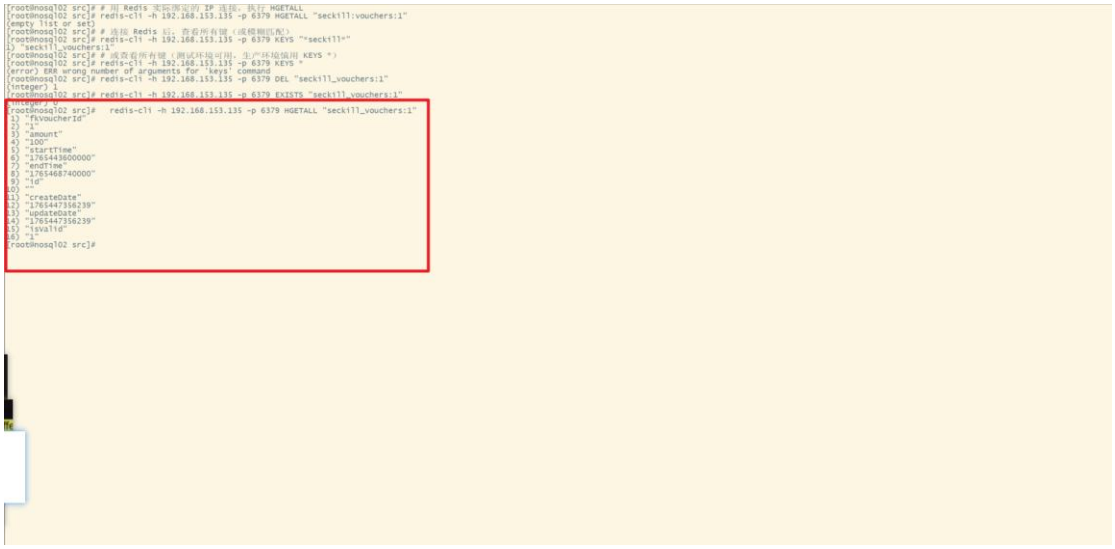


```java
package com.imooc.seckill.mapper;

import com.imooc.commons.model.pojo.SeckillVouchers;
import org.apache.ibatis.annotations.*;

/**
 * 秒杀代金券 Mapper
 */
public interface SeckillVouchersMapper {

    // 新增秒杀活动
    @Insert("insert into t_seckill_vouchers (fk_voucher_id, amount, start_time, end_time, is_valid, create_date, upd
            " values (#{fkVoucherId}, #{amount}, #{startTime}, #{endTime}, 1, now(), now())")
    @Options(useGeneratedKeys = true, keyProperty = "id")
    int save(SeckillVouchers seckillVouchers);

    // 根据代金券 ID 查询该代金券是否参与抢购活动
    @Select("select id, fk_voucher_id, amount, start_time, end_time, is_valid " +
            " from t_seckill_vouchers where fk_voucher_id = #{voucherId}")
    SeckillVouchers selectVoucher(Integer voucherId);

    // 减库存
    @Update("update t_seckill_vouchers set amount = amount - 1 " +
            " where id = #{seckillId}")
    int stockDecrease(@Param("seckillId") int seckillId);

}
```



```java
package com.imooc.seckill.mapper;

import com.imooc.commons.model.pojo.VoucherOrders;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;

/**
 * 代金券订单 Mapper
 */
public interface VoucherOrdersMapper {

    // 根据食客 ID 和秒杀 ID 查询代金券订单
    @Select("select id, order_no, fk_voucher_id, fk_diner_id, qrcode, payment," +
            " status, fk_seckill_id, order_type, create_date, update_date, " +
            " is_valid from t_voucher_orders where fk_diner_id = #{dinerId} " +
            " and fk_voucher_id = #{voucherId} and is_valid = 1 and status between 0 and 1 ")
    VoucherOrders findDinerOrder(@Param("dinerId") Integer dinerId,
                                 @Param("voucherId") Integer voucherId);

    // 新增代金券订单
    @Insert("insert into t_voucher_orders (order_no, fk_voucher_id, fk_diner_id, " +
            " status, fk_seckill_id, order_type, create_date, update_date,  is_valid)" +
            " values (#{orderNo}, #{fkVoucherId}, #{fkDinerId}, #{status}, #{fkSeckillId}, " +
            " #{orderType}, now(), now(), 1)")
    int save(VoucherOrders voucherOrders);

}
```

Redis 实验

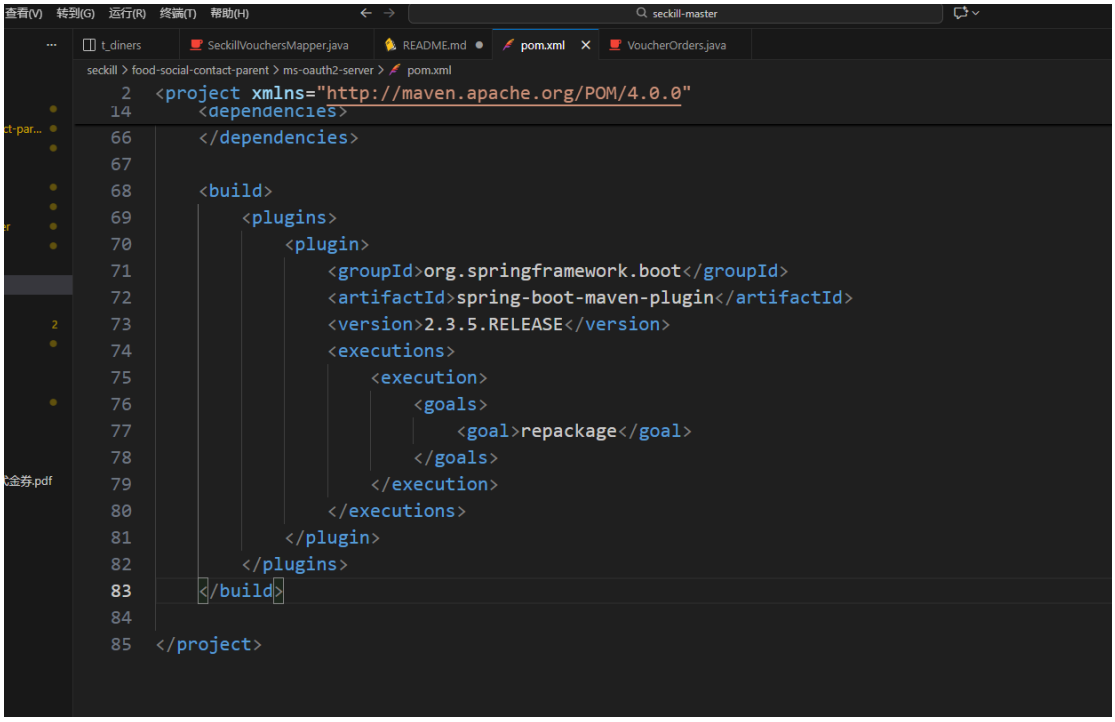


```
[root@nosql02 src]# redis-cli -h 192.168.153.135 -p 6379 HGETALL "seckill_vouchers:1"
 1) "fkVoucherId"
 2) "1"
 3) "amount"
 4) "100"
 5) "startTime"
 6) "1765443600000"
 7) "endTime"
 8) "1765468740000"
 9) "id"
10) "1"
11) "createDate"
12) "1765447356239"
13) "updateDate"
14) "1765447356239"
15) "isValid"
16) "1"
[root@nosql02 src]#
```

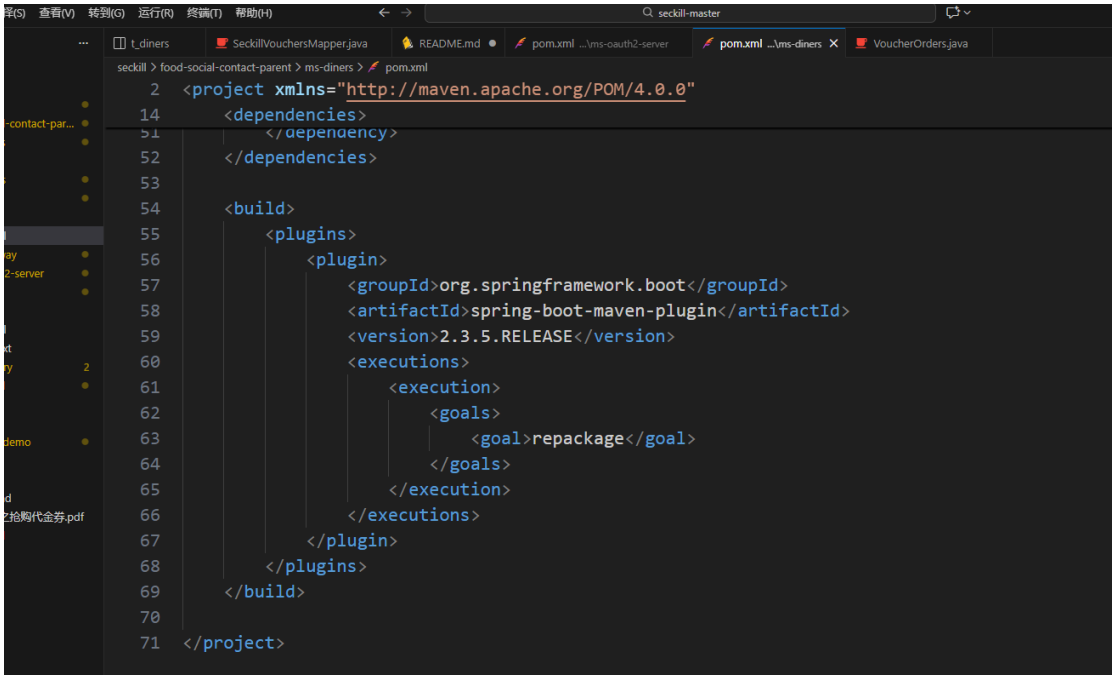
OAuth2 服务用于用户认证和授权

抢购接口需要 access_token，由此服务提供

OAuth2 服务的 pom.xml 缺少 Spring Boot Maven 插件配置，导致无法打包成可执行 jar

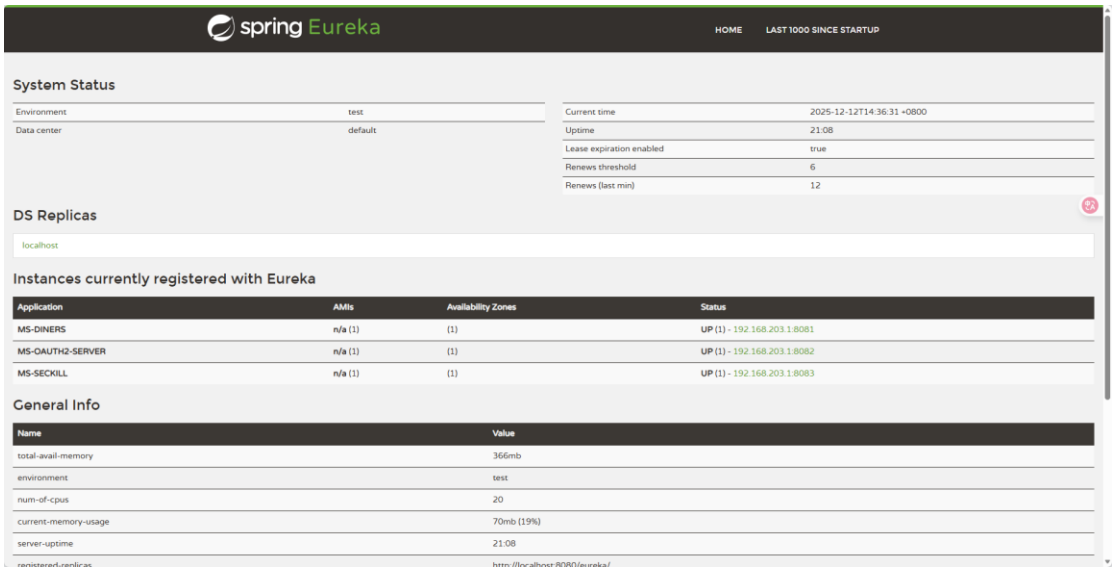


为 ms-diners 服务添加 Spring Boot Maven 插件配置、

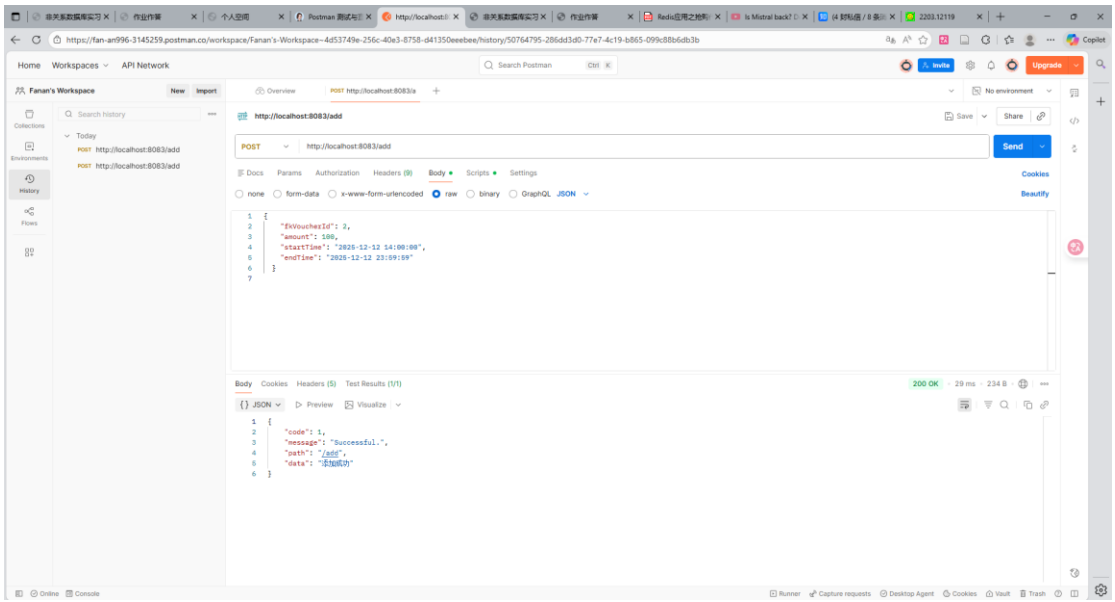


5.1 启动服务

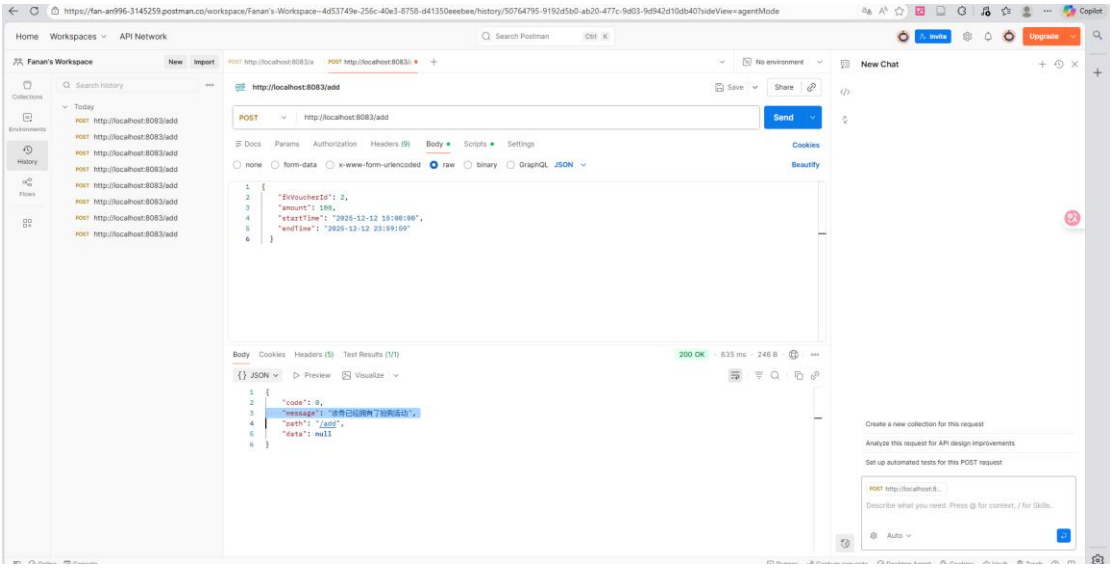
Eureka 注册中心



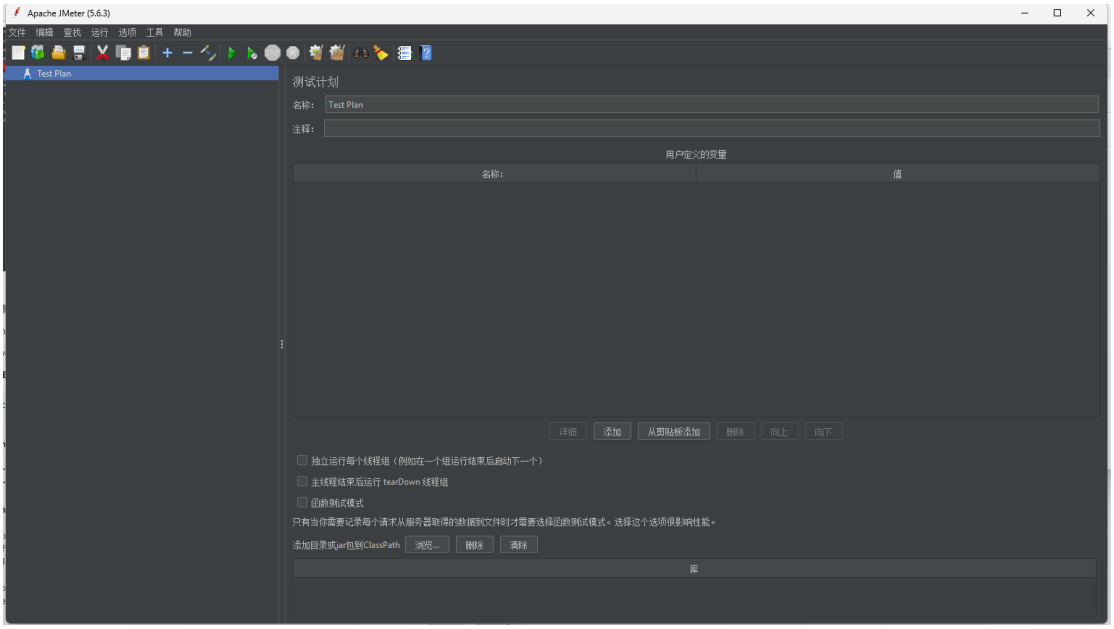
Postman 测试：



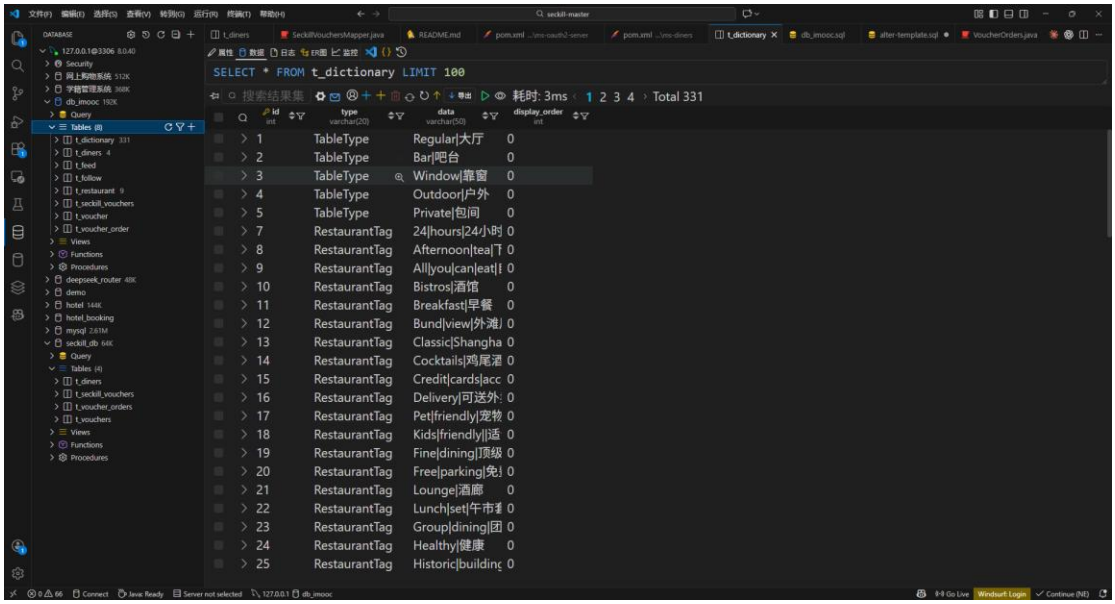
多次输入会提示


Home
Workspaces
API Network
Fanan's Workspace
http://localhost:8083/add
POST
Send
Body
raw
JSON
{
"fkVoucherId": 2,
"amount": 100,
"startTime": "2025-12-12 15:00:00",
"endTime": "2025-12-12 23:59:59"
}
200 OK
{
"code": 0,
"message": "该券已经拥有了抢购活动",
"path": "/add",
"data": null
}
New Chat

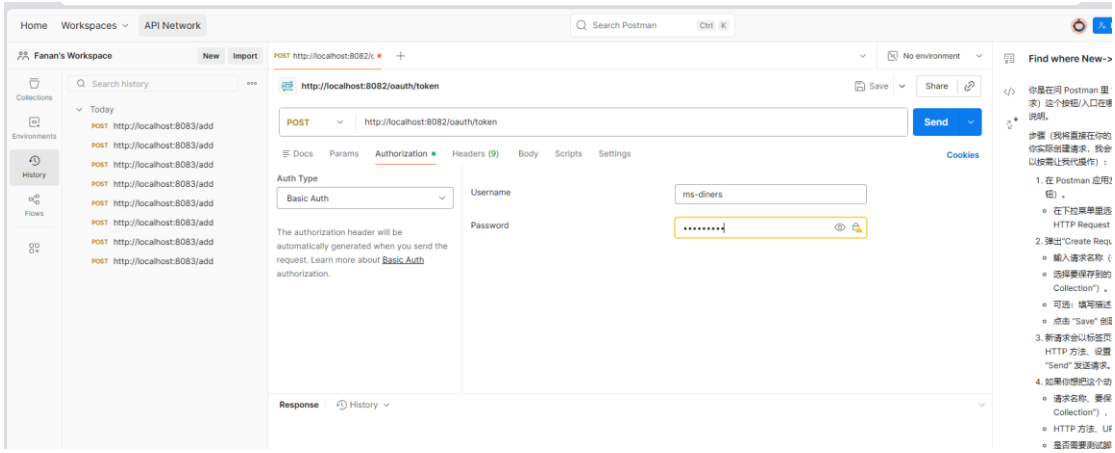
JMeter


Apache JMeter (5.6.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
测试计划
名称:
Test Plan
注释:
用户定义的变量
名称:
值
详细
添加
从剪贴板添加
删除
向上
向下
独立运行每个线程组（例如在一个组运行结束后启动下一个）
主线程结束后运行 tearDown 线程组
函数测试模式
只有当你需要记录每个请求从服务器取得的数据到文件时才需要选择函数测试模式。选择这个选项很影响性能。
添加目录或jar包到ClassPath
浏览...
删除
清除
库

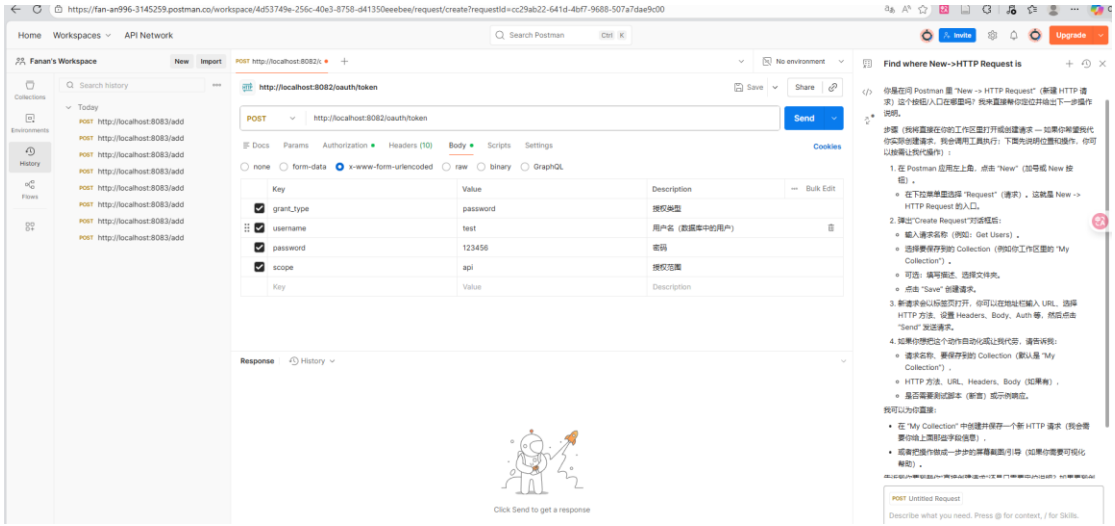
导入数据:



配置 Authorization

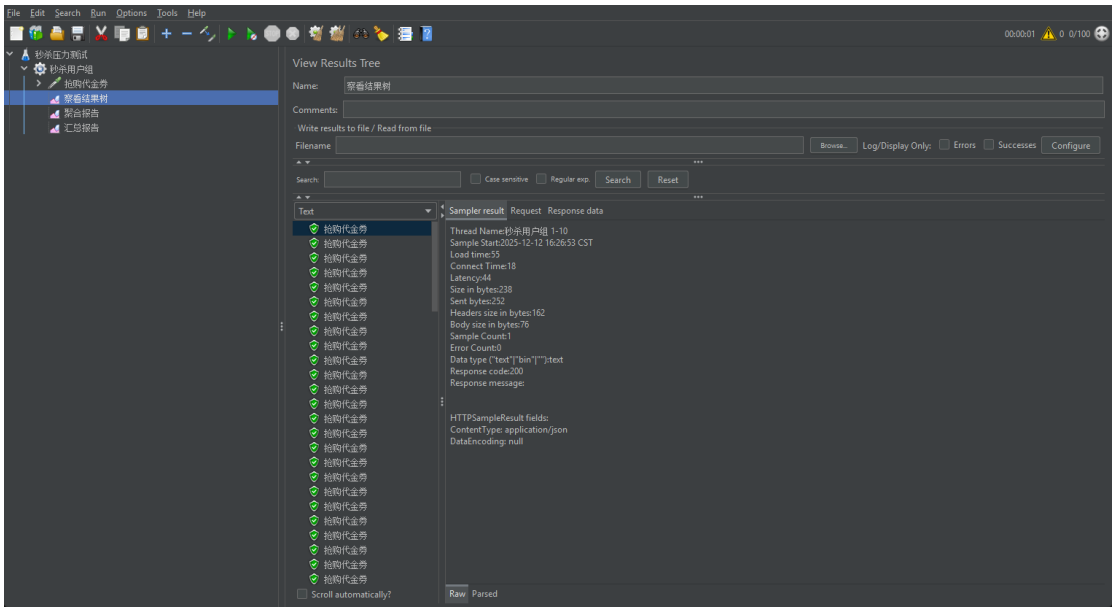


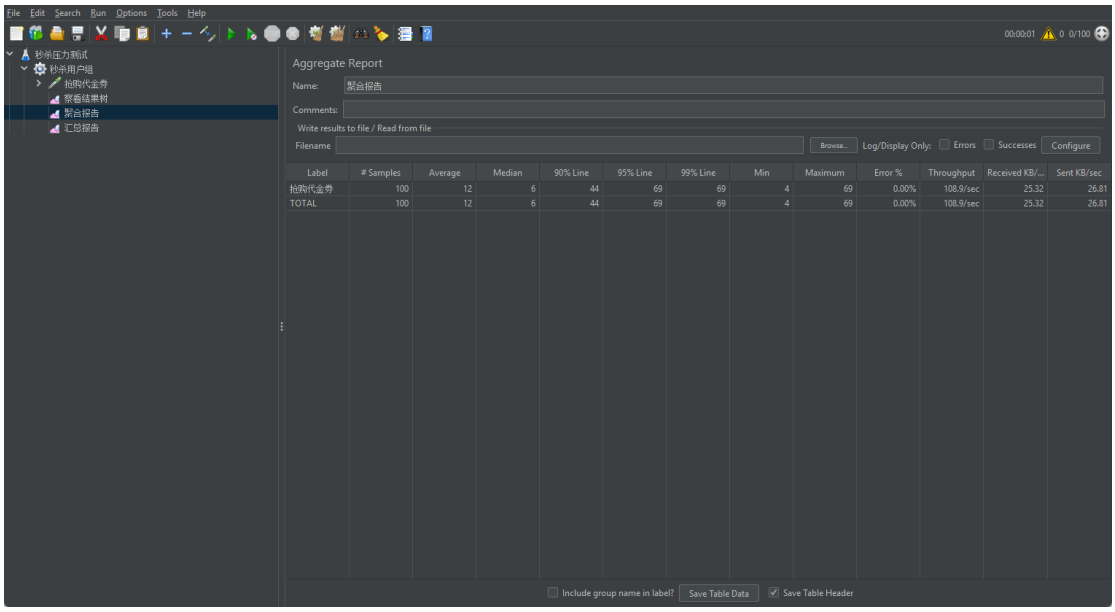
配置 body

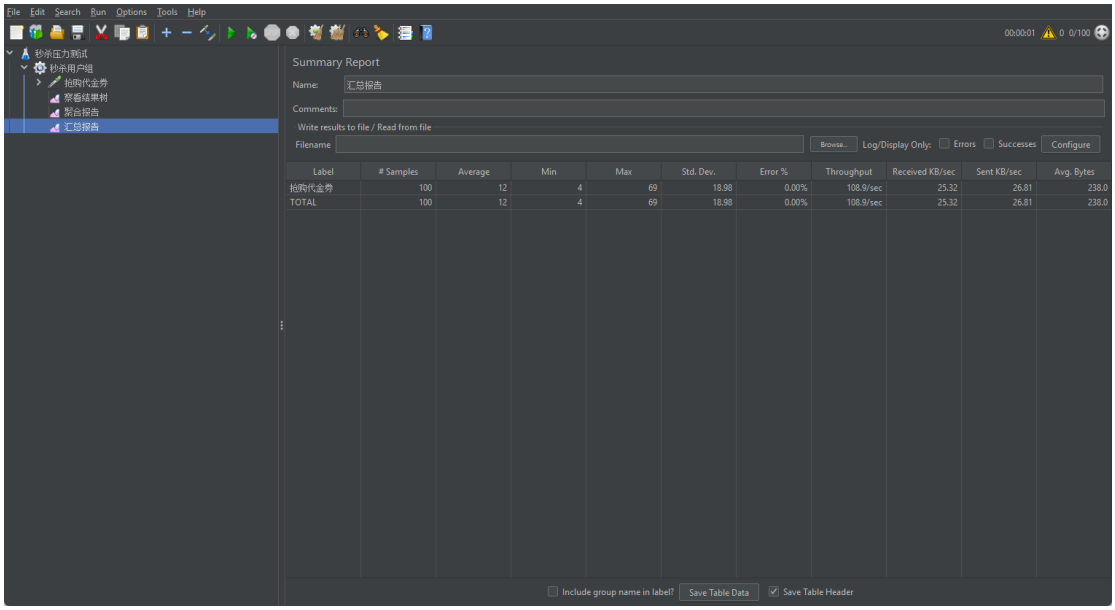


报错了，不用配置，把 redis 的密码删了就好了

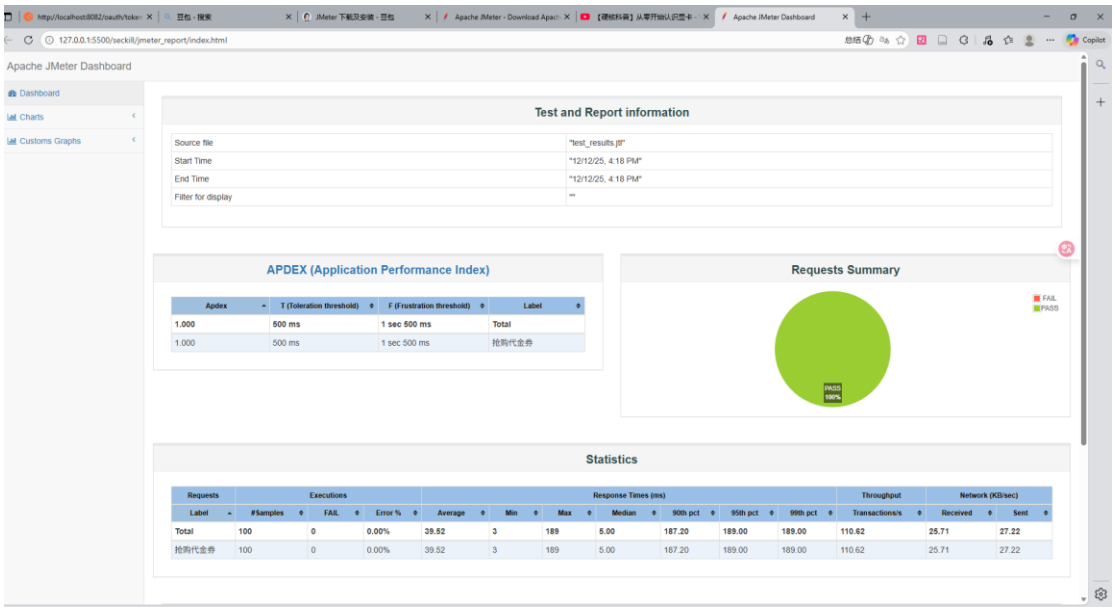
JMetre 测试运行结果：



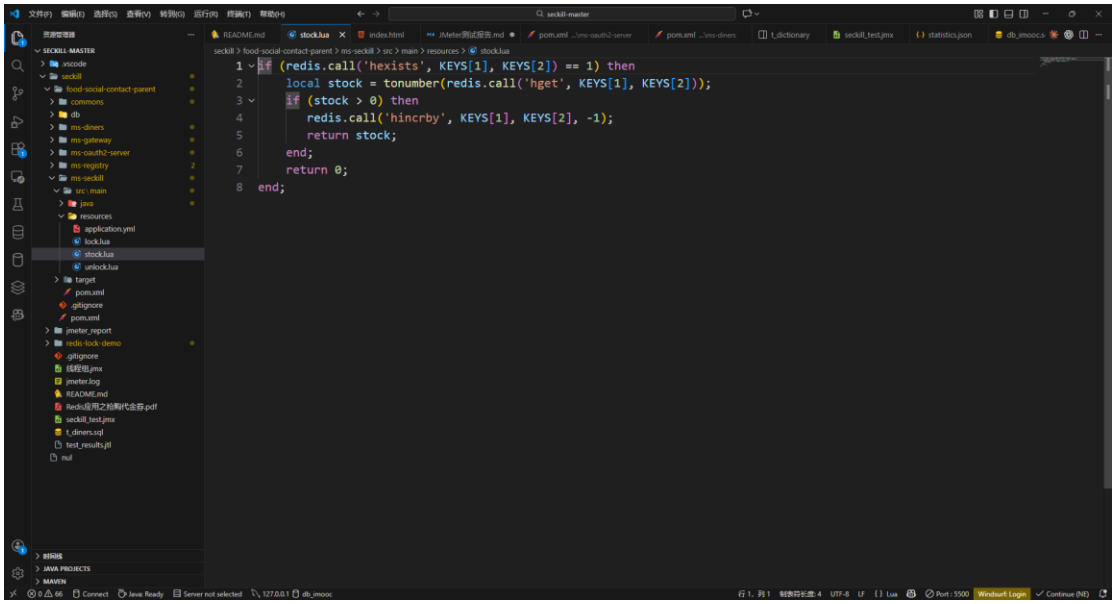




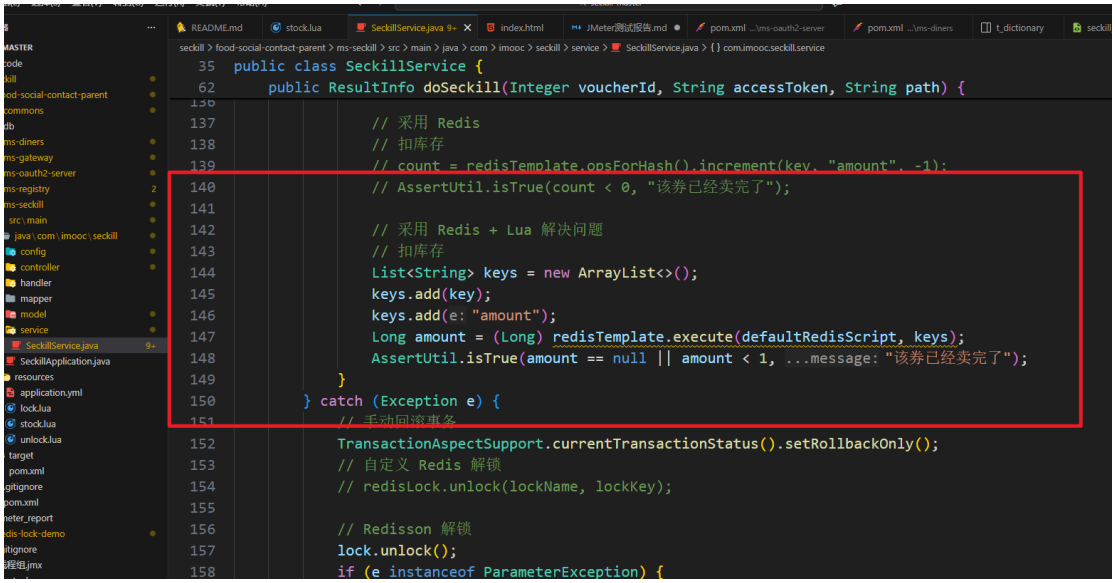
图形化页面：



6 Redis 防止超卖（Lua 脚本原子性扣减库存）



```lua
if (redis.call('hexists', KEYS[1], KEYS[2]) == 1) then
    local stock = tonumber(redis.call('hget', KEYS[1], KEYS[2]));
    if (stock > 0) then
        redis.call('hincrby', KEYS[1], KEYS[2], -1);
        return stock;
    end;
    return 0;
end;
```



```java
public class SeckillService {
    public ResultInfo doSeckill(Integer voucherId, String accessToken, String path) {
                // 采用 Redis
                // 扣库存
                // count = redisTemplate.opsForHash().increment(key, "amount", -1);
                // AssertUtil.isTrue(count < 0, "该券已经卖完了");

                // 采用 Redis + Lua 解决问题
                // 扣库存
                List<String> keys = new ArrayList<>();
                keys.add(key);
                keys.add(e: "amount");
                Long amount = (Long) redisTemplate.execute(defaultRedisScript, keys);
                AssertUtil.isTrue(amount == null || amount < 1, ...message: "该券已经卖完了");
            }
        } catch (Exception e) {
            // 手动回滚事务
            TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
            // 自定义 Redis 解锁
            // redisLock.unlock(lockName, lockKey);

            // Redisson 解锁
            lock.unlock();
            if (e instanceof ParameterException) {
```

Redisson 分布式锁实现

README.md | stock.lua | SeckillService.java 9+ | index.html | JMeter测试报告.md | pom.xml ...\ms-oauth2-server | pom.xml ...\ms-diners | t_dictionary

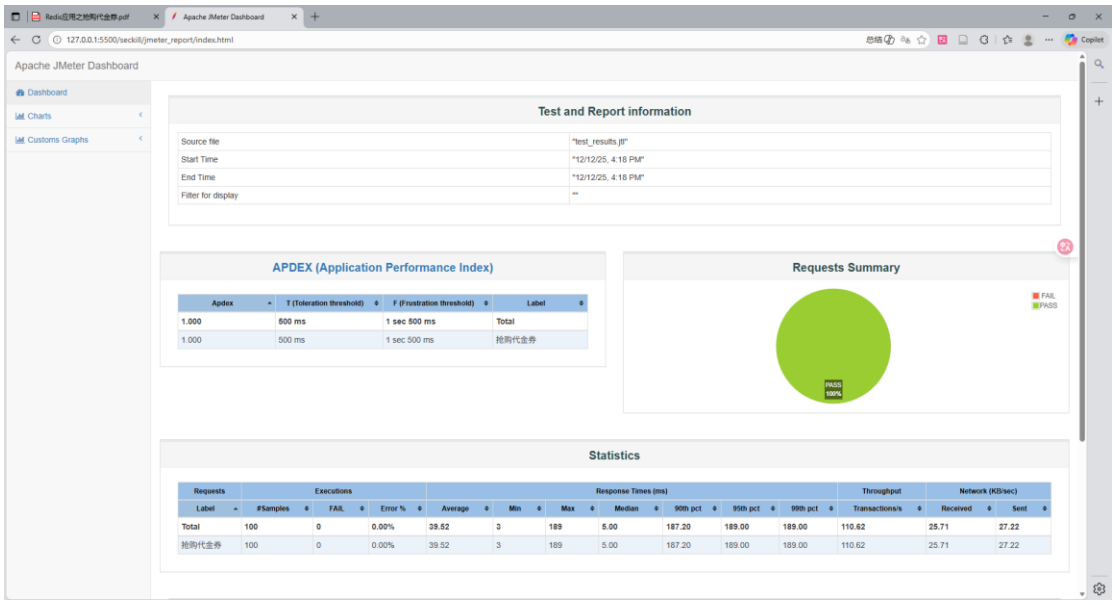
seckill > food-social-contact-parent > ms-seckill > src > main > java > com > imooc > seckill > service > SeckillService.java > { } com.imooc.seckill.service

```
35  public class SeckillService {
62      public ResultInfo doSeckill(Integer voucherId, String accessToken, String path) {
105         // 使用 Redis 锁一个账号只能购买一次
106         String lockName = RedisKeyConstant.lock_key.getKey()
107                 + dinerInfo.getId() + ":" + voucherId;
108         long expireTime = seckillVouchers.getEndTime().getTime() - now.getTime();
109
110         // 自定义 Redis 分布式锁
111         //String lockKey = redisLock.tryLock(lockName, expireTime);
112
113         // Redisson 分布式锁
114         RLock lock = redissonClient.getLock(lockName);
115
116         try {
117             // 不为空意味着拿到锁了，执行下单
118             // 自定义 Redis 分布式锁处理
119             // if (StrUtil.isNotBlank(lockKey)) {
120
121             // Redisson 分布式锁处理
122             boolean isLocked = lock.tryLock(expireTime, TimeUnit.MILLISECONDS);
123             if (isLocked) {
124                 // 下单
125                 VoucherOrders voucherOrders = new VoucherOrders();
126                 voucherOrders.setFkDinerId(dinerInfo.getId());
127                 // Redis 中不需要维护外键信息
128                 // voucherOrders.setFkSeckillId(seckillVouchers.getId());
129                 voucherOrders.setFkVoucherId(seckillVouchers.getFkVoucherId());
130                 String orderNo = IdUtil.getSnowflake(1, 1).nextIdStr();
131                 voucherOrders.setOrderNo(orderNo);
132                 voucherOrders.setOrderType(orderType: 1);
```

总结：

说明：当前忽略了启动 redis 的步骤，需要自行启动 VM 和 CRT

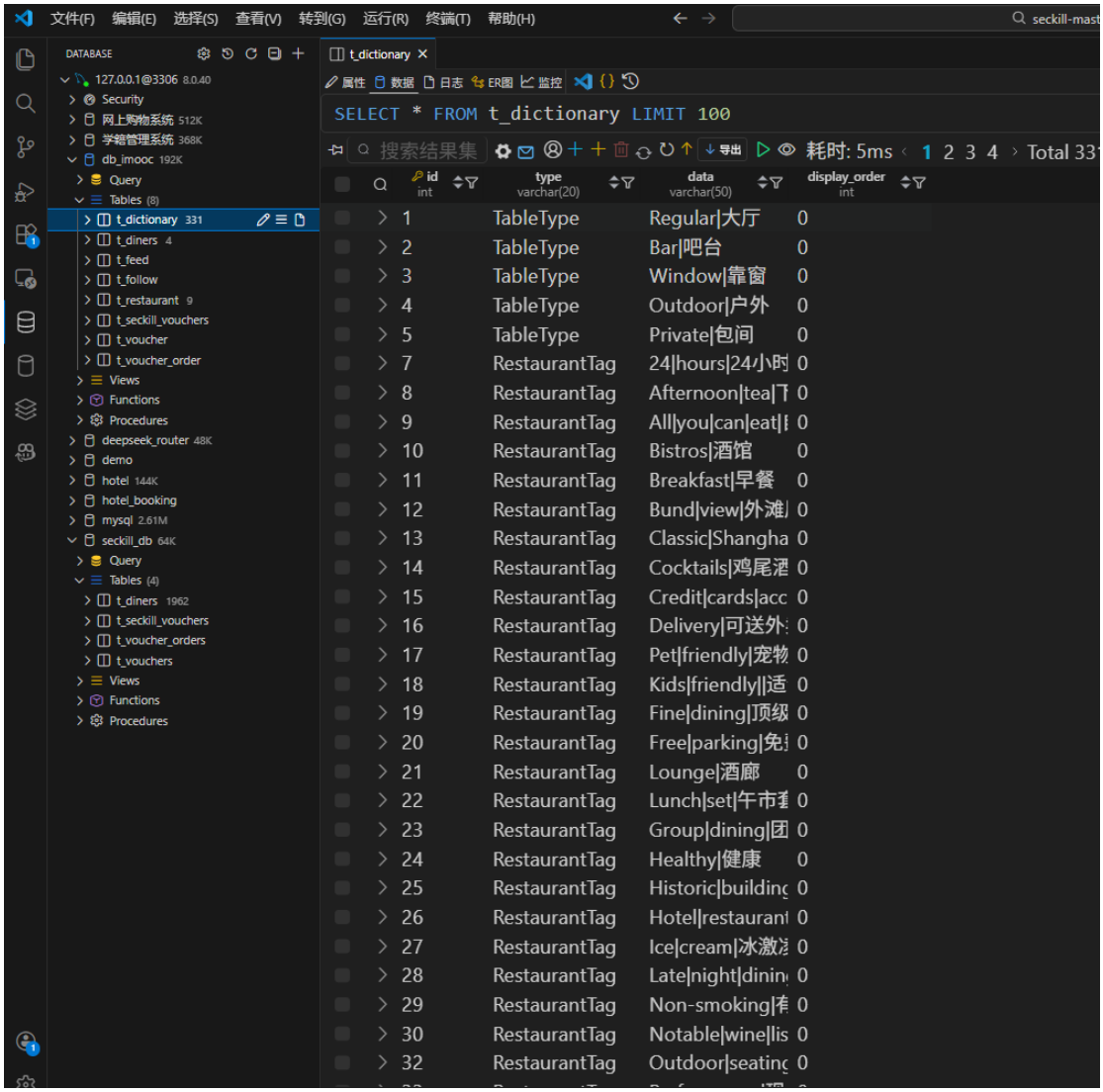
总测试结果和聚合报告

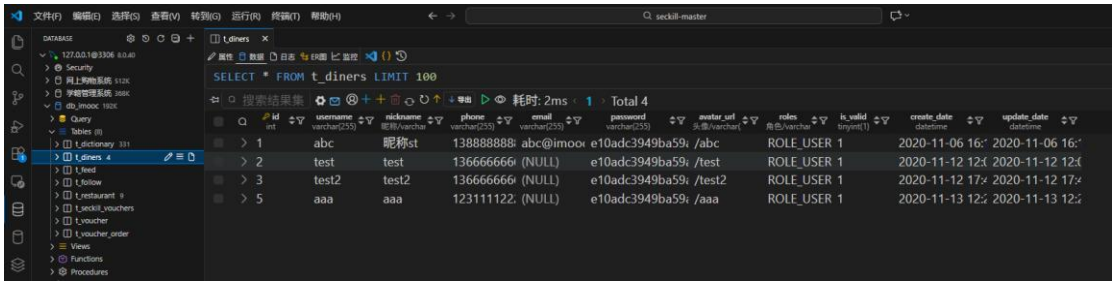


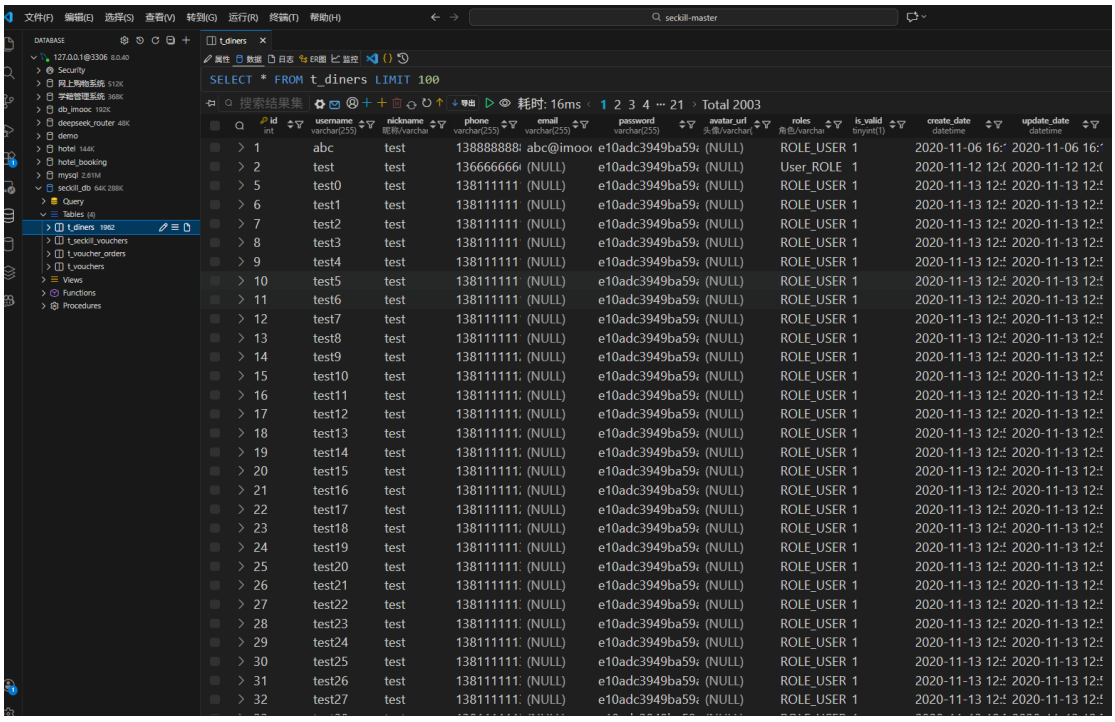
```
{
  "Total" : {
    "transaction" : "Total",
    "sampleCount" : 100,
    "errorCount" : 0,
    "errorPct" : 0.0,
    "meanResTime" : 39.52000000000002,
    "medianResTime" : 5.0,
    "minResTime" : 3.0,
    "maxResTime" : 189.0,
    "pct1ResTime" : 187.20000000000005,
    "pct2ResTime" : 189.0,
    "pct3ResTime" : 189.0,
    "throughput" : 110.61946902654867,
    "receivedKBytesPerSec" : 25.710384402654867,
    "sentKBytesPerSec" : 27.222759955752213
  },
  "抢购代金券" : {
    "transaction" : "抢购代金券",
    "sampleCount" : 100,
    "errorCount" : 0,
    "errorPct" : 0.0,
    "meanResTime" : 39.52000000000002,
    "medianResTime" : 5.0,
    "minResTime" : 3.0,
    "maxResTime" : 189.0,
    "pct1ResTime" : 187.20000000000005,
    "pct2ResTime" : 189.0,
    "pct3ResTime" : 189.0,
    "throughput" : 110.61946902654867,
    "receivedKBytesPerSec" : 25.710384402654867,
    "sentKBytesPerSec" : 27.222759955752213
  }
}
```


附录：

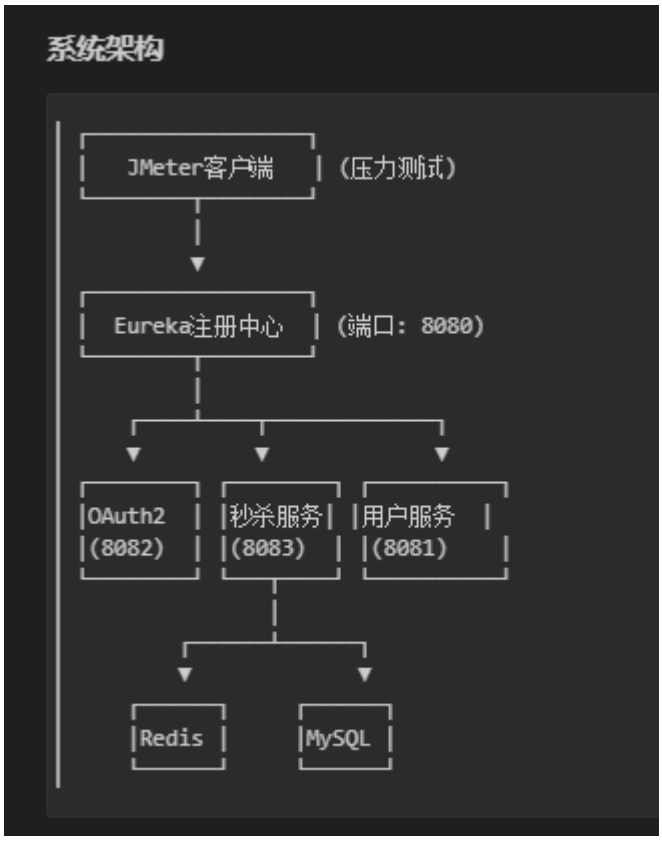
一些数据库表汇总：







Readme 未在打包文件中展示，固本项目相关介绍和指令在此一并补充：



1.数据库准备

 # 创建数据库
mysql -h localhost -u root -p123456 -e "CREATE DATABASE IF NOT EXISTS db_imooc DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;"

2. Redis 服务准备

 # 检查 Redis 是否运行
redis-cli -h 192.168.153.135 -p 6379 ping
 # 预期输出: PONG
 # 如果 Redis 未启动，启动 Redis 服务
redis-server /path/to/redis.conf

3. 项目编译

设置好环境变量（Java11，报错方面修改了很多，原项目对 11 和 17 都不兼容，需修改，比如 Spring Boot Maven 插件配置、还有 Redis 密码删掉等等，在此不一一列举）

编译秒杀服务：
cd D:\你的路径\seckill-master\seckill\food-social-contact-parent\ms-seckill
mvn clean package -DskipTests

编译 Eureka 注册中心（ms-registry）
cd D:\你的路径\seckill-master\seckill\food-social-contact-parent\ms-registry
mvn clean package -DskipTests

编译 OAuth2
cd D:\你的路径\seckill-master\seckill\food-social-contact-parent\ms-oauth2-server
mvn clean package -DskipTests

编译用户服务
cd D:\你的路径\seckill-master\seckill\food-social-contact-parent\ms-diners
mvn clean package -DskipTests

相关验证测试

测试 1: 添加秒杀活动
工具: Postman 或 curl
接口: POST http://localhost:8083/add
请求头:
Content-Type: application/json
请求体:

```
{
   "fkVoucherId": 1,
   "amount": 100,
   "startTime": "2025-12-11 17:00:00",
   "endTime": "2025-12-11 23:59:59"
}
```

使用 curl 测试（Windows CMD）:

```
curl -X POST http://localhost:8083/add -H "Content-Type: application/json" -d "{\"fkVoucherId\": 1, \"amount\": 100, \"startTime\": \"2025-12-11 17:00:00\", \"endTime\": \"2025-12-11 23:59:59\"}"
```

预期响应:

```
{
   "code": 1,
   "message": "Successful.",
   "path": "/add",
   "data": "添加成功"
}
```

测试 2: 验证 Redis 数据
运行位置: Linux 虚拟机（192.168.153.135）

```
# 查看秒杀券数据
redis-cli -h 192.168.153.135 -p 6379 HGETALL "seckill_vouchers:1"
```

预期输出:

```
 1) "fkVoucherId"
 2) "1"
 3) "amount"
 4) "100"
 5) "startTime"
 6) "1765443600000"
```

7) "endTime"
8) "1765468740000"
9) "id"
10) ""
11) "createDate"
12) "1765447356239"
13) "updateDate"
14) "1765447356239"
15) "isValid"
16) "1"
说明:

- 数据成功存储在 Redis 的 Hash 结构中
- key 格式: seckill_vouchers:{voucherId}
- 时间戳格式为毫秒级 Unix 时间戳

测试 3: 抢购代金券（需要 OAuth2 服务）
接口: POST http://localhost:8083/{voucherId}?access_token={token}
步骤:

1. 先获取 access_token（需要 OAuth2 服务）
2. 调用抢购接口

使用 Postman 测试:
POST http://localhost:8083/1?access_token=your_access_token_here
预期响应:

```
{
   "code": 1,
   "message": "Successful.",
   "path": "/1",
   "data": "抢购成功"
}
```

7.1 Redis 缓存秒杀券信息 测试

步骤 1: 添加秒杀活动
在 Windows 命令提示符（CMD）中执行:
curl -X POST http://localhost:8083/add -H "Content-Type: application/json" -d "{\"fkVoucherId\": 1, \"amount\": 100, \"startTime\": \"2025-12-12 00:00:00\", \"endTime\": \"2025-12-12 23:59:59\"}"
预期响应:

```
{
   "code": 1,
   "message": "Successful.",
   "path": "/add",
   "data": "添加成功"
}
```

步骤 2: 在 Redis 中查看缓存数据

在 Linux 虚拟机上执行：

```
# 查看秒杀券的完整信息
redis-cli -h 192.168.153.135 -p 6379 HGETALL "seckill_vouchers:1"
```

预期输出：

```
 1) "fkVoucherId"
 2) "1"
 3) "amount"
 4) "100"
 5) "startTime"
 6) "1734019200000"
 7) "endTime"
 8) "1734105599000"
 9) "isValid"
10) "1"
11) "createDate"
12) "1734019356239"
13) "updateDate"
14) "1734019356239"
```

7.2 Redis 防止超卖（Lua 脚本） 测试

步骤 1: 准备测试环境

确保已添加秒杀活动（库存 100），并获取 access_token。

步骤 2: 使用 JMeter 进行并发测试

配置 JMeter：

- 线程数：100（模拟 100 个用户同时抢购）
- Ramp-Up 时间：1 秒
- 循环次数：1 次

执行测试后，查看 Redis 中的库存：

```
# 查看剩余库存
redis-cli -h 192.168.153.135 -p 6379 HGET "seckill_vouchers:1" "amount"
```

7.3 Redis 限制一人一单（分布式锁）测试

步骤 1: 获取 access_token

```
curl -X POST "http://localhost:8082/oauth/token" ^
   -H "Content-Type: application/x-www-form-urlencoded" ^
   -u "appId:123456" ^
   -d "grant_type=password&username=test&password=123456&scope=api"
```

记录返回的 accessToken。

步骤 2: 第一次抢购（应该成功）

```
curl -X POST "http://localhost:8083/1?access_token=your_access_token_here"
```

预期响应：

```
{
   "code": 1,
   "message": "Successful.",
```

```
    "path": "/1",
    "data": "抢购成功"
}
```

步骤 3: 第二次抢购（应该失败）

使用相同的 access_token 再次请求：

```
curl -X POST "http://localhost:8083/1?access_token=your_access_token_here"
```

预期响应:

```
{
    "code": 0,
    "message": "该用户已抢到该代金券，无需再抢",
    "path": "/1",
    "data": null
}
```

步骤 4: 查看 Redis 中的分布式锁

在 Linux 虚拟机上执行：

```
# 查看分布式锁的 key（抢购过程中可能存在）
redis-cli -h 192.168.153.135 -p 6379 KEYS "lockby:*"
```

JMeter 压力测试

步骤 1: 获取 OAuth2 访问令牌

运行位置: Windows 命令提示符（CMD）

```
# 获取 access_token
curl -X POST "http://localhost:8082/oauth/token" ^
    -H "Content-Type: application/x-www-form-urlencoded" ^
    -u "appId:123456" ^
    -d "grant_type=password&username=test&password=123456&scope=api"
```

步骤 2: 添加秒杀活动

```
# 添加秒杀活动（库存 100）
curl -X POST http://localhost:8083/add ^
    -H "Content-Type: application/json" ^
    -d "{\"fkVoucherId\": 1, \"amount\": 100, \"startTime\": \"2025-12-12 00:00:00\", \"endTime\": \"2025-12-12 23:59:59\"}"
```

非 GUI 执行压力测试

```
# 执行 JMeter 测试（非 GUI 模式）
E:\apache-jmeter-5.6.3\bin\jmeter.bat -n -t seckill_test.jmx -l test_results.jtl -e -o jmeter_report
```

也可使用图形化页面，找到相应的 bat 文件打开，再打开我们的工作目录（*seckill_test.jmx*）即可

默认测试配置：

并发用户：100

Ramp-Up 时间：1s
循环次数：1 次
总请求数：100 个
注意事项：

1. 每次测试前清除结果（GUI 模式）
    - 点击 运行 → 清除全部
    - 避免数据混淆
2. 重置秒杀活动（如需重复测试）
3. *# 删除旧的秒杀活动*
4. redis-cli -h 192.168.153.135 -p 6379 DEL "seckill_vouchers:1"
5.
6. *# 重新添加秒杀活动*
7. curl -X POST http://localhost:8083/add -H "Content-Type: application/json" -d "{\"fkVoucherId\": 1, \"amount\": 100, \"startTime\": \"2025-12-12 00:00:00\", \"endTime\": \"2025-12-12 23:59:59\"}"
8. 测试规模建议
    - GUI 模式：适合 < 500 个用户
    - 命令行模式：适合 > 500 个用户的大规模测试
9. 观察服务器状态
    - 测试时注意观察后台服务日志
    - 如果出现大量错误，检查服务是否正常运行
10. access_token 过期
    - 如果测试时出现 401 错误，重新获取 access_token
    - 默认过期时间约 30 天